Eine agentenbasierte Architektur für Programmierung mit gesprochener Sprache

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Dipl.-Inform Sebastian Weigelt
aus Weimar

Tag der mündlichen Prüfung:	14.05.2021
Erster Gutachter:	Prof. Dr. Walter F. Tichy
Zweiter Gutachter:	Prof. Dr.-Ing. Tamim Asfour

Zusammenfassung

Sprachgesteuerte Computersysteme werden heutzutage von Millionen von Nutzern verwendet; Chatbots, virtuelle Assistenten, wie *Siri* oder *Google Assistant*, und Smarthomes sind längst fester Bestandteil des Alltags vieler Menschen. Zwar erscheinen derartige Systeme inzwischen intelligent; tatsächlich reagieren sie aber nur auf einzelne Befehle, die zudem bestimmte Formulierungen erfordern. Die Nutzer sind außerdem auf vorgefertigte Funktionalitäten beschränkt; neue Befehle können nur von Entwicklern einprogrammiert und vom Hersteller zur Verfügung gestellt werden. In Zukunft werden Nutzer erwarten, intelligente Systeme nach ihren Bedürfnissen anzupassen, das heißt *programmieren* zu können.

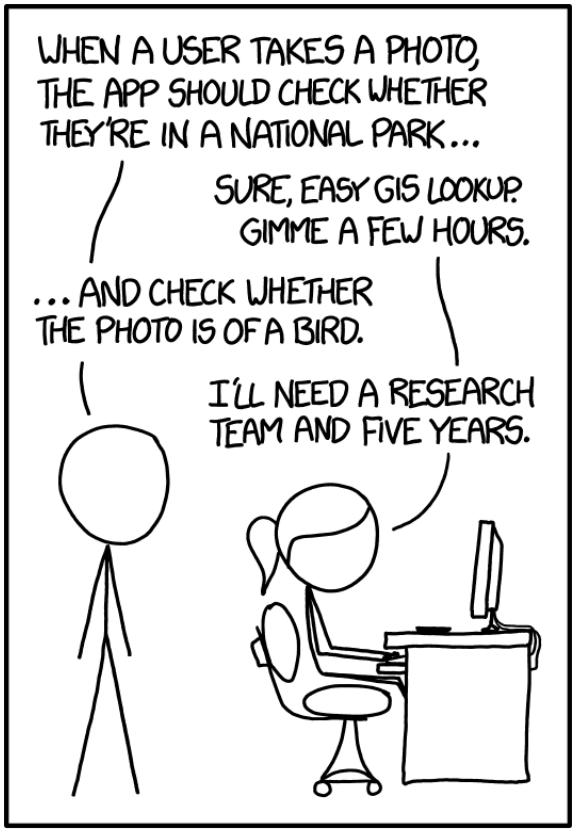
Das in dieser Arbeit beschriebene System ProNat ermöglicht Endnutzer-Programmierung mit gesprochener Sprache. Es befähigt Laien dazu, einfache Programme für unterschiedliche Zielsysteme zu beschreiben und deren Funktionalität zu erweitern. ProNat basiert auf PARSE, einer eigens entworfenen agentenbasierten Architektur für tiefes Sprachverständnis. Das System ermöglicht die Verwendung alltäglicher Sprache zur Beschreibung von Handlungsanweisungen. Diese werden von ProNat als Programm für ein Zielsystem interpretiert, das eine Anwendungsschnittstelle zur Endnutzer-Programmierung anbietet. Bisherige Ansätze zur Programmierung mit natürlicher Sprache ermöglichen nur die Erzeugung *kurzer* Programme anhand *textueller* Beschreibungen. Da die meisten Systeme monolithisch entworfen wurden, können sie zudem nur *mit großem Aufwand adaptiert* werden und sind überwiegend auf *die Anwendung einer Technik* (z. B. maschinelles Lernen) sowie auf *eine Anwendungsdomäne* festgelegt (z. B. Tabellenkalkulation). Ansätze, die gesprochene Sprache verarbeiten, können hingegen bisher nur *einzelne Befehle* erfassen.

Um die Restriktionen bisheriger Ansätze aufzuheben, wird eine neuartige Architektur entworfen. Die Kernkomponenten der Architektur PARSE bilden unabhängige Agenten, die je einen bestimmten Aspekt der natürlichen Sprache analysieren. Die Kapselung in unabhängige Agenten ermöglicht es, je Teilaspekt zum Verständnis der Sprache eine andere Technik zu verwenden. Die Agenten werden nebenläufig ausgeführt. Dadurch können sie von Analyseergebnissen anderer Agenten profitieren; unterschiedliche Sprachanalysen können sich so gegenseitig unterstützen. Beispielsweise hilft es, sprachliche Referenzen wie Anaphern aufzulösen, um den Kontext des Gesagten zu verstehen; manche Referenzen können wiederum nur mithilfe des Kontextes aufgelöst werden. Ihr Analyseergebnisse hinterlegen die Agenten in einer geteilten Datenstruktur, einem Graphen. Die Architektur stellt sicher, dass keine Wettlaufsituationen eintreten und nur gültige Änderungen am Graphen durchgeführt werden. Die Agenten werden so lange wiederholt ausgeführt, bis keine oder nur noch zyklische Änderungen eintreten. Neben den Agenten gibt PARSE die Verwendung von Fließbändern zur Vor- und Nachverarbeitung vor. Zudem können externe Ressourcen, wie Wissensdatenbanken oder Kontextmodellierungen, angeschlossen werden.

Das System ProNat entsteht, indem konkrete Agenten und Fließbandstufen für die Rahmenarchitektur PARSE bereitgestellt werden. Zusätzlich werden Informationen über die Anwendungsdomäne (das heißt die Anwendungsschnittstelle des Zielsystems und gegebenenfalls eine Modellierung der Systemumgebung) in Form von Ontologien als externe Ressource angebunden. Eine gesprochene Äußerung wird von ProNat vorverarbeitet, indem zunächst das Audiosignal in eine textuelle Wortsequenz überführt wird. Anschließend erfolgt eine grundlegende syntaktische Analyse, bevor ein initialer Graph als Analysegrundlage für die Agenten erzeugt wird. Die Interpretation des Gesagten als Programm obliegt den Agenten. Es wurden sechzehn Agenten entwickelt, die sich in drei Kategorien unterteilen lassen: Erstens, Agenten, die allgemeine Sprachverständnis-Analysen durchführen, wie die Disambiguierung von Wortbedeutungen, die Auflösung von sprachlichen Referenzen oder die Erkennung von Gesprächsthemen. Zweitens, Agenten, die das Gesagte auf programmatische Strukturen, wie Anwendungsschnittstellenaufrufe oder Kontrollstrukturen, untersuchen; hierzu zählt auch ein Agent, der aus verbalisierten Lehrsequenzen Methodendefinitionen synthetisiert. Zusätzlich wurden funktionale Agenten entwickelt, wie beispielsweise ein Dialog-Agent, der dem Nutzer gezielt Rückfragen stellt. Da die Agenten unabhängig voneinander agieren, kann zur Lösung der jeweiligen Problemstellung eine beliebige Technik eingesetzt werden. Die Agenten zur Erkennung von Kontrollstrukturen verwenden beispielsweise Heuristiken, die auf syntaktischen Strukturen basieren, um ihre Analysen durchzuführen. Andere Agenten, wie die Agenten zur Disambiguierung von Wortbedeutungen oder zur Bestimmung der Gesprächsthemen, verwenden *Wikipedia*, *Wordnet* oder ähnliche Quellen und inferieren anhand dieser Informationen. Zuletzt verwenden einige Agenten, wie beispielsweise der Agent zur Erkennung von Lehrsequenzen, maschinelles Lernen. Die Interpretation einer gesprochenen Äußerung erfolgt dementsprechend mittels einer Kombination von sowohl regel- als auch statistik- und wissensbasierten Techniken. Dank der strikten Trennung der Agenten können diese einzeln (und zumeist unabhängig voneinander) evaluiert werden. Hierzu wurden parallel zur Entwicklung der Agenten fortwährend mithilfe von Nutzerstudien realistische Eingabebeispiele gesammelt. Für jeden Agenten kann somit überprüft werden, ob er einen zufriedenstellenden Beitrag zur Interpretation des Gesagten beiträgt. Das gemeinschaftliche Analyseergebnis der Agenten wird in der Nachverarbeitung sukzessive in ein konkretes Programm übersetzt: Zunächst wird ein abstrakter Syntaxbaum generiert, der anschließend in Quelltext zur Steuerung eines Zielsystems überführt wird.

Die Fähigkeit des Systems ProNat, aus gesprochenen Äußerungen Quelltext zu generieren, wurde anhand von drei unabhängigen Untersuchungen evaluiert. Als Datengrundlage dienen alle in den Nutzerstudien gesammelten natürlichsprachlichen Beschreibungen. Zunächst wurden für eine Online-Studie *UML*-Aktivitätsdiagramme aus gesprochenen Äußerungen generiert und 120 Probanden zur Bewertung vorgelegt: Der überwiegende Teil der Aktivitätsdiagramme (69%) wurde von der Mehrheit der Probanden als vollständig korrekt eingestuft, ein vielversprechendes Ergebnis, da die gesprochenen Äußerungen die Synthese von bis zu 24 Anweisungen (bzw. Aktivitäten) sowie Kontrollstrukturen erfordern. In einer zweiten Untersuchung wurde *Java*-Quelltext, bestehend aus Aufrufen einer Anwendungsschnittstelle zur Steuerung eines humanoiden Roboters, synthetisiert und mit einer Musterlösung verglichen: ProNat konnte Aufrufe meist korrekt erzeugen (F_1: 0,746); auch die Synthese von Kontrollstrukturen gelingt in 71% der Fälle korrekt. Zuletzt wurde untersucht, wie gut ProNat anhand von natürlichsprachlichen Beschreibungen neue Funktionen erlernen kann:

Verbalisierte Lehrsequenzen werden mit einer Genauigkeit von 85% in Äußerungen erkannt. Aus diesen leitet ProNat Methodendefinitionen ab; dabei gelingt es in über 90% der Fälle, einen sprechenden Methodennamen zu erzeugen. Auch der Aufruf der neu erlernten Funktion (durch natürlichsprachliche Anweisungen) gelingt mit einer Genauigkeit von 85%. Zusammengenommen zeigen die Untersuchungen, dass ProNat grundsätzlich in der Lage ist, Programme aus gesprochenen Äußerungen zu synthetisieren; außerdem können neue Funktionen anhand natürlichsprachlicher Beschreibungen erlernt werden.



Vorwort

In der Informatik haben selbst Experten häufig Schwierigkeiten, einzuschätzen (bzw. zu begründen), welche Aufgaben einfach gelöst werden können und welche nahezu unmöglich sind. Ähnliche Situationen, wie im Comic auf der vorangegangen Seite gezeigt, ereignen sich tagtäglich. Daher ist es auch nicht verwunderlich, dass der Comic auf einer wahren Begebenheit beruht. 1966 gab Marvin Minsky – einer der Wegbereiter des Forschungsgebiets der künstlichen Intelligenz – einigen seiner Studenten am MIT als Aufgabe über den Sommer, einen Algorithmus zu entwickeln, der es ermöglicht, Objekte in einem Bild zu erkennen und zu benennen[1]. Er ahnte damals nicht, dass er seine Studenten damit vor eine Aufgabe stellen würde, die bis heute noch nicht zur Gänze gelöst ist.

Ähnlich wie mit der Objekterkennung in Bildern verhält es sich mit der Analyse und dem Verständnis natürlicher Sprache durch Computersysteme. Auch diese Aufgabe erscheint intuitiv einfach lösbar, schließlich ist es für Menschen ein Leichtes, natürlichsprachliche Äußerungen zu interpretieren. Bei der Bewertung der Schwierigkeit einer Aufgabenstellung wird allerdings mitunter völlig außer Acht gelassen, dass das menschliche Gehirn grundlegend anders arbeitet als ein Computersystem – daran hat (bisher) auch die Entwicklung immer leistungsfähigerer künstlicher neuronaler Netze nichts geändert. Wer sich jedoch jemals eingehend mit der Computerlinguistik befasst hat, fühlt sich schnell wie Alice aus Lewis Carrols Buch *Alice im Wunderland*, die sich immer tiefer im Kaninchenbau verliert. Unumgänglich wird einem offenbar, dass eine umfassende und allgemeingültige Interpretation natürlicher Sprache nahezu unmöglich, gerade deshalb aber als Forschungsgegenstand besonders reizvoll ist. Schickt man sich darüber hinaus an, natürliche Sprache nicht nur interpretieren, sondern aus dieser Interpretation auch noch Programmquelltext synthetisieren zu wollen[2], erhält man für gewöhnlich zweierlei Rückmeldungen. Zum einen gibt es solche, die dies aufgrund der eingangs thematisierten Intuition für natürliche Sprache als einfach erachten: „Ja klar, ist doch keine große Sache!“ Zum anderen sind da die, die aufgrund von Wissen um die Komplexität natürlicher Sprache, diese Ziel für unerreichbar halten: „Programme aus natürlichsprachlichen Beschreibungen erstellen? Unmöglich!“ Abseits dieser beiden Extreme bleibt viel ungefüllter Raum (der einen Nachwuchsforscher leicht zum Zweifeln bringen kann).

Umso wichtiger ist es, dass es Menschen gibt, die einen unterstützen, (heraus-)fordern, fördern und an die eigene Forschungsidee glauben. Ich schätze mich glücklich, dass mich eine Vielzahl solcher Menschen durch meine Promotion begleitet haben – ohne dieses Umfeld hätte ich alles, was mir

[1] siehe hierzu: `https://www.explainxkcd.com/wiki/index.php/1425:_Tasks`, zuletzt besucht am 21.06.2021.

[2] Die Programmierung und Steuerung von Computersystemen mit natürlicher Sprache wurde ebenfalls (möglicherweise nicht ganz) zufälligerweise 1966 das erste Mal konkret als Zielstellung formuliert. Wie die Objekterkennung ist auch diese Aufgabe bisher ungelöst.

in den sieben (!) Jahren abverlangt wurde, wohl kaum bewältigen können. An erster Stelle ist hier natürlich Walter zu nennen. Er gab mir nicht nur die Chance, an seinem Lehrstuhl zu promovieren, sondern ließ mir inhaltlich freie Hand, sowohl bei der Themenfindung, als auch bei der Festlegung des Lösungswegs. Diese Freiheit, die ich sehr zu schätzen weiß, bedeutete aber nicht, dass ich ohne doktorväterliche Führung hätte auskommen müssen: Walters Tür stand mir jeder Zeit offen; er half mir bei so mancher kniffligen Frage und streute immer wieder neue Ideen ein, wenn es galt, eine inhaltliche Lücke zu schließen. Vor allem aber ist er mir über all die Jahre ein Vorbild hinsichtlich einer (wirklich) wissenschaftlichen Arbeitsweise gewesen. Einen wesentlichen Anteil am Erfolg meiner Promotion hat natürlich auch der gesamte restliche Lehrstuhl, der für mich immer das perfekte und unkomplizierte Arbeitsumfeld war. Angefangen bei Hildegard, die als gute Seele des Lehrstuhls im Sekretariat immer ein offenes Ohr (oder auch zwei) für uns Mitarbeiter hatte, über die Systemadministratoren (zunächst) Heinz und (später) Andrea, die nicht nur technische Probleme immer ohne viel Aufhebens lösen konnten, sondern auch immer als sachkundige Ansprechpartner mit Rat und Tat zur Verfügung standen, bis hin zu den Mit-Promovierenden des Lehrstuhls (bzw. Leidensgenossen). Ja, die lieben Kollegen: Ein zusammengewürfelter Haufen von (mehr oder minder) jungen Menschen, die für die Forschung brennen, die Welt verbessern wollen oder einfach nur nach einem Stück Selbstverwirklichung suchen. Dankbar bin ich Ihnen für all die Diskussionen, die unabhängig davon, ob man sich gerade mit einer echten Forschungsfrage oder der *korrekten* Rezeptur von Kartoffelsalat auseinandersetzt, immer mit Tiefgang geführt wurden und dadurch immer bereichernd waren. Trotz inhaltlich teils völlig konträrer Ausrichtung wurde versucht, sich in das Forschungsthema des jeweils anderen einzudenken, kritisch zu hinterfragen aber auch neue Ideen und Lösungsansätze zu entwickeln. Fast noch wichtiger waren aber all die kleinen und großen gemeinsamen Auszeiten – nie vergessen werde ich die Kaffee-Runden im legendären *Competence Center*. Viele (für meine Promotion) entscheidende Ideen haben sich während solcher Auszeiten und dem gemeinsamen Blödeln mit den Kollegen entwickelt. Drei der Kollegen, die sich auch nie für eine Blödelei zu schade waren, möchte ich besonders danken: Mathias, der als erster an mich und meine Fähigkeiten geglaubt hat (und letztlich auch hauptverantwortlich dafür war, dass ich mich dazu entschieden habe zu promovieren) und mit dem ich wunderbar produktiv arbeiten aber auch über alles mögliche andere reden konnte. Martin, mit dem ich gemeinsam einige Lehrveranstaltung abhalten durfte und mit dem es einfach immer lustig war – auch wenn es die eine oder andere hitzige Diskussion gab. Und Tobias, der erst als Student eine herausragende Abschlussarbeit ablieferte und später als mein langjähriger Büro-Gegenüber nicht nur im Diskussions-Ping-Pong glänzte, sondern auch als kongenialer Koautor an (bisher) zwölf gemeinsamen Veröffentlichungen beteiligt war.

Außer von Walter und meinen Kollegen habe ich auch jede Menge durch die vielfältigen Aufgaben gelernt, die an einem Lehrstuhl anfallen. Vieles davon hätte ich mir vorher nie träumen lassen (und wahrscheinlich auch nicht zugetraut), wie beispielsweise die Administration einer Lernplattform oder eines DevOps-Systems. Besondere Freude hat mir über all die Jahre aber vor allem die Lehre bereitet; die Weitergabe von Wissen an die nächste Generation ist ein tolles Erlebnis, insbesondere, wenn man das Gefühl hat, dass ab und an auch eine Aufnahme stattfindet. Egal ob die Zweitsemester-Großveranstaltung mit teilweise über 700 Studenten, eine Vertiefungsfachvorlesung mit Master-Studenten kurz vor dem Abschluss oder das (von Martin und mir ins Leben gerufene) Praktikum zur Programmierung mit natürlicher Sprache, alles hatte seine Herausforderungen aber

auch besondere Reize. Die schönsten Erfahrungen im Kontext der Lehre habe ich allerdings bei der Betreuung von (in Summe 26) Abschlussarbeiten gesammelt, die ja für sich gesehen auch kleine Forschungsprojekte sind. Diskussionen über Herausforderungen einer Forschungsfrage oder das gemeinsame Entwickeln von Lösungsstrategien am Whiteboard war mit den meisten Studenten eine wirkliche Freude und hat mir nicht zuletzt selbst die eine oder andere neue Idee beschert. Außerdem ist es ungemein spannend, die persönliche Entwicklung der Studenten über eine oder manchmal sogar zwei Abschlussarbeiten hinweg zu beobachten, beginnend mit dem wachsenden Interesse für ein Thema über die Entwicklung eigener Ideen durch immer tieferes Verständnis des Forschungsgebiets bis hin zum „Durchbeißen“, um die Arbeit rechtzeitig abzuschließen. Dabei war es mir vergönnt einige sehr talentierte junge Menschen kennen lernen zu dürfen, unter anderem Tobias und Jan, die später beide zu Kollegen wurden oder Vanessa, die nicht nur zwei herausragende Abschlussarbeiten abgeliefert hat, sondern zudem als Studentin Koautorin von sieben Veröffentlichungen wurde (darunter eine bei der Hauptkonferenz der Computerlinguistik und eine weitere, die mit einem „Best Paper Award“ ausgezeichnet wurde). Aber auch alle hier nicht genannten haben beinahe ausnahmslose gute Arbeit geleistet und auf die eine oder andere Weise einen Beitrag zu meiner Promotion geliefert.

Aber um der Wahrheit genüge zu tun, bedeutet eine Promotion neben all diesen schönen Erfahrungen vor allem eines: viel Stress! Der Spagat zwischen eigener Forschungs- und Lehrtätigkeit erfordert die richtige Balance, damit nicht eines der beiden plötzlich das Übergewicht bekommt. Hinzu kommt die Eigenorganisation und -motivation, um beispielsweise während eines arbeitsaufwendigen Sommersemesters als Übungsleiter einer Großveranstaltung nebenher auch noch Abschlussarbeiten zu betreuen, Forschungsergebnisse zu publizieren, die nächsten Projekte vorzubereiten, organisatorische und administrative Aufgaben zu erledigen und die Promotion voranzutreiben.

Damit das Leben am Ende nicht nur noch aus Arbeit besteht (was tatsächlich nicht nur unbefriedigend ist, sondern letztlich auch irgendwann unproduktiv wird), braucht es ausgleichende Elemente, die außerhalb des Universitätsumfelds angesiedelt sind. Solche „Elemente“ sind unter anderem Freunde und Bekannte (die ich in letzter Zeit sträflich vernachlässigt habe); entspannte Runden in netter Gesellschaft sind das beste Mittel, um den Kopf frei zu bekommen. Außerdem wird man daran erinnert, dass es noch eine Welt außerhalb der eigenen Informatik-Blase gibt. Sehr viel verdanke ich aber vor allem meiner Familie, insbesondere meinen Eltern, meiner Schwester und meinen Großeltern. Sie haben mich letztlich zu dem Menschen geformt, der ich heute bin. Doch nicht nur damit haben sie zu meiner Promotion beigetragen, sondern auch durch Unterstützung und Zuspruch über viele Jahre hinweg (vor allem auch in schwierigen Zeiten während des Studiums). Zuletzt möchte ich ganz besonders Anna-Lena danken. Sie hat mich während meiner gesamten Promotionszeit nach Leibeskräften unterstützt und hat selbst auf vieles verzichten müssen und häufig zurückgesteckt. Anna-Lena hat mir nicht nur den Rücken freigehalten und viel Alltagsstress von mir ferngehalten, sondern mit Fachkenntnis (und einem gewissen Hang zum Perfektionismus) auch das Lektorat der Dissertation übernommen. Bildlich gesprochen, war sie bei gutem (Forschungs-)Wetter der Wind in den Segeln (meines „Promotions-Schiffs“) und der sichere Hafen, wenn ein Sturm aufzog; vor allem aber war sie meine Sonne während der wenigen gemeinsamen Freizeit.

Die Promotion war eine lange, beschwerliche Reise – aber ich bin froh, dass ich sie angetreten habe, denn ich habe viel erlebt und noch mehr gelernt. Ich danke allen, die mich auf dieser Reise begleitet haben. Genauer betrachtet geht die Reise aber noch weiter, denn es gibt noch einiges zu tun bis Computersysteme Sprache so wie ein Mensch verstehen können und ich hoffe, auch zukünftig einen Beitrag zum Erreichen dieses Ziels leisten zu können. Eigentlich brauche ich nur ein Forschungsteam und fünf Jahre Zeit. *Eigentlich*.

Inhaltsverzeichnis

1 Programmierung mit gesprochener Sprache – Eine Notwendigkeit

„There are only two kinds of [programming] languages:
the ones people complain about and the ones nobody uses.“

– Bjarne Stroustrup

Computersysteme haben sich zu omnipräsenten Begleitern im Alltag vieler Menschen entwickelt. Egal ob Laptop, Smartphone, Heimautomation oder das Navigationssystem im Auto – Menschen nutzen persönliche Computersysteme heutzutage nahezu unentwegt. Allerdings liegt der Fokus für den Großteil tatsächlich auf der *Nutzung* vorhandener Funktionalität. Der wirkliche Mehrwert von Computersystemen besteht aber darin, diese *programmieren* zu können. Erst durch die Programmierung werden Computersysteme zu Werkzeugen zur Lösung persönlicher Aufgaben. Diese Art der Verwendung bleibt jedoch denjenigen vorbehalten, die die Programmierung mit einer der zahlreichen Programmiersprachen beherrschen. Über fundierte Programmierkenntnisse verfügt allerdings nur ein geringer Anteil der Menschheit. Auch wenn die Programmierung immer stärker in den Fokus schulischer Ausbildung rückt, steht auch in Zukunft zu erwarten, dass nur wenige Menschen die Programmierung von Computersystemen tatsächlich beherrschen. Für viele Menschen stellt schlicht das Erlernen einer formalen Sprache eine unüberwindbare Hürde dar. Daher gilt es Mittel und Wege zu finden, auch Laien zu befähigen, zumindest einfache Programme erstellen zu können.

Eine Lösung für diese Problematik besteht darin, Laien die Möglichkeit zu bieten, alltägliche Sprache zur Programmierung verwenden zu können. Für Menschen gibt es keine natürlichere Art zu kommunizieren als mittels natürlicher Sprache. Menschen sind es von Kindesbeinen an gewohnt, anhand natürlichsprachlicher Beschreibungen zu lernen. Auch eigene Ideen zu verbalisieren und damit auf andere zu übertragen, gelingt Menschen in Alltagssprache intuitiv. Dem Gegenüber eine Wegbeschreibung geben oder erklären, wie man einen Autoreifen wechselt, sind letztlich auch so etwas wie kleine natürlichsprachliche Programmierungen. Für Menschen wäre die Programmierung mit gesprochener Sprache dementsprechend nichts Neues; vielmehr sollte sie sich für die meisten ganz natürlich anfühlen. Das Ziel dieser Arbeit ist es, ein System zu entwickeln, dass die Programmierung von Computersystem durch Laien mit gesprochener Sprache ermöglicht. Um dieses Ziel zu erreichen, muss das System in der Lage sein, alltägliche Sprache tiefgehend zu verstehen, sodass gesprochene Äußerungen als Programm interpretiert werden können.

1.1 Bestandsaufnahme

Natürliche Sprache scheint die offensichtliche Lösung zu sein, um Laien zur Programmierung zu befähigen, und es stellt sich die Frage, ob es sich hierbei um eine neue Idee handelt. Tatsächlich ist das Gegenteil der Fall. Die Frage, ob es möglich ist, mit natürlicher Sprache zu programmieren, wird seit den Anfängen der Informatik diskutiert. Bereits 1966 proklamierte Jean Sammet, eine Pionierin der Informatik, folgendes [Sam66]:

> „*The only way a person can truly concentrate on his problem and solve it* [...] *are if he is able to communicate directly with the computer without having to learn some specialized intermediate language.*“[1]

Seit dieser Zeit wurden zwar verschiedene vielversprechende Ansätze zur Programmierung mit natürlicher (geschriebener) Sprache entwickelt; ein wirklicher Durchbruch konnte jedoch nicht erzielt werden. Gerade in den letzten Jahren aber steigt das Interesse der Forschungsgemeinde an der natürlichsprachlichen Programmierung zusehends. Diese Beobachtungen werfen wiederum die Fragen auf, was die gegenwärtige Situation von der vor knapp 55 Jahren unterscheidet und warum Programmierung mit natürlicher Sprache heutzutage möglich sein könnte. Zunächst ist die Notwendigkeit mehr denn je gegeben. Jean Sammet zielte mit ihrer Proklamation höchstwahrscheinlich noch auf professionelle Entwickler ab, für die der Nutzen natürlichsprachlicher Programmierung tatsächlich fraglich ist. Heutzutage stehen stattdessen die unzähligen Endnutzer von Computersystemen im Fokus, die es trotz mangelnder Programmierkenntnisse zu befähigen gilt, einfache Programme entwickeln zu können. Weiterhin ist die Akzeptanz für sprachgesteuerte Systeme heutzutage größer. Dank virtueller Assistenten und ähnlichen Systemen mit konversationellen Schnittstellen ist es inzwischen nichts Ungewöhnliches mehr, mit einem Computersystem zu sprechen. Vielmehr steht zu erwarten, dass Nutzer sich schon bald nicht mehr mit der ausschließlichen Nutzung von System mittels Sprache zufriedengeben und sich der Wunsch entwickelt, Systeme durch natürlichsprachliche Programmierung an die eigenen Bedürfnisse anzupassen. Schließlich wurden auf dem Gebiet der Computerlinguistik, das sich mit der rechnergestützten Analyse natürlicher Sprache beschäftigt, in den letzten Jahren (auch aufgrund des gestiegenen öffentlichen Interesses) enorme Fortschritte erzielt. Heutzutage steht – anders als 1966 oder auch noch vor zehn Jahren – eine Vielzahl von Standardwerkzeugen, vortrainierten Sprachmodellen und linguistischen Datensammlungen zur Verfügung; viele dieser Ressourcen können uneingeschränkt verwendet werden. Die Voraussetzungen für die Entwicklung eines Systems zur Programmierung mit gesprochener Sprache sind dementsprechend günstiger denn je. Damit jedoch das enorme Potenzial, das die Computerlinguistik-Forschungsgemeinde geschaffen hat, auch nutzbar wird, muss es gelingen, die unterschiedlichen Techniken und Ressourcen auf geeignete Weise zu kombinieren. Die rasante Entwicklung der vergangenen Jahre verdeutlicht zudem, dass es unumgänglich ist, Systeme offen zu gestalten, sodass sie einfach an neue Gegebenheiten angepasst und neue Techniken und Verfahren angewendet werden können.

[1] Zu deutsch: Die einzige Weise, auf die sich eine Person wirklich auf ihr Problem konzentrieren und dieses lösen kann [...] besteht in der Befähigung, direkt mit dem Computer kommunizieren zu können ohne eine spezielle Zwischensprache erlernen zu müssen.

1.2 Zielstellung und Einschränkungen

In dieser Arbeit wird ein System entworfen und umgesetzt, dass Laien zur Programmierung befähigen soll. Programme sollen aus gesprochenen Handlungsanweisungen abgeleitet werden. Die Handlungsanweisungen sollen in alltäglicher Sprache beschrieben werden können; das bedeutet, das System schränkt die Formulierungen, die ein Nutzer verwenden kann, um sein Programm zu beschreiben, nicht ein.

Zur Umsetzung dieser Anforderungen muss das System ein tiefes Verständnis gesprochener Sprache erlangen. Aktuelle Systeme, die über eine konversationelle Schnittstelle angesprochen werden können, wie beispielsweise virtuelle Assistenten, können nur einzelne Befehle verstehen. Dieses Maß an Sprachverständnis ist für die Programmierung noch unzureichend. Um ein Programm aus einer gesprochenen Äußerung ableiten zu können, muss eine Folge von Handlungsanweisungen erkannt und programmatisch interpretiert werden. Das bedeutet, anhand natürlichsprachlicher Beschreibungen soll Quelltext synthetisiert werden, der beliebig viele Anweisungen enthält. Zudem soll es möglich sein, Handlungsanweisungen so zu formulieren, dass sie nur unter einer bestimmten Bedingung oder wiederholt auszuführen sind; im programmatischen Sinne entspricht dies der Erkennung von Kontrollstrukturen, die dementsprechend ebenfalls synthetisiert werden sollen. Grundsätzlich liegt der Fokus allerdings auf einfachen (kurzen und skriptartigen) Programmen, wie sie von Laien zur Steuerung eines Computersystems zu erwarten sind.

Da ein konkretes System gesteuert werden soll, beschränken sich die Anweisungen auf Aufrufe von Schnittstellenfunktionen. Das bedeutet, die Programme sollen aus einer Komposition von Grundfunktionalitäten eines beliebigen Zielsystems gebildet werden. Es soll aber möglich sein, dem System neue Funktionen zu lehren; dadurch kann der Umfang der Grundfunktionalitäten eines Systems erweitert werden. Das in dieser Arbeit entwickelte System agiert somit als intelligente konversationelle Schnittstelle zur Programmierung beliebiger Zielsysteme.

Denkbar wäre beispielsweise folgendes Szenario: Ein Nutzer erklärt einem Haushaltsroboter mit konversationeller Schnittstelle Arbeitsabläufe, wie *Wäsche waschen* oder *Spülmaschine ausräumen*. Der Roboter erstellt daraus ein Ablaufskript. Sollte der Roboter Teile der Anweisungen nicht verstehen, stellt er Rückfragen an den Nutzer. Der Roboter führt die Anweisungssequenz nicht nur aus, sondern merkt sich die Abläufe. Beim nächsten Mal, wenn er Wäsche waschen oder die Spülmaschine ausräumen soll, kann er die gelernten Abläufe abrufen. Ein derartiges Verhalten wird durch die Verwendung einer intelligenten konversationellen Schnittstelle möglich, solange das Robotersystem bereits eine Reihe von grundlegenden Funktionen anbietet, wie beispielsweise das Greifen von Objekten oder die Fahrt (oder das Gehen) an einen (benennbaren) Ort. Die Schnittstelle sorgt dann für das Verständnis der gesprochenen Sprache – gegebenenfalls inklusive Rückfragen an den Nutzer – und eine geeignete Komposition der Grundfunktionen. Darüber hinaus sorgt die konversationelle Schnittstelle dafür, dass der erlernte Ablauf nicht nur generalisiert, sondern auch für zukünftige Verwendungen abgespeichert wird.

1.3 Abgrenzung

Wie bereits angedeutet, gibt es bereits eine Vielzahl von Ansätzen zur Programmierung mit natürlicher Sprache[2]. Diese befassen sich allerdings nahezu ausschließlich mit der Synthese von Programmen anhand *textueller* Beschreibungen; sie können auch nicht ohne Weiteres auf die Verarbeitung *gesprochener* Äußerungen übertragen werden, da sich gesprochene fundamental von textueller Sprache unterscheidet: Unter anderem ist sie meist ungrammatikalisch und enthält Unterbrechungen, Verzögerungslaute sowie Selbstkorrekturen; all dies erschwert die Analyse erheblich (insbesondere für Verfahren, die für Textdokumente entwickelt wurden). Daher betrachten nur wenige Ansätze überhaupt gesprochene Äußerungen und wenn dann auch nur unterstützend für ein anderes Verfahren[3]. Außerdem schränken viele die Sprache ein, die zur Programmierung verwendet werden darf. Das bedeutet, Handlungsanweisungen können nicht frei formuliert werden; beispielsweise können Nutzer nur festgelegte Satzschablonen verwenden oder sind gezwungen, ihre Beschreibung mit der Formulierung einer Bedingung zu beginnen. Die meisten verwandten Ansätze stellen zudem Speziallösungen dar. Sie wurden für genau einen Anwendungsfall entwickelt, beispielsweise für die Synthese von Tabellenkalkulationsmakros anhand textueller Beschreibungen. Durch die Einschränkung auf eine bestimmte Anwendungsdomäne, können bei der Analyse der natürlichen Sprache Annahmen bzw. Einschränkungen getroffen werden, die die Interpretation deutlich vereinfachen. Allerdings können derartige Systeme nicht oder nur mit großem Aufwand auf andere Anwendungsdomänen übertragen werden. Fast alle bekannten Systeme zur Programmierung mit natürlicher Sprache sind zudem monolithisch entworfen; das bedeutet, unter Anwendung einer bestimmten Technik wird die Sprache analysiert und (ohne Zwischenergebnisse) direkt ein Quelltext-Generat erzeugt; die Systeme agieren als sogenannter *schwarzer Kasten* (engl. *black box*). Dadurch können Fehlerquellen, die bei der Interpretation der natürlichen Sprache auftreten, kaum ermittelt und die angewandten Techniken dementsprechend auch nicht zielgerichtet optimiert werden. Zudem können monolithische Architekturen nur schwierig erweitert oder adaptiert werden.

Das in dieser Arbeit beschriebene System ermöglicht die Programmsynthese anhand gesprochener Äußerungen unter Verwendung beliebiger Formulierungen. Außerdem liegt dem System eine offene Architektur zugrunde; diese unterteilt das System in definierte Einzelkomponenten, sogenannte *Agenten*. Durch diesen Ansatz kann das System mit geringem Aufwand an andere Anwendungsdomänen (das heißt andere Zielsystemschnittstellen, die eine beliebige Programmiersprache verwenden) angepasst werden; bei einer Adaption bleibt der Großteil des Systems unverändert – insbesondere die Teile, die natürliche Sprache interpretieren und entsprechend aufwendig zu entwickeln sind. Außerdem ist das System erweiterbar; das Sprachverständnisvermögen kann beliebig ausgedehnt werden. Falls festgestellt wird, dass die Interpretation der natürlichen Sprache nur unzureichend gelingt, können neue Sprachverständniskomponenten jederzeit hinzugefügt werden. Die Unterteilung des Systems in Einzelkomponenten hat zudem den Vorteil, dass diese unabhängig voneinander evaluiert und Fehlerquellen dementsprechend exakt bestimmt werden können.

[2] Ein eingehender Vergleich mit verwandten Arbeiten wird in Kapitel 3 angestellt.

[3] Beispielsweise bieten einige Ansätze die Möglichkeit visuelle Programmierung (durch Zeigegesten) durch einzelne gesprochene Anweisungen zu unterstützen.

1.4 Eine agentenbasierte Architektur für tiefes Sprachverständnis

Als Grundlage für ein System zur Programmierung mit gesprochener Sprache wird zunächst eine generische Architektur für tiefes Sprachverständnis entworfen. Die Architektur PARSE (kurz für ***P**rogramming **Ar**chitecture for **S**poken **E**xplanations*) kann für die Entwicklung beliebiger Systeme zur Analyse der Semantik natürlicher Sprache verwendet werden; die Programmierung mit gesprochener Sprache ist nur ein möglicher Anwendungsfall. PARSE sieht die Untergliederung in unabhängige (bzw. entkoppelte) Komponenten vor, sodass einzelne Aspekte der natürlichen Sprache getrennt voneinander analysiert werden können. Sprachanalysen werden als unabhängige Agenten implementiert, die nebenläufig ausgeführt werden. Über eine definierte Schnittstelle können zudem externe Wissensquellen angebunden werden. Die Architektur zeichnet sich durch folgende Merkmale gegenüber bestehenden Ansätzen aus:

- *Verwendung unabhängiger Agenten für Sprachanalysen*: Die Analyse der Semantik natürlichsprachlicher Artefakte erfolgt durch unabhängige Agenten. Während für die Analyse der Syntax geschriebener Sprache häufig Fließbänder verwendet werden, sind derzeitige Systeme für Sprachverständnis (das heißt zur Analyse der Semantik und Pragmatik) zumeist monolithisch und auf die Verwendung einer Technik festgelegt. Durch die Verwendung unabhängiger Agenten können statistische, sowie regel- und wissensbasierte Techniken kombiniert und verschiedentliche Ressourcen angebunden werden. Zudem können Systeme durch Austausch oder Hinzufügen von Agenten einfach erweitert bzw. angepasst werden.
- *Parallele und wiederholte Durchführung der Sprachanalysen*: Die Agenten führen ihre Sprachanalysen parallel und wiederholt aus. Während sequenzielle Fließbänder in der Computerlinguistik ein Standardkonstrukt darstellen, wurden die Möglichkeiten parallel ausgeführter Analyseschritte bisher kaum untersucht. Die parallele und wiederholte Ausführung der Analysen ermöglicht es, gegenseitige Abhängigkeiten aufzulösen. Dadurch kann ein tieferes Sprachverständnis erreicht werden.
- *Graphbasierte Darstellung der Sprachanalysen*: Die (Zwischen-)Ergebnisse der Agenten werden in einem Graphen (als eine Art Interlingua) dargestellt. Bestehende Ansätze erlauben keine Betrachtung von Zwischenergebnissen; es können lediglich Regelanwendungen oder Modelle (bei Nutzung maschinellen Lernens) betrachtet werden. Die Darstellung als Graph ist einerseits für den Menschen leicht verständlich; andererseits können auch programmatische Strukturen einfach abgebildet und weit entfernte semantische Abhängigkeiten dargestellt werden. Ebenso können Informationen aus externen Wissensquellen hinzugefügt werden.

Die Architektur gliedert sich, wie in Abbildung 1.1 dargestellt, in drei Hauptbestandteile:

1. eine sequenzielle Vorverarbeitung (in einem Fließband),
2. eine parallele Komponente für Sprachverständnis und
3. eine sequenzielle Nachverarbeitung (in einem weiteren Fließband).

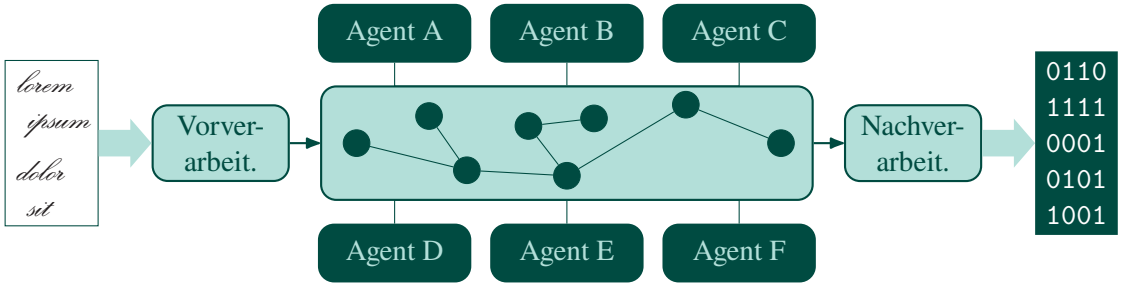


Abbildung 1.1: Struktureller Aufbau der agentenbasierten Architektur für tiefes Sprachverständnis PARSE.

Innerhalb der Komponente zur Vorverarbeitung wird das zu analysierende natürlichsprachliche Artefakt (beispielsweise ein Textdokument oder ein Audiosignal) für die semantische Analyse vorbereitet. Hierzu können Standardtechniken der Computerlinguistik angewendet werden, wie automatische Spracherkennung, oder Zerlegung der Eingabe in eine Wortsequenz. Das Ergebnis der Vorverarbeitung wird in eine Graph-Darstellung überführt. Die Komponente für Sprachverständnis bildet den Kern der Architektur. In unabhängigen Agenten können unterschiedliche Analyseaufgaben implementiert werden; die Architektur sorgt dafür, dass die Agenten parallel ausgeführt werden. Dadurch können sie potenziell von Analyseergebnissen anderer Agenten profitieren; unterschiedliche Sprachanalysen können sich so gegenseitig unterstützen. Analyseergebnisse werden von den Agenten als Änderungen am Graphen hinterlegt. Das endgültige gemeinsame Analyseergebnis aller Agenten wird schließlich an die sequenzielle Nachverarbeitung übergeben. Dort wird das Analyseergebnis interpretiert und eine Ausgabe generiert; welche Art von Ausgabe erzeugt wird, hängt vom konkreten Anwendungsfall des auf Basis der Architektur erstellten Systems ab. Das in dieser Arbeit entwickelte System zur Programmierung mit natürlicher Sprache erzeugt in der Nachverarbeitung beispielsweise Quelltext.

1.5 Ein System zur Endnutzer-Programmierung mit gesprochener Sprache

Auf Grundlage der Architektur PARSE wird im zweiten Teil der Arbeit das System ProNat zur Endnutzer-Programmierung mit gesprochener Sprache entwickelt. Hierzu werden konkrete Implementierungen für die durch die Architektur vorgesehenen Bestandteile (das heißt Agenten und Fließbandstufen) bereitgestellt. Informationen über die Anwendungsdomäne, beispielsweise die vom Zielsystem angebotenen Grundfunktionen, werden in Ontologien mit fester Struktur modelliert; dadurch bildet sich eine Art Schnittstelle zum Domänenwissen. Die Agenten und Fließbandstufen werden so entwickelt, dass sie die Struktur der Ontologien als vorausgesetzt annehmen, aber keine Annahmen über den Inhalt machen; beispielsweise ist bekannt, dass es eine Anwendungsschnittstelle gibt, die konkreten Funktionen müssen aber zur Laufzeit erfragt werden. Der Vorteil der Modellierung der Anwendungsdomäne in Ontologien ist, dass diese einfach ausgetauscht werden können; soll ProNat ein neues Zielsystem ansteuern, genügt es eine entsprechende Ontologie bereitzustellen. Der Rest des Systems kann nahezu unverändert weiterverwendet werden.

Da Nutzer Programme mit gesprochener Sprache beschreiben können sollen, wird die von PARSE vorgesehene Vorverarbeitung dazu genutzt, das Audiosignal zunächst in eine textuelle Form zu überführen. Anschließend wird die Syntax der Äußerung analysiert. Die Interpretation der Handlungsbeschreibung erfolgt durch ein Ensemble von Agenten. Insgesamt werden sechzehn Agenten verwendet, die unterschiedliche Aufgaben erfüllen. Die Agenten können drei Aufgabenfeldern zugeordnet werden: Ein Teil der Agenten übernimmt allgemeine Aufgaben zur Analyse der Semantik, wie die Auflösung von sprachlichen Referenzen oder die Modellierung des sprachlichen Kontextes. Weitere Agenten untersuchen die Äußerung auf Strukturen, die als Programmbestandteile interpretiert werden können, wie Anweisungen und Kontrollstrukturen. Zuletzt übernehmen Agenten funktionale Aufgaben, wie beispielsweise Dialogführung. Durch das Zusammenspiel der Agenten entsteht ein tiefes Verständnis der Äußerung und gleichzeitig eine Interpretation der Handlungsbeschreibung des Nutzers als Programm. ProNat verwendet die von PARSE vorgesehene Nachverarbeitung dazu, das Analyseergebnis der Agenten zunächst in einen abstrakten Syntaxbaum umzuwandeln. Dieser kann anschließend in Quelltext zur Ausführung auf einem beliebigen Zielsystem übersetzt werden. Das System ProNat zeichnet sich durch folgende Merkmale aus:

- *Verständnis uneingeschränkter Sprache*: Das Ensemble von Agenten erzeugt ein tiefes Verständnis *gesprochener* Sprache erzeugt. Die Unterteilung in unabhängige Agenten, die einzelne Aspekte der Sprache analysieren, erlaubt zielgerichtete und hoch spezialisierte Lösungen, die zusammengenommen eine umfassende Interpretation *gesprochener* Äußerungen ermöglichen. Dadurch ist es nicht nötig die natürliche Sprache einzuschränken; das bedeutet, Nutzer können beliebige Formulierungen zur Beschreibung ihrer Programme verwenden. Im Gegensatz zu bestehenden Lösungen ist ProNat außerdem in der Lage, nicht nur einzelne Befehle, sondern auch längere Befehlsketten samt Kontrollstrukturen zu verarbeiten. Zudem kann ProNat neue Funktionen erlernen und ist somit in der Lage, den Funktionsumfang der Anwendungsschnittstelle des Zielsystems zu erweitern.

- *Evaluations-getriebene Entwicklung*: Die Untergliederung des Systems in unabhängige Einzelbestandteile (das heißt Fließbandstufen und Agenten) ermöglicht es, diese auch unabhängig voneinander zu evaluieren. Dadurch kann für jeden neu entwickelten Bestandteil bestimmt werden, ob er die ihm zugedachte Aufgabe zufriedenstellend löst und somit einen Beitrag zur Sprachverständnisfähigkeit des Gesamtsystems leistet. Auf diese Weise können Fehlerquellen (bzw. Optimierungspotenzial) präzise bestimmt werden und es wird ein konstanter (Evaluations-getriebener) Entwicklungsfortschritt sichergestellt.

- *Domänenunabhängigkeit*: Die Sprachanalysen der Agenten sind zwar (teilweise) auf die Interpretation der Äußerungen als Programm ausgelegt; sie sind jedoch nicht auf ein konkretes Zielsystem oder eine Programmiersprache festgelegt. Dadurch ist ProNat in der Lage aus beliebigen Äußerungen Pseudo-Quelltext zu erzeugen. Informationen über das zu programmierende Zielsystem werden in einer Ontologie modelliert, deren festgelegte Struktur eine Art Schnittstelle zu Domänenwissen erzeugt. Erst während der Nachverarbeitung wird das Analyseergebnis der Agenten auf konkreten Quelltext abgebildet. Dadurch muss ProNat nur geringfügig angepasst werden, wenn ein anderes Zielsystem angesprochen werden soll. Insbesondere können alle Agenten unverändert weiterverwendet werden.

1.6 Thesen

Das System ProNat soll Laien befähigen, einfache Programme zu erstellen, die auf der Verwendung von Anwendungsschnittstellen beliebiger Zielsysteme beruhen. Die Programme sollen mit alltäglicher Sprache beschrieben werden können. Zudem soll das System es erlauben, neue Funktionen zu definieren, ebenfalls durch gesprochene Beschreibungen. ProNat basiert auf der Architektur PARSE, welche die Verwendung von unabhängigen Agenten zur Erzeugung von tiefem Sprachverständnis vorsieht. Das tiefe Verständnis der natürlichsprachlichen Programmbeschreibungen ermöglicht die Synthese von beliebig langen Aufruffolgen und die Erkennung von Kontrollstrukturen sowie die Generierung von Methodendefinitionen. Um prüfen zu können, ob die Zielstellung unter Anwendung des beschriebenen Ansatzes erfüllt wird, werden die folgenden Thesen aufgestellt:

T$_1$: Aus gesprochenen Äußerungen kann Quelltext zur Programmierung von Zielsystemen mit einer Endnutzer-Anwendungsschnittstelle synthetisiert werden.

- **T**$_{1.1}$: Anhand gesprochener Handlungsbeschreibungen können skriptartige Programme, die aus Aufrufen von Funktionen einer Anwendungsschnittstelle bestehen, erzeugt werden.
- **T**$_{1.2}$: Anhand lehrender Beschreibungen können Methodendefinitionen synthetisiert und dadurch der Funktionsumfang der Anwendungsschnittstelle eines Zielsystems erweitert werden.
- **T**$_{1.3}$: Es können automatisch Strukturen in natürlicher Sprache erkannt werden, welche die Verwendung von Kontrollstrukturen in einem Programm erfordern: Schleifen, bedingte Verzweigungen und nebenläufige Abschnitte können in die erzeugten Quelltexte integriert werden.
- **T**$_{1.4}$: Die Interpretation einer gesprochenen Äußerung als Programm kann größtenteils unabhängig vom konkreten Zielsystem durchgeführt werden. Dadurch ist der Aufwand zur Adaption an ein neues Zielsystem gering.

T$_2$: Die Verwendung unabhängiger Agenten für einzelne Sprachanalyseaufgaben, die nebenläufig und wiederholt ausgeführt werden, erzeugt bei geeigneter Auswahl der Agenten ein tiefes Verständnis gesprochener Sprache.

T$_3$: Durch gezielte Dialogführung können Unklarheiten bei der Interpretation einer gesprochenen Äußerungen beseitigt werden.

In den nachfolgenden Kapiteln werden zur Untersuchung der Thesen zunächst die Architektur PARSE und das System ProNat zur Endnutzer-Programmierung mit gesprochener Sprache beschrieben. Anschließend werden die Thesen diskutiert und geprüft, inwieweit die definierten Ziele erreicht werden konnten.

1.7 Aufbau der Arbeit

Die weitere Arbeit ist wie folgt strukturiert: Im nachfolgenden Kapitel werden zunächst die notwendigen Grundlagen zum Verständnis der Arbeit gelegt (Kapitel 2). Anschließend werden verwandte Arbeiten aus dem Gebiet der Programmierung mit natürlicher Sprache und daran angrenzender Themengebiete diskutiert (Kapitel 3). Als Nächstes wird die Architektur PARSE vorgestellt (Kapitel 4). Hierzu werden zunächst Anforderungen an eine Architektur für tiefes Verständnis natürlicher Sprache aufgestellt. Ausgehend von den Anforderungen wird der Aufbau von PARSE abgeleitet. Es folgt die Beschreibung des Systems ProNat zur Endnutzer-Programmierung mit natürlicher Sprache, das auf der Architektur PARSE basiert (Kapitel 5). Dabei werden die Zieldefinition präzisiert, der Forschungsansatz erläutert und ein Überblick über die Bestandteile (Fließbandstufen und Agenten) gegeben. Die Beschreibungen zu den konkreten Bestandteilen schließen sich in den darauffolgenden Kapiteln an. Zunächst wird die Vorverarbeitung einer natürlichsprachlichen Äußerungen in einem Fließband erläutert (Kapitel 6). Es folgt die Beschreibung der Analyse der Semantik durch die Agenten (Kapitel 7). Schließlich wird die Nachverarbeitung der Analyseergebnisse, die zur Synthese von Quelltext führt, dargelegt (Kapitel 8). Die drei letztgenannten Kapitel enthalten jeweils intrinsische Evaluationen der betrachteten Bestandteile. Zusätzlich wird das Zusammenspiel der Bestandteile im Gesamtsystem ProNat evaluiert (Kapitel 9). Zuletzt wird die Arbeit zusammengefasst (Kapitel 10). Dabei werden die aufgestellten Thesen betrachtet und geprüft, für welche die Arbeit Belege liefern konnte. Außerdem werden weiterführende Forschungsfragen diskutiert.

2 Grundlagen

„We believe that modern Natural Language Processing techniques can make possible the use of natural language to (at least partially) express programming ideas, thus drastically increasing the accessibility of programming to non-expert users.“

– Rada Mihalcea et al.

In diesem Kapitel werden die zum Verständnis der Arbeit nötigen Grundlagen erläutert. Da sich die Arbeit dem Forschungsgebiet der Endnutzer-Programmierung zuordnen lässt, wird dieses zunächst kurz eingeführt (Abschnitt 2.1). Anschließend werden die wichtigsten Konzepte des maschinellen Lernens beschrieben (Abschnitt 2.2). Zur Umsetzung des in dieser Arbeit vorgeschlagenen Ansatzes zur Endnutzer-Programmierung mit natürlicher Sprache werden eine Vielzahl von Theorien, Techniken und Verfahren der Computerlinguistik verwendet. Daher schließt sich eine eingehende Einführung in dieses Forschungsgebiet an (Abschnitt 2.3). Unter anderem werden Standardtechniken zur Syntaxanalyse, Datenbanken linguistischen Wissens und Datenmodelle zur Repräsentation linguistischer Informationen vorgestellt. Zuletzt werden Evaluationsmetriken eingeführt, auf die in den nachfolgenden Kapiteln Bezug genommen wird (Abschnitt 2.4).

2.1 Endnutzer-Programmierung

Die Endnutzer-Programmierung (engl. *end-user programming*, kurz *EUP* oder *end-user development*, kurz *EUD*) ist ein Forschungsgebiet der Informatik. Grundsätzlich verfolgt die Endnutzer-Programmierung das Ziel, Endnutzer ohne (oder mit geringer) Programmiererfahrung in die Lage zu versetzen, einfache Programme zu erstellen. Hierzu sollen die Nutzer nicht zunächst eine Programmiersprache erlernen müssen (oder wenn dann eine intuitive). Die Endnutzer-Programmierung als eigenständiger Forschungszweig wird erstmals 2006 von Liebermann et al. wie folgt definiert [Lie+06].

> „*End-User Development can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artifact.*“[1]

[1] Zu deutsch: Endnutzer-Programmierung kann definiert werden als eine Menge von Methoden, Techniken und Werkzeugen, die es Nutzern eines Softwaresystems, die keine professionellen Software-Entwickler sind, erlauben, ein Software-Artefakt zu erstellen, zu modifizieren oder zu erweitern.

Der Hauptgrund für das steigende Interesse an diesem Forschungsgebiet ist die stark waschende Zahl an Endnutzern in den letzten Jahrzehnten. Heutzutage kommen Menschen in ihrem Alltag ständig in Kontakt mit Computersystemen. Abseits von Arbeits- und Heim-PCs steigt die Anzahl persönlicher Computersysteme stetig, wie beispielsweise Smartphones und Tablets, oder Heimautomationssysteme etc. Allerdings ist nur ein geringer Teil der Nutzer in der Lage diese Systeme zu programmieren. Das volle Potenzial von Computersystemen kann aber nur genutzt werden, wenn diese auch programmiert werden können. Da nicht jeder Nutzer gewillt ist, Programmierung (und Programmiersprachen) vollumfänglich zu erlernen, wird die Endnutzer-Programmierung immer relevanter.

Unter dem Oberbegriff Endnutzer-Programmierung wird eine Vielzahl unterschiedlicher Strömungen zusammengefasst. Eine seit jeher von vielen Anwendern benutzte Form der Endnutzer-Programmierung ist die Erstellung von Formeln mittels Bibliotheksfunktionen in Tabellenkalkulation-Software. Auch die Erzeugung von Makros zählt zum Gebiet der Endnutzer-Programmierung. Eine weitere populäre Form ist die visuelle Programmierung (engl. *visual programming*), bei der Endnutzer in einer graphischen Benutzeroberfläche Programmkonstrukte, wie Anweisungen, Kontrollstrukturen und Methodensignaturen, die meist als Blöcke visualisiert werden, zusammenfügen können und dadurch Programme erstellen. Eine andere Herangehensweise wird bei der Programmierung anhand von Beispielen (engl. *programming by example*, kurz *PbE*) verfolgt. Hier definiert der Nutzer Ein- und Ausgabebeispiele für ein zu erstellendes Programm; das *PbE*-System versucht aus diesen Beispielen ein Programm zu inferieren. Je nach Verfahren kann der Nutzer interaktiv in den Vorgang eingreifen oder das Programm nachträglich modifizieren. Nicht zuletzt stellt auch die Programmierung mit natürlicher Sprache eine Form der Endnutzer-Programmierung dar (sofern es sich nicht um Verfahren zum Quelltextdiktat handelt). Dadurch, dass Nutzer Programme in Alltagssprache beschreiben können, statt eine spezielle Programmiersprache erlernen zu müssen, wird eine der hauptsächlichen Hürden beim Programmieren ausgeräumt. Davon abgesehen existieren auch Mischformen, wie beispielsweise die Unterstützung der Programmierung anhand von Beispielen durch textuelle Beschreibungen oder die Synthese von Makros zur Bearbeitung von Tabellen aus natürlichsprachlichen Äußerungen (siehe Kapitel 3). Darüber hinaus gibt es viele weitere Varianten der Endnutzer-Programmierung. Welche Systeme und Techniken hinzuzuzählen sind, hängt auch davon ab, wie restriktiv die zu Beginn des Abschnitts angegebene Definition von Lieberman et al. ausgelegt wird.

2.2 Maschinelles Lernen

Der Begriff *maschinelles Lernen* (engl. *machine learning*) beschreibt in der Informatik Programme, die durch Erfahrung ihre Leistung automatisch steigern [Mit97]. Das zugehörige Forschungsgebiet beschäftigt sich mit der Aufstellung von Theorien und Modellen sowie der Bereitstellung von Verfahren und Techniken. Der Großteil der Ansätze, die dem maschinellen Lernen zugeordnet werden, versuchen anhand von Beispieldaten Lösungen für Problemstellungen zu erlernen. Der Lernprozess wird häufig mithilfe mathematischer Modellierungen bewerkstelligt; dieser Lernprozess wird gemeinhin auch als Trainingsphase (engl. *training phase*) und die verwendeten Beispiele als Trainingsdaten (engl. *training data*) bezeichnet. Zumeist werden probabilistische Modellierungen

der Problemstellungen verwendet. Das Ergebnis des Lernprozesses (das heißt das erlernte Wissen) ist eine konkrete Modell-Instanz. Anhand des gelernten Modells können anschließend die Ausgaben für zukünftiger Eingabedaten erzeugt werden (das heißt, dass beispielsweise die Zugehörigkeit zu einer Klasse vorhergesagt wird). Die drei wesentlichen Varianten maschinellen Lernens sind die folgenden [Mit97]:

- **überwachtes maschinelles Lernen**: Unter überwachtem maschinellen Lernen (engl. *supervised maschine learning*) versteht man das Lernen aus Ein- und Ausgabebeispielen zu einer gewissen Problemstellung. Das bedeutet, anhand der Beispiele wird unter Verwendung einer mathematischen Modellierung eine Zuordnung von Ein- zu Ausgabe erlernt. Mithilfe des gelernten Modells können die Ausgaben für zukünftige (ungesehene) Eingabedaten vorhergesagt werden. Diese Form des Lernens wird als *überwacht* bezeichnet, da die Art der Ausgabe bekannt ist und für das Training des Modells Beispielausgaben (zu Eingabedaten) zur Verfügung stehen.

- **unüberwachtes maschinelles Lernen**: Unter unüberwachtem maschinellen Lernen (engl. *unsupervised maschine learning*) versteht man das Lernen ausschließlich anhand von Eingabedaten. Da keine erwarteten Ausgaben bekannt sind, wird diese Form des Lernen als *unüberwacht* bezeichnet. Da keine Ausgabedaten zum Abgleich verwendet werden können, wird stattdessen versucht, innerhalb der Eingabedaten Strukturen zu erkennen. Die häufigste Form des unüberwachten Lernens sind Verfahren, die Eingabedaten gruppieren (engl. *clustering*); anhand von Eigenschaften der Eingabedaten werden unter Verwendung einer mathematischen Modellierung Gruppenzugehörigkeiten erlernt.

- **bestärkendes maschinelles Lernen**: Unter bestärkendem Lernen (engl. *reinforcement learning*) versteht man das Lernen anhand von Rückmeldungen der Umgebung. Ansätze dieser Kategorie verwenden (einfache) initiale Modelle anhand derer sie Entscheidungen (bzw. Vorhersagen) treffen. Die Umgebung (meist ein weiteres Programm oder ein menschlicher Nutzer) gibt dem Verfahren Rückmeldung zur zuletzt getroffenen Entscheidung, im einfachsten Fall nur, ob diese *richtig* oder *falsch* war. Anhand dieser Rückmeldung wird das Modell automatisch adaptiert (meist gemäß einer mathematischen Modellierung).

Abgesehen von diesen Grundvarianten existiert eine Vielzahl abgeleiteter Variationen, wie beispielsweise halb-überwachtes Lernen (engl. *semi-supervised learning*) oder überwachtes Lernen anhand unüberwacht erzeugter Ausgabedaten (engl. *distant supervision*). Außerdem gibt es eine Reihe von Verfahren, die bestehende Ansätze optimieren. Unter anderem gibt es Verfahren, die erlernen, welche Eigenschaften (engl. *feature*) der Eingabe für die Problemstellung relevant sind (engl. *feature learning*). Ein weiterer Vertreter ist das sogenannte Meta-Lernen (engl. *meta learning*); Verfahren dieser Art erlernen ein Modell, das Vorhersagen darüber machen kann, welches Modell aus einer Menge von Modellen in einer bestimmten Situation (bzw. bei bestimmten Eingaben) verwendet werden sollte[2].

[2] Ein Meta-Lern-Ansatz wurde beispielsweise auch im Projekt *Watson* von *IBM* verfolgt [Fer+10]. Vereinfacht gesagt wurde eine Menge von Modellen erlernt, die Antworten zu Fragestellungen liefern können. Das Meta-Modell wiederum entscheidet anhand der Eingabe, welchem Modell vertraut werden soll.

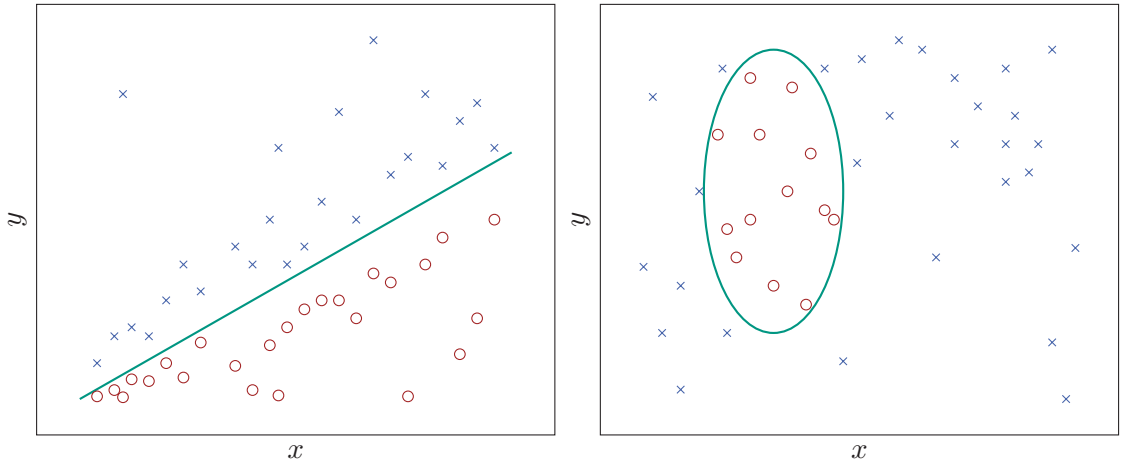
Die in den nachfolgenden Kapiteln beschriebenen Ansätze setzen (bei Verwendung von Lernverfahren) ausschließlich überwachtes Lernen ein. Auch die im direkt anschließenden Kapitel beschriebenen verwandten Arbeiten verwenden größtenteils überwachtes Lernen (sofern ein Lernverfahren verwendet wird). Daher wird im folgenden Abschnitt diese Variante des maschinellen Lernens näher betrachtet.

2.2.1 Überwachtes maschinelles Lernen

Überwachtes maschinelles Lernen bezeichnet das Erlernen anhand von Ein- und Ausgabebeispielen in einer Trainingsphase; einzelne Beispieldaten, die zum Trainieren verwendet werden, bezeichnet man als Trainingsinstanzen (engl. *training instance*). Das erlernte Wissen kann verwendet werden, um Vorhersagen über zukünftige, zuvor ungesehene Eingaben zu treffen. Als Eingabe werden üblicherweise keine Rohdaten verwendet (wie zum Beispiel nur die Wörter eines Textes); stattdessen wird aus den Rohdaten eine Menge sogenannter Eigenschaften (engl. *feature*) extrahiert. Für ein Wort eines Textes könnten solche Eigenschaften eine Aufzählung der umgebenden Wörter, die Wortlänge, aber auch das Wort selbst sein. Die gesammelten Eigenschaften je Datum werden häufig als Vektor dargestellt; man spricht dann von Eigenschaftsvektoren (engl. *feature vector*). Je nach Problemstellung ist die Ausgabe entweder eine diskrete Klasse oder ein kontinuierlicher numerischer Wert; man unterscheidet dementsprechend Klassifikations- und Regressionsprobleme. In beiden Fällen ist das Ziel, mithilfe einer mathematischen Modellierung Muster in den Eigenschaften der Eingabedaten zu finden, anhand derer die Ausgabe vorhergesagt werden kann. Das erlernte Wissen entspricht dementsprechend einer konkreten Modell-Instanz der allgemeinen Modellierung. Die Art der möglichen Modellierung hängt davon ab, ob ein Regressions- oder Klassifikationsproblem gelöst werden soll. Im Folgenden werden vorrangig Klassifikationen diskutiert. Ein Klassifikationsproblem ist beispielsweise die Zuweisung von Wortarten zu Wörtern anhand von Eigenschaften der Wörter, wie der Wortumgebung (siehe Abschnitt 2.3.4.3). Erfordert eine Problemstellung die Verwendung von genau zwei Klassen (beispielsweise *wahr* und *falsch*), spricht man von einem binären Klassifikationsproblem (engl. *binary classification*). Sind hingegen mehrere Klassen erforderlich (wie beispielsweise bei der Bestimmung von Wortarten als Ausgabeklassen), handelt es sich um ein Mehrfachklassen-Klassifikationsproblem (engl. *multiclass classification*)[3].

Konkrete Verfahren des überwachten maschinellen Lernens zur Klassifikation unterscheidet man vor allem dahingehend, welche mathematische Modellierung zugrunde gelegt wird. Das grundsätzliche Prinzip ist dabei immer dasselbe; mithilfe der mathematischen Modellierung soll für die Eingabebeispiele anhand ihrer Eigenschaften eine bestmögliche Trennung ermittelt werden, die der Zuordnung zu den Ausgabeklassen entspricht. Bestmöglich bedeutet in diesem Zusammenhang, dass durch die erzeugte Trennung möglichst wenige Eingabeinstanzen falschen Ausgabeklassen zugeordnet werden. Die Annahme, die diesem Vorgehen zugrunde liegt, ist folgende: Zukünftige Eingaben mit ähnlichen Eigenschaften, lassen sich auf dieselbe Art trennen und somit Klassenzugehörigkeiten vorhersagen.

[3] Weiterhin unterscheidet man Klassifikationsprobleme dahingehend, ob einem Eingabedatum genau eine Klasse zugewiesen werden darf, oder ob mehrere Klassen zulässig sind. Im zweiten Fall spricht man von einer sogenannten Mehrfach-Etiketten-Klassifikation (engl. *multi-label classification*).



(a) linear **(b)** nicht-linear

Abbildung 2.1: Zwei beispielhafte Trennungen von Trainingsinstanzen gemäß der erwarteten Ausgabeklassen: Abbildung (a) zeigt ein lineares und Abbildung (b) ein nicht-lineares Klassifikationsproblem. Die Trennungen sind jeweils durch eine grüne Linie symbolisiert.

Abbildung 2.1 zeigt zwei Grafiken zur Veranschaulichung. In beiden Fällen werden Eingabedaten im zweidimensionalen Raum[4] gemäß ihrer Klassenzugehörigkeit getrennt. Die grünen Linien symbolisieren die Trennung und somit das erlernte Modell; in beiden Fällen trennen die Modelle die Trainingsbeispiele perfekt[5]. Die erlernte Trennung entspricht der konkreten Modell-Instanz. Je nach Verfahren schließt sich an die Trainingsphase eine sogenannte Validierungsphase (engl. *validation phase*) an. In diesem Fall wird nur ein Teil der Beispieldaten (meist 80%) zum Training verwendet; anhand der übrigen Daten wird das Modell getestet und Modellparameter werden automatisch angepasst, falls Fehlklassifikationen festgestellt werden. Abbildung 2.1 veranschaulicht außerdem die Unterscheidung nach linearen und nicht-linearen Klassifikationsproblemen. Vereinfacht gesagt, bezeichnete man ein Klassifikationsproblem als *linear*, falls die Daten (in Annäherung) durch eine Hyperebene getrennt werden können; im zweidimensionalen Raum entspricht dies einer Geraden. *Nicht-lineare* Klassifikationsprobleme lassen sich nicht auf diese Weise trennen; es gibt keine Gerade die eine gute Trennung ermöglicht. Ob es sich um ein lineares oder nicht-lineares Klassifikationsproblem handelt, hat direkten Einfluss auf die möglichen mathematischen Modellierungen, die verwendet werden können. Einige Modellierungen eignen sich ausschließlich zur Darstellung linearer Zusammenhänge. Andere können zwar für beide Arten von Klassifikationsproblemen verwendet werden, erzielen aber deutlich bessere Ergebnisse (das heißt bessere Trennungen) für eine der beiden. Weitere können sowohl mit linearen als auch mit nicht-linearen Problemstellungen gleichermaßen umgehen, benötigen aber sehr viele Ressourcen; beispielsweise benötigen einige Verfahren mehr Trainingsdaten als andere, die Trainingsphase nimmt mehr Zeit in Anspruch oder sie erzeugen größere Modelle (im Sinne des Speicherbedarfs).

[4] Das bedeutet, die Eigenschaftsvektoren der Eingabedaten sind in diesem Beispiel zweidimensional.

[5] Anhand einer perfekten Trennung der Trainingsbeispiele durch das erlernte Modell kann keine Aussage über die Fähigkeit des Modells, ungesehene Eingaben zu trennen, getroffen werden. Das gelernte Modell kann trotz perfekten Trainings beliebig schlecht sein. Perfekte Trainingsgenauigkeiten werden vielmehr häufig kritisch gesehen, da sie ein Indiz für eine mögliche Überanpassung des Modells sind (mehr hierzu folgt am Ende des Abschnitts).

2.2.1.1 Die Modellierung als Einflussfaktor für die Klassifikationsgüte

Da je nach Klassifikationsproblem und Randbedingungen (beispielsweise Umfang der Trainingsdaten oder Speicher-Ressourcen) die Verwendung einer anderen Modellierung die bestmögliche Trennung (und damit Klassifikation) erzielen kann, gibt es eine Vielzahl mathematischer Modellierungen für überwachtes maschinelles Lernen. Die im Folgenden kurz vorgestellten Klassen von Modellierungen werden häufig (vor allem auch in der Computerlinguistik) verwendet und werden auch in den nachfolgenden Kapiteln für unterschiedliche Problemstellungen eingesetzt[6].

Naïve-Bayes

Verfahren dieser Klasse formulieren die Aufgabe der Zuweisung von Klassen zu Eingabedaten als einfaches probabilistisches Modell. Das Modell besagt, dass die Wahrscheinlichkeit P für eine Klasse k nur von den Merkmalen der beobachteten Eingabe (Eigenschaftsvektor) $\vec{e}$ abhängt:

$$P(k|\vec{e}) \tag{2.1}$$

Da Eigenschaftsvektoren viele Dimensionen besitzen und die einzelnen Dimensionen viele unterschiedliche Werte annehmen können, ist eine Berechnung der Werte anhand einer Wahrscheinlichkeitstabelle zu aufwendig. Stattdessen wird die Modellformulierung mithilfe des Satzes von Bayes umgeformt:

$$P(k|\vec{e}) = \frac{P(k) \cdot P(\vec{e}|k)}{P(\vec{e})} \tag{2.2}$$

Da der Zähler nicht von k abhängt, wird dieser zur einfacheren Berechenbarkeit durch einen konstanten skalierenden s Faktor ersetzt. Zusätzlich wird vereinfachend (bzw. naiv) angenommen, dass die einzelnen Eigenschaften unabhängig voneinander sind. Dadurch kann das Problem wie folgt angenähert werden:

$$P(k|\vec{e}) = \frac{1}{s} \cdot P(k) \prod_{i=1}^{n} P(e_i|k) \tag{2.3}$$

Auf dieser Grundlage können anhand der Trainingsdaten Modelle gelernt werden, indem einerseits die Vorkommen der Ausgabeklasse (entspricht $P(k)$) und anderseits die Vorkommen einer Klasse gemeinsam mit einer Eigenschaft (entspricht $P(e_i|k)$) ermittelt werden (Maximum-Likelihood-Schätzung).

Naïve-Bayes-Klassifikationsverfahren skalieren hinsichtlich des Trainingsaufwands und der Modellgröße gut für große Datenmengen und Eigenschaftsvektoren mit vielen Dimensionen bzw. großen Wertebereichen. Da sie zudem – trotz der naiven Unabhängigkeitsannahme, die in den meisten Fällen inkorrekt ist – gute Ergebnisse erzielen können, werden sie häufig als erste Variante für neuartige Klassifikationsprobleme eingesetzt. Im direkten Vergleich mit den nachfolgenden Modellierungen ist die Güte der Klassifikation (unter denselben Randbedingungen) jedoch meist schlechter.

[6] Nähere Informationen zu diesen und weiteren Klassifikationsverfahren können den Büchern von Mitchell [Mit97] und Bishop [Bis06] entnommen werden.

Entscheidungsbäume

Verfahren dieser Klasse verwenden zur Modellierung des Klassifikationsproblems Entscheidungsbäume (engl. *decision trees*). Da sowohl die Struktur der Bäume als auch die Entscheidungskriterien innerhalb der Knoten automatisch erlernt werden, spricht man von Entscheidungsbaumlernen (engl. *decision tree learning*). Die Trennung der Eingaben erfolgt anhand einer Reihe von Wenn-Dann-Regeln, die innerhalb der Knoten des Baums angewendet werden. Die Regeln beziehen sich auf einzelne Eigenschaften oder eine Menge von Eigenschaften der Trainingsinstanzen.

Verfahren, die auf Entscheidungsbäumen basieren, haben den Vorteil, dass sie in der Regel auch gute Modelle erlernen können, wenn nur wenige Trainingsbeispiele zur Verfügung stehen. Außerdem können konkrete Modelle (und auf diesen basierende Klassifikationsergebnisse) einfach interpretiert und nachvollzogen werden. Andererseits gelten Entscheidungsbäume als nicht besonders robust; bereits geringfügige Änderungen der Trainingsinstanzen können zu gänzlich anders geformten Modellen führen. Außerdem neigen Entscheidungsbaum-basierte Verfahren zur Überanpassung auf die Trainingsdaten und generalisieren daher schlechter als andere Verfahren.

Stützvektormethode

Verfahren, welche auf der Stützvektormethode (engl. *support vector machine*, kurz *SVM*) basieren, trennen die Trainingsinstanzen anhand einer Hyperebene. Dabei wird während des Trainings diejenige Hyperebene bestimmt, welche die Trainingsinstanzen gemäß ihrer Klassenzugehörigkeit bestmöglich trennt und im Mittel den größtmöglichen Abstand zu den Vektoren aufweist. Die optimale Hyperebene kann entweder exakt bestimmt oder angenähert werden (beispielsweise durch das Gradientenverfahren). Die aus dem Training hervorgegangene Hyperebene wird durch ihre Stützvektoren beschrieben, die namensgebend für diese Klasse von Verfahren sind. Da die Hyperebene exakt bestimmt werden kann und keine Annahmen über die Verteilung der Instanzen des Datensatzes getroffen werden müssen, ist die Stützvektormethode nicht probabilistisch. Durch die Verwendung einer Hyperebene können nur lineare Klassifikationsprobleme gelöst werden. Um die Stützvektormethode auch auf nicht-lineare Probleme anwenden zu können, werden die Eigenschaftsvektoren in einen höherdimensionalen Raum projiziert, in dem eine Trennung möglich ist[7].

Klassifikationsverfahren, welche die Stützvektormethode verwenden, gelten als sehr robust und erzielen (auch für Problemstellungen der Computerlinguistik) gute Klassifikationsergebnisse. Die Stützvektormethode kann allerdings nur auf binäre Klassifikationsprobleme angewendet werden; zur Anwendung auf Mehrfachklassen-Probleme müssen diese auf binäre zurückgeführt werden, wodurch zusätzlicher Trainingsaufwand entsteht und deutlich speicherintensivere Modelle erzeugt werden[8].

[7] Die Projektion erfolgt aufgrund des Aufwands, der damit einhergeht, meist nur implizit mithilfe des sogenannten Kernel-Tricks.

[8] Sollen n Klassen unterschieden werden, müssen n binäre Modelle trainiert werden, was in etwa einen n-fachen Zeit- und Speicheraufwand nach sich zieht.

Logistische Regression

Entgegen der Namensgebung werden Verfahren dieser Klasse für Klassifikations- und nicht für Regressionsprobleme eingesetzt. Die Klassifikation wird auf Grundlage einer Regressionskurve bewerkstelligt. Diese wird anhand der Trainingsinstanzen gebildet, indem die Parameter einer logistischen Funktion abgeschätzt werden. Die erzeugte Kurve dient zur Trennung der Klassenzugehörigkeit und bildet somit die Modell-Instanz. Die logistische Regression kann nicht nur für binäre, sondern auch für Mehrfachklassen-Klassifikationsprobleme verwendet werden. Klassifikationsverfahren, die auf logistischer Regression basieren, werden vor allem in der Computerlinguistik häufig eingesetzt (siehe Abschnitt 2.3) und erzielen für diverse Problemstellungen, wie beispielsweise die Erkennung von Wortarten, sehr gute Ergebnisse[9]. Ein Grund hierfür ist, dass zur Anwendung von logistischer Regression nicht wie bei Naïve-Bayes-Verfahren die Unabhängigkeit der Eigenschaften angenommen werden muss; viele sprachliche Eigenschaften, wie beispielsweise Wortumgebungen, sind stark voneinander abhängig. Ein wesentlicher Nachteil der logistischen Regression ist der deutlich höhere Trainingsaufwand gegenüber vergleichbaren Ansätzen, wie Naïve-Bayes-Verfahren.

2.2.1.2 Weitere Einflussfaktoren

Neben der Wahl der Modellierung gibt es eine Reihe weiterer Einflussfaktoren, welche die Güte des erlernten Klassifikationsmodells beeinflussen. Ein wesentlicher Faktor ist der Umfang des Trainingsdatensatzes. Im Allgemeinen gilt: je mehr Trainingsdaten zur Verfügung stehen, desto besser. Verfügt das Verfahren in der Trainingsphase über mehr Ein- und Ausgabebeispiele, kann das Modell feiner justiert werden und generalisiert besser. Das Problem hierbei ist, dass Ausgabebeispiele (das heißt Musterklassifikationen) zumeist manuell erstellt werden müssen, was einen erheblichen Aufwand mit sich bringt (siehe Abschnitt 2.3.3). Dementsprechend stehen für viele Problemstellungen, die mit überwachtem maschinellen Lernen gelöst werden könnten, nicht ausreichend Trainingsdaten zur Verfügung. Das Hauptproblem des überwachten maschinellen Lernens ist also die inhärente Datenknappheit.

Ein weiterer Faktor ist die Auswahl der Daten; diese sollte hinsichtlich der Problemstellung möglichst repräsentativ sein. Im besten Fall treten alle Ausgabeklassen gleich verteilt auf. Werden die Daten nicht repräsentativ gewählt oder ist der Umfang zu gering, sodass das Modell anhand von wenigen Beispielen (je Klasse) erlernt wird, kommt es zu einer sogenannten Überanpassung (engl. *overfitting*) des Modells auf die Eingabebeispiele. Anschaulich bedeutet das, die Klassenzuweisungen werden auswendig gelernt und das gelernte Modell generalisiert schlecht. Die Folge davon ist, dass die Vorhersagen für ungesehene Beispiele ungenau sind, da bereits geringe Abweichungen von den Trainingsbeispielen (in den Eigenschaften) vom Modell nicht mehr erfasst werden können.

Ein letzter an dieser Stelle diskutierter Einflussfaktor ist die Dimension der Eigenschaftsvektoren. Auch hier gilt grundsätzlich, je mehr, desto besser; stehen mehr Eigenschaften der Eingaben zur

[9] Vor dem neuerlichen Aufkommen von künstlichen neuronalen Netzen (siehe Abschnitt 2.2.2) wurden für viele Aufgabenstellungen der Computerlinguistik die besten Ergebnisse durch Klassifikatoren erzielt, die auf logistischer Regression basieren.

Verfügung, werden diese prinzipiell auch leichter unterscheidbar. Allerdings führen höherdimensionale Eigenschaftsvektoren zu einer Steigerung der Komplexität der Modelle (je nach Modellierung fallen die Steigerungen unterschiedlich stark aus). Ein weiterer Nachteil der Verwendung vieler Eigenschaften ist, dass mit steigender Anzahl der Dimensionen die aufgespannten Räume immer dünner (durch die Vektoren) besetzt werden. Diesen Effekt bezeichnet man als *Fluch der Dimensionalität* (engl. *curse of dimensionality*). Dünn besetzte Räume sind für viele Modellierungen problematisch, beispielsweise für solche, die Abstandsmaße für Vektoren zur Trennung verwenden. Zuletzt kann eine Vielzahl schlecht gewählter Eigenschaften auch dazu führen, dass die Trennung der Eingaben nicht mehr möglich ist; anschaulich entsteht eine Art weißes Rauschen, das informative Eigenschaften überdeckt. Eigenschaften gelten dann als schlecht gewählt, wenn sie keinen Beitrag zur Trennbarkeit der Eingaben gemäß der Problemstellung liefern. Für die Bestimmung von Wortarten ist beispielsweise die Wortlänge eine schlecht gewählte Eigenschaft. Auch von diesem Effekt werden Verfahren je nach zugrundeliegender mathematischer Modellierung unterschiedlich stark negativ beeinflusst. Aufgrund dieser Problematiken ist eine sorgfältige Auswahl der Eigenschaften unerlässlich. Um die Auswahl zu unterstützen, werden eigens Verfahren entwickelt, die (zumeist unüberwacht) erlernen, welche Eigenschaften für eine konkrete Klassifikationsaufgabe relevant sind.

2.2.2 Künstliche neuronale Netze

Als künstliche neuronale Netze (engl. *artificial neural network*) bezeichnet man Verfahren des maschinellen Lernens, die eine Problemstellung – inspiriert von einem biologischen Gehirn – mithilfe von miteinander verbundenen Neuronen modellieren [Mit97]. In dem aus Neuronen und Verknüpfungen aufgespannten Netz können Informationen verarbeitet werden, indem diese von Neuronen über die Verknüpfungen an andere Neuronen weitergereicht werden. Eine Art der Informationsverarbeitung ist das Treffen einer Entscheidung anhand einer gegebenen Datenlage, was einer Klassifikation, wie im vorangegangen Abschnitt eingeführt, entspricht. Das Konzept der neuronalen Netze ist älter als viele andere Modellierungen, die für maschinelles Lernen verwendet werden. Bereits 1943 wurden die mathematischen Grundlagen gelegt, 1965 das erste funktionierende künstliche neuronale Netze vorgestellt und seit 1975 können sie effizient trainiert werden [Bis06]. Aber erst in den letzten zehn Jahren gewinnen künstliche neuronale Netze zunehmen an Popularität; der Hauptgrund hierfür ist, dass nur Netze, die aus einer großen Anzahl von Neuronen bestehen und auf umfangreichen Datensätzen trainiert wurden, anderen Verfahren deutlich überlegen sind. Für das Training und die Anwendung leistungsstarker neuronaler Netze werden dementsprechend Rechenleistungen und Speicher-Ressourcen benötigt, die erst durch moderne Computersysteme (und vor allem verteilte Systeme) bereitgestellt werden können.

Künstliche neuronale Netze sind prinzipiell folgendermaßen aufgebaut [Mit97; Bis06]: Die wesentliche Baueinheit sind Neuronen, die miteinander verknüpft sind. Neuronen nehmen Eingaben entgegen und können Ausgaben an verknüpfte Neuronen senden. Sie entscheiden eigenständig darüber, ob eine Ausgabe generiert wird und welche. Die Entscheidung hängt von der Summe der Eingaben an das Neuron ab. Diese Summe dient wiederum als Eingabe an eine sogenannte Aktivierungsfunktion (engl. *activation function*), anhand derer die letztliche Entscheidung getroffen wird. Die Aktivierungsfunktion kann prinzipiell beliebig gewählt werden, es gibt allerdings eine

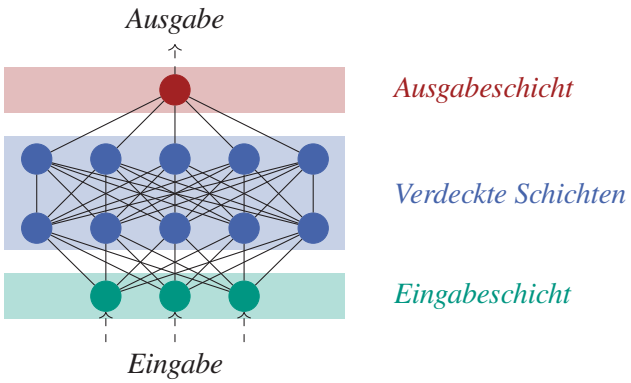


Abbildung 2.2: Schematischer Aufbau eines künstlichen neuronalen Netzes.

Reihe etablierter Funktionen, unter anderem lineare, logistische (auch als *Sigmoid*-Funktion bezeichnet) und die sogenannte *Softmax*-Funktion. Letztere eignet sich vor allem zur Generierung der Ausgabe bei Mehrfachklassen-Klassifikationsproblemen. Die Verbindungen zu anderen Neuronen sind gerichtet und gewichtet. Die zuvor beschriebene Summe der Eingaben wird entsprechend der Verbindungsgewichte gewichtet. Neuronen werden üblicherweise in Schichten angeordnet, wobei nur Neuronen nebeneinanderliegender Schichten verknüpft werden. Grundsätzlich wird vollständig verknüpft; das bedeutet, alle Neuronen der einen Schicht werden mit allen der anderen verbunden. Alle künstlichen neuronalen Netze verfügen über zwei spezielle Schichten: die Eingabe- und die Ausgabeschicht. Die Neuronen der Eingabeschicht sind dafür zuständig, die Eingaben (üblicherweise in Form von Eigenschaftsvektoren) entgegenzunehmen und weiter zu propagieren. Die Neuronen in der Ausgabeschicht erzeugen eine Ausgabe, beispielsweise eine Klassifikation. Zwischen diesen Schichten können (müssen aber nicht) beliebig viele weitere Schichten angeordnet werden. Diese Zwischenschichten werden auch als verdeckte Schichten (engl. *hidden layer*) bezeichnet. Abbildung 2.2 zeigt ein beispielhaftes künstliches neuronales Netz. Durch die Konstruktion von immer *tieferen* Netzen durch das Hinzufügen weiterer Schichten wurde der Begriff *tiefes Lernen* (engl. *deep learning*) geprägt.

Künstliche neuronale Netze werden trainiert, indem die Gewichtungen (und gegebenenfalls weitere anpassbare (Schwellen-)Werte), automatisch angepasst werden [Mit97]. Hierzu wird das Netz zunächst mit zufälligen Gewichten konfiguriert. Anschließend wird anhand des Trainingsdatensatzes (Ein- und Ausgabebeispiele) der Anteil der Fehlklassifikationen bestimmt. Die Gewichte aller Kanten werden daraufhin (ausgehend von der Ausgabeschicht) gezielt mithilfe des sogenannten Rückpropagierungsverfahrens (engl. *backpropagation*) angepasst. Dieses verwendet wiederum das Gradientenabstiegsverfahren zur Minimierung der Fehlklassifikationen.

Für den Aufbau und das Training von künstlichen neuronalen Netzen müssen Werte für die sogenannten Hyperparameter des Netzes gewählt werden. Die Wahl der Hyperparameter kann die Klassifikationsgüte der Netze stark beeinflussen. Welche Hyperparameter definiert werden müssen, hängt auch von der Art des Netzes ab. Im Folgenden werden die wichtigsten Hyperparameter kurz erläutert [Mit97; GBC16]:

- **Anzahl der Neuronen einer Schicht**: Für jede verdeckte Schicht eines neuronalen Netzes muss die Anzahl der enthaltenen Neuronen definiert werden. Die Anzahl der Neuronen der Ein- und Ausgabeschichten ergibt sich aus der Form der Eingabe bzw. der Anzahl der (Ausgabe-)Klassen.
- **Epochen**: Eine Epoche entspricht einem vollständigen Durchlauf durch den Trainingsdatensatz (zur Optimierung der Kantengewichte mittels Rückpropagierungsverfahrens). Da ein Durchlauf im Allgemeinen nicht ausreicht, um ein gutes Ergebnis zu erzielen, kann definiert werden, wie viele Epochen ein Netz (maximal) trainiert werden soll. Das Trainieren über viele Epochen ermöglicht eine bessere Justierung der Kantengewichte, erhöht allerdings auch den Trainingsaufwand und birgt die Gefahr einer Überanpassung.
- **Stapel-Größe**: Ein Stapel (engl. *batch*) ist die Menge von Trainingsinstanzen, die dem Netz präsentiert werden, bevor die Kantengewichte angepasst werden. Über den Hyperparameter Stapel-Größe (engl. *batch size*) kann somit definiert werden, wie viele Trainingsinstanzen jeweils entnommen werden sollen. Eine Epoche wird somit in mehrere Stapel unterteilt. Kleinere Stapel-Größen führen häufig zu besseren Ergebnissen, steigern den Trainingsaufwand allerdings drastisch.
- **Ausfall-Wert**: Die sogenannte Verdünnung (engl. *dilution*) ist eine Technik zur Vermeidung einer Überanpassung des Netzes an die Trainingsdaten. Dabei wird ein (zufälliger) Teil der Neuronen vor dem nächsten Trainingsdurchgang (Stapel) deaktiviert (das heißt diese Neuronen fallen aus). Über den Hyperparameter Ausfall-Wert (engl. *drop-out*) kann bestimmt werden, wie groß der Anteil ausfallender Neuronen im Netz sein soll. Verdünnung ist eine effektive Technik zur Vermeidung einer Überanpassung; allerdings werden (meist deutlich) mehr Trainingsepochen benötigt, wenn der Ausfall-Wert erhöht wird.
- **Lernrate**: Die Lernrate (engl. *learning rate*) ist ein skalierender Faktor für den Gradientenabstieg. Wird eine geringe Lernrate gewählt, können die Kantengewichte feiner justiert werden, allerdings konvergiert das Gradientenabstiegverfahren langsamer; es werden wiederum mehr Trainingsepochen benötigt. Wird die Lernrate hingegen zu groß gewählt, konvergiert das Verfahren nicht.

Künstliche neuronale Netze werden hinsichtlich ihrer Netzorganisation üblicherweise in drei unterschiedliche Kategorien eingeteilt. Die erste Kategorie bilden einfache vorwärts gerichtete neuronale Netze (engl. *feed-foward neural network*). Diese reichen Informationen von Schicht zu Schicht ausschließlich vorwärts gerichtet weiter. Da diese Netzorganisation das Konzept des künstlichen neuronalen Netzes in seiner Grundform darstellt, wird diese Kategorie auch häufig einfach als künstliche neuronale Netze (engl. *artificial neural network*, kurz *ANN*) bezeichnet. Die zweite Kategorie bilden faltende neuronale Netze (engl. *convolutional neural network*, kurz *CNN*). Faltende neuronale Netze bestehen häufig aus einer Vielzahl von Schichten, wobei es spezialisierte Schichten gibt, deren Neuronen eine Faltung durchführen; das bedeutet, die Aktivierungsfunktionen der Neuronen dieser Schichten entsprechen einer mathematischen Faltungsoperationen[10]. Faltende

[10] Faltende neuronale Netze verfügen daher über mindestens einen weiteren Hyperparameter, welcher die Dimension der quadratischen Faltungsmatrix (auch Filter- oder Faltungskern, engl. *convolution kernel*) festlegt.

Schichten sind untereinander nicht vollständig verknüpft. Die letzte Kategorie bilden rekurrente neuronale Netze (engl. *recurrent neural network*, kurz *RNN*). Diese Netzorganisation erlaubt Verknüpfungen von Neuronen derselben Schicht oder sogar Rückverbindungen zu Neuronen aus vorangegangen Schichten. Auf diese Weise können Reihenfolgeeffekte (das heißt Abhängigkeiten von vorherigen Eingaben) besser erfasst werden.

Zuletzt werden in diesem Abschnitt zwei spezielle Architekturen rekurrenter neuronaler Netze näher betrachtet, die häufig für Problemstellung im Kontext der Computerlinguistik eingesetzt werden: Netze mit einem sogenannten langen Kurzzeitgedächtnis (engl. *long short-term memory*, kurz *LSTM*) [HS97; GSC00; Gre+17] und Netze mit gesteuerten rekurrenten Einheiten (engl. *gated recurrent unit*, kurz *GRU*) [Cho+14]. Beide Architekturen lösen ein wesentliches Problem von *RNNs*: Rekurrente neuronale Netze sind zwar in der Lage (durch Verbindung von benachbarten Neuronen einer Schicht) Kontexte zu erfassen, der Kontextumfang ist allerdings sehr gering. *LSTMs* ermöglichen hingegen die Erkennung von Zusammenhängen über größere Distanzen (z. B. können Wörter mit Wörtern in vorangegangen Sätzen in Verbindung gebracht werden). Hierzu werden spezialisierte Neuronen in sogenannten Zellen (engl. *cells*) angeordnet. Jede Zelle hat einen inneren Zustand (definiert durch die Gewichte der Verknüpfungen der enthaltenen Neuronen). Die Zellen werden wiederum in einer Kette angeordnet und bilden gemeinsam das Gedächtnis des Netzes. Jede Zelle kann Informationen entgegennehmen, weiterreichen und vergessen. Durch die Zustandshaltung können Informationen prinzipiell beliebig lang weitergetragen werden. Aus technischer Sicht wird der Informationsfluss über spezielle Ein- und Ausgänge gesteuert; diese entsprechen speziell verknüpften Neuronen der Zelle. Es gibt zwei Eingänge in eine Zelle: einen, der aktuelle Eingaben (beispielsweise ein Wort) entgegen nimmt (engl. *input gate*), und einen weiteren, der bestimmt, welche alten Informationen beibehalten und welche vergessen werden sollen (engl. *forget gate*). Anhand dieser beiden Eingaben und dem inneren Zustand der Zelle, wird eine Ausgabe erzeugt, die über eine Ausgabeverknüpfung (engl. *output gate*) an die nächste Zelle weitergereicht wird. Während der Trainingsphase werden die optimalen Gewichte für die gesteuerten Verknüpfungen (Ein- und Ausgabe, sowie Erinnerung und Zustand) erlernt.

GRUs bilden eine moderne Alternative zu *LSTMs*. Sie verfolgen prinzipiell denselben Ansatz mit verketteten Zellen. Allerdings werden nur zwei gesteuerte Verknüpfungen verwendet. Der Aktualisierungseingang (engl. *update gate*) übernimmt die Funktionalitäten des *input gate* und des *forget gate* bei einem *LSTM*. Ein weiterer Rücksetzungseingang (*reset gate*) bestimmt zusätzlich, wann (sehr) alte Informationen vergessen werden sollten. Da nur zwei gesteuerte Verknüpfungen verwendet werden, ist der Trainingsaufwand bei *GRUs* geringer. Allerdings sind die Klassifikationsergebnisse (je nach Problem) schlechter als bei Verwendung eines *LSTMs*.

Sowohl für *LSTMs* als auch für *GRUs* existieren bidirektionale Architekturvarianten. Das bedeutet nichts anderes, als dass eine zweite Schicht verketteter Zellen eingeführt wird, welche die Eingabe in entgegengesetzter Richtung verarbeiten. Die Zellen der beiden Schichten werden wiederum untereinander verknüpft. Der Vorteil der Verwendung bidirektionaler Architekturen ist, dass auf diese Weise beidseitige Kontexte in die Entscheidungsfindung einbezogen werden können; beispielsweise können zur Klassifikation eines Wortes nicht nur vorangegangene Wörter, sondern auch Information, die später im Text auftreten, verwendet werden.

2.3 Computerlinguistik

Die Computerlinguistik (engl. *computational linguistics*), auch bezeichnet als Rechnerlinguistik oder Sprachverarbeitung (engl. *natural language processing*, kurz *NLP*)[11], ist die Lehre von der Aufstellung und Anwendung von Theorien und Modellen, die natürliche Sprache beschreiben und auf Computersystemen ausgeführt werden können [JM09b]. Die Modelle werden mit dem Ziel angewendet, natürliche Sprache zu analysieren und zu formalisieren. Untersucht werden Chararkteristika natürlichsprachlicher Einheiten (Wörter, Phrasen, Sätze etc.). Hierzu werden Sprachanalyseaufgaben definiert, die einzelne Eigenschaften der Sprache untersuchen, wie beispielsweise Wortarten oder die Satzstellung. Die Sprachanalysen entsprechen Algorithmen, die auf einem Computersystem ablaufen können. Auf diese Weise können Informationen aus natürlichsprachlichen Artefakten, wie Textdokumenten oder gesprochenen Äußerungen, gewonnen werden.

2.3.1 Syntaktik, Semantik und Pragmatik

Charakteristika natürlicher Sprache können auf unterschiedlichen Ebenen beschrieben werden. Man unterscheidet zwischen Syntax, Semantik und Pragmatik. Unter syntaktischen Eigenschaften eines natürlichsprachlichen Artefakts versteht man dessen strukturelle Eigenschaften. Die Syntax einer Sprache beschreibt dementsprechend das Regelwerk, das bestimmt, wie Wörter (oder größerer Einheiten wie Phrasen) zueinander stehen, sprich den grammatikalischen Aufbau von Sätzen. Semantische Eigenschaften sind hingegen solche, welche die Bedeutung einzelner natürlichsprachlicher Einheiten beschreiben. Dies kann die Bedeutung eines Wortes im Kontext sein[12] oder auch die Interpretation eines Satzes. Die Pragmatik beschreibt hingegen die satzübergreifende und äußere Bedeutung. Das bedeutet, es werden Bedeutungen untersucht, die sich nur im Kontext eines Diskurses (beispielsweise mehrere aufeinanderfolgende Sätze oder ein Dialog) erschließen. Dementsprechend betrachtet die Pragmatik Eigenschaften, die Äußerungsabsichten beschreiben. Die drei Ebenen können nicht losgelöst voneinander betrachtet werden. Wissen über syntaktische Regeln ermöglicht erst die semantische Interpretation von Phrasen und Sätzen. Die Erfassung der Bedeutung eines Satzes ist wiederum die Voraussetzung zur Interpretation der Äußerungsabsicht [Oll72].

Aus Sicht der Computerlinguistik bedeutet die Unterscheidung zwischen Syntax, Semantik und Pragmatik nicht nur verschiedentliche Betrachtungsebenen, sondern auch zunehmende Komplexität. Ein Großteil der Sprachanalysen untersuchen ausschließlich syntaktische Eigenschaften. Die Semantik von natürlichsprachlichen Artefakten wird seltener untersucht und die Pragmatik wird nur in sehr eingeschränktem Umfang von einigen Speziallösungen betrachtet. Der Grund hierfür ist, dass sich syntaktische Eigenschaften sehr gut formalisieren lassen. In den meisten Fällen können

[11] In der Literatur wird gelegentlich zwischen Computerlinguistik und Sprachverarbeitung unterschieden. Dabei wird nicht zwischen unterschiedlichen Konzepten unterschieden; es werden lediglich andere Schwerpunkte gesetzt, entweder verstärkt auf das linguistische Fundament von Theorien oder auf die rechnergestützte Verarbeitung. Es herrscht aber kein Konsens bei dieser Unterscheidung. Daher wird in dieser Arbeit auf eine Unterscheidung verzichtet; beide Begriffe werden stattdessen synonym verwendet.

[12] Das Wort *Bank* kann beispielsweise je nach Kontext unter anderem die Bedeutung *Sitzgelegenheit*, *Geldinstitut* oder *Bank-Gebäude* einnehmen.

syntaktische Modellierungen direkt aus linguistischen Theorien oder Grammatikregeln abgeleitet werden. Für die Beschreibung der Semantik und Pragmatik besteht kein derartiges historisch gewachsenes Regelwerk, weshalb sich Formalismen erst schrittweise entwickeln. Zumindest für semantische Eigenschaften ist es der Computerlinguistik-Forschungsgemeinde in den letzten Jahrzehnten gelungen, zunehmend allgemeingültige Modellierungen, wie beispielsweise semantische Rollen (siehe Abschnitt 2.3.4.8), zu erstellen.

2.3.2 Paradigmen der Computerlinguistik

Seit den Anfängen der Computerlinguistik wurde eine Vielzahl unterschiedlicher Ansätze entwickelt, die verschiedene sprachliche Phänomene formalisieren und untersuchen. In den nachfolgenden Abschnitten wird eine Auswahl der untersuchten Phänomene und die zugehörigen Ansätze diskutiert. Die verschiedenen Verfahren lassen sich hinsichtlich des verfolgten Ansatzes unterschiedlichen Paradigmen zuordnen[13]:

- **regelbasierte**: Unter den regelbasierten Ansätzen werden Verfahren zusammengefasst, die vorrangig ein händisch erstelltes Regelwerk zur Analyse von Eingabedaten verwenden. Diese Ansätze werden auch als Heuristik-basiert oder Expertensysteme bezeichnet.
- **statistikbasiert**: Statistikbasierte Ansätze sind Verfahren, die vorwiegend verteilungsbasierte Analysen verwenden. Hierzu zählen vor allem Verfahren des überwachten und unüberwachten maschinellen Lernens (siehe Abschnitt 2.2). Diese werden anhand von Beispieldaten trainiert, wodurch Modelle entstehen; die Modelle können anschließend für die Analyse unbekannter Eingabedaten verwendet werden. Statistikbasierte Ansätze werden auch als probabilistisch bezeichnet, da sie meist auf probabilistischen Modellierungen basieren.
- **wissensbasiert**: Als wissenbasierte Ansätze werden Verfahren bezeichnet, die vorrangig Wissen aus Datenbanken oder Modellierungen (wie beispielsweise Ontologien) verwenden, um Eingabedaten zu analysieren (siehe Abschnitt 2.3.15). Der Erkenntnisgewinn gelingt bei diesen Ansätzen, indem Teile der Eingabedaten auf Einträge der Wissensdatenbank abgebildet und anschließend neue Informationen inferiert werden (z. B. anhand der Struktur der Wissensdatenbank). Für die Inferenz werden wiederum regel- oder statiskbasierte Ansätze verwendet.

Die unterschiedlichen Paradigmen haben Vor- und Nachteile; daher sollte je Aufgabe entschieden werden, welches Paradigma geeignet ist. Für die meisten Aufgabenstellungen verwendet ein Großteil der Verfahren im Kontext der Computerlinguistik heutzutage jedoch statistkbasierte Ansätze [JM09b].

Regelbasierte Ansätze können zunächst einfach konstruiert werden. Zudem werden auch keine Datensätze mit Beispieleingaben zur Erstellung der Regelsätze benötigt; Regeln können gegebenenfalls aus bestehenden linguistischen Regelwerken abgeleitet werden. Die erzeugten Ausgaben sind zudem nachvollziehbar. Allerdings werden Domänenexperten zur Erstellung der Regelsätze benötigt. Um

[13] Jurafsky und Martin geben in ihrem Buch *Speech and Language Processing* einen historischen Überblick und gehen dabei auch auf die unterschiedlichen Paradigmen ein, die sich im Laufe der Zeit entwickelt haben [JM09b].

gute Analyseergebnisse zu erhalten, muss zudem meist ein umfangreicher und komplexer Regelsatz verwendet werden. Dadurch sind regelbasierte aufwendig zu warten und nur schwer zu erweitern. Zudem erreichen die erzeugten Ausgaben im Allgemeinen nicht die Qualität von Ansätzen, die den beiden anderen Paradigmen zugeordnet werden.

Statistikbasierte Ansätze sind normalerweise algorithmisch einfacher umzusetzen; es wird kein Regelwerk oder ähnliches benötigt. Dadurch können auch semantisch komplexe Analyseaufgaben gelöst werden, die sich gegebenenfalls nicht durch Regeln darstellen lassen. So können auch nicht-lineare Zusammenhänge gefunden und modelliert werden. Zudem erzielen statistikbasierte Ansätze im Allgemeinen die qualitativ besten Ergebnisse. Allerdings werden Beispieldaten im ausreichenden Umfang benötigt, deren Beschaffung gegebenenfalls aufwendig sein kann. Überwachte Lernverfahren benötigen zudem Musterlösungen, die zunächst (zumeist händisch) erstellt werden müssen. Außerdem besteht bei Lernverfahren die Gefahr der Überanpassung der Modelle auf die Trainingsdaten. Zusätzlich benötigen statistikbasierte Ansätze meistens viele Ressourcen (Speicher und Rechenzeit). Zudem kann normalerweise nicht nachvollzogen werden, wie Analyseergebnisse erzeugt wurden.

Zur Erstellung von wissensbasierten Ansätze werden keine Beispieldaten benötigt. Stattdessen wird bereits vorhandenes (und modelliertes) Wissen verwendet. Dadurch sind wissensbasierten Ansätze häufig einfach zu erstellen; die Komplexität liegt stattdessen in der Erstellung der Wissensmodellierung. Außerdem kann die Qualität der Verfahren automatisch besser werden, wenn die Wissensmodellierung erweitert wird. Andererseits können wissensbasierte Ansätze nur Informationen verarbeiten, die in der Wissensmodellierung hinterlegt sind. Bei der Erstellung ist zudem Wissen über die Struktur und den Inhalt der Wissensmodellierung notwendig. Außerdem verwenden auch wissensbasierte Ansätze häufig viele Ressourcen. Die Qualität der erzeugten Ergebnisse ist zudem qualitativ meist schlechter als die von statistikbasierten Ansätzen.

Dementsprechend sind regelbasierte Ansätze dafür geeignet, schnell eine erste Lösung für ein neues Problem zu erstellen. Sie eignen sich auch, wenn wenige einfache Regeln genügen, um das Problem zu lösen, oder wenn bereits ein Regelsatz existiert. Statistikbasierte Ansätze liefern jedoch fast immer die qualitativ besten Lösungen, können aber nur verwendet werden, wenn Datensätze mit ausreichendem Umfang zur Verfügung stehen. Zudem muss auch das Problem der potenziellen Überanpassung beachtet werden. Bei semantisch komplexen Aufgaben und gleichzeitig zu geringem Datenumfang, besteht die Gefahr, dass die Modelle die vorhandenen Beispiele auswendig lernen. Wissensbasierte Ansätze stellen eine gute Alternative dar, wenn zwar nur wenige Beispieldaten, dafür aber geeignete Wissensmodellierungen vorhanden sind.

2.3.3 Korpora

Ein *Korpus* (Plural *Korpora*) ist eine (zumeist systematisch angelegte) Sammlung natürlichsprachlicher Artefakte [BCR98]. Die in den Sammlungen enthaltenen Artefakte sind entweder in Dokumenten enthaltene Texte oder in Audiodateien gespeicherte gesprochene Äußerungen. Im ersten Fall spricht man von einem Textkorpus und im zweiten entsprechend von einem Sprachkorpus. Sprachkorpora enthalten neben Audiodateien zumeist zusätzlich zugehörige Transkriptionen. Unter

einer Transkription versteht man die verschriftlichte Form einer gesprochenen Äußerung[14]. Korpora bilden das Rückgrat der modernen Computerlinguistik. Sie dienen der Auffindung und Untersuchung neuer sprachlicher Phänomene; davon ausgehend können Analysetechniken für diese Phänomene abgeleitet werden. Viel wichtiger ist jedoch, dass Korpora es ermöglichen, natürliche Sprache empirisch zu untersuchen. Die Untersuchungsergebnisse bilden wiederum den Ausgangspunkt für die Entwicklung von verteilungsbasierten bzw. probabilistischen Verfahren zur Analyse sprachlicher Phänomene; Korpora werden wiederum als Datengrundlage für derartige Verfahren genutzt.

Das Korpus einer Sprache wird eigentlich durch die Gesamtheit aller natürlichsprachlichen Artefakte der Sprache gebildet. Da eine vollständige Sammlung im Allgemeinen nicht praktikabel ist, werden Korpora stattdessen durch eine repräsentative Auswahl von Artefakten erstellt. Ob eine Auswahl repräsentativ ist, hängt vor allem von der späteren Verwendung ab. Soll beispielsweise die Sprache eines Autors untersucht werden, ist ein Korpus dann repräsentativ, wenn er alle Werke des Autors enthält. Ähnliches gilt für die Untersuchung bestimmter zeitlicher Epochen oder anderweitig eingeschränkter Domänen. Für die Computerlinguistik sind sogenannte Auswahlkorpora von besonderer Bedeutung. Um diese zu bilden, wird ein Querschnitt durch die Sprache gebildet; das bedeutet, es werden Artefakte unterschiedlicher Herkunft, Genres und Autoren gesammelt. Einige Auswahlkorpora enthalten außerdem sowohl Textdokumente als auch Audiodateien (bzw. Transkriptionen).

Außerdem unterscheidet man zwischen unbearbeiteten und bearbeiteten Korpora. Unbearbeitete Korpora enthalten nur die reinen Texte bzw. Audioaufnahmen (und gegebenenfalls Transkriptionen). In bearbeiteten Korpora sind die Artefakte mit zusätzlichen Informationen angereichert. Üblicherweise werden Etiketten verwendet, um linguistische Charakteristika sprachlicher Einheiten auszuweisen. Beispielsweise enthalten Korpora Wortartetiketten, die an einzelne Wörter angebracht werden oder Syntax-Bäume, die für ganze Sätze gültig sind (siehe Abschnitte 2.3.4.3 und 2.3.4.6).

Bearbeitete Korpora bilden die Grundlage moderner probabilistischer Verfahren der Computerlinguistik. Dabei werden die Etiketten als Musterlösung für Klassifikationen mittels überwachter Lernverfahren verwendet (siehe Abschnitt 2.2.1). Das erste weit verbreitete bearbeitete Auswahlkorpus war das 1961 veröffentlichte *Brown Corpus*, welches auch heute noch häufig Anwendung findet. Ein weiteres oft genutztes Korpus (insbesondere im Zusammenhang mit Wortart-Etikettierung und syntaktischer Zerteilung) ist das *WSJ Corpus*, das aus Artikeln des *Wallstreet Journal* besteht. Die *Penn Treebank* ist eine Zusammenstellung bearbeiteter Auswahlkorpora mit einheitlichem Auszeichnungsformat (das heißt Etiketten) und beinhaltet unter anderem auch das *WSJ Corpus* und das *Brown Corpus*. Eine Übersicht über die meisten verfügbaren Korpora bietet das *Linguistic Data Consortium*, welches Sprachressourcen verwaltet, katalogisiert und lizenziert[15]. In den letzten Jahren rücken auch unbearbeitete Korpora stärker in den Fokus. Diese können unter anderem dafür verwendet werden, generische Sprachmodelle zu trainieren (siehe Abschnitt 2.3.17). Der Vorteil unbearbeiteter Korpora besteht darin, dass sehr einfach große Sammlungen angelegt werden können, da keine manuelle Nachverarbeitung nötig ist (wie beispielsweise das Anbringen von Etiketten).

[14] Weitere Details dazu, wie Audioaufnahmen verschriftlicht werden, befinden sich in Abschnitt 5.5.2.

[15] *Linguistic Data Consortium*: `https://www.ldc.upenn.edu/`, zuletzt besucht am 24.02.2021.

2.3.4 Computerlinguistfließbänder

Das Ziel der Computerlinguistik ist die explizite linguistische Beschreibung von natürlichsprachlichen Artefakten, wobei vorwiegend Textdokumente analysiert werden. Das bedeutet, Texte sollen hinsichtlich ihrer Syntax, Semantik und Pragmatik untersucht werden (siehe Abschnitt 2.3.1); gewonnene Informationen werden dem Text als Annotationen hinzugefügt. Die linguistische Analyse von Texten erfolgt üblicherweise mithilfe von Computerlinguisikfließbändern (engl. *NLP pipelines*)[16]. Die stufenweise Verarbeitung des Textes in Fließbändern hat sich etabliert, da viele Analysen aufeinander aufbauen. Beispielsweise können Wortarten erst dann zugeordnet werden, wenn bereits analysiert wurde, welche Zeichenketten einzelne Wörter darstellen; Wortarten sind wiederum die Voraussetzung zur Bestimmung der Wortgrundform. Jede Stufe des Fließbands emittiert Annotationen (bzw. Etiketten), welche die jeweilig untersuchte linguistische Eigenschaft beschreiben, beispielsweise eine Etikette zur Beschreibung der Wortart. Für viele Annotationen existieren standardisierte Etikettensätze, sodass es möglich ist, Stufen auszutauschen. Die Analysen der meisten Fließbandstufen sind heutzutage statistikbasiert; das bedeutet, sie verwenden überwachtes, maschinelles Lernen. Die genau Zusammensetzung der Fließbänder ist nicht festgelegt, auch die Reihenfolge kann variieren[17]. In den folgenden Abschnitten werden die üblicherweise verwendeten Stufen in der gängigen Reihenfolge beschrieben. Die Auswahl und Reihenfolge der vorgestellten Stufen orientiert sich am Werkzeug *CoreNLP* der *Natural Language Processing Group* der *Stanford University* [Man+14].

2.3.4.1 Portionierung

Die Portionierung (engl. *Tokenization*) markiert üblicherweise den Beginn der Verarbeitung von textuellen Dokumenten in einem Computerlinguistikfließband [WK92; DMS00; JM09b]. Die Aufgabe besteht darin, eine Sequenz von Schriftzeichen in syntaktische Abschnitte zu untergliedern. Diese Abschnitte werden als sogenannte *Token* bezeichnet. Sie bilden die kleinste Einheit für die syntaktischen (und gegebenenfalls lexikalischen) Analysen in den nachfolgenden Fließbandstufen. Normalerweise entsprechen die Token den Wörtern und Interpunktionen im Textdokument. Abbildung 2.3 zeigt die Portionierung des Satzes „The quick brown fox jumps over the lazy dog.“.

Die meisten Ansätze zur Portionierung von Textdokumenten sind regelbasiert[18]. Unter anderem werden reguläre Ausdrücke verwendet; der Analyseprozess wird häufig als endlicher Zustandsautomat

[16] Die besten Ergebnisse für einzelne Sprachanalyseaufgaben werden heutzutage allerdings nicht von Computerlinguistikfließbändern erzielt. Stattdessen werden verstärkt tiefe neuronale Netzen (siehe Abschnitt 2.2.2) und vortrainierte Sprachmodelle eingesetzt (siehe Abschnitt 2.3.17). Da die Verwendung (und Erzeugung) dieser Modelle jedoch ressourcenintensiv ist, werden als Grundlage für weiterführende Analysen weiterhin vorwiegend Fließbandarchitekturen verwendet.

[17] Allerdings bestehen häufig Abhängigkeiten, weshalb einzelne Stufen zumindest relativ zueinander angeordnet werden müssen.

[18] Da die Bildung von Token in den meisten Sprachen einem grammatikalischen Regelwerk folgt, kann auch die Analyse der Token regelbasiert erfolgen. Dies gilt (bis auf wenige Ausnahmen) insbesondere für die englische Sprache. Für Sprachen, bei denen die Identifikation der Token schwieriger ist (z. B. weil sie von der Semantik abhängig ist), wird überwachtes maschinelles Lernen verwendet (siehe Abschnitt 2.2.1) [Che+17].

Abbildung 2.3: Portionierung des Satzes „The quick brown fox jumps over the lazy dog.“: Jeder Block ist ein Token.

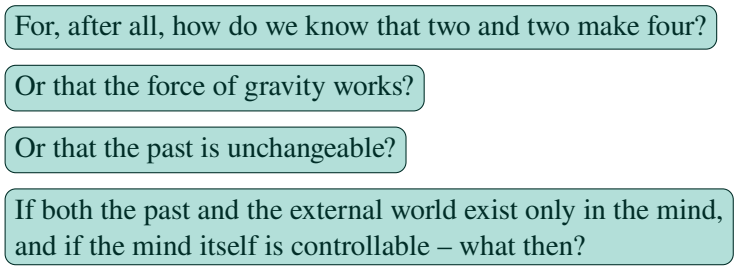


Abbildung 2.4: Satztrennung für einen Ausschnitt aus dem Buch *Nineteen Eighty-Four* von George Orwell [Orw49]: Zur Wahrung der Übersichtlichkeit wurde auf die Darstellung der Token innerhalb der Sätze verzichtet.

implementiert[19]. Zusätzlich verwenden die meisten Ansätze Heuristiken zur Behandlung von Ausnahmen, beispielsweise, um zu erkennen, ob es sich bei einem einfachen Anführungszeichen um einen Wortbestandteil handelt oder nicht[20] oder ob ein Punkt als Satz-beendendes Satzzeichen fungiert oder nicht (z. B. Dezimalstellen bei Dezimalzahlen im Englischen).

2.3.4.2 Satztrennung

Die zweite Stufe bildet üblicherweise die Satztrennung (engl. *sentence splitting*, auch bekannt als *sentence boundary disambiguation*, *sentence breaking*, *sentence boundary detection* oder *sentence segmentation*) [DMS00]. Die Aufgabe besteht darin, die zuvor erkannten Token zur nächstgrößeren syntaktischen Einheit zusammenzufassen; normalerweise ist diese Einheit ein *Satz*. Auch bei der Satztrennung verfahren die meisten Ansätze regelbasiert. Bei entsprechender Portionierung kann die Satztrennung direkt abgeleitet werden: wann immer ein Satz-beendendes Satzzeichen auftritt, wie z. B. Punkt (.), Ausrufezeichen (*!*) oder Fragezeichen (*?*), wird ein Satz abgeschlossen. Zusätzlich verwenden die meisten Ansätze Ausnahmeregeln, beispielsweise falls Satzzeichen gruppiert oder als Teil von wörtlichen Reden auftreten. Abbildung 2.4 zeigt die Satztrennung für einen Ausschnitt aus dem Buch *Nineteen Eighty-Four* von George Orwell[Orw49].

Neben regelbasierten Ansätzen existieren (auch für das Englische) statistikbasierte Methoden (überwachte Lernverfahren, siehe Abschnitt 2.2.1) [KS06]. Diese ermitteln anhand eines zuvor gelernten Modells die wahrscheinlichsten Satztrennungen und sind regelbasierten Verfahren insbesondere dann überlegen, wenn zuvor keine Portionierung des Textes erfolgt ist. Je nach Anwendungsfall können Dokumente weiter unterteilt werden, beispielsweise in Paragraphen oder Kapitel. Gerade bei langen Dokumenten ist dies hilfreich, falls beispielsweise später Themen-Etiketten je Paragraph vergeben werden sollen (siehe Abschnitt 2.3.9).

[19] Beispielsweise ist der häufig verwendete *PTBTokenizer* der *Natural Language Processing Group* der *Stanford University* als endlicher Zustandsautomat implementiert: `https://nlp.stanford.edu/software/tokenizer.shtml`, zuletzt besucht am 24.02.2021.

[20] Im Englischen kann zum Beispiel die Zeichenkette *Peter's* entweder als ein Token mit Genitivendung oder als zwei Token, bestehend aus dem Eigennamen *Peter* und der Kontraktion *'s* des Wortes *is* interpretiert werden.

Abbildung 2.5: Wortartetiketten für den Satz „The quick brown fox jumps over the lazy dog.“: Die Bedeutung der jeweiligen Etiketten kann Tabelle 2.1 entnommen werden.

Tabelle 2.1: Auszug der in der *Penn Treebank* definierten Wortartetiketten [TMS03]: Die vollständige Liste befindet sich im Anhang in Abschnitt A.1.

Etikett	Bedeutung (englisch)	Bedeutung (deutsch)
DT	Determiner	Determinativ (Artikel)
IN	Preposition or subordinating conj.	Präposition oder unterordnende Konj.
JJ	Adjective	Adjektiv, Positiv (Grundform)
NN	Noun, singular or mass	Nomen, Singular oder Kontinuativum
VBZ	Verb, 3rd person singular present	Verb, 3. Person Singular Präsens
.	Sentence-final punctuation	Satz-beendendes Satzzeichen

2.3.4.3 Wortartenerkennung

Nachdem innerhalb der ersten beiden Stufen die Grundeinheiten (Token, Sätze, Paragraphen etc.) bestimmt wurden, folgt die syntaktische (und lexikalische) Analyse dieser. Sukzessive werden Informationen generiert; die Analyseergebnisse werden für gewöhnlich als Etiketten (engl. *tag* oder *label*) an die jeweilige Einheit angebracht bzw. annotiert.

Als erste solche Etikettierungsaufgabe führen die meisten Computerlinguistikfließbänder eine Wortarterkennung (engl. *part-of-speech tagging*, kurz *PoS tagging*) durch [GR71; Cut+92; JM09b][21]. Das Ziel dieser Analyse ist es, je Token ein Etikett zu emittieren, welches die jeweilige Wortart des Wortes (bzw. Tokens) angibt. Welche Etiketten zur Verfügung stehen, ist in sogenannten Etikettensätzen (engl. *tag sets*) festgelegt. Zwar gibt es für die Wortarterkennung keinen standardisierten Etikettensatz, der für die *Penn Treebank* erstellte Satz (siehe Abschnitt 2.3.3), gilt aber als De-Facto-Standard für die englische Sprache und wird von nahezu allen Ansätzen verwendet [TMS03].

In Abbildung 2.5 sind die Wortartetiketten für den bereits in Abschnitt 2.3.4.1 eingeführten Beispielsatz dargestellt. Tabelle 2.1 listet die im Beispiel verwendeten Etiketten der *Penn Treebank* samt einer kurzen Beschreibung der jeweiligen Bedeutung. Wie aus dem Beispiel ersichtlich wird, werden nicht nur die Wortarten, sondern auch die morphologische Form sowie gegebenenfalls die grammatikalische Funktion bestimmt.

Verfahren zur Wortarterkennung lassen sich grob in drei Kategorien unterteilen, je nachdem welche Kriterien die jeweilige Analyse vorwiegend zugrunde legt:

[21] Alternativ wird als erster Schritt eine syntaktische Analyse durchgeführt (siehe Abschnitt 2.3.4.6). Wortartetiketten entstehen dann als Nebenprodukt der syntaktischen Zerteilung des Satzes.

- *Kriterien der Form*: Es erfolgt eine lexikalische Analyse. Die Wortart wird entweder direkt anhand der Oberflächenform oder mithilfe von Heuristiken, die formbildende Wortteile (Wortstamm, Präfixe oder Suffixe) extrahieren, bestimmt. In beiden Fällen erfolgt anschließend ein Abgleich mit einem Lexikon.

- *Kriterien der Stellung*: Anhand der Wortumgebung wird die jeweilige Wortart abgeleitet. Gegebenenfalls können Regeln mithilfe von Schablonen erlernt werden[22].

- *Kriterien der Auftrittswahrscheinlichkeit*: Wortarten werden mithilfe zuvor bestimmter Auftrittswahrscheinlichkeiten einer Einheit oder einer Sequenz von Einheiten bestimmt. Die Wahrscheinlichkeiten entstammen Modellen überwachter Lernverfahren (siehe Abschnitt 2.2.1). Die Modelle werden anhand annotierter Korpora erlernt (siehe Abschnitt 2.3.3).

Verfahren, die sich den ersten beiden Kategorien zuordnen lassen, sind vorwiegend regelbasiert; Verfahren, die sich der letzten Kategorie zuordnen lassen, sind hingegen statistikbasiert. Heutzutage verwenden nahezu alle Ansätze überwachte Lernverfahren, sind also statistikbasiert. Allerdings legen die Modelle fast aller Lernverfahren ebenso die Wortform und die Stellung der Einheiten für die Klassifikation zugrunde. Die besten Verfahren erreichen heutzutage Genauigkeiten (siehe Abschnitt 2.4.4) von knapp 98%[23]. Allerdings ist eine hohe Genauigkeit bei der Bestimmung der Wortarten auch erforderlich, schließlich bildet diese die Grundlage aller weiteren Analysen; daher müssen Fehlerfortpflanzungen vermieden werden. Die hohen Genauigkeitswerte aktueller Verfahren relativieren sich zudem, wenn man betrachtet, wie diese entstehen. Nahezu alle Verfahren werden auf dem Trainingsabschnitt des *WSJ Corpus* trainiert und auf dem Testabschnitt evaluiert (siehe Abschnitt 2.3.3). Das bedeutet, das Evaluationskorpus enthält zwar unbekannte Texte, diese entstammen aber derselben Domäne. Die Genauigkeit der Verfahren kann daher bei Verwendung in anderen Domänen deutlich geringer ausfallen. Außerdem sind selbst Genauigkeiten von über 97% für die Wortarterkennung laut Manning nicht ausreichend, da – betrachtetet man die Genauigkeit auf Satzebene – lediglich 55% bis 57% der Sätze keine fehlerhaften Wortartetiketten enthalten [Man11].

2.3.4.4 Bestimmung der Wortgrundform

In der nächsten Stufe üblicher Computerlinguistikfließbänder wird die Grundform der Wörter bestimmt. Wortgrundformen werden gebildet, um nachfolgende Analysen gegebenenfalls generischer durchzuführen oder die Wörter im Dokument effizient zu indizieren. Verfahren zur Bestimmung der Wortgrundform lassen sie grob in zwei Kategorien einteilen. Die Verfahren der ersten Kategorie führen nur eine einfache Wortstammbildung (engl. *stemming*) durch [Por80]; das bedeutet, sie verwenden einfache Heuristiken, die im Wesentlichen regelbasiert Endungen entfernen, um den

[22] Ein Beispiel für einen Ansatz, der anhand eines annotierten Korpus Regeln erlernt, ist der sogenannte *Brill tager* [Bri92].

[23] Siehe hierzu: `https://nlpprogress.com/english/part-of-speech_tagging.html`, zuletzt besucht am 24.02.2021 bzw. `https://aclweb.org/aclwiki/POS_Tagging_(State_of_the_art)`, zuletzt besucht am 24.02.2021.

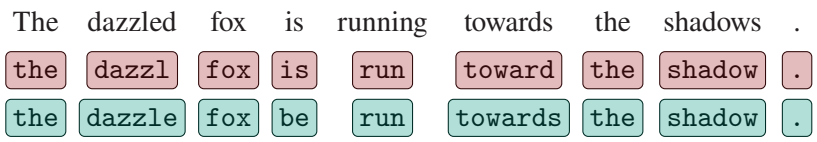


Abbildung 2.6: Wortstammbildung und Lemmatisierung für den Beispielsatz „The dazzled fox is running towards the shadows.“: Die gebildeten Wortstämme sind rot (zweite Zeile) und die Lemmata grün hinterlegt (dritte Zeile).

Wortstamm zu bilden. Verfahren der zweiten Kategorie führen hingegen eine sogenannte Lemmatisierung (engl. *lemmatization*) durch [JM09b]; sie bestimmen also je Wort das zugehörige Lemma[24]. Um das Lemma zu bestimmen, wird für gewöhnlich die Wortart (und gegebenenfalls die Stellung des Wortes im Satz) in die Analyse einbezogen. Daher wird die Bestimmung der Wortart und der Wortgrundform in vielen Fließbändern in einer Stufe gemeinsam oder in aufeinanderfolgenden Stufen durchgeführt [Man+14].

Abbildung 2.6 veranschaulicht die Wortstammbildung und Lemmatisierung für den Beispielsatz „The dazzled fox is running towards the shadows.“. Wie im Beispiel zu sehen, erzeugt die einfache Wortstammbildung teilweise Zeichenketten, die nicht grammatikalisch korrekten Wörtern entsprechen; dies kann gegebenenfalls bei weiterführenden Analysen zu Problemen führen. Darüber hinaus können Generalisierungen entstehen, die semantisch falsch sind (beispielsweise werden die Wörter *university*, *universe* und *universal* alle auf den Wortstamm *univers* zurückgeführt, obwohl die Wörter semantisch nicht gleich oder ähnlich sind).

2.3.4.5 Erkennung von Eigennamen

Die Aufgabe der nächsten Stufe ist die Erkennung von Eigennamen, oder genauer, die Erkennung benannter Entitäten (engl. *named entity recognition*, kurz *NER*) [GS96; NS07][25]. Dazu sollen in Token (oder zusammenhängende Tokensequenzen) enthaltene benannte Entitäten erkannt und eine entsprechende Etikette zugewiesen werden. Auch für die Erkennung von Eigennamen existiert kein standardisierter Etikettensatz. Viele Verfahren verwenden den Satz der Vergleichsaufgabe der *Conference on Computational Natural Language Learning* aus dem Jahr 2003 (*CoNLL shared Task 2003*) [TD03]; dieser umfasst die folgenden Etiketten:

- PERSON (der Name einer Person)
- LOCATION (der Name eines Ortes)
- ORGANIZATION (der Name einer Organisation)
- MISCELLANEOUS (sonstiger Name)

[24] Unter dem Wortlemma versteht man die Oberflächenform eines Wortes, unter der dieses in Lexika gelistet ist. Bei Verben ist dies beispielsweise im Englischen die Infinitivform, bei Nomen die Singularform.

[25] Aufgabenbeschreibung *Named Entity Recognition* der *Message Understanding Conference* des Jahres 1995 (*MUC-6*): `https://cs.nyu.edu/grishman/NEtask20.book_1.html`, zuletzt besucht am 24.02.2021.

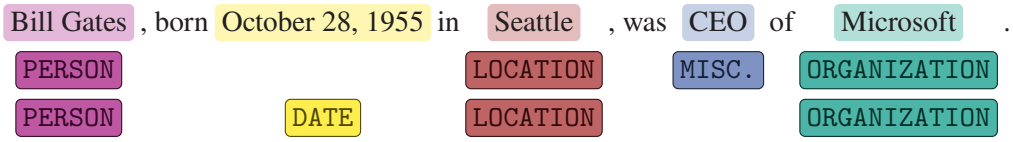


Abbildung 2.7: Eigennamenetiketten für den Beispielsatz „Bill Gates, born October 28, 1955 in Seattle, is was CEO of Microsoft.“ aus zwei Etikettensätzen: Die Etiketten in der zweiten Zeile entsprechen dem Satz der Vergleichsaufgabe der *Conference on Computational Natural Language Learning* aus dem Jahr 2003 (*CoNLL shared Task 2003*) [TD03], die in der dritten Zeile dem Satz der *Message Understanding Conference* aus dem Jahr 1998 (*MUC-7*) [CR98].

Da diese vier Klassen gegebenenfalls unzureichend sind, um benannte Entitäten geeignet zu differenzieren, ist ein zweiter (konkurrierender) Etikettensatz verbreitet[26]. Dieser entstammt der Vergleichsaufgabe der *Message Understanding Conference* aus dem Jahr 1998 (*MUC-7*) [CR98] und umfasst die folgenden sieben Etiketten:

- PERSON (der Name einer Person)
- LOCATION (der Name eines Ortes)
- ORGANIZATION (der Name einer Organisation)
- DATE (vollständiger oder teilweiser Datumsausdruck)
- TIME (vollständiger oder teilweiser Zeitausdruck)
- MONEY (monetärer Ausdruck)
- PERCENT (eine Prozentzahl)

Abbildung 2.7 zeigt einen Beispielsatz, der mit Etiketten aus beiden Etikettensätzen annotiert ist. Wie im Beispiel zu sehen, können unter Verwendung des zweiten Etikettensatzes potenziell mehr benannte Entitäten annotiert werden. Andererseits fehlt diesem Satz ein Etikett für sonstige Namen, wodurch Eigennamen, die dem Modell unbekannt sind, kein Etikett erhalten; bei Verwendung des ersten Etikettensatzes kann diesen Entitäten das Etikett *MISCELLANEOUS* zugewiesen werden.

Verfahren zur Erkennung von Eigennamen bzw. benannten Entitäten waren ursprünglich regelbasiert und verwendeten Heuristiken die auf einer Kombination aus Wortarten, regulären Ausdrücken und Signalwörtern basierten. Heutzutage werden ausschließlich überwachte Lernverfahren eingesetzt, vor allem neuronale Netze. Mit diesen können aktuell Genauigkeiten von bis zu 94,3% erreicht werden (*CoNLL shared Task 2003*)[27].

2.3.4.6 Syntaktische Zerteilung

Ein wichtiger Bestandteil nahezu aller Computerlinguistfließbänder ist die Stufe zur syntaktische Zerteilung (engl. *(syntactic) parsing*, auch bekannt als *syntax analysis* oder *syntactic analysis*). Es

[26] Viele Verfahren ermöglichen daher die Auswahl des zu verwendenden Etikettensatzes.

[27] Siehe hierzu: `https://nlpprogress.com/english/named_entity_recognition.html`, zuletzt besucht am 24.02.2021.

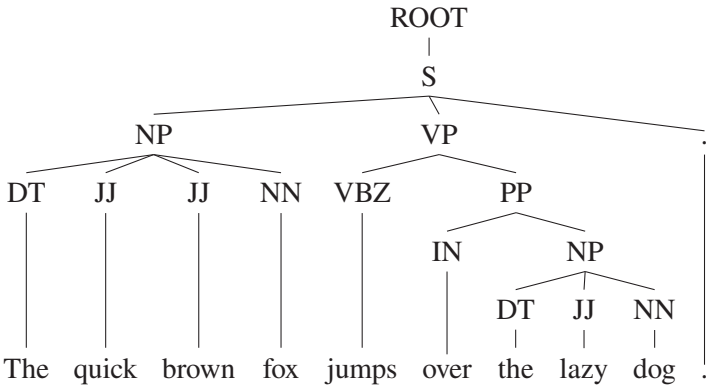


Abbildung 2.8: Syntaxbaum für den Satz „The quick brown fox jumps over the lazy dog.“: Die Bedeutungen der Etiketten, die syntaktische Kategorien beschreiben, können Tabelle 2.2 entnommen werden.

gibt zwei Ausprägungen syntaktischer Zerteiler, die sich hinsichtlich der Art (bzw. des erzeugten Ergebnisses) der Zerteilung unterscheiden. Konstituenzzerteiler (engl. *constituency parser*) erzeugen Syntaxbäume (engl. *parse trees*) [Per76; JM09b], wohingegen Dependenzzerteiler (engl. *dependency parser*) Abhängigkeitsgraphen generieren (engl. *dependency graphs*) [JM09b]. Syntaxbäume und Abhängigkeitsgraphen bieten unterschiedliche Sichtweisen auf die Satzsyntax. Während erstere den Fokus auf die (hierarchische) syntaktische Struktur des Satzes legen, steht bei zweiteren die Funktion der Satzglieder und der Zusammenhang dieser im Vordergrund (weshalb Abhängigkeitsgraphen neben der Syntax auch Teile der Semantik des Satzes erfassen können). Da beide Darstellungsformen Vor- und Nachteile mit sich bringen, bieten die meisten Computerlinguistikfließbänder beide als separate Stufen an.

Erzeugung von Syntaxbäumen

Konstituenzzerteiler erzeugen je Satz eine Baumstruktur, die dessen Syntax darstellt. Die Token bilden die Blätter des Baumes, intermediäre Knoten überspannen Phrasen und die Wurzel symbolisiert den gesamten Satz. Das bedeutet, ein Syntaxbaum stellt nur dann eine gültige Zerteilung des Satzes dar, wenn alle im Satz enthaltenen Token die Blätter des Baumes (in der korrekten Reihenfolge) bilden und die Wurzel dem Startsymbol entspricht. Die Erzeugung der Bäume erfolgt grundsätzlich durch die Anwendung einer vorher definierten formalen Grammatik [McE19]. Die Form der Grammatik ist dabei prinzipiell beliebig. Eine einheitliche Wahl der Symbole ist jedoch vorteilhaft. Als Startsymbol fungiert meist das Etikett *ROOT*. Die Verwendung der Token als Terminalsymbole ergibt sich aus der Zielsetzung. Nichtterminalsymbole der Grammatik sind zum einen die Wortartetiketten (siehe Abschnitt 2.3.4.3) und zum anderen Etiketten, die syntaktische Kategorien bzw. Phrasentypen repräsentieren. Für diese zweite Art von Nichtterminalsymbolen gelten wiederum die für die *Penn Treebank* definierten als De-Facto-Standard [TMS03]. Abbildung 2.8 zeigt einen Syntaxbaum für den bereits in Abschnitt 2.3.4.1 eingeführten Beispielsatz; Tabelle 2.2 listet alle im Beispiel verwendeten Etiketten für syntaktische Kategorien.

Verfahren zur Erzeugung von Syntaxbäumen sind prinzipiell generativ. Mithilfe der Ableitungsregeln werden die Bäume sukzessive entweder von unten nach oben (engl. *bottom-up*) oder von oben nach unten (engl. *top-down*) aufgebaut. Da beide Verfahren Nachteile mit sich bringen, werden stattdessen

Tabelle 2.2: Auszug der in der *Penn Treebank* definierten Etiketten für syntaktische Kategorien [TMS03]: Die vollständige Liste befindet sich im Anhang in Abschnitt A.3.

Etikett	Bedeutung (englisch)	Bedeutung (deutsch)
NP	Noun phrase	Nominalphrase
PP	Prepositional phrase	Präpositionalphrase
S	Simple declarative clause	Einfacher Deklarativsatz
VP	Verb phrase	Verbalphrase

häufig Mischformen verwendet, wie beispielsweise *Left-Corner-Parsing* [RL70]. Grammatiken, die ausreichend sind, um einen Großteil der Sätze einer natürliche Sprache wie Englisch zerteilen zu können, bestehen aus umfangreichen Regelsätzen. Häufig können verschiedene Ableitungen (bzw. Ableitungssequenzen) angewendet werden, um Syntaxbäume zu erzeugen. Das bedeutet, dass Zerteilungen in der Praxis meist nicht eindeutig sind. Zwar können die möglichen Zerteilungen effizient bestimmt werden[28], allerdings muss unter diesen die wahrscheinlichste gewählt werden, um einen eindeutigen Syntaxbaum zu erhalten. Daher sind auch Konstituenzzerteiler heutzutage probabilistisch; den derzeitigen Stand der Technik bilden Ansätze, die auf auf neuronalen Netzen basieren. Die besten Verfahren erreichen aktuell für das F_1-Maß Werte von über 96%[29].

Erzeugung von Abhängigkeitsgraphen

Abhängigkeitsgraphen bieten eine Sichtweise auf die Syntax eines Satzes, die näher an der Semantik ist. Die Graphen beschreiben die grammatikalische Funktion der Phrasen. Das Ziel ist, für jedes Prädikat einen gerichteten Graphen aufzubauen; besitzt ein Satz mehrere Prädikate, werden diese durch Kanten verbunden. Ausgehend von den Prädikaten werden abhängige Satzglieder, wie beispielsweise Subjekte und indirekte Objekte, über typisierte Kanten zugeordnet. Diesen untergeordnet werden wiederum Modifikatoren (Adjektive, Artikel etc.). Während zu Beginn jeder Ansatz andere Kantentypen (Etiketten) verwendete[ST93; Lin03], wurde über die Jahre der Etikettensatz der *Natural Language Processing Group* der *Stanford University* zum De-Facto-Standard [dM08]. Inzwischen konnte ein einheitlicher Standard gebildet werden: Das Projekt *Universal Dependencies* (kurz *UD*) hat einen allgemeingültigen Etikettensatz für viele Sprachen etabliert [Niv+16][30]. Abbildung 2.9 zeigt einen Abhängigkeitsgraphen, der die Kantentypen des *UD*-Projekts verwendet. Die für das Beispiel relevanten Kantentypen (bzw. Etiketten) sind zusammen mit einer Kurzbeschreibung der Bedeutung in Tabelle 2.3 aufgeführt.

Verfahren zur Erzeugung von Abhängigkeitsgraphen verwendeten zunächst vorwiegend spezialisierte händisch erstellte Grammatiken [ST93]. Auch bei der Erzeugung von Abhängigkeitsgraphen verwenden moderne Verfahren nahezu ausschließlich neuronale Netze und vortrainierte Sprachmodelle[31].

[28] Hierzu werden in der Regel sogenannte *Chart-Parser* verwendet. Viele Ansätze dieser Art basieren auf dem Algorithmus von Cocke, Younger und Kasami [Kas66; You67; Koz97]

[29] Siehe hierzu: `https://nlpprogress.com/english/constituency_parsing.html`, zuletzt besucht am 24.02.2021.

[30] Universal Dependencies: `https://universaldependencies.org/`, zuletzt besucht am 24.02.2021.

[31] Siehe hierzu: `https://nlpprogress.com/english/dependency_parsing.html`, zuletzt besucht am 24.02.2021.

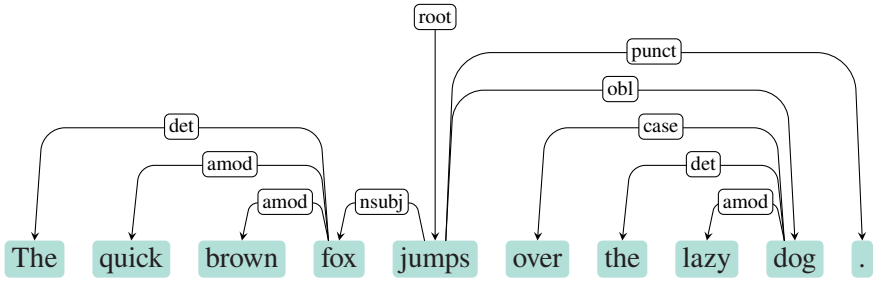


Abbildung 2.9: Ein Abhängigkeitsgraph für den Satz „The quick brown fox jumps over the lazy dog.“: Die Bedeutungen der Etiketten können Tabelle 2.3 entnommen werden.

Tabelle 2.3: Auszug der Kantentypen zur Erzeugung von Abhängigkeitsgraphen, wie sie im Projekt *Universal Dependencies* definiert sind [Niv+16].

Etikett	Bedeutung (englisch)	Bedeutung (deutsch)
root	Root	Wurzel
amod	Adjectival modifier	Adjektivischer Modifikator
case	Case marking	Kasus-markierendes Element
det	Determiner	Determinativ (Artikel)
nsubj	Nominal subject	Nominales Subject
obl	Oblique nominal	Abhängiges Nomen (z. B. Präpositionalobjekt)
punct	Punctuation	Interpunktion

2.3.4.7 Korreferenzanalyse

Diese Fließbandstufe analysiert und annotiert natürlichsprachliche Referenzen im zugrundeliegenden Dokument. In der Linguistik unterscheidet man verschiedene Formen von Referenzen [Cry11]. Die Unterschiede werden anhand der folgenden Beispiele verdeutlicht:

1. *The fox jumps and it runs.*
2. *The fox jumps and runs.*
3. *The fox looks over there.*

Zunächst unterscheidet man *explizite* und *implizite* Referenzen. Erstere werden mithilfe eines explizit referenzierenden Ausdrucks gebildet (*it* in Beispiel 1), wohingegen zweitere nur implizit zwischen zwei Satzbestandteilen existieren (z. B. aufgrund von elliptischen Auslassungen, wie in Beispiel 2). Zudem unterscheidet man *endophorische* und *exophorische* Referenzen. Bei ersteren handelt es sich um Referenzen, die einen konkreten Bezugspunkt (Referenten) innerhalb des Textes haben (*the fox* in den ersten beiden Beispielen). Referenzierende Ausdrücke exophorischer Referenzen beziehen sich hingegen auf einen Bezugspunkt, der außerhalb des textuellen Diskurses liegt (*there* in Beispiel 3). Daher fokussiert sich die Computerlinguistik-Forschungsgemeinde nahezu ausschließlich auf die Analyse endophorischer Referenzen[32].

[32] Es existieren zwar Verfahren zur Erkennung exophorischer Referenzen. Allein mit den Mitteln der Computerlinguistik kann allerdings kein Bezugspunkt ermittelt werden.

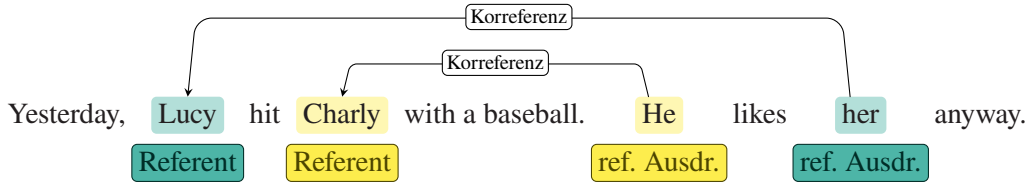


Abbildung 2.10: Zwei Beispielsätze, die zwei Korreferenzen mit unterschiedlichen Referenten und referenzierenden Ausdrücken beinhalten.

Implizite endophorische Referenzen bedürfen häufig einer semantischer Analyse des Textes. Daher sind Ansätze dieser Art häufig anwendungsspezifische Speziallösungen. Eine Ausnahme bilden elliptische Auslassungen, wie in Beispiel 2. Der implizite Zusammenhang zwischen *the fox* und *runs* wird durch Dependenzzerteiler mittels einer Kante (*nsubj*) zwischen den beiden Bestandteilen expliziert.

Als allgemeine Problemstellung der Computerlinguistik ist jedoch lediglich die Analyse expliziter endophorischer Referenzen als sogenannte Korreferenzanalyse definiert[Hob78; JM09b]. Gibt es innerhalb eines Textes einen referenzierenden Ausdruck, der sich auf einen ebenfalls im Text befindlichen Referenten bezieht, so bilden diese beiden eine Korreferenz. Im obigen Beispiel 1 referenziert das Personalpronomen *it* den Referenten *the fox*[33].

Ein weiteres Beispiel mit zwei unterschiedlichen Korreferenzbeziehungen ist in Abbildung 2.10 dargestellt. Die referenzierenden Ausdrücke sind jedoch nicht wie in den Beispielen gezeigt auf Personalpronomen beschränkt. Prinzipiell können beliebige Nominalphrasen andere Nominalphrasen referenzieren. Der ehemalige Präsident der Vereinigten Staaten von Amerika George W. Bush könnte beispielsweise durch folgende Nominalphrasen referenziert werden:

- *he*
- *the former president of the US*
- *the seventy-four-year-old*
- *the old Texan*

Die letzten drei können nicht mit syntaktischen Analysen sondern nur mit Weltwissen als Referenzen erkannt werden. Die Referenz *the old Texan* erfordert zudem Wissensinferenz, denn zum einen wurde George W. Bush zwar in Conneticut geboren, ist aber in Texas aufgewaschen, zum anderen muss auch das Adjektiv *old* gedeutet werden. Aufgrund dieser Komplexität beschränken sich die meisten Ansätze auf die Erkennung eindeutiger Korreferenzen; das bedeutet, die referenzierenden Ausdrücke sind entweder Personalpronomen oder weisen eine ähnliche Oberflächenform auf

[33] Genauer gesagt ist der Referent einer Korreferenz ein benanntes Konzept, hier *fox*. Alle textuellen Nennungen sind referenzierende Ausdrücke, im Beispiel also sowohl *it* als auch *the fox*. In der Praxis spielt diese Unterscheidung jedoch keine Rolle. Als Referent wird üblicherweise die erste Nennung des Konzepts in Form einer benannten Entität angesehen.

(beispielsweise *George W. Bush*, *George Bush* und *Bush*)[34]. In der Linguistik wird zudem zwischen *Anaphern* und *Kataphern* unterschieden. Anaphern stellen einen Rückbezug dar, wie in allen zuvor gezeigten Beispielen. Kataphern sind hingegen Vorwärtsreferenzen, wie in „*It* jumped over the dog. *It* was the fox.“. Da Kataphern sehr selten auftreten, fokussieren sich die meisten Ansätze auf die Analyse anaphorischer Referenzen. Dies gilt auch für die Vergleichsaufgaben der *Conference on Computational Natural Language Learning* der Jahre 2011 und 2012 (*CoNLL shared Task 2011* und *CoNLL shared Task 2012*) zur Korreferenzanalyse [Pra+11; Pra+12a].

Da es sich bei der Korreferenzanalyse (nicht zuletzt durch die Bereitstellung einer Vergleichsaufgabe) um eine allgemeingültige und definierte Aufgabe handelt, ist diese inzwischen Bestandteil vieler Computerlinguistikfließbänder. Verfahren zur Analyse von Korreferenzen verwendeten ursprünglich Heuristiken, die auf syntaktischen Mustern basierten. Heutzutage sind nahezu alle Verfahren probabilistisch, wobei vor allem neuronale Netze verwendet werden. Dabei werden für das F_1-Maß inzwischen Werte von über 80% erreicht[35].

2.3.4.8 Erkennung semantischer Rollen

Die letzte (hier beschriebene) Stufe des Computerlinguistikfließbands ist für die Erkennung semantischer Rollen (engl. *semantic role labeling*, kurz *SRL*) zuständig [Gru65; Fil66; Sim73; JM09b]. Dem Namen nach handelt es sich hierbei um eine Analyse der Semantik des Texts. Allerdings wird die Erkennung semantischer Rollen grundsätzlich anhand syntaktischer Eigenschaften durchgeführt; das Ziel ist es, die kleinsten zusammenhängenden semantischen Grundeinheiten zu bilden und diese miteinander in Beziehung zu setzen. Semantische Rollen bilden dementsprechend lediglich die Grundlage weiterer semantischer Analysen. Zur Bildung der Rollen werden syntaktischen Phrasen (siehe Abschnitt 2.3.4.6) semantische Funktionen zugeordnet. Die zugrundeliegende Annahme bei der Vergabe der Rollen ist, dass die Semantik einer Aussage (bzw. eines Satzteils) immer anhand des Prädikats beschrieben werden kann (unabhängig davon, ob es sich um eine Handlungs-, Ereignis-, oder Zustandsbeschreibung etc. handelt). Dementsprechend werden semantische Strukturen gebildet, die immer aus genau einer Verbalphrase bestehen. Der Verbalphrase werden Argumente in Form anderer Phrasen zugeordnet[36]. Das bedeutet, semantische Rollen werden immer gemeinsam pro Aussage vergeben. Es kann jedoch vorkommen, das Phrasen mehrere (gegebenenfalls überlappende) Rollen in unterschiedlichen semantische Strukturen zugeordnet werden.

Als De-Facto-Standard für semantische Rollen gilt der Etikettensatz, der im Zuge der Vergleichsaufgaben der *Conference on Computational Natural Language Learning* der Jahre 2004 und 2005 (*CoNLL shared Task 2004* und *CoNLL shared Task 2005*) entwickelt wurde [CM04; CM05]. Dieser

[34] Begründet werden kann diese Beschränkung durch folgende Abwägung: Diese eindeutigen Korreferenzen machen einen Großteil aller Korreferenzen aus. Können diese richtig erkannt werden, kann also bereits eine hohe Ausbeute erzielt werden. Eine Analyse der komplexen Referenzen (die Weltwissen erfordern) birgt die Gefahr, viele falsch positive Ergebnisse zu erzeugen, wodurch die Präzision sinkt.

[35] Siehe hierzu: `https://nlpprogress.com/english/coreference_resolution.html`, zuletzt besucht am 24.02.2021.

[36] Damit ähnelt die grundsätzliche Herangehensweise der Erzeugung von Abhänigkeitsgraphen, wie in Abschnitt 2.3.4.6 beschrieben. Allerdings liegt der Fokus bei semantischen Rollen auf der Semantik der jeweiligen Phrase, während Abhängigkeitsgraphen eher die grammatikalische Funktion erfassen.

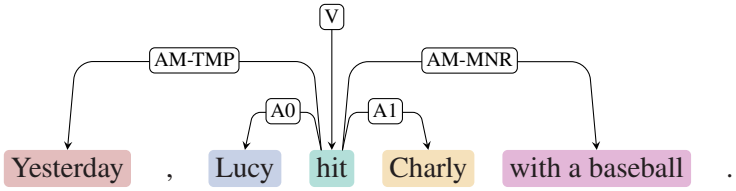


Abbildung 2.11: Semantische Rollen für den Satz „Yesterday, Lucy hit Charly with a baseball.“: Die Bedeutungen der Etiketten können Tabelle 2.4 entnommen werden.

Tabelle 2.4: Auszug der Etiketten für semantische Rollen [CM04; CM05]: Die vollständige Liste befindet sich im Anhang in Abschnitt A.4.

Etikett	Bedeutung (englisch)	Bedeutung (deutsch)
V	verb	Verb (Prädikat)
A0	agent	Handelnder (Agent)
A1	1st patient or theme	1. Patiens oder Thema
AM-MNR	manner	Art und Weise
AM-TMP	temporal	Zeitausdruck

sieht vor, dass das Prädikat jeder semantischen Struktur ein *V*-Rollen-Etikett erhält, dem Handelnden (engl. *agent*) die Rolle *A0* zugeordnet wird und das behandelte Objekt oder Thema (engl. *patient* bzw. *theme*) die Rolle *A1* erhält. Weitere Satzobjekte – sofern vorhanden – erhalten die Rollen *A2* bis *A5*[37]. Diese Rollen werden im Folgenden unter dem Begriff *A*-Rollen* zusammengefasst. Zusätzlich können Rollen für modifizierende Argumente (*AM**-Rollen) und Referenzen auf andere Rollen (*R**-Rollen) vergeben werden. Bis auf die *V*-Rolle sind alle Rollen zur Bildung einer gültigen semantischen Struktur optional.

In Abbildung 2.11 ist eine semantische Struktur zu einem Beispielsatz dargestellt. Die semantischen Rollen sind als Etiketten an gerichteten Kanten dargestellt, die jeweils auf die entsprechende Phrase zeigen. Die im Beispiel verwendeten Etiketten des Etikettensatzes und ein Kurzbeschreibung der jeweiligen Bedeutung sind in Tabelle 2.4 aufgeführt. Erste Verfahren zur Erkennung semantischer Rollen orientierten sich stark an der Satzsyntax, entweder indem sie mithilfe von Heuristiken aus syntaktischen Strukturen semantische Rollen ableiteten oder indem syntaktische Eigenschaften als Eingabe für überwachte Lernverfahren dienten [HTS73; Nas75; Bob+77; GJ02]. Moderne Verfahren verwenden hingegen neuronale Netze und vortrainierte Sprachmodelle und erreichen bei der Annotation von semantischen Rollen für das F_1-Maß Werte von bis zu 87%[38].

2.3.5 Stoppwörter

Unter Stoppwörtern versteht man in der Computerlinguistik solche Wörter, die keinen oder nur einen geringen Beitrag zur Semantik (einer Phrase oder eines Satzes) beitragen. Diese werden

[37] Darüber hinaus gibt es die Rolle *AA* um weitere Objekte zu erfassen (beispielsweise bei langen Aufzählungen).

[38] Siehe hierzu:`https://paperswithcode.com/sota/semantic-role-labeling-on-ontonotes`, zuletzt besucht am 24.02.2021.

Yesterday	,	Lucy	hit	Charly	with	a	baseball	.
B-AM-TMP	O	B-A0	B-V	B-A1	B-AM-MNR	I-AM-MNR	I-AM-MNR	O
S-AM-TMP	O	S-A0	S-V	S-A1	B-AM-MNR	I-AM-MNR	E-AM-MNR	O

Abbildung 2.12: Beispielhafte Verwendung der Formate *IOB* (grün markiert) und *IOBES* (rot markiert) für die Etikettierung semantischer Rollen.

in sogenannten Stoppwortlisten gesammelt und können bei Sprachanalysen dafür verwendet werden, Stoppwörter nicht zu betrachten und den Fokus stattdessen auf relevante Wörter zu legen. Prinzipiell können (je nach Anwendungsfall) beliebige Wörter eine Stoppwortliste bilden. Üblicherweise bestehen diese jedoch aus häufig auftretenden Wörtern, die nur strukturierende Funktionen übernehmen. Dies sind vor allem Konjunktionen, Artikel und Präpositionen, im Englischen also beispielsweise *the*, *is*, *at*, *which*, und *on* [JM09b].

Es gibt verschiedene Stoppwortlisten, die unterschiedliche Charakteristika aufweisen. Sie unterscheiden sich zumeist dahingehend, wie restriktiv sie sind, das heißt wie viele Wörter als irrelevant gelten; beispielsweise beinhalten manche Listen die Verben *be* und *have* und andere nicht. Grundsätzlich muss bei der Verwendung von Standard-Stoppwortlisten beachtet werden, dass je nach Anwendungsgebiet Wörter, die normalerweise keine Semantik tragen, im Kontext der zu lösenden Aufgabe relevant sein können.

2.3.6 Die Formate IOB und IOBES

Die Formate *IOB* und *IOBES* werden in der Computerlinguistik dazu verwendet, Token zu zusammenhängenden Bereichen (Phrasen, Teilsätze oder Sätze) zusammenzufassen und mit einem gemeinsamen Etikett zu versehen. Etiketten zur Auszeichnung sprachlicher Charakteristika werden üblicherweise an Token angebracht, da diese die kleinste zusammenhängende sprachliche Einheit bilden (siehe Abschnitt 2.3.4.3). Mithilfe der beiden Formate können Etiketten für zusammenhängende Bereiche vergeben werden. Technisch gesehen wird weiterhin ein Etikett pro Token vergeben; die Etikettenbereiche entstehen durch die Verwendung spezieller Präfixe, die durch die Formate vorgeben werden. Die Präfixe übernehmen die folgenden Funktionen:

- *B*: markiert den Beginn eines Bereichs
- *I*: zeigt an, dass ein Token innerhalb eines Bereichs liegt
- *O*: zeigt an, dass ein Token zu keinem Bereich gehört
- *E*: markiert das Ende eines Bereichs
- *S*: zeigt an, dass ein Bereich nur aus genau einem Token besteht

Die beiden Formate unterscheiden sich dahingehend, ob das Ende eines Bereichs sowie einelementige Bereiche explizit modelliert werden oder nicht. Abbildung 2.12 zeigt die Verwendung der Formate für die Etikettierung semantischer Rollen anhand des bereits aus Abschnitt 2.3.4.8 bekannten Beispiels.

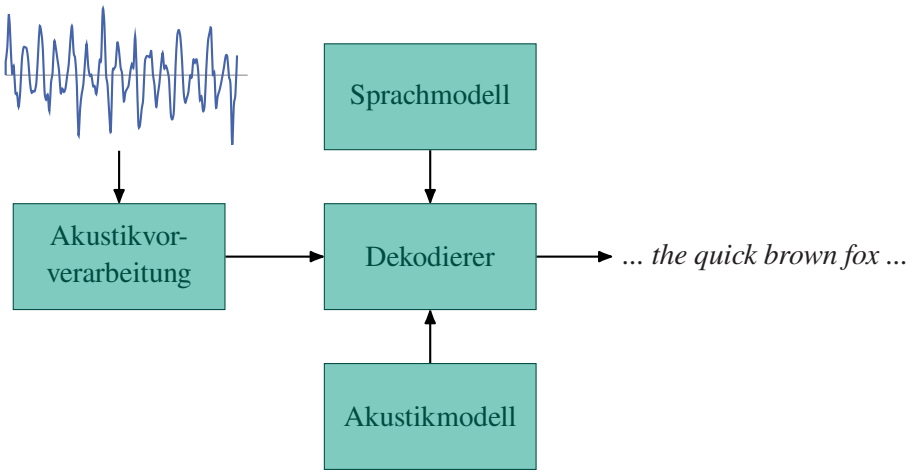


Abbildung 2.13: Grundlegender Aufbau von Systemen zur automatischen Spracherkennung: Die Darstellung entspricht einer verallgemeinerten Variante der von Jurafsky und Martin vorgestellten Architektur für ein ASR-System, das auf dem *Hidden-Markov-Modell* basiert [JM09b].

2.3.7 Automatische Spracherkennung

Systeme zur automatischen Spracherkennung (engl. *automatic speech recognition*, kurz *ASR*) überführen ein Audiosignal in eine textuelle Form, das heißt in eine Sequenz von Wörtern bzw. Token [JM09b]. Diese Aufgabe ist insofern herausfordernd, als jeder Sprecher Wörter anders ausspricht und ein ihm eigenes Vokabular verwendet. Daher unterteilen sich Ansätze zur automatischen Spracherkennung seit jeher in solche, die sprecherspezifisch und solche, die sprecherunabhängig sind. Sprecherspezifische Verfahren werden auf einen Sprecher (bzw. eine Gruppe von Sprechern) angepasst, um entweder die Qualität der Überführung in Textform zu erhöhen oder um Ressourcen zu sparen. Sprecherunabhängige Verfahren können hingegen potenziell Äußerungen beliebiger Sprecher, welche dieselbe Sprache sprechen, verschriftlichen. Darüber hinaus gibt es hybride Ansätze, die einen sprecherunabhängigen Ansatz sukzessive auf einen spezifischen Sprecher anpassen.

Systeme zur automatischen Spracherkennung werden seit Beginn der 1950er Jahre entwickelt. Zur Modellierung der Problemstellung wird seit jeher das Rausch-Kanal-Modell von Shannon zugrunde gelegt [Sha48]. Dabei wird das akustische Signal als verrauschte Variante der textuellen Wortsequenz aufgefasst. Einer Forschungsgruppe der *Bell Laboratories* gelang es 1952 das erstes System zur automatischen Spracherkennung umzusetzen [DBB52]; dieses konnte sprecherspezifisch die Ziffern Null bis Neun erkennen. Moderne Verfahren verwenden statistikbasierte Ansätze, wobei unterschiedliche Lernverfahren zum Einsatz kommen.

Der grundsätzliche Aufbau fast aller Systeme zur automatischen Spracherkennung, wie in Abbildung 2.13 dargestellt, ist indes gleich. Es werden zwei unterschiedliche Modelle benötigt, die vorab anhand von Korpora trainiert werden: Zum einen wird ein Sprachmodell verwendet, welches einerseits das bekannte Vokabular beinhaltet (z. B. durch Verwendung eines Lexikons) und anderseits Häufigkeiten von Wortfolgen (z. B. als N-Gramme) modelliert. Zum anderen wird ein Akustikmodell verwendet, das Abbildungen von Audiosignalen auf Phoneme beinhaltet. Die beiden

Modellierungen werden in einem Dekodierer zusammengeführt, beispielsweise indem eine durch das Akustikmodell erzeugte Phonemsequenz durch das Sprachmodell auf die wahrscheinlichste Wortfolge abgebildet wird. Verfahren unterscheiden sich hauptsächlich in der Funktionsweise des Dekodierers. Ein akustisches Signal wird von einem System zur automatischen Spracherkennung wie folgt verarbeitet: Zunächst wird das akustische Signal vorverarbeitet; das bedeutet, das Signal wird digitalisiert und Eigenschaften, wie Amplitude, Frequenz etc. in eine Darstellung überführt, die vom Akustikmodell verarbeitet werden kann. Im Dekodierer werden die beiden Modelle wie zuvor beschrieben auf dieses Signal angewendet; das Ergebnis ist eine Tokensequenz; wobei die meisten Systeme mehrere Hypothesen mit zugehörigen Konfidenzen erzeugen, da zum Dekodieren üblicherweise probabilistische Ansätze verwendet werden.

Die durch ein System zur automatischen Spracherkennung erzeugte Wortsequenz, kann durch ein Computerlinguistkfließband weiterverarbeitet werden (siehe Abschnitt 2.3.4). Allerdings muss beachtet werden, dass eine verschriftlichte gesprochene Äußerung andere Eigenschaften hat als ein geschriebener Text. Gesprochene Sprache folgt im Allgemeinen weniger strikt grammatikalischen Regeln. Das bedeutet, gesprochene Äußerungen enthalten häufig nur Satzfragmente. Häufig vertauschen Sprecher auch Satzteile in einer Weise, die nicht konform mit der Grammatik der Sprache ist. Außerdem enthält spontane gesprochene Sprache häufig sogenannte Disfluenzen. Als Disfluenzen bezeichnet man Unterbrechungen im Sprechfluss, die beispielweise durch das Füllen von (Denk-)Pausen mit Hesitationswörtern wie *uhm*, *ehm* etc. oder durch die (Selbst-)Korrektur von vorangegangen Äußerungen entstehen (siehe Abschnitt 7.1). Zuletzt werden bei der automatischen Spracherkennung normalerweise keine Satzzeichen erzeugt. Aus diesen Gründen können Sprachanalyseverfahren, die für ihr Vorgehen grammatikalisch korrekte vollständige Sätze annehmen bzw. deren Modelle auf solchen trainiert wurden, unter Umständen nicht verwendet werden bzw. erzeugen diese deutlich schlechtere Ergebnisse.

2.3.8 Disambiguierung von Wortbedeutungen

Der wesentliche Unterschied von natürlichen Sprachen gegenüber formalen Sprachen ist ihre inhärente Mehrdeutigkeit [JM09b]. Die meisten natürlichen Sprachen sind sowohl hinsichtlich ihrer Syntax, als auch Semantik und Pragmatik mehrdeutig (siehe Abschnitt 2.3.1). Während syntaktische Mehrdeutigkeiten meist über Regelsätze erfasst werden können, stellen semantische und pragmatische Mehrdeutigkeiten bis heute die wesentlichen Herausforderungen der Computerlinguistik dar.

Bei der Auflösung von Mehrdeutigkeiten orientieren sich die meisten Ansätze an der menschlichen Herangehensweise. Menschen nutzen zur Auflösung sprachlicher Mehrdeutigkeiten den Kontext einer Aussage (umgebende Wörter, vorangegangene Sätze, erläuternde Paragraphen, Überschriften usw.). Diese Beobachtung machte bereits 1957 der britische Linguist John Rupert Firth; er leitete daraus folgende Proklamation ab [Fir57]:

> „*You shall know a word by the company it keeps!*“[39]

[39] Zu deutsch: Du sollst ein Wort anhand seiner Umgebung (er)kennen/verstehen!

Abbildung 2.14: Disambiguierung des Wortes *Bank* in einem Beispielsatz anhand eines Etikettensatzes für Bedeutungen: Die gewählte Etikette ist grün hervorgehoben.

In Anlehnung an diese Aussage konstruiert Lin folgendes Beispiel, um seinen Ansatz zur Bestimmung von Wortähnlichkeiten zu motivieren [Lin98][40]:

- A bottle of *tezgüino* is on the table.
- Everybody likes *tezgüino*.
- *Tezgüino* makes you drunk.
- We make *tezgüino* out of corn.

Lin argumentiert, dass die Bedeutung des unbekannten Wortes von Menschen intuitiv aus dem Kontext abgeleitet werden kann; ohne weitere Hilfestellungen können Menschen aus diesen vier Beispielen ableiten, dass es sich bei *tezgüino* um eine Art alkoholisches Getränk handeln muss. Dieses Gedankenkonstrukt lässt sich auch auf mehrdeutige Wörter übertragen. Beispielsweise lässt sich aus dem folgenden Beispielsatz die Bedeutung des Wortes *Bank* als Sitzgelegenheit eindeutig bestimmen, obwohl das Substantiv *Bank* viele unterschiedliche Bedeutungen hat: „Er sitzt auf einer *Bank* am Wegesrand.“

Diese Form der semantischen Mehrdeutigkeit (bzw. die Auflösung ebendieser) bezieht sich ausschließlich auf die Bedeutung einzelner Wörter im sprachlichen Kontext. Sie stellt damit die einfachste Form semantischer Mehrdeutigkeiten dar. In der Computerlinguistik ist die Auflösung von Wortmehrdeutigkeiten (engl. *Word Sense Disambiguation*, kurz *WSD*) die einzige formalisierte Aufgabe zur Mehrdeutigkeitsauflösung. Hierzu wurden verschiedene Vergleichsaufgaben definiert, unter anderem *SemEval* und *Senseval*, die jeweils in unterschiedlichen Versionen vorliegen[41] [EC01; MCK04; LH07; NJV13; MN15].

Bei der Auflösung von Wortmehrdeutigkeiten unterscheidet man zwei unterschiedliche Aufgaben [JM09b]: Die erste erfordert die Disambiguierung einer (kleinen) festen Menge von Wörtern, bezeichnet als *lexical sample task*. Die zweite fordert hingegen die Auflösung von Mehrdeutigkeiten bei allen (Inhalts-)Wörtern eines Textes, bezeichnet als *all-words task*. In beiden Fällen soll jedoch für jedes zu disambiguierende Wort eine Etikette vergeben werden, welche die Bedeutung des Wortes beschreibt. Abbildung 2.14 veranschaulicht diese Aufgabe anhand des zuvor eingeführten

[40] Das Beispiel ist eine abgewandelte Variante des bereits 1975 von Nida konstruierten [Nid15].
[41] Die Versionen unterscheiden sich einerseits in den zugrundeliegenden Textkorpora und zum anderen in den verwendet Etikettensätzen.

Beispielsatzes. Die Form der Etiketten ist nicht einheitlich. Einige Ansätze verwenden einen eigens definierten Etikettensatz, wodurch sie nicht mit anderen Verfahren vergleichbar sind. Andere Ansätze verwenden die Etikettensätze, die durch die zuvor genannten Vergleichsaufgaben festgelegt werden. Sehr verbreitet ist die Verwendung von *Synsets* aus *WordNet* (siehe Abschnitt 2.3.15.1) zur Beschreibung von Wortbedeutungen. Die Aufgabenstellung des *lexical sample task* findet heutzutage kaum mehr Beachtung. Moderne Ansätze betrachten nahezu ausschließlich die Disambiguierung aller Wörter. Die Schwierigkeit dieser Aufgabe besteht darin, dass einzelne Bedeutungen vergleichsweise selten auftreten und die Etikettensätze sehr umfangreich sind (beispielsweise im Vergleich zur Wortarterkennung, siehe Abschnitt 2.3.4.3). Aufgrund dieser Eigenschaften eignen sich intuitiv Verfahren, die auf überwachten maschinellen Lernen basieren, nicht für diese Aufgabe[42]. Nichtsdestotrotz stellen diese heutzutage den Stand der Technik dar[43]. Darüber hinaus gibt es einige wissensbasierte Ansätze, die sich beispielsweise die baumartige Struktur der Relationen in *WordNet* zunutze machen.

2.3.9 Themen-Extraktion, -Modellierung und -Etikettierung

Die Begriffe Themen-Extraktion (engl. *topic extraction*), Themen-Modellierung (engl. *topic modelling*) und Themen-Etikettierung (engl. *topic labeling*) bezeichnen sich überlappende Forschungsgebiete. Sie befassen sich mit der Problemstellung, für natürlichsprachliche Einheiten Themen zu bestimmen. Die betrachteten Einheiten sind beispielsweise einzelne Sätze, Paragraphen, Dokumente oder Dokumentsammlungen. Den natürlichsprachlichen Einheiten werden Themen in Form kurzer (wiederum natürlichsprachlicher) Beschreibungen zugeordnet. Diese Beschreibungen bestehen (je nach Verfahren) entweder aus einzelnen Schlagwörtern, einer Wortmenge oder einer kurzen Phrase und sollen den Inhalt der betrachteten Einheit möglichst treffend zusammenfassen. Der Großteil der Ansätze, die den drei Forschungsgebieten zugeordnet werden können, betrachtet ausschließlich geschriebene Sprache. Themenbeschreibungen dienen in der Computerlinguistik als Ausgangspunkt für weiterführende semantische Analysen. Beispielsweise können sie verwendet werden, um eine automatische Zusammenfassung zu erzeugen (engl. *automatic text summarization*). Themen werden weiterhin unter anderem für die Verschlagwortung, Katalogisierung und Indexierung von Dokumenten bzw. Dokumentmengen verwendet.

Die drei Forschungsgebiete unterscheiden sich hinsichtlich der konkreten Herangehensweisen zur Erzeugung von Themen. Die Unterschiede werden nachfolgend anhand eines Textdokuments als natürlichsprachliche Einheit beschrieben. Verfahren, die der Themen-Extraktion zugeordnet werden können, erzeugen Themen, indem sie relevante Wörter direkt aus dem Dokument entnehmen. Relevante Wörter können beispielsweise einfach häufige Wörter oder Wörter mit einem hohen

[42] Überwachte Lernverfahren können (im Allgemeinen) nur Klassen zuordnen, die bereits während der Trainingsphase bekannt sind. Aufgrund der genannten Eigenschaften steht jedoch zu erwarten, dass während des Training nicht alle bekannten Bedeutungen aufgetreten sind. Außerdem gelingt die Klassifikation besser, wenn mehr Beispiel derselben Klasse während des Trainings betrachtet werden können; auch dies ist aufgrund von selten auftretenden Bedeutungen unwahrscheinlich.

[43] Siehe hierzu `https://nlpprogress.com/english/word_sense_disambiguation.html`, zuletzt besucht am 24.02.2021.

Informationsgehalt sein. Die Themen bestehen meist aus Wortmengen, seltener aus Phrasen. In jedem Fall kommen sie wortwörtlich im Dokument vor. Themen-Extraktionsverfahren erzeugen zudem meist Themen für das gesamte Dokument, es werden keine einzelnen Themen für einzelne Abschnitte vergeben. Bei der Modellierung von Themen besteht die Aufgabe darin, Abschnitte im Dokument zu bestimmen, die thematisch zusammenhängen. Das bedeutet, derartige Verfahren untergliedern Dokumente, indem sie thematisch zusammenhängende Bereiche bilden, diese aber nicht benennen, also keine Beschreibung erzeugen. Diese Bereiche können im Dokument verteilt und damit unzusammenhängend sein. Verfahren aus dem Bereich der Themen-Etikettierung bestimmen für zuvor erzeugte Themen-Modelle Beschreibungen in Form von sogenannten Themen-Etiketten. Das bedeutet, für die einzelnen thematisch zusammenhängenden Bereiche wird je eine Etikette emittiert. Hierzu werden unterschiedliche Verfahren verwendet. Entweder werden ähnlich wie bei der Themen-Extraktion relevante Wörter direkt aus den jeweiligen Bereichen extrahiert oder es werden externe Themen-Etiketten vergeben. Externe Themen können unter anderem als Themenliste vorgegeben werden, oder es können Konzepte aus Wissensdatenbanken entnommen werden. Einige Ansätze verwenden beispielsweise Artikelnamen aus *Wikipedia* (siehe Abschnitt 2.3.15.5) als Etiketten. In jedem Fall sind diese Themen-Etiketten nicht dem betrachteten Dokument entnommen. Die Begrifflichkeiten werden in der Literatur uneinheitlich verwendet. Insbesondere werden Themen-Extraktion und Themen-Modellierung häufig gleichgesetzt.

2.3.10 Dialogsysteme

Dialogsysteme sind seit jeher eines der wichtigsten Anwendungsfelder der Computerlinguistik [JM09a]. Sie sind nicht nur für Interesse für die Forschungsgemeinde, auch die Industrie kann sie direkt (als Produkt) vermarkten. Virtuelle Assistenten oder sprachgesteuerte Navigationssysteme sind letztlich nichts anderes als mit Hintergrundwissen ausgestattete Dialogsysteme. Während sich die Forschung in den letzten Jahren zunehmend auf die Verwendung von statistischen Methoden zur Umsetzung von Dialogsystemen fokussiert, ist der Großteil der vermarkteten Systeme regelbasiert [TD11]. Als statistische Methoden werden hauptsächlich *teilweise beobachtbare Markov-Entscheidungsprozesse* (engl. partially observable Markov decision processes, kurz *POMDP*) [RPT00; TSY08; You+10], bestärkendes Lernen [Gaš+14] und neuronale Netze [Ser+16; Wen+17; Li+16] eingesetzt. Die von der Industrie favorisierten regelbasierten Ansätze untergliedern sich hingegen weiter in Aufruffluss-basierte (engl. *call flow*) [McT98; Pie+01], Rahmen-Semantik-basierte (engl. *frame semantic*) [SP00], Agenda-basierte [RX99; BR03] und Informationsstand-basierte (engl. *information state*) Ansätze [Mor+14]. Diese strikte Trennung der verwendeten Techniken zwischen Industrie und Forschung lässt sich auf die unterschiedlichen Rahmenbedingungen der jeweiligen Felder zurückführen. Während seitens der Forschung vor allem neue Techniken entwickelt und Grundlagen für zukünftige Entwicklungen gelegt werden, erfordert die Entwicklung vermarktbarer Produkte seitens der Industrie robuste sowie vorhersag- und nachvollziehbare Lösungen.

Die Architektur der meisten Dialogsysteme – unabhängig davon, ob sie einen statistik- oder regelbasierten Ansatz verfolgen – entspricht dem in Abbildung 2.15 dargestellten Schema. Üblicherweise bestehen Dialogsysteme dementsprechend aus den folgenden Modulen: (automatische)

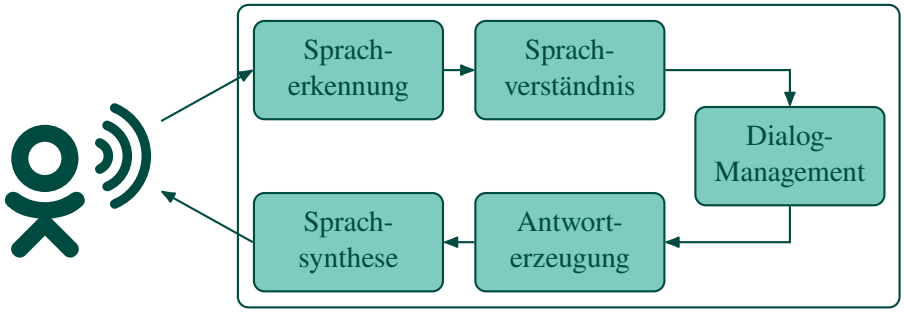


Abbildung 2.15: Grundlegende Architektur der meisten Dialogsysteme nach Jokinen und McTear [JM09a].

Spracherkennung, Sprachverständnis, Dialog-Management, Antworterzeugung und Sprachsynthese[44]. Die Spracherkennung überführt akustische Signale in eine Phonem- oder Wortsequenz (siehe Abschnitt 2.3.7). Die erzeugten Transkriptionen werden an das Sprachverständnis-Modul weitergereicht, welches die Äußerungen interpretiert. Üblicherweise wird an dieser Stelle die Wortsequenz in eine logikbasierte Darstellung überführt. Das Dialog-Management-Modul interpretiert anschließend die Ergebnisse des Sprachverständnis-Moduls. Die Interpretation fußt zumeist auf der letzten sowie vorangegangenen Äußerungen und – sofern verfügbar – zusätzlichen Informationen, wie situativem Kontext oder Weltwissen. Wie dieser Interpretationsprozess genau abläuft, ist der wesentliche Unterschied zwischen verschiedenen Dialogsystem-Ansätzen. Aber unabhängig davon, ob ein Ansatz neuronale Netze zur Interpretation nutzt oder Agenda-basiert ist, die zugrundelegende Idee ist wie folgt: Das Dialog-Management hat einen internen Zustand, der von vorherigen Eingaben und weiteren Einflussfaktoren abhängt. Eintreffende Äußerungen verändern den internen Zustand und lösen eine Reaktion des Systems aus. Im Antworterzeugungs-Modul wird die Systemreaktion verbalisiert; das bedeutet, es wird wieder eine natürlichsprachliche Wortsequenz erzeugt. Viele Ansätze nutzen hierfür Satzschablonen. Zuletzt wird im Sprachsynthese-Modul die Wortsequenz in ein Audiosignal überführt und ausgegeben.

Dialogsysteme werden hinsichtlich ihres Verhaltens gegenüber dem Nutzer in die folgenden drei Kategorien eingeteilt:

- *System-initiativ*: Das System tritt aktiv in den Dialog mit dem Nutzer ein. Das bedeutet, das System beginnt den Dialog, indem der Nutzer angesprochen wird, beispielsweise durch eine Begrüßung wie folgende: „Hello! How may I help you?“ Auch im weiteren Gesprächsverlauf bleiben derartige Systeme aktiv, beispielsweise durch weitere Nachfragen wie diese: „Would you like anything else?“
- *Nutzer-initiativ:* Das System bleibt vorwiegend passiv. Es tritt erst in den Dialog mit dem Nutzer ein, wenn dieser das System anspricht, beispielsweise durch eine Anrede wie folgende: „Hey System!“ Auch im weiteren Gesprächsverlauf, überlässt das System dem Nutzer die Initiative. Das bedeutet, das System beantwortet lediglich die Fragen des Nutzers und stellt üblicherweise keine Rückfragen.

[44] Die Unterscheidung nach statistik- oder regelbasiert manifestiert sich zumeist nur im *Dialog-Management*.

- *gemischt initiativ*: Das Verhalten des Systems wechselt je nach Gesprächssituation zwischen aktiv und passiv. Es gibt unterschiedliche Ausprägungen der gemischten Alternative. Manche Systeme sind zunächst passiv, stellen aber Fragen, um das Gespräch fortzusetzen. Andere sind zunächst aktiv, werden aber passiv, sobald sie das Anliegen des Nutzers für erledigt erachten. Zudem gibt es Unterschiede hinsichtlich der Verteilung der aktiven und passiven Anteile.

2.3.11 Verständnis natürlicher Sprache

Das Forschungsgebiet, das sich dem Verständnis natürlicher Sprache widmet (engl. *natural language understanding*, kurz *NLU*), ist ein Teilgebiet der Computerlinguistik und befasst sich mit der Aufstellung von Theorien und Modellen zur Darstellung von Bedeutungen natürlichsprachlicher Einheiten sowie Verfahren zur Analyse von Bedeutung [All95]. Unter anderem wird untersucht, welche Arten von Bedeutungen in natürlicher Sprache enthalten sein können, wie sich diese formalisieren und (maschinell verarbeitbar) darstellen lassen und wie Bedeutungen aus natürlichsprachlichen Artefakten extrahiert (und benannt) werden können. Das Forschungsgebiet befasst sich dementsprechend vorrangig mit der Semantik und Pragmatik natürlichsprachlicher Artefakte (siehe Abschnitt 2.3.1).

Tiefes Sprachverständnis

Das Ziel der meisten Ansätze ist nicht, das menschliche Sprachverständnis allgemeingültig nachzubilden[45]. Stattdessen wird durch ein partielles Verständnis einiger Aspekte der natürlichen Sprache versucht, die Grundlage für weiterführende Anwendungen, die Sprachverständnis benötigen, zu bilden. Wie umfangreich das partielle Verständnis eines Ansatzes ist, lässt sich im Wesentlichen anhand zweier Dimensionen bemessen. Die erste Dimension ist die Breite des Sprachverständnisses; das bedeutet, wie allgemein können Bedeutungen aus unterschiedlichen (inhaltlichen) Domänen erfasst werden (das heißt wie groß ist das unterstützte Vokabular). Die zweite Dimension ist die Tiefe; das bedeutet, wie genau (und aussagekräftig) können einzelne Bedeutungen beschrieben und erfasst werden. Semantische Rollen, wie in Abschnitt 2.3.4.8 beschrieben, bieten beispielsweise ein sehr breites Sprachverständnis; die vergleichsweise generischen Rollen beschreiben die Bedeutungseinheiten allerdings nicht in der Tiefe. Je *breiter* und *tiefer* das Verständnis der Sprache, desto *umfassender* ist es. In der Literatur wird nahezu ausschließlich der Begriff *tiefes Sprachverständnis* (engl. *deep natural language understanding*) verwendet, obwohl zumeist *umfassendes Sprachverständnis* gemeint ist. Aus diesem Grund wird in dieser Arbeit auch der Begriff *tiefes Sprachverständnis* in der Bedeutung des *umfassenden Sprachverständnisses* verwendet. Eine Vergrößerung des Sprachverständnisses in Richtung einer der beiden Dimensionen ist im Allgemeinen äußerst aufwendig, weshalb sich die meisten Ansätze darauf beschränken, das Sprachverständnis gerade so weit zu entwickeln, wie es weiterführende Anwendungen benötigen. Auch aus diesem Grund existieren heutzutage viele sehr gute Speziallösungen, aber keine, die allgemeines Sprachverständnis erreicht.

[45] Diese Zielstellung kann zum aktuellen Stand der Technik noch nicht erreicht werden. Gelänge es jedoch ein System zu erstellen, welches über allgemeines Sprachverständnis verfügt, könnte dieses den Turing-Test bestehen [Tur50]. Der von Alan Turing vorgeschlagene Test sieht vor, die (künstliche) Intelligenz eines Computersystems anhand dessen Sprachvermögens wie folgt zu bemessen: Könnte ein Mensch nach einem Dialog mit dem Computersystem und einem weiteren menschlichen Gesprächspartner nicht entscheiden, welches von beiden das Computersystem ist, so besteht das System den Test.

Anwendungsgebiete

Viele Anwendungen verwenden heutzutage Sprachverständnis-Komponenten. Beispielsweise beinhalten nahezu alle Dialogsysteme eine Einheit für Sprachverständnis (siehe Abschnitt 2.3.10) Diese wird benötigt, um die Äußerung des Nutzers deuten zu können und eine angemessene Antwort geben zu können. Auch in der maschinellen Übersetzung wird verstärkt auf Sprachverständnis gesetzt; schließlich kann die exakte Übersetzung von der Bedeutung eines Wortes, einer Phrase oder eines gesamten Satzes abhängen. Ansätze zum Sprachverständnis werden auch verwendet, um Dokumente und Dokumentsammlungen (semantisch) zu kategorisieren und deren Inhalt zu analysieren.

Auch Systeme zur Programmierung mit natürlicher Sprache (siehe Kapitel 3) müssen die Bedeutung natürlichsprachlicher Programmbeschreibungen verstehen, um daraus programmatische Strukturen synthetisieren zu können. Die Deutung natürlicher Sprache als Programm stellt insofern eine Herausforderung dar, als dass sich natürliche Sprachen deutlich von formalen Sprachen (wie Programmiersprachen) unterscheiden. Der wesentlichste Unterschied ist die inhärente Mehrdeutigkeit natürlicher Sprache; je nach Kontext können sich natürlichsprachliche Einheiten (wie Wörter, Phrasen oder Sätze) auf unterschiedliche Konzepte beziehen[46]. In Programmiersprachen werden Konzepte (wie beispielsweise Klassen und Methoden) hingegen immer eindeutig benannt. Ein weitere wichtiger Unterschied ist die Präsenz von impliziten Referenzen in natürlicher Sprache; auch Bezüge zwischen zwei (oder mehr) natürlichsprachlichen Einheiten können häufig nur aus dem Kontext erschlossen werden. In Programmiersprachen werden Referenzen immer explizit hergestellt. Dementsprechend kann eine Abbildung von natürlicher Sprache auf eine Programmiersprache nicht mithilfe einfacher Such- oder Übersetzungstechniken gelingen; stattdessen muss die Bedeutung der natürlichen Sprache vielmehr interpretiert werden (das heißt unter anderem müssen Mehrdeutigkeiten aufgelöst und implizite Referenzen expliziert werden), damit aus einer natürlichsprachlichen Beschreibung Quelltext synthetisiert werden kann.

2.3.12 Verständnis gesprochener Sprache

Das Forschungsgebiet Verständnis von gesprochener Sprache (engl. *spoken language understanding*, kurz *SLU*) ist Teilgebiet des Forschungsgebiets zum Verständnis von natürlicher Sprache, das im vorangegangenen Abschnitt beschrieben wurde. Der Fokus der Betrachtungen liegt jedoch auf den Phänomenen gesprochener Sprache und den daraus folgenden Herausforderungen bei der Analyse der Bedeutung gesprochener Artefakte [TD11].

Herausforderung im Vergleich zu geschriebener Sprache

Eine wesentliche Herausforderung ist, dass gesprochene Sprache im Allgemeinen ungrammatisch ist. Natürlichsprachliche Äußerungen werden häufig nicht aus vollständigen Sätzen gebildet; außerdem enthalten sie häufig grammatikalische Fehler oder zumindest unübliche Konstruktionen. Dies gilt insbesondere für spontane Sprache, bedingt aber auch für geplante Sprache, wie beispielsweise vorbereitete Reden. Vor allem spontane Sprache enthält zudem häufig sogenannte Disfluenzen.

[46] Eine Form der Mehrdeutigkeit ist die Mehrdeutigkeit einzelner Wörter (siehe Abschnitt 2.3.8).

Disfluenzen sind Unterbrechungen des Sprechflusses. Derartige Unterbrechungen könnten entweder sogenannten Hesitationen sein, das heißt kurze Pausen beim Sprechen. Verwendet der Sprecher Verzögerungslaute, wie beispielsweise *uhm* oder *äh*, um die Pause zu füllen, spricht man von sogenannten gefüllten Pausen. Eine weitere Form von Disfluenzen sind verbalisierte Korrekturen bei denen ein Sprecher entweder zuvor getätigte Aussagen inhaltlich korrigiert (bzw. revidiert) oder eine Aussage, die einen Versprecher enthielt, verbessert. Außerdem hängt gesprochene Sprache deutlich stärker von individuellen Charakteristika des Sprechers ab als geschriebene Sprache; gemeint sind vor allem die Betonung, der Dialekt und der Akzent des Sprechers, aber auch bevorzugte Formulierungen (bzw. Redewendung) sowie das Vokabular. Zuletzt werden üblicherweise Systeme zur automatischen Spracherkennung verwendet, um das Audiosignal in eine textuelle Form zu überführen (siehe Abschnitt 2.3.7), bevor die Bedeutung des Gesagten analysiert werden kann. Spracherkennungssysteme arbeiten jedoch nicht fehlerfrei; das bedeutet, die Verfahren zum Verständnis gesprochener Sprache müssen damit umgehen können, das in den zu analysierenden Äußerungen fehlerhafte Wörter enthalten sind. Außerdem erzeugen Spracherkennungssysteme normalerweise keine Satzzeichen, vornehmlich weil es nicht möglich ist, für ungrammatische Sprache sinnhafte Interpunktionen zu inferieren.

Herangehensweisen

Aufgrund dieser Unterschiede zwischen gesprochener und geschriebener Sprache können viele *NLU*-Ansätze, die vorrangig für geschriebene Sprache entwickelt wurden, nicht unmittelbar angewendet werden. Um dieses Problem zu beheben, gibt es im Wesentlichen zwei Herangehensweisen. Die erste sieht vor, ein vorhandenes *NLU*-System zu verwenden bzw. ein *NLU*-System zu erstellen und diesem ein System zur automatischen Spracherkennung voranzustellen (siehe Abschnitt 2.3.7). Um den Herausforderungen gesprochener Sprache gerecht zu werden, werden die Sprachanalysen des *NLU*-Systems angepasst bzw. erweitert[47]. Ansätze, welche der zweiten Herangehensweise zuzuordnen sind, verknüpfen das Sprachverständnis direkt mit der Spracherkennung. Das bedeutet, Bedeutungsanalysen werden direkt in den automatischen Spracherkenner integriert. Beispielsweise erzeugen einige Verfahren bedeutungsbeschreibende Etiketten direkt auf Phonemebene.

Anwendungsgebiete

Für das Verständnis gesprochener Sprache gibt es unterschiedliche Anwendungsgebiete; für alle dieser Anwendungsgebiete gibt es jeweils Ansätze, die sich den beiden zuvor beschriebenen Herangehensweisen zuordnen lassen. Die erste Anwendung ist zugleich der meist betrachtete Forschungsgegenstand: Dialogsysteme. Wie schon im vorangegangen Abschnitt beschrieben, müssen Dialogsysteme Äußerungen des Nutzers verstehen, um etwas Sinnhaftes erwidern zu können.

Eine weitere Anwendung ist die Informationsrückgewinnung aus Sprachartefakten (engl. *speech information retrieval*, kurz *SRI*). Dieser Forschungszweig beschäftigt sich damit, Sprachartefakte (beispielsweise aufgezeichnete Reden oder Dialogmitschnitte) derart aufzubereiten, dass sie anhand ihres Inhalts auffindbar sind (z. B. durch Suchmaschinen). Zu den Aufgaben gehören somit die Katalogisierung und Inhaltsanalyse von Sprachartefakten. Ein weiteres Anwendungsgebiet ist

[47] Auch der in dieser Arbeit verfolgt Ansatz lässt sich dieser ersten Herangehensweise zuordnen (siehe Kapitel 5).

die Beantwortung von Fragen (engl. *question answering*). Ansätze dieser Kategorie verwenden Sprachverständnisverfahren, um Nutzeranfragen zu interpretieren und in eine Darstellung zu überführen, die es erlaubt, Informationen aus einer Wissensquelle zu beziehen (siehe Abschnitt 2.3.15). Beispielsweise können aus natürlichsprachlichen Äußerungen Datenbankabfragen erzeugt werden (siehe Abschnitt 3.4).

Auch virtuelle Assistenzsysteme, wie *Siri*, *Google Assistant* oder *Amazon Alexa* gehören prinzipiell zu dieser Kategorie von Anwendungen. Allerdings besitzen Assistenzsysteme zusätzlich (zumeist rudimentäre) Fähigkeiten zur Dialogführung und können nicht nur Fragen beantworten, sondern auch Aufgaben erfüllen. Aus technischer Sicht unterscheiden sich die benötigten Sprachverständnisfähigkeiten zur Umsetzung der beiden Funktionalitäten jedoch nicht. Unabhängig davon, ob ein Nutzer das morgige Wetter erfragt (Fragebeantwortung) oder ein neuer Kalendereintrag erstellt werden soll (Aufgabenerfüllung), erfordert beides die Synthese eines schnittstellenkonformen Aufrufs an einen Dienstgeber.

Um die Programmierung anhand gesprochener natürlichsprachlicher Äußerung zu ermöglichen (wie es in der hiesigen Arbeit angestrebt wird), wird ein besonders tiefes Verständnis natürlicher Sprache benötigt. Einerseits müssen, ähnlich wie bei Systemen zur Fragebeantwortung oder virtuellen Assistenzsystemen, Anweisungen (bzw. Aufrufe) synthetisiert werden. Allerdings bestehen Programme nicht nur aus einer einzelnen Anweisung, sondern setzen sich aus einer Reihe (teilweise voneinander abhängiger) Anweisungen zusammen. Hinzu kommen Kontrollstrukturen, die nur erzeugt werden können, wenn natürlichsprachliche Äußerungen entsprechend interpretiert werden. Damit die Programmierung mit gesprochener Sprache gelingen kann, muss also die Semantik und Pragmatik von Äußerungen verstanden und interpretiert werden (siehe Abschnitt 2.3.1).

2.3.13 Ähnlichkeitsmaße für Zeichenketten

In der Computerlinguistik ist es häufig erforderlich zu bestimmen, wie ähnlich sich zwei Wörter hinsichtlich ihrer Oberflächenform sind. Hierzu werden die Zeichenketten verglichen, aus denen die Wörter bestehen. Ähnlichkeitsmaße können dementsprechend aus Distanz-Metriken für Zeichenketten abgeleitet werden. Es existieren viele unterschiedliche Ähnlichkeitsmaße für Zeichenketten[48]; an dieser Stelle werden nur die im Kontext der vorliegenden Arbeit relevanten vorgestellt: die Levenshtein-, die Jaro-Winkler- und die unscharfe Ähnlichkeit.

2.3.13.1 Levenshtein-Distanz und Levenshtein-Ähnlichkeit

Die Levenshtein-Distanz ist eine Metrik zur Bestimmung der Unterschiede zweier Zeichenketten [Lev66]. Die Distanz zweier Zeichenketten wird bestimmt, indem die minimale Anzahl an Editieroperationen (das heißt Einfügungen, Entfernungen und Vertauschungen) bestimmt wird, die benötigt wird, um die erste Zeichenkette in die zweite zu überführen. Aus diesem Grund wird die Levenshtein-Distanz häufig auch als Editierdistanz (engl. *edit distance*) bezeichnet. Formal ist die

[48] Ein Vergleich der verschiedenen Maße kann der Arbeit von Cohen et al. entnommen werden [CRF03]

Levenshtein-Distanz Lev zweier Zeichenketten a und b durch folgende Gleichung definiert, wobei i und j zu Beginn den Längen der Zeichenketten, also $|a|$ und $|b|$ entsprechen:

$$Lev_{a,b}(i,j) = \begin{cases} max(i,j) & \text{falls } min(i,j) = 0, \\ min \begin{cases} Lev_{a,b}(i-1,j) + 1 \\ Lev_{a,b}(i,j-1) + 1 \\ Lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{andernfalls.} \end{cases} \tag{2.4}$$

Die Levensthein-Ähnlichkeit Sim_{Lev} zweier Zeichenketten a und b wird aus der Levenshtein-Distanz abgeleitet, indem diese über die Länge der längeren Zeichenkette normiert wird:

$$Sim_{Lev}(a,b) = 1 - \frac{Lev_{a,b}(|a|,|b|)}{max(|a|,|b|)} \tag{2.5}$$

Auf diese Weise wird der Anteil der übereinstimmenden Zeichenkettenbestandteile im Wertebereich [0; 1] angegeben.

2.3.13.2 Jaro-Winkler-Ähnlichkeit

Die Jaro-Winkler-Ähnlichkeit basiert auf der Jaro-Ähnlichkeit [Win90], die wiederum eine Variante der Jaro-Distanz ist [Jar89]. Die Jaro-Ähnlichkeit Sim_{Jaro} bestimmt die Anzahl und Reihenfolge übereinstimmender Zeichen zweier Zeichenketten a und b:

$$Sim_{Jaro}(a,b) = \frac{1}{3}\left(\frac{m(a,b)}{|a|} + \frac{m(a,b)}{|b|} + \frac{m(a,b)-t}{m(a,b)}\right) \tag{2.6}$$

Dabei bezeichnet $m(a,b)$ die Anzahl der übereinstimmenden Zeichen der beiden Zeichenketten a und b. Zwei Zeichen gelten als übereinstimmend, wenn $a_i = b_j$ und der Abstand geringer ist als die Hälfte der längeren der beiden Zeichenketten weniger eins[49]. Das bedeutet, folgende Ungleichung muss erfüllt sein:

$$|i-j| \leq \left\lfloor \frac{max(|a|,|b|)}{2} \right\rfloor - 1 \tag{2.7}$$

Die beiden Zeichenketten *aebdcz* und *bacde* stimmen gemäß dieser Definition an vier Stellen überein (da die beiden e zu weit voneinander entfernt stehen), das heißt $m(aebdcz, bacde) = 4$). Weiterhin ist t definiert als die Anzahl an Transpositionen, die nötig sind um $m(a,b)$ zu bestimmen. Für die beiden zuvor gezeigten Zeichenketten sind zwei Vertauschungen (in einer der Zeichenketten) nötig, um die Übereinstimmung ermitteln zu können: die Zeichen a und b und c und d müssen getauscht werden. Für t ergibt sich somit der Wert zwei.

[49] Wiederholende Zeichen in den Zeichenketten spielen keine Rolle. Für jedes Zeichen einer Zeichenkette wird nur maximal genau ein Entsprechung innerhalb der anderen Zeichenkette bestimmt – und zwar das mit dem geringsten Abstand.

Die Jaro-Winkler-Ähnlichkeit Sim_{JW} verwendet die Jaro-Ähnlichkeit, erzeugt jedoch höhere Ähnlichkeitswerte, wenn die Zeichenketten ein gemeinsames Präfix besitzen. Hierzu wird die Länge l des gemeinsamen Präfixes bestimmt; die maximale Länge des Präfix, der in die Berechnung eingeht, wird auf vier begrenzt ($l_{max} = 4$). Der Einfluss der Präfixübereinstimmung auf die Distanzbewertung kann über den Faktor p angepasst werden[50].

$$Sim_{JW}(a, b) = Jaro(a, b) + l \cdot p(1 - Jaro(a, b)) \tag{2.8}$$

Der Wertebereich der Jaro-Winkler-Ähnlichkeit ist [0; 1] (bei geeigneter Wahl von p), wobei der Wert 1 eine exakte und 0 keine Übereinstimmung der Zeichenketten bedeutet.

2.3.13.3 Unscharfe Ähnlichkeit

Unscharfe Zeichenkettenvergleiche sind eigentlich eine Klasse von Verfahren. Die hier beschriebene unscharfe Ähnlichkeit basiert auf der unscharfen Bewertung (engl. *fuzzy score*), wie sie von der Bibliothek *Apache Commons* implementiert wird[51]. Die Bewertung $Fuzzy(a, b)$ basiert auf der Anzahl von Übereinstimmungen einzelner Zeichen a_i und b_j zweier Zeichenketten a und b. Teilsequenzenkönnen können dabei verschoben werden, um an der anderen Zeichenkette ausgerichtet zu werden, Vertauschungen sind jedoch nicht zulässig. Zusammenhängende Sequenzen werden ab der zweiten Übereinstimmung dreifach gewertet. Die unscharfe Bewertung für die Zeichenketten *loremipsum* und *ipsumlorem* wird demnach wie folgt bestimmt:

```
l o r e m i p s u m - - - - -
- - - - - i p s u m l o r e m
0 0 0 0 0 1 3 3 3 3 0 0 0 0 0   = 13
```

Für die Zeichenketten *abcabcabc* und *abababab* wird hingegen folgende Bewertung bestimmt:

```
a b c a b c a b c
a b - a b - a b -
1 3 0 1 3 0 1 3 0   = 12
```

Die unscharfe Bewertung betrachtet nur Übereinstimmungen, Unterschiede zwischen den Zeichenketten werden nicht betrachtet. Da die unscharfe Bewertung einen unbegrenzten Wertebereich hat, wird die zugehörige Ähnlichkeit Sim_{Fuzzy} definiert, indem die Bewertung normiert wird:

$$Sim_{Fuzzy}(a, b) = \frac{Fuzzy(a, b)}{3 \cdot max(|a|, |b|) - 2} \tag{2.9}$$

[50] Winkler verwendet in seiner Arbeit als Wert 0,1; auch die Bibliothek *Apache Commons* verwendet diesen. Es sollte in jedem Fall ein Wert gewählt werden, der kleiner ist als der Kehrwert der maximalen Länge des Präfixes (das heißt $p \leq 1/l_{max}$), da die Jaro-Winkler-Ähnlichkeit andernfalls Werte größer 1 annehmen kann. Für $l_{max} = 4$ sollte dementsprechend $p \leq 0,25$ gelten.

[51] *Apache Commons*: `https://commons.apache.org/proper/commons-text/apidocs/org/apache/commons/text/similarity/FuzzyScore.html`, zuletzt besucht am 24.02.2021.

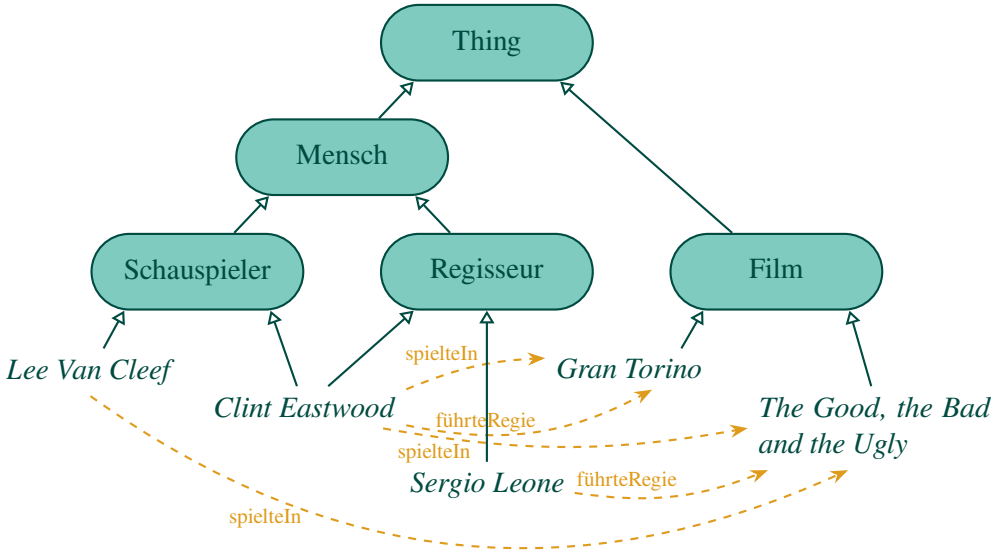


Abbildung 2.16: Ein Auschnitt einer Ontologie zur Domäne *Film*: Konzepte sind als grüne Knoten dargestellt, Individuen sind kursiv geschrieben und Relationen orange hervorgehoben.

Der Nenner der Gleichung zur Normierung lässt sich auf die maximal erreichbare Bewertung bei völliger Übereinstimmung zweier Zeichenketten zurückführen. In diesem Fall werden alle Zeichen bis auf das erste dreifach gewertet.

2.3.14 Ontologien

Der Begriff Ontologie bezeichnet den Fachbereich der Philosophie, der sich mit allem Seienden und den Zusammenhängen zwischen Entitäten sowie den Strukturen, die sich daraus bilden, beschäftigt. Davon leitet sich das Konstrukt der Ontologie ab, wie es im Kontext der Informatik verwendet wird. Gruber definiert Ontologien folgendermaßen [Gru93]:

> „*An ontology is an explicit specification of a shared conceptualization.*“[52]

Das bedeutet, Ontologien werden dazu verwendet, Wissen über einen Gegenstandsbereich (das heißt eine Domäne) explizit zu modellieren[53], sodass dieses Modell genutzt werden kann, um Wissen auszutauschen. Ontologien bestehen grundlegend aus Konzepten, Individuen und Relationen. Konzepte (auch als Klassen bezeichnet) beinhalten Individuen (auch als Instanzen bezeichnet); dabei ist es möglich, dass ein Individuum mehreren Konzepten angehört. Relationen können zwischen Konzepten oder zwischen Individuen gelten. Konzepte können wiederum in übergeordneten Konzepten enthalten sein (sogenannte Oberkonzepte); dementsprechend bilden sich für gewöhnlich Konzepthierarchien. Als abstraktes Oberkonzept einer Ontologie, in die alle weiteren Konzepte eingeordnet werden, wird zumeist das Konzept *Thing* (dt. *Ding*) verwendet. Relationen können uni- oder bidirektional sein; außerdem können Konzepten und Individuen mithilfe von Datenrelationen auch Attribute und Werte zugewiesen werden. Abbildung 2.16 zeigt den Ausschnitt einer

[52] Zu deutsch: Eine Ontologie ist eine explizite Spezifikation einer geteilten Konzeptualisierung.
[53] Im Gegensatz zur Philosophie modellieren Ontologien immer nur einen Ausschnitt des Wissens über die Wirklichkeit.

Beispielontologie zur Domäne *Film*. Ontologien eignen sich zur Bereitstellung von Wissen für Computersysteme. Einerseits entsprechen Ontologien in ihrem Aufbau der menschlichen Gedankenwelt und können daher von Domänenexperten ohne Programmierkenntnisse erstellt werden. Andererseits bieten sie eine wohl definierte und explizite Darstellung von Wissen, sodass dieses algorithmisch verwendet werden kann. In der Computerlinguistik werden Ontologien vorrangig dafür genutzt, Sprachanalyseverfahren Wissen über die Domäne der sprachlichen Artefakte zur Verfügung zu stellen oder menschliches Allgemeinwissen zu modellieren, um Sprachverständnisfähigkeiten von Menschen zu imitieren. Das Datenformat *OWL* gilt als De-Facto-Standard zur Darstellung von Ontologien; *OWL* ist eine formale Auszeichnungssprache und basiert auf *RDF*, welche zur Darstellung von Prädikaten entwickelt wurde [Av04].

2.3.15 Wissensdatenbanken

Im Kontext der Computerlinguistik sind zwei Arten von Wissensdatenbanken von besonderem Interesse. Zum einen solche, die sprachliches Wissen beinhalten; diese werden vor allem zur semantischen Interpretation von (zuvor extrahierten) syntaktischen Informationen verwendet. Zum anderen werden Datenbanken verwendet, die Faktenwissen bzw. sogenanntes Weltwissen beinhalten; diese werden dazu genutzt, Sprachanalysen mit Hintergrundwissen auszustatten und so menschliches Allgemeinwissen nachzubilden. In den folgenden Abschnitten wird eine Auswahl von Wissensdatenbanken vorgestellt, die für die Arbeit relevant sind. Hinsichtlich der Wissensdatenbanken, die sprachliches Wissen enthalten, sind dies *WordNet*, *FrameNet*, *VerbNet* und *PropBank*. Seitens der Weltwissensdatenbanken werden *Wikipedia* (bzw. *DBpedia*) und *Cyc* betrachtet.

2.3.15.1 WordNet

WordNet ist eine manuell angelegte, hierarchisch organisierte, lexikalische Datenbank der englischen Sprache, die seit 1985 von der *Princeton University* entwickelt wird [Fel98; Fel10]. Die Einträge der Datenbank bilden sogenannte *Synsets*. Diese bestehen aus einer Menge von (Beinahe-)Synonymen (Begriffen) und bilden eine Bedeutungseinheit. Zur Beschreibung der Bedeutung ist jedem *Synset* außerdem eine Glosse zugeordnet. Einzelne Begriffe können in mehreren *Synsets* enthalten sein. Das *Synset* zum Begriff *computer* im Sinne einer Maschine, die automatisch Berechnungen durchführt, lautet wie folgt:

[computer, computing machine, computing device, data processor,
electronic computer, information processing system]
– Glosse: *a machine for performing calculations automatically*

Außerdem sind in *WordNet* lexikalische Relationen hinterlegt, die zwischen *Synsets* bestehen. Unter anderem gibt es Ober-Unterbegriff- (Hyperonymie und Hyponymie) und Teil-Ganzes-Beziehungen (Meronymie und Holonymie). Durch diese Relationen wird eine Hierarchie von *Synsets* gebildet; *WordNet* ist somit auch ein Thesaurus. Die *Synsets* sind nach Wortarten untergliedert; es existieren *Synset*-Sammlungen für Substantive, Verben, Adjektive und Adverbien. Der Umfang der aktuellen

Version 3.1, die 2011 veröffentlicht wurde, beträgt 117659 *Synsets*, davon 82115 für Nomen, 13767 für Verben, 18156 für Adjektive und 3621 für Adverbien[54].

2.3.15.2 FrameNet

FrameNet ist eine manuell angelegte, lexikalische Datenbank der englischen Sprache zur Beschreibung der Semantik, die seit 1997 vom *International Computer Science Institute* in Berkeley entwickelt wird [BFL98]. Die Einträge bilden sogenannte semantische Rahmen (engl. *semantic frames*), welche auf die von Fillmore entwickelte Theorie der *frame semantics* zurückgehen [Fil76]. Die Theorie besagt, dass die Semantik einer sprachlichen Einheit (z. B. eines Wortes) immer im Kontext von Weltwissen beschrieben werden muss. Ein Wort erzeugt dementsprechend (zusammen mit dem sprachlichen Kontext) beim Leser (oder Hörer) eine innere Vorstellung, welche mithilfe eines semantischen Rahmens beschrieben werden kann. Folglich beinhaltet ein Rahmen weitere sprachlichen Einheiten, die im selben Kontext auftreten. Die sprachlichen Einheiten bilden sogenannte *lexical units*. Diese sind Wörter (oder Phrasen) in einer bestimmten Bedeutung; das heißt pro Wort existieren potenziell mehrere *lexical units*. Ein semantischer Rahmen besteht aus *lexical units* und Relationen zwischen diesen. Der Rahmen *Text_creation* beinhaltet beispielsweise die *lexical units Author*, *Text* und *write* sowie die Relationen, die den Fakt beschreiben, dass Autoren Texte schreiben. Darüber hinaus können Rahmen weitere optionale Relationen und *lexical units* enthalten. Für den Rahmen *Text_creation* gibt es beispielsweise eine optionale Relation zur Beschreibung eines Adressaten. Außerdem können auch semantische Rahmen über spezielle Relationen miteinander in Verbindung stehen. Dadurch bildet sich ein Netz von semantischen Rahmen. *FrameNet* umfasst derzeit 1224 semantische Rahmen, die 6985 *lexical units* enthalten, sowie 1878 Relationen, die Rahmen verknüpfen[55].

2.3.15.3 VerbNet

VerbNet ist eine weitere manuell angelegte, lexikalische Datenbank der englischen Sprache zur Beschreibung der Semantik, welche jedoch Verben als zentrales Element definiert [Sch05; Kip+06]. *VerbNet* basiert auf der Annahme, dass die Semantik einer Phrase anhand des Verbs und dessen Argumenten erfasst werden kann. Ausgehend davon werden analog zu *FrameNet* Rahmen für Verben definiert und Argumente zugeordnet. *VerbNet* umfasst derzeit 5257 solcher Verb-Rahmen[56].

2.3.15.4 PropBank

Eine weitere Ressource, welche die Semantik ausgehend von Verben beschreibt, ist die *Proposition Bank*, kurz *PropBank* [KP02; PGK05]. Bei der PropBank handelt es sich um ein bearbeitetes

[54] Statistiken zu *WordNet 3.0*: `https://wordnet.princeton.edu/documentation/wnstats7wn`, zuletzt besucht am 24.02.2021.

[55] Statistiken zu *FrameNet*: `https://framenet.icsi.berkeley.edu/fndrupal/current_status`, zuletzt besucht am 24.02.2021.

[56] Statistiken zu *VerbNet*: `https://verbs.colorado.edu/verbnet/`, zuletzt besucht am 24.02.2021.

Korpus, welches aus einem Teil der *Penn Tree Bank* besteht, der mit Etiketten zur Beschreibung der Semantik annotiert wurde. Die Etiketten beschreiben die konkrete Semantik von *A**-Rollen (siehe Abschnitt 2.3.4.8) in Abhängigkeit vom Prädikat des Satzes. Anhand des Korpus konnten wiederum semantische Rahmen abgeleitet werden, die in einer lexikalischen Datenbank zusammengefasst wurden. Die semantischen Rahmen bilden abstrakte *A**-Rollen auf konkrete Rollen ab, die im Kontext eines Verbs mit einer bestimmten Bedeutung gültig sind. Der Rahmen für das Verb *give* im Sinne von *eine Sache übergeben* ist beispielsweise wie folgt definiert:

A0: giver (dt. *Gebender*)

A1: thing given (dt. *übergebene Sache*)

A2: entity given to (dt. *erhaltende Entität bzw. Empfänger*)

Für die Rahmen der drei lexikalischen Datenbanken *FrameNet*, *VerbNet* und *PropBank* existieren Abbildungen. Es ist also möglich für einen konkreten *PropBank*-Rahmen den entsprechenden *FrameNet*- oder *VerbNet*-Rahmen zu bestimmen. Dadurch können potenziell etwaige Lücken in den einzelnen Datenbanken bei der Verwendung der semantischen Repräsentationen ausgefüllt werden.

2.3.15.5 Wikipedia und DBpedia

Die *Wikipedia* ist eine offene Online-Enzyklopädie, die seit 2001 existiert und kontinuierlich erweitert wird[57]. Sie ist heutzutage auch die Enzyklopädie mit dem größten Gesamtumfang; alleine der englischsprachige Teil der *Wikipedia* umfasst derzeit knapp 6,3 Millionen Artikel[58]. Insgesamt sind Artikel in mehr als 300 Sprachen verfügbar[59]. Jeder Artikel umfasst ein Thema, beispielsweise beschreibt der Artikel *Karlsruher Institut für Technologie* die in Karlsruhe ansässige technische Universität, ihre Geschichte, organisatorische Gliederung, den Forschungsbetrieb usw.[60] Nahezu alle Artikel enthalten Begriffe, die in einem anderen Artikel beschrieben werden; für diese Begriffe können (üblicherweise beim erstmaligen Auftreten) Querverweise in Form von Verknüpfungen (engl. *hyper link*) zum erklärenden Artikel eingefügt werden. Der Artikel *Karlsruher Institut für Technologie* enthält beispielsweise einen Querverweis zum Artikel *Technische Universität*.

Die *Wikipedia* wird von der *Wikimedia Foundation Inc.* betrieben, die Artikel stammen jedoch von registrierten und nicht registrierten Nutzern der *Wikipedia*; auch die Qualitätssicherung geschieht durch die Nutzer. Trotz dieses offenen Konzepts zur Erstellung von Artikeln zeigte eine Untersuchung, dass die *Wikipedia* bereits 2005 (vier Jahre nach Beginn des Projekts) eine vergleichbare Genauigkeit und Themenabdeckung wie die *Encyclopedia Britannica* aufwies.

[57] Startseite der englischsprachigen *Wikipedia*: `https://en.wikipedia.org/wiki/Main_Page`, zuletzt besucht am 24.02.2021.

[58] Derzeit sind 6182360 Artikel in der englischsprachigen *Wikipedia* enthalten. Siehe hierzu: `https://en.wikipedia.org/wiki/Special:Statistics?action=raw`, zuletzt besucht am 24.02.2021.

[59] Derzeit gibt es in 319 Sprachen mindestens einen Artikel. Siehe hierzu: `https://meta.wikimedia.org/wiki/List_of_Wikipedias`, zuletzt besucht am 24.02.2021.

[60] Artikel zum *Karlsruher Institut für Technologie*: `https://en.wikipedia.org/wiki/Karlsruhe_Institute_of_Technology`, zuletzt besucht am 24.02.2021.

Aufgrund des enormen Umfangs[61] und der relativ einheitlichen Sprache der Artikel wird die *Wikipedia* im Kontext der Computerlinguistik häufig als Korpus verwendet.

Das Projekt *DBpedia* verfolgt das Ziel, die *Wikipedia* als elektronisch verarbeitbare Wissensdatenbank aufzubereiten [Aue+07; Leh+15]. Hierzu wird aus den Inhalten der *Wikipedia* automatisch ein Graph (bzw. eine Ontologie) generiert. Dadurch wird das in der *Wikipedia* enthaltene Wissen direkt algorithmisch verwendbar. In *DBpedia* bilden die Artikel Konzepte (sogenannte *things*), die untereinander und mit Faktenwissen, das aus den Artikeln extrahiert wurde, verknüpft sind. Die Relationen werden unter anderem durch die Querverweise in den Artikeln gebildet. Hinzu kommen Kategorie-, Oberbegriff- und ähnliche Beziehungen, die eine Hierarchie der Konzepte erzeugen. Zur Erzeugung der *DBpedia* werden neben der englischsprachigen *Wikipedia* weitere große Wikipedias in anderen Sprachen verwendet; alleine der englischsprachige Ausschnitt von *DBpedia* umfasst knapp 4,6 Millionen Konzepte[62]. Viele der Konzepte von *DBpedia* sind mit Konzepten anderer Wissensdatenbanken verknüpft, unter anderem auch mit der öffentlich zugänglichen Variante der im nachfolgenden Abschnitt beschriebenen Weltwissenontologie *Cyc*.

2.3.15.6 Cyc

Das *Cyc*-Projekt verfolgt seit 1984 das Ziel eine Weltwissenontologie zu erschaffen [Len95]. Die Ontologie soll dementsprechend nicht nur einen Ausschnitt der Wirklichkeit beschreiben (so wie es Ontologien normalerweise tun); stattdessen werden Fakten, Konzepte und Relationen erstellt, die das Wissen über die Welt möglichst vollumfänglich wiedergeben. Die Ontologie enthält daher auch vergleichsweise banale Fakten, wie beispielsweise, dass Flüssigkeiten einen oben offenen Behälter verlassen, wenn dieser mit der Öffnung nach unten gedreht wird, oder dass Menschen nachts schlafen und dass man Menschen nur wecken kann, wenn sie zuvor geschlafen haben. Die Zielstellung, sämtliches Wissen über die Welt zu modellieren, wirft seit Beginn des Projekts zwei wesentliche Probleme auf. Zum einen ist der Umfang an Informationen zur Beschreibung allen Wissens über die Welt grundsätzlich unbeschränkt; trotz mehr als 400000 Konzepten und über einer Million Individuen, die bisher gesammelt werden konnten[63], beschreibt die *Cyc*-Ontologie immer noch nur einen geringen Teil des Allgemeinwissens. Zum anderen können sich Informationen je nach Kontext widersprechen; ähnlich wie Wörter, die je nach sprachlichem Kontext unterschiedliche Bedeutungen annehmen, können auch Fakten je nach Kontext variieren. Beispielsweise verliert der zu Beginn genannte Fakt zum Verhalten von Flüssigkeiten in Behältnissen seine Gültigkeit, wenn sich beides statt auf der Erde im Weltall befindet. Um diese Probleme zu lösen, werden sogenannte Mikrotheorien (engl. *micro theories*) gebildet; diese bilden Subräume, in denen Konzepte im Kontext anderer Konzepte beschrieben werden. Die in Mikrotheorien gesammelten Informationen sind widerspruchsfrei, können aber Informationen in

[61] Zusammengenommen beinhalten die Artikel der englischsprachigen *Wikipedia* derzeit mehr als 3,8 Milliarden (3800720274) Wörter. Siehe hierzu: `https://en.wikipedia.org/wiki/Special:Statistics?action=raw`, zuletzt besucht am 24.02.2021.

[62] *DBpedia*: `https://wiki.dbpedia.org/about`, zuletzt besucht am 24.02.2021.

[63] Stand 2017.

anderen Mikrotheorien widersprechen. Mikrotheorien bilden sozusagen Domänen; *Cyc* ist somit eigentlich eine Sammlung vieler einzelner Domänen, die für sich abgeschlossen sind.

Es existieren (bzw. existierten) verschiedene Varianten der *Cyc*-Ontologie, die sich hauptsächlich hinsichtlich ihres Umfangs unterscheiden. Die vollständige *Cyc*-Ontologie ist nur nach Erwerb einer kommerziellen Lizenz zugänglich. Zu Forschungszwecken frei verfügbar wurde zudem über viele Jahre ein Großteil der Ontologie unter dem Namen *ResearchCyc* veröffentlicht; das Projekt *ResearchCyc* wurde jedoch Ende 2019 eingestellt. Die dritte Variante ist *OpenCyc*; unter diesem Namen wurde bis 2015 ein kleiner Ausschnitt der *Cyc*-Ontologie frei zugänglich gemacht. Zwar wird auch *OpenCyc* nicht mehr offiziell unterstützt; es existieren jedoch Versionen des letzten veröffentlichen Standes der Ontologie im *OWL*-Format (siehe Abschnitt 2.3.14).

2.3.16 Repräsentation von Wörtern als Vektoren

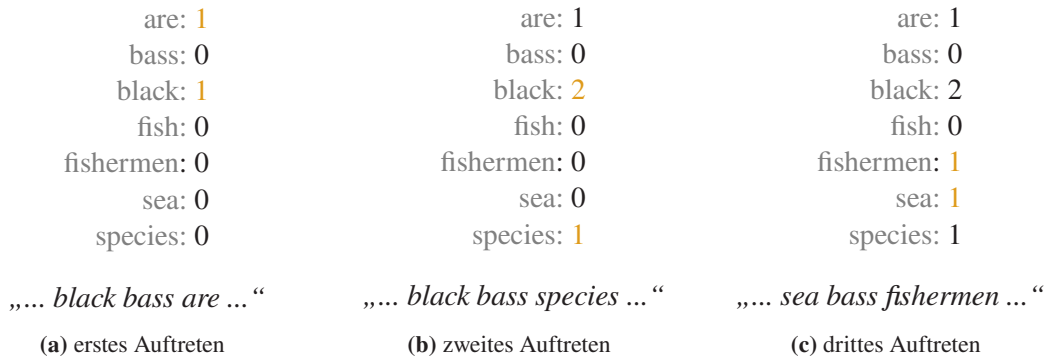
In der Computerlinguistik werden Wörter häufig als Vektoren dargestellt. Eine Wortvektorrepräsentation hat den Vorteil, dass Vektoren mithilfe von Vektordistanzmaßen verglichen werden können, um beispielsweise Ähnlichkeiten zwischen Wörtern zu bestimmen. Außerdem erwarten viele Verfahren des maschinellen Lernens die Darstellung der Eingabe als Eigenschaftsvektoren (siehe Abschnitt 2.2.1).

Die einfachste Variante stellt die Verwendung sogenannter *1-aus-n*-Vektoren (engl. *one hot vector*) dar [GBC16]. Hierzu wird ein Vektorraum aufgespannt, dessen Dimensionalität dem bekannten Vokabular (beispielsweise eines Korpus) entspricht. Jede Dimension repräsentiert genau ein Wort[64]; Wörter werden in diesem Modell repräsentiert, indem die entsprechende Dimension auf eins gesetzt wird, während alle anderen null sind. Diese Darstellungsform wird im Kontext des maschinellen Lernens häufig zur Darstellung von Elementen beliebiger Mengen verwendet, beispielsweise auch zur Darstellung der Ausgabeklassen bei Mehrfachklassen-Klassifikationsproblemen (siehe Abschnitt 2.2.1). Für die Repräsentation von Wörtern birgt dieses einfache Modell allerdings erhebliche Nachteile. Zum einen sind die Vektoren sehr dünn besetzt (je nur genau eine Dimension), wodurch Lernverfahren und Vektormaße weniger gut angewendet werden können (siehe Abschnitt 2.2.1). Außerdem wird nur das Wort isoliert von seiner Umgebung repräsentiert. Häufig kann die Bedeutung eines Wortes aber nur über den Kontext erschlossen werden (siehe Abschnitt 2.3.8).

Daher repräsentiert eine zweite Darstellungsform Wörter anhand ihrer Wortumgebung: die sogenannten *Bag-of-Words*-Vektoren, kurz *BoW* [JM09b]. Auch zur Erzeugung von *BoW*-Vektoren wird zunächst ein Vektorraum aufgespannt, dessen Dimensionalität dem Umfang des Vokabulars entspricht[65]. Jede Dimension repräsentiert wiederum ein Wort. Die Vektoren zur Darstellung einzelner Wörter geben Auskunft darüber, von welchen anderen Wörtern das betrachtete Wort häufig umgeben wird. Hierzu muss festgelegt werden, wie groß die betrachtete Umgebung (links und rechts) der Wörter sein soll. Außerdem muss die Art der Vektoreinträge definiert werden.

[64] Zur Optimierung des Verfahrens können zunächst die Lemmata der Wörter bestimmt werden (siehe Abschnitt 2.3.4.4).

[65] Je nach Einsatzgebiet sind nur bestimmte Umgebungswörter relevant. In diesem Fall begrenzt sich die Dimension des Vektorraums auf den Umfang der zu betrachtenden Wörter.



(a) erstes Auftreten **(b)** zweites Auftreten **(c)** drittes Auftreten

Abbildung 2.17: Schrittweiser Aufbau eines *BoW*-Vektors für das Wort *bass* anhand von drei beispielhaften Auftreten in einem Korpus.

Beispielsweise können Vorkommen in der Umgebung binär kodiert werden, gezählt oder anteilig angegeben werden. Zur Erzeugung von *BoW*-Vektoren müssen konkrete Auftreten von Wörtern und die jeweiligen Umgebungen anhand eines Korpus bestimmt werden. Abbildung 2.17 zeigt diesen Vorgang anhand eines Beispiels. *BoW*-Vektoren ermöglichen die Darstellung von Wörtern anhand ihres Kontextes. Allerdings sind auch diese immer noch verhältnismäßig dünn besetzt. Außerdem sind die Vektorräume beider bisher betrachteter Verfahren hochdimensional, wodurch sie für die meisten Verfahren des maschinellen Lernens unbrauchbar sind (siehe Abschnitt 2.2.1).

Eine bessere Alternative stellen sogenannte *Worteinbettungen* (engl. *word embeddings*) dar. Auch diese repräsentieren Wörter anhand der umgebenden Wörter. Die Anzahl der Dimensionen der Vektoren ist jedoch unabhängig vom Vokabular und deutlich geringer; häufig werden 150 bis 300 Dimensionen definiert. Außerdem sind die Vektoren wesentlich dichter besetzt. Zur Erzeugung der Vektoren wird ein Trick angewandt, der prinzipiell folgendermaßen funktioniert. Für ein vorwärts gerichtetes neuronales Netz (siehe Abschnitt 2.2.2) wird eine Pseudo-Aufgabe trainiert, die sich auf Wortumgebungen bezieht, beispielsweise die Vorhersage des nächsten Wortes bei Eingabe eines beliebigen Wortes. Das Netz verfügt über mindestens eine verdeckte Schicht, deren Neuronenanzahl der Dimension der zu erzeugenden Einbettungsvektoren entspricht. Das so konfigurierte Netz wird anschließend auf einem möglichst großen Korpus trainiert[66]. Da die Aufgabe darin besteht, das nächste Wort eines Textes vorherzusagen, müssen keine Ausgabebeispiele bereitgestellt werden[67] (siehe Abschnitt 2.2.1). Durch das Training anhand dieser Pseudo-Aussage erlernt das Netz implizit Wortkontexte. Ist das Training abgeschlossen, bilden die Kantengewichte der verdeckten Schicht die Worteinbettungsvektoren. Um einen konkreten Vektor zu erhalten, wird dem Netz das entsprechende Wort präsentiert; die sogenannte Netzantwort (das heißt die Werte der Aktivierungsfunktionen der Neuronen der verdeckten Schicht) wird dann als Worteinbettungsvektor verwendet. Die entstandenen Vektoren repräsentieren das Wort anhand beobachteter Kontexte; die Vektoren können auch abseits von künstlichen neuronalen Netzen angewandt werden, beispielsweise können Vektordistanzmaße zum Vergleich der Vektoren benutzt werden. Allerdings können Worteinbettungen

[66] Die Wörter werden dabei als *1-aus-n*-Vektoren repräsentiert.

[67] Derartige Klassifikationsprobleme sind eine Sonderform der überwachten Klassifikationsprobleme und werden als *selbstüberwacht* (engl. *self-supervised*) bezeichnet.

nur für Wörter erzeugt werden, die im Trainingsdatensatz aufgetreten sind; unbekannte Wörter werden (zumeist) auf den Null-Vektor abgebildet. Es existieren unterschiedlich Varianten zur Erzeugung von Worteinbettungen, die sich in technischen Details unterscheiden, etwa in der Definition der Pseudo-Aufgabe oder in der Anzahl der verdeckten Schichten. Bekannte Ansätze sind *Word2Vec* [Mik+13a; Mik+13b], *GloVe* [PSM14] und die *Word2Vec*-Weiterentwicklung *fastText*[68] [Boj+17; Jou+17]. Von den Entwicklern der jeweiligen Verfahren werden vortrainierte Vektoren, die nach den genannten Verfahren erzeugt wurden, für unterschiedliche Sprachen, verschiedene Dimensionen und trainiert auf unterschiedlichen Korpora, angeboten.

2.3.17 Vortrainierte Sprachmodelle

Vortrainierte Sprachmodelle (engl. *pre-trained language model*) sind Modelle, die mit Verfahren des maschinellen Lernens – zumeist unter Verwendung tiefer neuronaler Netze – auf umfangreichen Textkorpora trainiert wurden. Die Modelle bieten ein tiefes syntaktisches und semantisches Verständnis natürlicher Sprache. Darauf aufbauend können prinzipiell beliebige Problemstellungen der Computerlinguistik gelöst werden (sofern sich die Aufgabe auf ein Klassifikationsproblem abbilden lässt). Hierzu wird das vortrainierte Sprachmodell als Basis für ein neuronales Netz verwendet. An die letzte Schicht des Sprachmodells anschließend werden weitere Schichten angefügt, welche das eigentliche Problem lösen[69]. Das auf diese Weise kombinierte Netz muss zwar auch trainiert werden, es werden aber deutlich weniger Trainingsinstanzen und -durchläufe (das heißt Epochen) benötigt als wenn ein neuronales Netz, das eine ähnliche Klassifikationsgüte erzielt, von Grund auf trainiert werden würde. Das vortrainierte Sprachmodell kann zudem für unterschiedliche Problemstellungen wiederverwendet werden.

Die grundlegende Herangehensweise zur Erzeugung von Sprachmodellen ist dieselbe wie bei Worteinbettungen (siehe Abschnitt 2.3.16); ein neuronales Netz wird anhand einer Pseudo-Aufgabe trainiert. Dementsprechend können auch die Netze, die bei der Erzeugung von Worteinbettungen entstehen, als vortrainierte Sprachmodelle aufgefasst werden; in der Praxis werden sie auch ebenso verwendet. Der Grund für die Entwicklung komplexerer Sprachmodelle ist der wesentliche Nachteil von Worteinbettungen: für jedes Wort wird genau eine Worteinbettung erzeugt. Das bedeutet, bei mehrdeutigen Wörtern überlagern sich die Bedeutungen. Ist eine der Bedeutungen dominant (das heißt tritt sie im Sprachgebrauch deutlich häufiger auf als alle anderen) verdeckt sie die anderen Bedeutungen; gibt es hingegen gleichwertige Bedeutungen, repräsentiert der zugehörige Worteinbettungsvektor das Mittel aus diesen Bedeutungen und ist daher nahezu wertlos.

Um dieser Problematik entgegenzuwirken, sieht eine Weiterentwicklung von Worteinbettungen vor, Bedeutungsrepräsentationen in Abhängigkeit der (aktuellen) Wortumgebung zu erstellen. Zur Modellerzeugung verwenden derartige Ansätze rekurrente neuronale Netze (siehe Abschnitt 2.2.2). Der bekannteste Vertreter dieser Art ist *ELMo* (kurz für *Embeddings from Language Models*) [Pet+18]. Zur Erzeugung des *ELMo*-Modells wurde ein bidirektionales *LSTM* verwendet (siehe Abschnitt 2.2.2);

[68] *fastText* ist in der Lage Teile von unbekannten Wörtern auf Teilwortvektoren abzubilden.

[69] Die Kombination von vortrainierten Sprachmodellen mit spezialisierten Schichten zur Lösung eines konkreten Gebiets wird dem Konzept des *übertragenden Lernens* (engl. *transfer learning*) zugeordnet [GBC16].

dadurch können beidseitige Wortkontexte erfasst werden. *ELMo* kann als vortrainiertes Sprachmodell verwendet werden; alternativ werden auch Worteinbettungsvektoren bereitgestellt.

Moderne Verfahren zur Erzeugung von vortrainierten Sprachmodellen legen den Fokus auf die Ausweitung des Kontextverständnisses. Dazu werden vor allem sehr tiefe neuronale Netze erstellt, das heißt es werden sehr viele verdeckte Schichten genutzt. Außerdem werden auch neuartige Netzarchitekturen bzw. Techniken verwendet. Allen voran wird heutzutage die sogenannte *Transformer*-Architektur genutzt [Vas+17]. Der größte Vorteil dieses Ansatzes ist der sogenannte *Attention*-Mechanismus; dieser erlaubt es dem Netz zu erlernen, welcher Teil der Eingabe für die Lösung der Problemstellung relevant ist. Besonders effektiv funktioniert dies, wenn mehrere *Transformer* zusammengeschaltet werden. Hierzu werden *Transformer* wiederum in Schichten angeordnet. Ein weiterer Vorteil der *Transformer*-Architektur ist, dass keine rekurrenten neuronalen Netze verwendet werden müssen, um Wortumgebungen zu erfassen; es genügen einfache vorwärts gerichtete Netze. Dies hat wiederum den Vorteil, dass große Textmengen am Stück verarbeitet werden können (und nicht sequenziell eingelesen werden). In der Folge können Zusammenhänge zwischen Wörtern und Phrasen in einem auf *Transformern* basierenden Sprachmodell unabhängig von der (Wort-)Reihenfolge erkannt werden. Ein inzwischen weit verbreitetes vortrainiertes Sprachmodell, das auf Grundlage der *Transformer*-Architektur erzeugt wurde, ist das von *Google Research* erzeugte *BERT* (kurz für *Bidirectional Encoder Representations from Transformers*) [Dev+19]. Für *BERT* wurden *Transformer* bidirektional verknüpft und in 24 Schichten angeordnet. Diese Netzarchitektur wurde anschließend auf einem Korpus trainiert, das aus der vollständigen englischsprachigen Wikipedia (ca. 2,5 Milliarden Wörter) und zusätzlich aus 11038 Büchern (ca. 800 Millionen Wörter) besteht. Die Pseudo-Aufgabe besteht darin, ausgeblendete Textpassagen vorherzusagen; hierzu wurden 15% der Eingabe ausgeblendet. Neben dem vollständigen Modell ($BERT_{LARGE}$) gibt es auch eine reduzierte Variante, die nur aus zwölf *Transformer*-Schichten besteht ($BERT_{BASE}$). Beide Modelle ermöglichen die semantische Interpretation natürlicher Sprache. Daher basieren viele Lösungen zu Aufgabenstellungen der Computerlinguistik, die den Stand der Technik darstellen, inzwischen auf einem *BERT*-Modell. Der wesentliche Nachteil der Modelle ist ihr Speicherbedarf. Um $BERT_{BASE}$ zu verwenden, wird eine Grafikkarte mit mindestens 12 GB Arbeitsspeicher benötigt. Daher werden inzwischen weitere reduzierte Modelle angeboten, die zwar ein weniger tiefes, aber je nach Anwendungsgebiet ausreichendes Sprachverständnis ermöglichen [Tur+19]. Inzwischen bieten andere Ansätze noch größere Sprachmodelle, die eine noch weitreichendere semantische Interpretation der Sprache ermöglichen sollen, wie beispielsweise *XLNet* [Yan+20] oder *GPT-3* (kurz für *Generative Pre-trained Transformer 3*) [Bro+20].

2.4 Evaluationsmetriken

Die im Folgenden beschriebenen Metriken dienen der Bemessung der Qualität einer durch ein Verfahren erzeugten Ausgabe, wie es zur Evaluation üblich ist. Ein gebräuchliches Vorgehen zur Qualitätsmessung ist es, Ausgaben als Klassifikation aufzufassen und diese mit einem Goldstandard zu vergleichen. Das bedeutet, für ein Eingabedatum werden die erwarteten Ausgaben festgelegt. Diese können anschließenden mit der Ausgabe des Verfahrens verglichen werden. Handelt es sich um

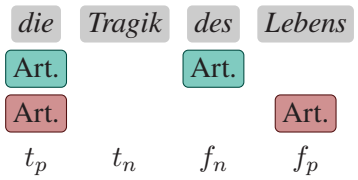


Abbildung 2.18: Bewertung von Ausgaben eines beispielhaften Klassifikators, der Artikel in Texten erkennen soll: Die erste Zeile zeigt die zu klassifizierenden Einheiten (Text). In der zweiten Zeile sind grün hinterlegt die erwarteten Klassifikationen dargestellt; die dritte zeigt rot hinterlegt die durch den Klassifikator emittierten Klassifikationen. In der vierten Zeile sind die Bewertungen der Klassifikationen angegeben.

ein echtes Klassifikationsproblem – wie es in dieser Arbeit bei den meisten betrachteten Verfahren der Fall ist – können die erzeugten Klassifikationen bestimmten Teilen der Eingabe (sogenannten Einheiten) zugeordnet werden. Beispielsweise entsprechen die zu klassifizierenden Einheiten bei der Erkennung von Wortarten (siehe Abschnitt 2.3.4.3) den Token und die emittierten Wortarten den Klassifikationsergebnissen, die wiederum mit einem Goldstandard, wie dem *WSJ Corpus*, verglichen werden können (siehe Abschnitt 2.3.3). Beim Vergleich von Goldstandard und erzeugter Ausgabe, können folgende vier Fälle auftreten:

- *richtig positiv*: Als richtig positives Ergebnis (engl. *true positive*, kurz t_p) bezeichnet man den Ausgang, bei dem eine zu klassifizierende Einheit durch das Verfahren die erwartete Klassifikation erhalten hat, das heißt richtig klassifiziert wurde.
- *richtig negativ*: Als richtig negatives Ergebnis (engl. *true negative*, kurz t_n) bezeichnet man den Ausgang bei dem eine zu klassifizierende Einheit laut Goldstandard *nicht* klassifiziert werden sollte und auch durch das Verfahren keine Klassifikation erfahren hat.
- *falsch positiv*: Als falsch positives Ergebnis (engl. *false positive*, kurz f_p) bezeichnet man den Ausgang bei dem eine zu klassifizierende Einheit laut Goldstandard *nicht* klassifiziert werden sollte, aber durch das Verfahren fälschlicherweise klassifiziert wurde.
- *falsch negativ*: Als falsch negatives Ergebnis (engl. *false negative*, kurz f_n) bezeichnet man den Ausgang bei dem eine zu klassifizierende Einheit laut Goldstandard klassifiziert werden sollte, aber durch das Verfahren nicht klassifiziert wurde.

Abbildung 2.18 zeigt die Klassifikatonsergebnisse eines beispielhaften Klassifikators; der Vergleich mit den erwarteten Ergebnissen führt jeweils zu einer Zuordnung zu den zuvor beschriebenen Fällen. Diese Art der Bewertung von Klassifikationsergebnissen ist nur für binäre Klassifikationsprobleme zulässig. Soll die Fallunterscheidung für Mehrfachklassenprobleme angewendete werden, müssen die Fälle je Klasse bestimmt werden. Auch die im Folgenden beschriebenen Metriken müssen je Klasse berechnet werden, können anschließend jedoch gemittelt werden, um eine Gesamtbewertung zu erhalten.Nachfolgend werden zunächst Metriken zur Bewertung von Klassifikationensergebnissen beschrieben, welche die zuvor beschriebenen Fälle zugrunde legen: Ausbeute, Präzision F_β-Maß und Genauigkeit. Anschließend werden mit der Wortfehlerrate, *BLEU* und der Übersetzungsfehlerrate Metriken beschrieben, die zur Bewertung von Verfahren zur automatischen Spracherkennung verwendet werden (siehe Abschnitt 2.3.7). Zuletzt wird mit dem Fleiss-Kappa ein Maß zur Bestimmung der Übereinstimmung von Bewertungen vorgestellt.

2.4.1 Präzision

Die Präzisionen gibt den Anteil der korrekten Klassifikationen eines Klassifikators gegenüber allen getätigten Klassifikationen an. Das bedeutet, neben den korrekten Klassifikationen, gehen auch fälschlich getätigte Klassifikationen in die Normierung ein:

$$Präzision = \frac{t_p}{t_p + f_p} \tag{2.10}$$

Ein Klassifikator erzielt demnach hohe Präzisionswerte, wenn nur wenige Fehlklassifikationen (das heißt zusätzliche, fehlerhafte Klassifikationen) stattfinden.

2.4.2 Ausbeute

Die Ausbeute gibt den Anteil der korrekten Klassifikationen eines Klassifikators gegenüber den erwarteten Klassifikationen (eines Goldstandards) an. Das bedeutet, neben den korrekten Klassifikationen, gehen auch die fehlenden Klassifikationen (das heißt nicht gefundene Klassifikationen des Goldstandards) in die Normierung ein:

$$Ausbeute = \frac{t_p}{t_p + f_n} \tag{2.11}$$

Ein Klassifikator erzielt demnach hohe Ausbeutewerte, wenn möglichst viele der erwarteten Klassifikationen auch durch den Klassifikator klassifiziert werden.

2.4.3 F-Maße

Das F_β-Maß kombiniert die Bemessung von Ausbeute und Präzision; β kann variiert werden, um die Gewichtung von Ausbeute und Präzision bei der Berechnung anzupassen. Das allgemeine F_β-Maß ist wie folgt definiert:

$$F_\beta = (1 + \beta^2) \cdot \frac{Präzision \cdot Ausbeute}{(\beta^2 \cdot Präzision) + Ausbeute} = \frac{(1 + \beta^2) \cdot t_p}{(1 + \beta^2) \cdot t_p + \beta^2 \cdot f_n + f_p} \tag{2.12}$$

Wird β gleich 1 gesetzt, entspricht das F_β-Maß dem harmonischen Mittel von Ausbeute und Präzision:

$$F_1 = 2 \cdot \frac{Präzision \cdot Ausbeute}{Präzision + Ausbeute} = \frac{t_p}{t_p + \frac{1}{2}(f_p + f_n)} \tag{2.13}$$

Wird für β ein Wert kleiner 1 gewählt, wird die Präzision höhere gewichtet. Das bedeutet, der Anteil des Präzisionswert am Ergebnis steigt. Das $F_{0,5}$-Maß gewichtet beispielsweise die Präzision doppelt so hoch wie die Ausbeute:

$$F_{0,5} = 1,25 \cdot \frac{Präzision \cdot Ausbeute}{(0,25 \cdot Präzision) + Ausbeute} = \frac{1,25 \cdot t_p}{1,25 \cdot t_p + 0,25 \cdot f_n + f_p} \tag{2.14}$$

Für Werte von β, die größer als 1 sind, tritt der gegenteilige Effekt ein. Dann wird die Ausbeute höhere gewichtet, im Fall des F_2-Maßes doppelt so hoch:

$$F_2 = 5 \cdot \frac{Präzision \cdot Ausbeute}{(4 \cdot Präzision) + Ausbeute} = \frac{5 \cdot t_p}{5 \cdot t_p + 4 \cdot f_n + f_p} \tag{2.15}$$

2.4.4 Genauigkeit

Die Genauigkeit gibt an, wie korrekt ein Klassifikator insgesamt klassifiziert. Im Gegensatz zu Präzision, Ausbeute und F_β-Maß bezieht die Genauigkeit auch die korrekterweise nicht klassifizierten Einheiten (*true negatives*) mit ein:

$$Genauigkeit = \frac{t_p + t_n}{t_p + t_n + f_p + f_n} \tag{2.16}$$

Die Genauigkeit bietet grundsätzlich die präziseste Bewertungsmöglichkeit von Klassifikatoren. Bei Klassifikationsproblemen, welche die Klassifikation von nur wenigen Einheiten erfordern, ist der Anteil der (erwarteten) richtig negativen Klassifikationen überproportional groß. Dadurch ist der Genauigkeitswert für diese Probleme sehr hoch (und überschätzt die Qualität der Klassifikatoren). Daher wird im Allgemeinen das F_β-Maß zur Bewertung von Klassifikatoren bevorzugt.

Die Genauigkeit eignet sich allerdings zur Bewertung von Klassifikatoren bei Problemstellungen, welche die Klassifikation aller (oder nahezu aller) Einheiten erfordern, wie beispielsweise der Wortarterkennung (siehe Abschnitt 2.3.4.3).

2.4.5 Falsch-Positiv-Rate

Die Falsch-Positiv-Rate (kurz f_p-Rate) gibt den Anteil der fehlerhaften (bzw. überzähligen) Klassifikationen eines Klassifikators gegenüber allen *nicht* zu klassifizierenden Einheiten (eines Goldstandards) an. Das bedeutet, neben den Fehlklassifikationen (f_p) gehen auch korrekterweise nicht durchgeführten Klassifikationen (t_n) in die Normierung ein:

$$Falsch\text{-}Positiv\text{-}Rate = \frac{f_p}{f_p + t_n} \tag{2.17}$$

Im Gegensatz zu den zuvor eingeführten Metriken sollte die Falsch-Positiv-Rate eines Klassifikators möglichst gering sein. Ein Klassifikator erzielt niedrige Werte, wenn möglichst wenige Einheiten fälschlicherweise einer Klasse zugeordnet wurden.

2.4.6 Wortfehlerrate

Die Wortfehlerrate (engl. *word error rate*, kurz *WER*) wird zur Bewertung von Systemen zur automatischen Spracherkennung verwendet [Nie+00; JM09b]. Sie gibt den Anteil der fehlerhaften Wörter einer durch ein System erzeugten Hypothese gegenüber einer Referenztranskription an. Hierzu werden die Änderungen gezählt, die nötig sind, um die Hypothese c in die Referenztranskription r zu überführen. Mögliche Operationen sind Vertauschungen S, Entfernungen D und Hinzufügungen I von Wörtern (bzw. Token): Die Summe der Änderungen wird über die Länge (Wortanzahl) der Referenztranskription normiert:

$$WER(c,r) = \frac{S(c,r) + D(c,r) + I(c,r)}{|wörter(r)|} \tag{2.18}$$

Die Wortfehlerrate ist abgeleitet von der Levenshtein-Distanz zur Bestimmung von Unterschieden zweier Zeichenketten (siehe Abschnitt 2.3.13.1).

2.4.7 BLEU

BLEU (kurz für *bilingual evaluation understudy*) ist eigentlich ein Maß zur Bewertung von automatischen Übersetzungen [Pap+02]. Hierzu wird eine generierte Übersetzung mit einer Menge von (durch Menschen angefertigten) Referenzübersetzungen verglichen und Übereinstimmungen gezählt. Als Übereinstimmung können entweder einzelne Wörter gelten oder N-Gramme. *BLEU* ist somit eine Variante des Präzisionsmaßes (siehe Abschnitt 2.4.1), welches den Vergleich gegen mehreren Referenzen und die Betrachtung von N-Grammen ermöglicht. Formal wird das *BLEU*-Maß für eine generierte Übersetzung c in Abhängigkeit zur Menge der Referenzübersetzungen R wie folgt berechnet:

$$BLEU(c,R) = P_n(c,R) = \frac{\sum\limits_{\text{n-Gramm}\in c} \max\limits_{r_i \in R}(Anzahl_{begrenzt}(\text{n-Gramm}, r_i))}{|\text{n-Gramm}|_C} \tag{2.19}$$

Im Zähler wird je N-Gramm der generierten Übersetzungen das maximale Auftreten in einer der Referenzübersetzungen bestimmt; dieser Wert wird auf die Anzahl der Vorkommen des N-Gramms in der generierten Übersetzungen beschränkt. Die Werte je N-Gramm werden summiert. Normiert wird diese Summe über die Anzahl der in der generierten Übersetzung enthaltenen N-Gramme. Es zeigt sich, dass Übersetzungen, die für das *BLEU*-Maß bei Verwendung von Tetragrammen (*BLEU-4*) hohe Bewertungen erhalten, auch von menschlichen Bewertern als besonders passend angesehen werden [Pap+02]; daher wird zumeist *BLEU-4* verwendet.

Das *BLEU*-Maß wird auch zur Bewertung von Systemen zur automatischen Spracherkennung verwendet. Hierzu wird eine erzeugte Hypothese als automatische Übersetzung angesehen und händische Transkriptionen als Referenzübersetzungen.

2.4.8 Übersetzungsänderungsrate

Die Übersetzungsänderungsrate (engl. *translation edit rate*, kurz *TER*) ist ebenfalls ein Maß zur Bestimmung der Qualität automatischer Übersetzungen, welches auch zur Bewertung von Systemen zur automatischen Spracherkennung verwendet wird [Sno+06]. Die Übersetzungsänderungsrate eines Kandidaten c in Abhängigkeit einer Menge von Referenzkandidaten R wird wie folgt berechnet:

$$TER(c, R) = \frac{\min\limits_{r_i \in R}(e(c, r_i))}{\frac{1}{n} \cdot \sum\limits_{i=0, r_i \in R}^{n} |wörter(r_i)|} \tag{2.20}$$

Die Funktion $e(a, b)$ im Zähler bestimmt die minimal nötigen Änderungen, um den Kandidaten in eine der Referenzübersetzungen zu überführen. Die erlaubten Operationen zur Änderung sind, wie bei der Bestimmung der Wortfehlerrate (siehe Abschnitt 2.4.6), Vertauschungen, Entfernungen und Hinzufügungen; die Übersetzungsänderungsrate führt allerdings als weitere Operation die Verschiebung eines Wortes an eine beliebige andere Stelle ein. Der erhaltene Wert wird über das arithmetische Mittel der Längen der Referenzübersetzungen normiert.

Zur Bestimmung der erweiterten Übersetzungsänderungsrate (engl. *translation edit rate plus*, kurz *TERp*) [Sno+09] werden die Kosten der einzelnen Änderungsoperationen geändert; während bei *TER* alle Operationen den fixen Kostenwert 1 besitzen, verwenden die Autoren einen Optimierungsalgorithmus, um Kosten für einzelne Operationen festzulegen.

2.4.9 Fleiss-Kappa

Das Fleiss-Kappa ist ein Maß zur Bewertung der Inter-Annotator-Übereinstimmung [Fle71]; das bedeutet, es bestimmt, wie einig sich (menschliche) Bewerter bei der Vergabe von Etiketten, Beurteilungen oder Kategorien sind. Im Gegensatz zu Cohens Kappa [Coh60] kann das Kappa nach Fleiss zur Bewertung der Übereinstimmung von mehr als zwei Bewertern verwendet werden. Das Fleiss-Kappa κ ist wie folgt definiert:

$$\kappa = \frac{\sum\limits_{i}^{N} p_i - \frac{1}{z^{d-1}}}{1 - \frac{1}{z^{d-1}}} \tag{2.21}$$

N ist die Menge der zu bewertenden Einheiten, d die Anzahl der Bewerter und z die Anzahl der möglichen (Bewertungs-)Kategorien; p_i gibt die Übereinstimmung der Bewerter bei der Bewertung der i-ten Einheit an und wird wie folgt bestimmt:

$$p_i = \frac{1}{d(d-1)} \sum_{j=1}^{Z} (d_{ij}^2 - d_{ij}) \tag{2.22}$$

Dabei entspricht d_{ij} der Anzahl der Bewerter, die sich bei der Bewertung der i-ten Einheit für die Kategorie j aus der Menge der Kategorien Z entschieden haben.

3 Verwandte Arbeiten

„I suspect that machines to be programmed in our native tongues
– be it Dutch, English, American, French, German, or Swahili –
are as damned difficult to make as they would be to use.“

– Edsger W. Dijkstra

Dieses Kapitel betrachtet verwandte Arbeiten aus den Bereichen Programmierung mit natürlicher Sprache und Architekturen für Sprachverständnissysteme in diesem Kontext. Die Idee, Computersysteme mit natürlicher Sprache zu steuern und zu programmieren, ist keineswegs neu. Der Abschnitt 3.1 beleuchtet kurz die Historie dieses Forschungsgebiets. Bei modernen Systemen für das Programmieren mit natürlicher Sprache gilt es im Wesentlichen zwei Arten zu unterscheiden: Die erste Gruppe richtet sich an Software-Entwickler (Abschnitt 3.2); diese Systeme sind darauf ausgerichtet, Experten bei der Erstellung von Software zu unterstützen. Die zweite Gruppe richtet sich gezielt an Endnutzer mit keiner oder kaum Programmiererfahrung (Abschnitt 3.3); die Zielsetzung dieser Systeme ist, Laien in die Lage zu versetzen, Computersystemen komplexe Befehle zu erteilen beziehungsweise neue Funktionalität in Form von Skripten oder Makros anzulernen. Neben diesen beiden Hauptströmungen gibt es eine Reihe von Arbeiten aus angrenzenden Forschungsgebieten (Abschnitt 3.4). Hierzu zählen die Erzeugung automatischer Testfälle aus natürlichsprachlichen Beschreibungen, das Abfragen von Informationen aus Datenbanken mit natürlicher Sprache und virtuelle Assistenzsysteme. Zuletzt werden alle verwandten Arbeiten anhand der in Abschnitt 1.2 aufgestellten Zieldefinition mit ProNat verglichen (Abschnitt 3.5)[1].

3.1 Historie

Bereits 1966 fordert Jean Sammet, dass der Umgang mit Computersystemen für den Menschen möglichst natürlich gestaltet werden müsse, um die Produktivität zu maximieren [Sam66]. Für Sammet ist dies einzig durch die Verwendung natürlicher Sprache zu erreichen. Sie schlägt nicht nur vor, die Steuerung von Computersystemen mit Alltagssprache zu ermöglichen, sondern auch die Programmierung. Die Umsetzbarkeit von Sammets Idee wird jedoch schnell von Kritikern angezweifelt und teils sogar theoretisch widerlegt [Dij63; Dij64; Hil72; Dij79]. Die Kritiker scheinen zunächst im Recht, denn nur langsam entwickeln sich erste Systeme. Ein erster Durchbruch gelingt

[1] Die in diesem Kapitel diskutierten verwandten Arbeiten stellen eine Auswahl dar; die ausgewählten Publikationen sollen ein möglichst umfangreichen Gesamteindruck hinsichtlich aufgestellter Problemstellungen (bzw. Zielsetzungen), verwendeter Techniken, Einschränkungen (bzw. Übertragbarkeit) usw. bieten.

Terry Winograd mit *SHRLDU*, einem System zur Dialog-basierten Steuerung eines virtuellen Roboterarmes, der geometrische Figuren bewegt [Win72]. *SHRLDU* weist augenscheinlich ein tiefes Verständnis natürlicher Sprache auf. Dieses erreicht Winograd mit einer Ontologie-artigen Darstellung der Szenerie und einem umfangreichen Satz von Inferenzregeln. Weitere ähnliche Spezialsysteme entstehen, wie das von Ballard und Biermann entwickelte *NLC* zur sprachgesteuerten Tabellenkalkulation [BB79; BB80; BBS83]. Ihr System ermöglicht es, im Dialog mit dem Nutzer Funktionsfolgen zu erlernen, ähnlich den Makros, wie sie in modernen Tabellenkalkulationssoftware üblich sind. Auch Ballard und Biermann verwenden für *NLC* einen umfangreichen Regelsatz, den sie in Kombination mit einem spezialisierten Zerteiler einsetzen. Die grundsätzliche Möglichkeit, Computersysteme mit natürlicher Sprache, wenn auch zunächst rein textuell, zu steuern und auch zu programmieren, konnte also nachgewiesen werden. Allerdings wird deutlich, dass die Systeme nur in sehr eingeschränkten Domänen funktionieren und von einem generellen Sprachverständnis weit entfernt sind. Da dieses auch konzeptionell aufgrund der Verwendung komplexer Regelwerke und deren mangelnder Skalierbarkeit nicht umsetzbar erscheint, geht das Interesse an derartigen Systemen zunächst stark zurück. Erst durch theoretische Studien, wie denen von Pane et al. über die verwendete Sprache von Laien zur Lösung programmatischer Problemstellungen [PRM01; PM06] oder der Machbarkeitsstudie zur Programmierung mit natürlicher Sprache von Lieberman und Liu [LL06], erhält das Forschungsgebiet neue Impulse. Neue Systeme zur Unterstützung von Entwicklern (siehe Abschnitt 3.2), für die Entwicklung einfacher Programme durch Endnutzer mit natürlicher Sprache (siehe Abschnitt 3.3) und für ähnliche Aufgaben, wie die Erzeugung von automatischen Tests aus Beschreibungstexten oder Abfragen von Informationen aus Datenbanken über natürlichsprachliche Fragen (siehe Abschnitt 3.4) entstehen. Durch die rasante Weiterentwicklung der Rechnerlinguistik in den letzten Jahren und das zunehmende Interesse großer Technologieunternehmen wie *Apple*, *Microsoft*, *IBM* und *Google*, erfährt auch die Idee der Programmierung mit natürlicher Sprache wieder gesteigerte Aufmerksamkeit. So beschreibt eine der Vergleichsaufgaben des Jahrgangs 2017 der *Association for Computational Linguistics* ein System zur konversationellen Endnutzer-Programmierung von Makros für virtuelle Assistenten (*SemEval-Task 11, 2017* der *Association of Association for Computational Linguistics*) [SHF17][2]. Auch wenn diese Aufgabe nur einfache Funktionskombinationen vorsieht und die Domäne auf virtuelle Assistenten eingeschränkt ist, zeigt sie doch das Interesse der Forschungsgemeinde an dieser Zielsetzung.

3.2 Unterstützung von Entwicklern

Die erste Kategorie von Systemen für Programmierung mit natürlicher Sprache richtet sich an Software-Entwickler. Das Ziel dieser Systeme ist die aktive Unterstützung des Entwicklers beim Erstellen von Software, indem übliche Tätigkeiten über eine konversationelle Schnittstelle zugänglich gemacht werden. Die Unterstützung reicht von direkter Quelltexteingabe mittels Sprache über Entwurfsextraktion aus Textdokumenten bis hin zur Erzeugung von Programmschnipseln oder Makros. Systeme dieser Art unterliegen der Annahme, dass auch für Entwickler die natürliche

[2] SemEval-Task 11, 2017: `http://alt.qcri.org/semeval2017/task11/`, zuletzt besucht am 24.02.2021.

Sprache die einfachste Möglichkeit darstellt, Ideen auszudrücken und die Entwicklung vereinfacht oder beschleunigt werden kann. Die Systeme zur Unterstützung von Entwicklern können hinsichtlich ihrer Zielsetzung beziehungsweise ihres Einsatzgebietes in drei Gruppen untergliedert werden.

3.2.1 Quelltext-Diktat

Die erste Gruppe bilden Systeme zur Quelltexteingabe über eine Sprachschnittstelle. Diese erlauben es Entwicklern, verbalisierten Quelltext zu diktieren. Die Eingabe wird dabei nicht analysiert, sondern lediglich wortwörtlich übersetzt. Derartige Systeme verlassen sich insbesondere darauf, dass der Nutzer ausschließlich semantisch korrekten Quelltext diktiert. Das bedeutet, dass diese Systeme keine semantische Diskrepanz zwischen natürlicher Sprache und Programmiersprache überbrücken müssen. Stattdessen muss lediglich gewährleistet werden, dass die syntaktische Übersetzung korrekt vollzogen wird. Daher liegt der Fokus dieser Systeme auf der zuverlässigen Erkennung von Programmiersprachen-typischen Direktiven. Das bedeutet, die Systeme erwarten zum Beispiel nach der Angabe einer Bedingung (über ein *If-Statement*) auch die Beschreibung der zugehörigen Konsequenz (als *Then-Block*). Für Systeme, die gesprochene Diktate unterstützen, bedeutet diese Fokussierung unter anderem auch, dass der verwendete Spracherkenner (beziehungsweise dessen Modelle) angepasst werden, sodass zum Beispiel das Wort *throw* eher erkannt wird als das phonetisch ähnliche *though*. Neben der Eingabe von Quelltext werden meist auch einfache Befehle zur Navigation in der Entwicklungsumgebung unterstützt. Als Motivation für diese Art von Systemen gilt die intuitive Benutzung der natürlichen Sprache durch den Menschen, die es selbst erfahrenen Entwicklern erlauben soll, Programme einfacher und schneller zu erstellen. Das von Price et al. entwickelte Werkzeug *NaturalJava* ermöglicht die textuelle Eingabe von *Java*-Quelltext mittels natürlicher Sprache [Pri+00]. *NaturalJava* nutzt Techniken aus dem Forschungsgebiet der Informationsrückgewinnung (engl. *information retrieval*), um die Rollen von Kasus-Rahmen (engl. *case frames*) zu besetzen (siehe Abschnitt 2.3.4.8); vollständige Kasus-Rahmen werden anschließend in *Java*-Quelltext überführt. Das Einsprechen von *Java*-Quelltext erlaubt *Spoken Java* von Begel und Graham [Beg04; BG05]. Laut der Autoren ist *Spoken Java* für Entwickler gedacht, die ihre Hände temporär oder dauerhaft nicht für die Quelltexteingabe mittels Tastatur verwenden können, zum Beispiel infolge einer Sehnenscheidenentzündung. *VoiceCode*, entwickelt von Désilets et al., unterstützt gesprochene Quelltexteingaben für beliebige Programmiersprachen [DFN06]. Keine der Publikationen zu den drei Werkzeugen beinhaltet eine Evaluation; stattdessen wird jeweils anhand beispielhafter Eingaben die Funktionsweise demonstriert. Die Beispiele sind aber nicht umfangreich genug, um die Fähigkeiten und Grenzen des jeweiligen Ansatzes abschätzen zu können.

3.2.2 Unterstützung des Programm-Entwurfs

Systeme der zweiten Gruppe assistieren dem Entwickler beim Programm-Entwurf. Im Gegensatz zu den Systemen zur direkten Quelltexteingabe erlauben diese freie und natürliche Formulierungen. Der Nutzer übergibt dem System einen deskriptiven Text, meist in Form einer Geschichte, wie zum Beispiel eine Anwendererzählung (engl. *user story*). Das System erzeugt daraus Klassen-Skelette und

Methodensignaturen und gegebenenfalls Attribute. Die erzeugten Skelette enthalten jedoch keine Funktionalität und sind nicht ausführbar. Vielmehr sollen sie als Ausgangspunkt für den Entwurf und die anschließende Implementierung dienen. Das von Liu und Lieberman vorgestellte *Metafor* erstellt Klassen-Skelette, Methodensignaturen und Attribute aus Prosatexten [LL05a; LL05b]. *Metafor* verwendet hierfür einen angepassten Zerteiler, der natürlichsprachliche Sätze in sogenannte *SVOO*-Strukturen überführt. Die *SVOO*-Strukturen dienen als abstrakte Darstellung einzelner im Satz enthaltener Handlungen, bestehend aus einem Subjekt, einem Prädikat und einem direkten sowie einem indirekten Objekt. Einzelne *SVOO*-Strukturen werden anschließend regelbasiert in Programmkonstrukte überführt. Die verwendeten Regeln orientieren sich dabei am Abbott'schen Verfahren [Abb83]; das bedeutet, dass unter anderem Subjekte und Objekte in Klassen und Verben in Methoden überführt werden. Die Ähnlichkeit der Übersetzungsregeln zu den von Abbott entwickelten unterstreichen die Zielsetzung, den Entwickler lediglich beim Entwurf unterstützen zu wollen. Sowohl die angepassten Grammatiken als auch die Übersetzungsregeln beruhen auf einer systematischen Untersuchung der englischen Sprache auf ihre Eignung, programmatische Semantiken auszudrücken [LL04]. Die Nützlichkeit von *Metafor* wurde in einer Nutzerstudie evaluiert: Dreizehn Studenten, davon sieben mit mittlerer und sechs mit geringer Programmiererfahrung, sollten kleine Programme entwickeln. Dabei sollten sie zum einen frei entwerfen (das heißt mit Stift und Papier) und zum anderen *Metafor* beim Entwurf zu Hilfe nehmen[3]. Die Auswertung ergab, dass vor allem Programmieranfänger von *Metafor* profitieren; alle Probanden gaben aber an, die Unterstützung durch *Metafor* gegenüber dem freien Entwurf zu bevorzugen. Mihalcea et al. präsentieren mit *NLP for NLP* eine Weiterentwicklung von *Metafor* [MLL06]: Diese kann einerseits zusätzlich wiederholende Aktionen erkennen und daraus Schleifen synthetisieren und andererseits Aussagen deuten, die lediglich als Kommentare dienen (und entsprechend in Programm-Kommentare überführen). Außerdem enthält die Publikation eine quantitative Untersuchung der Sprachverständnisfähigkeiten des (erweiterten) *Metafor*-Systems: Anhand von 25 Prosatexten, die aus Aufgabenbeschreibungen für Programmieranfänger entnommen wurden, wurde die Korrektheit der Anweisungs- und Schleifensynthese analysiert. Die Autoren ermittelten für die Generierung von Anweisungen eine Präzision von 0,860 bei einer Ausbeute von 0,754. Schleifen werden weniger treffsicher erzeugt; die erzielte Präzision beträgt aber immerhin 0,806 und die Ausbeute 0,714. Diese Ergebnisse unterstreichen, dass das *Metafor*-System einen vielversprechenden Ansatz zur Erzeugung von Programm-Skeletten darstellt; allerdings ist ungewiss, ob auf Basis des Ansatzes (durch entsprechende Weiterentwicklungen) auch ausführbarer Quelltext generiert werden könnte.

3.2.3 Synthese von Programmschnipseln und Makros

Die dritte Gruppe von Systemen liefert dem Entwickler aus natürlichsprachlichen Anfragen synthetisierte Programmschnipsel, Makros oder ähnliches. Im Gegensatz zu den zuvor diskutierten Ansätzen ist die Idee hinter diesen Systemen, die eigentlichen Arbeitsabläufe des Entwicklers

[3] Bei der Formulierung der Prosatexte zur Eingabe in *Metafor* wurden die Probanden durch den Experimentator unterstützt; das bedeutet, der Experimentator änderte einzelne Formulierungen, wenn diese von den *SVOO*-Strukturen nicht erfasst werden können. Eine Bewertung der Fähigkeit von *Metafor*, natürliche Sprache zu interpretieren, kann daher nicht stattfinden.

unberührt zu lassen. Stattdessen soll nur gezielt bei der Umsetzung (zumeist kleiner) Teilschritte unterstützend eingegriffen werden. Hierbei wird angenommen, dass Entwickler Ideen für Makros, neue Funktionen oder Programmabschnitte vor der eigentlichen Umsetzung zunächst verbalisieren. Diese Systeme werden unter der Arbeitshypothese entwickelt, dass die passenden Programmschnipsel auch direkt aus der natürlichsprachlichen Beschreibung synthetisiert werden können. Die Synthese ist vergleichsweise einfach und zeitsparend gegenüber der Einarbeitung eines Entwicklers in eine API und zudem hinreichend korrekt, zumal das Ergebnis vom Entwickler überprüft werden kann. Die meisten dieser Systeme erlauben die Verwendung alltäglicher Sprache und schränken die möglichen Formulierungen nicht oder nur geringfügig ein. Die Herausforderung, korrekte Funktionsaufrufe beziehungsweise Skripte zu synthetisieren, ist für die hier beschriebenen Systeme im Vergleich zu denen in Abschnitt 3.3 geringer, da die erzeugten Skripte vergleichsweise kurz sind und angenommen werden kann, dass von Entwicklern formulierte Anfragen exakter formuliert werden als von Laien.

Gulwani und Marron präsentieren mit *NLyze* ein System, das erfahrene Nutzer dabei unterstützt, Makros für die Tabellenkalulationssoftware *Microsoft Excel* zu erstellen [GM14]. Der Nutzer kann die gewünschte Aktionsfolge in alltäglicher Sprache (in Textform) beschreiben und sich dabei auch auf Tabelleninhalte oder Spalten- und Zeilenbezeichner beziehen. In einer umfangreichen Evaluation konnte gezeigt werden, dass die Generierung von Makros durchgängig in weniger als 20 Millisekunden gelingt; das System ist somit echtzeitfähig. In einer Nutzerstudie gelang es dem System außerdem in 90% der Fälle, die korrekte Aktionsfolge für eine natürlichsprachliche Funktionsbeschreibung zu erzeugen. Die gewählten Aufgaben waren aber vergleichsweise einfach; außerdem wurde den Probanden die Funktionsweise des Systems zunächst anhand einer Reihe von Beispielen demonstriert, sodass eine Beeinflussung (der Formulierung der Beschreibungen) der Probanden nicht auszuschließen ist.

Andere Ansätze finden passende API-Funktionen zu natürlichsprachlichen Anfragen von Entwicklern [GK15; RK17]. Dabei werden Genauigkeiten zwischen 21% und 45% erzielt (je nach API und bei ausschließlicher Betrachtung der höchstbewerteten Vorschläge). In den genannten Publikationen werden die Ansätze allerdings nur anhand von synthetischen Beispielen evaluiert, die von den jeweiligen Autoren erstellt wurden. Weiterführenden Ansätzen, wie dem von Raghothaman entwickelten *SWIM*, gelingt es, zusammenhängende Programmschnipsel zu erzeugen [RWH16]. In einer Evaluation anhand von (echten) Suchmaschinenanfragen konnten die Autoren zeigen, dass in 44% der Fälle unter den fünf von *SWIM* am höchsten bewerteten Vorschlägen der zur Anfrage passende Programmschnipsel enthalten ist[4]. Gu et al. verwenden für dieselbe Zielsetzung rekurrente neuronale Netze [Gu+16]. In einer vergleichenden Evaluation anhand des *SWIM*-Datensatzes zeigen die Autoren, dass es mit diesem Ansatz gelingt, die Trefferquote unter den fünf am höchsten bewerteten Vorschlägen auf 50% zu steigern.

Das Ziel, den Umgang mit Programmiersprachen natürlicher zu gestalten und insbesondere Programmieranfänger den Einstieg zu erleichtern, verfolgen Wang et al. [Wan+17]. Ihr System *Voxelrun* lernt interaktiv neue Aliase für bestehende API-Funktionen und ermöglicht auch die Komposition von einfachen Funktionssequenzen. Nutzer können in einer virtuellen 3D-Welt Objekte (aus vorhandenen

[4] Das bedeutet, *SWIM* erzielt in der Evaluation eine *Ausbeute@5* von 0,440.

Bausteinen) erstellen und Abläufe beschreiben. Zur Beschreibung wird eine Programmiersprache verwendet, die an die natürliche (englische) Sprache angelehnt ist (Beispiel: *while not has color red [select left of this]*). Verwendet ein Nutzer eine dem System unbekannte Funktion, wird er aufgefordert die erwartete Funktionalität anhand von Grundfunktionen zu beschreiben; auf diese Weise werden kontinuierlich neue (abstraktere) Funktionen erstellt. In einem Experiment, das über eine Schwarmauslagerungsplattform (engl. *crowd-sourcing plattform*) mit 45 Probanden durchgeführt wurde, konnten die Autoren nachweisen, dass *Voxelrun* wie erwartet funktioniert. Insgesamt gaben die Probanden *Voxelrun* über 100000 Anweisungen. Während das System zu Beginn 40% der Anweisung nicht verarbeiten konnte, sank der Anteil zum Ende des Experiments auf unter 11%. Das bedeutet, das System konnte das eigene Vokabular stetig erweitern; zum Ende des Experiments waren knapp 70% der Anweisungen der Probanden an das System zuvor erlernte. *Voxelrun* kann allerdings nur zur Programmierung der virtuellen 3D-Welt verwendet werden; inwieweit das Konzept auf andere Anwendungen übertragbar ist, wird von den Autoren nicht diskutiert. Der wesentliche Nachteil von *Voxelrun* ist, dass die zu verwendende Sprache einer Programmiersprache recht ähnlich ist und daher von Laien nicht ohne vorheriges Erlernen der Sprache verwendet werden kann.

3.3 Endnutzer-Programmierung mit natürlicher Sprache

Systeme für Endnutzer-Programmierung mit natürlicher Sprache sollen Laien ohne oder mit geringer Programmiererfahrung befähigen, einfache Programme zu entwickeln. Der Nutzer soll dabei seine Kreativität möglichst frei entfalten können. Das bedeutet insbesondere auch, dass alltägliche Sprache verwendet werden und jedes beliebige Zielsystem in beliebiger Systemumgebung (gemeinsam nachfolgend als Domäne bezeichnet) programmiert werden kann. In der Realität schränken nahezu alle bekannten Systeme entweder die Sprache oder die Domäne ein. Beides geschieht, um die Komplexität der Quelltext-Synthese zu begrenzen. Für die Erzeugung der Skripte aus natürlichsprachlichen Beschreibungen wird meist davon ausgegangen, dass das anzusprechende Zielsystem bereits grundlegende Funktionalitäten zur Verfügung stellt, zum Beispiel in Form einer API. Die Quelltext-Synthese besteht dadurch im Wesentlichen aus einer geeigneten Komposition von Grundfunktionalitäten. Gegebenenfalls müssen dem Skript noch Kontrollstrukturen, Variablen und ähnliches hinzugefügt werden.

3.3.1 Syntax- und wissensbasierte Ansätze

Interessante Einsichten zu den Herausforderungen des Programmierens mit natürlicher Sprache präsentieren Knöll und Mezzini bei der Beschreibung der Vision eines Systems namens *Pegasus* [KM06]. Die Autoren argumentieren, dass es eine semantische Lücke zwischen natürlicher Sprache und bekannten Programmiersprachen gibt, die geschlossen werden muss, um zuverlässig mit natürlicher Sprache programmieren zu können. Diese Lücke entsteht vor allem dadurch, dass in Programmiersprachen alles explizit ausgedrückt wird, wohingegen natürliche Sprache viele implizite

Annahmen trifft. Diese reichen von impliziten Referenzen bis hin zu Annahmen über das Wissen des Gesprächspartners. Ist der Gesprächspartner ein Computersystem, muss alles Implizite durch die konversationelle Schnittstelle expliziert werden. Der in *Pegasus* angestrebte Ansatz verwendet die sogenannte *Idea Notation*, eine Ontologie-artige Darstellung von Konzepten der realen Welt. Ob die *Idea Notation* tatsächlich die semantische Lücke zwischen natürlicher und Programmiersprache schließt, kann nicht beurteilt werden, da *Pegasus* nie umgesetzt wurde.

Little und Miller verfolgen hingegen einen stark simplifizierten Ansatz; sie synthetisieren erfolgreich einzelne Kommandos mithilfe einer einfachen Schlüsselwortsuche [LM06]. Beispielsweise kann mithilfe des Befehls „refresh“ die Aktualisierung einer Internetseite in einem Browser angestoßen werden. In einer Nutzerstudie mit neun Probanden (fünf mit und vier ohne Programmierkenntnisse) gelingt es ihrem System in den meisten Fällen, den korrekten Befehl zu einer natürlichsprachlichen Anweisung auszuführen: Die Erfolgsquote betrug 84% für die Gruppe ohne Programmierkenntnisse und 95% für die Gruppe mit Programmierkenntnissen; allerdings benötigten die Probanden teilweise mehrere Versuche. Die 36 Aufgaben (aus der Domäne Internet-Browser) der Studie waren alle sehr einfach gewählt und konnten mit wenigen Wörtern beschrieben werden. Inwieweit der Ansatz auf komplexere Aufgabenstellungen oder andere Domänen übertragbar ist, wird von den Autoren nicht diskutiert, erscheint aber aufgrund der Einfachheit fraglich.

SmartSynth von Le et al. erzeugt wiederum kurze Skripte für Smartphones aus englischsprachigen Beschreibungen [LGS13]. *SmartSynth* setzt dabei stark auf die Interaktion mit dem Nutzer. So werden zum Beispiel Parameter, die für den Aufruf einer API-Funktion benötigt werden und nicht inferiert werden konnten, über einen Dialog erfragt. In einer Nutzerstudie mit elf Studenten, die mithilfe von *SmartSynth* 50 unterschiedliche Skripte erstellen sollten, konnten die Autoren zeigen, dass ihr System wie erhofft funktioniert: die Erfolgsquote betrug (gegebenenfalls nach Rückfragen an die Probanden) 90%. Allerdings kann *SmartSynth* nur sehr kurze Skripte, bestehend aus ein oder zwei (zusammengesetzten) Befehlen erzeugen, gegebenenfalls unter Hinzunahme einer Bedingung. Der verfolgte Ansatz ist zudem regelbasiert und in einen Großteil der Regeln fließt Domänenwissen ein. Dadurch kann *SmartSynth* nur für die Erzeugung von Smartphone-Skripten eingesetzt werden und die Frage nach der Übertragbarkeit des Ansatzes bleibt offen.

Desai et al. verwenden maschinelles Lernen, um anhand syntaktischer Eigenschaften eine Abbildung von englischen Beschreibungstexten auf Skripte in verschiedenen domänenspezifischen (Programmier-)Sprachen (engl. *domain-specific language*, kurz *DSL*) herzustellen [Des+16]. Der Ansatz erlaubt zwar grundsätzlich die Abbildung von natürlicher Sprache auf beliebige DSLs; allerdings muss für jede domänenspezifischen Sprache eine entsprechende Datenmenge an Beispielabbildungen geliefert werden. Dementsprechend kann der Ansatz auch nur in Domänen evaluiert werden, für die ein ausreichend großer Datensatz zur Verfügung steht. Die Autoren verwenden zur Evaluation das *ATIS*-Korpus, das eine Sammlung von Anfragen an ein Fluginformations- und -buchungssystem inklusive erwarteter Systemaktionen enthält [Dah+94]. Außerdem nutzen sie Lehrbuchbeschreibungen (einfacher) endlicher Automaten und Beschreibungen zur Erstellung von Makros zur Textverarbeitung (aus Büchern und Foreneinträgen). Der Ansatz erreicht in allen Domänen eine Genauigkeit von über 80%, für den *ATIS*-Datensatz sogar 88%. Allerdings sind die zu erzeugenden Programme sehr kurz; die allermeisten bestehen nur aus einem einzelnen Aufruf.

Landhäußer et al. versuchen mit ihrem System *NLCI*, die semantische Analyse der Sprache und die Abbildung auf das Zielsystem weitestgehend zu entkoppeln [LWT17a; LWT17b]. Hierzu entwickeln sie eine generische Darstellung für Zielsysteme in Form einer Ontologie. Dadurch wird *NLCI* zwar nicht völlig unabhängig vom Zielsystem – die passende Zielsystemontologie muss immer noch gewählt werden – der Wechsel zwischen unterschiedlichen Zielsystemen wird allerdings vereinfacht. Außerdem ermöglicht *NLCI* prinzipiell die Erzeugung von beliebig langen Skripten anhand von Prosatexten, wobei alle Anweisungen API-Aufrufe von Zielsystemfunktionen darstellen; auch die Synthese von Kontrollstrukturen (bedingte Verzweigungen, Schleifen und Nebenläufigkeit) wird unterstützt. Die semantische Interpretation einer natürlichsprachlichen Beschreibung erfolgt ausschließlich regelbasiert auf Basis von Abhängigkeitsgraphen (siehe Abschnitt 2.3.4.6), die mithilfe der Werkzeugsammlung *Stanford CoreNLP*[5] erzeugt werden. Zur Abbildung auf die Zielsystemfunktion wird je Handlungsanweisung eine unscharfe Suche (in der Ontologie) durchgeführt; die gefundenen Abbildungskandidaten werden mithilfe eines eigens entwickelten Bewertungssystems sortiert und die beste Abbildung in das Programm eingefügt. Dieses Vorgehen kann in beliebigen Domänen ohne Anpassung verwendet werden, sofern eine Ontologie-Darstellung des Zielsystems existiert. *NLCI* wurde in zwei Domänen anhand von Beschreibungstexten evaluiert, die von Probanden angefertigt wurden: Die erste ist die Programmierlehrplattform *Alice*[6] in der programmatisch Animationen in einer 3D-Welt erstellt werden können; die zweite ist *openHAB*[7], ein Heimautomationssystem. Für *Alice*-Beschreibungstexte erzielt *NLCI* eine Ausbeute von 0,670 bei einer Präzision von 0,741 (bei Betrachtung einzelner API-Aufrufe); für *openHAB* sind es 0,700 bzw. 0,737. *NLCI* lässt sich zwar in beliebigen Domänen einsetzen; ein Einsatz für gesprochene Sprache (wie durch ProNat angestrebt) erscheint jedoch aufgrund der Abhängigkeit (der semantischen Interpretation) von tiefgehenden syntaktischen Analysen, wie der Erzeugung von Abhängigkeitsgraphen, nicht möglich. Außerdem birgt die semantische Interpretation (aufgrund des regelbasierten Vorgehens) prinzipiell Verbesserungspotenzial, wie auch die Evaluationsergebnisse nachweisen.

Einen ebenfalls Ontologie-basierten Ansatz verfolgen Atzeni und Atzori [AA18]. Ihre ontologische Darstellung fokussiert sich allerdings stärker auf die Struktur von Programmiersprachen. Da ihre Ontologie öffentlich zugänglich ist, kann Sie von anderen Wissenschaftlern erweitert und für eigene Ansätze zur semantischen Analyse der natürlichen Sprache verwendet werden. Die semantische Interpretation einer natürlichsprachlichen Beschreibung erfolgt (ähnlich wie bei *NLCI*) auf Grundlage von Abhängigkeitsgraphen (die ebenfalls mithilfe von *Stanford CoreNLP* erzeugt werden). Für die Abbildung auf in der Ontologie hinterlegte API-Funktionen verwenden Atzeni und Atzori ebenfalls einen Suchmechanismus; die Bewertung der Kandidaten erfolgt allerdings anhand eines Ensembles von Bewertungsmetriken, vorrangig solcher zur Bestimmung von Zeichenkettenähnlichkeiten: Unter anderem werden die Levenshtein-Ähnlichkeit (siehe Abschnitt 2.3.13.1) und Kosinus-Ähnlichkeit der Wortvektoren (siehe Abschnitt 2.3.16) der Such- und Ergebniszeichenkette verwendet. Evaluiert wurde der Ansatz anhand von zwei Datensätzen: Zum einen anhand einer Sammlung von Fragen aus einem Forum, welche auf den Aufruf einer API-Funktion abzielen und zum anderen anhand

[5] *Stanford CoreNLP*: `https://stanfordnlp.github.io/CoreNLP/`, zuletzt besucht am 24.02.2021.
[6] *Alice*: `https://www.alice.org/`, zuletzt besucht am 24.02.2021.
[7] *openHAB*: `https://www.openhab.org/`, zuletzt besucht am 24.02.2021.

einer Menge von natürlichsprachlichen Anfragen an ein (imaginäres) virtuelles Assistenzsystem. In beiden Domänen erzielt der Ansatz eine Präzision von 90% bei der Generierung der Aufrufe. Allerdings besteht ein Großteil der erzeugten Skripte aus genau einem Aufruf, nur wenige beinhalten zwei bis vier und mehr als vier Aufrufe sind zur Lösung der Aufgaben nie nötig. Ob der Ansatz auch zur Erzeugung von längeren Skripten verwendet werden kann, bleibt unklar; außerdem bleiben Kontrollstrukturen gänzlich unbeachtet.

3.3.2 Semantisches Zerteilen

Seit Beginn der 2000er Jahre hat sich das sogenannte semantische Zerteilen (engl. *semantic parsing*) etabliert. Beim semantischen Zerteilen werden Sätze – ähnlich dem üblichen (syntaktischen) Zerteilen – in Teilstrukturen zerlegt. Einzelne Satzabschnitte erhalten so eine eindeutige semantische Funktion. In den meisten Fällen werden semantische Zerteilungen mithilfe von kombinatorischen Kategorialgrammatiken (engl. *Combinatory Categorial Grammar*, kurz *CCG*) [Ste87] erzeugt bzw. dargestellt. Wie Grammatik-basierte Ansätze im Allgemeinen eignen sie sich allerdings nur für die Beschreibung von Beziehungen über kurze Distanzen. Das semantische Zerteilen wird neben anderen Anwendungen aus dem Bereich des Verständnisses der natürlichen Sprache vor allem für die Endnutzer-Programmierung mit (zumeist geschriebener) Sprache eingesetzt. Beispielsweise verwenden Vadas und Curran *CCGs* als Grundlage für einen regelbasierten Ansatz zur Erzeugung von *Python*-Skripten anhand englischsprachiger Prosatexte [VC05]: Sie Verwenden einen spezialisierten Zerteiler (engl. *parser*), der *CCG*-Ableitungen erzeugt. Ausgehend von der semantischen Zerteilung einer (laut der Autoren) uneingeschränkten natürlichsprachlichen Eingabe wird mithilfe von händisch erzeugten Regeln die Beschreibung als Programm inklusive einfacher Kontrollstrukturen interpretiert. Die Publikation enthält einige Beispiele, die Systemeingaben und -ausgaben illustrieren, aber keine Evaluation; daher ist es nicht möglich einzuschätzen, ob der Ansatz tatsächlich uneingeschränkte, freie Formulierungen als Programm interpretieren kann. Es ist aber eher unwahrscheinlich, dass das Verfahren uneingeschränkte Sprache versteht, da regelbasierte Verfahren im Allgemeinen kein breites Sprachverständnis besitzen (siehe Abschnitt 2.3.11); das bedeutet, das Vokabular, welches analysiert werden kann, ist geringer als bei statistik- oder wissensbasierten Verfahren[8].

Das von Zettlemoyer et al. entwickelte allgemeingültige Verfahren für die semantische Zerteilung mit *CCGs* liefert vielen weiterführenden Ansätzen Zugang zu *CCG*-Zerteilungen als Analysegrundlage [ZC05; ZC07]. Weitere Vorarbeiten zur semantischen Analyse natürlichsprachlicher Beschreibungen, zum Beispiel zur Analyse von zeitlichen Abläufen durch Lee et al. [Lee+14], einer Untersuchung zur Abdeckung der semantischen Strukturen durch Artzi et al. [ALZ15] und zur verbesserten Extraktion dieser Strukturen aus natürlicher Sprache durch Misra und Artzi [MA16], ermöglichten die Entwicklung verschiedener Ansätze zur Anwendung des semantischen Zerteilens mit und ohne *CCGs*. Konstas et al. nutzen ein sogenanntes Sequenz-zu-Sequenz-Modell (engl. *sequence-to-sequence model*) um eine abstrakte Bedeutungsrepräsentation (engl. *abstract meaning*

[8] Oder anders gesagt: Üblicherweise ist die Ausbeute gegenüber statistik- oder wissensbasierten Verfahren geringer (siehe Abschnitt 2.3.2).

representation, kurz *AMR*) für natürlichsprachliche Eingaben zu erzeugen [Kon+17]. Diese Repräsentation beschreibt die semantischen Zusammenhänge zwischen genannten Entitäten, Aktionen, Ereignissen und Zuständen und kann als Erweiterung von semantischen Rollen angesehen werden. Ein ähnliches Ziel verfolgen Liu et al. indem sie eine Bedeutungsrepräsentation für einen vollständigen Diskurs, ebenfalls mit einem Sequenz-zu-Sequenz-Modell, erzeugen [LCL18]. Ihr Ansatz basiert auf *CCGs* zur Darstellung der Diskurssemantik, welche mithilfe eines bidirektionalen *LSTMs* erzeugt wird (siehe Abschnitt 2.2.2). Der Ansatz von Srivastava et al. lernt automatisch abstrakte Konzepte anhand kleiner Datensätze [SLM17]. Die Autoren verwenden hierfür ebenfalls maschinelles Lernen; sie experimentieren mit logistischer Regression und Naïve-Bayes als Klassifikationsmodelle.

Ein erster Ansatz, der semantische Zerteilung konkret zur Erzeugung von Programm-artigen Skripten aus natürlichsprachlichen Texteingaben verwendet[9], wird von Long et al. präsentiert [LPL16]. Die Aufgabenstellung wird unter der Prämisse betrachtet, dass lediglich indirekte Überwachung (engl. *indirekt supervision*) zur Modellerzeugung möglich ist; das bedeutet, es kann nur die Ausgabe des erzeugten Programms untersucht werden, die einzelnen Quelltextbestandteile (das heißt Anweisungen) bleiben verborgen. Die Autoren verwenden zur Lösung des Problems drei unterschiedliche Modellarten: Das erste Modell verwendet alle möglichen *CCG*-Zerteilungen als Eingabe; die anderen beiden führen die Zerteilungen systematisch zusammen. Alle Zerteilungen werden letztlich mithilfe eines *Beam-Search*-Verfahrens auf (Programm-)Anweisungen abgebildet. Zur Evaluation erstellen sie über eine Schwarmauslagerungsplattform drei Datensätze, die ähnlich geartet sind. Drei unterschiedliche (imaginäre) Systeme sollen von einem Zustand in einen anderen (bekannten) Zustand überführt werden. Probanden sollen die nötigen Anweisungen bzw. Aktionen (die in einer logischen Repräsentation, das heißt als mehrstellige Prädikate mit bekannten Prädikatbezeichnern und Argumenten) erteilen, wobei sie bis zu fünf Aktionen verwenden dürfen. Zusätzlich werden natürlichsprachliche Beschreibungen für die Überführungen gesammelt. Anhand dieser Paare können die unterschiedlichen Modelle erzeugt werden. Die Autoren zeigen in der Evaluation, dass Skripte, die bis zu drei Aktionen benötigen, mit einer Genauigkeit von 23% bis 65% (je nach verwendeten Modell und Datensatz) erzeugt werden; bei Skripten, die bis zu fünf Aktionen enthalten, sinken die Werte auf 15% bis 52%. Guu et al. kombinieren verstärkendes Lernen (engl. *reinforcement learning*) sowie die Maximal-Marginal-Likelihood-Methode zur Lösung derselben Aufgabe auf Basis der drei von Long et al. erstellten Datensätze [Guu+17]. Ihr System passt das Modell je Eingabe so lange an, bis der erwartete Zustand erreicht ist. Auf diese Weise gelingt es ihnen die Genauigkeit auf 67% (Skripte mit drei Aktionen) bzw. 37% (fünf Aktionen) zu steigern. Auch Suhr und Artzi präsentieren eine Lösung für die von Long et al. definierte Problemstellung [SA18]. Sie verwenden ein neuronales Netz, welches den situativen Kontext (das heißt den aktuellen Systemzustand) mit in den Übersetzungsprozess einbezieht. In diesem Fall schließt der situative Kontext den sprachlichen Diskurs zwischen System und Nutzer sowie Wissen über die Systemumgebung bzw. den aktuellen Zustand des Systems ein. Auf diese Weise gelingt es den Autoren, die Genauigkeit bei der Erzeugung der Skripte weiter zu steigern: bis zu 83% für Skripte bestehend aus drei Aktionen und 72% bei

[9] Tatsächlich wird das Ziel verfolgt, natürlichsprachliche Äußerungen in Textform in eine logische Repräsentation zu überführen, die Quelltext ähnelt. Die betrachteten Zielsysteme sind allerdings synthetisch und es wird kein tatsächlich ausführbarer Quelltext erzeugt. Die logische Repräsentation ist aber so entworfen, dass sie glaubhaft einer Anwendungsschnittstelle ähnelt.

fünf Aktionen. Das grundsätzliche Problem mit der von Long et al. definierten Aufgabe ist, dass es sich lediglich um imaginäre Systeme handelt. Das bedeutet, es wird kein echter Quelltext erzeugt, sondern nur Anweisungen in Prädikatschreibweise, die ähnlich wie programmatische Anweisungen geformt sind. Außerdem ist der Funktionsumfang der betrachteten Systeme sehr gering. Aus diesen Gründen lässt sich schwer abschätzen, inwieweit die drei diskutierten Ansätze auf andere Domänen und echte Anwendungsschnittstellen übertragbar sind, auch wenn alle konzeptionell übertragbar erscheinen. Zudem verwenden alle maschinell erlernte Modelle, die in jedem Fall zur Verwendung in anderen Domänen neu trainiert werden müssten. Einzig der ursprüngliche Ansatz von Long et al. erscheint besser übertragbar, da als Analysegrundlage *CCG*-Zerteilungen verwendet werden[10].

Quirk et al. widmen sich der Erzeugung von kurzen bedingten Programmen aus textuellen Beschreibungen [QMG15]. Das bedeutet, es sollen bestimmte Aktionen (API-Aufrufe) ausgeführt werden, sobald ein äußerer Impuls (engl. *trigger*) ausgelöst wird. Aus programmatischer Sicht kann diese Anforderung mithilfe von *Wenn-Dann*-Strukturen (engl. *if-then expression*), also durch bedingte Verzweigungen, umgesetzt werden. Als Basis für ihren Ansatz verwenden die Autoren einen Datensatz, den sie automatisch über den Webservice *IFTTT*[11] zusammentragen: *IFTTT* ist eine Sammlung sogenannter *If-This-Then-That Recipes* (deutsch: *Wenn-Dies-Dann-Das-Rezepte*), das heißt kurzer Programme (z. B. für Heimautomation), die ausgelöst werden, wenn eine definierte Bedingung zutrifft. Für jedes Programm gibt der Ersteller zudem eine kurze natürlichsprachliche Beschreibung an. Die Beschreibungen (als Eingabebeispiele) und das jeweils zugehörige bedingte Programm (als Ausgabe) bilden den Datensatz[12]. Quirk et al. vergleichen auf Grundlage des *IFTTT*-Datensatzes in ihrer Publikation verschiedene Ansätze (vor allem maschinell erlernte Klassifikatoren). Die besten Ergebnisse erzielt ein Klassifikator, der logistische Regression verwendet (siehe Abschnitt 2.2.1.1); dieser erzielt eine Genauigkeit von 35% für die Erzeugung vollständiger *IFTTT Recipes*. In einer Erweiterung des Ansatzes durch Beltagy und Quirk wird zusätzlich ein vorwärts gerichtetes neuronales Netz (siehe Abschnitt 2.2.2) trainiert und außerdem die Klassifikation des Netzes mit der des Klassifikators basierend auf logistischer Regression verknüpft (als Klassifikator-Ensemble); letzteres steigt die Genauigkeit auf 43%. Mit der Betrachtung von bedingten Programmen liefern Quirk et al. eine interessante Problemstellung, da Kontrollstrukturen ansonsten in der Literatur wenig Beachtung finden. Allerdings sind die natürlichsprachlichen Beschreibungen des *IFTTT*-Datensatzes nicht allgemeingültig: Sie sind sehr kurz und die Bedingung wird fast immer zu Beginn formuliert, was die Erkennung deutlich vereinfacht. Außerdem enthalten die *Dann*-Blöcke der Programme nur wenige Anweisungen, meist sogar nur eine. Ob der Ansatz auf umfangreichere Blöcke übertragbar ist, kann nicht prognostiziert werden. Nicht betrachtet werden zudem *Andernfalls*-Blöcke (engl. *else*). Auch die Möglichkeit, dass *Wenn-Dann-Andernfalls*-Strukturen Teil eines längeren Skriptes sein könnten (und die Bedingung eben nicht zu Beginn formuliert wird) ist nicht Teil der Betrachtung.

[10] Dies hat den Vorteil, dass Modelle Klassifikationsentscheidungen gegebenenfalls anhand abstrakter (Zerteilungs-)Strukturen treffen statt anhand konkreter (Schlüssel-)Wörter oder Phrasen.

[11] *IFTTT*: `https://ifttt.com/`, zuletzt besucht am 24.02.2021.

[12] Seit September 2020 ist die Nutzung von *IFTTT* kostenpflichtig. Ein derartiger Datensatz kann dementsprechend nicht mehr ohne weiteres nachgebildet werden. Der Original-Datensatz von 2015 ist jedoch weiterhin verfügbar: `https://github.com/Jungyhuk/Latent-Attention/blob/master/dataset/IFTTT/msr_data.pkl`, zuletzt besucht am 24.02.2021.

Mithilfe rekurrenter neuronaler Netze erzeugen Chen et al. ein Sequenz-zu-Sequenz-Modell, welches aus englischsprachigen Texten ausführbare Aktionen synthetisiert [CSH18]. Die Netze lernen implizit sogenannte *Aktionseinbettungen*, ähnlich den bekannten Worteinbettungen (siehe Abschnitt 2.3.16). Die Autoren evaluieren ihren Ansatz anhand von drei Datensätzen, dem (bereits im vorangegangen Abschnitt erwähnten) *ATIS*-Datensatz, *GeoQuery*, einem Datensatz bestehend aus Datenbankabfragen und zugehörigen natürlichsprachlichen Anfragen[13] [ZM96], sowie ebensolchen Anfragepaaren an ein synthetisches virtuelles Assistenzsystem. Das bedeutet, alle Programme bestehen lediglich aus einzelnen Aufrufen[14]. Der Ansatz erzielt sehr gute Genauigkeiten: 86% für *ATIS*, 89% für *GeoQuery* und 79% für das virtuelle Assistenzsystem. Allerdings muss für jede Domäne das Klassifikationsmodell von Grund auf neu trainiert werden; der Ansatz ist also nur auf Domänen übertragbar, für die ein ausreichend großer Datensatz zur Verfügung steht.

Ling et al. widmen sich der Objektorientierung und bilden mit ihrem Ansatz textuelle Beschreibungen auf Klassenimplementierung ab. Hierzu verwenden sie eine angepasste Variante eines bidirektionalen *LSTMs* (siehe Abschnitt 2.2.2). Interessant ist vor allem der Datensatz, den die Autoren zur Evaluation ihres Ansatzes verwenden: Sie erzeugen einen Korpus mithilfe virtueller Kartenspiele (das heißt Computerspielvarianten von Strategie-Sammelkartenspielen). Jede Spielkarte besitzt eine Beschreibung (ihrer Fähigkeiten) sowie eine zugehörige Implementierung (bestehend aus einer Klasse mit Attributen und öffentlichen Methoden). Die Implementierungen der Klassen unterscheiden sich deutlich, da jede Karte in den jeweiligen Spielen unterschiedliche Fähigkeiten besitzt. Ling et al. erzeugen zwei Datensätze, indem sie automatisch die Beschreibungen und Implementierungen der Karten der beiden Spiele *Hearthstone*[15] (*HS*-Datensatz) und *Magic the Gathering*[16] (*MtG*-Datensatz) sammeln; die Karten von ersterem sind in *Python* und von zweiterem in *Java* implementiert. Zwar erreichen Ling et al. mit ihrem Ansatz nur Genauigkeiten von 5% (*MtG*-Datensatz) bzw. 6% (*HS*-Datensatz); für die *BLEU*-Metrik (siehe Abschnitt 2.4.7) werden aber immerhin Werte von 61% bzw. 66% erzielt. Auch wenn die Ergebnisse deutliches Verbesserungspotenzial aufweisen, präsentieren Ling et al. eine interessante und herausfordernde Problemstellung; insbesondere die nicht offensichtliche Abbildung einer eher abstrakten textuellen Beschreibung auf eine feingranulare Implementierung stellt eine interessante Aufgabe dar. Leider diskutieren die Autoren nicht, ob ihr Ansatz auf die Implementierung andersartiger bzw. komplexerer Klassen übertragbar ist.

Yin und Neubig verwenden rekurrente neuronale Netze, um aus textuellen Beschreibungen abstrakte Syntaxbäume (engl. *abstract syntax tree*, kurz *AST*) zu erzeugen [YN17]. Ausgehend von den erzeugten ASTs kann deterministisch Quelltext erzeugt werden. Die Autoren evaluieren ihren Ansatz anhand drei sehr unterschiedlicher (und anspruchsvoller) Domänen. Zum einen verwenden sie den *HS*- und den *IFTTT*-Datensatz (bzw. einen Teil des letzteren). Zum anderen nutzen sie den sogenannten *DJANGO*-Datensatz [Oda+15]; dieser enthält Tupel, bestehend aus *Python*-Quelltext

[13] Eine ausführlichere Beschreibung des *GeoQuery*-Datensatzes befindet sich in Abschnitt 3.4. Dort wird die Arbeit von Zelle und Mooney thematisiert, die den Datensatz eingeführt hat.

[14] Der Ansatz hätte dementsprechend auch dem Forschungsgebiet *Erzeugung von Datenbankabfragen aus natürlichsprachlichen Äußerungen* (siehe Abschnitt 3.4) zugeordnet werden können. Die Einordnung an dieser Stelle erfolgte aufgrund der Wahl des virtuellen Assistenzsystem als weitere Domäne und der angewandten Techniken.

[15] Virtuelles Online-Kartenspiel *Hearthstone*: `https://playhearthstone.com`, zuletzt besucht am 24.02.2021.

[16] Virtuelles Online-Kartenspiel *Magic the Gathering*: `https://magic.wizards.com`, zuletzt besucht am 24.02.2021.

und zugehörigen natürlichsprachlichen (Pseudo-)Quelltext-Beschreibungen aus einer vorherigen Arbeit von Neubig. Für den *HS*-Datensatz geben die Autoren eine Genauigkeit von 16% an, was einer Verbesserung um immerhin 10 Prozentpunkte gegenüber der Arbeit von Ling et al. entspricht. Auf dem *DJANGO*-Datensatz wird eine Genauigkeit von 72% erreicht. Für *IFTTT* wird sogar ein Wert von 90% erzielt; allerdings wurde nur ein kleiner Ausschnitt des *IFTTT*-Datensatzes verwendet, der nur 758 ausgewählte natürlichsprachliche Beschreibungen und zugehörige Implementierungen enthält, die in einer Nutzerstudie von einer Mehrheit von menschlichen Bewertern (mindestens drei von fünf) als *sehr gut* bewertet wurden. [17]. Der Original-Ansatz von Quirk und Beltagy erzielt für diesen Teil des Datensatzes auch eine deutliche höhere Genauigkeit (als zuvor berichtet) von 83%. Ein wesentlicher Nachteil des Ansatzes von Yin und Neubig ist, dass dieser die ASTs in Abhängigkeit von der Zielprogrammiersprache erlernt; das bedeutet, für jede Programmiersprache muss ein eigenes Netz trainiert werden. Das Verfahren könnte aber auf andere Domänen übertragbar sein, solange die Programmiersprache gleich und die Aufgabe sehr ähnlich ist, nachgewiesen wurde dies in der Publikation jedoch nicht[18].

Der Ansatz von Rabinovich et al. erzeugt ebenfalls ASTs mithilfe eines bidirektionalen *LSTMs* aus textuellen Beschreibungen [RSK17]. Allerdings lassen sich die erzeugten ASTs laut der Autoren prinzipiell auf beliebige Programmiersprachen abbilden; präsentiert wird in der Evaluation jedoch lediglich eine Abbildung auf *Python* (*HS*-Datensatz) sowie die domänenspezifische Sprachen der *ATIS*- und *GeoQuery*-Datensätze. Für die beiden letzten erzielt der Ansatz sehr gute Genauigkeiten von 86% (*ATIS*) bzw. 87% (*GeoQuery*); allerdings müssen in diesen Domänen nur einzelne Aufrufe erzeugt werden. In der deutlich anspruchsvollen Domäne des *HS*-Datensatzes gelingt Rabinovich et al. aber ebenfalls eine deutliche Verbesserung der Genauigkeit (gegenüber Ling et al. sowie Yin und Neubig) auf über 22%. Die Besonderheit an dem eingesetzten *LSTM* ist der modulare Aufbau des Dekodierers. Je nach zu erzeugendem AST-Bestandteil (Aufruf, Bedingung, Schleife usw.) wird ein anderes speziell trainiertes Dekodierer-Modul verwendet.

Dong und Lapata implementieren einen zweistufigen Ansatz für semantische Zerteilung [DL18]. Im ersten Schritt wird nur eine leichtgewichtige Bedeutungsrepräsentation der natürlichsprachlichen Eingabe erzeugt. Die Repräsentation dient zusammen mit der eigentlichen Äußerung als Eingabe für ein bidirektionales *LSTM*. Der Ansatz kann für unterschiedliche Programmiersprachen und Zielsysteme verwendet werden; das *LSTM*-Modell (zweiter Teil des Ansatzes) muss jedoch für jedes anzusprechende Zielsystem neu trainiert werden. Dong und Lapata evaluieren ihren Ansatz anhand der Datensätze *ATIS*, *GeoQuery* und *DJANGO*; zusätzlich verwenden sie als weiteren Datensatz *WikiSQL* [ZXS17], eine Sammlung von *SQL*-Abfragen von Wikipedia[19]. Diese Datensätze enthalten ausschließlich kurze textuelle Beschreibungen, die jeweils auf einen einzelnen Aufruf abgebildet werden müssen. Der Ansatz erzielt durchweg gute Genauigkeiten: circa 88% für *GeoQuery* und *ATIS* sowie 74% für *DJANGO* und 72% für *WikiSQL*.

[17] Möglicherweise handelt es sich bei diesen Datenpunkten um besonders einfache bzw. offensichtliche Tupel (bestehend aus Beschreibung und Implementierung).

[18] Die in der Evaluation betrachteten Datensätze verwenden unterschiedliche Programmiersprachen bzw. domänenspezifische Sprachen. Dementsprechend wurden die verwendeten Modelle von Grund auf trainiert.

[19] Der Datensatz entstammt der Arbeit von Zhong et al., die im nachfolgenden Abschnitt 3.4 diskutiert wird.

Das Werkzeug *NL2Bash*, entwickelt von Lin et al., ist auf ein Zielsystem und eine Programmiersprache festgelegt [Lin+18]; *NL2Bash* generiert aus natürlichsprachlichen Anweisungen *Bash*-Kommandos für das Betriebssystem *Linux*. Die Autoren vergleichen für diese Abbildung verschiedene neuronale Netze, vor allem rekurrente Netz-Architekturen. Zusätzlich implementieren sie einen hybriden Ansatz, bestehend aus einem maschinell erlernten Modell, dessen Klassifikation nachträglich mithilfe von Heuristiken angepasst wird. Den Datensatz für das Training und den Test der Klassifikatoren erstellen Lin et al. wie folgt: Sie lassen zehn professionelle Entwickler *Bash*-Kommandos (Ausgabe) sowie zugehörige Beschreibungen (Eingabe) erstellen und sammeln auf diese Weise 9305 Ein-Ausgabe-Tupel. Die meisten Beschreibungen bestehen aus einzelnen Sätzen und auch die Kommandos sind größtenteils Einzelkommandos; teils werden aber auch zwei oder mehr verknüpft. In der Evaluation zeigen Lin et al., dass sich eine der rekurrenten Netzkonfigurationen am besten für diese Problemstellung eignet: Der Klassifikator erzielt eine Genauigkeit auf der Testmenge von 36%. Dieses Ergebnis zeigt, dass noch deutliches Verbesserungspotential besteht. Abgesehen davon präsentieren Lin et al. mit der Erzeugung von *Bash*-Kommandos aus natürlichsprachlichen Beschreibungen eine interessante Problemstellung. Inwieweit etwaige Lösungen auf andere Aufgaben bzw. Domänen übertragbar sind, bleibt jedoch offen. Der von Lin et al. vorgestellte Ansatz ist auf alle Fälle nicht ohne Weiteres wiederverwendbar: Das gelernte Modell überführt Text direkt in *Bash*-Kommandos, was bedeutet, dass alleine zur Anbindung neuer Befehle ein neues Modell trainiert werden müsste.

3.3.3 Ansätze aus der Robotik-Domäne

Besondere Relevanz besitzt die Programmierung mit natürlicher Sprache im Kontext der humanoiden Robotik. Um humanoide Roboter zukünftig im Alltag einsetzen zu können, muss eine für den Nutzer möglichst natürliche Interaktion zwischen Mensch und Maschine gewährleistet werden. Das menschliche Erscheinungsbild dieser Roboter weckt im menschlichen Gegenüber die Erwartung, dass mit dem Roboter wie mit einem Menschen kommuniziert werden kann. Dies schließt unter anderem die Verwendung natürlicher Sprache zur Erteilung von Anweisungen und Erklären von Arbeitsabläufen ein. Es ist daher nicht verwunderlich, dass sich eine Vielzahl von Ansätzen zur Programmierung mit natürlicher Sprache auf diese Domäne fokussiert. Ausgehend von einer Voruntersuchung zur Analyse von Richtungsangaben in natürlichsprachlichen Instruktionen [Kol+10] präsentieren Tellex et al. einen Ansatz zur Roboter-Navigation mit natürlicher Sprache [Tel+11]. Ihr Ansatz lernt Ausführungspläne aus einer Sammlung natürlichsprachlicher Instruktionen. Die Datenbasis für ihr Modell wurde über eine Schwarmauslagerungsplattform gesammelt. Als Eingabe akzeptiert der Ansatz nur stark eingeschränkte Formulierungen: Es sind ausschließlich Handlungsanweisungen im Imperativ zulässig; diese dürfen maximal zwei Aktionen umfassen (die über die Konjunktion *and* verknüpft sind). Die Autoren evaluieren ihren Ansatz anhand des gesammelten Datensatzes: Das gelernte Modell erzielt eine Genauigkeit auf der Testmenge (siehe Abschnitt 2.2) von 90%. Da das Modell direkt anhand von Tupeln, bestehend aus natürlichsprachlichen Handlungsanweisung (als Eingabe) und der jeweils zugehörigen erwarteten Instruktionsfolge (als Ausgabe), trainiert wird, ist die Übertragbarkeit des Ansatzes auf ein anderes Robotersystem oder eine gänzlich andere Domäne fraglich; zumindest müsste das Modell erneut trainiert werden.

Nyga und Beetz beschreiben, welche Art von Wissensbasis nötig ist, damit Haushaltsroboter natürlichsprachliche Instruktionen verstehen können und wie diese bereitgestellt werden könnte [NB12]. Ihr Wissensmodell, genannt *PRAC*, entsteht durch die Zusammenführung von Informationen aus unterschiedlichen Wissensquellen: Unter anderem werden die *WordNet*-Taxonomie (siehe Abschnitt 2.3.15.1), *FrameNet*-Rahmen (siehe Abschnitt 2.3.15.2) und *WikiHow*-Beschreibungen[20], für die Abhängigkeitsgraphen mithilfe des Stanford'schen Zerteilers erzeugt werden, verwendet. Die (automatisch) zusammengeführten Informationen werden als Markov-Modell repräsentiert. Die Publikation enthält keine Evaluation; stattdessen diskutieren Nyga und Beetz Verwendungsmöglichkeiten für ihr Wissensmodell anhand von Beispielen.

Statt einer Wissensbasis verwenden Lincoln und Verres ein zielorientiertes Planungsmodell, in welches neben textuellen Beschreibungstexten auch eine Systemumgebungs- und Zustandsrepräsentation einfließt [LV12]. Die Autoren erzeugen ein geteiltes Planungsmodell für die beiden Kooperationspartner Mensch und Maschine, welches die gemeinsamen Ziele beziehungsweise Absichten repräsentiert. Allerdings wirkt die von den Autoren verwendete Sprache sehr technisch, was auf die angestrebt maschinennahe Programmierung der Roboter zurückgeführt werden kann. Die Publikation zeigt exemplarisch, wie der Ansatz zur Programmierung zweier unterschiedlicher Robotersysteme eingesetzt wurde; eine quantitative Untersuchung wurde jedoch nicht durchgeführt.

Der Ansatz von Mutuszek et al. lernt ein Modell, das die Semantik natürlichsprachlicher Handlungsanweisungen analysieren und in ausführbare Instruktionen an ein Robotersystem überführen kann [Mat+13]. Als Grundlage für ihren Ansatz verwenden die Autoren *CCGs* zur Erfassung der Semantik der natürlichsprachlichen Beschreibungen. Die *CCG*-Zerteilungen dienen wiederum als Eingabe für ein Modell, das die Zerteilungen in kurze Skripte bestehend aus Aufrufen an ein Roboter-Kontrollsystem überführt. In einer Evaluation wird ein Modell für die Aufgabe der *Wegfindung* trainiert: Anhand von natürlichsprachlichen Wegbeschreibungen sollen Navigationsbefehle für das Roboter-Kontrollsystem erzeugt werden; dies gelingt bei kurzen Beschreibungstexten in 66% der Fälle; bei längeren Beschreibungen sind hingegen nur 49% der generierten Befehle korrekt. Leider kann anhand der Publikation nicht eingeschätzt werden, wie generisch der von Mutuszek et al. verfolgte Ansatz ist. In jedem Fall muss ein neues Modell trainiert werden, wenn ein anderes Zielsystem angesprochen werden soll, höchstwahrscheinlich aber auch, wenn sich die Art der Aufgabenstellung (an den Roboter) ändert.

Ein Ansatz, der nicht nur einzelne Aktionen oder Aktionsfolgen, sondern neue Funktionalität erzeugen soll, wurde von She et al. entwickelt [She+14]. Neue Funktionen können im Dialog zwischen Mensch und Maschine erlernt werden, indem der Nutzer eine Beschreibung in natürlicher Sprache abgibt. Die Beschreibungen müssen jedoch einem vordefinierten Schema mit festen Reihenfolgen und Formulierungen entsprechen. Der Ansatz übersetzt die Beschreibung mithilfe von semantischen Rollen (siehe Abschnitt 2.3.4.8) und einer Stichwortsuche in ein Ablaufmodell. Die in der Publikation beschriebene Evaluation umfasst das Erlernen von sechs neuen Funktionalitäten. Diese werden anschließend in jeweils zwanzig unterschiedlichen Szenarien mittels natürlichsprachlicher Anweisungen (nach festem Schema) aufgerufen; der Aufruf gelingt in 88% der Fälle. Da der Ansatz

[20] *WikiHow* ist eine Online-Sammlung von Kurzanleitungen für diverse (Alltags-)Probleme: `https://www.wikihow.com/`, zuletzt besucht am 24.02.2021.

im Wesentlichen auf semantischen Rollen und Stichwörtern basiert, sollte er einfach auf andere Aufgaben übertragbar sein; die Festlegung auf einen Roboter als Zielsystem erscheint hingegen (aufgrund der Gestalt der verwendeten Ableitungsregeln) nicht ohne Weiteres änderbar.

Thomason et al. nutzen semantisches Zerteilen für ihren Dialog-basierten Ansatz zur natürlichsprachlichen Steuerung von humanoiden Robotern [Tho+15]. Hierzu lernt ihr Ansatz ein Modell aus einer Datenbasis an Instruktionen, die wiederum über eine Schwarmauslagerungsplattform erzeugt wurde. Das Modell wird zur Laufzeit sukzessive mithilfe vergangener Dialoge mit verschiedenen Nutzern angepasst. In der Evaluation konnten die Autoren zeigen, dass diese sukzessive Verbesserung gelingt. In einer Nutzerstudie gab ein Großteil der Probanden nach der Verwendung des Systems über mehrere (Dialog)-Iterationen an, dass der Roboter sie mit der Zeit besser versteht. Eine zweite Evaluation konnte diese Ergebnisse bestätigen: Bei der Lösung von unterschiedlichen Aufgabenstellungen steigt die Erfolgsquote mit sukzessiven Lernen auf 60%; ohne gelerntes Wissen beträgt sie lediglich 20%.

Misra et al. verfolgen einen anderen Ansatz [Mis+15]: Sie überführen natürlichsprachliche Beschreibungstexte und Informationen aus einem Systemumgebungsmodell in eine gemeinsame logische Repräsentation. Anschließend wird ein Planungsmodell für die auszuführenden Roboteraktionen aus der logischen Repräsentation generiert. Hierzu wird ein Modell zur Abbildung von natürlicher Sprache auf Roboteraktionen maschinell erlernt, wobei unklar bleibt, ob und wie gut dieses generalisiert. Das erlernt Modell kann jedoch lange Anweisungsfolgen (als Programmskripte) generieren. In der Evaluation enthalten die erzeugten Skripte im Durchschnitt 21 Aktionen. Leider erfolgt die Bemessung der Qualität anhand eigens definierter Metriken; diese sind zwar für die betrachtete Problemstellung aussagekräftig aber nicht vergleichbar. Eine der Metriken ist von der Levenshtein-Distanz abgeleitet (siehe Abschnitt 2.3.13.1): Die sogenannte *Instruction Edit Distance* (kurz *IED*) betrachtet die Änderungen an einem erzeugten Skript, die notwendig sind um das zugehörige Musterlösungsskript zu erhalten. Für diese Metrik erzielt der Ansatz im Mittel einen Wert von 22,8.

Markievic et al. nutzen Aufgabenbeschreibungen, die ursprünglich für menschliche Leser verfasst wurden, um Robotersysteme anzulernen [Mar+17]. Sie versuchen so die Problematik der Anpassung von Modellen auf neue Aufgabenstellungen zu umgehen. Die Autoren verwenden semantische Rollen, um die Semantik der natürlichen Sprache zu erfassen. Allerdings erlaubt ihr Ansatz lediglich die Verknüpfung von zwei Handlungsanweisungen in einer Beschreibung. Der Ansatz wurde anhand von 87 Beschreibungen (nicht genannten Ursprungs) evaluiert; die Beschreibungen bestehen fast ausschließlich aus einzelnen Handlungsanweisungen, die zudem bereits wie Funktionsaufrufe formuliert wurden (das bedeutet auch, dass die Benennungen identisch mit den Bezeichnern der Systemfunktionen bzw. Parameter waren)[21]. Bei der Auswertung der Ergebnisse wurden nicht die generierten Aufrufe betrachtet; stattdessen wurde ausschließlich die Präzision bei der Abbildung der Einzelbestandteile (das heißt Funktionsname, erster Parameter, zweiter Parameter usw.) bemessen; diese Abbildungen erreichen für die Metrik Präzision Werte zwischen 0,828 und 0,977.

[21] Die Publikation listet einen Großteil der für die Evaluation verwendeten Beschreibungen und erwarteten Aufrufe.

3.3.4 Integration anderer Techniken der Endnutzer-Programmierung

Einige Ansätze kombinieren Techniken aus anderen Bereichen der Endnutzer-Programmierung und das Programmieren mit natürlicher Sprache. Entweder werden Aktionen demonstriert, zum Beispiel durch Gesten beziehungsweise das Zeigen auf Objekte, oder es werden Ein- und Ausgabebeispiele für das zu erzeugende Skript angegeben. Letzteres bezeichnet man als Programmierung mithilfe von Beispielen (engl. *Programming by Example*, kurz *PbE*, siehe Abschnitt 2.1).

Ein Vertreter dieser Gattung wurde von Manshadi et al. entwickelt [MGA13]. Die Autoren trainieren ein Modell, welches mögliche Sequenzen von Funktionsaufrufen zu Ein- und Ausgabebeispielen generiert und ordnet. Zusätzlich werden lexikalische Informationen aus den textuellen Beschreibungen verwendet, um die Reihenfolge der Sequenz-Kandidaten neu zu sortieren. Anhand einer vergleichenden Evaluation können die Autoren zeigen, dass durch die Integration lexikalischer Informationen in einen *PbE*-Ansatz die Genauigkeit gesteigert werden kann. Dies gilt insbesondere, wenn nur wenige Ein-/Ausgabebeispiele für das Training des *PbE*-Modells zur Verfügung stehen; beispielsweise zeigen die Autoren, dass bei Verwendung lexikalischer Informationen die Genauigkeit bei der Programmerzeugung von 19% auf 42% gesteigert werden kann, wenn nur ein Beispielpaar zur Verfügung steht.

APPINITE, entwickelt von Li et al. [Li+18], verbindet Programmierung mit natürlicher Sprache mit Programmierung durch Demonstration. Ein Nutzer kann entweder eine Folge von Aktionen als Makro aufnehmen und diese anschließend mit einem natürlichsprachlichen Aufruf-Namen versehen oder aber eine textuelle Beschreibung (einen einzelnen kurzen Satz) abgeben und sich von *APPINITE* eine Sequenz von Aktionen vorschlagen lassen. In beiden Fällen kann der Nutzer das Ergebnis im Dialog mit dem System nachträglich anpassen. *APPINITE* wurde in einer Nutzerstudie mit sechs Probanden evaluiert; diese sollten mithilfe von *APPINITE* zwanzig neue Makros zur Lösung unterschiedlichen Aufgaben erstellen. In 87% der Fälle waren die Probanden dabei erfolgreich, teilweise erst nach einem Dialog mit dem System. Allerdings wurden hauptsächlich Zeigegesten verwendet und alle ermittelten Fehler konnten auf fehlerhafte (bzw. nicht durchführbare) Zerteilungen natürlichsprachlicher Beschreibungen zurückgeführt werden. *APPINITE* verwendet intern einen spezialisierten Zerteiler (engl. *parser*), der die natürliche Sprache in eine logische Zwischensprache überführt und dabei interpretiert; diese Überführung scheint jedoch nicht immer möglich. Der Zerteiler wurde anhand von Ein-/Ausgabebeispielen trainiert; das bedeutet auch, dass ein neues Modell trainiert werden muss, wenn andere Arten von Makros erzeugt oder andere Services angesprochen werden sollen.

3.4 Angrenzende Forschungsgebiete

Neben den Arbeiten zur Programmierung mit natürlicher Sprache existieren verschiedene Arbeiten aus angrenzenden Forschungsgebieten, die gewisse Aspekte mit dem hier vorgestellten Ansatz teilen. Arbeiten aus dem Bereich der automatischen Erzeugung von Testfällen aus natürlicher Sprache verfolgen eine ähnliche Zielstellung. Auch hier werden ausführbare Skripte aus natürlichsprachlichen

Beschreibungen erzeugt, nur handelt es sich hierbei um Test-Skripte und keine eigentlichen Programm-Skripte. Andere Ansätze ähneln dem hier vorgestellten in den verwendeten Techniken. Arbeiten aus den Bereichen konversationelle Schnittstellen und virtuelle Assistenten analysieren ebenfalls natürliche Sprache, überführen sie in eine semantische Zwischenrepräsentation und bilden diese anschließend entweder auf eine Abfragesprache wie *SQL* oder Dienstgeberaufrufe ab.

3.4.1 Automatisierte Erstellung von Testfällen aus natürlichsprachlichen Beschreibungen

Ansätze aus dem Forschungsgebiet der Erzeugung von Test-Skripten aus natürlichsprachlichen Beschreibungen beschränken den Nutzer häufig auf kontrollierte Sprache. Außerdem wird zumeist ein Format vorgeben, welches die Eingaben zusätzlich strukturiert. Häufig werden hierzu Formulare mit voneinander abhängenden Feldern ausgefüllt. Diese Einschränkungen sind insofern unkritisch, als sich diese Ansätze vorwiegend an Software-Entwickler richten. Für diese soll das Testen von Software lediglich vereinfacht und strukturiert werden.

Ein Ansatz dieser Art wurde von Thumalapenta et al. entwickelt [Thu+12; Thu+13]; dieser eignet sich zum Testen von grafischen Benutzeroberflächen. Die Eingabe der Testfälle erfolgt textuell und besteht aus fortlaufend nummerierten Kommandos im Imperativ. Die Interpretation der natürlichen Sprache erfolgt mithilfe von Schlüsselwörtern und einer oberflächlichen bzw. seichten Zerteilung (engl. *shallow parsing*, siehe auch Abschnitt 6.2). Zusätzlich verwendet das System einen Rücksetzungsmechanismus (engl. *backtracking*), beispielsweise falls eine (anhand der Sprache inferierte) Aktion aufgrund des aktuellen Zustands der grafischen Benutzeroberflächen nicht ausführbar ist. Die Autoren zeigen in einer Evaluation, dass gut 82% der für ein Testskript benötigten Aktionen korrekt erzeugt werden können. Hierzu wird in vielen Fällen aber der Rücksetzungsmechanismus verwendet, der wiederum eine konkrete Systemantwort voraussetzt (beispielsweise eine Fehlermeldung bei einer nicht erfolgreichen Anmeldung am getesteten System). Das bedeutet, die Lösung ist stark auf die konkrete Problemstellung angepasst; eine Übertragbarkeit auf andere Testaufgaben oder auf die Erzeugung von Programmen (statt Tests) scheint nicht ohne Weiteres möglich.

Wang et al. generieren System-Tests und -Modelle aus Anwendungsfall-Spezifikationen [Wan+15]. Die Autoren siedeln ihren Ansatz in sicherheitskritischen Domänen an. Aus diesem Grund müssen die natürlichsprachlichen Testfälle auch in einem vorgegebenen Format übergeben werden. Zudem werden nur kurze, eingeschränkte Formulierungen unterstützt. Die Korrektheit der so erzeugten Tests wird von den Autoren nicht direkt vermessen. Stattdessen präsentieren sie in ihrer Publikation eine Fallstudie, in der sie anhand eines Projekts die Abdeckung der automatisch erzeugten Tests mit Tests vergleichen, die manuell von einem Test-Ingenieur erstellt wurden. Die Auswertung zeigt, dass die automatisch erzeugten Tests wesentlich mehr Anwendungsfälle abdecken als die manuell erzeugten Tests.

Das Werkzeug *NAT2TEST* von Carvalho et al. überführt natürlichsprachliche Anforderungsdokumente in Testfälle [Car+15]. Als internen Formalismus zur Darstellung von Abhängigkeiten aufeinander folgender Aktionen des zu erstellenden Testskripts verwendet der Ansatz die von Hoare

formulierte Sprache zur Kommunikation sequenzieller Prozesse (engl. *Communicating Sequential Processes*, kurz *CSP*). Die einzelnen Aktionen und zugehörigen Parameter extrahiert der Ansatz von Carvalho et al. mithilfe thematischer Rollen. Leider enthält die Publikation keine quantitative Untersuchung zur Leistungsfähigkeit des Ansatzes; stattdessen demonstrieren die Autoren lediglich die Verwendung des Werkzeugs anhand ausgewählter Beispiele.

3.4.2 Erzeugung von Datenbankabfragen aus natürlichsprachlichen Äußerungen

Auch wenn es nicht offensichtlich erscheint, herrscht zwischen den Forschungsgebieten der Endnutzer-Programmierung mit natürlicher Sprache und der Erzeugung von Datenbankabfragen aus natürlichsprachlichen Äußerungen eine enge Verwandtschaft. Um Informationen aus Datenbanken abzufragen, müssen ebenfalls programmatische Anweisungen in einer bestimmten Abfragesprache generiert werden; dies ähnelt stark der Erzeugung von API-Aufrufen (wie von den meisten Ansätzen aus dem Gebiet der Programmierung mit natürlicher Sprache durchgeführt). Auch die eingesetzten Techniken ähneln sich[22]; es ist daher nicht verwunderlich, dass viele Ansätze zur Endnutzer-Programmierung mit natürlicher Sprache (insbesondere solche, die semantische Zerteilungen verwenden) unter anderem anhand von Datensätzen evaluiert werden, die ihren Ursprung im Gebiet der Erzeugung von Datenbankabfragen aus natürlichsprachlichen Äußerungen haben, wie beispielsweise *GeoQuery* und *WikiSQL* (siehe Abschnitt 3.3). Die verschiedenen Ansätze aus dem Bereich natürlichsprachlicher Anfragen an Datenbanken ähneln sich hinsichtlich ihrer Zielstellung aufgrund der beschränkten Domäne stark. Unterschieden wird diesbezüglich lediglich dahingehend, ob unterschiedliche Datenbankformate und -arten angesprochen werden können. Die meisten Ansätze fokussieren sich allerdings auf eine konkrete Abfragesprache, wie zum Beispiel *SQL*. Auch hinsichtlich der Mächtigkeit der potenziell unterstützten Primitive der Abfragesprache unterscheiden sich die Ansätze kaum. Folglich unterscheiden sie sich hauptsächlich hinsichtlich der verwendeten Übersetzungstechnik und in der Folge in der Qualität der erzeugten Abfragen.

Eine übliche Technik ist die Verwendung spezieller Zerteilungsgrammatiken (engl. *parser grammar*). Zelle und Mooney trainieren beispielsweise einen spezialisierten Zerteiler (engl. *parser*) namens *CHILL* [ZM96]. *CHILL* entspricht in seiner Funktionsweise im Wesentlichen einem syntaktischen Zerteiler (siehe Abschnitt 2.3.4.6); allerdings werden als Ergebnis der Zerteilung direkt Datenbankabfragen erzeugt. Zelle und Mooney führen zudem den bereits in Abschnitt 3.3 erwähnten *GeoQuery*-Datensatz ein. Dieser basiert auf *GEOBASE*[23], einer Sammlung von Information über die Geographie der Erde, kodiert als *Prolog*-Fakten. Zum Zeitpunkt der Erstellung des *GeoQuery*-Datensatzes umfasste *GEOBASE* 800 *Prolog*-Fakten. Für diese Fakten wurden von 50 Personen 250

[22] Beispielsweise werden in beiden Forschungsgebieten Techniken aus dem Bereich der Informationsrückgewinnung (engl. *information retrieval*) und semantisches Zerteilen eingesetzt; auch die Überführung der natürlichsprachlichen Beschreibung in eine logische (Zwischen-)Repräsentation und anschließende Generierung programmatischer Anweisungen anhand der Repräsentation ist ein in beiden Forschungsgebieten gängiges Vorgehen.

[23] *GEOBASE*: `https://www.elsevier.com/solutions/engineering-village/content/geobase`, zuletzt besucht am 12.02.2021.

passende natürlichsprachliche Anfragen gesammelt[24]. Die Autoren trainieren und evaluieren (bzw. testen) *CHILL* anhand dieses Datensatzes; dabei wird eine Genauigkeit von 84% auf der Testmenge erreicht.

Rao et al. verwenden für ihren Ansatz sogenannte semantische Grammatiken (engl. *semantic grammar*) [Rao+10]. Diese erweitern Grammatiken zur syntaktischen Zerteilung um zusätzliche Nichtterminalsymbole und zugehörige händisch erstellte Produktionsregeln, die domänenspezifische Semantiken kodieren. Mithilfe der Produktionsregeln werden *SQL*-Abfragen direkt aus natürlichsprachlichen Eingaben erzeugt. Die Publikation enthält keine Evaluation des Ansatzes; stattdessen demonstrieren die Autoren die Funktionsweise ihres Ansatzes anhand ausgewählter Beispiele. Ein grundsätzlicher Nachteil des Ansatzes sind die manuell erstellten Produktionsregeln. Im Allgemeinen sind händisch erstellte Regelsätze nur schwierig erweiterbar; zur Erstellung wird ein eingehendes Verständnis der Domäne benötigt und es kann normalerweise nur schwer abgeschätzt werden, ob eine neue Regel tatsächlich eine Verbesserung bewirkt. Ein weiterer Nachteil sowohl dieses Ansatzes als auch des Ansatzes von Zelle und Mooney ist die unmittelbare Abbildung der natürlichsprachlichen Anfragen auf Datenbankabfragen. Dadurch können die Ansätze nur für genau diesen Anwendungsfall verwendet werden; zur Übertragung auf eine andere Domäne (oder nur eine andere Datenbank-Abfragesprache) müssen die Ableitungsregeln neu erstellt bzw. gelernt werden.

Andere Ansätze verwenden daher eine semantische Zwischenrepräsentation der natürlichen Sprache. *C-Phrase*, entwickelt von Minock et al., verwendet Tupelkalküle als logische Zwischenrepräsentation [MON08]. Natürlichsprachliche Eingaben werden von *C-Phrase* zunächst syntaktisch zerteilt und die Syntaxbäume mithilfe von Heuristiken in eine generische Zwischenrepräsentation überführt. Anschließend wird mithilfe eines zweiten Regelsatzes aus der Zwischenrepräsentation eine entsprechende Datenbankabfrage generiert. Zur Evaluation von *C-Phrase* verwenden die Autoren 33 zufällig ausgewählte Beispiele des *GeoQuery*-Datensatzes; dabei ist nicht nur die Aussagekraft aufgrund dieses geringen Umfangs begrenzt, auch die Ergebnisse sind schwer zu deuten. Die Publikation enthält nur eine Grafik und keine Diskussion der erzielten Werte; die Präzision scheint aber bei circa 86% und die Ausbeute bei circa 88% zu liegen. Unabhängig von den Ergebnissen ist die Verwendung einer generischen Zwischenrepräsentation ein wesentlicher Vorteil des Ansatzes; die von Minock et al. verwendete Zwischenrepräsentation erfasst die Semantik der natürlichsprachlichen Anfrage unabhängig von der konkreten Datenbank-Abfragesprache. Das bedeutet, zur Anbindung einer neuen Abfragesprache müssen lediglich neue Regeln für die Abbildung von der Zwischenrepräsentation auf konkrete Datenbankabfragen erstellt werden. Eine Übertragung des Ansatzes auf die Erzeugung von Programmskripten (abseits von Datenbankabfragen) erscheint jedoch nicht möglich: Die Zwischenrepräsentation ist zwar generisch, aber auf Datenbankabfragen spezialisiert; außerdem bleibt offen, wie groß der Aufwand für den zweiten Teil der Abbildung (Zwischenrepräsentation auf konkretes Sprachkonstrukt) tatsächlich ist.

Eine weitere Gruppe von Ansätzen verwendet zur Synthese der Datenbankabfragen Domänenwissen, welches meist als Ontologie repräsentiert wird. Das Werkzeug *PANTO* von Wang et al. erzeugt

[24] Der *GeoQuery*-Datensatz wurde später auf 880 Anfragen erweitert und in dieser erweiterten Variante auch von den anderen in diesem Kapitel diskutierten Arbeiten verwendet. Zur Evaluation des Original-Ansatzes von Zelle und Mooney wurde allerdings nur der ursprüngliche Datensatz verwendet.

(teil-)automatisch aus einer Ontologie ein zugehöriges Lexikon [Wan+07]. Mithilfe des Lexikons können anschließend syntaktische Zerteilungen von natürlichsprachlichen Anfragen interpretiert und in *SPARQL*-Abfragen überführt werden. *PANTO* kann ausschließlich dafür verwendet werden, Informationen aus der Ontologie anzufragen. Wird die Ontologie ausgetauscht, muss lediglich das Lexikon neu generiert werden. Das bedeutet, *PANTO* kann prinzipiell zur Abfrage beliebiger Informationen verwendet werden, solange sich diese in einer Ontologie darstellen lassen. Die Autoren evaluieren *PANTO* unter anderem anhand des *GeoQuery*-Datensatzes. *PANTO* erzielt eine Präzision von 88% und eine Ausbeute von 86%; diese Werte sind vergleichbar mit den deutlich komplexeren (und aufwendigeren) Ansätzen, die semantische Zerteilungen verwenden (siehe Abschnitt 3.3). Unklar ist allerdings, wie groß der Einfluss des Lexikons bei der Interpretation der natürlichsprachlichen Anfragen und wie groß der Aufwand bei einer manuellen Anpassung bzw. Erweiterung des Lexikons ist. Außerdem kann *PANTO* ausschließlich *SPARQL*-Abfragen erzeugen; ob sich ihr Ansatz auch mit anderen (Abfrage-)Sprachen verwenden lässt, lassen die Autoren offen.

Auch *AquaLog*, entwickelt von Lopez et al., erzeugt ein Lexikon aus einer Ontologie [LM04; LPM05; Lop+07]. Allerdings ist *AquaLog* in der Lage, mehrere Wissensbasen mithilfe derselben Ontologie anzufragen. Natürlichsprachliche Anfragen werden in Prädikat-artige *Abfragetripel* übersetzt, mit denen die Ontologie abgefragt werden kann. Das Ergebnis wird dem Nutzer ebenfalls als Tripel präsentiert; ist das Ergebnis falsch oder wurde keine Antwort erzeugt, kann der Nutzer mit dem System interagieren und seine Anfrage anpassen. Durch diese Rückmeldungen des Nutzers passt *AquaLog* sukzessive auch seine Übersetzungsregeln an. Die Autoren evaluieren ihren Ansatz in einem anspruchsvollen Kontext; sie verwenden Ontologien zu zwei Domänen: *Universitätsstrukturen*[25] und *Wein*[26]. Zu beiden Domänen sammeln Lopez et al. in einer Nutzerstudie 69 bzw. 68 Anfragen von Probanden, geben jedoch nur die Domäne, nicht aber die verfügbaren Informationen vor. Dadurch wird sichergestellt, dass die Probanden ihr Anfragen unvoreingenommen stellen; andererseits kann so versucht werden, Informationen zu erfragen, die gar nicht in der Ontologie enthalten sind. Dementsprechend fallen auch die Evaluationsergebnisse aus. *AquaLog* kann zunächst nur 33 der 69 Anfragen zur ersten Domäne komplett richtig beantworten; davon können alleine siebzehn aufgrund fehlender Informationen in der Ontologie nicht beantwortet werden. Durch Interaktion mit den Probanden können zwölf weitere Fragen korrekt beantwortet werden, das heißt insgesamt 65%. Die Ergebnisse für die zweite Domäne fallen schlechter aus: Initial können nur zwölf von 68 Anfragen korrekt beantwortet werden; 35 können aufgrund fehlender Informationen in der Ontologie nicht beantwortet werden. Mit Nutzerinteraktion können immerhin fünf weitere korrekt beantwortet werden, insgesamt somit 25%. Letztlich lassen diese Ergebnisse keine abschließende Bewertung des Ansatzes zu. Die Problemstellung erscheint deutlich schwieriger als bei vergleichbaren Datensätzen (z. B. *ATIS* oder *GeoQuery*); außerdem ist der Datenumfang sehr gering, wodurch die Fragen, die aufgrund fehlender Modellierungen nicht beantwortet werden können, besonders ins Gewicht fallen. Die Autoren selbst attestieren *AquaLog*, dass die Fähigkeiten zur Interpretation der Semantik

[25] Die *AKT Reference Ontology* ist ein Ontologie, die beispielhaft universitäre Organisationsstrukturen, das heißt unter anderem Institute, Lehrstühle und Mitarbeiter, modelliert sind: `http://projects.kmi.open.ac.uk/akt/ref-onto/`, zuletzt besucht am 24.02.2021.

[26] Die *Wine Ontology* ist eine Beispiel-Ontologie zur Domäne *Wein*: `https://www.w3.org/TR/owl-guide/`, zuletzt besucht am 24.02.2021.

natürlichsprachlicher Anfragen noch nicht ausreichend sind, da viele Anfragen (bzw. Anfragetypen) nicht beantwortet werden können, weil das System diese nicht zuverlässig interpretieren kann.

Die Weiterentwicklung *PowerAqua*, ebenfalls entwickelt von Lopez et al., bietet eine verbesserte Sprachanalyse-Komponente und ist zudem in der Lage, automatisch relevante Wissensquellen zu identifizieren, die über das Internet zugänglich sind [LMU06; Lop+12]. Antworten werden gegebenenfalls aus mehreren partiellen Informationen verschiedener Wissensquellen synthetisiert. Diese Problemstellung ist natürlich noch anspruchsvoller als die zuvor für *AquaLog* definierte: Zunächst müssen für eine natürlichsprachliche Anfrage Wissensquellen (das heißt Ontologien) identifiziert werden, die potenziell relevante Informationen enthalten. Anschließend muss die syntaktische Zerteilung der Anfrage auf Konzepte bzw. Individuen der Ontologien abgebildet werden; letztlich müssen die gewonnenen Informationen zu einer Antwort zusammengeführt werden. Für die Evaluation verwenden die Autoren dieses Mal zwei sehr umfangreiche Ontologien: *SWETO*[27] und *DBpedia* (siehe Abschnitt 2.3.15.5). Als Testeingabe verwenden Lopez et al. erneut 69 natürlichsprachliche Anfragen von sieben Nutzern; dieses Mal stellen sie jedoch sicher, dass für alle Anfragen mindestens eine der Ontologien die notwendigen Informationen enthält, um die Anfragen korrekt beantworten zu können. *PowerAqua* gelingt dies in der Evaluation in 48 von 69 Fällen (70% Genauigkeit). *PowerAqua* ist insofern interessant, als es Anfragen anhand von Informationen aus unterschiedlichen Quellen beantworten kann. Allerdings resümieren die Autoren auch für *PowerAqua*, dass die Sprachverständnisfähigkeiten für die Problemstellung (noch) nicht ausreichend sind.

Moderne Ansätze verwenden maschinelles Lernen. Aufgrund der verwendeten Techniken und den teils gleichen Evaluationsdatensätzen, könnten diese auch den Ansätzen, die semantische Zerteilung nutzen (siehe Abschnitt 3.3), zugeordnet werden; die beiden nachfolgend diskutierten Ansätze werden jedoch ausschließlich zur Erzeugung von Datenbankabfragen verwendet, weshalb sie an dieser Stelle eingeordnet werden. Das Werkzeug *Seq2SQL* von Zhong et al. verwendet ein Sequenz-zu-Sequenz-Modell in Form eines neuronalen Netzes zur Erzeugung von *SQL* anhand von natürlichsprachlichen Anfragen [ZXS17]; die genutzte Netzarchitektur ist ein bidirektionales *LSTM* (siehe Abschnitt 2.2.2). Die Besonderheit des Ansatzes liegt im Trainingsmechanismus; die Autoren verwenden eine Art bestärkendes Lernen (engl. *reinforcement learning*, siehe Abschnitt 2.2). Konkret bedeutet das: Zunächst wird auf konventionellem Weg ein Grundmodell gelernt, das Datenbankabfragen erzeugt; die Systemantwort (der Datenbank) wird mit dem Goldstandard verglichen und das Model erhält einen Belohnung (engl. *reward*), falls ein korrektes Ergebnis erzielt wurde. Zum Trainieren und Testen des Modells erstellen die Autoren einen neuen Datensatz, den bereits in Abschnitt 3.3 erwähnten *WikiSQL*-Datensatz. Zhong et al. verwenden eine *SQL*-Datenbankrepräsentation von *Wikipedia* und erzeugen anhand des Schemas und der Einträge zufällige (valide) Datenbankabfragen. Über eine Schwarmauslagerungsplattform (engl. *crowd-sourcing plattform*) lassen sie jeweils einen Probanden eine natürlichsprachliche Beschreibung erstellen; zwei weitere Probanden verifizieren, dass die Beschreibung zur Datenbankabfrage passt. Nach dem Training erzielt der Ansatz auf der Testmenge (dieses Datensatzes) eine Genauigkeit von 48%. Wie viele vergleichbare Ansätze

[27] *SWETO*, kurz für *Semantic Web Technology Evaluation Ontology*: `https://corescholar.libraries.wright.edu/knoesis/741/`, zuletzt besucht am 24.02.2021.

verwenden auch Zhong et al. ein Sequenz-zu-Sequenz-Modell, dass eine direkte Abbildung von natürlichsprachlicher Anfrage als Eingabe auf eine Datenbankabfrage (in einer bestimmten Sprache) erlernt; dadurch muss das Modell neu trainiert werden, sobald neuartige Anfragen gestellt, eine andere Datenbank angesprochen oder eine andere Anfragesprache verwendet werden soll.

Iyer et al. präsentieren einen Ansatz, der mit verhältnismäßig geringem Aufwand an neue Domänen adaptiert werden kann [Iye+17]. Auch ihr Ansatz basiert auf einem bidirektionalem *LSTM*[28]. Sie trainieren allerdings nur ein Grundmodell, das sich automatisch an eine Domäne adaptiert und zusätzlich durch Nutzerrückmeldung verbessert werden kann. Sie demonstrieren diesen Ansatz anhand einer Datenbank, die Informationen über die Domäne *wissenschaftliche Veröffentlichungen* enthält. Das Modell wird lediglich anhand von 1746 synthetischen Ein-/Ausgabebeispielen trainiert, die automatisch anhand von 22 generischen Anfrage-Schablonen und dem Datenbankschema erzeugt werden. Anschließend veröffentlichen sie ihr System (bestehend aus dem vortrainierten Grundmodell und einer Formular-basierten Nutzerschnittstelle) zu Testzwecken über eine Schwarmauslagerungsplattform und lassen Nutzer mit dem System interagieren. Zunächst kann das System nur 25% der natürlichsprachlichen Anfragen korrekt beantworten; zum Ende des Versuchs liegt die Genauigkeit, dank der Nutzerrückmeldungen, bei 65%. Iyer et al. zeigen mit ihrem Ansatz, wie die Adaption maschinell erlernter Modelle an eine neue Domäne mit vertretbarem Aufwand möglich ist. Wie viel Nutzerrückmeldung notwendig ist, um das Modell so weit zu adaptieren, dass es zuverlässig gute Ergebnisse erzielt, bleibt allerdings unklar. Das liegt zum einen daran, dass der Ansatz nur in einem Feldversuch getestet wurde. Zum anderen ist aus der Publikation nicht eindeutig ersichtlich, wie viele Mensch-System-Interaktionen notwendig waren, um das letztendliche Ergebnis zu erzielen; wahrscheinlich wurden aber 200 natürlichsprachliche Anfragen (und etwaige Rückmeldungen der Nutzer zur Korrektheit der Systemantwort) verarbeitet. Unklar ist auch, ob dieser Ansatz auf andere (und größere) Domänen übertragbar wäre; auch die Übertragbarkeit auf die Programmierung (von Skripten) erscheint fraglich: Die Erzeugung des Grundmodells basiert auf Datenbank-Schemata (die bei der einer Abbildung auf Quelltext nicht zur Verfügung stehen) und Anfrage-Schablonen, die auf Datenbankabfragen zugeschnitten sind. Außerdem werden nur einzelne Anweisungen (bzw. Abfragen) erzeugt; ob auch Sequenzen von Instruktionen und Kontrollstrukturen (zur Erzeugung von Programmen) mit diesem Ansatz gebildet werden könnten, wird von Autoren nicht diskutiert, erscheint aber nicht ohne Weiteres möglich.

3.4.3 Virtuelle Assistenten

Virtuelle Assistenten ähneln natürlichsprachlichen Datenbankschnittstellen hinsichtlich Zielsetzung und verwendeter Techniken stark. Virtuelle Assistenten beantworten Fragen des Nutzers, indem sie Informationen aus geeigneten Wissensbasen abfragen oder führen Befehle aus, indem sie die entsprechenden Dienstgeber ansprechen. In beiden Fällen wird die Nutzereingabe semantisch analysiert und in eine Abfragesprache überführt. Die weite Verbreitung von Smartphones sowie

[28] Zunächst weisen sie nach, dass ihre verwendete Netzkonfiguration ähnlich gute Ergebnisse wie vergleichbare Ansätze erzielen kann. Hierzu trainieren sie es Ende-zu-Ende auf den Datensätzen *GeoQuery* und *ATIS* und erzielen Testgenauigkeiten von 85% und 86%.

Tabelle 3.1: Verwandte Assistenzsysteme und Entwicklungs-Plattformen für konversationelle Schnittstellen (oder virtuelle Assistenten) je Unternehmen.

Unternehmen	Virtueller Assistent	Entwicklungs-Plattform
Google LLC	Google Assistant[29]	Dialogflow[30]
Amazon.com, Inc.	Amazon Alexa[31]	Amazon Lex[32]
Apple Inc.	Siri[33]	SiriKit[34]
Microsoft Corporation	Cortana[35]	LUIS[36]
Facebook, Inc.	M[37]	wit.ai[38]
Samsung Electronics	Bixby (VIV)[39]	Bixby Developer[40]
Alibaba Group Holding Limited	AliGenie[41]	AliGenie
Brainasoft	Braina[42]	Inforobo[43]
BlackBerry Limited	BlackBerry Assistant[44]	–
Tencent Holdings Limited	Xiaowei[45]	–
Yandex N.V.	Alice[46]	–
Naver Corporation	Clova[47]	–
Nuance Communications, Inc	–	Nina[48]
IBM	–	Watson Assistant[49], Watson Virtual Agent[50]

technische Fortschritte in Spracherkennung und Sprachverständnis haben dazu geführt, dass alle bedeutenden Technologieunternehmen virtuelle Assistenten entwickeln. Die Tabelle 3.1 listet bekannte virtuelle Assistenzsysteme. Über die jeweilige technische Umsetzung der Systeme ist leider wenig bekannt. Auch die tatsächliche Leistungsfähigkeit lässt sich schwer einschätzen.

[29] Google Assistant: `https://assistant.google.com/`, zuletzt besucht am 24.02.2021.
[30] Dialogflow: `https://cloud.google.com/dialogflow/`, zuletzt besucht am 24.02.2021.
[31] Amazon Alexa: `https://developer.amazon.com/en-US/alexa/`, zuletzt besucht am 24.02.2021.
[32] Amazon Lex: `https://aws.amazon.com/lex/`, zuletzt besucht am 24.02.2021.
[33] Siri: `https://www.apple.com/siri/`, zuletzt besucht am 24.02.2021.
[34] SiriKit: `https://developer.apple.com/siri/`, zuletzt besucht am 24.02.2021.
[35] Cortana: `https://www.microsoft.com/en-us/cortana/`, zuletzt besucht am 24.02.2021.
[36] LUIS: `https://www.luis.ai/`, zuletzt besucht am 24.02.2021.
[37] Facebook M (eingestellt zum 19.01.2018): `https://www.theverge.com/2018/1/8/16856654/facebook-m-shutdown-bots-ai/`, zuletzt besucht am 24.02.2021.
[38] wit.ai: `https://wit.ai/`, zuletzt besucht am 24.02.2021.
[39] Bixby: `https://www.samsung.com/global/galaxy/apps/bixby/`, zuletzt besucht am 24.02.2021.
[40] Bixby Developer: `https://bixbydevelopers.com/`, zuletzt besucht am 24.02.2021.
[41] AliGenie: `https://iap.aligenie.com/`, zuletzt besucht am 24.02.2021.
[42] Braina: `https://www.brainasoft.com/braina/`, zuletzt besucht am 24.02.2021.
[43] Inforobo: `https://inforobo.com/`, zuletzt besucht am 24.02.2021.
[44] BlackBerry Assistant (stark eingeschränkte Funktionalität seit 01.08.2019): `https://www.blackberry.com/us/en/legal/blackberry-assistant`, zuletzt besucht am 24.02.2021.
[45] Xiaowei: `https://xiaowei.qcloud.com/`, zuletzt besucht am 24.02.2021.
[46] Alice: `https://alice.yandex.ru/`, zuletzt besucht am 24.02.2021.
[47] Clova: `https://clova.ai/ko`, zuletzt besucht am 24.02.2021.
[48] Nina: `https://www.nuance.com/omni-channel-customer-engagement/digital/virtual-assistant/nina.html`, zuletzt besucht am 24.02.2021.
[49] Watson Assistant: `https://www.ibm.com/cloud/watson-assistant/`, zuletzt besucht am 24.02.2021.

Die Studie von Lopez et al. vergleicht *Alexa*, *Cortana* und *Google Assistant* unter anderem hinsichtlich der Korrektheit und Natürlichkeit der Antworten [LQG18]. Im Ergebnis lassen sich kaum Unterschiede erkennen, wobei laut der Autoren *Siri* tendenziell mehr korrekte Antworten gibt und die Antworten des *Google Assistant* am natürlichsten wirken. Neben proprietären virtuellen Assistenzsystemen bieten die meisten Unternehmen zusätzlich Entwicklungsplattformen (ebenfalls gelistet in Tabelle 3.1), mit denen eigene Anwendungen an die entsprechende Infrastruktur angebunden werden kann. Das bedeutet, Entwickler können konversationelle Schnittstellen für eigene Softwareprodukte entwickeln. Diese sind dann jedoch nur mit der Infrastruktur des anbietenden Unternehmens kompatibel; mit *SiriKit* von *Apple* entwickelte Produkte können beispielsweise ausschließlich an *Siri* angebunden werden. Der Aufwand zur Entwicklung von konversationellen Schnittstellen auf diesen Plattformen ist erheblich. Um beispielsweise eine Applikation an das von *Google* angebotene *Dialogflow* anzubinden, muss der Entwickler nicht nur natürlichsprachliche Beispieleingaben liefern, sondern für diese zusätzlich den Zweck (engl. *intent*), die auszuführende Aktion und eine Parameter-Abbildung angeben. Die Qualität der erzeugten Schnittstelle hängt dabei stark von der Menge und Diversität der Beispieleingabe ab.

In dem von Guzzoni et al. entwickelten *Active* wird eine Domäne, zum Beispiel *Restaurantbesuche*, als Ontologie modelliert, die eine Baumstruktur bilden [GBC06; GCB06; GBC07]. Zusätzlich werden die Ontologien mit Verarbeitungsregeln versehen, wodurch sie *aktiv* werden. Diese *Active Ontologies* können dann natürlichsprachliche Eingaben verarbeiten, indem Blattknoten auf einzelne Wörter reagieren, die inneren Knoten die Informationen sukzessive zusammenführen und der Wurzelknoten schließlich einen Dienstgeberaufruf erzeugt. Die Logik innerhalb der Knoten muss dabei manuell über *Prolog*-artige Regelsätze definiert werden. Der Aufwand für den Entwickler ist dementsprechend hoch. Das US-Patent Nummer 8677377 [CG05] legt nahe, dass *Active* zumindest zu Beginn den Kern von *Siri* gebildet hat. Keine der Publikationen enthält eine quantitative Untersuchung der Fähigkeiten von *Active*, weshalb nicht eingeschätzt werden kann, ob der von Guzzoni et al. verfolgte Ansatz statthaft ist.

Eine quelloffene und modulare Plattform zur Entwicklung virtueller Assistenten präsentieren Campagna et al. mit *Almond* [Cam+17]. *Almond* besteht aus drei Subsystemen: einem Sprachverständnis-Modul, der Wissensdatenbank *Thingpedia* und der Laufzeitumgebung *ThingSystem*. Entwickler können in *ThingPedia* ausführbare Aktionen beliebiger Zielsysteme inklusive Parameterbelegungen und Bedingungen hinterlegen. *Almond* übersetzt anschließend natürlichsprachliche Eingaben mithilfe dieser Einträge in die System-eigene Sprache *ThingTalk* und bringt das erzeugte Skript auf *ThingSystem* zur Ausführung. Campagna et al. evaluieren *Almond* unter anderem anhand einer Nutzerstudie, die sie über eine Schwarmauslagerungsplattform (engl. *crowd-sourcing plattform*) durchführen. Probanden wurden aufgefordert, Anfragen an einen (imaginären) virtuellen Assistenten zu stellen. Hierzu sollten sich die Probanden in ein alltägliches Szenario hineinversetzen und versuchen, beliebige (zum Szenario passende) Aufgaben zu lösen[51]. Auf diese Weise konnten 327 natürlichsprachliche (textuelle) Äußerungen gesammelt werden. Allerdings wurde ein Großteil (256)

[50] Watson Virtual Agent: `https://www.ibm.com/watson/developercloud/doc/virtual-agent/de/index.html`, zuletzt besucht am 24.02.2021.

[51] Beispielsweise sollten sich die Probanden in die Rolle eines Restaurantmangers oder eines Ladenbesitzers hineinversetzen.

noch vor der Auswertung aussortiert; betroffen waren solche Äußerungen, die (laut der Autoren) keine sinnvollen Anfragen darstellten, nicht vom Funktionsumfang von *Almond* abgedeckt werden (das heißt nicht auf eine in *Thingpedia* hinterlegte Zielsystemfunktion abgebildet werden können) oder nicht im richtigen Format abgegeben wurden[52]. Für die übrigen 71 Anfragen wurden Musterlösungen erstellt. *Almond* kann für 34% dieser Anfragen einen korrekten (*ThingTalk*-)Aufruf erzeugen. Die Autoren resümieren, dass *Almonds* Sprachverständnisfähigkeiten noch nicht ausreichend sind, um frei formulierte Anfragen in einem offenen Szenario zuverlässig zu verarbeiten.

Almonds Nachfolger *Genie*, ebenfalls entwickelt von Campagna et al., setzt genau an diesem Punkt an [Cam+19]: Die Weiterentwicklung des Sprachverständnis-Moduls wird vereinfacht. Hierzu entwerfen die Autoren die sogenannte *Virtual Assistant Programming Language* (kurz *VAPL*). *VAPL* ermöglicht die einfache Definition von Funktionen, die ein bestimmter Service anbietet. Damit *Genie* zuvor definierte Funktionen abrufen kann, müssen zudem (laut der Autoren) nur wenige Beispielanfragen (in Form von Schablonen) und einige Paraphrasierungen je Konstrukt angeben werden. Anhand der Schablonen werden automatisch synthetische natürlichsprachliche Konstrukte erzeugt, die wiederum zum Training eines neuronalen Netzes verwendet werden. Auf diese Weise kann mit geringem Aufwand seitens eines Entwicklers die Sprachverständnis-Komponente an einen neuen Dienst angepasst werden. Leider wird dieser neue Ansatz nicht vergleichend anhand des zuvor (für *Almond*) erstellten Datensatz evaluiert. Stattdessen wird für einen Musik-Dienst demonstriert, wie gut das Sprachverständnismodell anhand synthetischer Eingaben und (manuell erstellter) Paraphrasierungen trainiert werden kann; auf der Testmenge, die aus einem Teil der Paraphrasierungen besteht, wird eine Genauigkeit von 85% erreicht. Bedingt durch den Versuchsaufbau ähneln sich die Anfragen in der Trainings- und Testmenge allerdings stark; eine Nutzerstudie mit echten Anfragen an das System wäre zur Einschätzung der Sprachverständnisfähigkeiten von *Genie* wünschenswert. Die beiden Systeme *Almond* und *Genie* zeigen aber, wie Sprachverständnis und Zielsysteme effektiv entkoppelt werden können, wodurch die Anbindung neuer Systeme mit geringerem Aufwand möglich wird. Die Sprachverständnis-Komponente benötigt nichtsdestotrotz eine Menge von Anfrage-Schablonen und Paraphrasierungen, die je Funktion, die ein Dienstgeber anbietet, manuell erstellt und dem System zur Verfügung gestellt werden müssen. Anhand der publizierten Evaluationen lässt sich nicht abschätzen, wie hoch der tatsächliche manuelle Aufwand ist, um das Sprachverständnis an eine Domäne bzw. ein neues Zielsystem anzupassen.

3.5 Vergleich und Zusammenfassung

Die in den vorangegangen Abschnitten vorgestellten verwandten Arbeiten unterscheiden sich hinsichtlich Zielstellung, Sprachanalysefähigkeiten und Funktionsumfang. Auch im Vergleich mit ProNat zeigen sich Gemeinsamkeiten und Unterschiede. In den Tabellen 3.2, 3.3 und 3.4 (am Ende Abschnitts) werden die verwandten Arbeiten anhand der in Abschnitt 1.2 aufgestellten Zieldefinition

[52] Hierzu zählen beispielsweise Anfragen, die Zeichenketten (zur Beschreibung von Eigennamen) enthielten, die nicht in Anführungszeichen gesetzt wurden.

mit ProNat verglichen. Als Vergleichsgrundlage dient eine Auswahl der ebendort formulierten Ziele, die durch ProNat erreicht werden sollen. Nicht alle zuvor diskutierten Arbeiten fließen in den Vergleich ein. So werden die historischen Systeme, Arbeiten, die nur Teilaufgaben betrachten, wie zum Beispiel die zeitliche Ordnung von Aktionen, sowie alle proprietären Systeme vom Vergleich ausgeschlossen.

Die meisten Ansätze sind für die Verwendung durch Programmier-Laien geeignet. Nur die in Abschnitt 3.2 diskutierten Arbeiten, die sich direkt an Entwickler richten, sowie Ansätze zur automatischen Testerzeugung aus Beschreibungstexten bilden Ausnahmen. Darüber hinaus sind einige Ansätze, die für die Domäne Robotik entwickelt wurden, nicht oder nur eingeschränkt für die Verwendung durch Nicht-Programmierer geeignet. Dies kann über die systemnahe Programmierung und der daraus resultierenden sehr technischen Sprache begründet werden. Mit Ausnahme der Systeme zum gesprochenen Quelltext-Diktat (*Spoken Java* und *VoiceCode*), *APPINITE* und den virtuellen Assistenten *Almond* und *Genie* ist kein Ansatz für die Verwendung gesprochener Sprache geeignet. Konzeptionell lassen sich viele Ansätze zwar auf den Einsatz mit gesprochener Sprache übertragen, zum Beispiel durch die Verwendung eines vorgeschalteten automatischen Spracherkennungssystems. Allerdings betrachtet keine Arbeit die besonderen Herausforderungen gesprochener Sprache im Vergleich zu geschriebener Sprache, wie ungrammatikalische Formulierungen, Füllwörter, Reparaturen oder Wortfehlerkennungen durch ein automatisches Spracherkennungssystem (siehe Abschnitt 2.3.12). Eine eingehende Diskussion dieser Problematik befindet sich in Kapitel 5.

Hinsichtlich der Mächtigkeit der Sprachverständnis-Komponenten unterscheiden sich die Ansätze stark. Zwar ermöglichen mehr als zwei Drittel prinzipiell uneingeschränkte natürlichsprachliche Formulierungen. Allerdings ist anhand der jeweils verwendeten Techniken zur Analyse der natürlichen Sprache anzunehmen, dass die Qualität der Erkennung einzelner Sprachkonstrukte stark schwankt. Außerdem passen einige Ansätze ihre Sprachanalyse-Techniken auf die jeweilige Domäne an. Dadurch verbessert sich zwar die Qualität unter der jeweiligen Zielstellung, die Allgemeingültigkeit wird jedoch eingeschränkt. Bei Ansätzen, die maschinelles Lernen verwenden, hängt die Qualität der Sprachanalyse stark von dem verwendeten (Text-)Korpus ab. Enthält dieses nur wenig unterschiedliche oder stark auf die Domäne fokussierte Formulierungen, ist eine Überanpassung wahrscheinlich.

Bis auf die Systeme *Metafor* und *Pegasus* erzeugen alle ausführbaren Quelltext. Selbst die Ansätze, die Entwickler lediglich durch die automatische Erzeugung von Makros und Programmschnipsel unterstützen, erzeugen diese laut der Autoren so, dass sie direkt verwendet werden können. Eine Überprüfung durch den Entwickler ist allerdings empfehlenswert. Bei den Systemen zur Endnutzer-Programmierung mit natürlicher Sprache ist ein Eingreifen durch den Benutzer nicht vorgesehen, weshalb das Quelltext-Generat ausführbar sein sollte.

Das Erzeugen von komplexen Skripten wird nur von gut der Hälfte der Ansätze unterstützt, wobei die tatsächlich mögliche Komplexität stark schwankt: Einige Ansätze können lediglich zwei Funktionsaufrufe sequenziell verknüpfen, andere erlauben einzelne Kontrollstrukturen und einige sogar geschachtelte Kontrollstrukturen und die Verwendung lokaler Variablen. Bei den Ansätzen zum Quelltext-Diktat ergibt sich die Unterstützung komplexer Programmabläufe nicht aus der Qualität der Ansätze, sondern aus der Übertragung der Komplexität auf den Nutzer. Dadurch, dass der

Nutzer jedwedes Konstrukt programmatisch beschreiben muss, inklusive zum Beispiel dem Beginn und Ende von bedingten Verzweigungsblöcken, sind komplexe Strukturen bei korrektem Diktat möglich. Bei den natürlichsprachlichen Datenbankschnittstellen gehören komplexe Aufrufe schlicht nicht zur Zielstellung, obwohl mehrfache verknüpfte Datenbankabfragen natürlich möglich sind. Ob diese allerdings sinnvoll mit natürlicher Sprache formuliert werden können, ist fraglich. Einzig *PowerAqua* synthetisiert zusammengesetzte (Teil-)Abfragen von unterschiedliche Wissensbasen und erzeugt für den Nutzer eine fusionierte Antwort.

Die wenigsten Arbeiten unterstützen die Definitionen neuer Funktionen; die meisten dienen folglich streng genommen nur der *Steuerung*, nicht aber der *Programmierung* von Computersystemen. Dies gilt insbesondere für alle Ansätze zur Endnutzer-Programmierung mit natürlicher Sprache (mit Ausnahme von *SmartSynth* und einigen Ansätzen aus der Domäne Robotik). Für die Quelltext-Diktat-Ansätze gilt auch hier die gleiche Argumentation wie für die Komposition komplexer Skripte. Dass einige Ansätze, die auf die Verwendung in der Domäne Robotik spezialisiert sind, die Definition neuer Funktionalität unterstützen, unterstreicht die Wichtigkeit der natürlichsprachlichen Programmierung in diesem Bereich.

Nahezu alle Systeme sind (mehr oder weniger) unabhängig von dem konkreten Zielsystem, mit dem sie verwendet werden können. Einzig die Systeme aus der Robotik-Domäne sind bereits konzeptionell stärker an das Zielsystem gebunden und treffen häufig Annahmen über das jeweils verwendete Robotersystem. Lediglich der Ansatz von Markievic et al. erlaubt es, unterschiedliche Robotersysteme unter Verwendung derselben natürlichsprachlichen Schnittstelle anzusprechen. Viele der anderen Ansätze unterstützen ausschließlich einen bestimmten Typ von Zielsystem, zum Beispiel Datenbanken oder grafische Benutzeroberflächen. Mit Ausnahme der Quelltext-Diktat-Ansätze sind die übrigen Arbeiten zwar konzeptionell auf beliebige Zielsysteme übertragbar, müssen aber angepasst werden. Im Fall von Ansätzen, die überwachtes maschinelles Lernen verwenden, bedeutet dies, dass pro Zielsystem ein hinreichend umfangreiches Korpus, inklusive Goldstandard, vorhanden sein muss. Andere Ansätze, wie die von Atzeni und Atzori oder *NLCI*, lassen sich einfach und gegebenenfalls (teil-)automatisch mithilfe von Domänen-Ontologien konfigurieren. Auch Pro*Nat* bildet hier keine Ausnahme und muss ebenfalls mit Domänen-Ontologien konfiguriert werden, die ähnlich zu denen von *NLCI* verwendeten strukturiert sind (siehe Abschnitt 5.4). Allerdings kann Pro*Nat* die Konfiguration zur Laufzeit automatisch vornehmen; das bedeutet, Pro*Nat* kann aus einer Auswahl bereitgestellter Domänen-Ontologien diejenige bestimmen, die am besten zu einer natürlichsprachlichen Äußerung passt (siehe Abschnitt 7.9).

Fast alle Ansätze sind auf die Verwendung mit einer Programmiersprache festgelegt; einige Autoren argumentieren, dass sich ihr Ansatz (konzeptionell) auf andere Programmiersprachen übertragen lässt, ein Nachweis wird jedoch selten erbracht. Für die Ansätze, die überwachte Lernmethoden verwenden, gilt ähnliches wie für die Zielsystemunabhängigkeit, da diese in den meisten Fällen mit der verwendeten Programmiersprache zusammenhängt. Das bedeutet, ein Ansatz, der Abbildungen von natürlicher Sprache auf ein Robotersystem, programmiert in *Java*, lernen kann, kann dies prinzipiell auch für ein Hausautomationssystem, programmiert in *C*, sofern die entsprechenden Lerndaten zur Verfügung stehen (was normalerweise nicht der Fall ist). Die natürlichsprachlichen Datenbankschnittstellen unterstützen wiederum ausschließlich Datenbanksprachen. Einige davon

unterstützen nur bestimmte Sprachen, andere konzeptionell alle. *AquaLog* und *PowerAqua* hingegen funktionieren durch die Verwendung von Abfragetripeln unabhängig von der jeweiligen Datenbanksprache. ProNat ist nahezu völlig unabhängig von der verwendeten Programmiersprache. Die voneinander unabhängige Erzeugung des abstrakten Syntaxbaums, der Zielsystemfunktionsabbildung und der Programmiersprachendefinition, ermöglicht die Anbindung neuer Programmiersprachen mit geringerem Aufwand (siehe Kapitel 5 und 8).

Zusammenfassend lässt sich festhalten, dass keine der verwandten Arbeiten die mit ProNat verfolgten Ziele umsetzt. Zwar sind die meisten Systeme für eine Verwendung durch Programmier-Laien geeignet, jedoch unterscheiden sich die Ansätze hinsichtlich der Qualität der Sprachanalysen, was die Formulierung, die ein Nutzer verwenden kann, unterschiedlich stark einschränkt. Die Verwendung von gesprochener Sprache (als vorrangiges Eingabemedium) zur Endnutzer-Programmierung wird von keiner verwandten Arbeit unterstützt. Zwar erzeugen nahezu alle Ansätze ausführbaren Quelltext, aber nur gut die Hälfte ermöglicht die Formulierung komplexer Abläufe; vor allem moderne Ansätze zur Endnutzer-Programmierung, die semantische Zerteilungen bzw. maschinelles Lernen verwenden, sind größtenteils nur zur Erzeugung einzelner Funktionsaufrufe geeignet. Nur einige sehr wenige Ansätze betrachten die Definition neuer Funktionen (und damit die eigentliche Programmierung). Während die meisten Ansätze zumindest konzeptionell vom konkreten Zielsystem unabhängig sind, legen sich die meisten auf eine konkrete Programmiersprache fest.

Tabelle 3.2: Vergleich von ProNat mit verwandten Arbeiten anhand von Kriterien, die ProNat erfüllen soll: Ein ausgefüllter Kreis symbolisiert (•), dass der jeweilige Ansatz das Kriterium vollumfänglich erfüllt, und ein leerer Kreis (○), dass das Kriterium teilweise (bzw. mit Einschränkungen) erfüllt wird. Ist keine Markierung vorhanden, erfüllt der Ansatz das Kriterium nicht.

System	Laien-geeignet	gesprochene Sprache	uneingeschränkte Sprache	ausführbarer Quelltext	komplexe Skripte	neue Funktionen	Zielsystem-unabhängig	Programmiersprachen-unabhängig
Entwicklungsunterstützung – Quelltext-Diktat								
NaturalJava [Pri+00]				•	•	•	•	
Spoken Java [Beg04; BG05]		•		•	•	•	•	
VoiceCode [DFN06]		•		•	•	•	•	•
Entwicklungsunterstützung – Entwurfsunterstützung								
Metafor [LL05a; LL05b]	•		•		•	○	•	
NLP for NLP [MLL06]	•		•		•	○	•	
ProNat	•	•	•	•	•	•	•	•

Tabelle 3.3: Vergleich von ProNat mit verwandten Arbeiten anhand von Kriterien, die ProNat erfüllen soll (erste Fortsetzung): Ein ausgefüllter Kreis symbolisiert (•), dass der jeweilige Ansatz das Kriterium vollumfänglich erfüllt, und ein leerer Kreis (○), dass das Kriterium teilweise (bzw. mit Einschränkungen) erfüllt wird. Ist keine Markierung vorhanden, erfüllt der Ansatz das Kriterium nicht.

System	Laien-geeignet	gesprochene Sprache	uneingeschränkte Sprache	ausführbarer Quelltext	komplexe Skripte	neue Funktionen	Zielsystem-unabhängig	Programmiersprachen-unabhängig
Entwicklungsunterstützung – Makros und Programmschnipsel								
Gvero und Kuncak [GK15]				•			•	
Richardson und Kuhn [RK17]				•			•	
NLyze [GM14]	○		○	•		•		
SWIM [RWH16]				•	•		•	
Gu et al. [Gu+16]				•	•		•	
Voxelrun [Wan+17]	○			•		•	•	
Endnutzer-Programmierung – Syntax-/Wissens-basiert								
Pegasus [KM06]	•		•				•	•
Little und Miller [LM06]	•			•			○	
SmartSynth [LGS13]	•		•	•		•		
Desai et al. [Des+16]	•		•	•			○	○
NLCI [LWT17b]	•		•	•	•		•	•
Atzeni und Atzori [AA18]	•		•	•			•	•
Endnutzer-Programmierung – Semantisches Zerteilen								
Vadas und Curran [VC05]	•		○	•	•		•	
Long et al. [LPL16]	•		•	○	○		○	
Guu et al. [Guu+17]	•		•	○	○		○	
Suhr und Artzi [SA18]	•		•	○	○		○	
Quirk et al. [QMG15; BQ16]	•		•	•	○		•	
Chen et al. [CSH18]	•		•	○			○	○
Ling et al. [Lin+16]	•		•	•	•		○	○
Yin und Neubig [YN17]	•		•	•	•		○	○
Rabinovich et al. [RSK17]	•		•	•	•		○	○
Dong und Lapata [DL18]	•		•	•			○	•
NL2Bash [Lin+18]	•		•	•				
ProNat	•	•	•	•	•	•	•	•

Tabelle 3.4: Vergleich von ProNat mit verwandten Arbeiten anhand von Kriterien, die ProNat erfüllen soll (zweite Fortsetzung): Ein ausgefüllter Kreis symbolisiert (•), dass der jeweilige Ansatz das Kriterium vollumfänglich erfüllt, und ein leerer Kreis (○), dass das Kriterium teilweise (bzw. mit Einschränkungen) erfüllt wird. Ist keine Markierung vorhanden, erfüllt der Ansatz das Kriterium nicht.

System	Laien-geeignet	gesprochene Sprache	uneingeschränkte Sprache	ausführbarer Quelltext	komplexe Skripte	neue Funktionen	Zielsystem-unabhängig	Programmiersprachen-unabhängig
Endnutzer-Programmierung – Robotik-Domäne								
Tellex et al. [Tel+11]	○		○	•				
Lincoln und Verres [LV12]			○	•	•	•		
Mutuszek et al. [Mat+13]	•		•	•				
She et al. [She+14]	○			•	•	•		
Thomason et al. [Tho+15]	•		•	•				
Misra et al. [Mis+15]	•		•	•	•			
Markievic et al. [Mar+17]	•		•	•	•	•	○	○
Endnutzer-Programmierung – Kombinierte Ansätze								
Manshadi et al. [MGA13]	•		•	•	○	•	○	○
APPINITE [Li+18]	•	○	•	•	•	•		
Angrenzend – Test-Erzeugung								
Thum. et al. [Thu+12; Thu+13]			○	•	•		○	○
Wang et al. [Wan+15]				•	•		○	
NAT2TEST [Car+15]			○	•	•		○	○
Angrenzend – Natürlichsprachliche Datenbankschnittstellen								
Zelle und Mooney [ZM96]	•		•	○				
Rao et al. [Rao+10]	•		•	○				
C-Phrase [MON08]	•		•	•			○	○
PANTO [Wan+07]	•		•	•			○	
AquaLog [LM04; LPM05; Lop+07]	•		•	•			○	○
PowerAqua [LMU06; Lop+12]	•		•	•			•	○
Seq2SQL [ZXS17]	•		•	•				
Iyer et al. [Iye+17]	•		•	•			○	
Angrenzend – (Quelloffene) virtuelle Assistenten								
Almond [Cam+17]	•	○	•	•	○		○	
Genie [Cam+19]	•	○	•	•	○		○	
ProNat	•	•	•	•	•	•	•	•

4 PARSE – Eine agentenbasierte Architektur für tiefes Sprachverständnis

„Data is not information, information is not knowledge,
knowledge is not understanding, understanding is not wisdom.“

– Clifford Stoll

In dieser Arbeit wird ein neuartiger Ansatz zur Programmierung mit natürlicher Sprache präsentiert und als konkretes System umgesetzt (siehe Kapitel 5). Das System ProNat soll die bekannten Einschränkungen verwandter Ansätze überwinden. Insbesondere soll ProNat ein tieferes Verständnis natürlicher Sprache ermöglichen und dadurch die Möglichkeit bieten, auch komplexe Programmstrukturen aus gesprochenen Äußerungen zu synthetisieren. Außerdem soll ProNat weitestgehend domänenunabhängig agieren und dadurch mit geringem Aufwand auf neue Zielsysteme und Umgebungen adaptiert werden können (siehe Abschnitt 1.2).

Um diese Zielstellungen umsetzen zu können, wurde als Grundlage für ProNat die Architektur PARSE (kurz für **P**rogramming **Ar**chitecture for **S**poken **E**xplanations) entworfen [WT15]. Die Architektur ist agentenbasiert und ermöglicht die Entwicklung von Systemen für tiefes Sprachverständnis gesprochener Äußerungen. Ein tiefes Verständnis natürlicher Sprache ist die Voraussetzung, um Programmierung mit gesprochener Sprache zu ermöglichen (siehe Abschnitte 2.3.11 und 2.3.12).

Verwandte Ansätze zur Programmierung mit natürlicher Sprache stellen fast ausnahmslos Speziallösungen dar (siehe Abschnitt 3.3); das bedeutet, sie wurden zur Lösung einer bestimmten Problemstellung entwickelt und sind für diese optimiert (wie z. B. die Funktionserweiterung eines intelligenten Assistenzsystems [Li+18]). Die Umsetzung erfolgt üblicherweise monolithisch; das bedeutet, es wird genau eine Technik verwendet, um die Problemstellung zu lösen: Ausgehend von der natürlichsprachlichen Eingabe analysieren derartige Systeme entweder regel-, statistik- oder wissensbasiert die Sprache und generieren ein Ergebnis, z. B. Aufrufe an eine Web-API. Die syntaktische Analyse und semantische Interpretation der Sprache sowie die daraus abgeleitete Synthese programmatischer Strukturen werden als eine zusammenhänge Analyse modelliert[1]. Dadurch sind derartige Ansätze nur schwer erweiter- und änderbar und können nur im Ganzen evaluiert werden. Damit ist auch eine zielgerichtete bzw. punktuelle Verbesserung der Analyseergebnisse

[1] Einige Ansätze verwenden zwar auch Kombinationen (z. B. eine regelbasierte Nachverarbeitung einer statistikbasierten Klassifikation), eine Systematisierung der Kombinationsmöglichkeiten findet aber nicht statt.

kaum möglich. Die Spezialisierung verwandter Ansätze bringt ein weiteres Problem mit sich. In die durchgeführten Sprachanalysen fließt häufig Wissen über die Anwendungsdomäne ein. Zusammen mit dem zumeist monolithischen Aufbau hat dies zur Folge, dass unklar ist, inwiefern sich derartige Ansätze auf andere Domänen übertragen lassen und welchen Aufwand dies nach sich ziehen würde.

Der wesentliche Unterschied der Architektur PARSE gegenüber bestehenden Ansätzen liegt in der starken Modularisierung. PARSE ermöglicht die Verwendung von (voneinander unabhängigen) Agenten zur Analyse der Semantik natürlichsprachlicher Äußerungen. Dadurch ist es möglich, für einzelne sprachliche Phänomene Agenten einzusetzen (beispielsweise zur Analyse von sprachlichen Referenzen, Konditionalsätzen oder Bedeutungen einzelner Wörter im Kontext). Durch die Verwendung unabhängiger Agenten kann zudem jeweils ein anderer Analyseansatz gewählt werden. Damit ermöglicht PARSE die Entwicklung von Systemen, die eine Kombination von sowohl statistik-, regel- als auch wissensbasierten Analysen durchführen. Die Kombination der Analyseergebnisse wird von der Architektur vorgenommen. Durch das Zusammenspiel unterschiedlicher, spezialisierter Agenten kann ein tieferes Verständnis natürlicher Sprache erreicht werden als es mit monolithischen Ansätzen möglich ist. PARSE ermöglicht zudem die Trennung von Modulen zur Sprachanalyse und der Anwendungsdomäne (z. B. die API einer konkreten Anwendung), indem eine Schnittstelle zu Domänenwissen angeboten wird. Durch diese Trennung können auf PARSE basierende Systeme einfach adaptiert werden; es muss lediglich das Domänenwissen ausgetauscht werden; die Sprachanalysen können wiederverwendet werden. Ein weiterer Vorteil der Modularisierung ist, dass einzelne Agenten unabhängig voneinander evaluiert, adaptiert und ausgetauscht werden können.

Die Definition von Teilproblemen und die damit einhergehende Modularisierung von Werkzeugen ist ein etabliertes Vorgehen in der Computerlinguistik. Die Verwendung von Fließbändern zur Analyse der Syntax natürlicher Sprache ist ein häufig angewandter Ansatz (siehe Abschnitt 2.3.4). Allerdings werden Fließbänder vorrangig zur Analyse von Textdokumenten verwendet; Ansätze zur Analyse von gesprochener Sprache sind hingegen weitestgehend monolithisch. Zudem beschränken sich die meisten Computerlinguistikfließbänder auf die Analyse syntaktischer Eigenschaften. Die Schwierigkeit bei der Analyse der Semantik ist, dass zwar auch für diese Teilaufgaben definiert werden können. Allerdings ist im Allgemeinen nicht klar, welche Reihenfolge für die einzelnen Analysen gewählt werden sollte. Vorarbeiten haben zudem gezeigt, dass zirkuläre Abhängigkeiten zwischen Analysen bestehen [LWT17b]. Daher verwendet PARSE zur Analyse der Semantik keine Fließbänder. Stattdessen führen die Agenten ihre Analysen nebenläufig aus. Dadurch können Agenten Teilergebnissen anderer Agenten für ihre eigenen Analysen verwenden und ihrerseits neue Analysegrundlagen schaffen. Die eigentliche Problemstellung (z. B. Programmierung mit gesprochener Sprache) wird durch das Zusammenspiel der Agenten gelöst.

Die Architektur PARSE ermöglicht die Entwicklung beliebiger Systeme zur Analyse gesprochener Sprache. Das bedeutet, PARSE ist nicht auf eine bestimmte Problemstellung festgelegt. Neben der Verwendung für die Programmierung mit gesprochener Sprache, wie in Kapitel 5 beschrieben, können auch andere Anwendungen, die ein tiefes Verständnis natürlicher Sprache voraussetzen, mit PARSE umgesetzt werden. Auf Grundlage von PARSE werden neben ProNat bereits zwei weitere Systeme entwickelt. Zum einen wird ein Ansatz umgesetzt, der anhand semantischer Modelle für Anforderungstexte und zugehörige Quelltexte Rückverfolgbarkeitsinformationen generiert [Hey19].

Zum anderen entsteht derzeit ein System, welches die Konsistenz von Modellbeschreibungen und konkreten Softwarearchitekturen prüft [KK19; KSK19].

In den nachfolgenden Abschnitten werden zunächst die Anforderungen an die Architektur und die daraus abgeleiteten Design-Prinzipien diskutiert (Abschnitt 4.1). Anschließend wird der Aufbau von PARSE beschrieben und die Verwendungsmöglichkeiten der einzelnen Bestandteile dargelegt (siehe Abschnitt 4.2).

4.1 Anforderungen und Design-Prinzipien

Der Entwurf der Architektur PARSE unterliegt vorab definierten Design-Prinzipien. Die Prinzipien werden aus Anforderungen an die Architektur bzw. potenzielle Systeme, die auf PARSE basieren, abgeleitet. Die Anforderungen sind Präzisierungen der in Abschnitt 1.2 definierten Zielstellung bzw. daraus folgende Qualitätseigenschaften. Im Folgenden werden zunächst die Anforderungen und anschließend die daraus resultierenden Prinzipien diskutiert.

4.1.1 Anforderungen

Auf Grundlage der Architektur PARSE soll es möglich sein, Systeme zu entwickeln, die ein *tiefes Verständnis natürlicher Sprache* ermöglichen[2]. Auf PARSE basierende Systeme sollen zudem für *beliebige Anwendungsgebiete* eingesetzt werden können; ein Anwendungsgebiet stellt die Programmierung mit gesprochener Sprache dar. Die Architektur soll so gestaltet werden, dass darauf aufbauende konkrete Systeme mit möglichst *geringem Aufwand erweitert* werden können. Um dies zu ermöglichen, sollten die einzelnen Bestandteile möglichst *unabhängig voneinander agieren* und dementsprechend auch weitestgehend unabhängig voneinander *getestet und evaluiert* werden können. Diese Anforderungen werden nachfolgend präzisiert.

Tiefes Verständnis natürlicher Sprache

Menschen kommunizieren über ihre Muttersprache ganz selbstverständlich untereinander. Kommunizieren Nutzer hingegen über eine Sprachschnittstelle mit einem Computersystem (z. B. mit einem virtuellen Assistenzsystem), lässt sich beobachten, dass sie ihr Sprachverhalten verändern [PFS17]. Beispielsweise versuchen Nutzer, sich besonders deutlich zu artikulieren, verwenden einfache Wörter, äußern nur kurze Aussagen (bzw. Anfragen) oder benutzen einfache Satzstrukturen. Der Grund hierfür ist, dass Nutzer sich an das System anpassen; aus Erfahrungen im Umgang mit dem jeweiligen System haben sie gelernt, wie sie Anfragen formulieren müssen, damit das System sie versteht. Zwar gelingt es Nutzern auf diese Weise das gewünschte Systemverhalten zu erzielen, die Kommunikation mit dem System gestaltet sich jedoch umständlich und unnatürlich. Die Nutzungserfahrung ist dementsprechend nicht optimal und die Bereitschaft die Sprachschnittstelle zu verwenden sinkt möglicherweise.

[2] Wie in Abschnitt 2.3.11 eingeführt, ist damit ein *umfassendes* Verständnis natürlicher Sprache gemeint. Das bedeutet, es sollen sowohl in der *Breite* möglichst viele sprachliche Konstrukte analysiert als auch in der *Tiefe* möglichst genau und aussagekräftig beschrieben werden können.

Um die Nutzungserfahrung zu verbessern, sollten Nutzer sich nicht an das System anpassen müssen. Vielmehr sollte das System Nutzer so verstehen, wie es ein menschliches Gegenüber könnte. Dementsprechend sollte ein System für tiefes und allgemeines Verständnis natürlicher Sprache die Ausdrucksformen nicht einschränken. Die Möglichkeit Äußerungen frei zu formulieren bedeutet insbesondere, nicht an ein bestimmtes Vokabular gebunden zu sein. PARSE muss es daher ermöglichen, Sprachanalysen zu entwickeln, die über die Erkennung von Signalwörtern hinausgehen. Auch hinsichtlich der Struktur von Äußerungen sollten Nutzer nicht eingeschränkt werden. Das bedeutet, auf PARSE basierende Systeme müssen in der Lage sein, die Semantik einer Äußerung unabhängig von der syntaktischen Struktur zu analysieren. Es sollte also beispielsweise keinen Unterschied machen, ob ein Nutzer eine Anfrage im Aktiv oder Passiv formuliert oder ob ein Modifikator durch ein Adjektiv ausgedrückt wird oder in einem Nebensatz erklärt wird[3]. Die Länge einer an das System gerichteten Äußerung sollte prinzipiell unbeschränkt sein[4]. Eine zugrundeliegende Architektur muss dementsprechend die Verarbeitung von Eingaben beliebiger Länge ermöglichen. Auch bezüglich des Inhalts sollten Nutzer nicht eingeschränkt werden. Es sollte beispielsweise möglich sein, Rückbezüge aufzustellen und Bedingungen zu formulieren. Eine Architektur sollte die Analyse und Darstellung komplexer und (sprachlich) weit entfernter Zusammenhänge ermöglichen. Letztlich sollten Nutzer möglichst spontan kommunizieren können. Spontane gesprochene Sprache schließt aber das potenzielle Auftreten von Versprechern, Verzögerungslauten und Selbstkorrekturen ein. Auch hierfür muss eine Architektur Darstellungsformen und Analysemöglichkeiten bieten.

Trennung von Sprachanalysen und der Anwendungsdomäne

Systeme für Endnutzer-Programmierung mit natürlicher Sprache sind meist auf eine Anwendungsdomäne zugeschnitten (siehe Abschnitt 3.3). Beispielsweise ermöglichen sie die Programmierung eines speziellen Robotersystems, die interaktive Erweiterung eines virtuellen Assistenten oder die Erzeugung von Makros für *Microsoft Excel*. Die Ansätze, die derartigen Systemen zugrunde liegen, sind meist nicht auf andere Anwendungsdomänen übertragbar; sie stellen damit unflexible Speziallösungen dar. Eine Ursache hierfür ist, dass die meisten Ansätze Informationen über die Domäne in die Analyse der Sprache einfließen lassen. Dies gilt auch für statistikbasierte Ansätze; die probabilistischen Modelle werden anhand domänenspezifischer Daten erlernt. Daher ist unklar, inwieweit diese übertragbar sind.

Um ein System zur Programmierung mit natürlicher Sprache auf andere Anwendungsdomänen übertragbar zu machen, sollte die Analyse der natürlichsprachlichen Eingaben konzeptionell so weit wie möglich von der Anwendung in einer konkreten Domäne getrennt werden. Das bedeutet, die zugrundeliegende Architektur sollte so aufgebaut sein, dass Module, die sprachliche Analysen durchführen, unabhängig von der Anwendungsdomäne entwickelt werden können. Gleichzeitig muss die Architektur einen wohldefinierten Zugang zu Domänenwissen ermöglichen (z. B. Wissen über

[3] Ein Großteil der verwandten Ansätze kann nur Eingaben verarbeiten, die eine gewisse syntaktische Struktur aufweisen (siehe Kapitel 3). Einige fordern sogar Eingaben, die vorgeschriebenen Schablonen entsprechen, wie die sogenannten *Wenn-Dann-Andernfalls-Rezepte* [LGS13; QMG15; BQ16]. Viele Zusammenhänge lassen sich aber nicht in wenigen Worten oder in festgelegten Schablonen formulieren.

[4] Die meisten verwandten Ansätze begrenzen entweder die Länge der Eingabe oder schränken die Menge der verwendbaren Formulierungen stark ein (siehe Kapitel 3).

die Funktionen einer API), da anderenfalls keine konkreten Verfahren für konkrete Anwendungsdomänen entwickelt werden können. Durch die Trennung von Sprachanalyse und Domäne können Sprachanalysen weitestgehend wiederverwendet werden, wenn sich die Anwendungsdomäne ändert. Auf Architekturebene kann die Trennung jedoch nur ermöglicht, nicht aber erzwungen werden. Wie Domänen-agnostisch ein System entwickelt wird, obliegt der Umsetzung des konkreten Systems[5].

Erweiterbarkeit

Das tiefe und allgemeine Verständnis natürlicher Sprache stellt eine anspruchsvolle Aufgabe dar. Welche Analyseschritte nötig sind, um die Semantik einer Äußerung zu erfassen, kann vorab nur schwer definiert werden. Dies gilt insbesondere, wenn – wie zuvor diskutiert – eine Architektur entstehen soll, welche die Umsetzung von Systemen erlaubt, die in unterschiedlichen Anwendungsdomänen eingesetzt werden können. Unterschiedliche Anwendungsdomänen können verschiedene Anforderungen an die erforderlichen Sprachanalysen stellen; dementsprechend kann es bei der Anpassung auf eine Anwendungsdomäne nötig sein, Analysen zu verfeinern oder neue spezialisierte Analyseschritte hinzuzufügen. Sprachanalysen sollten also (sukzessive) verfeinert und neue Analyseschritte sollten mit geringen Auswirkungen auf das Gesamtsystem hinzugefügt werden können. Das bedeutet, dass Systeme möglichst einfach erweiterbar sein sollten. Diese Erweiterbarkeit sollte bereits von der zugrundeliegenden Architektur unterstützt werden.

Unabhängigkeit

Teilschritte von Sprachanalysen sollten möglichst unabhängig voneinander sein. Dadurch können unterschiedliche Ansätze pro Teilschritt gewählt werden. Die meisten verwandten Arbeiten setzen einen monolithischen Ansatz um, der entweder regel-, statistik- oder wissensbasiert ist (siehe Kapitel 3). Wird ein System hingegen in unabhängige Teilschritte untergliedert, kann für jeden Teilschritt ein anderer Ansatz gewählt werden. Auch die zuvor diskutierte Erweiterbarkeit von Sprachanalysesystemen lässt sich durch eine hohe Unabhängigkeit der Einzelbestandteile einfacher umsetzen. Zudem können Systeme so nicht nur erweitert werden, die Umsetzung einzelner Teilschritte kann auch durch einen anderen Ansatz ersetzt werden, ohne dass andere Bestandteile beeinflusst werden. Um Systeme entwickeln zu können, die aus unabhängigen Teilschritten bestehen, sollte eine zugrundeliegende Architektur die Verwendung abgeschlossener, unabhängiger Module anbieten und die Kommunikation zwischen diesen definieren.

Evaluierbarkeit

Aufgrund der herausfordernden Aufgabe der umfassenden Analyse der Semantik sollten Systeme für tiefes Verständnis natürlicher Sprache kontinuierlich evaluiert werden. Dabei sollte nicht nur die Qualität des Gesamtsystems evaluierbar sein; stattdessen sollte die zugrundeliegende Architektur die Test- und Evaluierbarkeit auf unterschiedlichen Granularitätsebenen unterstützen; neben dem Gesamtsystem sollten auch unabhängige Teilschritte – wie im vorangegangenen Paragraphen beschrieben – sowie Kombinationen dieser evaluiert und getestet werden können. Dies erfordert auch die Architektur-seitige Unterstützung, manuell erzeugte Zwischenergebnisse (das heißt Musterlösungen) Teilschritten als Analysegrundlage (das heißt als Eingabe) zur Verfügung zu stellen.

[5] Gegebenenfalls kann die Entwicklung domänenabhängiger Sprachanalysen auch vorteilhaft sein, etwa zur Steigerung der Effizienz oder der Genauigkeit des Gesamtsystems.

4.1.2 Prinzipien

Für den Entwurf der Architektur PARSE werden Design-Prinzipien formuliert. Diese leiten sich aus den zuvor diskutierten Anforderungen an die Architektur bzw. konkrete Systeme, die auf Basis der Architektur entwickelt werden sollen, ab. Dementsprechend setzt jedes Design-Prinzip eine oder mehrere Anforderungen um. Nachfolgend werden die Prinzipien erläutert und gegebenenfalls Bezüge zu den entsprechenden Anforderungen hergestellt.

- *Modularität*:
  PARSE wird stark modularisiert aufgebaut. Das bedeutet, die Architektur wird es ermöglichen, unterschiedliche Sprachanalysen in einzelnen Modulen umzusetzen. Einzelne Module können unabhängig voneinander entwickelt und evaluiert werden. Die Modularität ermöglicht es zudem Systeme nachträglich zu erweitern. Auch die Informationen über die Anwendungsdomäne können in einem eigenen Modul gekapselt werden; dadurch werden die Sprachanalysen konzeptionell von konkreten Anwendungsdomänen getrennt.

- *Parallelität*:
  PARSE wird es ermöglichen, Sprachanalysen nebenläufig zueinander auszuführen. Die parallele Ausführung semantischer Analysen ermöglicht ein tieferes Verständnis natürlichsprachlicher Äußerungen. Im Gegensatz zur Analyse der Syntax (siehe Abschnitt 2.3.4) kann die optimale Reihenfolge der Analysen zur Untersuchung der Semantik natürlichsprachlicher Äußerungen nicht allgemeingültig bestimmt werden. Durch die parallele Ausführung der Analysen können Teilergebnisse generiert und Teilergebnisse anderer Analysen verwendet werden. Analyseergebnisse können somit verfeinert und zyklische Abhängigkeiten aufgelöst werden[6].

- *Definierte Kommunikation:*
  PARSE ermöglicht den Austausch von Informationen zwischen den Modulen. Um die Module möglichst unabhängig voneinander gestalten zu können, ist jedoch keine direkte Kommunikation vorgesehen. Stattdessen werden Informationen über eine geteilte Datenstruktur ausgetauscht: Module hinterlegen ihre Analyseergebnisse in der Datenstruktur und können ihrerseits Ergebnisse anderer Agenten verwenden indem sie diese auslesen. Eine Datenstruktur, die auf einem wohldefinierten Modell basiert, ermöglicht es zudem, Musterlösungen für Analyseergebnisse zu erstellen und diese als Evaluationsgrundlage zu nutzen, entweder zum Vergleich mit durch Sprachanalysen erzeugten Ergebnissen oder als Grundlage zur Durchführung anderer Sprachanalysen (die auf die Ergebnisse anderer Module angewiesen sind). Das für PARSE verwendete Datenmodell soll die Darstellung beliebiger Analyseergebnisse ermöglichen; das schließt insbesondere auch die Darstellung komplexer, abstrakter und impliziter sprachlicher Zusammenhänge ein.

[6] Beispielsweise könnte ein Teil einer Analyse *A* durchgeführt werden. Die Ergebnisse von *A* bilden die Grundlage für eine zweite Analyse *B*. Die Ergebnisse von *B* wiederum sorgen dafür, dass *A* weitere Analyseschritte durchführen kann. Derartige Verschränkungen können beliebig viele Analysen betreffen und wiederholt auftreten.

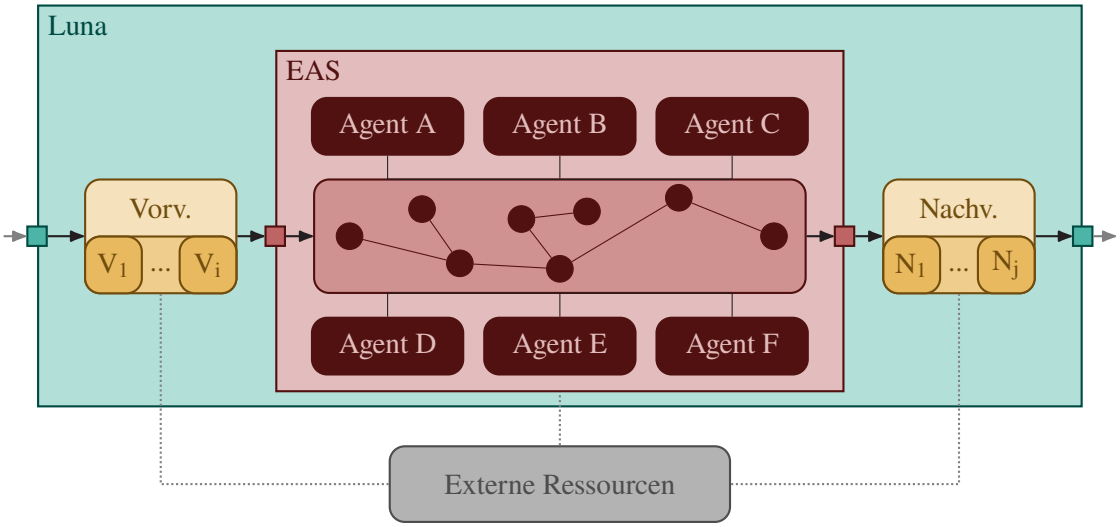


Abbildung 4.1: Struktureller Aufbau der Architektur PARSE

4.2 Aufbau der Architektur PARSE

Der Entwurf der Architektur PARSE beruht auf den zuvor beschriebenen Design-Prinzipien. Strukturell setzt sich PARSE grundlegend aus drei Bestandteilen zusammen:

1. einem Vorverarbeitungsfließband
2. der Einheit zur Analyse der Semantik (kurz *EAS*) und
3. einem Nachverarbeitungsfließband.

Diese Komponenten sind eingebettet in die Rahmenarchitektur (engl. *framework*) [BHS07] Luna (**L**anguage **Un**derstanding **A**rchitecture). Sämtliche Module, die zur Analyse von natürlicher Sprache dienen, das heißt die Stufen der beiden Fließbänder und die Agenten der *EAS*, werden als Einschübe (engl. *Plug-Ins*) umgesetzt. Die Rahmenarchitektur sorgt dafür, dass die Einschübe geladen und innerhalb der jeweiligen Komponente ausgeführt werden; die Fließbandstufen werden sequenziell in festgelegter Reihenfolge ausgeführt, die Module der *EAS* hingegen parallel. Die Rahmenarchitektur stellt darüber hinaus definierte Schnittstellen für den Zugriff auf externe Ressourcen, wie etwa eine Repräsentation der Anwendungsdomäne, zur Verfügung. Konkrete Systeme zur Analyse natürlicher Sprache können auf Basis von PARSE entwickelt werden, indem der Rahmenarchitektur Einschübe für die Fließbänder und die *EAS* sowie die benötigten externen Ressourcen zur Verfügung gestellt werden. Abbildung 4.1 stellt den Aufbau der Architektur schematisch dar.

Für die einzelnen Komponenten sieht die Architektur PARSE in konkreten Systemen folgende Verwendungen vor: Vorverarbeitungsfließbänder dienen zur Anwendung von grundlegenden lexikalischen und syntaktischen Analysen. Somit sollen Fließbänder natürliche Sprache (in Textform oder als Audiosignal) entgegennehmen und soweit vorverarbeiten, dass anschließend die Semantik analysiert werden kann. Die vorverarbeitete Eingabe wird an die Einheit zur Analyse der Semantik weitergereicht.

Zur Analyse der Semantik natürlicher Sprache werden innerhalb der *EAS* sogenannte *Agenten* verwendet, die unabhängig voneinander agieren; für jedes (natürlichsprachliche) Phänomen kann ein eigener Agent entwickelt werden (z. B. zur Erkennung sprachlicher Referenzen). Die Agenten können für ihre Analysen intern beliebige Ansätze verfolgen. Ihre Analyseergebnisse hinterlegen sie in einer von der Rahmenarchitektur verwalteten Datenstruktur. Aus dieser Datenstruktur können die Agenten auch Informationen entnehmen und als Grundlage ihrer Analysen nutzen. Somit können die Agenten über die Datenstruktur Informationen (in Form von Analyseergebnissen) austauschen ohne direkt miteinander zu kommunizieren. Dadurch, dass Agenten für einzelne Phänomene entwickelt werden können, können einerseits jeweils hoch spezialisierte Lösungen implementiert werden. Andererseits können die Agenten einzeln dahingehend evaluiert werden, wie gut sie das jeweilige Phänomen erfassen. Die Agenten werden von der *EAS* parallel und wiederholt ausgeführt. Dadurch können sie potenziell von Analyseergebnissen anderer Agenten profitieren; unterschiedliche Sprachanalysen können sich so gegenseitig unterstützen. Beispielsweise hilft es, sprachliche Referenzen wie Anaphern aufzulösen, um den Kontext des Gesagten zu verstehen. Gleichzeitig können manche Referenzen nur mithilfe des Kontextes aufgelöst werden. Der Vorteil der Architektur besteht darin, dass durch parallele Verarbeitung der Eingabe zunächst einige Referenzen aufgelöst werden können. Mit diesen Informationen kann ein partieller Kontext aufgebaut werden, der es wiederum erlaubt, weitere Referenzen aufzulösen und so weiter. Durch die Kombination von Agenten können beliebig komplexe Systeme für tiefes Sprachverständnis entwickelt werden. Zudem können Agenten jederzeit weiterentwickelt, ausgetauscht oder hinzugefügt werden, ohne dass andere Agenten angepasst werden müssen.

Ist die Analyse der Semantik abgeschlossen, erfolgt die Nachverarbeitung. Innerhalb dieses zweiten Fließbandes werden die Analyseergebnisse interpretiert und eine Ausgabe erzeugt. Ein System zur Programmierung mit natürlicher Sprache kann mithilfe des Nachverarbeitungsfließbandes beispielsweise schrittweise Quelltext generieren.

Durch den dreigliedrigen Aufbau von PARSE können darauf aufbauende Systeme flexibel gestaltet werden. Für jedes System kann frei entschieden werden, welche Analysen sequenziell innerhalb der Fließbänder ausgeführt werden sollen und welche Analysen als Agenten umgesetzt werden sollten, da sie von einer parallelen Ausführung innerhalb der *EAS* profitieren. Da sowohl Agenten als auch Fließbandstufen als Einschübe umgesetzt sind, können Systeme beliebig zusammengesetzt werden. Einschübe können ausgetauscht, hinzugefügt oder entfernt werden, oder im Falle von Fließbandstufen in eine andere Reihenfolge gebracht werden. Natürlich ist es auch möglich, die konkrete Ausprägung eines Einschubs zu ändern; beispielsweise kann eine Fließbandstufe mit geringen Anpassungen auch als Agent verwendet werden und umgekehrt.

Alle Bestandteile von PARSE sind unabhängig von einer konkreten Anwendungsdomäne. PARSE kann somit als Architektur für allgemeines Sprachverständnis verwendet werden. Damit auf Grundlage von PARSE konkrete Systeme entwickelt werden können, muss jedoch ein Zugang zu Informationen über die Anwendungsdomäne erfolgen. Hierzu bietet PARSE die Möglichkeit, über Luna externe Ressourcen zu laden und zu verwalten. Die Art und Form der Ressourcen wird nicht eingeschränkt. Somit können beispielsweise beliebige Wissensdatenbanken oder lexikalische Datenbanken geladen werden (siehe Abschnitt 2.3.15).

Zur Repräsentation der Anwendungsdomäne bietet PARSE die Möglichkeit Ontologien zu verwenden. Konkrete Systeme können sämtliches Wissen über die Anwendungsdomäne (wie beispielsweise die durch eine API angebotenen Funktionen) in Ontologien hinterlegen. Die Repräsentation in einer Ontologie stellt eine Art Schnittstelle zum Domänenwissen dar. Dadurch erhalten Agenten und Fließbandstufen Zugriff auf Informationen über die Anwendungsdomäne, ohne dass Domänenwissen in die Entwicklung der Sprachanalysen einfließen muss. Diese Trennung sorgt dafür, dass auch konkrete Systeme, die auf PARSE basieren, einfach an neue Anwendungsdomänen adaptiert werden können. Im Idealfall müssen zur Adaption lediglich die Nachverarbeitungsstufen und die Domänenrepräsentation angepasst werden. Das Konzept der Trennung von sprachlichen Analysen und der Anwendungsdomäne, sowie der Repräsentation von Domänenwissen in einer Ontologie, entstammt der Arbeit von Landhäußer et al. [LWT17b].

4.2.1 Architekturübersicht

Wie zuvor erläutert, besteht die Architektur PARSE aus drei Bestandteilen: einem sequenziellen Vorverarbeitungsfließband, der Einheit zur Analyse der Semantik, die unabhängige Agenten parallel und wiederholt ausführt, und einem weiteren Fließband zur Nachverarbeitung der Analyseergebnisse. In den folgenden Abschnitten werden die Funktionsweisen der einzelnen Bestandteile präzisiert. Zuletzt wird erläutert, wie die Rahmenarchitektur Luna die einzelnen Bestandteile nacheinander zur Ausführung bringt. Innerhalb von PARSE tauschen sowohl Fließbandstufen als auch Agenten Analyseergebnisse über geteilte Datenstrukturen aus. Im nächsten Abschnitt werden daher zunächst die zugrundeliegenden Datenmodelle vorgestellt.

4.2.2 Datenmodelle

Die Module (Fließbandstufen und Agenten) sollen in PARSE möglichst unabhängig voneinander agieren. Daher ist keine direkte Kommunikation zwischen den Modulen vorgesehen; ein Austausch von Informationen soll ausschließlich anhand von geteilten Datenstrukturen erfolgen. Dementsprechend muss für die Fließbänder und die Einheit zur Analyse der Semantik jeweils ein Datenmodell definiert werden.

Die Fließbänder dienen zur Verarbeitung der Eingabe und zur Erzeugung der Ausgabe. Damit sind sie stark vom konkreten Anwendungsfall des jeweiligen Systems abhängig und können (im Gegensatz zu Agenten) größtenteils nicht wiederverwendet werden. Daher ist es auch nicht nötig, Vorgaben für das in den Fließbandstufen zu nutzende Datenmodell zu machen. Die Rahmenarchitektur stellt dementsprechend lediglich sicher, dass ein Datenobjekt von einer Fließbandstufe zur nächsten weitergereicht wird. Der Inhalt der Datenstruktur wird bis auf eine Ausnahme nicht vorgegeben. Diese Ausnahme ist das Datenmodell, welches die Agenten verwenden (siehe Abschnitt 4.2.2.2). Eine Instanz dieses Datenmodells müssen alle Datenstrukturen der Fließbandstufen aufnehmen können. Dies ist erforderlich, da die Rahmenarchitektur erwartet, dass das Ergebnis des Vorverarbeitungsfließbands in dieses Modell überführt wird. Gleichermaßen wird das Ergebnis der semantischen Analyse durch die Agenten dem Nachverarbeitungsfließband in Form dieses Modells übergeben.

Für die Einheit zur Analyse der Semantik ist hingegen die Vorgabe eines Datenmodells sinnvoll, schließlich sollen die Agenten innerhalb der *EAS* nicht nur völlig unabhängig voneinander agieren können, sie sollen möglichst auch unabhängig von der konkreten Anwendungsdomäne allgemeingültig Sprache semantisch analysieren können. Daher wird für die Agenten ein gemeinsames Datenmodell vorgegeben, über das die Agenten miteinander (implizit) kommunizieren können.

Das zu verwendende Datenmodell muss bestimmte Kriterien erfüllen, damit es im Kontext von PARSE eingesetzt werden kann. Diese werden nachfolgend gemeinsam mit potenziellen Modellen (bzw. Modelltypen) diskutiert, bevor das in PARSE verwendete Datenmodell erläutert wird.

4.2.2.1 Potenzielle Datenmodelle für die Einheit zur Analyse der Semantik

Innerhalb der *EAS* soll die Semantik einer natürlichsprachlichen Sequenz analysiert werden. Das bedeutet, ein geeignetes Datenmodell muss natürliche Sprache repräsentieren können und die Agenten befähigen, Analysen auf einer Instanz des Modells durchzuführen und Analyseergebnisse einzupflegen.

Aus der in Abschnitt 1.2 formulierten Zieldefinition sowie den in Abschnitt 4.1 aufgestellten Design-Prinzipien für PARSE werden die folgenden Anforderungen an ein Datenmodell abgeleitet:

(A_1) *Repräsentation natürlicher Sprache*: Das Datenmodell muss es ermöglichen, natürlichsprachliche Sequenzen zu repräsentieren. Das bedeutet, die einzelnen Token einer Äußerung müssen dargestellt werden können (siehe Abschnitt 2.3.4.1). Dabei muss die Ordnung der Sequenz erhalten bleiben bzw. repräsentiert werden können.

(A_2) *Repräsentation linguistischer Relationen*: Das Datenmodell muss es ermöglichen, syntaktische Einheiten (z. B. Wortsequenzen) zu bilden. Linguistische Relationen zwischen den Einheiten müssen dargestellt werden können. Zusätzlich muss es möglich sein, Abstraktionen zu bilden und diese in Relation zu den Einheiten zu setzen (z. B. zur Bildung von Ober-Unterbegriff-Beziehungen (*Hyperonymie*)). Dementsprechend muss das Datenmodell auch die Darstellung hierarchischer Relationen ermöglichen.

(A_3) *Repräsentation verteilter Informationen*: Die in (A_2) beschriebenen Einheiten müssen nicht zusammenhängend sein; syntaktische oder semantische Einheiten können auch von Wörtern gebildet werden, die in der Sequenz nicht aufeinander folgen. Zusätzlich gibt es in der Computerlinguistik Relationen, die zwischen Einheiten gebildet werden, die (potenziell) weit entfernt sind (z. B. Korreferenzen, siehe Abschnitt 2.3.4.7). Das Datenmodell muss dementsprechend eine effiziente Darstellung verteilter (und potenziell innerhalb der Sequenz weit entfernter) Informationen ermöglichen.

(A_4) *Repräsentation externer Informationen*: In die Analyse der Semantik natürlichsprachlicher Sequenzen sollen gegebenenfalls auch Informationen aus externen Datenquellen einfließen (wie z. B. *Wikipedia* oder *WordNet*, siehe Abschnitt 2.3.15). Auch diese Art von Informationen müssen mithilfe des Datenmodells dargestellt werden können.

(A_5) *Dynamisch änderbar*: Die Agenten sollen die Semantik natürlichsprachlicher Sequenzen anhand einer Instanz des Datenmodells analysieren. Die Analyseergebnisse der Agenten erzeugen neue Informationen, die in das Datenmodell eingepflegt werden müssen. Das bedeutet, das Datenmodell muss dynamisch und mit geringem Aufwand änderbar sein.

(A_6) *Kommunikation zwischen den Agenten*: Die in (A_5) beschriebenen Änderungen des Datenmodells zur Darstellung der Analyseergebnisse dienen auch der Kommunikation zwischen den Agenten. Dementsprechend muss es das Datenmodell ermöglichen, jedwede Form von Analyseergebnis zu repräsentieren; dies können linguistische Relationen, wie in (A_2) beschrieben, aber auch von den Agenten frei gewählte Relationen bzw. Darstellungen sein. Zusätzlich müssen Informationen in nachvollziehbaren und einheitlichen Strukturen kodiert werden können, um den Austausch zwischen Agenten einfach zu gestalten.

(A_7) *Unterstützung von Multimodalität (optional)*: Optional sollte das Datenmodell Multimodalität unterstützen. Das bedeutet, auch Informationen aus anderen Eingabequellen, wie beispielsweise erkannte Gesten eines Bildverarbeitungssystems, sollten dargestellt werden können. Zwar fordert bereits (A_4) die Darstellbarkeit von Informationen aus externen Datenquellen. Diese basieren jedoch auf textuellen Einträgen; die hier geforderte Multimodalität erfordert die Darstellung gegebenenfalls nicht verbalisierbarer Konzepte. Da der Fokus von PARSE jedoch auf der Verarbeitung natürlicher Sprache liegt, ist diese Anforderung lediglich optional.

Für den Vergleich unterschiedlicher Datenmodelle wird eine Liste von Kandidaten aufgestellt. Betrachtet werden zum einen Datenmodelle, die bereits erfolgreich zur Darstellung von natürlicher Sprache verwendet wurden. Hierzu zählt die Token-Etiketten-basierte Darstellung, wie sie üblicherweise von Computerlinguistikfließbändern verwendet wird (siehe Abschnitt 2.3.4). Auch das zur Darstellung von Texten samt Meta-Informationen verwendete *TEI*-Format [IV95][7] gehört zu dieser Kategorie. Weitere Datenmodelle entstammen verwandten Arbeiten, wie die *Idea Notation* [KM06] oder Kategorialgrammatiken [Ste87; ZC05] (siehe Abschnitt 3.3). Des Weiteren werden Datenmodelle betrachtet, die sich zur Repräsentation von Wissensstrukturen eignen; zu dieser Kategorie zählen *Topic Maps* und Ontologien (siehe Abschnitt 2.3.14). Zuletzt wird mit einem Graphen als Datenmodell eine abstrakte Struktur betrachtet, die sich ebenfalls eignet, um Einheiten und Relationen zwischen diesen zu repräsentieren. Die Liste erhebt keinen Anspruch auf Vollständigkeit; vielmehr wurden Datenmodelle gewählt, die im Kontext der Computerlinguistik ihre Tauglichkeit bereits unter Beweis gestellt haben. Außerdem sind die betrachteten Modelle teilweise als Stellvertreter einer Gruppe zu verstehen; beispielsweise ist *TEI* ein Vertreter *XML*-basierter Formate und Kategorialgrammatiken dienen als Repräsentant Grammatik-basierter Ansätze. Nachfolgend werden die Modelle anhand der aufgestellten Anforderungen auf ihre Eignung für PARSE untersucht.

Token-Etiketten

Die Verwendung von Token in Verbindung mit Etiketten zur Darstellung linguistischer Eigenschaften ist ein weit verbreiter Ansatz. Durch die Verwendung von Token kann die Sequenz der Wörter einer natürlichsprachlichen Äußerung gemäß ihrer Reihenfolge repräsentiert werden (A_1). Durch

[7] *TEI - Text Encoding Initiative*: `https://tei-c.org/`, zuletzt besucht am 24.02.2021.

die Verwendung des *IOB*(*ES*)-Formats (siehe Abschnitt 2.3.6) können auch zusammenhängende Einheiten in Form von aufeinanderfolgenden Token- bzw. Etikettensequenzen gebildet werden. Die Darstellung von Relationen gestaltet sich hingegen schwierig. Zwar können Relationen zwischen Token dargestellt werden, indem das Quelltoken ein Etikett erhält, welches auf das Zieltoken verweist; diese Darstellung ist jedoch unübersichtlich und fehleranfällig. Abstrakte Konzepte und Hierarchien lassen sich mit Token-Etiketten nicht oder nur mit hoher Redundanz repräsentieren; zur Darstellung einer Baumstruktur muss beispielsweise für jede Baumebene ein Etikett je überspannten Token erzeugt werden (A_2). Auch verteilte Informationen lassen sich aus ebendiesen Gründen nur unzureichend repräsentieren (A_3). Externe Informationen können in Etiketten hinterlegt werden. Soll hingegen ein externes Konzept mit einer Menge von Token in Verbindung gebracht werden, ergeben sich die bereits diskutierten Schwierigkeiten (A_4). Die dynamische Änderbarkeit der Darstellung ist hingegen einfach; Etiketten können geändert, hinzugefügt, gelöscht werden (A_5). Durch die bereits diskutierten Einschränkungen bei der Darstellung von Relationen sind Token-Etiketten trotzdem nur bedingt geeignet, um Analyseergebnisse zu hinterlegen und damit die Kommunikation zwischen den Agenten zu ermöglichen (A_6). Auch Multimodalität ist nur schwer umzusetzen, da Token-Etiketten den Fokus auf Wörter als informationstragende Einheit legen (A_7).

TEI

Das *TEI*-Format basiert auf *XML* und bietet die Möglichkeit, Text mit hierarchisch organisierten (Meta-)Informationen anzureichern. Abbildung 4.2 zeigt zwei Beispielsätze, dargestellt im *TEI*-Format. Zwar wird das *TEI*-Format normalerweise zur Auszeichnung (engl. *mark up*) von Sätzen oder Paragraphen verwendet, eine Unterteilung pro Wort ist aber prinzipiell möglich; das Format ist zudem reihenfolgeerhaltend (A_1). Linguistische Relationen – insbesondere hierarchische – lassen sich im *TEI*-Format repräsentieren (A_2). Die Darstellung verteilter Informationen ist hingegen (ähnlich wie bei Token-Etiketten) nur über Verweise auf eine andere Struktur möglich; bei einer großen Anzahl an Verweisen ergeben sich die im vorangegangen Paragraphen diskutierten Schwierigkeiten (A_3). Externe Informationen können einfach hinzugefügt werden; die Auszeichnungen des *TEI*-Formats sind eben dafür gedacht (A_4). Da *TEI* auf *XML* beruht, sind die aufgebauten Strukturen jederzeit änderbar (A_5). Zur Kommunikation zwischen den Agenten ist das *TEI*-Format trotzdem nur bedingt geeignet; durch den starken Fokus auf hierarchische Strukturen, können nicht alle Arten von Relationen und damit auch nicht beliebige Analyseergebnisse dargestellt werden (A_6). Auch das Einpflegen multimodaler Eingaben ist nur umständlich möglich (z. B. durch die Erstellung einer entsprechenden Auszeichnung), da der Fokus des *TEI*-Formats stark auf der Modellierung verbalisierbarer Informationen liegt (A_7).

Idea Notation

Der Zweck der *Idea Notation* ist die Modellierung von Konzepten, die in einer natürlichsprachlichen Äußerung auftreten. Somit bildet die *Idea Notation* eine Art Ontologie für den Gegenstandsbereich einer Äußerung. Abbildung 4.3 zeigt eine beispielhaftes konkretes Modell aus Originalveröffentlichung von Knöll und Mezini [KM06]. Bei der Erstellung der Konzepte und Relationen zwischen den Konzepten spielt die Reihenfolge der Wörter in der Sequenz keine Rolle und kann auch nicht dargestellt werden. Dadurch eignet sich die *Idea Notation* nicht zur Darstellung einer Tokensequenz (A_1). Die Darstellung von (linguistischen) Relationen ist hingegen der hauptsächliche Anwendungszweck

```
<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
    <teiHeader>
        <fileDesc>
            <titleStmt>
                <title>
                    Quote by Noam Chomsky
                </title>
            </titleStmt>
        </fileDesc>
    </teiHeader>
    <text>
        <body>
            <p>
                <s>
                    <w>Language</w><w>is</w><w>a</w>
                    <w>process</w><w>of</w>
                    <w>free</w><w>creation</w><w>;</w>
                </s>
                <s>
                    <cl>
                        its laws and principles are fixed,
                    </cl>
                    <cl>
                        but the manner in which the principles of
                        generation are used is free and infinitely varied.
                    </cl>
                </s>
            </p>
        </body>
    </text>
</TEI>
```

Abbildung 4.2: Zwei Sätze (Etikette <s>) eines Zitats von Noam Chomsky im *TEI*-Format: Der erste Satz wurde wortweise (Etikette <w>) und der zweite phrasenweise (Etikette <cl>) unterteilt.

(A_2). Das gilt auch für verteilte Informationen und weite Relationen; Konzepte können unabhängig von der ursprünglichen Tokensequenz in Verbindung gebracht werden (A_3). Da die *Idea Notation* primär zur Repräsentation von Konzepten, die in einer natürlichsprachlichen Äußerung auftreten, konzipiert wurde, können Informationen aus externen Quellen nicht ohne Weiteres dargestellt werden. Prinzipiell könnten für diese aber Konzepte und Relationen angelegt werden (A_4). Die *Idea Notation* ist änderbar; Konzepte und Relationen können beliebig hinzugefügt, geändert oder entfernt werden (A_5). Durch die einfache Änderbarkeit und die umfangreichen Darstellungsmöglichkeiten für Relationen eignet sie sich auch zur Kommunikation von Analyseergebnissen (A_6). Auch Multimodalität ist prinzipiell möglich; hierzu könnten – wie zur Darstellung von Informationen aus externen Quellen – entsprechende Konzepte und Relationen angelegt werden (A_7).

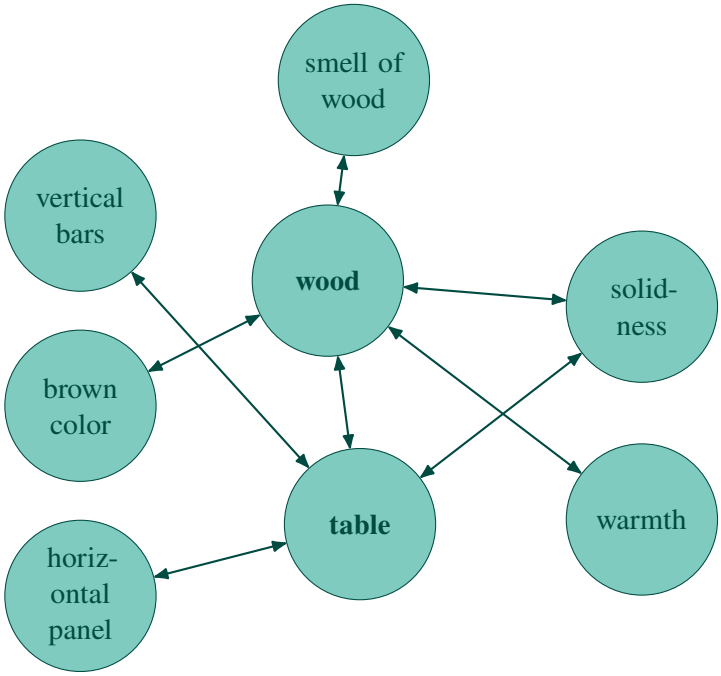


Abbildung 4.3: Eine konkrete Instanz der *Idea Notation*, wie von Knöll und Mezini vorgeschlagen [KM06].

$$\dfrac{\dfrac{\text{What}}{(S/(S\backslash NP))/N \quad \lambda f.\lambda g.\lambda x.f(x) \wedge g(x)} \quad \dfrac{\text{states}}{N \quad \lambda x.state(x)}}{S/(S\backslash NP) \quad \lambda g.\lambda x.state(x) \wedge g(x)} > \quad \dfrac{\dfrac{\text{border}}{(S\backslash NP)/NP \quad \lambda x.\lambda y.borders(y,x)} \quad \dfrac{\text{Texas}}{NP \quad texas}}{(S\backslash NP) \quad \lambda y.borders(y,texas)} >$$

$$\dfrac{}{S \quad \lambda x.state(x) \wedge borders(x,texas)} >$$

Abbildung 4.4: Vollständige semantische Ableitung eines Beispielsatzes mithilfe von Kategorialgrammatiken [ZC05].

Kategorialgrammatiken

Kategorialgrammatiken ermöglichen es, natürlichsprachliche Sequenzen (üblicherweise Sätze) semantisch zu zerteilen. Sie ähneln den Grammatiken, die zur syntaktischen Zerteilung verwendet werden (siehe Abschnitt 2.3.4.6); allerdings unterteilen sie die Eingabe in semantische anstelle syntaktischer Kategorien. Abbildung 4.4 zeigt einen Beispielsatz der mithilfe einer Kategorialgrammatik semantische zerteilt wurde. Die Grundlage für die semantischen Zerteilungen bilden Tokensequenzen (A_1). Die semantischen Kategorien, die von Kategorialgrammatikzerteilern vergeben werden, umfassen unter anderem auch linguistische Relationen, insbesondere hierarchische (A_2). Die Repräsentation externer (durch die Grammatik nicht vorgesehener) Informationen ist hingegen nicht möglich (A_3). Verteilte Informationen werden durch die semantischen Zerteilungen umfassend erfasst; allerdings sind hierzu teils komplexe Zerteilungen notwendig, die wiederum

schwierig zu interpretieren sind (A_4). Eine Änderung einer einmal erzeugten Zerteilung ist – ähnlich wie bei syntaktischen Zerteilern – nicht vorgesehen (A_5). Dadurch sind Kategorialgrammatiken auch ungeeignet, um Analyseergebnisse der Agenten zu kommunizieren (A_6). Da es nicht möglich ist, externe Informationen in die Zerteilungen einfließen zu lassen, ist die Umsetzung von Multimodalität mit Kategorialgrammatiken ebenfalls nicht möglich (A_7).

Topic Maps

Der Fokus von *Topic Maps* liegt vorrangig auf der Formalisierung von Konzeptwissen. Daher ist die Darstellung einer Tokensequenz nicht ohne Weiteres möglich. Es wäre zwar denkbar, je Token ein sogenanntes *Vorkommen* (engl. *occurence*) zu einem *Thema* (engl. *topic*) zu definieren, die Reihenfolge der Token ließe sich dann aber nur aufwendig darstellen (A_1). Linguistische Relationen lassen sich in *Topic Maps* hingegen darstellen, indem Relationen zwischen Themen aufgestellt werden. Allerdings können Relationen ausschließlich zwischen Themen und nicht zwischen Vorkommen (oder zwischen Vorkommen und Themen) gebildet werden, was die Eignung zur Repräsentation linguistischer Relationen einschränkt (A_2). Die Repräsentation verteilter Informationen gelingt hingegen einfach, da Themen direkt über Relationen miteinander verbunden werden können (A_3). Externe Informationen können ebenso eingepflegt werden, sofern ein entsprechendes Thema gebildet werden kann (A_4). *Topic Maps* lassen sich anpassen: Konzepte, Auftreten und Relationen können jederzeit hinzugefügt, geändert oder gelöscht werden (A_5). Da die Bildung von Relationen nur beschränkt möglich ist, eignen sich *Topic Maps* auch nur bedingt zum Austausch von Informationen zwischen Agenten; gegebenenfalls können Analyseergebnisse nur verlustbehaftet weitergegeben werden (A_6). Eine multimodale Verwendung von *Topic Maps* ist hingegen gut möglich, sofern sich die Eingabe als Thema darstellen lässt (A7).

Ontologien

Wie bei *Topic Maps* liegt der Fokus von Ontologien auf der Darstellung von Konzeptwissen. Die Darstellung natürlicher Sprache als Tokensequenz ist nicht vorgesehen. Trotzdem können Ontologien verwendet werden, um einzelne Wörter zu repräsentieren, beispielsweise als Individuen oder Klassen. Ähnlich wie bei *Topic Maps* ist jedoch die Erhaltung der Reihenfolge problematisch (A_1). Relationen aller Art sind hingegen mit Ontologien darstellbar; dabei spielt es auch keine Rolle, ob es sich um hierarchische oder verteilte Informationen oder solche aus externen Quellen handelt (A_2 - A_4)[8]. Die in einem Datenformat wie *OWL* (siehe Abschnitt 2.3.14) kodierten Konzepte, Individuen, Relationen etc. können zur Laufzeit geändert werden (A_5). Dementsprechend eignen sich Ontologien auch vollumfänglich zur Darstellung beliebiger Analyseergebnisse der Agenten (A_6). Multimodale Eingaben ließen sich ebenfalls (beispielsweise durch die Einführung zusätzlicher Konzepte und Relationen) repräsentieren (A_7).

Graphen

Graphen können als Datenmodell universell zur Repräsentation beliebiger Informationen eingesetzt werden. Nichtsdestotrotz erscheint eine Verwendung zur Darstellung von natürlicher Sprache

[8] Letztlich könnte auch die Reihenfolge der Token mithilfe von Relationen dargestellt werden. Gegenüber anderen Darstellungsformen ist diese aber vergleichsweise aufwendig.

Tabelle 4.1: Vergleich unterschiedlicher Datenmodelle, die zur Verwendung innerhalb der Einheit zur Analyse der Semantik innerhalb von PARSE infrage kommen: Die Datenmodelle werden anhand der zu Beginn von Abschnitt 4.2.2.1 gestellten Anforderungen verglichen. Ein ausgefüllter Kreis symbolisiert (•) eine vollumpfängliche Erfüllung der Anforderungen und leere Kreise (○) die teilweise Erfüllung. Ist kein Symbol angegeben, kann das Datenmodell die entsprechende Anforderung nicht erfüllen.

Repräsentationsart	Repräsentation natürlicher Sprache (A_1)	Repräsentation linguistischer Relationen (A_2)	Repräsentation verteilter Informationen (A_3)	Repräsentation externer Informationen (A_4)	Dynamisch änderbar (A_5)	Kommunikation zwischen Agenten (A_6)	Unterstützung von Multimodalität (optional) (A_7)
Token-Etiketten	•	○	○	○	•	○	
TEI	•	•	○	•	•	○	○
Idea Notation		○	•	○	•	•	○
Kategorialgrammatiken	•	•	•		○		
Topic Maps	○	○	•	•	•	○	•
Ontologien	○	•	•	•	•	•	•
Graphen	•	•	•	•	•	•	•

zunächst ähnlich unintuitiv wie bei *Topic Maps* und Ontologien. Tatsächlich werden Graphen in der Computerlinguistik häufig zur Darstellung komplexer Zusammenhänge eingesetzt[9]. Graphen bieten unter anderem die Möglichkeit, Token in Knoten zu repräsentieren; mittels gerichteter Kanten können diese zudem in eine Reihenfolge gebracht werden (A_1). Beliebige linguistische Relationen zwischen Token können durch Hinzunahme weiterer (gerichteter) Kanten dargestellt werden. Sollen hierarchische Informationen repräsentiert werden, können zusätzliche Knoten eingeführt werden, die abstrakte Konzepte repräsentieren (A_2). Auch die Darstellung verteilter oder weit entfernter Relationen stellt kein Problem dar; hierzu werden die entsprechenden Knoten, die syntaktische oder semantische Einheiten repräsentieren, durch Kanten verbunden (A_3). Die Darstellung von Informationen aus externen Quellen gelingt ebenfalls mithilfe geeigneter Knoten-Kanten-Kombinationen (A_4). Damit ist auch Multimodalität einfach umsetzbar (A_7). Graphen können verändert werden (sofern dies durch die verwendete Rahmenarchitektur unterstützt wird); Knoten und Kanten können hinzugefügt und entfernt werden (A_5). Durch die flexiblen Darstellungsmöglichkeiten sind Graphen auch zur Darstellung beliebiger Analyseergebnisse der Agenten und damit zur Kommunikation zwischen diesen geeignet (A_6).

[9] Beispielsweise erzeugen sowohl Konstituenz- als auch Dependenzzerteiler Graph-basierte Darstellungen der Satzsyntax (siehe Abschnitt 2.3.4.6). Ein weiteres Beispiel für die Verwendung von Graphen ist die Korreferenzanalyse (siehe Abschnitt 2.3.4.7).

Tabelle 4.1 fasst die gesammelten Erkenntnisse über die potenziellen Darstellungsformen zusammen. Wie der Tabelle entnommen werden kann, erfüllt lediglich eine Graph-basierte Darstellung die zu Beginn des Abschnitts definierten Anforderungen an ein Datenmodell vollumfänglich. Zwar stellen auch Ontologien und *Topic Maps* gute Alternativen dar; letztlich sind beide aber auch nur spezialisierte Graph-Modelle, die syntaktische bzw. semantische Einschränkungen bei der Darstellung des zu modellierenden Wissens machen. Der Grund hierfür ist, dass beide den Fokus auf die Repräsentation von Konzeptwissen legen; zur Darstellung einer natürlichsprachlichen (Token-)Sequenz, müssten sie auf eine nicht vorgesehene Weise verwendet werden. Daher wird in PARSE ein Graph als zentrales Datenmodell während der semantischen Analyse durch die Agenten verwendet. Nachfolgend werden die Eigenschaften des verwendeten Graphmodells erläutert.

4.2.2.2 PARSE-Graphen

Das für PARSE entworfene Graphmodell entspricht einem typisierten, gerichteten Multigraphen. Das bedeutet, der Graph G besteht aus Knoten (engl. *vertices*) V und Kanten (engl. *edges*) E. Die Kanten sind gerichtet, haben also eine Orientierung: Kanten verbinden immer eine Quelle mit einer Senke (beides Knoten). Dabei ist es im PARSE-Graphen möglich, dass die Quelle und die Senke identisch sind (reflexive Kante). Ebenso kann ein Knotenpaar über unterschiedliche Kanten miteinander verbunden werden (Multigraph). PARSE-Graphen können zudem Zyklen enthalten.

Sowohl Knoten als auch Kanten sind typisiert. Typen können wiederum benannte, typisierte Attribute zugewiesen werden. Die Attributtypen sind nicht eingeschränkt, es können sowohl primitive als auch nicht-primitive Datentypen verwendet werden. In konkreten Knoten können den Attributen dann Werte hinzugefügt werden.

Beispiel:
Erzeugung von Knoten und Kanten für einen PARSE-Graphen

In diesem Beispiel soll die Erzeugung von zwei Knoten und einer Kante zwischen diesen demonstriert werden. Das Beispiel nimmt Bezug auf den in Abbildung 4.5 dargestellten Graphen.

Erläuterung

Um den in der Abbildung gezeigten Graphen zu erhalten, wird zunächst ein Knotentyp `VType1` erzeugt (falls dieser noch nicht vorhanden ist). Der Knotentyp erhält zwei Attribute, `pos` und `val`; das erste ist vom Typ `int` das zweite vom Typ `String`. Anschließend wird der Knoten `V1` erzeugt und als Typ `VType1` festgelegt. Den Attributen im Knoten können nun Werte zugewiesen werden; alternativ kann die Wertzuweisung zu jedem späteren Zeitpunkt geschehen. Der Knoten `V2` wird analog dazu angelegt und ist ebenfalls vom Typ `VType1`. Auch bei der Erstellung der Kante muss zunächst ein Kantentyp (im Beispiel `EType1`) erzeugt werden (sofern dieser noch nicht vorhanden ist). Bei der Erzeugung der Kante werden zudem die Quelle und die Senke angegeben, im Beispiel `V1` und `V2`.

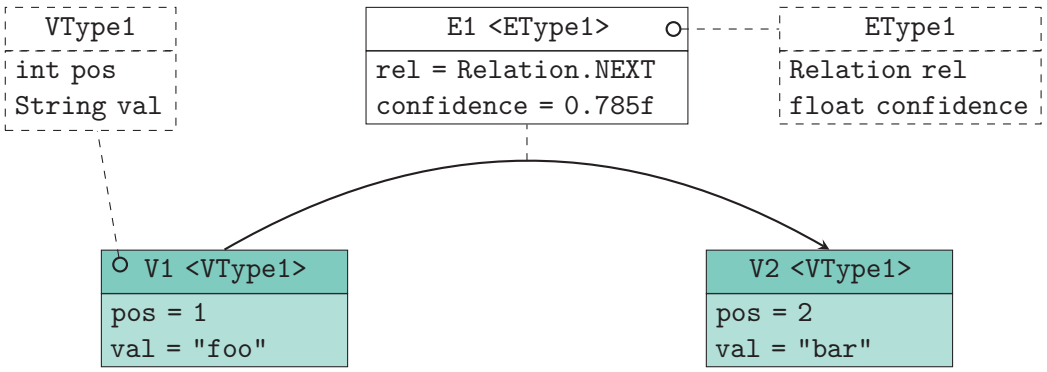


Abbildung 4.5: Ein beispielhafter PARSE-Graph, bestehend aus zwei Knoten und einer Kante zwischen diesen.

Knoten und Kanten können gelöscht werden. Ebenso können Kanten umgelegt werden; das bedeutet, sowohl die Quelle als auch die Senke können nachträglich angepasst werden. Knoten- und Kantentypen können hingegen nicht gelöscht werden. Auch die typisierten Attribute eines Knoten- oder Kantentypen können nicht entfernt werden; das nachträgliche Hinzufügen weiterer Attribute ist hingegen möglich. Die Werte der Attribute können jederzeit geändert werden. Die Rahmenarchitektur ermöglicht es, Knoten, die über keine Kante mit einem anderen Knoten verbunden sind, automatisch aus dem Graphen zu entfernen. Da dieses Verhalten aber zu unerwünschten Ergebnissen führen kann, ist es möglich, diese Funktion nicht zu verwenden[10]. Weiterhin erlaubt die Rahmenarchitektur die Erzeugung mehrerer PARSE-Graphen; diese Möglichkeit kann beispielsweise genutzt werden, um parallel Alternativhypothesen zu verarbeiten, die durch ein automatisches Spracherkennungssystem erzeugt wurden (siehe Abschnitt 2.3.7).

PARSE bietet darüber hinaus die Möglichkeit tiefe Kopien (engl. *deep copy*) eines Graphen zu erzeugen; das bedeutet, es werden nicht nur Knoten und Kanten kloniert, sondern auch die zugehörigen Knoten- und Kantentypen sowie die Attribute (sofern die Typen der Attribute Klonierung unterstützen). Es kann auch geprüft werden, ob ein PARSE-Graph inhaltlich äquivalent zu einem anderen ist. Geklonte Graphen sind beispielsweise äquivalent (obwohl durch die Klonierung keine Objektgleichheit herrscht). Wird jedoch beispielsweise ein Attributwert geändert, sind die Graphen nicht mehr äquivalent, obwohl weiterhin alle Knoten und Kanten übereinstimmen. Die Möglichkeit tiefe Kopien zu erstellen und Äquivalenzprüfung sind eine wesentliche Voraussetzung für die nebenläufige Verwendung der PARSE-Graphen durch die Agenten[11] (siehe Abschnitt 4.2.4).

4.2.3 Vor- und Nachverarbeitungsfließbänder

Die Architektur PARSE ist so entworfen, dass vor und nach der semantischen Analyse durch die Agenten Fließbänder zur Verarbeitung natürlichsprachlicher Sequenzen eingesetzt werden (siehe Abschnitt 4.2.1). Das Vorverarbeitungsfließband dient dazu, die Eingabe (z. B. ein Audiosignal)

[10] In der Standardkonfiguration von PARSE wird die Funktion *nicht* verwendet.

[11] Die Agenten erhalten zur Durchführung ihrer Analysen eine tiefe Kopie des aktuellen Graphen. Analyseergebnisse eines Agenten werden später nur übernommen, wenn der Agent selbst Änderungen am Graphen durchgeführt hat und in der Zwischenzeit kein anderer Agent den (Haupt-)Graph (durch Zurückschreiben der eigenen Kopie) verändert hat.

entgegenzunehmen und soweit vorzubereiten, dass eine Analyse der Semantik dieser durch die Agenten möglich wird. Das Nachverarbeitungsfließband nimmt das Analyseergebnis der Agenten entgegen und erzeugt daraus eine Ausgabe (z. B. Quelltext). Die in PARSE verwendeten Fließbänder entsprechen einer klassischen *nicht-parallelen* Fließband-Architektur (engl. *pipeline architecture*). Das heißt es wird immer das vollständige Eingabedatum (und etwaige hinzugefügte Informationen) von einer Fließbandstufe zur nächsten weitergereicht und die Stufen werden nacheinander, *nicht* nebenläufig ausgeführt. Die Fließbandstufen sind als Einschübe umgesetzt. Dadurch können für ein konkretes System, das auf PARSE basiert, beliebige Fließbandstufen implementiert und durch die Rahmenarchitektur geladen werden (siehe Abschnitt 4.2.5). Die Einschübe müssen gemäß der durch PARSE vorgegebenen Schnittstelle drei Funktionen umsetzen.

Die erste Funktion dient der Initialisierung der Fließbandstufe (`init`-Methode). Diese wird von der Rahmenarchitektur einmalig nach der Erzeugung des Fließbandstufenobjekts aufgerufen und kann beispielsweise dazu genutzt werden, Ressourcen zu laden. Die zweite Funktion enthält die eigentliche Funktionalität der Fließbandstufe (`exec`-Methode). Diese wird von der Rahmenarchitektur zur Ausführung der Stufe im Fließband aufgerufen. Beim Aufruf wird der Stufe das geteilte Datenobjekt übergeben. Jede Fließbandstufe legt ihre Analyseergebnisse in ebendiesem ab, bevor es an die nächste Stufe weitergereicht wird. Die dritte Funktion sorgt dafür, dass jede Fließbandstufe einen eindeutigen Namen erhält (`getID`-Methode). Für jede Fließbandstufe muss eine ID gewählt werden, welche über die Funktion zurückgegeben wird. Die Benennung der Fließbandstufen ist nötig, da die Rahmenarchitektur unter allen verfügbaren Stufen anhand des Namens die auszuführenden auswählt (siehe Abschnitt 4.2.5). Darüber hinaus macht PARSE keine Vorgaben über die Interna der Fließbandstufen. Allerdings gibt es durch die Verwendung des PARSE-Graphen als Datenmodell für die Einheit zur Analyse der Semantik Vorgaben für die letzte Stufe des Vorverarbeitungsfließbandes und die erste Stufe des Nachverarbeitungsfließbandes. Die letzte Stufe des Vorverarbeitungsfließbandes muss die Analyseergebnisse der vorangegangen Stufen in einem PARSE-Graphen zusammenfassen und diesen im geteilten Datenobjekt hinterlegen[12]. Die erste Stufe des Nachverarbeitungsfließbandes muss entsprechend in der Lage sein, den durch die Agenten veränderten Graphen zu verarbeiten.

4.2.4 Einheit zur Analyse der Semantik

Die Einheit zur Analyse der Semantik *EAS* bildet die Kernkomponente von PARSE. Innerhalb der *EAS* werden die Agenten für tiefes Sprachverständnis ausgeführt. Sie sind dafür zuständig die Semantik natürlichsprachlicher Sequenzen zu analysieren. Durch das Zusammenspiel verschiedener Agenten soll ein möglichst tiefes Verständnis natürlicher Sprache entstehen. Die technische Umsetzung der Agenten in PARSE wird nachfolgend erläutert. Die Agenten agieren ebenfalls als Einschübe mit den denselben Schnittstellenfunktionen wie die Fließbandstufen (siehe Abschnitt 4.2.3 und Abschnitt 4.2.5). Allerdings werden die Agenten von der Rahmenarchitektur nebenläufig und wiederholt zur Ausführung gebracht. Das bedeutet, Luna ruft die `exec`-Methoden aller Agenten

[12] In ProVal wird hierfür eine dedizierte Stufe zur Erzeugung eines Graphen verwendet (siehe Abschnitt 6.5). Alternativ könnte die letzte Stufe aber auch noch eigene Analysen durchführen und dann den Graphen erzeugen.

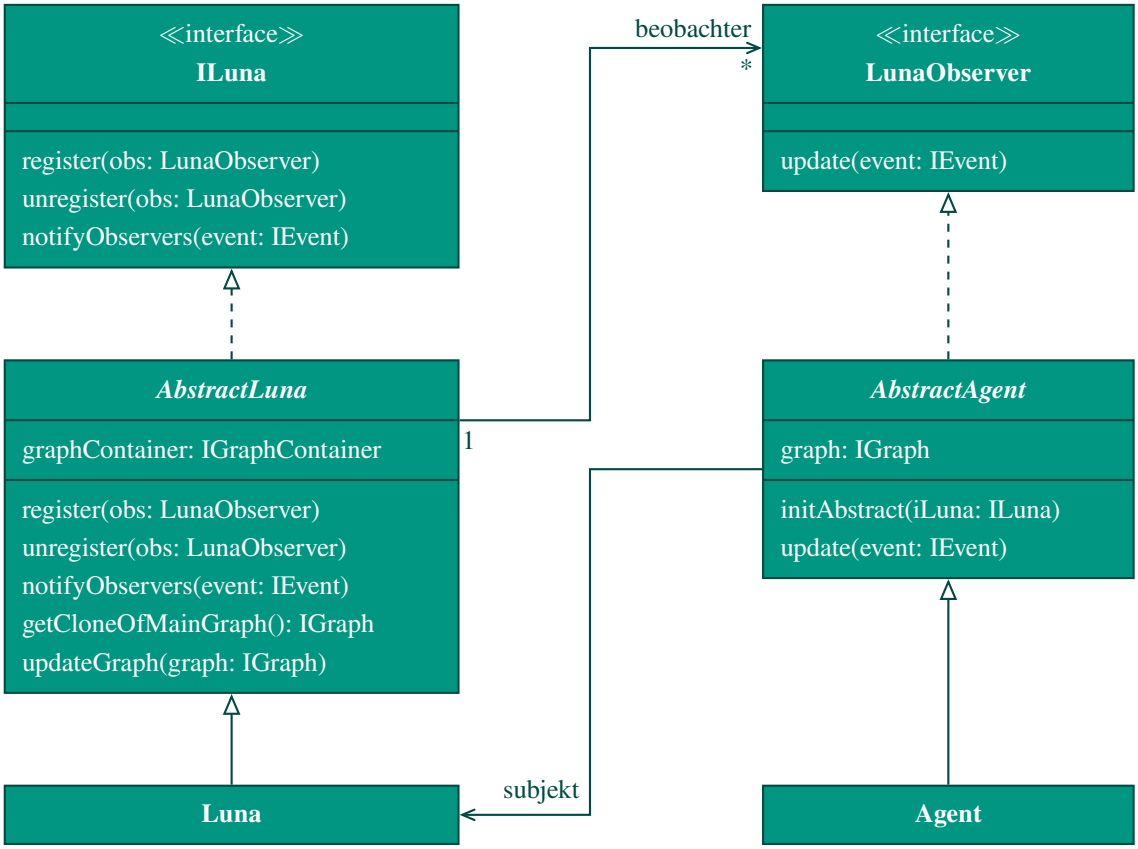


Abbildung 4.6: UML-Klassendiagramm zur Umsetzung des Entwurfsmusters Beobachter in PARSE: *LunaObserver* entsprechen dabei Beobachtern; *AbstractAgents* sind die konkreten Beobachter. *AbstractLuna* übernimmt die Rolle des Subjekts und *Luna* die des konkreten Subjekts.

gleichzeitig auf. Beim Aufruf erhalten die Agenten eine Kopie des aktuellen Graphen. Diese bildet die Grundlage der in einem konkreten Agenten umgesetzten Analysen. Ein Agent kann Knoten, Kanten und Attribute aus der Graphkopie auslesen und gemäß dem in Abschnitt 4.2.2.2 definierten Regelwerks verändern. Hat ein Agent seine Analyse abgeschlossen, legt er die Analyseergebnisse in der Graphkopie ab; hierzu können beispielsweise neue Attribute, Knoten oder Kanten angelegt oder bestehende bearbeitet werden. Zuletzt bietet der Agent seine Graphkopie der Rahmenarchitektur als neuen (Haupt-)Graphen an. Die Rahmenarchitektur prüft, ob die Kopie tatsächlich Änderungen enthält und ob nicht in der Zwischenzeit ein anderer Agent eine aktuellere Version des Graphen hinterlegt hat (siehe Abschnitt 4.2.5). Durch die Verwendung von privaten Kopien des Graphen für die Agenten und das gesteuerte Zurückschreiben durch die Rahmenarchitektur, können keine Wettlaufsituationen auftreten und keine Inkonsistenzen entstehen. Wurde der Graph durch einen Agenten verändert, wird dies allen Agenten bekannt gemacht und die `exec`-Methode aller Agenten wird erneut aufgerufen (siehe Abschnitt 4.2.5); die Agenten erhalten Kopien des neuen Graphen. Um die Agenten über Änderungen zu informieren, wird das Entwurfsmuster *Beobachter* (engl. *observer*) verwendet [Gam+94], wobei die Agenten Beobachter sind, die sich beim Subjekt, im Falle von PARSE die Rahmenarchitektur Luna, registrieren. Der interne, veränderliche Zustand des Subjekts ist der geteilte Graph, der von der Rahmenarchitektur verwaltet wird. Abbildung 4.6 veranschaulicht die Umsetzung des Beobachter-Musters in PARSE.

Durch die parallele und wiederholte Ausführung ergeben sich für die Umsetzung konkreter Agenten zwei Herausforderungen:

1. Zum einen kann bei der Umsetzung einer Sprachanalyse in einem Agenten nicht davon ausgegangen werden, dass Ergebnisse anderer Agenten, die für die eigenen Analysen benötigt werden, zur Verfügung stehen; aufgrund der nebenläufigen Ausführung kann es vorkommen, dass der entsprechende (zuliefernde) Agent seine Analyseergebnisse noch nicht im Graphen hinterlegt hat. Agenten müssen also das Vorhandensein der benötigten Informationen eigenständig prüfen, z. B. durch Prüfung des Vorhandenseins eines speziellen Knotentyps. Falls notwendige Informationen im Graphen fehlen, sollte ein Agent dementsprechend seine Analyse beenden und auf den nächsten Aufrufzyklus warten (siehe Abschnitt 4.2.5). Verwendet ein Agent die Ergebnisse mehrerer anderer Agenten, sollte – sofern möglich – eine gestaffelte Prüfung durchgeführt werden. Falls nur ein Teil der benötigten Informationen vorliegt, kann ein Agent gegebenenfalls seine Analysen bereits teilweise durchführen und Teilergebnisse im Graph hinterlegen. Im nächsten Aufrufzyklus kann die Analyse dann vertieft werden, sofern neue Ergebnisse anderer Agenten zur Verfügung stehen.

2. Zum anderen muss bei der Umsetzung eines konkreten Agenten beachtet werden, dass die `exec`-Methode potenziell beliebig oft von der Rahmenarchitektur aufgerufen wird. Daher muss ein Agent prüfen, ob der Graph bereits eigene Analyseergebnisse enthält (z. B. indem geprüft wird, ob der Graph einen Kantentyp enthält, der vom Agenten erzeugt wird). Andernfalls könnte der Agent fälschlicherweise gedoppelte (bzw. redundante) Analyseergebnisse erzeugen[13]. Je nach Agenten kann ein unterschiedlicher Umgang mit diesem Umstand sinnvoll sein. Sind seitens des Agenten keine inkrementellen Analysen vorgesehen, genügt es, die Ausführung zu beenden, falls der Agent feststellt, dass bereits eigene Analyseergebnisse im Graphen vorhanden sind. Sieht der Agent hingegen inkrementelle Analysen vor, kann der Graph auf neue Informationen hin untersucht und die eigenen Analysen vertieft werden. In diesem Fall muss allerdings beim Einpflegen der neuen Analyseergebnisse darauf geachtet werden, dass keine redundanten Informationen erzeugt werden. Auch hierfür sind unterschiedliche Strategien denkbar. Ein Agent könnte beispielsweise alle eigenen Informationen aus einem vorherigen Durchlauf aus dem Graph löschen bevor die neuen Ergebnisse eingefügt werden oder bei jeder Einfügung einer Kante, eines Knoten usw. prüfen, ob sich das entsprechende Element bereits im Graphen befindet.

PARSE macht hinsichtlich beider Prüfungen (Vorhandensein von Analyseergebnissen anderer Agenten und eigener Analyseergebnisse) keine Vorgaben. Der Grund hierfür ist, dass für diese Fälle kein generelles Vorgehen erzwungen werden soll. Je nach Agent, bzw. je nachdem welche Art von Analyse im Agenten umgesetzt wird, muss ein anderer Umgang mit den zuvor beschriebenen Herausforderungen umgesetzt werden.

[13] Beispielsweise könnte ein Agent zur Persistierung seiner Analyseergebnisse eine neue Kante zwischen zwei bereits bestehenden Knoten erzeugen. Wird der Agent erneut aufgerufen und führt seine Analysen ohne Beachtung des Ist-Zustandes durch, würde er dieselbe Kante ein zweites Mal dem Graph hinzufügen, ohne dass diese Information trägt. Dies würde sich bei jedem Aufruf des Agenten wiederholen.

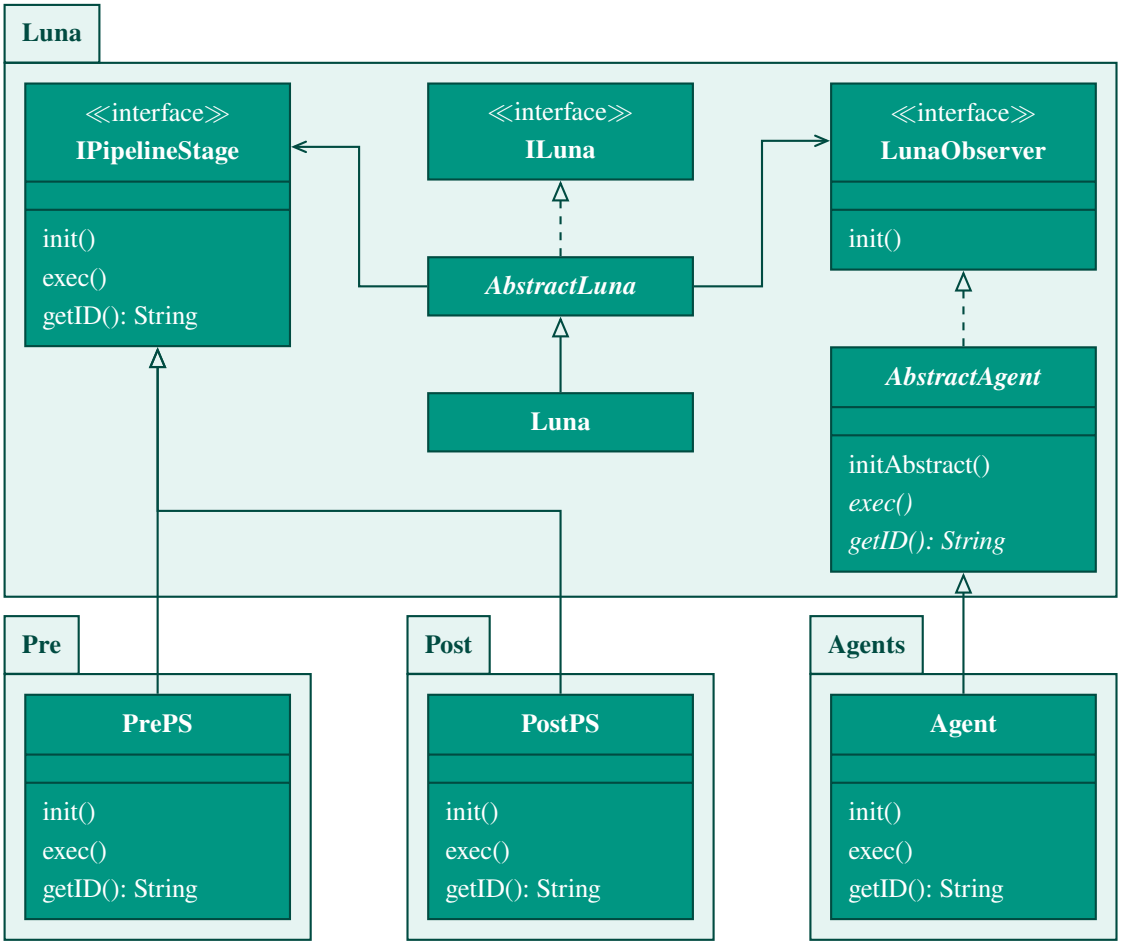


Abbildung 4.7: *UML*-Paketdiagramm, welches das Zusammenspiel zwischen der Rahmenarchitektur und den verschiedenen Einschüben veranschaulicht.

4.2.5 Rahmenarchitektur

Die Rahmenarchitektur Luna ist dafür zuständig, die einzelnen Bestandteile zur Ausführung zu bringen: das Vorverarbeitungsfließband, die *EAS* und das Nachverarbeitungsfließband. Zusätzlich werden die externen Ressourcen bereitgestellt. Der Aufbau der Rahmenarchitektur und die Beziehung zu den Einschüben kann (ausschnitthaft) anhand des *UML*-Paketdiagramms in Abbildung 4.7 nachvollzogen werden.

Zunächst werden alle zur Verfügung gestellten Fließbandstufen und Agenten geladen[14]. Mithilfe von Konfigurationsdateien kann definiert werden, welche dieser zur Verfügung stehenden Stufen und Agenten genutzt werden sollen; die Auswahl wird anhand der `getID`-Methode getroffen. Für die Fließbänder kann zudem angegeben werden, in welcher Reihenfolge die Stufen ausgeführt werden.

[14] Da PARSE Einschub-basiert arbeitet, werden die Einschübe aus *Jar*-Containern geladen, die in einem definierten Verzeichnis abgelegt sind. Geladen werden die konkreten Fließbandstufen und Agenten anschließend mithilfe der *Java-ServiceLoader*-Architektur anhand der durch die Rahmenarchitektur vorgegebenen Schnittstellen. *Java-ServiceLoader*: `https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/ServiceLoader.html`, zuletzt besucht am 24.02.2021.

Algorithmus 1 Schablonenmethode der abstrakten Agenten (run-Methode, vereinfacht)

```
while true do
  if event = UpdateEvent then
    graph ← luna.getGraphCopy()
    exec()
    luna.updateGraph(graph)
  else if event = TerminationEvent then
    break
  else
    luna.wait()
  end if
end while
```

Anschließend wird das Vorverarbeitungsfließband erstellt und die zugehörigen Stufen werden mittels ihrer `init`-Methoden initialisiert. Zusätzlich wird das Modul geladen, das die externen Ressourcen bereitstellt, und das Datenobjekt für das Vorverarbeitungsfließband erzeugt. Dann wird das Vorverarbeitungsfließband ausgeführt, indem alle Stufen in der definierten Reihenfolge aufgerufen werden. Das Datenobjekt wird entsprechend von Stufe zu Stufe weitergereicht.

Nachdem das Vorverarbeitungsfließband vollständig durchlaufen wurde, liest Luna den erzeugten PARSE-Graphen aus dem Datenobjekt aus. Anschließend werden die Agenten innerhalb der *EAS* initialisiert (siehe Abschnitt 4.2.4). Allerdings wird hierzu nicht direkt die `init`-Methode der Agenten aufgerufen. Stattdessen wird eine Initialisierungsfunktion der abstrakten Oberklasse aufgerufen (`initAbstract`). Der Grund hierfür ist folgender: Damit das Beobachter-Muster zur Signalisierung der Agenten umgesetzt werden kann (siehe Abschnitt 4.2.4), benötigen die Beobachter (das heißt die Agenten) eine Referenz auf das Subjekt (hier Luna). Die konkreten Agenten sollen aber unabhängig von der Rahmenarchitektur entwickelt werden können und keinen Zugriff auf dieses besitzen. Daher wird die Referenz innerhalb der abstrakten Agenten erzeugt, die Teil der Rahmenarchitektur sind (siehe Abbildung 4.6). Das bedeutet, in PARSE sind die konkreten Beobachter die abstrakten Agenten. Zudem wird in der Initialisierungsfunktion der abstrakten Oberklasse der initiale Graph hinterlegt. Die Agenten erhalten jeweils eine Kopie des PARSE-Graphen (siehe Abschnitt 4.2.2.2). Zuletzt wird die eigentliche `init`-Methode der Agenten aufgerufen[15]. Nach der Initialisierung wird für jeden Agenten ein Ausführungsfaden (engl. *thread*) erzeugt. Die Fäden werden anschließend nebenläufig zur Ausführung gebracht; hierzu wird die `run`-Methode der abstrakten Agenten aufgerufen[16]. Die `run`-Methode fungiert als Schablonenmethode mit der `exec`-Methode als einzigem Einschub. Innerhalb der Schablonenmethode ist folgender Ausführungszyklus, wie in Algorithmus 1 veranschaulicht, für abstrakte Agenten definiert: In einer Endlosschleife wird das aktuell durch die Rahmenarchitektur versendete Signal geprüft. Luna verwendet die Events *Aktualisierung* und *Terminierung* zur Signalisierung der Agenten. Wird das Event *Terminierung* übermittelt, beenden die Agenten ihre Ausführung. Das Event

[15] Innerhalb des abstrakten Agenten wird die *init*-Methode der implementierten Schnittstelle aufgerufen. Durch die dynamische Bindung entspricht dies zur Laufzeit der `init`-Methode des jeweiligen konkreten Agenten.

[16] Da die abstrakten Agenten in separaten Fäden ausgeführt werden sollen, implementieren sie die *Java*-Schnittstelle `Runnable`.

Aktualisierung macht den Agenten hingegen bekannt, dass eine neue Version des Graphen vorliegt und dementsprechend eine (erneute) Durchführung der Analysen angestoßen werden sollte[17]; Die Schablonenmethode sieht in diesem Fall vor, dass zunächst eine aktuelle Kopie des Graphen von der Rahmenarchitektur angefordert wird. Anschließend wird die Einschubmethode `exec` aufgerufen; der konkrete Agent führt nun seine Sprachanalysen durch und hinterlegt die Ergebnisse in seiner Graphkopie. Ist die Ausführung der `exec`-Methode durch den konkreten Agenten beendet, wird in der Schablonenmethode die `updateGraph`-Methode der Rahmenarchitektur aufgerufen und die Graphkopie des Agenten übergeben. Der Agent bietet also der Rahmenarchitektur seine private Graphkopie als neuen (Haupt-)Graphen an. Die Rahmenarchitektur Luna prüft, ob der Agent der erste ist, der einen neuen Graphen zurückschreiben will[18]. Ist dies der Fall, wird der vom Agenten erzeugte Graph als aktueller Graph in Luna hinterlegt. Andernfalls wird das Zurückschreiben abgelehnt. Das bedeutet, je Ausführungszyklus kann nur ein Agent seine Analyseergebnisse tatsächlich im Graph hinterlegen[19]. Nach einer Änderung des Graphen werden alle Beobachter, sprich die Agenten, informiert, das Event *Aktualisierung* wird übermittelt. Die Rahmenarchitektur macht keine Vorgaben, wann eine Terminierung der Agenten stattfinden soll. Allerdings werden Schnittstellen für sogenannte *Funktionswächter-Agenten* (engl. *watchdog agents*) angeboten. Konkrete Funktionswächter-Agenten sollen prüfen, wann kein Fortschritt mehr bei der Analyse der Semantik erzielt wird. Dazu kann beispielsweise der Graph überwacht werden[20]. Konkrete Funktionswächter-Agenten sind kein Bestandteil von PARSE, sondern müssen separat entwickelt werden. PARSE sieht jedoch vor, dass Funktionswächter-Agenten spezielle Knoten im Graphen hinterlegen. Luna prüft vor jeder Änderungsbenachrichtigung an die Agenten, ob ein solcher Knoten im Graph hinterlegt wurde. Ist dies der Fall, wird den Agenten ein *Terminierungs*-Event übermittelt. Durch die Auslagerung der Prüfung von Terminierungsbedingungen in Funktionswächter-Agenten können jederzeit neue Bedingungen durch die Entwicklung neuer Agenten hinzugefügt werden. Außerdem sind die Bedingungen dadurch nicht bereits Architektur-seitig vorgegeben, sondern können für jedes System passend kombiniert werden.

Nachdem die *EAS* ihre Ausführung beendet hat, wird das Datenobjekt für das Nachverarbeitungfließband erzeugt und der Graph hinterlegt. Anschließend werden die einzelnen Stufen initialisiert und analog zum Vorverarbeitungsfließband ausgeführt. Prinzipiell sind die Stufen des Nachverarbeitungsfließbands für die Generierung eines Ergebnisses bzw. einer Ausgabe zuständig; Luna führt lediglich das Fließband aus und beendet anschließend die Ausführung von PARSE.

[17] Während der Initialisierung wird den Agenten zunächst das Event *Aktualisierung* signalisiert. Somit ist sichergestellt, dass alle Agenten zunächst mit der Ausführung ihrer jeweiligen Analysen beginnen.

[18] Hierzu werden zwei Prüfungen durchgeführt. Luna hinterlegt innerhalb des PARSE-Graphen, wie häufig dieser bereits geändert wurde. Dieser Wert muss bei dem vom Agenten angebotenen Graphen gleich dem Wert des in Luna hinterlegten Graphen sein. Ist dies nicht der Fall, wird der vom Agenten angebotene Graph verworfen. Da Agenten aber auch inhaltlich unveränderte Graphen anbieten können, wird zusätzlich überprüft, ob die Graphen äquivalent sind (siehe Abschnitt 4.2.2.2). Auch in diesem Fall wird der vom Agenten angebotene Graph verworfen.

[19] Dadurch wird sichergestellt, dass keine Inkonsistenzen im Graph entstehen. Die Alternative, eine Zusammenführung mehrerer Graphen, ist hingegen nicht trivial. Durch diesen konservativen Ansatz wird die Verarbeitung innerhalb der *EAS* natürlich ineffizient. Solange die Analysen der Agenten jedoch leichtgewichtig sind und ausreichend Ressourcen (CPUs) verfügbar sind, fällt dies aber nicht ins Gewicht. Nichtsdestotrotz könnten zukünftig Methoden zur Verschmelzung der Graphen entwickelt werden und so die Effizienz gesteigert werden (siehe Abschnitt 10.2).

[20] Mögliche Ausprägungen konkreter Funktionswächter-Agenten könnten beispielsweise nach Ablauf eines Zeitlimits die Terminierung initiieren oder falls nur zyklische Änderungen des Graphen stattfinden (siehe Abschnitt 7.11).

5 ProNat – Ein System zur Endnutzer-Programmierung mit gesprochener Sprache

„Understanding is nothing else than conception caused by speech."

– Thomas Hobbes

Ein Computersystem programmieren, ohne dafür eine Programmiersprache lernen zu müssen – stattdessen einfach in alltäglicher Sprache erklären, was das System tun soll. Mit ProNat wird dies möglich. ProNat ist ein System zur Endnutzer-Programmierung mit gesprochener Sprache; es nimmt Eingaben in Form von natürlichsprachlichen Äußerungen entgegen, analysiert die Semantik dieser und synthetisiert anhand der Ergebnisse Quelltext.

ProNat entsteht auf Grundlage der Architektur PARSE; das bedeutet, es werden konkrete Agenten und Fließbandstufen als Einschübe (engl. *plug-in*) zur Ausführung durch die Rahmenarchitektur Luna bereitgestellt. Weiterhin werden Anbindungen an externe Ressourcen geschaffen. Einerseits erhalten Agenten und Fließbandstufen so Zugriff auf Wissensdatenbanken. Andererseits wird auf diese Weise auch die Anwendungsdomäne über den von PARSE vorgesehene Ontologie-Zugriff angebunden. Da ProNat ein System zur Endnutzer-Programmierung mit gesprochener Sprache darstellt, entspricht die Anwendungsdomäne Anwendungsschnittstellen (engl. *application programming interface*, kurz *API*); gegebenenfalls können in den Ontologien zusätzlich Systemumgebungen modelliert werden.

In den nachfolgenden Abschnitten wird zunächst die Zielbestimmung von ProNat präzisiert (Abschnitt 5.1). Anschließend wird der Forschungsansatz erläutert, welcher der Entwicklung von ProNat zugrunde liegt (Abschnitt 5.2). Daraufhin wird eine Übersicht über die für ProNat entwickelten Einzelkomponenten gegeben (Abschnitt 5.3). Die konkreten Fließbandstufen und Agenten werden dann in den nachfolgenden Kapiteln 6, 7 und 8 beschrieben. Es folgt eine Erläuterung dazu, wie in ProNat die Anwendungsdomänen als Ontologien modelliert und somit von den Sprachanalysen getrennt werden (Abschnitt 5.4). Im letzten Abschnitt wird erläutert, wie mithilfe von Studien zu Entwicklungs- und Evaluationszwecken Datensammlungen, bestehend aus Sprachaufnahmen und textuellen Beschreibungen von Probanden, angelegt werden (Abschnitt 5.5).

5.1 Zielbestimmung

Im Folgenden wird die Zielbestimmung von ProNat erläutert. Zunächst werden die in Abschnitt 1.2 formulierten Ziele präzisiert. Anschließende werden grundlegende Methodiken zur Umsetzung der Ziele erläutert. Abschließend wird diskutiert, welche Einschränkungen für die Umsetzung getroffen werden müssen.

Mit ProNat soll ein System entstehen, dass Endnutzer in die Lage versetzt, Zielsysteme durch gesprochene Anweisungen zu programmieren. Durch die Festlegung auf Endnutzern als Zielgruppe muss ProNat laienhafte Beschreibungen interpretieren können. Zur Verwendung von ProNat soll kein Vorwissen über die Funktionsweise von ProNat oder des Zielsystems nötig sein. Außerdem muss ProNat in der Lage sein, alltägliche gesprochene Sprache zu verstehen. Nutzer von ProNat sollen ihre Anweisungen frei formulieren können; das bedeutet, ProNat soll die Sprache nicht beschränken. Dies schließt eine unbeschränkte Wortwahl und die Verwendung beliebiger Ausdrucksformen bzw. Satzstrukturen ein[1]. Die Beschreibungen sollen inhaltlich beliebig komplex gestaltet werden dürfen; das bedeutet, Nutzer können beispielsweise Rückbezüge durch Korreferenzen oder Konditionalsätze formulieren. Nutzern soll es zudem erlaubt sein, Beschreibungen unbeschränkter Länge abzugeben. Der Umgang mit vielfältiger und ausdrucksstarker Sprache ist nötig, damit Nutzer befähigt werden, umfangreiche Programme beschreiben zu können. Verwandte Ansätze erlauben lediglich die Erzeugung einzelner Anweisungen oder Aufrufe bzw. die Kombination weniger, gegebenenfalls in Verbindung mit einzelnen Kontrollstrukturen. ProNat soll hingegen die Erzeugung komplexerer Programme ermöglichen; es sollen Skripte synthetisiert werden, die aus beliebig vielen Anweisungen und Kontrollstrukturen bestehen. Neben bedingten Verzweigungen, sollen auch Schleifen und nebenläufige Abschnitte definiert und beliebig verschachtelt werden können. ProNat soll die Programmierung von Zielsystemen ermöglichen, die für die Endnutzer-Programmierung geeignet sind und eine entsprechende Anwendungsschnittstelle anbieten. Beispiele für derartige Systeme sind humanoide Roboter, Heimautomationssysteme oder virtuelle Assistenten. Zielsysteme können gegebenenfalls mit ihrer Umgebung interagieren[2]; ProNat muss es dementsprechend ermöglichen, Modelle von Systemumgebungen einzubeziehen. Ein Zielsystem mit einer konkreten Systemumgebung bildet die Anwendungsdomäne. ProNat soll möglichst Domänen-agnostisch sein; das bedeutet, dass bei einer Änderung der Anwendungsdomäne (das heißt, des Zielsystem, der Umgebung oder von beidem) möglichst wenige Änderungen am Gesamtsystem durchgeführt werden müssen.

Zur Umsetzung dieser Ziele kommen in ProNat die im Folgenden beschriebenen Methodiken bzw. Techniken zum Einsatz: Um frei formulierte und beliebig lange Beschreibungen interpretieren zu können, wird ein möglichst tiefes Verständnis der natürlichen Sprache angestrebt. Die Interpretation einer gesprochenen Äußerung als Programm, das beliebig viele Anwendungsschnittstellenaufrufe

[1] Das bedeutet, Beschreibungen können prinzipiell in beliebigen Modi abgegeben werden. Es zeigt sich jedoch, dass Anweisungsbeschreibungen für gewöhnlich im Indikativ-Aktiv oder Passiv formuliert werden. Neben der freien Wahl des Modus können auch beliege Satzstrukturen, einschließlich beliebiger Formen von Nebensätzen, verwendet werden.

[2] Beispielsweise interagieren humanoide Roboter mit Objekten (Erfassen, Greifen, etc.) oder Heimautomationssysteme mit Geräten bzw. Anlagen.

und Kontrollstrukturen enthalten kann, erfordert die Schließung der von Knöll und Mezini beschriebenen *semantischen Lücke*, die zwischen natürlichen Sprachen und Programmiersprachen klafft (siehe Abschnitt 3.3). Natürliche Sprache ist inhärent mehrdeutig; unter anderem können sowohl die Bedeutung einzelner Wörter als auch Satzstrukturen je nach Kontext unterschiedlich ausgelegt werden. Außerdem enthalten natürlichsprachliche Äußerungen häufig implizite Relationen[3]. Programmiersprachen folgen hingegen einer eindeutigen Syntax und Referenzen sind explizit. Das bedeutet, um natürliche Sprache als Programmiersprache verwenden zu können, müssen Mehrdeutigen soweit wie möglich aufgelöst und implizite Relationen expliziert werden. Hierzu kommen in ProNat Agenten zum Einsatz, die unabhängig voneinander bestimmte Aspekte der Semantik natürlichsprachlicher Äußerungen analysieren. Die Agenten können beliebige Ansätze verfolgen; sowohl heuristische Verfahren als auch maschinelles Lernen und auf Inferenz basierende Techniken werden angewandt. Die Semantik-Analysen der Agenten werden soweit wie möglich unabhängig von konkreten Anwendungsdomänen durchgeführt. Dadurch können Agenten wiederverwendet werden, wenn sich die Anwendungsdomäne ändert. Natürlich besteht bei einigen Agenten die Notwendigkeit, Information über die Domäne in die Analyse einzubeziehen[4]. Das Wissen über die Anwendungsdomäne wird in Ontologien modelliert. Die Ontologien besitzen eine vorgegebene Struktur, die den Agenten bekannt gemacht wird; die genauen Inhalte (das heißt die in der Ontologie enthaltenen Individuen) können von Agenten zur Laufzeit abgefragt werden. Dadurch bilden die Ontologien eine Art Schnittstelle zu Domänenwissen[5].

Die Entwicklung von ProNat soll Evaluations-getrieben erfolgen. Das bedeutet, die Entwicklung der einzelnen Bestandteile (Fließbandstufen und Agenten) wird von kontinuierlichen Evaluationen anhand realistischer Eingabedaten begleitet (siehe Abschnitt 5.5). Die Bestandteile werden (größtenteils) unabhängig voneinander und unabhängig von der Anwendungsdomäne evaluiert. Letzteres ist aufgrund der zuvor beschriebenen Trennung von Sprachanalysen und Anwendungsdomäne möglich. Durch die kontinuierliche Evaluation der Einzelbestandteile kann bestimmt werden, wie gut diese ihre jeweilige Analyseaufgabe lösen und welchen Beitrag sie zum Gesamtsystem leisten. Gleichzeitig wird aus Sicht des Gesamtsystems offengelegt, welche Bestandteile Verbesserungspotenzial besitzen, um die Qualität der Gesamtlösung zu verbessern. So wird ein kontinuierlicher Entwicklungsfortschritt sichergestellt und die Erfolgsaussichten des Ansatzes können bereits vor der Evaluation des Gesamtsystems abgeschätzt werden. Durch die Einzelevaluationen der Agenten und Fließbandstufen entsteht zudem sukzessive ein Datensatz für die abschließende Evaluation von ProNat.

Um die anfangs genannten Ziele mit den beschriebenen Methodiken umsetzen zu können, müssen für ProNat Einschränkungen getroffen werden. Die erste betrifft die natürliche Sprache, die als Eingabe akzeptiert wird. ProNat beschränkt sich ausschließlich auf die Interpretation *englischsprachiger* Äußerungen. Die Computerlinguistik-Forschungsgemeinde fokussiert einen Großteil ihrer Bemühung auf die englische Sprache. Folglich stehen Ressourcen, das heißt Werkzeuge, Korpora

[3] Beispiele hierfür sind elliptische Auslassungen oder die Referenzierung von Entitäten anhand eines Oberbegriffs (z. B. „There are *the fox* and *the dog*. *The two animals* stroll about.“).

[4] Beispielsweise muss ein Agent, der Anwendungsschnittstellenaufrufe synthetisiert, wissen, welche Funktionen die Anwendungsschnittstelle anbietet (siehe Abschnitt 7.8).

[5] Den Agenten wird auf diese Weise zum Beispiel bekannt gemacht, dass es ein Zielsystem mit Klassen, Methoden, Parametern usw. gibt. Welche das genau sind, wird hinter der Schnittstelle (das heißt in der Ontologie) verborgen.

und vortrainierte Modelle, vor allem für Englisch zur Verfügung[6]. Da ProNat auf die Verwendung unterschiedlicher Ressourcen angewiesen ist, steht zu erwarten, dass die bestmöglichen Ergebnisse für englischsprachige Eingaben erzielt werden können[7]. Weitere Einschränkungen müssen hinsichtlich des Inhalts der erzeugten Quelltextgenerate getroffen werden. Der Fokus von ProNat richtet sich auf die Programmierung gegen APIs. Daher sind alle durch ProNat synthetisierten Anweisung entweder API-Aufrufe oder Aufrufe selbst definierter Funktionen. Diese entstehen wiederum durch Komposition primitiver Operationen, die vom Zielsystem angeboten werden (siehe Abschnitt 7.8). Die API des Zielsystems muss das passende – das heißt ein vergleichsweise hohes – Abstraktionsniveau aufweisen. Das bedeutet, die angebotenen Funktionen müssen in etwa Handlungen (bzw. Aktionen oder Überprüfungen) entsprechen, die in der Gedankenwelt von Laien vorkommen[8]. Dies ist allerdings üblicherweise bei Anwendungsschnittstellen zur Endnutzer-Programmierung der Fall, auf die ProNat ausgelegt ist[9]. Durch diese Ausrichtung auf Endnutzer-Programmierung werden durch die Verwendung von ProNat keine komplexen Systeme entstehen können. Das bedeutet, es sollen ausschließlich skriptartige Programme synthetisiert werden; die Erzeugung von Modulen (bzw. Klassen) soll nicht unterstützt werden. Außerdem soll der generierte Quelltext vorrangig die vom Nutzer beschriebene Funktionalität umsetzen. Dabei werden keinerlei Effizienzbetrachtungen bei der Synthese einbezogen. Auch Sicherheitsaspekte werden explizit nicht betrachtet.

Im Folgenden werden die für ProNat definierten Ziele, die grundlegenden Methodiken und die getroffenen Einschränkungen in einer Übersicht zusammengefasst:

Ziele:

(Z_1) Programmierung durch gesprochene Anweisungen

(Z_2) Eignung für Endnutzer bzw. Laien (hinsichtlich des Sprachverständnisses)

(Z_3) Erzeugung von beliebig langen Programmen

(Z_4) Erkennung und Synthese von Kontrollstrukturen

(Z_5) Beschreibung neuer Funktionalität (Methodendefinitionen)

(Z_6) Abbildung auf Anwendungsschnittstellenaufrufe (API-Aufrufe)

(Z_7) Adaptierung an andere Domänen möglich (mit geringen Aufwand)

[6] Viele Werkzeuge, wie Systeme zur automatischen Spracherkennung oder Computerlinguistikfließbänder bieten zwar auch die Verwendung von Modellen für andere Sprachen an, die besten Ergebnisse werden aber fast ausschließlich für Englisch erzielt. Auch Korpora stehen prinzipiell auch für andere Sprachen zur Verfügung. Größere Korpora, insbesondere bearbeitete (das heißt mit Etiketten versehene), finden sich aber nur für die englische Sprache. Dementsprechend sind auch vortrainierte Sprachmodelle, die auf ebendiesen Korpora basieren, für das Englische am ausgereiftesten.

[7] Die in den anschließenden Kapiteln beschriebenen Sprachanalysen hängen nur geringfügig von der konkreten natürlichen Sprache ab, z. B. durch die Verwendung von Signalwörtern. Dementsprechend könnte ProNat zukünftig mit (voraussichtlich) geringen Aufwand auch für andere Sprachen eingesetzt werden.

[8] Beispielsweise steht nicht zu erwarten, dass Nutzer die Motorbewegungen eines Roboterarms beschreiben, die zum Greifen eines Objekts nötig sind; stattdessen sollte die API ein Primitiv wie *Greifen* anbieten.

[9] Bietet die API nicht das passende Abstraktionsniveau, können entweder entsprechende Funktionen zur API hinzugefügt werden oder bei der Erstellung der Zielsystemontologie passende Primitive angelegt werden, die aus API-Aufrufen zusammengesetzt sind (siehe Abschnitt 5.4).

Methodiken:

- (M_1) Erzeugung von tiefem Sprachverständnis
- (M_2) Verwendung von spezialisierten Agenten zur Analyse der Semantik
- (M_3) Trennung von Sprachanalysen und Domänenwissen
- (M_4) Evaluations-getriebene Entwicklung

Einschränkungen:

- (E_1) Unterstützung von ausschließlich englischsprachigen Eingaben
- (E_2) Beschränkung auf Zielsysteme, die Programmierung durch Endnutzer erlauben
- (E_3) Beschränkung auf Anwendungsschnittstellen mit hohem Abstraktionsniveau
- (E_4) Alle Anweisungen entsprechen Anwendungsschnittstellenaufrufen (API-Aufrufe)
- (E_5) Keine Erzeugung komplexer, effizienter oder sicherheitskritischer Programme
- (E_6) Beschreibung von Modulen (bzw. Klassen) nicht möglich

5.2 Forschungsansatz

Mit ProNat soll ein System zur Endnutzer-Programmierung mit gesprochener Sprache entstehen. Grundlegend basiert der Ansatz zum einen auf der Verwendung von Agenten, die unabhängig voneinander unterschiedliche sprachliche Phänomene analysieren und im Zusammenspiel ein tiefes Verständnis natürlichsprachlicher Sprache ermöglichen. Zum anderen wird die Anwendungsdomäne von den Sprachanalysen getrennt, wodurch diese allgemeingültig angelegt werden können.

Das System ProNat entsteht durch konkrete Implementierungen von Modulen, wie sie von der Architektur PARSE vorgesehen sind. Das bedeutet, es werden Fließbandstufen und Agenten als Einschübe (für die Rahmenarchitektur Luna) bereitgestellt, die jeweils eine dedizierte Sprachanalyseaufgabe lösen (siehe Abschnitt 4.2.5). Außerdem werden externe Ressourcen zur Verfügung gestellt. Zum einen werden Wissendatenbanken, wie zum Beispiel *WordNet*, *DBpedia* oder *PropBank*, zur Verfügung gestellt. Zum anderen wird die Repräsentation der Anwendungsdomäne angebunden. Diese wird in ProNat als Ontologie mit festgelegter Struktur umgesetzt; dadurch entsteht eine Art Schnittstelle zu Domänenwissen (siehe Abschnitt 5.4). Die Anwendungsdomänen in ProNat untergliedern sich in ein zu programmierendes Zielsystem (z. B. ein humanoider Roboter) und eine optionale Systemumgebung (z. B. eine Küche mit Geräten, Orten und Objekten). Im nachfolgenden Abschnitt werden die einzelnen Komponenten von ProNat und in Abschnitt 5.4 konkrete Anwendungsdomänen beschrieben. Die Entwicklung von ProNat erfolgt Evaluations-getrieben. Das bedeutet, jedes Modul soll soweit wie möglich für sich evaluiert werden; dies gilt insbesondere für die Agenten für tiefes Sprachverständnis. Evaluiert werden die Agenten anhand von Sprachaufnahmen, die Nutzerstudien entstammen. Für nahezu jeden Agenten wird eine Studie durchgeführt, wodurch Korpora entstehen. Der jeweilige Agent kann anhand der jeweils gesammelten Daten

intrinsisch evaluiert werden. Dadurch entsteht sukzessive ein Gesamtkorpus, das für die Evaluation des Gesamtsystems verwendet werden kann (siehe Kapitel 9). Die Nutzerstudien und die daraus entstandenen Korpora werden in Abschnitt 5.5 beschrieben.

5.3 Komponentenübersicht

Das System ProNat entsteht durch die Bereitstellung von Einschüben für die Rahmenarchitektur Luna; konkrete Einschübe sind entweder Agenten oder Fließbandstufen (siehe Abschnitt 4.2.5). Die Rahmenarchitektur sorgt dafür, dass die in Einschüben umgesetzten Analysen ausgeführt werden. Zunächst werden alle Stufen des von der Architektur PARSE vorgesehen Vorverarbeitungsfließbandes nacheinander ausgeführt. Das Vorverarbeitungsfließband ist Architektur-seitig dafür vorgesehen, die Eingabe entgegenzunehmen und einen initialen Graphen zu erzeugen. Für das System ProNat bedeutet das: Ein Audiosignal wird entgegengenommen und daraus eine Tokensequenz erzeugt; diese wird anschließend mithilfe von grundlegenden syntaktischen Analysen vorverarbeitet und aus den gesammelten Informationen ein Graph generiert. Anschließend werden die bereitgestellten Agenten innerhalb der Einheit zur Analyse der Semantik parallel und wiederholt ausgeführt (siehe Abschnitt 4.2.4). Die für ProNat entwickelten konkreten Agenten werden in der Folge als ProNat-Agenten bezeichnet. dasselbe gilt das Graphmodell von PARSE: die konkrete Instanz wird als ProNat-Graph bezeichnet. Die Agenten analysieren die Semantik und erzeugen im Zusammenspiel ein tiefes Verständnis der natürlichsprachlichen Äußerung. ProNat-Agenten lassen sich gemäß ihres Zwecks in drei Kategorien unterteilen: Agenten für generelles Sprachverständnis (z. B. zur Auflösung sprachliche Referenzen), Agenten zur Erkennung programmatischer Strukturen (z. B. zur Erkennung von Kontrollstrukturen) und funktionale Agenten (z. B. zur Signalisierung der Terminierung). Nach Beendigung der semantischen Analyse wird aus dem Analyseergebnis schrittweise Quelltext erzeugt. Hierzu führt Luna die für das Nachverarbeitungsfließband bereitgestellten Stufen aus. Die Ausgabe der letzten Stufe ist ein Quelltextgenerat für ein konkretes Zielsystem, das in einer konkreten Umgebung agiert. Damit dies gelingen kann, müssen den Agenten und Fließbandstufen Informationen über die Anwendungsdomäne (Zielsystem und Systemumgebung) bekannt gemacht werden. Domänenwissen wird in ProNat, wie von der Architektur PARSE vorgesehen, in Ontologien modelliert. Für jede Anwendungsdomäne muss eine Ontologie bereitgestellt werden. Um ProNat zur Programmierung eines Zielsystems in einer bestimmten Umgebung verwenden zu können, muss ProNat mit der passenden Anwendungsdomäne konfiguriert werden [10] [11]. Zusätzlich werden weitere externe Ressourcen, wie Wissensdatenbanken, den Agenten und Fließbandstufen über die Rahmenarchitektur Luna bereitgestellt.

[10] Anstatt ProNat mit einer Domänenontologie zu konfigurieren, kann auch ein Agent verwendet werden, der zur Laufzeit für eine natürlichsprachliche Äußerung automatisch das wahrscheinlichste Zielsystem und die passende Systemumgebung auswählt (siehe Abschnitt 7.9).

[11] Anstatt pro Domäne eine Ontologie zu verwenden, könnten auch alle Informationen über Zielsysteme und Umgebungen in einer einzigen Ontologie repräsentiert werden. Dadurch entfiele die Notwendigkeit ProNat mit einer Ontologie zu konfigurieren. Andererseits sind Suchen nach Individuen (anhand von Zeichenkettenähnlichkeiten) in einer großen Ontologie unschärfer; je Suchanfrage werden deutlich mehr Ergebnisse zurückgeliefert (siehe Abschnitt 5.4.2).

Das System ProNat setzt sich dementsprechend aus den folgenden konkreten Bestandteilen (der Architektur PARSE) zusammen:

- einer Menge von Vorverarbeitungsfließbandstufen,
- einer Menge von Agenten,
- einer Menge von Nachverarbeitungsfließbandstufen,
- genau einer Domänenontologie (austauschbar) und
- weiteren externen Ressourcen.

Nachfolgend wird zunächst in Abschnitt 5.4 beschrieben, wie die Anwendungsdomänen für ProNat als Ontologien repräsentiert werden. Die für ProNat entwickelten Einschübe werden in den darauffolgenden Kapiteln ausgeführt: die Vorverarbeitungsfließbandstufen in Kapitel 6, die Agenten in Kapitel 7 und zuletzt die Nachverarbeitungsfließbandstufen in Kapitel 8[12].

5.4 Repräsentation der Anwendungsdomäne

Anwendungsdomänen werden in ProNat als Ontologien repräsentiert. Die Struktur der Ontologien, das heißt die Klassenhierarchie und mögliche Relationen, wird vorab festgelegt. Diese Struktur wird den anderen Modulen von ProNat bekannt gemacht; verborgen wird (während der Entwicklung) hingegen der konkrete Inhalt, das heißt die Individuen innerhalb der Klassen und Ausprägungen von Relationen zwischen den Individuen. Die Agenten und Fließbandstufen können allerdings zur Laufzeit Informationen abfragen. Beispielsweise können alle Individuen einer Klasse zurückgeliefert oder mithilfe von Zeichenkettenähnlichkeitsmetriken (siehe Abschnitt 2.3.13) nach Individuen gesucht werden. Dadurch können Agenten und Fließbandstufen auf Domänenwissen zugreifen, ohne vorab Annahmen über konkrete Inhalte treffen zu müssen. Das bedeutet, die in ProNat verwendeten Domänenontologien bilden eine Art Schnittstelle zu Domänenwissen. Der Vorteil der Trennung von Sprachanalysen und Domänenwissen ist, dass Agenten und Fließbandstufen unabhängig von konkreten Anwendungsdomänen entwickelt werden können. Ändert sich die Anwendungsdomäne, in der ProNat verwendet wird, muss lediglich die Ontologie ausgetauscht werden; das übrige System kann prinzipiell unverändert weiterverwendet werden.

ProNat unterteilt Anwendungsdomänen in Zielsysteme und Systemumgebungen. Zielsysteme sind beliebige (Computer-)Systeme, die sich zur Endnutzer-Programmierung eignen und eine entsprechende Anwendungsschnittstelle bereitstellen. Mögliche Zielsysteme sind beispielsweise humanoide Robotersysteme, Heimautomationssysteme oder virtuelle Assistenzsysteme. Systemumgebungen sind hingegen Sammlung von Objekten, die nicht Teil eines Zielsystems sind, mit denen Systeme interagieren können. Beispielsweise könnte ein Roboter in unterschiedlichen Räumen eines Hauses agieren; jeder Raum sowie enthaltene Orte, Objekte etc. könnten als Systemumgebungen modelliert

[12] Die Anbindung der externen Ressourcen wird nicht dediziert beschrieben. Stattdessen erfolgt die Beschreibung dann, wenn eine Fließbandstufe oder ein Agent eine externe Ressource verwendet.

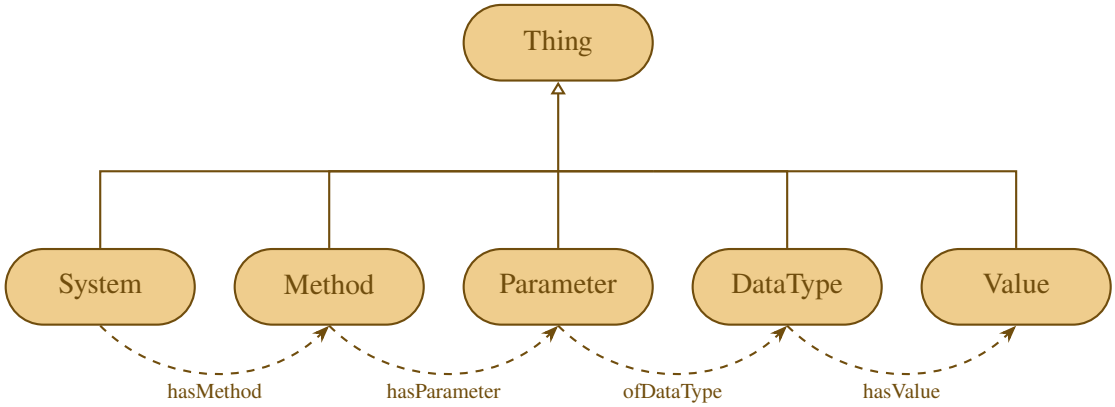


Abbildung 5.1: Struktur der für ProNat verwendeten Zielsystemontologien.

werden. Für ein virtuelles Assistenzsystem können hingegen Applikationen, mit denen das Assistenzsystem interagiert, als Umgebung modelliert werden; beispielsweise könnte eine Umgebung *Kalender* mit den Objekten *Termin*, *Uhrzeit* etc. verwendet werden. Aufgrund dieser Aufteilung der Anwendungsdomänen werden für ProNat auch zwei Arten von Ontologien verwendet. Dadurch können konkrete Anwendungsdomänen flexibel gestaltet werden: Zielsysteme können in unterschiedlichen Umgebungen verwendet werden und umgekehrt. Eine konkrete Anwendungsdomäne wird gebildet, indem Zielsystem- und Umgebungsontologien fusioniert werden (engl. *ontology merging*). Für ProNat wurden vier konkrete Zielsystem- und neun Umgebungsontologien erstellt. Diese können prinzipiell beliebig kombiniert werden (wobei nicht alle Kombinationen sinnhaft sind). Nachfolgend werden zunächst die für ProNat festgelegten Strukturen der Zielsystem- und Umgebungsontologien erläutert (Abschnitt 5.4.1). Anschließend wird beschrieben, wie auf Domänenwissen zugegriffen werden kann (Abschnitt 5.4.2). Zuletzt werden die konkreten, für ProNat erstellten Ontologien vorgestellt (Abschnitt 5.4.3).

5.4.1 Struktur der Ontologien

Hinsichtlich der Struktur der für ProNat verwendeten Ontologien werden zwei Vorgaben gemacht: Zum einen wird die Klassenhierarchie festgelegt; zum anderen werden vorab die Relationen zwischen Klassen und damit die potenziellen Relationen zwischen Individuen der jeweiligen Klassen bestimmt. ProNat sieht die Verwendung von zwei Arten von Ontologien vor: Zielsystem- und Umgebungsontologien. Beide besitzen eine unterschiedliche, aber feste Struktur. Abbildung 5.1 zeigt die für Zielsysteme vorgesehene Struktur. Wie zu sehen ist, besitzen die Zielsystemontologien eine flache Klassenhierarchie. Unterhalb des obersten Konzepts der Ontologie (`Thing`) werden direkt alle Unterklassen als disjunkte (Geschwister)-Klassen angeordnet. Diese repräsentieren die Elemente der Anwendungsschnittstelle: Klassen bzw. Module (`System`), Methoden (`Method`), Parameter (`Parameter`), Datentypen (`DataType`) und Werte bzw. Wertebereiche (`Value`). Zusätzlich wird durch die Relationen modelliert, dass Klassen öffentliche Methoden anbieten (`hasMethod`), Methoden Parameter besitzen können (`hasParameter`), Parameter einem gewissen Datentyp zugehörig sind (`ofDataType`) und Datentypen bestimmte Werte annehmen können (`hasValue`). Die

Tabelle 5.1: Konzepte der für ProNat festgelegten Zielsystemontologiestruktur: Neben der Struktur ist für jedes Konzept eine Kurzbeschreibung der enthaltenen Individuen angegeben; zusätzlich werden in Klammern jeweils beispielhafte Individuen genannt.

Konzept	Beschreibung
Thing	Oberstes Konzept der Ontologie
∟ System	Systeme und Subsysteme, d.h. API-Klassen bzw. Module (z. B. *Robot, Drone* oder *Alexa*)
∟ Method	Systemfunktionen, d. h. API-Methoden (z. B. *move, open, makeVideo* oder *playSong*)
∟ Parameter	Parameternamen, die in den API-Methoden auftreten (z. B. *move.where, move.howFast* oder *play.what*)
∟ DataType	Datentypen, die dem System bekannt sind (z. B. *int, String, Speed* oder *Graspable*)
∟ Value	Werte bzw. Wertebereiche, die Datentypen annehmen können (z. B. *int_range, Speed.FAST* oder *Speed.SLOW)*
∟ Object	externe Umgebungsobjekte <hier leer>
∟ State	Zustände der Umgebungsobjekte <hier leer>

Modellierung von Anwendungsschnittstellen ist damit vergleichsweise einfach gehalten; verwandte Arbeiten, wie beispielsweise der Ansatz von Atzeni und Atzori [AA18], verwenden aufwendigere Modellierungen. Für ProNat wurde diese generische Darstellungsform gewählt, um möglichst viele unterschiedliche Typen von APIs und Programmiersprachen repräsentieren zu können; außerdem ist diese Modellierung für die hiesigen Zwecke ausreichend.

Tabelle 5.1 führt die einzelnen Klassen und den jeweiligen Zweck für die Modellierung von Zielsystemen auf. In der Tabelle sind in den letzten beiden Zeilen zwei zusätzliche Klassen aufgeführt: `Object` und `State`. Diese werden verwendet, wenn das Zielsystem in Verbindung mit einer Umgebung eingesetzt werden soll. Die beiden Klassen entsprechen den auf oberster Hierarchieebene angesiedelten Klassen der Umgebungsontologien, wie in Tabelle 5.2 gezeigt. Durch die Struktur von Umgebungsontologien wird dementsprechend festgelegt, dass eine Umgebung aus Objekten (`Object`) besteht, die Zustände annehmen können (`State`). Innerhalb der Objekt-Klasse können Unterklassen angelegt werden, um Objekte zu typisieren. Auf diese Weise können auch andere Umgebungselemente modelliert werden; beispielsweise könnten Orte bzw. Positionen mithilfe eines Objekttyps `Location` repräsentiert werden. Es können beliebig viele Unterklassen der Objekt-Klasse definiert werden; eine weitere Hierarchieebene ist nicht vorgesehen. Innerhalb der Zielsystemontologien bleiben die beiden Klassen `Object` und `State` leer und bilden so die Ansatzpunkte für die Fusionierung mit Umgebungsontologien, wie am Ende des Abschnitts beschrieben.

Konkrete Zielsystem- und Umgebungsontologien entstehen, in dem die vordefinierten Strukturen mit Individuen befüllt werden. Im Fall der Zielsysteme sind die Ontologie-Individuen Repräsentationen konkreter API-Elemente, das heißt Klassen, Methoden, Parametername etc., der jeweiligen Anwendungsschnittstelle. Die Individuen enthalten einen Rückverweis auf das tatsächliche API-Element,

Tabelle 5.2: Konzepte der für ProNat festgelegten Umgebungsontologiestruktur: Neben der Struktur ist für jedes Konzept eine Kurzbeschreibung der enthaltenen Individuen angegeben; zusätzlich werden in Klammern jeweils beispielhafte Individuen genannt.

Konzept	Beschreibung
Thing	Oberstes Konzept der Ontologie
∟ Object	Umgebungsobjekte (z. B. *Table, Kitchen.CeilingLight* oder *Date*)
∟ Graspable	Objekte, die greifbar sind (z. B. *Cup, Fork* oder *Fridge.Door*)
∟ Openable	Objekte, die geöffnet werden können (z. B. *WaterBottle, Microwave.Door* oder *Fridge.Door*)
∟ Closeable	Objekte, die geschlossen werden können (z. B. *OrangeJuice, Cupboard.Door* oder *Dishwasher.Door*)
...	...
∟ State	Zustände der Umgebungsobjekte (z. B. *opened, closed, clean* oder *dirty*)

damit innerhalb des Nachverarbeitungsfließbands API-Aufrufe synthetisiert werden können (siehe Abschnitt 8.3). Die Modellierung der API-Elemente kann manuell erstellt werden; dies ist jedoch aufwendig und daher nur für Anwendungsschnittstellen mit geringem Umfang umsetzbar. Alternativ können API-Elemente beispielsweise mithilfe von Zerteilern (engl. *parser*) ausgelesen und die Modellierung automatisiert werden. Werkzeugen zur automatischen Überführung von API-Elementen in ein Ontologiemodell können mit geringem Aufwand erstellt werden. Beispielsweise umfasst das entsprechende Werkzeug zum Auslesen von *Java*-Schnittstelen lediglich 436 Quelltextzeilen [WLB19]. Bei der automatischen Überführung entsprechen die Namen der Ontologie-Individuen den Bezeichnern der API-Elemente; das bedeutet, Zielsystemontologien können nur automatisch erzeugt werden, wenn die API sprechende Namen verwendet[13]. Sind die Benennungen der Individuen nach der automatischen Überführung nicht optimal, können sie nachträglich händisch verbessert werden. Konkrete Umgebungsontologien können auf ähnliche Weise erstellt werden. Entweder können vorhandene Modellierungen mithilfe eines Konvertierungswerkzeugs automatisch in das ProNat-Format überführt werden (wobei der Aufwand von der jeweiligen Modellierung abhängt). Alternativ ist auch für die Umgebungsontologien eine manuelle Modellierung möglich.

Zur Bildung einer konkreten Anwendungsdomäne wird eine Systemontologie mit beliebig vielen Umgebungsontologien fusioniert (also gegebenenfalls auch keiner)[14]. Diese Fusionierung ist erforderlich, da PARSE die Verwendung von genau einer Ontologie zur Repräsentation von

[13] Einige ProNat-Agenten bilden natürlichsprachliche Sequenzen auf Ontologie-Individuen ab. Die Abbildung erfolgt (unter anderem) anhand der Namen der Individuen, weshalb eine sinnhafte Namensgebung für diese erforderlich ist (siehe beispielsweise Abschnitt 7.8).

[14] Manche Anwendungsdomänen bestehen ausschließlich aus einer Anwendungsschnittstelle; es findet keine Interaktion mit der Umgebung statt. Andererseits können manche Systeme gleichzeitig in mehreren Umgebungen verwendet werden. Beispielsweise könnte ein humanoider Roboter sowohl in der Umgebung *Küche* als auch in der Umgebung *Wohnzimmer* verwendet werden.

Domänenwissen vorsieht (siehe Kapitel 4). Zur Fusionierung wird eine neue Ontologie-Instanz erzeugt, die der zu verwendenden Zielsystemontologie entspricht. In diese Ontologie werden anschließend die Inhalte der ausgewählten Umgebungsontologien in die dafür vorgesehenen Platzhalter-Klassen eingefügt (siehe Tabelle 5.1). Außerdem wird für die Unterklassen der `Objekt`-Klasse der Umgebungsontologien jeweils ein Individuum innerhalb der `DataType`-Klasse der Zielsystemontologie angelegt. Dadurch können Objekttypen als Datentypen verwendet werden. Die Fusionierung der Ontologien kann entweder manuell durchgeführt werden. Als Alternative wurde ein ProNat-Agenten entwickelt, der automatisch zur Laufzeit anhand der Themen einer natürlichsprachlichen Äußerung Zielsystem- und Umgebungsontologien bestimmen kann (siehe Abschnitt 7.9).

5.4.2 Steuerung des Zugriffs auf Domänenontologien

Die Domänenontologien werden innerhalb eines Moduls geladen und mithilfe der Rahmenarchitektur Luna den Agenten und Fließbandstufen zur Verfügung gestellt. Das Modul bietet eine Schnittstelle für geregelte Zugriffe auf Domänenwissen. Ontologien werden in ProNat im *OWL*-Format abgelegt (siehe Abschnitt 2.3.14). Das Ontologie-Modul ermöglicht Agenten und Fließbandstufen den lesenden Zugriff auf die Ontologie-Darstellung im *OWL*-Format. Da die Ontologien in ProNat festlegten Strukturen folgen, kann das Ontologie-Modul weitere Zugriffsmöglichkeiten anbieten. Unter anderem können alle Methoden des Zielsystems zurückgegeben werden. Ebenso können Relation dazu genutzt werden, für eine Methode alle zugehörigen Parameter zu liefern. Gleiches wird für die anderen Klassen und Relationen der Ontologien angeboten. Darüber hinaus kann in der Ontologie und in einzelnen Klassen (anhand der Namen) nach Individuen gesucht werden; Suchen liefern immer Mengen von Individuen. Neben der Möglichkeit, anhand exakter und teilweiser Übereinstimmung der Zeichenketten zu suchen, können auch sogenannten *Suchstrategien* verwendet werden. Suchstrategien bestehen aus einem Ähnlichkeitsmaß für Zeichenketten und einem zugehörigen Schwellenwert. Ein Individuum wird bei Verwendung einer Suchstrategie genau dann in die Ergebnismenge aufgenommen, wenn der Wert für das Ähnlichkeitsmaß beim Vergleich des Suchbegriffs mit dem Individuen-Namen den Schwellenwert übersteigt. Folgende Ähnlichkeitsmaße werden für die Suchstrategien angeboten:

- Levenshtein-Ähnlichkeit (siehe Abschnitt 2.3.13.1),
- Jaro-Winkler-Ähnlichkeit (siehe Abschnitt 2.3.13.2.) und
- unscharfe Ähnlichkeitsbewertung (engl. *fuzzy score*, siehe Abschnitt 2.3.13.3).

Die Verwendung der Suchstrategien hat den Vorteil, dass nicht nur bloße Mengen von Individuen geliefert werden, sondern die einzelnen Individuen zusätzlich eine Konfidenz erhalten; die Konfidenz entspricht dem erzielten Wert für das Ähnlichkeitsmaß.

Tabelle 5.3: Anzahl der in den Konzepten enthaltenen Individuen je Zielsystemontologie.

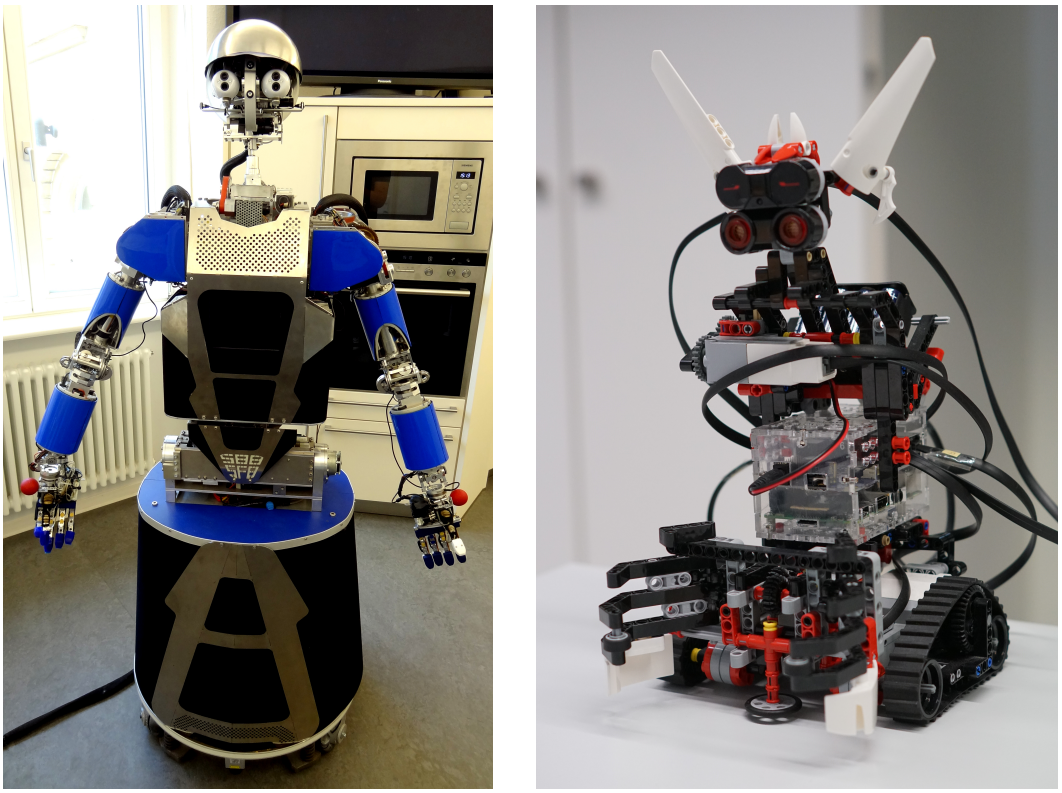
	Systeme	Methoden	Parameter	Datentypen	Werte
Humanoider Roboter	3	92	59	14	34
Lego Mindstorms	1	39	57	21	30
Flugdrone	4	15	3	9	9
Virtueller Assistent	2	14	9	17	22

5.4.3 Verwendete Ontologien

Zu Demonstrations- und Evaluationszwecken wurden für ProNat vier exemplarische Zielsystem- und neun Umgebungsontologien erstellt. Die Zielsysteme umfassen zwei Robotersysteme sowie eine (zivile) Flugdrohne und einen virtuellen Assistenten. In Tabelle 5.3 sind die Zielsysteme und die jeweils zur Modellierung erstellten Individuen für die einzelnen Klassen aufgeführt. Das erste System ist ein humanoider Roboter, dessen Funktionsweise stark an den am KIT entwickelten *ARMAR-III* angelehnt ist [Asf+06]. Abbildung 5.2 zeigt eine Foto von *ARMAR-III*. Der Haushaltsroboter agiert vorwiegend in einer eigens angefertigten Küchenumgebung. In dieser kann er sich frei bewegen und mit seiner Umgebung interagieren. *ARMAR-III* kann unter anderem Objekte erkennen und greifen, aber auch komplexere Aktionen durchführen, wie beispielsweise Schränke öffnen oder Geräte bedienen. Mit *ArmarX* wird ein Programmiersystem bereitgestellt, das es erlaubt Aktionen mit hohem Abstraktionsniveau zu erstellen, wie das *Greifen* oder *Lokalisieren* von Objekten [Wel+13]. Ausgehend von den Möglichkeiten von *ArmarX* wurde die Zielsystemontologie *Humanoider Roboter* händisch erstellt. Die Funktionen wurden so modelliert, dass eine Umsetzung mit *ArmarX* möglich ist. Das bedeutet, das Abstraktionsniveau wurde so gewählt, dass primitive Aktionen ausgeführt werden, die von Menschen beschrieben werden können. Es wurden aber keine komplexen Aktionen modelliert, die eine Reihe primitiver Aktionen erfordern. Nahezu alle Funktionen wurden in mehreren Varianten, das heißt als überladene Methoden, modelliert. Dadurch entsteht ein Funktionsumfang von 92 Methoden. Unter anderem enthält die Ontologie die folgenden Funktionen:

- `move(where)`
- `move(howFast)`
- `move(where, howFast)`
- `locate(what)`
- `locate(what, where)`

Der verhältnismäßig große Funktionsumfang und die vielen Methodenüberladungen wurden so umgesetzt, da diese Zielsystemontologie vorrangig zur Evaluation des Gesamtsystems ProNat genutzt werden soll (siehe Kapitel 9). Damit die Quelltextgenerierung nicht trivial ist, sollte die Ontologie einen gewissen Mindestumfang und eine gewisse Komplexität aufweisen. Die modellierten Datentypen entsprechen hauptsächlich (primitiven) Standarddatentypen, wie *Integer* und *String*. Zusätzlich wurden Enumerationen zur Repräsentation von Geschwindigkeiten, Farben etc. erstellt.



(a) ARMAR-III **(b)** Lego-Mindstorms-Roboter

Abbildung 5.2: Fotos des *ARMAR-III* (a) und des *Lego-Mindstorms*-Roboters (b).

Die zweite Zielsystemontologie modelliert die *Java*-API eines *Lego-Mindstorms*-Roboters[15]. *Lego-Mindstorms*-Roboter bestehen aus Lego-Steinen, Sensoren, Motoren und einer Steuereinheit. Das in Abbildung 5.2 gezeigte Modell kann sich auf einem Kettenfahrwerk fortbewegen und Farben, Linien sowie Objekte erkennen. Außerdem kann der Roboter mithilfe eines Greifers kleine Objekte anheben. Als Steuereinheit wird nicht die von Lego vorgesehen sondern eine freiprogrammierbare verwendet[16] Dadurch können Anwendungsschnittstellen in verschiedenen Programmiersprachen genutzt werden, um den Roboter zu steuern. Die für ProNat verwendete Ontologie zur Modellierung des *Lego-Mindstorms*-Roberts wurde automatisch anhand einer *Java*-API generiert. Hierzu wurde ein Werkzeug entwickelt, das beliebige *Java*-APIs auslesen und automatisch in *OWL*-Ontologie-Repräsentation gemäß der in Abschnitt 5.4.1 definierten Struktur überführen kann. Die Anwendungsschnittstelle befindet sich auf einem ähnlichen Abstraktionsniveau wie die für den humanoiden Roboter modellierte. Das Zielsystem verfügt jedoch über weniger Methoden; diese sind allerdings stärker parametrisiert. Außerdem verwendet die Schnittstelle eine Reihe Enumerationen zur Modellierung von Geschwindigkeitsstufen, Farben, Richtungen etc. Daher enthält die Ontologie viele explizite Werte.

[15] *Lego Mindstorms EV3*: `https://education.lego.com/en-us/products/lego-mindstorms-education-ev3-core-set/5003400#lego-mindstorms-education-ev3`, zuletzt besucht am 24.02.2021.

[16] Verwendet wird die sogenannte *BrickPi*-Steuereinheit, die auf einem *Raspberry Pi 3 Model B* basiert: `https://www.dexterindustries.com/brickpi/`, zuletzt besucht am 24.02.2021.

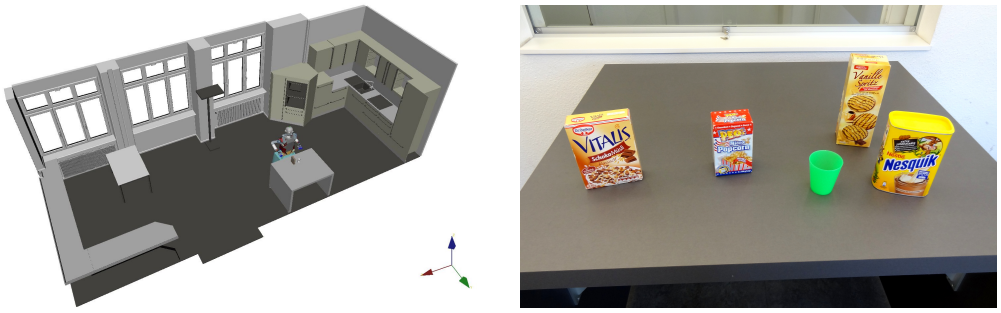
Tabelle 5.4: Anzahl der in den Konzepten enthaltenen Individuen und Unterkonzepte (das heißt Objekttypen) je Umgebungsontologie.

	Objekte	Objekttypen	Zustände
Küche	70	6	10
Bar	56	8	0
Garten	24	6	0
Schlafzimmer	26	3	0
Kinderzimmer	15	4	0
Waschküche	11	6	6
Music	37	4	0
Heizung	6	2	2
Parcours	13	1	0

Zusätzlich zu den beiden Robotern wurden zwei synthetische Zielsysteme entwickelt: eine Flugdrohne und ein virtueller Assistent. Synthetisch bedeutet in diesem Zusammenhang, dass die beiden Ontologien keiner existierenden Anwendungsschnittstelle entsprechen. Die enthaltenen API-Elemente sind aber an reale Systeme angelehnt; das bedeutet, es werden Methoden- und Parameternamen verwendet, die in ihrer Form den üblichen Konventionen entsprechen. Die Ontologie für die Flugdrohne enthält Funktionen zur Flugsteuerung, sowie zur Erstellung von Foto- und Videoaufnahmen. Der modellierte virtuelle Assistent ist an *Amazon Alexa* angelehnt (siehe Abschnitt 3.4). Allerdings beschränkt sich die Funktionalität hier auf Uhrzeitfunktionen (Wecken, Timer stellen etc.) sowie Funktionen zum Abspielen von Musik und zur Steuerung eines Heimautomationssystems.

An Umgebungsontologien wurden zunächst sechs räumliche Umgebungen erstellt: eine Küche, eine Bar, ein Garten, ein Schlafzimmer, ein Kinderzimmer und eine Waschküche. Die Umgebungen enthalten Objekte, bedienbare Geräte und Orte. Objekte und Geräte können gegebenenfalls Zustände annehmen. Beispielsweise kann ein Ofen ein- oder ausgeschaltet und eine Flasche geöffnet oder geschlossen sein. Zusätzlich sind auch Teil-Ganzes-Beziehungen in den Ontologien modelliert; eine Kühlschranktür ist beispielsweise ein Teil eines Kühlschranks. Neben den räumlichen Umgebungsmodellierungen wurden zwei abstrakte Systemumgebungen erstellt: Musik und Hausgeräte. In der Musik-Ontologie sind Eigenschaften von Musikstücken, wie Genrezugehörigkeit oder Interpreten modelliert. Die Hausgeräte-Ontologie enthält eine Menge von Geräten, die mit einer Heimautomationssoftware gesteuert werden können. Zuletzt steht mit der *Parcours*-Ontologie eine Umgebung zur Verfügung die eine Menge kleiner Objekte enthält, die der *Lego-Mindstorms*-Roboter erkennen und greifen kann, beispielsweise eine Tasse, ein Schwamm und ein Glas.

Die Umgebung *Küche* entspricht im Wesentlichen der Küchenumgebung in der *ARMAR-III* üblicherweise agiert. Abbildung 5.3 zeigt ein 3D-Modell und greifbare Objekte der Küchenumgebung. Neben den in der Küche vorhanden Geräten und Objekten wurden zusätzliche Objekte modelliert. Diese sind jedoch so gewählt, dass sie sich in das Szenario einfügen. Beispielsweise wurde ein Objekt `AppleJuice` (dt. *Apfelsaft*) in Anlehnung an das tatsächlich in der Küche vorhandene Objekt `OrangeJuice` (dt. *Orangensaft*) erstellt. Die Objekte der Umgebung *Parcours* sind ausnahmslos Realweltobjekte, wie zuvor genannt. Die weiteren Umgebungsontologien enthalten imaginäre



(a) Küchenmodell **(b)** Greifbare Objekte

Abbildung 5.3: 3D-Modell der Küche, in der *ARMAR-III* agiert (a) und Foto fünf exemplarischer Objekte, die der Roboter erkennen und greifen kann (b).

Objekte und Zustände. Bei der Modellierung wurde darauf geachtet, die Umgebungen mit sinnhaften und nachvollziehbaren Objekten und Zuständen zu befüllen[17]. Aus den vorhandenen Zielsystem- und Umgebungsontologien können Anwendungsdomänen erzeugt werden, indem eine Zielsystem- mit beliebig vielen Umgebungsontologien fusioniert wird (siehe Abschnitt 5.4.1). Dabei stellt nicht jede Kombination eine sinnhafte Anwendungsdomäne dar. Beispielsweise kann der humanoide Roboter nicht mit der Musik-Umgebung oder der virtuelle Assistent nicht mit Küchen-Objekten interagieren. Der *Lego-Mindstorms*-Roboter ist gänzlich auf die *Parcours*-Umgebung beschränkt. Andererseits kann die gleichzeitige Verwendung mehrerer Umgebungen hilfreich sein: der humanoide Roboter könnte innerhalb einer Anweisungsfolge in unterchiedlichen Umgebungen agieren, beispielsweise im *Kinderzimmer* und im *Schlafzimmer*. Werden die beiden jeweils umfangreichsten Zielsystem- und Umgebungsontologien fusioniert, entsteht die Anwendungsdomäne *humanoider Roboter in einer Küchenumgebung*[18]. Bei der Fusionierung werden die Individuen (und Unterklassen) der Umgebungsontologien in die dafür vorgesehenen Platzhalter-Klassen der Zielsystemontologie kopiert. Zusätzlich werden die Objekttypen (Unterklassen der Objekt-Klassen) als zusätzliche Datentypen und die enthaltenen Umgebungsojekte als Werte des Zielsystems modelliert. Die so entstandene Domänenontologie hat folgenden Umfang (Individuen je Klasse): drei Systeme, 92 Methoden, 59 Parameternamen, zwanzig Datentypen und 104 Werte und Wertebereiche. Aus dieser fusionierten Ontologie können unter anderem die folgende Informationen ausgelesen werden:

- Es gibt ein Roboter-System (Individuum `Armar`), das verschiedene Funktionen anbietet.
- Eine der Funktionen ist das Greifen (`grasp`).
- Diese Funktion hat einen formalen Parameter (erwartet also ein Argument): das zu greifende Objekt (`grasp.what`).
- Der Datentyp dieses Parameters ist `Graspable`.
- Greifbare Objekte in der Umgebung sind unter anderem `AppleJuice`, `Glass` oder `Dishwasher.Door`.

[17] Da es sich bei den Umgebungen um alltägliche, anschauliche Beispiele handelt, ist eine glaubhafte Modellierung enthaltener Objekte und Zustände unproblematisch.

[18] Dieser Aufbau entspricht im Wesentlichen *ARMAR-III* in seiner Küchenumgebung bei Verwendung des Programmiersystems *ArmarX*.

5.5 Korpora

ProNat soll Evaluations-getrieben entwickelt werden (siehe Abschnitte 1.2 und 5.1). Das bedeutet, die Einzelkomponenten (Agenten und Fließbandstufen) werden unabhängig voneinander intrinsisch evaluiert. Dadurch kann der Beitrag jeder Komponente zum Gesamtergebnis bemessen und optimiert werden. Abschließend wird das Gesamtsystem, das heißt das Zusammenspiel der Einzelkomponenten, in einer Ende-zu-Ende-Evaluation bemessen (siehe Kapitel 9). Um eine Evaluation durchführen zu können, wird eine Vergleichsgrundlage benötigt. Soll ein System evaluiert werden, das Daten verarbeitet, bilden Eingabedaten und erwartete Ausgaben die Vergleichsgrundlage. Eine derartige Datenbasis sollte mindestens die folgenden drei Kriterien erfüllen:

1. *Angemessenheit*: Die Evaluationsdaten sollten (soweit wie möglich) erwartbaren Eingaben an das System entsprechen; das bedeutet, die Beispiele sollten realistische Eingabewerte widerspiegeln.
2. *Unabhängigkeit*: Die Evaluationsdaten sollten unabhängig von Wissen über die innere Funktionsweise des Systems angelegt werden; das bedeutet, in die Beispiele sollte kein Wissen über verwendete Techniken einfließen.
3. *Beständigkeit*: Die Evaluationsdaten sollten gemeinsam mit den zugehörigen erwarteten Ausgaben persistiert werden können; das bedeutet, die Datenbasis sollte unveränderlich sein, sodass Evaluationsergebnisse wiederholbar sind.

Im Kontext der Computerlinguistik werden zu Evaluationszwecken Datensammlungen angelegt, die natürlichsprachliche Artefakte enthalten. Entweder werden vorhandene Datenquellen, wie Zeitschriften, Bücher oder Datensammlungen verwandter Ansätze verwendet oder es werden Artefakte gesammelt, beispielsweise durch Studien. Auf diese Weise bilden sich Korpora (siehe Abschnitt 2.3.3)[19]. Neben der Evaluation können Korpora auch verwendet werden, um sprachliche Phänomene zu entdecken oder um Lösungsansätze für Problemstellungen zu entwickeln. Beides ist im Kontext von ProNat relevant, da Agenten teilweise Lösungen zu zuvor noch nicht betrachteten Problemstellungen liefern sollen. Wird ein Teil eines Korpus zur Entwicklung eines Ansatzes verwendet, dürfen die entsprechenden Datenpunkte nicht zur Evaluation verwendet werden, da andernfalls das zweite der zuvor genannten Kriterien (*Unabhängigkeit*) verletzt wird.

Da mit ProNat ein System zur Programmierung mit gesprochener Sprache entsteht, sollten Korpora aus Artefakten bestehen, die natürlichsprachliche Beschreibungen des gewünschten Verhaltens eines Systems enthalten. Bestehende Korpora zur Endnutzer-Programmierung mit natürlicher Sprache (bzw. die enthaltenen Artefakte) sind hinsichtlich der in Abschnitt 5.1 festgelegten Zielstellung für das System ProNat nicht angemessen. Einige Korpora enthalten nur sehr einfach Beschreibungen, wie die in Kapitel 3 vorgestellten Datensätze *ATIS* [Dah+94], *GeoQuery* [ZM96],*WikiSQL* [ZXS17] und das von Lin et al. angelegte Korpus bestehend aus *Bash*-Kommandos und zugehörigen natürlichsprachlichen Beschreibungen [Lin+18]; die in diesen Korpora enthalten (natürlichsprachlichen)

[19] Genauer gesagt werden üblicherweise Auswahlkorpora gebildet, wobei das Auswahlkriterium meist *Angemessenheit* für das zu evaluierende System ist.

Artefakte bestehen aus einzelnen Anweisungen (oder sehr kurzen Anweisungsfolgen), zumeist ohne Kontrollstrukturen. Dasselbe gilt für das Korpus, das im Zuge der von Sales et al. erstellten Vergleichsaufgabe zur Endnutzer-Programmierung mit natürlicher Sprache für *Association of Association for Computational Linguistics* angelegt wurde [SHF17]. Andere Datensammlungen schränken die Ausdrucksform (der natürlichsprachlichen Eingaben) ein, wie der von von Quirk et al. erstellte *IFTTT*-Datensatz, bei dem alle Beschreibungen immer mit einer Bedingung beginnen [QMG15], oder haben einen sehr geringen Umfang, wie die von Tellex et al. und Bastanelli et. al angelegten Korpora [Tel+11; Bas+14]. Wieder andere haben schlicht einen anderen Fokus: Der *HS*-Datensatz von Ling et al. enthält beispielsweise textuelle Beschreibungen zu Implementierungen virtueller Karten (aus Kartenspielen) [Lin+16]. Mit Ausnahme des von Bastanelli et. al angelegten Korpus [Bas+14], das jedoch nicht öffentlich zugänglich ist, bestehen zudem alle genannten Korpora ausschließlich aus textuellen Artefakten und sind daher inhaltlich unpassend.

Daher werden eigens für ProNat neue Korpora angelegt, die laienhafte natürlichsprachliche Beschreibungen zur Programmierung eines Systems enthalten. Dadurch wird sichergestellt, dass die Artefakte angemessen für die Entwicklung und Evaluation des Systems sind (Kriterium 1). Die Artefakte werden mithilfe von Studien gesammelt; den Probanden wird lediglich die Zielstellung, nicht aber die Funktionsweise des (Teil-)Systems bekannt gemacht, um die Beeinflussung der Probanden zu minimieren (Kriterium 2). Die Sammlung der Daten erfolgt in unterschiedlichen Studien. Zum einen werden Laborstudien in kontrollierter Umgebung und mit einheitlicher Konfiguration durchgeführt, in denen Audioaufnahmen der Beschreibungen von Probanden getätigt werden (im Folgenden als stationäre Datensammlung bezeichnet). Zum anderen werden durch Online-Studien textuelle Beschreibungen gesammelt (im Folgenden als Online-Datensammlung bezeichnet). In beiden Fällen werden die gewonnen Daten persistiert (Kriterium 3). In den nachfolgenden Abschnitten wird zunächst das grundsätzliche Vorgehen für beide Datensammlungen beschrieben. Anschließend wird das Vorgehen für die stationäre und zuletzt für die Online-Datensammlung erläutert.

5.5.1 Grundsätzliches Vorgehen

Die Datensammlungen für ProNat werden mithilfe von Studien angelegt. Hierzu werden natürlichsprachliche Anweisungen von Probanden an ein System gesammelt. Die Anweisungsbeschreibungen werden entweder in Textform abgeben oder es werden Audioaufnahmen gesprochener Anweisungsfolgen getätigt. Die Probanden sollen ihre Beschreibungen frei formulieren können und sollen laienhafte Alltagssprache verwenden. Auch hinsichtlich des Inhalts der Beschreibungen sollen die Probanden nicht eingeschränkt werden; das bedeutet, Probanden können beliebige Vorgänge und Zusammenhänge auf beliebigen Abstraktionsebenen beschreiben. Allerdings müssen alle Beschreibungen in Englisch abgeben werden (siehe Abschnitt 5.1).

Um derartige natürlichsprachliche Beschreibungen zu erhalten, wurden alle im Kontext von ProNat durchgeführten Studien wie folgt aufgebaut. Es werden Szenarien verwendet. Jedes Szenario besteht aus einer Situationsbeschreibung und einer Aufgabe, die ein System lösen soll. Zusätzlich veranschaulichen Bilder die Ausgangssituation. Die Probanden werden aufgefordert, sich in die Situation zu versetzen und dem System natürlichsprachliche Anweisungen zu geben. Die Anweisungen sollen

das System befähigen die Aufgabe zu lösen. Das bedeutet, die Probanden können davon ausgehen, dass das System grundlegende Funktionen besitzt. Die Beschreibung der Probanden entsprechen also (im besten Fall) einer Komposition von grundlegenden Systemfunktionen zur Lösung einer komplexen Aufgabe. Den Probanden werden die grundlegenden Systemfunktionen nicht bekannt macht, sondern nur, dass es solche gibt. Durch die Verwendungen von Szenarien wird nur ein grober Rahmen für die Beschreibung der Probanden vorgegeben. Die Szenariobeschreibungen sind zudem kurz gefasst und verwenden eine einfache, nicht technische Sprache mit stark eingeschränktem Vokabular[20]. Eine derartige Gestaltung der Szenarien soll den Probanden möglichst wenig vorgeben und ihre Beschreibungen nicht beeinflussen. Im besten Fall entstehen so frei formulierte und voneinander verschiedene Anweisungsbeschreibungen. Statt Szenariobeschreibungen und unterstützende Bilder zu verwenden, können auch Videos eingesetzt werden. Diesen Ansatz verfolgten beispielsweise Tellex et al. [Tel+11] und Landhäußer et al. [LWT17b]. Die Probanden wurden in diesen Studien aufgefordert, zu beschreiben, was sie sehen. Durch die Verwendung von Videos können viele vergleichbare Artefakte gesammelt werden. Dafür werden die Probanden inhaltlich eingeschränkt. Da die Probanden lediglich die Ereignisse im Video wiedergeben, entstehen keine kreativen Problemlösungen. Außerdem werden so auch die Funktionen des System offengelegt. Daher wird für die Datensammlungen im Kontext von ProNat auf den Einsatz von Videos verzichtet.

Zur Durchführung der Studien werden konkrete Szenarien benötigt. Diese müssen für Laien leicht verständlich und greifbar sein. Das bedeutet, die beschriebene Situation sollte einer alltäglichen Situation entsprechen (oder an eine solche angelehnt sein). Das muss sich zur Endnutzer-Programmierung eignen. Außerdem sollte auch das System den Probanden vertraut oder zumindest anschaulich sein. Das bedeutet auch, dass die Probanden grundlegende Funktionen des Systems erkennen bzw. erahnen können müssen. Unter diesen Anforderungen bietet sich die Verwendung des humanoiden Haushaltsroboter *ARMAR-III* als Zielsystem an (siehe Abschnitt 5.4.3). Bei einem humanoiden Roboter können sich selbst Laien vorstellen, welche Funktionalität dieser (mindestens) bietet. Außerdem existiert mit der für *ARMAR-III* modellierten Küchenumgebung auch ein anschauliches Szenario, in welchem das System agiert. Als Grundszenario für alle Datensammlung wird daher der *ARMAR-III* in seiner Küchenumgebung verwendet. In den konkreten Szenarien wird die Ausgangssituation beschrieben; das bedeutet, es wird erläutert, wie die Umgebung beschaffen ist, welche Umgebungsobjekte relevant sind, wo sich der Roboter befindet und welche Aufgabe er lösen soll. Die Szenariobeschreibungen werden jeweils durch Bilder unterstützt. Das erste Szenario der stationären Datensammlung (siehe Tabelle 5.5), in dem der Roboter eine Popcorn-Packung von einem Tisch holen und dem Nutzer überreichen soll, wird beispielsweise wie folgt beschrieben:

> *In this scene you want the robot to get you popcorn. The popcorn bag stands on the kitchen table among other objects. Figure 1 shows your view on the scene, with the robot to your left and the table on the right side. Figure 2 (a) gives an overview of the objects, which are placed on the table. Figure 2 (b) shows the popcorn bag in detail.*

[20] Diese Art der Aufgabenbeschreibung wurde erstmals von Miller beschrieben [Mil81] und später innerhalb der Studie von Pane et. al umgesetzt [PRM01] (siehe Abschnitt 3.1).

Während der Studien wird den Probanden ein Studienbogen ausgehändigt[21]. Dieser beinhaltet eine kurze Einführung über den Ablauf und die Zielsetzung der Studie. Anschließend wird den Probanden das allgemeine Szenario beschrieben: der Roboter und die Küchenumgebung werden vorgestellt. Die Probanden werden aufgefordert, sich in die Situation hineinzuversetzen. Sie sollen sich dazu vorstellen, dass sie sich mit dem Roboter in der Küche befinden und der Roboter auf Anweisungen wartet. Außerdem werden die Probanden darauf hingewiesen, dass der Roboter grundlegende Funktionen besitzt und die Objekte in seiner Umgebung kennt. Im Studienbogen folgen anschließend eines oder mehrere der zuvor beschriebenen Szenarien. Die Probanden geben für jedes Szenario eine Beschreibung ab. Zuletzt werden anonymisiert persönlich Daten erhoben: Probanden werden aufgefordert ihre Sprach- und Programmierkenntnisse anzugeben.

Die meisten konkreten Szenarien sind so ausgelegt, dass sie die Untersuchung bestimmter sprachlicher Phänomene erlauben; das bedeutet, die jeweiligen Studien lassen sich einem konkreten Agenten, der eben dieses Phänomen analysiert, zuordnen und wurden im Zuge der Evaluation (bzw. vor der Entwicklung) des Agenten durchgeführt. Welche Szenarien welchen Agenten zugeordnet werden können, wird in den nachfolgenden Abschnitten beschrieben. Die Szenarien sind jedoch so gestaltet, dass sich die gesammelten Artefakte zur Evaluation aller Agenten und Fließbandstufen und auch zur Evaluation des Gesamtsystems eignen. Durch dieses Vorgehen, werden Korpora geschaffen, die sich in Teilen insbesondere zur Evaluation einzelner Agenten eignen. Gleichzeitig entsteht auf diese Weise sukzessive eine umfangreiche Datensammlung für die die Evaluation des Gesamtsystems (siehe Kapitel 9).

5.5.2 Stationäre Datensammlung

Der erste Teil der Datensammlung besteht aus Audioaufnahmen von Probanden, die in einer kontrollierten Umgebung aufgenommen wurden. Hierzu wurden sieben Einzelstudien über fünf Jahre durchgeführt. Trotz dieses langen Zeitraums wurden alle Studien unter den gleichen Rahmenbedingungen durchgeführt. Das bedeutet, zur Aufzeichnung wurde immer dasselbe Mikrophon, derselbe Raum und gleichartige Studienbögen, die sich lediglich in den konkreten Szenariobeschreibungen unterscheiden, verwendet. Die Studien wurden von unterschiedlichen Experimentatoren durchgeführt; diese wurden jedoch eingewiesen, um vergleichbare Ergebnisse zu erhalten[22]. Da die Aufzeichnung vor Ort in einem Raum unter Aufsicht eines Experimentators durchgeführt wurden, wird diese in Abgrenzung zur im anschließenden Abschnitt beschriebenen Online-Datensammlung als *stationäre* Datensammlung bezeichnet. Die Probanden sind allesamt freiwillige Studienteilnehmer und wurden mithilfe von Aushängen und über Aufrufe auf Web-Seiten und sozialen Medien rekrutiert. Die erste Studie wurde durchgeführt, um eine initiale Datenbasis zur Evaluation von Agenten und Fließbandstufen zu erhalten; die sechste dient der Erweiterung des Korpus um komplexere (und längere) Beschreibungen. Alle weiteren Studien wurden im Zuge der Evaluation eines der in Kapitel 7

[21] Für die Online-Datensammlung entspricht der Studienbogen aufeinanderfolgenden Web-Formularen.

[22] Unter anderem wurde darauf hingewiesen, Probanden nicht zu beeinflussen; das bedeutet, der Experimentator darf bei der Lösung der Aufgabe keine Hilfestellung geben und nur organisatorische Fragen beantworten.

Tabelle 5.5: Zusammenhang zwischen Szenarien und Agenten-Zugehörigkeit: Die horizontalen Linien unterteilen zudem die einzelnen Studien.

Sz.	Name (Kurzbeschreibung)	Agenten-Zugehörigkeit
1	Popcorn vom Tisch holen	(keine)
2	Becher in Spülmaschine stellen	(keine)
3	Saft aus Kühlschrank holen	(keine)
4	Spülmaschine mit Geschirr befüllen	Bed. Verzweigungen (Abschnitt 7.6.1)
5	Zutaten für Cocktail holen	Bed. Verzweigungen (Abschnitt 7.6.1)
6	Becher mit Wasser füllen	Sprl. Kontext (Abschnitt 7.4)
7	Essen zubereiten	Sprl. Kontext (Abschnitt 7.4)
8	Wäsche machen	Dialog (Abschnitt 7.10)
9	Becher holen, Nachrichten lesen	Nebenläufigkeit (Abschnitt 7.6.3)
10	Spülmaschine ausräumen	Schleifen (Abschnitt 7.6.4)
11	Saft holen, Kühlschrank schließen	(keine)
12	Fenster schließen, Becher füllen	(keine)
13	Becher, Saft, Fenster	Disfluenzen (Abschnitt 7.1)
14	Becher füllen, Fenster schließen	Disfluenzen (Abschnitt 7.1)

beschriebenen Agenten durchgeführt[23]. Tabelle 5.5 zeigt den Zusammenhang zwischen Studien, Szenarien und Agenten-Zugehörigkeit.

5.5.2.1 Durchführung

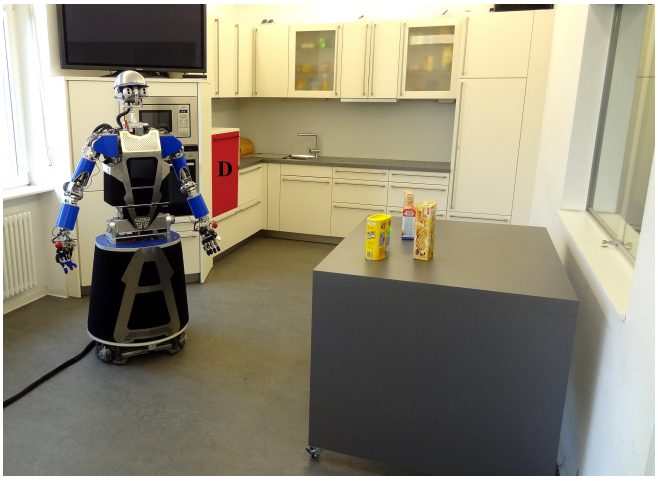
Die Studien für die stationäre Datensammlung wurden alle folgendermaßen durchgeführt. Alle Sitzungen wurden in demselben Raum abgehalten. Der Raum wurde so gewählt, dass äußere Störfaktoren – insbesondere akustische – möglichst ausgeschlossen werden können. Während einer Sitzung befinden sich nur der Proband und der Experimentator in dem Raum. Zu Beginn der Sitzung erhält der Proband den Studienbogen und wird darauf hingewiesen, dass er zu jeder Zeit von der Studienteilnahme zurücktreten kann. Der vollständige Studienbogen, wie er für die stationäre Datensammlung verwendet wurde, befindet sich im Anhang in Abschnitt C.1. Außerdem wird dem Probanden die Möglichkeit eingeräumt, organisatorische Fragen zu stellen (beispielsweise zum Ablauf der Sitzung). Anschließend unterschreibt der Proband die Einwilligung zur Datenerhebung. Daraufhin liest der Proband die Szenariobeschreibungen; die Studienbögen enthalten ein bis drei Szenarien. Eine vollständige Szenariobeschreibung mit den Anweisungen für die Probanden zum zweiten Szenario (*Becher in Spülmaschine stellen*) ist in Abbildung 5.4 dargestellt. Den Probanden ist es erlaubt, zur Vorbereitung der Aufnahme Notizen anzufertigen. Sobald der Proband bereit ist, aufgenommen zu werden, signalisiert er dies dem Experimentator. Dieser startet die Aufnahme.

[23] Das bedeutet, diese Studien können einem Agenten zugeordnet werden. Die Agenten-Zugehörigkeit hat in einigen Fällen Auswirkungen auf die Gestaltung der Szenarien. Beispielsweise wurden das vierte und das fünfte Szenario so entworfen, dass sie eine Fallunterscheidungen enthalten, um die Verwendung von Konditionalsätzen zu provozieren.

Scene II: Dishwasher

In this scene the robot should put an object in the dishwasher. This object is a green cup (Figure 2 (b)), located on the kitchen table (Figure 2 (a)) among other objects. Figure 1 (a) shows your view on the scene, with the robot to your left, the table on the right side and the red marked dishwasher. The opened dishwasher is displayed in Figure 2 (b). Maybe you want ARMAR to close the dishwasher after loading it.

When you are ready to talk to the robot please signal the survey advisor. Your instructions will be recorded via microphone. If you are not satisfied with the recording for any reason please contact the advisor. It is possible to rerecord or stop recording at any time. If you want to take notes please use the last page of this document.



(a) Kitchen (marked dishwasher)

(b) Dishwasher (open)

Figure 1: Scenery: Dishwasher

(c) Objects on the kitchen table

(d) Green cup

Figure 2: Scenery: Dishwasher (kitchen table)

Abbildung 5.4: Beschreibung des zweiten Szenarios (der ersten Studie).

Für die Aufnahmen wurde ein spezielles Mikrophon für Sprachaufnahmen verwendet[24]. Sofern der Proband dies wünscht, kann eine Aufnahme wiederholt werden. Es werden jedoch alle Aufnahmen gespeichert, falls der Proband nicht explizit die Löschung einer Aufnahme verlangt. Während der Aufnahme beschreibt der Proband zusammenhängend seine Anweisungen an das Zielsystem. Dabei erhält er keinerlei Rückmeldung. Nur während der Studie zur Evaluation des Dialog-Agenten treten Probanden mit dem System in einen Dialog (Szenarien vier, sechs und acht). Nach Beendigung der Aufnahmen, wird der Proband aufgefordert, einen Fragebogen auszufüllen. Es werden anonymisiert persönliche Daten erhoben. Unter anderem werden folgende Informationen erfragt:

- Alter
- Geschlecht
- Muttersprache
- derzeitiger Beruf
- Englischkenntnisse (fünfstufige Selbsteinschätzung: *rudimentär*, *grundlegend*, *fortgeschritten*, *versiert*, *verhandlungssicher*)
- Programmiererfahrung (fünfstufige Selbsteinschätzung: *keine*, *rudimentär*, *fortgeschritten*, *versiert*, *fachmännisch*)
- Erfahrung mit Programmierparadigmen (Selbsteinschätzung mit Mehrfachauswahl: *Objekt-orientiert*, *funktional*, *prozedural/imperativ*)

Die Beantwortung der Fragen ist freiwillig; allerdings haben alle Probanden den Fragebogen vollständig ausgefüllt. Nach dem Ausfüllen des Fragebogens ist die Sitzung beendet. Die Aufnahmen aller Probanden wurden archiviert. Zusätzlich wurde zu jeder Aufnahme manuell eine Transkription erstellt; das bedeutet; die Audioaufzeichnung wird vom Experimentator nach der Sitzung angehört und verschriftlicht. Die Transkription wurden gemäß der im *Buckeye Corpus Manual* festgelegten Regeln erstellt [Pit+05]:

- Alles was zu hören ist, muss transkribiert werden. Es sollte genau das verschriftlicht werden, was gesprochen wurde. Dementsprechend sollten auch Grammatikfehler oder im Kontext falsch verwendete Wörter ebenso transkribiert werden.
- Es dürfen keine Satzzeichen verwendet werden.
- Groß- und Kleinschreibung von Wörtern sollte angewandt werden.
- Zahlen müssen als Zahlwörter ausgeschrieben werden.
- Hintergrundgeräusche dürfen ignoriert werden.
- Pausen im Sprechfluss sollten durch Zeilenumbrüche modelliert werden.
- Abkürzungen sollten weitestgehend vermieden werden.

[24] Verwendet wurde das Mikrophonemodell *NT1-A* des Herstellers *Røde Microphones*: `https://rode.com/microphones/nt1-a`, zuletzt besucht am 24.02.2021.

- Kontraktionen dürfen verwendet werden.
- Hesitationswörter wie *ehm*, *uhm* usw. sollten (einheitlich) transkribiert werden.
- Geräusche des Sprechers sollten mithilfe von Etiketten verschriftlicht werden.

Die Transkriptionen werden zusammen mit den Audioaufnahmen und den anonymisierten persönlichen Daten in einer Datenbank aufbewahrt. Auf die Datenbank kann über folgende URL zugegriffen werden: `https://parse.ipd.kit.edu/`.

5.5.2.2 Analyse

Innerhalb von sechs Einzelstudien konnten 201 Aufnahmen von 81 unterschiedlichen Probanden zu dreizehn unterschiedlichen Szenarien gesammelt werden. Hinzu kommen Dialogmittschnitte, die zu Szenario acht, sowie als zusätzliche Artefakte zu den Szenarien vier und sechs aufgezeichnet wurden. Die Dialoge wurden von zehn weiteren Probanden geführt, wodurch 30 zusätzliche Aufzeichnungen entstanden[25]. Die Transkriptionen enthalten in Summe 8365 und im Durchschnitt 41,617 Wörter. Tabelle 5.6 schlüsselt die Werte pro Szenario auf. Die Werte in Klammern entsprechen den Dialogmittschnitten. Diese gehen nicht in die Statistik über die verwendeten Wörter ein. Die Verwendung von knapp 42 Wörtern pro Beschreibung deutet darauf hin, dass die Probanden zumeist längere Anweisungsfolgen beschreiben. Es zeigt sich zudem, dass sich dieser Wert je nach Szenarien deutlich unterscheidet. Während die ersten drei Szenarien einfach zu lösende Aufgaben beschreiben und die Lösung dadurch mit wenigen Wörtern beschrieben werden kann, sind insbesondere die Szenarien sechs, sieben und elf bis vierzehn deutlich komplexer. In diesen Szenarien sind mehrere Teilaufgaben enthalten, wodurch mehr Teilschritte und Zusammenhänge beschrieben werden müssen. Tabelle 5.7 zeigt ausgewählte Beispiele zu unterschiedlichen Szenarien.

Hinsichtlich der personenbezogenen Daten ergeben sich über alle Studien hinweg folgende Verteilungen. Von 91 Teilnehmern waren 23 weiblich und 68 männlich. Die Alterspanne reicht von achtzehn bis 50 Jahre (zum Zeitpunkt der Studie). Das durchschnittliche Alter beträgt jedoch nur 24,7 Jahre, da an den Studien hauptsächlich Studenten teilgenommen haben (77). Als Muttersprache gaben 81 Probanden Deutsch an; die übrigen zehn verteilen sich auf andere indoeuropäische Sprachen. Allerdings gab keiner der Teilnehmer als Muttersprache Englisch an. In Abbildung 5.5 ist die Verteilung für die Englischkenntnisse der Probanden dargestellt. Da es sich um Selbsteinschätzungen handelt, ist die Aussagekraft natürlich beschränkt. Dies könnte auch der Grund dafür sein, dass es eine starke Häufung beim mittleren Kenntnisniveau (*fortgeschritten*) gibt. Lediglich ein Proband schätzte seine Englischkenntnisse als *verhandlungssicher* ein. Abbildung 5.6 zeigt die angegebenen Programmierkenntnisse der Probanden; auch hierbei handelt es sich um Selbsteinschätzungen. Sechzehn Probanden gaben an, über keinerlei Programmierkenntnisse zu verfügen und weitere 22 besitzen nur rudimentäre Fähigkeiten. Andererseits gaben auch 27 Probanden als Kenntnisstand *versiert* und elf *fachmännisch* an. Wie Abbildung 5.7 zeigt, haben die meisten Probanden vor allem Erfahrung mit Objekt-orientieren Programmiersprachen; viele verfügen aber zusätzlich über

[25] Tatsächlich bestehen die Dialogmittschnitte jeweils aus mehreren Aufzeichnungen. In einer Sitzung wurden alle Nutzeraussagen aufgezeichnet und die jeweilige Systemreaktion als Transkription hinterlegt.

Tabelle 5.6: Statistiken zur stationären Datensammlung: Je Szenario ist angegeben, wie viele Probanden Beschreibungen abgegeben haben und wie viele Aufnahmen getätigt wurden. Zudem sind jeweils die Gesamtanzahl und die Durchschnittswerte der enthaltenen Wörter angegeben. Die Anzahl etwaiger Dialogmitschnitte sind in Klammern angegeben.

Szenario	Probanden	Aufnahmen	Wörter (gesamt)	Wörter (durchschnittlich)
1	22	22	318	14,5
2	22	26	650	25,0
3	22	23	443	19,3
4	19 (10)	19 (10)	556	29,3
5	19	19	611	32,2
6	10 (10)	10 (10)	736	73,6
7	10	11	818	74,4
8	(10)	(10)	–	–
9	10	11	314	28,5
10	10	11	319	29,0
11	10	10	420	42,0
12	10	10	517	51,7
13	10	15	1271	84,7
14	10	14	1392	99,4
Alle	81 (91)	201 (231)	8365	41,6

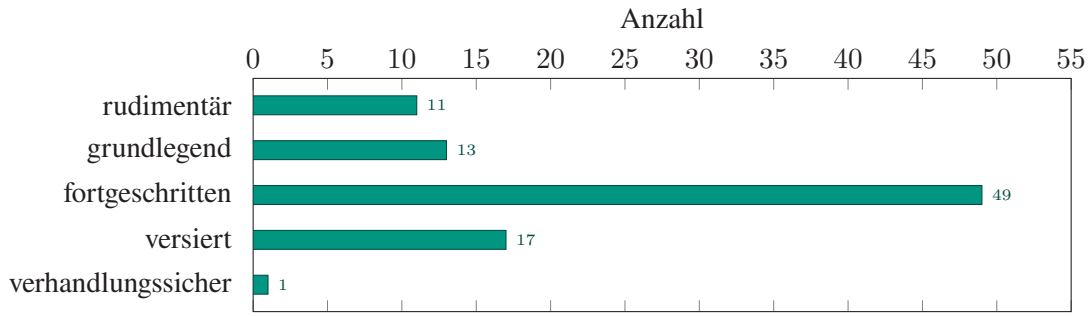


Abbildung 5.5: Verteilung der Englischkenntnisse der Probanden (gemäß Selbsteinschätzung).

Kenntnisse der prozeduralen oder funktionalen Programmierung. 32 Probanden gaben an, mit allen Programmparadigmen dieser Auswahl vertraut zu sein; 18 machten hingegen keine Angabe.

Da die Aufnahmen ausnahmslos von Nicht-Englisch-Muttersprachlern stammen, enthalten sie häufig Grammatikfehler oder eine ungewöhnliche Wortwahl (siehe das Beispiel zu Szenario neun in Tabelle 5.7). Außerdem finden sich vereinzelt Versprecher, Verzögerungslaute und Korrekturen (siehe Beispiele zu den Szenarien zwei, sieben, zwölf und vierzehn). Hinsichtlich linguistischer Merkmale unterscheiden sich die Aufzeichnungen von weiblichen und männlichen Probanden nicht; auch die Länge und Wortwahl ist vergleichbar. Dasselbe gilt hinsichtlich der Programmiererfahrung. Die meisten Probanden beschreiben aus der Sicht eines (naiven) Nutzers; das bedeutet, auch Probanden mit großer Programmiererfahrung verwenden keine technischen Begriffe.

Tabelle 5.7: Exemplarische Transkriptionen der stationären Datensammlung: Die Tabelle zeigt sieben Transkriptionen, die zu Aufnahmen von Probanden zu unterschiedlichen Szenarien angefertigt wurden.

Sz.	Transkription
2	uhm go to the table uhm take the green cup uhm then go to the dishwasher open the dishwasher uhm put the green green cup ehm in the dishwasher and close the dishwasher
4	hey Armar could you please have a look at the table if there are dirty dishes put them in the dishwasher if they are clean put them in the cupboard
7	go to the dishwasher then take one plate out wash this plate and then go to the fridge put the em meal instant meal on the plate and bring it to the mac microwave put it into the microwave then after it is warmed up put the ehm meal on the plate and bring the plate to the table
9	robo please move to the table grab the green cup and move to the sink then please wash the green cup and while you wash the cup please read the news
10	hey Armar please go to the dishwasher and open it while the dishwasher is not empty take another piece of dishware and store it into the cupboard
12	open the window and ehm then close it again afterwards eh go to the dishwasher and take one of the cups ehm before you fill the cup with water put it on the table and before you look at the cup that you filled open the window again
14	uh go to the window and open the window and then yah go to the dishwasher yah open it uh take the red cups which is on the first shelf and put the red cups hm on the table then close the dishwasher then go to the fridge and open the fridge and take the orange juice yah which is between water and the other juice then close the fridge yah put the uh orange juice yah on the table uh then fill uh fill in the red cup and the green cup hm the green cup is between Nesquik and cookies then go to the window and close it

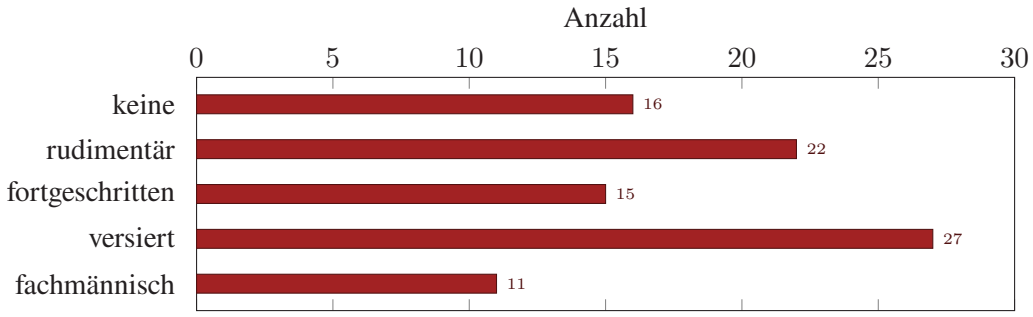


Abbildung 5.6: Verteilung der Programmierkenntnisse der Probanden (gemäß Selbsteinschätzung).

Zusammenfassend zeichnet sich die stationäre Datensammlung dadurch aus, dass die enthaltenen Artefakte gesprochene Audioaufnahmen sind, die in einer kontrollierten Umgebung und unter gleichbleibenden Bedingungen aufgezeichnet wurden. Die Beschreibungen der Probanden entstanden zudem auf Grundlage von 14 unterschiedlichen Szenarien; dadurch sind die Aufnahmen inhaltlich facettenreich. Andererseits wurden die Beschreibungen alle von Nicht-Muttersprachlern abgegeben; die Folge sind grammatikalisch falsche Ausdrücke oder eine ungewöhnliche Wortwahl. Zudem ist der Umfang mit 231 Datenpunkten zwar ausreichend, um aussagekräftige Evaluationen durchzuführen; die Datenbasis ist jedoch unzureichend, um auf ihrer Grundlage statistische Verfahren zur Analyse natürlichsprachlicher Äußerungen zu entwickeln. Um diese beiden Nachteile der stationären Datensammlung auszugleichen, wurde zusätzlich eine Online-Datensammlung durchgeführt.

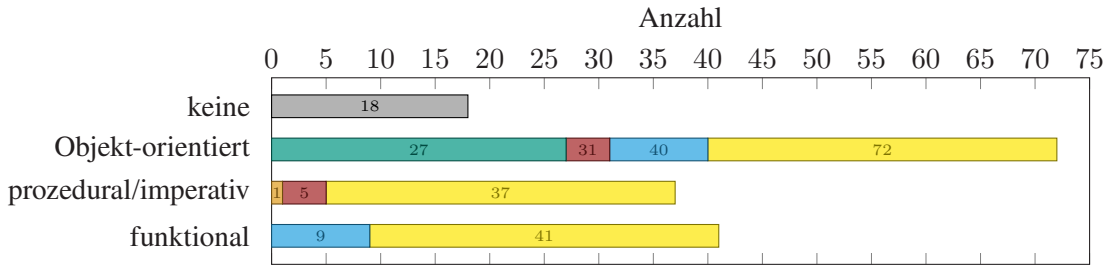


Abbildung 5.7: Verteilung der Programmierparadigmen, mit denen die Probanden Erfahrung haben: Die Balkeneinfärbungen geben die unterschiedlichen Wahlmöglichkeiten der Mehrfachauswahl wieder: grün = Objekt-orientiert, orange = prozedural, rot = Objekt-orientiert und prozedural, blau = Objekt-orientiert und funktional, sowie gelb = alle gewählt.

5.5.3 Online-Datensammlung

Der zweite Teil der Datensammlung besteht aus natürlichsprachlichen Artefakten, die online – wiederum mithilfe von Studien – gesammelt wurden [WST20a; WST20b]. Zur Umsetzung einer Online-Datensammlung wurde zunächst eine Web-Seite erstellt, welche den Studienbogen der ersten Studie der stationären Datensammlung als Web-Formulare umsetzt [Pas15][26]. Diese Art der Datensammlung wurde mit vierzehn Teilnehmern erprobt; es wurden 39 zusätzliche Aufnahmen zu den ersten drei Szenarien gesammelt. Allerdings ist die Qualität dieser Aufnahmen deutlich schlechter als die der stationären Datensammlung. Die Teilnehmer verwendeten zumeist minderwertige Mikrophone; in einigen Fällen stören zudem Umgebungsgeräusche die Aufzeichnung. Daher können diese Aufnahmen zu Evaluationszwecken kaum verwendet werden[27]. Aus diesem Grund wurden in einer weiteren Studien stattdessen textuelle Beschreibungen gesammelt. Außerdem wurde zunächst eine Vorstudie durchgeführt, um vorab zu prüfen, ob der Studienaufbau statthaft ist.

Um die Reichweite der Studie zu vergrößern und vermehrt Englisch-Muttersprachler zu erreichen, wurde zudem die Online-Plattform *Prolific*[28] verwendet. Die Studie umfasst vier weitere Szenarien, zu denen Probanden Beschreibungen erbringen sollen und wurde wiederum im Zuge der Entwicklung bzw. Evaluation zweier Agenten durchgeführt: dem Agenten zur Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache (siehe Abschnitt 7.7) sowie dem Agenten zur Synthese von Methodendefinitionen und Skripten (siehe Abschnitt 7.8).

5.5.3.1 Durchführung

Auch die Online-Datensammlung textueller Beschreibungen wird mithilfe von Studienbögen und Szenariobeschreibungen umgesetzt; allerdings werden Web-Formulare zur Darstellung verwendet. Ein Studienbogen, wie er für die Online-Datensammlung verwendet wurde, befindet sich im Anhang in Abschnitt C.2. Die Web-Formulare werden über die Online-Plattform *Prolific* veröffentlicht, welche auf die Durchführung wissenschaftlicher Studien spezialisiert ist. Studien über

[26] Auf die Studie kann unter folgender URL zugegriffen werden: `https://parse.ipd.kit.edu/asrat/`

[27] In Abschnitt 6.1 wird gezeigt, dass automatische Spracherkenner für diese Aufnahmen eine deutlich höhere Fehlerrate aufweisen, wodurch weiterführende Analysen negativ beeinflusst werden.

[28] Prolific: `https://www.prolific.co/`, zuletzt besucht am 24.02.2021.

eine Plattform durchzuführen, bietet zwei wesentliche Vorteile: Zum einen können Teilnehmer über *Prolific* schneller und in größerer Zahl rekrutiert werden[29] und zum anderen nehmen mehr Englisch-Muttersprachler teil[30]. *Prolific* bietet die Möglichkeit vorab die Teilnehmeranzahl für eine Studie festzulegen und den Personenkreis, der zur Teilnahme berechtigt ist, einzuschränken. Nach Veröffentlichung ist die Studie freigegeben und für angemeldete Nutzer zugreifbar. Probanden werden nach Abschluss der Studie mit geringen Geldbeträgen entlohnt, die vom Experimentator festgelegt werden können. Eine Entlohnung erfolgt nur, wenn der Experimentator die Einreichung akzeptiert; bei Ablehnung wird wieder ein Platz geschaffen und ein anderer Proband kann an der Studie teilnehmen.

Die Online-Studien sind wie folgt aufgebaut. Teilnehmer sind aufgefordert, zu einer Reihe von Szenarien natürlichsprachliche, textuelle Beschreibungen abzugeben. Als Szenariogrundlage dient wieder *ARMAR-III* und die zugehörige Küchenumgebung. Allerdings sind die Szenarien der Online-Datensammlung so gestaltet, dass die Probanden durch ihre Anweisungen dem Roboter neue Funktionen beibringen sollen. Folgende Szenarien wurden erstellt:

1. Jemanden Grüßen
2. Kaffee zubereiten
3. Getränke servieren
4. Einen Tisch für zwei eindecken

Wie auch schon bei der stationären Datensammlung agieren die Teilnehmer nicht direkt mit dem Roboter. Stattdessen erhalten sie eine Beschreibung der Szenerie inklusive Bildern. Für die hier durchgeführte Studie wurden die Teilnehmer zudem darauf hingewiesen, dass eine neue Funktionalität üblicherweise einen Namen hat und der Roboter eine Folge von Zwischenschritten durchführen muss, um eine Funktionalität zu erlernen. Da die Aufgabe eine neue Funktion zu lehren anspruchsvoller als die der anderen Szenarien ist und außerdem während der Durchführung den Probanden kein Experimentator für Rückfragen zur Verfügung steht, wird ein Beispielszenario zur Verfügung gestellt. Abbildung 5.8 zeigt das Beispielszenario *Spülmaschine anschalten* (engl. *start the dishwasher*) der Online-Studie. Neben dem Beispielszenario erhalten die Studienteilnehmer eine kurze Einführung hinsichtlich der Rahmenbedingungen und der Zielsetzung der Studie.

Anders als bei stationären Datensammlung werden keine Audioaufnahmen der natürlichsprachlichen Äußerungen getätigt. Stattdessen sind die Probanden aufgefordert schriftliche Beschreibungen abzugeben, dies aber möglichst spontan zu tun. Zudem werden die Probanden vorab dazu aufgefordert, sich in eine Gesprächssituation mit dem Roboter zu versetzen. Für die Beschreibung steht den Teilnehmern ein Textfeld unterhalb der Bilder in den Formularen zur Verfügung. Nachdem die Probanden alle Szenarien bearbeitet haben, werden sie aufgefordert zwei persönliche Fragen zu beantworten. Zum einen sollen sie ihre Muttersprache benennen und zum anderen, ob sie über

[29] Potenzielle Studienteilnehmer registrieren sich vorab und bekommen aktuell laufende Studien in einer Übersicht angezeigt. Derzeit sind mehr als 70000 Nutzer bei *Prolific* registriert (Stand: 24.02.2021)

[30] *Prolific* ist eine englischsprachige Plattform mit Firmensitz in Großbritannien. Der Großteil der registrierten Nutzer ist in Großbritannien oder den USA wohnhaft.

Abbildung 5.8: Beispielszenario der ersten Online-Studie: *Spülmaschine anschalten*.

Programmiererfahrung verfügen oder nicht. *Prolific* liefert zudem anonymisiert weitere persönliche Daten, wie Geschlecht, Beruf, Alter usw. Diese Angaben hinterlegen Nutzer freiwillig in ihrem Profil. Daher sind die Datensätze gegebenenfalls unvollständig. Um zu prüfen, ob Probanden auf Grundlage dieses Studienaufbaus in der Lage sind, inhaltlich sinnvolle Beschreibungen zu liefern, wurde zunächst eine Vorstudie mit zehn Probanden durchgeführt; verwendet wurden hierzu die ersten drei Szenarien. Da alle Einreichungen der Probanden aus der Vorstudie verwendbar waren, wurde die erste Studie für 860 Probanden mit vier Szenarien freigegeben; die Einreichungen der Vorstudie wurden mit in die Datensammlung aufgenommen. Der vollständige Datensatz kann über folgende URL abgerufen werden: `http://dx.doi.org/10.21227/zecn-6c61`.

5.5.3.2 Analyse

Durch die Online-Studie konnten insgesamt 3470 textuelle Beschreibungen zu vier Szenarien von 870 Probanden gesammelt werden. 302 der Beschreibungen wurden nach einer manuellen Prüfung entfernt[31]; somit verbleiben 3168 Beschreibungen im Datensatz. Tabelle 5.8 schlüsselt die Beiträge

[31] Ausgeschlossen wurden nur Beschreibungen, die offensichtlich nicht sinnhaft sind, unter anderem aus Wikipedia kopierte Passagen oder Abgaben ohne inhaltlichen Bezug zur Aufgabenstellung.

Tabelle 5.8: Statistiken zur Online-Datensammlung: Die Tabelle gibt pro Szenario an, wie viele Probanden jeweils teilgenommen haben und wie viele Beschreibungen dabei entstanden sind. Zudem sind jeweils die Gesamtanzahl und die Durchschnittswerte der Wörter der Beschreibungen angegeben.

Szenario	Probanden	Beschreibungen	Wörter (gesamt)	Wörter (durchschnittlich)
1	870	795	18205	22,9
2	870	794	26005	32,8
3	870	794	33001	41,6
4	860	785	31797	40,5
Alle	870	3168	109008	34,4

Tabelle 5.9: Exemplarische Beschreibungstexte der Online-Datensammlung: Sechs Beschreibungstexte, die von Teilnehmern zu unterschiedlichen Szenarien eingereicht wurden.

Sz.	Eingereichter Beschreibungstext
1	Look directly at the person. Wave your hand. Say 'hello'.
2	You have to place the cup under the dispenser and press the red button to make coffee.
2	Making coffee means you have to press the red button, put a cup underneath the hole and then pouring that comes out into your cup
3	To ring a beverage, open the fridge and select one of te beverages inside, pour it into one of the the kitchen counter and hand the glass over to the person.
4	collect cutlery from cupboard, bring them to the table and place down neatly
4	To set the table for two, Go to the cupboard and take two of each; plates, glasses, knives, and forks. Take them to the kitchen table and set two individual places.

je Szenario auf. Außerdem in der Tabelle aufgeführt sind die verwendeten Wörter je Szenario in Summe und im Durchschnitt. Mit durchschnittlich gut 34 Wörtern sind die Beschreibungen im Mittel etwas kürzer als die der stationären Datensammlung (ca. 42 Wörter je Beschreibung). Zudem zeigen sich deutliche Unterschiede zwischen den Szenarien; während die Beschreibungen zu Szenario eins im Mittel lediglich knapp 23 Wörter enthalten, verwendeten die Probanden zur Lösung von Szenario drei durchschnittlich gut 42 Wörter. Zwar sind die Aufgabenstellung ähnlich gestellt (lehren genau einer neuen Funktion), inhaltlich unterscheiden sie sich jedoch. Während im ersten Szenario das Begrüßen einer Person gelehrt werden soll, muss im dritten beschrieben werden, wie Getränke (im Allgemeinen) serviert werden sollen. Die Beispiele in Tabelle 5.9 zeigen, wie heterogen die Beschreibungen hinsichtlich Länge, Inhalt und Form sind. Außerdem enthalten die Beschreibungen teilweise Grammatik- oder Rechtschreibfehler (z. B. die dritte Beschreibung in der Tabelle).

Die Auswertung der personenbezogenen Daten lieferte folgende Erkenntnisse. Mehr als 60% der Probanden gaben als Muttersprache Englisch an. Laut Profilangaben, sind ein Großteil der Teilnehmer Staatsbürger von Großbritannien; weitere häufig genannte Länder sind die Vereinigten Staaten von Amerika, Polen und Portugal. Knapp 70% der Probanden gaben an, über keinerlei Programmierkenntnisse zu verfügen (bei der stationären Datensammlung waren es lediglich 18%). Dadurch eignet sich der Online-Datensatz insbesondere zur Analyse laienhafter Beschreibungen.

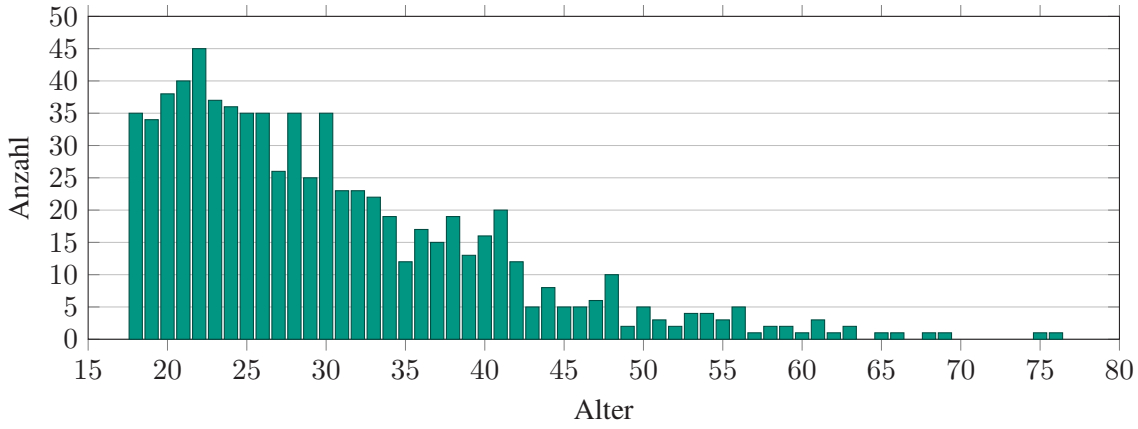


Abbildung 5.9: Altersverteilung der Probanden der Online-Datensammlung.

Hinsichtlich der Geschlechterverteilung der Probanden ist der Datensatz sehr ausgeglichen: 51% Männer und 49% Frauen. Wie Abbildung 5.9 zeigt, haben Probanden unterschiedlichen Alters an der Studie teilgenommen; die Spanne reicht von achtzehn bis 76 Jahren. Im Mittel betrug das Alter 29,5 Jahre; ein Großteil der Teilnehmer war 30 Jahre oder jünger.

Zusammenfassend lässt sich festhalten, dass der hauptsächliche Vorteile der Online-Datensammlung der Umfang ist. Mit geringem Aufwand können große Datenmengen gesammelt werden. Außerdem wurden die meisten Beschreibungen von Englisch-Muttersprachlern erstellt. Dass ein Großteil der Probanden über keine Programmiererfahrung verfügt, ist ein weiterer Vorteil gegenüber der stationären Datensammlung. Andererseits können online nur textuelle Beschreibungen gesammelt werden, Audioaufnahmen sind nicht praktikabel. Außerdem besteht der zweite Datensatz zwar aus mehr Datenpunkten, die Beschreibungen wurden aber zu weniger Szenarien abgegeben. Beide Datensätze haben also Vor- und Nachteile und sollten zielgerichtet verwendet werden. Daher dienen auch unterschiedliche Teile beider Datensätze als Grundlage zur Entwicklung und Evaluation, der in den nachfolgenden Kapiteln beschriebenen konkreten Fließbandstufen und Agenten für ProNat.

6 Vorverarbeitung

„Tu erst das Notwendige, dann das Mögliche,
und plötzlich schaffst du das Unmögliche.“

– Franz von Assisi

Das Vorverarbeitungsfließband markiert den Beginn der Verarbeitung einer gesprochenen Äußerung durch ProNat. Das Ziel ist es, das Audiosignal so vorzuverarbeiten, dass die Agenten anschließend die Semantik der Äußerung analysieren können. Hierzu erfolgt neben der Umwandlung des Audiosignals in eine textuelle Wortsequenz auch bereits eine grundlegende syntaktische Analyse des Gesagten. Zuletzt wird aus den gewonnen Informationen ein initialer ProNat-Graph erzeugt, der als Grundlage für die weitere Verarbeitung der Äußerung durch die Agenten dient (siehe Kapitel 7). Für ProNat wurden folgende Fließbandstufen entwickelt, die nacheinander durchlaufen werden:

1. Mehrfach-Automatische-Spracherkennung (Abschnitt 6.1)
2. Seichte Sprachverarbeitung (Abschnitt 6.2)
3. Erkennung von Eigennamen (Abschnitt 6.3)
4. Erkennung semantischer Rollen (Abschnitt 6.4)
5. Grapherzeugung (Abschnitt 6.5)

Die Fließbandstufen verwenden eine gemeinsame Datenstruktur (siehe Abschnitt 4.2.2), die auf der in der ersten Stufe erzeugten textuellen Repräsentation der Einzelwörter (bzw. der Token) basiert. Eine sortierte Liste aller Einzelwörter repräsentiert die gesamte Äußerung. Zu Token können in der Datenstruktur weitere Informationen hinterlegt werden. So können die Analyseergebnisse der Fließbandstufen zwei bis vier gespeichert werden. Die letzte Fließbandstufe erzeugt aus der internen Datenstruktur des Vorverarbeitungsfließbandes den initialen ProNat-Graphen.

6.1 Mehrfach-Automatische-Spracherkennung

Die erste Stufe im Vorverarbeitungsfließband von ProNat ist die Mehrfach-Automatische-Spracherkennung [Sch16]. Sie ist dafür zuständig das Audiosignal, welches als Eingabe dient, in eine Wortsequenz umzuwandeln. Die Fließbandstufe ist so entworfen, dass unterschiedliche Spracherkenner einfach angebunden werden können; zusätzlich können mehrere Spracherkenner nebenläufig ausgeführt und die Ergebnisse konsolidiert werden.

Die automatische Spracherkennung ist eine der Standard-Aufgaben der Computerlinguistik (siehe Abschnitt 2.3.7). Dadurch, dass in den vergangenen Jahren immer mehr Daten für das Training der Akustik- und Sprachmodelle zur Verfügung stehen (beispielsweise durch die Sammlung von Nutzerdaten von virtuellen Assistenten wie *Siri*), konnte die Qualität der Spracherkenner zuletzt stark gesteigert werden. Zugleich haben sich unterschiedliche Strömungen entwickelt, unter anderem proprietäre Systeme und Web-APIs mit vortrainierten Modellen, quelloffene Werkzeuge, die selbst trainiert werden müssen, oder solche, die es ermöglichen, vortrainierte Modelle anzupassen. Jede dieser Varianten bietet Vor- und Nachteile; je nach Verwendungskontext ist die Verwendung eines anderen Spracherkenners sinnvoll.

Da ProNat in beliebigen Umgebungen eingesetzt werden können soll, muss der verwendete Spracherkenner austauschbar sein. Die Fließbandstufe wurde daher so entworfen, dass beliebige Spracherkennungssysteme registriert und verwendet werden können. Hierzu muss je Spracherkenner lediglich ein Adapter zum eigentlichen Spracherkenner implementiert werden[1]. Die Vorverarbeitungsstufe spezifiziert zudem eine einheitliche Konfigurationsschnittstelle für alle Spracherkenner, etwa um die Anzahl der auszugebenden Alternativhypothesen vorzugeben.

Die Vorverarbeitungsstufe zur Mehrfach-Automatischen-Spracherkennung erlaubt nicht nur die Verwendung unterschiedlicher Spracherkennungssysteme; sie können zudem nebenläufig ausgeführt und die jeweiligen Ergebnisse konsolidiert werden. Dadurch können etwaige Schwächen eines Systems durch ein anderes ausgeglichen werden. Der Ansatz folgt dabei grundsätzlich den Arbeiten von Fiscus und Mangu et al. zur Konsolidierung und Optimierung von Spracherkennungsergebnissen [Fis97; MBS00]. Für die mehrfache automatische Spracherkennung führt die Fließbandstufe folgende Schritte aus:

1. Ausführung der einzelnen Spracherkennungssysteme
2. Konsolidierung der Ergebnisse
3. Erzeugung einer gemeinsamen Ausgabehypothese

Im ersten Schritt wird das Audiosignal an alle registrierten Spracherkennungssysteme übergeben. Diese liefern jeweils mindestens eine Haupthypothese zurück, die aus einer Tokensequenz besteht. Die meisten Systeme geben zudem Konfidenzen je Wort (oder für die gesamte Ausgabe) an. Ebenso erzeugen die meisten Systeme Alternativhypothesen, die wiederum mit Konfidenzen versehen sein können. Gegebenenfalls enthalten die Tokensequenzen Markierungen für gefüllte Pausen[2] und Satzzeichen. Die Satzzeichen – sofern vorhanden – werden entfernt, da sich zeigt, dass eine fehlerhafte Interpunktion spätere Analysen stärker negativ beeinflusst als korrekte Satzzeichen positive Effekte erzeugen[3]. Die Markierungen für gefüllte Pausen werden hingegen beibehalten, da diese die Analyse von Disfluenzen erleichtern (siehe Abschnitt 7.1).

[1] Dadurch wäre es auch möglich, eigens trainierte Spracherkenner zu verwenden. Der Umfang des in Abschnitt 5.5.2 beschriebenen Korpus ist jedoch nicht ausreichend, um sinnvolle Modelle zu erzeugen.

[2] Beispielsweise verwendet das Spracherkennungssystem *IBM Watson* das spezielle Token *%HESITATION*, um Verzögerungslaute wie *uhm* oder *ehm* zu markieren.

[3] Stattdessen werden in der Vorverarbeitungsstufe zur seichten Sprachverarbeitung einzelne Instruktionen erkannt (siehe Abschnitt 6.2).

Die erzeugten Hypothesen aller Systeme werden anschließend konsolidiert. Hierzu wird, wie von Mangu et al. beschrieben [MBS00], ein Konfusionsnetzwerk (engl. *confusion network*) verwendet[4]. Das Konfusionsnetzwerk wird wie folgt aufgebaut. Knoten werden verwendet, um den Beginn, das Ende und die Zwischenräume zwischen den einzelnen Wörtern (bzw. Token) zu markieren. Die eigentlichen Worthypothesen samt Konfidenzen[5] bilden die Kanten zwischen den Knoten. Stimmen Wörter mehrerer Hypothesen an derselben Stelle überein, werden diese zusammengefasst und nur eine Kante erzeugt; für die zugehörige Konfidenz wird das Maximum der Einzelworthypothesen verwendet. Unterscheiden sich die Hypothesen hinsichtlich der Wortanzahl, werden an den entsprechenden Stellen im Netzwerk Leer-Kanten eingefügt[6]. Durch die Verwendungen von Konfusionsnetzwerken können Hypothesen verschiedener System konsolidiert werden; gleichzeitig wird die Reihenfolge der Token innerhalb der Sequenz gewahrt, auch Konfidenzen und Leerstellen können so dargestellt werden. Jeder Pfad durch das Netzwerk stellt eine mögliche Ausgabehypothese (als Tokensequenz) dar. Um die Weiterverarbeitung der konsolidierten Hypothesen zu vereinfachen, können die Netzwerke optional vereinfacht werden. Zum einen können die Lemmata der Einzelwörter bestimmt[7] und bei Übereinstimmung zusammengefasst werden (beispielsweise können *going* und *gone* auf die Grundform *go* zurückgeführt werden). Werden Wörter auf diese Weise zusammengefasst, wird entweder die durchschnittliche oder höchste Konfidenz als neue Konfidenz verwendet[8]. Zum anderen können Worthypothesen verschmolzen werden, falls die Einzelwörter in *WordNet* im selben *Synset* auftreten (bspw. sind *dish washer* und *dishwasher* beide Bestandteil des *Synsets* [dishwasher, dish washer, dishwashing machine][9].) Beide Optimierungen erzeugen jedoch gegebenenfalls einen Informationsverlust, weshalb sie in der Standardkonfiguration nicht verwendet werden[10].

Aus dem Konfusionsnetzwerk können anschließend die Haupthypothese und gegebenenfalls Alternativhypothesen erzeugt werden. Zur Erzeugung der Haupthypothese wird der Pfad durch das Netzwerk gewählt, der die höchste Gesamtkonfidenz ergibt. Dadurch wird eine Ausgabehypothese erzeugt, die aus Bestandteilen der Hypothesen der unterschiedlichen Spracherkennungssysteme zusammengesetzt ist. Zusätzlich werden anstelle aller Alternativhypothesen die Alternativen je Wort, die sich jeweils aus den alternativen Kanten ergeben, abgelegt.

[4] Ein Konfusionsnetzwerk ist ein gerichteter azyklischer Graph mit den zusätzlichen Eigenschaften, dass es jeweils genau einen Quell- und einen Senkenknoten gibt und alle Pfade vom Quell- zum Senkenknoten alle anderen Knoten durchlaufen [BBR01]. Konfusionsnetzwerke werden zur kombinierten Darstellung mehrere Alternativen verwendet, vor allem zur Darstellung von Wortalternativen bei der Spracherkennung und maschinellen Übersetzung. Wörter werden hierzu als Knoten repräsentiert und durch die Kanten die alternativen Wortfolgen modelliert. Konfidenzen einzelner Alternativen werden als Kantengewichte dargestellt.

[5] Die Konfidenzen werden auf den Wertebereich [0; 1] transformiert.

[6] Hypothesen können sich hinsichtlich ihrer Länge unterscheiden, falls Spracherkennungssysteme für Teile einer Eingabe keine Hypothese erzeugen oder indem zusammengesetzte Wörter gebildet bzw. geteilt werden, wie beispielsweise *dishwasher* und *dish washer*.

[7] Zur Bestimmung der Lemmata wird der Wortartmarkierer (engl. *PoS tagger*) der Stanford University verwendet: `https://nlp.stanford.edu/software/tagger.shtml`, zuletzt besucht am 24.02.2021.

[8] Ob der Durchschnitt oder das Maximum verwendet wird, kann konfiguriert werden, standardmäßig wird jedoch das Maximum verwendet.

[9] zugehöriges *WordNet-Synset*: `http://wordnetweb.princeton.edu/perl/webwn?s=dishwasher&sub=Search+WordNet&o2=&o0=1&o8=1&o1=1&o7=&o5=&o9=&o6=&o3=&o4=&h=`, zuletzt besucht am 24.02.2021.

[10] Darüber hinaus erzeugen diese Optimierungen zusätzlichen Rechenaufwand, der sich nur bei großen Netzwerken durch einen verringerten Aufwand zur Erzeugung der Ausgabehypothese amortisiert.

Beispiel:
Erzeugung von Konfusionsnetzwerken zur Konsolidierung von Spracherkennungshypothesen

In diesem Beispiel soll die Erzeugung eines Konfusionsnetzwerks aus zwei Hypothesen demonstriert werden. Das Beispiel verwendet zwei synthetische Hypothesen, die jeweils durch ein hypothetisches Spracherkennungssystem erzeugt wurden.

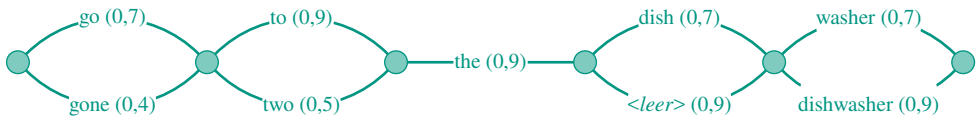
Natürlichsprachliche Sequenz

go to the dishwasher

Hypothesen der Spracherkennungssysteme

Spracherkennungssystem A: $go_{0,7}$ $to_{0,9}$ $the_{0,9}$ $dish_{0,7}$ $washer_{0,7}$
Spracherkennungssystem B: $gone_{0,4}$ $two_{0,5}$ $the_{0,8}$ $dishwasher_{0,9}$

Konfusionsnetzwerk



Erläuterung

Das Konfusionsnetzwerk wird erzeugt, indem für jede Worthypothese eine Kante samt Konfidenz erzeugt wird. Im Beispiel unterscheiden sich die Hypothesen in den ersten beiden Wörtern, weshalb je zwei Kanten erstellt werden. Beim dritten Wort (*the*) stimmen sie überein. Folglich wird nur eine Kante erzeugt und die höhere der beiden Konfidenzen verwendet. Anschließend beinhaltet eine der Hypothesen die Wörter *dish* und *washer*, während die andere beide Wörter zusammenfasst. Dementsprechend wird als Repräsentation der zweiten eine leere Kante parallel zur *dish*-Kante erzeugt und die Konfidenz des nachfolgenden Wortes übernommen. Für dieses Beispiel ergeben sich sechzehn mögliche Pfade durch das Netzwerk und damit ebenso viel potenzielle Ausgabehypothesen. Werden die beiden zuvor beschriebenen Optimierungen auf das Beispielnetzwerk angewandt, ergeben sich die folgende Änderungen. Da *go* und *gone* dasselbe Lemma (*go*) besitzen, würden diese Kanten zusammengelegt. Ebenso könnten die Kanten *dish* und *washer* verschmolzen werden, da diese Bestandteil desselben *WordNet-Synsets* wie *dishwasher* sind. In diesem Fall würde das Ergebnis der Verschmelzung zusätzlich mit den alternativen Kanten im Netzwerk zusammen fallen. Das resultierende Netzwerk würde dann nur noch zwei mögliche Pfade zulassen: die Alternativen *to* und *two*.

Tabelle 6.1: Evaluationsergebnisse verschiedener Systeme zur automatischen Spracherkennung: Als Datengrundlage wurde ein Korpus verwendet, das aus den 151 Sprachaufnahmen der stationären Datensammlung zu den Szenarien eins bis fünf besteht. Die Ergebnisse in Klammern entsprechen den Werten, die erzielt werden, wenn die 39 Aufnahmen zu den Szenarien eins bis drei, die mit unbekannten Mikrofonen und in unbekannten Umgebungen aufgenommen wurden, ausgeschlossen werden.

Konfiguration	WER	TERp	BLEU-4
Google	0,296 (0,289)	0,295 (0,289)	0,602 (0,611)
Google*	0,189 (0,176)	0,189 (0,176)	0,693 (0,708)
IBM Watson	0,273 (0,257)	0,272 (0,255)	0,606 (0,614)
G+W-Konfusionsnetzwerk	0,208 (0,184)	0,207 (0,184)	0,670 (0,695)

Für ProNat wurden Adapter zu den Web-APIs von *Google Speech API*[11] und *IBM Watson*[12] erstellt. Die Qualität der beiden Spracherkennungssysteme wurde anhand eines Teils des ProNat-Korpus bemessen. Zusätzlich wurde evaluiert, ob die Ergebnisse durch die Konsolidierung der Hypothesen beider Systeme positiv beeinflusst werden. Als Datengrundlage dienen die 151 Sprachaufnahmen der stationären Datensammlung zu den Szenarien eins bis fünf sowie die zugehörigen händischen Transkriptionen als Vergleichsgrundlage (siehe Abschnitt 5.5.2)[13]. Bewertet werden ausschließlich die Haupthypothesen, die von Spracherkennungssystemen (bzw. dem Konfusionsnetzwerk) erzeugt werden. Als Metriken dienen die Wortfehlerrate (kurz *WER*, siehe Abschnitt 2.4.6), die erweiterte Übersetzungsänderungsrate (kurz *TERp*, siehe Abschnitt 2.4.8), sowie die *BLEU-4*-Metrik (siehe Abschnitt 2.4.7). Die Ergebnisse sind in Tabelle 6.1 aufgeführt; in Klammern sind zudem die jeweiligen Werte angegeben, wenn die 39 Aufnahmen zu den Szenarien eins bis drei, die mit unbekannten Mikrofonen und in unbekannten Umgebungen aufgenommen wurden, ausgeschlossen werden (siehe Abschnitt 5.5.3). Die Web-API von *Google* liefert kein Ergebnis, falls die Hypothese zu unsicher ist; folglich werden diese Aufnahmen als fehlerhaft betrachtet. Schließt man diese Aufnahmen aus der Bewertung aus, ergeben sich die in der dritten Zeile der Tabelle mit einem Stern markierten Ergebnisse.

Die Wortfehlerraten der beiden Spracherkenner liegen bei 29,6% (*Google*) bzw. 27,3% (*IBM Watson*). Das bedeutet, das etwa jedes dritte bis vierte Wort falsch erkannt wird. Da die Werte für die Wortfehlerrate und die erweiterte Übersetzungsänderungsrate nahezu identisch sind, handelt es sich bei dem Großteil der Fehler zudem um solche, die die Semantik verändern[14]. Folglich könnte sich im Schnitt innerhalb jeder Anweisung mindestens ein falsches Wort befinden. Die *BLEU*-Metrik zeigt jedoch, dass beide Spracherkennungssysteme mehr als 61% korrekte Quadrigramme (4-Gramme) erzeugen. Die Ergebnisse können also insgesamt so gedeutet werden, dass Wortsequenzen häufig entweder komplett richtig oder mit vielen Wortfehlern erkannt werden. Auch die deutlich besseren Ergebnisse des Spracherkennungssystems falls unsichere Hypothesen aus der Bewertung

[11] Google Speech API: `https://cloud.google.com/speech-to-text`, zuletzt besucht am 24.02.2021.

[12] IBM Watson Speech-to-Text: `https://www.ibm.com/cloud/watson-speech-to-text`, zuletzt besucht am 24.02.2021.

[13] Unter den Aufnahmen zu den Szenarien eins bis drei sind auch solche, die mit unbekanntem Mikrofon aufgenommen wurden und daher eine schlechtere Gesamtqualität aufweisen.

[14] Wortfehler, die die Semantik nicht ändern, wie die Verwendung einer anderen Wortform, werden von *TERp* nicht als Fehler gewertet. Würden viele dieser Fehler auftreten, müsste der *TERp*- niedriger als der *WER*-Wert sein.

ausgeschlossen werden (*Google**), spricht für diese Deutung. Dieses Resultat ist insofern positiv, als dass nachfolgenden Analysen größtenteils auf vollständig korrekten Anweisungen durchgeführt werden können. Die Konsolidierung der Ergebnisse mithilfe von Konfusionsnetzwerken verbessert die Ergebnisse. Die Wortfehlerrate sinkt auf 20,8%, eine relative Verbesserung gegenüber der Spracherkennungssystem von *Google* um 29,7% und gegenüber *IBM Watson* um 23,8%. Auch der *BLEU-4*-Wert steigt auf 0,670.

Trotz dieser deutlichen Verbesserungen sind die Ergebnisse nicht zufriedenstellend. Da die automatische Spracherkennung den Beginn der Verarbeitung markiert und somit alle weiteren Analysen davon abhängen, müssen Fehler möglichst vermieden werden. Bei der Bewertung der Ergebnisse muss jedoch beachtet werden, dass sämtliche Aufnahmen von Nicht-Muttersprachlern stammen (siehe Abschnitt 5.5.2). Die Aussprache von Nicht-Muttersprachler unterscheidet sich häufig von der (korrekten) Aussprache von Muttersprachlern, wodurch das Akustik-Modell des Spracherkennungssystem Wörter schlechter erkennt. Auch das Sprach-Modell ist auf Muttersprachler angepasst und kann durch ungewöhnliche (gegebenenfalls grammatikalische falsche) Wortfolgen, wie sie Nicht-Muttersprachler mitunter verwenden, wahrscheinliche Wortfolgen weniger gut bestimmen. Die Ergebnisse zeigen auch, dass die Spracherkennung von guten Aufnahmebedingungen (kontrollierte Umgebung und professionelles Mikrofon) profitiert. Dies wird deutlich, wenn man die Werte betrachtet bei denen Aufnahmen aus unbekannten Umgebungen, die mit unbekannten Mikrofonen aufgezeichnet wurden, ausgeschlossen werden (Ergebnisse in Klammern)[15]. Auch wenn perfekte Bedingung in realistischen Umgebungen nicht gewährleistet werden können, zeigt sich, dass eine Optimierung der äußeren Gegebenheiten gewinnbringend sein kann. Da Wortfehler weiterführende Analyseergebnisse deutlich beeinflussen können, der Einfluss sprachlicher Unterschiede zwischen Muttersprachlern und Nicht-Muttersprachlern nicht quantifiziert werden kann und ProNat die Einbindung neuer (gegebenenfalls besserer) Spracherkennungssysteme jederzeit ermöglicht, werden im Folgenden – wenn nicht anders angegeben – die händischen Transkriptionen des ProNat-Korpus verwendet[16]. Nichtsdestotrotz sind sowohl die Fließbandstufen als auch die Agenten so entworfen, dass ihre Analysen robust gegenüber (stückweise) fehlerhaften Transkriptionen sind.

6.2 Seichte Sprachverarbeitung

Innerhalb der zweiten Fließbandstufe von ProNat erfolgt eine leichtgewichtige lexikalische und syntaktische Analyse der zuvor generierten Tokensequenz, die sogenannte seichte Sprachverarbeitung [Koc15]. Es werden ausschließlich Analysen durchgeführt, die für Transkriptionen gesprochener Sprache geeignet sind, also solche, die unter anderem robust gegenüber Wort- und Grammatikfehlern

[15] Zusätzlich zu den Aufnahmebedingungen unterscheiden sich auch die Sprecher. Die Ergebnisse sind damit nur bedingt vergleichbar (der Einfluss der Sprecher auf die Ausnahme wurde als Einflussfaktor nicht kontrolliert). Damit ist die Folgerung nur als Tendenz zu verstehen, die keinesfalls allgemeingültig ist.

[16] Wie in Abschnitt 5.5.2.1 beschrieben, stellen die Transkriptionen eine unbearbeitete Verschriftlichung der gesprochenen Äußerungen dar. Dementsprechend enthalten diese potenziell Disfluenzen, wie Hesitationswörter, Korrekturen etc., Grammatikfehler oder unübliche Formulierungen; außerdem fehlt die Interpunktion gänzlich. Folglich ergeben sich bei der Verarbeitung händischer Transkriptionen sämtliche Herausforderungen gesprochener Sprache (siehe Abschnitt 2.3.12) mit Ausnahme etwaiger Wortfehlerkennung durch ein System zur automatischen Spracherkennung.

sind. Hierzu zählen die Bestimmung von Wortarten (siehe Abschnitt 2.3.4.3), von Lemmata (siehe Abschnitt 2.3.4.4), von syntaktischen Abschnitten (engl. *chunks*) und von Instruktionen[17].

Während die ersten beiden Standardstufen üblicher Sprachverarbeitungsfließbänder darstellen (siehe Abschnitt 2.3.4), handelt es sich bei der Bestimmung der syntaktischen Abschnitte (in der Folge als *Chunks* bezeichnet) und der Instruktionen um angepasste syntaktische Analysen. Chunks werden anstelle von Syntaxbäumen oder Abhängigkeitsgraphen erzeugt. Diese beiden Darstellungsformen der Syntax werden durch Verfahren erzeugt, die auf Zeitungstexten (oder ähnlichem) trainiert wurden und daher sensibel gegenüber Wortfehlern und ungrammatikalischen Äußerungen sind. Außerdem werden beide je Satz aufgebaut. Transkriptionen gesprochene Sprache verfügen jedoch über keinerlei Interpunktion. Damit sind beide Darstellungsformen für gesprochene Sprache ungeeignet. Chunks können hingegen aus partiellen Zerteilungen (engl. *partial parsing*) erzeugt werden. Diese benötigen zur Generierung lediglich stückweise Wortart-Sequenzen. Die Chunks unterteilen eine Äußerung in syntaktische Abschnitte, das heißt Nominal-, Verbal-, Adverbialphrasen und so weiter [Abn92]. Damit entsprechen die Chunks in etwa den untersten (Nicht-Blatt-)Knoten eines Syntaxbaums. Chunks geben somit zwar weniger syntaktische Informationen als andere Darstellungsformen wieder, sind dafür jedoch auch für Transkriptionen gesprochener Sprache geeignet. Die Bestimmung von Instruktionen dient als Ersatz für die fehlende Satztrennung. Das verwendete Verfahren ist eine Eigenentwicklung, die anhand von Heuristiken die Tokensequenz in Abschnitte untergliedert, die je genau eine Instruktion (bzw. Aktion, Ereignis oder Aussage) enthalten. Die Heuristiken verwenden Chunk- und Wortartmuster. Näheres hierzu folgt am Ende des Abschnitts.

Der erste Schritt während der seichten Sprachverarbeitung ist die Analyse der Wortarten der Tokensequenz. Hierzu wird ein Standardwerkzeug verwendet. Um das am besten geeignete zu bestimmen, wurden zufällig vierzehn Transkriptionen zu den Szenarien eins bis drei des stationären Pro$\mathcal{N}at$-Korpus gezogen[18] (siehe Abschnitt 5.5.2) und für diese Muster-Wortartetiketten erstellt. Anhand dieses Goldstandards konnte die Qualität verschiedener Werkzeuge für Pro$\mathcal{N}at$-typische Äußerungen bestimmt werden. Vermessen wurden die folgenden Werkzeuge:

- Das Wortart-Erkennungs-Modul der Bibliothek *Apache OpenNLP*[19] (kurz *OpenNLP*)
- Das *Illinois*-Wortart-Erkennungssystem der *Cognitive Computation Group* der *University of Pennsylvania* [RZ98] (kurz *Illinois*)
- Das Wortart-Erkennungssystem der *Natural Language Processing Group* der *Stanford University* [TM00; Tou+03] (kurz *Stanford*)
- Das Wortart-Erkennungs-Modul aus der Software-Sammlung NLP4J[20] (kurz *NLP4J*)
- Das *MBSP*-Wortart-Erkennungssystem der *University of Antwerp* [DVD10] (kurz *MBSP*)
- Die Wortart-Informationen des Sprachverarbeitungssystems *SENNA* [Col+11] (kurz *SENNA*)

[17] Diese vier Analyseschritte werden in einer Fließbandstufe (statt in vier) durchgeführt, da sie stark voneinander abhängen und gemeinsam die Grundlage für alle weiteren syntaktischen und semantischen Analysen bilden.
[18] Die vierzehn Transkriptionen enthalten in Summe 468 Wörter, im Durchschnitt also 33,43.
[19] Apache OpenNLP: `https://opennlp.apache.org/`, zuletzt besucht am 24.02.2021.
[20] NLP4J: `https://emorynlp.github.io/nlp4j/`, zuletzt besucht am 24.02.2021.

Tabelle 6.2: Durch verschiedene Werkzeuge zur Wortarterkennung erzielte Genauigkeiten: Als Datengrundlage für den Vergleich wurden vierzehn Transkriptionen aus dem ProNat-Korpus verwendet. Die mit einem Stern gekennzeichnete Spalte listet die Ergebnisse, die erzielt werden, wenn die unterschiedlichen morphologischen Formen von Verben zusammengefasst werden.

System	Genauigkeit	Genauigkeit*
OpenNLP	0,902	0,934
Illinois	0,848	0,910
Stanford	0,904	0,938
NLP4J	0,908	0,923
MBSP	0,844	0,908
SENNA	0,895	0,949

Die Ergebnisse sind in Tabelle Tabelle 6.2 aufgeführt. Bestimmt wurde jeweils die Genauigkeit. Da einige Wortart-Erkennungssysteme Probleme bei der Unterscheidung der genauen Verbform aufwiesen[21], wurden zusätzlich die Genauigkeiten ohne Beachtung der genauen Verbform bestimmt (Spalte Genauigkeit*). Abgesehen von *Illinois* und *MBSP* erzielen alle System eine Genauigkeit von ca 90%. Wird die genaue Verbform nicht betrachtet zeigt *SENNA* das mit Abstand beste Ergebnis (0,949), gefolgt von *Stanford* (0,938) und *OpenNLP* (0,934). Daher wird für ProNat *SENNA* zur Erkennung der Wortarten verwendet. Zusätzlich ist es möglich, auch *Stanford* auszuführen und die Ergebnisse zusammenzuführen (Die Standardkonfiguration sieht die Verwendung beider Werkzeuge vor). Die Kombination betrachtet ausschließlich Verben. Beide Werkzeuge weisen auch abseits der genauen morphologischen Form Probleme bei der Erkennung von Verben auf; häufig werden Verben fälschlich als Adjektive oder Nomen markiert, allerdings nicht übereinstimmend durch beide Werkzeuge[22]. Daher werden die Ergebnisse von *SENNA* und *Stanford* zusammengeführt, indem eine Verbform etikettiert wird, falls mindestens einer der beiden ein Verb für das entsprechende Token erkannt hat. Anschließend werden die Grundformen der Wörter, das heißt, die Lemmata, bestimmt. Hierzu wird das Werkzeug *Stanford* verwendet[23]. Dieses erzeugt die Lemmata als Nebenprodukt der Wortart-Etiketten.

Im dritten Schritt erzeugt die Fließbandstufe Chunks. Hierzu wurden erneut Standardwerkzeuge vermessen. Zum einen das bereits zur Erkennung der Wortarten verwendete *OpenNLP* und zum anderen das Werkzeug *BIOS*[24]. Beide erwarten als Eingabe eine Sequenz von Wortart-Etiketten und emittieren je Wort eine Chunk-Etikette. Da die Chunks syntaktische Abschnitte darstellen, wird das *IOB*-Format zur Kennzeichnung zusammenhängender Sequenzen verwendet (siehe Abschnitt 2.3.6).

[21] Das bedeutet, es konnte zwar bestimmt werden, dass es sich um ein Verb handelt, die genaue morphologische Form konnte jedoch nicht bestimmt werden. Beispielsweise wurden häufig die Grundform (Etikette *VB*) und die Präsensform der ersten und zweiten Person (Etikette *VBP*) vertauscht. Beide sind im Englischen meist identisch, weshalb es sich gegebenenfalls zwar um eine fehlerhafte Erkennung handelt, diese jedoch kaum ins Gewicht fällt.

[22] Die Probleme bei der Erkennung von Verben können möglicherweise auf die fehlende Interpunktion und die häufige Verwendung des Imperativs in den Transkriptionen zurückgeführt werden.

[23] Da es sich bei der Bestimmung der Lemmata um eine vergleichsweise einfache Aufgabe handelt, ist ein Vergleich der verfügbaren Werkzeuge nicht nötig.

[24] BIOS: `http://www.talp.upc.edu/content/tools/bios-suite-syntactico-semantic-analyzers-english-includes-smart-tokenization-pos`, zuletzt besucht am 24.02.2021.

Tabelle 6.3: Durch die Werkzeuge *OpenNLP* und *BIOS* bei der Erzeugung von Chunk-Etiketten erzielte Genauigkeiten: Als Datengrundlage für den Vergleich wurden vierzehn Transkriptionen aus der stationären Datensammlung verwendet.

System	Genauigkeit
OpenNLP	0,931
BIOS	0,987

Für die Phrase *the green cup* würden somit folgende Etiketten emittiert werden: *B-NP*, *I-NP* und *I-NP*. Eine vollständige Liste des Etikettensatzes für Chunks befindet sich im Anhang in Abschnitt A.2. Zur Vermessung der Werkzeuge wurden dieselben vierzehn Texte wie zuvor verwendet und wiederum Musterlösungen für die Chunks erstellt; als Metrik dient die Genauigkeit. Die Ergebnisse der Vermessung sind in Tabelle 6.3 aufgeführt. Die Genauigkeit von *BIOS* übertrifft die von *OpenNLP* um 5,6 Prozentpunkte. Daher wird für ProNat *BIOS* zur Erzeugung von Chunks verwendet.

Im letzten Schritt werden einzelne Instruktionen innerhalb von Äußerungen erkannt. Dabei sollen zusammenhänge Aktionen gemeinsam mit abhängigen Entitäten und Modifikatoren Teil einer Instruktion sein, wie in folgendem Beispiel: *hey Robo* | *take the cup* | *and hand it over to me*. Als Markierung dienen fortlaufende Nummern, die je Wort erzeugt werden. Da für diese Aufgabe bisher keine bekannten Ansätze existieren, wurde eine heuristikbasierte Lösung entwickelt. Diese verwendet Muster bestehend aus Chunks, Wortarten und Signalwörtern[25], um die Äußerungen in Abschnitte zu teilen. Der Ansatz basiert auf der Annahme, dass Äußerungen im Kontext von ProNat im Wesentlichen aus einer Aneinanderreihung von Anweisungen an ein System bestehen. Derartige Formulierungen erfordern üblicherweise die Verwendung des Imperativs. Da Imperativsätze mit einer Verbalphrase beginnen, sieht die Heuristik grundlegend vor, die Äußerungen zu Beginn einer jeden Verbalphrase (Chunk-Etikett *B-VP*) zu teilen. Darüber hinaus gibt es eine Vielzahl von Ausnahmeregeln, unter anderem für erweiterte Infinitive, Neben- oder Konditionalsätze und Modifikatoren. Für die Instruktionserkennung können zwei Modi konfiguriert werden. Der erste, *konservative* Modus, erzeugt kurze Instruktionen und verwendet wenige Ausnahmeregelungen. Mithilfe des zweiten, *erweiterten* Modus, können dank der Verwendung aller Ausnahmeregelungen längere Instruktionen erkannt werden[26]; dies geht gegebenenfalls zulasten der Präzision.

Die Qualität der Analysen der Fließbandstufe zur seichten Sprachverarbeitung wurde anhand von zwanzig zufällig gezogenen Transkriptionen zu den Szenarien eins bis drei des stationären Korpus bestimmt (siehe Abschnitt 5.5.2)[27]. Je Transkription wurden Musterlösungen für die Etiketten erstellt, wobei die Lemmata ausgelassen und dementsprechend auch nicht evaluiert werden. Bemessen wurde jeweils die Genauigkeit, wobei zu beachten ist, dass (anders als bei der Auswahl der Werkzeuge) die aufeinander folgenden Schritte jeweils die emittierten Etiketten verwenden. Das bedeutet, die Chunks werden anhand der zuvor bestimmten Wortart-Etiketten generiert und die Erzeugung der Instruktionsnummern verwendet sowohl die generierten Chunk- als auch Wortart-Etiketten; potenziell können sich somit Fehler fortpflanzen. Die Fließbandstufe wurde in zwei Varianten

[25] Signalwörter können gruppiert nach ihrer Funktion konfiguriert werden.

[26] Die Standardkonfiguration der Fließbandstufe sieht die Verwendung des *erweiterten* Modus vor.

[27] Dabei wurde sichergestellt, dass keine der vierzehn Transkriptionen, die als Vergleichsgrundlage bei der Wahl der einzelnen Werkzeuge gedient haben, wiederverwendet werden.

Tabelle 6.4: Durch die Fließbandstufe zur seichten Sprachverarbeitung bei der Erkennung von Wortarten, Chunks und Instruktionen erzielte Genauigkeiten.

Konfiguration	Wortarten	Chunks	Instruktionen
SENNA – BIOS – Instruktionen	0,937	0,911	0,952
SENNA und Stanford – BIOS – Instruktionen	0,957	0,930	0,990

evaluiert: zur Wortarterkennung werden entweder *SENNA* und *Stanford* verwendet oder nur *SENNA*. Die Ergebnisse für beide sind in Tabelle 6.4 aufgeführt. Insgesamt erzielt die Fließbandstufe durchgängig Genauigkeitswerte über 90%; dabei kommt es zu keinen starken Fehlerfortpflanzungen. Stattdessen werden die besten Genauigkeiten für die Bestimmung der Instruktionen erreicht. Die zweite Konfiguration, die zur Wortarterkennung die kombinierten Ergebnisse von *SENNA* und *Stanford* verwendet, übertrifft die erste deutlich. Die Genauigkeit der Wortarterkennung steigt um zwei Prozentpunkte. Dadurch verbessert sich auch die Bestimmung der Chunks um ebenfalls knapp zwei Prozentpunkte. Die Instruktionserkennung profitiert noch stärker, die Genauigkeit steigert sich auf 99%. Die Ergebnisse zeigen, dass mit der Fließbandstufe zur seichten Sprachverarbeitung eine gute Grundlage für die weiteren Analysen der Sprache durch ProNat gelegt werden kann.

Beispiel:
Durch die seichte Sprachverarbeitung erzeugte Informationen

In diesem Beispiel soll demonstriert werden, welche Informationen durch die seichte Sprachverarbeitung erzeugt werden. Das Beispiel zeigt, welche Etiketten je Wort emittiert werden; Wortart-Etiketten sind grün hervorgehoben, Lemmata orange, Chunk-Etiketten rot und Instruktionsnummern blau.

Natürlichsprachliche Sequenz

place the orange juice back on the table then go to the window

Durch die seichte Sprachverarbeitung erzeugte Etiketten

place	the	orange	juice	back	on	the	table	then	go	to	the	window
VB	DT	NN	NN	RB	IN	DT	NN	RB	VB	TO	DT	NN
place	the	orange	juice	back	on	the	table	then	go	to	the	window
B-VP	B-NP	I-NP	I-NP	B-ADVP	B-PP	B-NP	I-NP	B-ADVP	B-VP	B-PP	B-NP	I-NP
0	0	0	0	0	0	0	0	1	1	1	1	1

Erläuterung

Je Wort werden vier Etiketten erzeugt. Phrasen, wie *the orange juice*, werden durch Chunk-Etiketten im *IOB*-Format markiert. Auch die Instruktionsnummern bilden Wortsequenzen; das Beispiel wird in zwei Einzelinstruktionen unterteilt.

6.3 Erkennung von Eigennamen

Die Erkennung von Eigennamen ist eine der Standardaufgaben der Computerlinguistik und daher auch Teil üblicher Computerlinguistik-Fließbänder (siehe Abschnitt 2.3.4.5). Im Kontext von ProNat ist die Erkennung von Eigennamen hauptsächlich für den späteren Aufbau des situativen Kontexts von Interesse (siehe Abschnitt 7.4).

Da für die Erkennung von Eigennamen weder durch die Fokussierung auf gesprochene Sprache noch durch die Zielsetzung der Programmierung besondere Anforderungen entstehen, kann eine bereits vorhandene Implementierung verwendet werden. Für ProNat wird, wie auch für die Erkennung der Wortart (siehe Abschnitt 6.2), das Werkzeug *SENNA* verwendet [Col+11]. *SENNA* erzielt für die Vergleichsaufgabe der *Conference on Computational Natural Language Learning* aus dem Jahr 2003 zur Eigennamenerkennung (*CoNLL shared Task 2003*) [TD03] für das F_1-Maß einen Wert von 0,896. Damit wird zwar nicht die Qualität moderner Ansätze erreicht, welche einen Wert von bis zu 0,943 erzielen[28]. Diese benötigen jedoch deutlich größere Modelle und klassifizieren damit nicht so schnell wie *SENNA*, weshalb dieses Werkzeug verwendet wird. Die Fließbandstufe zur Erkennung von Eigennamen dient als Adapter zu *SENNA*. Sie überführt die einzelnen Wörter in die von *SENNA* erwartete Darstellung und schreibt die erhaltenen Ergebnisse zurück. Da es sich bei den Wort-Etiketten um einen standardisierten Satz handelt (engl. *tag set*), können jederzeit andere Implementierungen angebunden werden.

6.4 Erkennung semantischer Rollen

Die Erkennung semantischer Rollen schlägt in der Computerlinguistik die Brücke zwischen der Syntax und der Semantik einer Äußerung. Zwar werden die Rollen aus syntaktischen Eigenschaften abgeleitet; die so erzeugten Prädikat-Argument-Strukturen bilden aber eine leichtgewichtige, semantische Interpretation (siehe Abschnitt 2.3.4.8). Im Kontext von ProNat können so einzelne Aktionen in Äußerungen erkannt und Akteure und Argumente zugeordnet werden. Diese werden später unter anderem zum Aufbau des situativen Kontexts, bei der Bestimmung von Schleifenkörpern und zur Abbildung der Äußerung auf ein Zielsystem (und die Umgebung) verwendet (siehe Abschnitte 7.4, 7.6 und 7.8).

Da die Erkennung semantischer Rollen ein inzwischen formalisiertes Problem der Computerlinguistik darstellt (siehe Abschnitt 2.3.4.8), existieren viele vergleichbare Implementierungen. Wie üblich verwenden diese vor allem überwachte Lernverfahren, die auf Textdokumenten trainiert werden, und sind damit nicht ohne Weiteres für gesprochene Äußerungen verwendbar. Zwei Eigenschaften gesprochener Sprache erschweren die Erkennung der semantischen Rollen besonders: ungrammatikalische Satzstrukturen und das Fehlen von Interpunktion. Erstere stehen im unmittelbaren Gegensatz zu den grammatikalisch korrekten Trainingsdaten. Für überwachte Lernverfahren gilt: Situationen, die während des Trainings nicht auftraten, können später nicht zuverlässig klassifiziert werden. Das

[28] Siehe hierzu: `https://nlpprogress.com/english/named_entity_recognition.html`, zuletzt besucht am 24.02.2021.

Fehlen der Interpunktion erschwert die Klassifikation zusätzlich. Während in Textdokumenten der Umfang der Prädikat-Argument-Strukturen durch die Satzgrenzen beschränkt werden, können diese in gesprochenen Äußerungen potenziell die ganze Äußerung überspannen.

Als Basis für die Fließbandstufe zur Erkennung semantischer Rollen [Hey16] wird in ProNat das Werkzeug *SENNA* verwendet[29] [Col+11]. *SENNA* erzielt für die Vergleichsaufgabe der *Conference on Computational Natural Language Learning* aus den Jahren 2004 und 2005 zur Klassifikation semantischer Rollen (*CoNLL shared Task 2004* und *2005*) [CM04; CM05] für das F_1-Maß einen Wert von 0,755. SENNA stellt die Prädikat-Argument-Strukturen nur in Form der abstrakten *A**-Rollen dar (siehe Abschnitt 2.3.4.8). Daher werden diese zusätzlich auf konkrete *PropBank*- und – sofern vorhanden – *FrameNet*- und *VerbNet*-Rollen-Kombinationen abgebildet. Die Verarbeitung einer Äußerung verläuft wie folgt. Zunächst werden die Token ausgelesen. Um die Problematik der fehlenden Interpunktion abzumildern, kann die Äußerung dabei anhand der zuvor bestimmten Instruktionen aufgegliedert werden bevor eine weitere Verarbeitung stattfindet (siehe Abschnitt 6.2). Ob Äußerungen als Ganzes oder nach Instruktionen unterteilt werden, kann konfiguriert werden[30]. Die Einzelwörter werden anschließend in das von *SENNA* erwartete Datenformat überführt und zur Klassifikation an *SENNA* übergeben. *SENNA* gibt die erkannten semantischen Rollen pro Prädikat unter Verwendung der *A**-Rollen und im *IOBES*-Format zurück (siehe Abschnitt 2.3.6). Die abstrakten *A**-Rollen werden anschließend auf konkrete *PropBank*- und – sofern vorhanden – auf *VerbNet*- und *FrameNet*-Rollen abgebildet. Hierzu wird zunächst für jedes Prädikat das entsprechende *PropBank*-Prädikat ermittelt. *PropBank* führt je nach Bedeutung mehrere Argumentsätze je Prädikat. Daher werden die Argumentsätze jeweils mit den vorhandenen *A**-Rollen abgeglichen. Kann mehr als ein Argumentsatz auf die *A**-Rollen abgebildet werden, wird der erste verwendet, da die Argumentsätze in *PropBank* gemäß ihrer Wahrscheinlichkeit geordnet sind. Zusätzlich stellt das Werkzeug *PIKES* [CRA16] Abbildungen zwischen *PropBank*, *FrameNet* und *VerbNet* zur Verfügen, welche übernommen werden.

Die Qualität der Klassifikation semantischer Rollen durch *SENNA* wurde anhand der 21 Transkriptionen zu den Szenarien sechs und sieben des ProNat-Korpus bemessen. Für die Erkennung des Prädikates (*V*-Rolle), der Argumente (*A**-Rollen) und des vollständigen Rollensatzes (*V*-Rolle und alle *A**-Rollen) wurden jeweils Präzision, Ausbeute und das F_1-Maß bestimmt. Die Ergebnisse sind in Tabelle 6.5 aufgeführt. Insbesondere die Bestimmung des Prädikates gelingt äußerst präzise; die Ausbeute bei der Erkennung der Argumente ist hingegen ausbaufähig; die Bestimmung aller Argumente stellt allerdings auch unter den gegebenen Bedingungen eine große Herausforderung dar (siehe hierzu die Diskussion zu Beginn des Abschnitts). Insgesamt wird jedoch ein sehr guter Wert für das F_1-Maß von 0,785 erreicht. Damit wird sogar das Ergebnis für die Vergleichsaufgabe übertroffen (F_1 für *CoNLL shared Task 2005*: 0,755). Es scheint, als würden die vergleichsweise

[29] In Vortests, die anhand von Äußerungen aus dem stationären Datensatz zu den Szenarien eins bis drei durchgeführt wurden, zeigte sich, dass die Klassifikation von *SENNA*, im Gegensatz zu vielen verwandten Ansätzen, robust gegenüber grammatikalischen Fehlern ist. Ein nähere Betrachtung der Qualität der Klassifikation durch *SENNA* befindet sich am Ende des Abschnitts.

[30] Die Standardkonfiguration sieht die Unterteilung in Instruktionen vor. Die Aufteilung nach Instruktionen birgt zwar die Gefahr, dass eigentlich zusammengehörende Prädikat-Argument-Strukturen aufgetrennt werden und somit nicht mehr korrekt klassifiziert werden können. Im Allgemeinen zeigt sich aber, dass die Erkennung der semantischen Rollen besser gelingt, wenn der Umfang der natürlichsprachlichen Sequenz geringer ist.

Tabelle 6.5: Evaluationsergebnisse für die Erkennung semantischer Rollen durch *SENNA*: Die Tabelle zeigt die Ergebnisse für die Erkennung des Prädikats, der zugehörigen Argumente, sowie das Gesamtergebnis.

Klassifizierter Bestandteil	Präzision	Ausbeute	F_1
Prädikat (*V*-Rolle)	0,939	0,799	0,863
Argumente (*A**-Rollen)	0,799	0,678	0,734
Gesamt (*V*- und *A**-Rollen)	0,854	0,726	0,785

einfachen Formulierungen, wie sie im ProNat-Korpus vorkommen, bei der Klassifikation stärker ins Gewicht fallen als die zum Trainingskorpus verschiedenen Rahmenbedingungen (gesprochene Sprache, fehlende Interpunktion, andere Inhalte etc.).

Beispiel:
Semantische Rollen

In diesem Beispiel soll der Unterschied zwischen abstrakten *V*- und *A**-Rollen, wie sie semantische Rollenerkenner erzeugen, und konkreten *PropBank*-Rollen veranschaulicht werden. Zur Erzeugung der konkreten Rollen wird der *A**-Rollensatz und das zugehörige Prädikat, hier *put*, benötigt.

Natürlichsprachliche Sequenz

Robo put the green cup on the table

Abstrakte *V*- und *A-Rollen (IOBES-Format)**

$$[\text{Robo}]_{\text{S-A0}}\ [\text{put}]_{\text{S-V}}\ [\text{the}]_{\text{B-A1}}\ [\text{green}]_{\text{I-A1}}\ [\text{cup}]_{\text{E-A1}}\ [\text{on}]_{\text{B-A2}}\ [\text{the}]_{\text{I-A2}}\ [\text{table}]_{\text{E-A2}}$$

Konkrete PropBank-Rollen

Erläuterung

SENNA erzeugt für die natürlichsprachliche Sequenz zunächst die semantischen *A**-Rollen im *IOBES*-Format. Mithilfe des Prädikates *put* kann der Rollensatz anschließend (in diesem Falle eindeutig) auf den *PropBank*-Eintrag `put.01` mit dem zugehörigen Argumentsatz abgebildet werden. In *PropBank* ist verzeichnet, dass für *put* in dieser Bedeutung, die *A**-Rolle *A0* dem *putter*, *A1* dem *thing put* und *A2* dem *where put* entspricht[31].

[31] *PropBank*-Eintrag zu `put.01`: `http://verbs.colorado.edu/propbank/framesets-english-aliases/put.html`, zuletzt besucht am 24.02.2021.

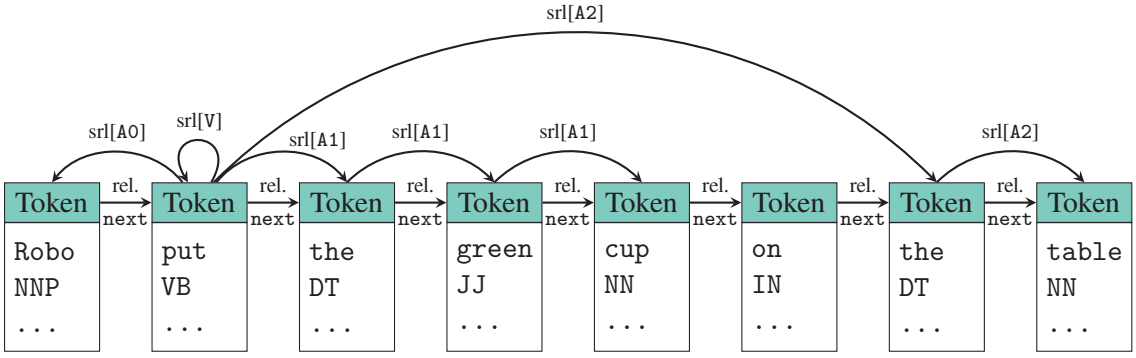


Abbildung 6.1: Beispielhafter initialer ProVAT-Graph, wie von der Fließbandstufe zur Grapherzeugung generiert.

6.5 Grapherzeugung

Die letzte Stufe des Vorverarbeitungsfließbandes von ProVAT dient zur Erzeugung eines initialen Graphen aus den zuvor generierten Informationen. Hierzu wird je Wort (bzw. Token) ein neuer Knoten vom Typ `Token` erzeugt und das jeweilige Wort als Attribut abgelegt (Attributname `value`)[32]. Die Knoten werden gemäß ihrer Reihenfolge in der Wortsequenz mit Kanten vom Typ `relation` verbunden. Die weiteren je Wort erzeugten Informationen aus den Fließbandstufen zur seichten Sprachverarbeitung und zur Erkennung von Eigennamen werden als Attribute in den Knoten hinterlegt (siehe Abschnitt 6.2 und Abschnitt 6.3). Anschließend werden zur Darstellung der semantischen Rollen Kanten vom Typ *srl* erzeugt (siehe Abschnitt 6.4). Je Prädikat(-Knoten) wird eine reflexive Kante generiert; dadurch wird die Wurzel der Prädikat-Argument-Struktur angezeigt. Ausgehend von diesem Knoten wird für jede zugehörige *A**-Rolle eine Kante zum ersten Wort(-Knoten) angelegt; weitere zur *A**-Rolle gehörende Wörter werden über weitere Kanten verbunden. Die zusätzlichen Informationen (Rollenname, *PropBank*, *VerbNet*- und *FrameNet*-Rollen etc.) werden als Attribute in den jeweiligen Kanten hinterlegt. Abbildung 6.1 zeigt einen beispielhaften initialen Graphen; visualisiert wird jedoch nur ein Teil der generierten Informationen, um die Übersichtlichkeit zu wahren. Der erzeugte Graph bildet die Grundlage für die anschließende semantische Analyse der Äußerung durch die ProVAT-Agenten.

[32] Eine vollständige Auflistung der verwendeten Typen und zugehörigen Attribute zur Erzeugung des initialen Graphen befindet sich im Anhang in Anhang B.

7 Agenten für Sprachverständnis

„Language is the dress of thought."

– Samuel Johnson

Die Aufgabe der Agenten in ProNat ist die Erzeugung eines tiefen Verständnisses natürlicher Sprache. Hierzu analysieren Sie auf Grundlage des während der Vorverarbeitung erzeugten initialen ProNat-Graphen die Semantik natürlichsprachlicher Äußerungen. Das Analyseergebnis bildet die Grundlage für die Erzeugung von Programmen (bzw. die Synthese von Quelltext) durch das Nachverarbeitungsfließband. Sämtlicher Erkenntnisgewinn, der dafür nötig ist, wird bereits durch die Agenten geleistet. Jeder Agent löst eine wohldefinierte und abgeschlossene Aufgabe zur Analyse der Semantik. Im Zusammenspiel erzeugen die Agenten ein tiefes Verständnis natürlicher Sprache. Da durch die Architektur PARSE festgelegt ist, dass Agenten unabhängig voneinander agieren, kann zur Lösung der jeweiligen Aufgabe des Agenten ein beliebiger Ansatz gewählt werden. Die Agenten können statisk-, regel- oder wissensbasierte Techniken anwenden; auch hybride Ansätze sind möglich. Die Beschreibung der einzelnen Agenten in den nachfolgenden Abschnitten wird aufzeigen, dass alle für ProNat entwickelten Agenten unterschiedliche Ansätze verfolgen. Die Agenten hinterlegen ihre Analyseergebnisse in einer geteilten Datenstruktur, dem ProNat-Graphen; durch die Verwendung des Graphen gelingt es, Resultate, die mithilfe unterschiedlicher Techniken gewonnen wurden, zu vereinen. Die Rahmenarchitektur Luna sorgt dafür, dass die Agenten parallel und wiederholt ausgeführt werden (siehe Abschnitt 4.2.4). Dadurch können zyklische Abhängigkeiten zwischen den ProNat-Agenten größtenteils aufgelöst werden und die Agenten können von Zwischenergebnissen anderer Agenten profitieren. Auf diese Weise wird der Graph sukzessive mit neuen Informationen angereichert; im Gesamten bilden die Analyseergebnisse der Agenten eine Interpretation der Semantik natürlichsprachlicher Äußerungen.

Um die gesetzte Zielsetzung der Programmierung mit gesprochener Sprache zu erfüllen, wurden für ProNat Agenten entwickelt, die sich in drei Kategorien gliedern. Die erste Kategorie umfasst Agenten, die allgemeine Sprachverständnisaufgaben lösen. Hierzu zählen unter anderen die Modellierung des sprachlichen Kontextes und die Analyse von Korreferenzen. Agenten, die sich der zweiten Kategorie zuordnen lassen, untersuchen natürlichsprachliche Äußerungen dahingehend, ob sich aus diesen programmatische Strukturen ableiten lassen. Zu den Aufgaben, die von diesen Agenten gelöst werden, zählen beispielsweise die Erkennung von bedingten Verzweigungen oder die Synthese von Methodendefinitionen. Die letzte Kategorie bilden die funktionalen Agenten. Diese gewährleisten die Funktionsfähigkeit des Gesamtsystems, beispielsweise durch die Initiierung von Rückfragen, falls eine Äußerung nicht interpretiert werden kann. Konkret wurden die folgenden Agenten für ProNat entwickelt:

Agenten für allgemeines Sprachverständnis

- Erkennung von Disfluenzen in gesprochener Sprache (Abschnitt 7.1)
- Disambiguierung von Wortbedeutungen (Abschnitt 7.2)
- Modellierung und Etikettierung von Diskurs-Themen (Abschnitt 7.3)
- Modellierung des sprachlichen Kontextes (Abschnitt 7.4)
- Korreferenzanalyse (Abschnitt 7.5)

Agenten zur Erkennung programmatischer Strukturen

- Synthese von Kontrollstrukturen (Abschnitt 7.6) inklusive
  - bedingten Verzweigungen,
  - Nebenläufigkeit und
  - Schleifen
- Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache (Abschnitt 7.7)
- Synthese von Methodendefinitionen und Skripten (Abschnitt 7.8)

Funktionale Agenten

- Automatische Auswahl von Zielsystem- und Umgebungsontologien (Abschnitt 7.9)
- Dialog-Agent (Abschnitt 7.10)
- Funktionswächter-Agenten (Abschnitt 7.11)

Der Grund für die Auswahl ebendieser Agenten ist folgender. Diese Agenten erzeugen die minimal erforderlichen Analyseschritte, um Programme aus natürlichsprachlichen, gesprochenen Äußerungen synthetisieren zu können. Sie bilden somit die minimal erforderliche Menge zur Erfüllung der Zielstellung. Durch die Entwicklung weiterer Agenten könnte die Qualität des Gesamtanalyseergebnisses gegebenenfalls verbessert werden. Wie in Kapitel 4 beschrieben, ist die Erweiterung von ProNat um weitere Agenten jederzeit möglich. Die für ProNat entwickelten Agenten werden in den folgenden Abschnitten beschrieben. Wie zuvor erläutert, lösen die Agenten jeweils eine abgeschlossene, wohldefinierte Aufgabe. Somit leistet jeder einzelne einen Beitrag zur Forschung. Dementsprechend wird in den jeweiligen Abschnitten nicht nur der zur Umsetzung gewählte Ansatz beschrieben, sondern auch verwandte Ansätze – sofern solche existieren – und zugrundeliegende Forschungsergebnisse diskutiert. Die Agenten können aufgrund ihrer durch PARSE definierten Unabhängigkeit einzeln, intrinsisch evaluiert werden; somit können für alle Agenten in den jeweiligen Abschnitten Evaluationsergebnisse präsentiert werden[1]. Damit wird auch der in Abschnitt 5.1 aufgestellten Forderung nach einer kontinuierlicher Evaluation von ProNat genüge getan.

[1] Ausgenommen hiervon sind die Funktionswächter-Agenten sowie einer der beiden Agenten zur Disambiguierung.

7.1 Erkennung von Disfluenzen in gesprochener Sprache

Der Fokus auf die Programmierung mit *gesprochener* Sprache bringt eine Vielzahl an Herausforderungen mit sich (siehe Abschnitt 2.3.12). Abgesehen davon, dass bei der Synthese von Quelltext aus gesprochenen Anweisungen die Besonderheiten natürlicher Sprache, wie beispielsweise Mehrdeutigkeit oder Implizitheit, beachtet werden müssen, ist die Analyse gesprochener Sprache (gegenüber der Schriftsprache) mit zusätzlichen Schwierigkeiten behaftet. So erzeugen automatische Spracherkenner beispielsweise keine zuverlässigen Interpunktionen, weshalb für ProNat vollständig darauf verzichtet wird (siehe Abschnitt 6.1). Ebenso zeigt sich, dass Sprecher bei spontanen Äußerungen teilweise grammatikalische Regeln außer Acht lassen. Während die meisten dieser Herausforderungen implizit in den Entwurf der Komponenten von ProNat eingehen[2] , soll ein weiteres Phänomen gesprochener Sprache explizit analysiert werden: das Auftreten von Disfluenzen (engl. *disfluencies*).

Unter *Disfluenzen* versteht man in der Linguistik Unterbrechungen im Sprachfluss, zum Beispiel durch die Verwendung eines Verzögerungslauts wie *ehm*. Menschen verwenden Verzögerungslaute vorwiegend um (Denk-)Pausen zu füllen; sie können aber auch verwendet werden, um eine Unsicherheit auszudrücken oder die Korrektur einer zuvor getätigten Äußerung einzuleiten, wie in folgendem Beispiel: „hey robo take *uhm* the apple *err* the orange“. Menschlichen Zuhörern gelingt die Interpretation derartiger Sprechsituationen in den meisten Fällen intuitiv. Eine programmatische Analyse von Disfluenzen gestaltet sich hingegen schwierig, da diese nicht einfach anhand syntaktischer oder lexikalischer Eigenschaften erkannt werden können. Für die mit ProNat angestrebte Programmierung mit natürlicher Sprache ist die Analyse von Disfluenzen jedoch unumgänglich. Weiterführende Analysen könnten durch auftretende Disfluenzen (bzw. Transkriptionen dieser) negativ beeinflusst werden. Auch Anweisungen, die korrigiert oder revidiert werden, haben Einfluss auf die Interpretation von Äußerungen, da sie die Intention des Gesagten verändern. Wird dies ignoriert, könnten in der Folge zum Beispiel falsche oder überflüssige Instruktionen erzeugt werden.

7.1.1 Disfluenzen als Problemstellung der Computerlinguistik

Disfluenzen stellen eine Unterbrechung des Sprachflusses dar. Sie treten hauptsächlich in spontaner, gesprochener Sprache auf. Disfluenzen äußern sich zumeist durch die Verwendung von Verzögerungslauten oder Wort- bzw. Phrasenwiederholungen. In der Literatur werden für gewöhnlich drei unterschiedliche Arten von Disfluenzen unterschieden [Shr94]: Reparaturen, entfernbare Disfluenzen und unkorrigierte Disfluenzen. Unterbricht ein Sprecher seine Äußerung, um Teile davon zu widerrufen oder zu korrigieren, spricht man von einer Reparatur, wie im eingangs diskutierten Beispiel. Je nach Art der durchgeführten Modifikation unterscheidet man Reparaturen zusätzlich nach Entfernungen, Einsetzungen, Wiederholungen, Ersetzungen oder Neustarts. Disfluenzen, die entfernt werden können, tragen keine Semantik, sondern unterbrechen lediglich den Sprachfluss

[2] Beispielsweise wird in der Vorverarbeitung aufgrund der unsicheren grammatikalischen Strukturen auf eine vollständige Zerteilung der Eingabe verzichtet und stattdessen eine seichte Variante bevorzugt (siehe Abschnitt 6.2).

durch eine (Denk-)Pause, beispielsweise durch einen Verzögerungslaut (Beispiel: „I *ehm* would like to [. . .]“). Die letzte Art von Disfluenzen bilden die unkorrigierten. Hier wird der Sprachfluss zwar unterbrochen und die Äußerung bleibt unvollständig, es findet aber keine Korrektur durch den Sprecher statt (Beispiel: „I would like to *th- err* everyone who [. . .]“). Alle Disfluenzen folgen laut Shriberg einer gewissen Struktur, die aus bestimmten Bestandteilen aufgebaut ist [Shr94]:

- Unterbrechungspunkt (engl. *interruption point*, IP): Der Unterbrechungspunkt bezeichnet die Stelle, an der der Sprachfluss unterbrochen wird.
- Interregnum (IM): Das Interregnum leitet eine Bearbeitungsphase ein; es dient der Strukturierung des Diskurses. Interregna können aus Verzögerungslauten (z. B. *ehm*), Füllwörtern (z. B. *so*), Diskursmarkern (z. B. *well*) oder expliziten Bearbeitungsbegriffen bzw. -phrasen (z. B. *I mean*) oder einer Kombination dieser gebildet werden.
- Reparandum (RM): Das Reparandum ist der fehlerhafte, zu reparierende Teil einer Äußerung.
- Reparatur (RR): Die Reparatur (häufig auch als *Reparans* bezeichnet) ist der Teil der Äußerung, welcher (semantisch gesehen) das Reparandum ersetzen soll.

Das Vorhandensein der Bestandteile kann dabei je nach Art der Disfluenz variieren. Grundsätzlich sind alle Bestandteile bis auf den Unterbrechungspunkt optional. Eine Reparatur tritt aber üblicherweise nur gepaart mit einem Reparandum auf.

Beispiel:
Struktur von Disfluenzen

In diesem Beispiel soll grundlegende Struktur von Disfluenzen veranschaulicht werden. Die Beispieläußerung enthält alle Bestandteile von Disfluenzen, wie zuvor definiert.

Natürlichsprachliche Sequenz

hey robo take uhm the apple err I mean the orange

Struktur

hey robo take @ uhm the apple @ err I mean the orange
IM RM IM RR

Erläuterung

Die Äußerung wird durch den Verzögerungslaut *uhm* unterbrochen. Dieser bildet zusammen mit dem Unterbrechungspunkt (hier dargestellt durch das Zeichen @) ein Interregnum. Es folgt das Reparandum, zu erkennen am darauffolgenden zweiten Interregnum. Dieses besteht neben dem Verzögerungslaut *err* aus der Bearbeitungsphrase *I mean*. Der letzte Bestandteil ist die Reparatur. Im Beispiel können aus Sicht der Semantik die Interregna entfernt werden; zusätzlich ersetzt die Reparatur das Reparandum. Folglich lautet die korrigierte Äußerung: „hey robo take the orange“.

Wie in den meisten Gebieten der Computerlinguistik wurden anfänglich regelbasierte Verfahren eingesetzt, um Disfluenzen und etwaige Reparaturen zu erkennen [Hin83; BDS92]. Moderne Verfahren sind jedoch nahezu ausnahmslos statistikbasiert; dabei haben sich im Wesentlichen drei Strömungen etabliert. Zunächst gibt es Verfahren, die Grammatiken für spezialisierte Zerteiler (engl. *parser*) lernen [CJ01; RT13; HJ14]. Diese Verfahren sehen Disfluenzstrukturen als partielle Zerteilung der Eingabe an. Die zweite Strömung verwendet klassische Lernverfahren, vor allem solche, die auf bedingten Zufallsfeldern (engl. *conditional random field*, kurz *CRF*) basieren [GWG10; ZOH14; FDK15]. In den letzten Jahren werden vor allem (rekurrente) neuronale Netze zur Disfluenz-Erkennung eingesetzt, insbesondere bidirektionale *LSTMs* (siehe Abschnitt 2.2.2) [ZOH16; JJ17]. Einige Ansätze kombinieren verschiedene Verfahren. Wang et al. verwenden beispielsweise ein bidirektionales *LSTM*, dessen Klassifikation sie durch die Verwendung eines bedingten Zufallsfelds optimieren [Wan+16].

Für das Training von Verfahren, die überwachtes maschinelles Lernen verwenden, wird ein (ausreichend großer und annotierter) Datensatz benötigt. Für die Disfluenz-Erkennung hat sich das *Switchboard*-Korpus als De-Facto-Standard etabliert [GHM92; GB00]. Dieses besteht aus 2438 Telefongesprächsmitschnitten zwischen jeweils zwei Gesprächspartnern. Insgesamt trugen 520 Sprecher zur Datensammlung bei. Zu allen Mitschnitten existieren händisch erzeugte Transkriptionen, die zwischen Äußerungen von Sprecher *A* und *B* unterscheiden. 1126 der Transkriptionen wurden später in Anlehnung an die Notation von Shriberg, wie zuvor beschrieben, mit Disfluenz-Informationen annotiert [Mar+99; TMS03]. Mit Ausnahme der frühen regelbasierten Ansätze verwenden alle Verfahren das *Switchboard*-Korpus als Vergleichsdatensatz. Dabei hat sich bei der Aufteilung in Test-, Entwicklungs- (oder Validierungs) und Testdatensatz das von Charniak und Johnson vorgeschlagene Verfahren etabliert [CJ01]. Diese teilten den mit Disfluenz-Informationen versehenen Teil des *Switchboard*-Korpus in vier etwa gleich große Abschnitte ein. Die Abschnitte zwei und drei verwendeten Charniak und Johnson als Trainingsdatensatz. Der vierte Abschnitt wurde wiederum in drei gleich große Abschnitte geteilt und der erste von diesen als Testdatensatz verwendet; die übrigen zwei Unterabschnitte können als Entwicklungs- oder Validierungsdatensatz verwendet werden. Auch wenn alle modernen Ansätze auf Grundlage des *Switchboard*-Korpus entwickelt wurden, ist ein direkter Vergleich der Ergebnisse nur bedingt möglich. Zum einen klassifizieren die Verfahren unterschiedliche Bestandteile der Disfluenz-Strukturen. Zwar fokussieren sich die meisten Ansätze auf die Erkennung des Reparandums, jedoch zählen einige Teile des Interregnums dazu, andere bestimmen nur das erste Wort eines Reparandums andere alle Wörter. Zum anderen unterscheidet sich die Datengrundlage. Während vor allem die Zerteiler-basierten Ansätze die Baumstruktur des Originaldatensatzes verwenden, überführen die meisten anderen Ansätze die Annotationen in eine flache Darstellung mit einer Etikette pro Wort; diese Transformationen sind nicht bei allen Ansätzen gleich, jedoch zumeist sehr ähnlich, weshalb ein Vergleich anhand des F_1-Maß trotzdem möglich ist. Das erste Verfahren, das die beschriebene Aufteilung des Korpus nutze (Charniak und Johnson), erreichte einen Wert von 0,782. Moderne Zerteiler-basierte Ansätze erreichen Werte von bis zu 0,841 [HJ14]. Unter den *CRF*-basierten Verfahren erzielt der Ansatz von Ferguson et al. einen Wert von 0,854 [FDK15]. Unter Verwendung von bidirektionalen *LSTMs* kann ein Wert von 0,868 erzielt werden [JJ17]. Durch die Kombination eines bidirektionalen *LSTMs* mit einem *CRF* gelingt Wang et al. eine weitere Steigerung auf 0,871[Wan+16].

Tabelle 7.1: Zur Beschreibung von Disfluenz-Bestandteilen verwendete Etiketten (inklusive Kurzbeschreibung der Bedeutung).

Etikett	Kurzbeschreibung
O-DF	Nicht Teil einer (bzw. außerhalb einer) Disfluenz
B-RM	Beginn eines Reparandums
I-RM	Innerhalb eines Reparandums
B-RS	Beginn eines Reparans
I-RS	Innerhalb eines Reparans
FP	Gefüllte Pause
EE	Expliziter Bearbeitungsbegriff
DM	Diskursmarker

7.1.2 Vorgehen zur Erkennung von Disfluenzen

Für ProNat soll ein Agent entwickelt werden, der Disfluenzen in Äußerungen erkennt und die Struktur der Disfluenzen, wie zuvor beschrieben, klassifiziert. Hierzu soll ein bidirektionales, rekurrentes, neuronales Netz mit langem Kurzzeitgedächtnis (engl. *bidirectional long short-term memory*, kurz *Bi-LSTM*) trainiert werden (siehe Abschnitt 2.2.2). Das Modell wird einmalig eingelernt (*Offline-Training*); der Agent verwendet das erzeugte Modell lediglich zur Klassifikation. Als Datensatz wird das *Switchboard*-Korpus verwendet. Dieses bietet zum einen eine ausreichend große Datengrundlage für das Training eines überwachten Lernverfahrens, zum anderen können die erzielten Ergebnisse später mit denen anderer Ansätze verglichen werden[3].

Um das *Switchboard*-Korpus als Datensatz für ein *Bi-LSTM* nutzen zu können, müssen Disfluenz-Strukturen, die in einer Baumstruktur dargestellt sind, in eine sequenzielle Darstellung überführt werden. Hierzu wird pro Wort ein Etikett angebracht, das die Zugehörigkeit zu einer Disfluenz-Struktur repräsentiert. Bei der Transformation und der Wahl der Etiketten diente die Arbeit von Zayats et al. als Orientierung [ZOH16]. Die verwendeten Etiketten sind in Tabelle 7.1 aufgeführt. Zu beachten ist, dass die Art der verwendeten Wörter im Interregnum feingliedrig unterschieden wird; die Etiketten *FP*, *EE* und *DM* beschreiben somit alle Bestandteile von Interregna. Die Reparanda, die Reparaturen (hier als *Reparans* bezeichnet) und alle Bestandteile, die nicht zu einer Disfluenz zugeordnet wurden, werden im *IOB*-Format dargestellt (siehe Abschnitt 2.3.6), welches Wörter zu Beginn eines Bestandteils mit einem gesonderten Etikett-Präfix *B* versieht. Alle weiteren Wörter werden mit dem Präfix *I* versehen; die Wörter, die nicht Teil einer Disfluenz sind, werden mit dem Präfix *O* markiert. Die in Abschnitt 7.1.1 beschriebene Beispieläußerung würde folgende Etiketten erhalten:

$$[\text{hey}]_{\text{O-DF}}\ [\text{robo}]_{\text{O-DF}}\ [\text{take}]_{\text{O-DF}}\ [\text{uhm}]_{\text{FP}}\ [\text{the}]_{\text{B-RM}}\ [\text{apple}]_{\text{I-RM}}\ [\text{err}]_{\text{FP}}\ [\text{I}]_{\text{EE}}$$
$$[\text{mean}]_{\text{EE}}\ [\text{the}]_{\text{B-RS}}\ [\text{orange}]_{\text{I-RS}}$$

[3] Die ProNat-Korpora eignen sich nicht als Grundlage. Das mithilfe der Online-Studie erzeugte Korpus enthält keine Disfluenzen, da es sich um textuelle Beschreibungen handelt, und das im Zuge der stationären Studien erzeugte Korpus hat einen zu geringen Umfang.

Zusätzlich werden alle Satzzeichen aus dem Korpus entfernt, da das Modell später im Kontext von ProNat verwendet werden soll[4]. Als Eingabesequenz dient somit die ganze Äußerung eines Sprechers bis ein Sprecherwechsel stattfindet. Das Entfernen der Satzzeichen erschwert die Klassifikationsaufgabe, da die Eingabesequenzen länger werden und Interpunktion potenziell wertvoll für Klassifikation ist.

Als Eingabe-Merkmale für den Klassifikator werden die Wortsequenz, die zugehörigen Wortartmarkierungen, Chunk-Etiketten und Umgebungsvektoren verwendet. Je Wort wird ein Eigenschaftsvektor erstellt, der ebendiese Informationen kodiert; der Vektor ist wie folgt zusammengesetzt. Das Wort an sich wird als Worteinbettung nach dem *GloVe*-Verfahren repräsentiert [PSM14]. Es wird ein vortrainiertes Modell verwendet, das auf zwei Milliarden Meldungen des Nachrichtendienstes *Twitter* trainiert wurde und 50 Dimensionen aufweist[5]. Die Wortarten und die Chunk-Etiketten werden jeweils mithilfe einer 1-aus-n-Kodierung dargestellt (siehe Abschnitt 2.3.16). Zusätzlich wird ein Vektor definiert, der die Wortumgebung beschreibt. Dieser entspricht im Wesentlichen dem von Wang et al. verwendeten [Wan+16]. Im Vektor wird folgendes kodiert. Wiederholt sich das Wort in einer Umgebung von fünfzehn Wörtern (rechts oder links des Wortes), wird an der entsprechenden Stelle des Vektors eine 1 gesetzt; innerhalb der nächsten 30 Dimensionen des Vektors wird ebendiese Information für die Wiederholung der Wortart hinterlegt. Zusätzlich wird auch das wiederholte Auftreten von Bigrammen bzw. Wortart-Bigrammen in einer Umgebung von vier Wörtern kodiert. Zuletzt wird hinterlegt, ob sich innerhalb von zwei Wörtern ähnliche Wörter befinden[6]. Als Maß wird die Jaro-Winkler-Ähnlichkeit verwendet[7]. Überschreitet die Ähnlichkeit den Schwellenwert von 0,8, wird an der betreffenden Stelle des Vektors eine 1 gesetzt. Durch diese Ähnlichkeitsbetrachtungen können Disfluenzen, wie in „get *th uhm the* juice“, explizit kodiert werden.

Das verwendete *Bi-LSTM* besteht aus 100 Neuronen in der verdeckten Schicht. In der Ausgabe-Schicht wird die *Softmax*-Funktion zur Erzeugung der acht Zielklassen verwendet (siehe Tabelle 7.1). Die Lernrate beträgt während des Trainings 0,1. Als Größe der Stapel (engl. *batch size*) wurde 20 festgelegt. Die maximale Anzahl an Trainingsepochen beträgt 20. Allerdings wurde die Technik *frühzeitiger Abbruch* (engl. *early stopping*) verwendet; dadurch wird das Training beendet, sobald nach drei aufeinanderfolgenden Trainingsdurchläufen keine Verbesserung mehr hinsichtlich der Evaluationsmetrik (hier F_1) erzielt werden kann. Getestet wird hierbei an den Äußerungen des Entwicklungsdatensatzes. Diese Konfiguration entspricht im Wesentlichen denen von Zayats et al. und Wang et al. vorgeschlagenen [ZOH16; Wan+16], die derzeit dem Stand der Technik entsprechen. Die maximale Länge der Eingabesequenz (Wörter) wurde jedoch vom üblichen Wert 50 auf 60 erhöht, um den längeren Eingabesequenzen, die durch das Zusammenfassen ganzer Äußerungen (ohne Interpunktion) entstehen, gerecht zu werden.

[4] In ProNat stehen keine Satzzeichen zur Verfügung (siehe Abschnitt 6.1)

[5] Verwendetes Modell: `glove.twitter.27B.50d`, `https://nlp.stanford.edu/projects/glove/`, zuletzt besucht am 24.02.2021.

[6] Wang et al. verwenden lediglich eine Umgebung von einem Wort links und rechts.

[7] Wang et al. definieren ein eigenes Ähnlichkeitsmaß, welches auch hier nachimplementiert wurde. Mit der Jaro-Winkler-Ähnlichkeit konnten in Vortests für Beispiele aus dem ProNat-Korpus jedoch bessere Ergebnisse erzielt werden.

Tabelle 7.2: Evaluationsergebnisse des Disfluenz-Klassifikators auf dem Testdatensatz des *Switchboard*-Korpus je Klasse und im Mittel.

Klasse	Präzision	Ausbeute	F_1
O-DF	0,951	0,990	0,970
B-RM	0,895	0,750	0,816
I-RM	0,848	0,591	0,697
B-RS	0,896	0,754	0,819
I-RS	0,824	0,434	0,559
FP	0,989	0,993	0,991
EE	0,970	0,938	0,953
DM	0,960	0,903	0,931
Ø	0,917	0,794	0,843

Nach Abschluss des Trainings kann das erzeugte Modell anhand des Testdatensatzes evaluiert werden. Die Qualität der Klassifikation wird mithilfe der Metriken Präzision, Ausbeute und F_1 je Klasse und im Mittel über alle Klassen ermittelt; Tabelle 7.2 zeigt die Ergebnisse. Der Wert für das F_1-Maß beträgt im Mittel 0,843 – ein sehr gutes Resultat[8]. Die Präzision übersteigt für fast alle Klassen die Ausbeute, ein erwünschter Effekt, schließlich sollen möglichst wenig *falsch positive* Klassifikationen erzeugt werden[9]. Dementsprechend ist auch das Ergebnis für die Klasse *O-DF* als positiv zu bewerten, da vor allem Wörter, die nicht Teil einer Disfluenz sind, korrekt klassifiziert werden sollten. Hinsichtlich der Werte der weiteren Klassen fällt auf, dass diejenigen, die Interregna beschreiben und zumeist aus bestimmten Wörtern oder Phrasen bestehen (*FP*, *EE* und *DM*), sehr exakt klassifiziert werden. Die Klassifikation der Reparanda und Reparans gelingt weniger gut; zwar wird häufig der Beginn des jeweiligen Bestandteils korrekt ermittelt (*B-**), der Umfang der Struktur wird aber häufig zu gering gewählt. Letzteres lässt sich an den geringen Ausbeute-Werten für die weiterführenden Wörter der Bestandteile (*I-**) ablesen.

Für eine abschließende Bewertung der Ergebnisse, werden sie mit denen des derzeit besten Ansatzes von Wang et al. verglichen [Wan+16]. Wie die meisten verwandten Arbeiten betrachtet der Ansatz von Wang et al. nur die Erkennung des Reparandums (*RM*); die Autoren unterscheiden auch nicht zwischen Beginn und weiterführenden Wörtern des Reparandums. Daher werden in Tabelle 7.3 nur die Ergebnisse für die Erkennung der Reparanda betrachtet (wortweise, **RM-* Etiketten). Der Wert für das F_1-Maß des hier entwickelten Klassifikators ist um 0,079 (9,1%) schlechter als das Vergleichsverfahren. Diese Differenz ist unter Betrachtung der unterschiedlichen Randbedingungen ein akzeptables Ergebnis. Das hiesige *Bi-LSTM* wurde auf einer Version des Korpus ohne Interpunktion trainiert, Wang et al. trainierten hingegen auf dem unveränderten Korpus. Zudem wurde das Vergleichsverfahren nur auf die Erkennung von Reparanda trainiert; das bedeutet, es handelt sich um eine binäre Klassifikation. Der hier entwickelte Klassifikator unterscheidet

[8] Bei der Bewertung der Ergebnisse muss jedoch berücksichtigt werden, dass die Klasse *O-DF*, die alle Wörter angibt, die nicht Teil einer Disfluenz sind, deutlich dominiert.

[9] Andere Agenten könnten als Disfluenz markierte Wörter ignorieren, weshalb die Präzision der Klassifikation im Vordergrund stehen sollte.

Tabelle 7.3: Vergleich des für ProNat entwickelten Disfluenz-Klassifikators mit dem besten Vergleichsverfahren von Wang et al. [Wan+16]: Betrachtet werden nur Reparanda (wortweise), Beginn und weiterführende Wörter werden nicht unterschieden.

Modell	Präzision	Ausbeute	F_1
Bi-LSTM ProNat	0,906	0,702	0,792
Bi-LSTM-CRF Wang et al. [Wan+16]	0,910	0,836	0,871

Tabelle 7.4: In der Standardkonfiguration des Agenten definierte Wortlisten zur Erkennung von Verzögerungslauten und expliziten Bearbeitungsbegriffen.

Typ	Wörter
Verzögerungslaute (*FP*)	um, uh, eh, uh-oh, un, er, eh, ehm, em, ah, ahm, oh, uhm, duh, ooh, gee, hum, hm, ugh, gosh, huh, ahm, yah, hmm
Explizite Bearbeitungsbegriffe (*EE*)	mean, sorry, excuse

hingegen acht Klassen gleichzeitig, eine deutlich schwierigere Klassifikationsaufgabe. Zuletzt verwenden Wang et al. zusätzlich ein bedingtes Zufallsfeld; dieser Zusatz verhilft ihnen jedoch nur zu leicht besseren Ergebnissen und steht in keinem Verhältnis zum zusätzlichen (Rechen-)Aufwand. Die Ergebnisse zeigen, dass sich der Klassifikator – unter Berücksichtigungen der projektspezifischen Rahmenbedingungen – für den Einsatz in ProNat eignet. Besonders positiv kann die erzielte Präzision bewertet werden. Diese liegt auf dem Niveau des Vergleichsverfahrens und ist für die Verwendung in ProNat, wie zuvor diskutiert, von besonderer Bedeutung.

7.1.3 Implementierung des Agenten zur Erkennung von Disfluenzen

Um das erzeugte *Bi-LSTM*-Modell verwenden zu können, wurde ein PARSE-Agent implementiert. Dieser liest zunächst alle Wörter aus dem ProNat-Graphen aus; zusätzlich werden die Wortarten und die Chunk-Etiketten ausgelesen. Damit ist der Agent zur Erkennung von Disfluenzen lediglich von der Vorverarbeitungsstufe zur seichten Sprachverarbeitung abhängig (siehe Abschnitt 6.2). Die ausgelesenen Informationen werden anschließend in die zuvor beschriebene Vektordarstellung überführt und zur Klassifikation an das Modell übergeben.

Optional wird eine Nachverarbeitung der Klassifikationsergebnisse durchgeführt[10]. Diese prüft zunächst anhand von Wortlisten, ob alle Verzögerungslaute (Etikett *FP*) und expliziten Bearbeitungsbegriffe (Etikett *EE*) korrekt klassifiziert wurden und ändert gegebenenfalls die Etiketten. Die Wortlisten können konfiguriert werden; in Tabelle 7.4 sind die Listen der Standardkonfiguration aufgeführt. Anschließend wird überprüft, ob innerhalb von einem oder zwei Wörtern Wortwiederholungen auftreten. Ist dies der Fall, wird davon ausgegangen, dass es sich um ein Reparandum-Reparans-Paar handelt und die jeweiligen Etiketten werden vergeben. Diese Art kurzer Reparatur-Strukturen klassifiziert das *Bi-LSTM* häufig falsch, weshalb diese nachträgliche Korrektur

[10] Die Nachverarbeitung ist in der Standardkonfiguration des Agenten aktiviert.

Tabelle 7.5: Der für die Evaluation des Agenten zur Erkennung von Disfluenzen verwendete Datensatz.

	Szenario 13	Szenario 14	Gesamt
Aufnahmen	15	14	29
davon wiederholte	5	4	9
Wörter	1269	1392	2661
gefüllte Pausen (*FP*)	54	17	71
Diskursmarker (*DM*)	0	0	0
Explizite Bearbeitungsbegriffe (*EE*)	0	0	0
Reparanda	23	25	48
Reparans	26	26	52
Aufnahmen ohne Reparaturen	8	9	17

nötig ist. Zuletzt wird überprüft, ob alle Reparanda- und Reparans-Etiketten zusammenhänge Blöcke bilden; etwaige Lücken werden durch Änderungen der Etiketten geschlossen.

Abschließend schreibt der Agent seine Ergebnisse in den ProNat-Graphen zurück. Hierzu werden die erzeugten Etiketten an die Wörter annotiert; das bedeutet, den Knoten vom Typ `Token` wird ein neues Attribut `disfluencyTag` hinzugefügt und das entsprechende Etikett als Wert gesetzt. Zusätzlich wird für alle Reparandum-Reparans-Paare eine neue Kante vom Typ `repair` angelegt, die vom ersten Wort des Reparans auf das erste des Reparandums weist.

7.1.4 Evaluation des Agenten zur Erkennung von Disfluenzen

Die Evaluation des Agenten zur Erkennung von Disfluenzen wird anhand von Transkriptionen zu den Szenarien dreizehn und vierzehn evaluiert (siehe Abschnitt 5.5.2). Diese Transkriptionen weisen mehr Disfluenzen auf als der restliche Teil des Korpus; sie wurden zudem während der Entwicklung des Agenten nicht betrachtet[11].

In Tabelle 7.5 sind die wesentlichen Charakteristika des verwendeten Evaluationsdatensatzes aufgeführt. Die 29 Aufnahmen enthalten in Summe 71 gefüllte Pausen, 48 Reparanda und 52 Reparans. Die letzten beiden bilden zusammen 27 Reparaturen (zwölf in Szenario dreizehn und fünfzehn in Szenario vierzehn); siebzehn Aufnahmen enthalten jedoch keinerlei Reparaturen. Die Sprecher verwendeten zudem keinerlei expliziten Bearbeitungsbegriffe oder Diskursmarker. Dieser Umstand kann möglicherweise darauf zurückgeführt werden, dass es sich bei den Sprechern nicht um Muttersprachler handelt (siehe Abschnitt 5.5.2.2).

Für die Transkriptionen der Aufnahmen wurde händisch ein Goldstandard mit den in Tabelle 7.1 aufgeführten Etiketten erstellt. Um das Vergleichsergebnis des Agenten zu erhalten, wurde das vollständige Vorverarbeitungsfließband ausgeführt (siehe Kapitel 6). Anschließend wurde nur der

[11] Während der Entwicklung wurden nur die Transkriptionen der ersten zwölf Szenarien verwendet; diese enthalten allerdings in Summe nur 60 Disfluenzen, zumeist gefüllte Pausen.

Tabelle 7.6: Evaluationsergebnisse für den Agenten zur Erkennung von Disfluenzen: Aufgeführt sind die Ergebnisse je Klasse für die beiden Disfluenzstrukturen Reparandum und Reparans, sowohl in IOB-Darstellung (*B** und *I**) als auch ohne die Unterscheidung zwischen Beginn- und Folgewörtern (**RM* und **RS*).

Klasse	Präzision			Ausbeute			F_1		
	Std.	Opt.	Δ	Std.	Opt.	Δ	Std.	Opt.	Δ
B-RM	0,750	0,690	− 0,060	0,500	0,667	+ 0,167	0,600	0,678	+ 0,078
I-RM	0,643	0,688	+ 0,045	0,500	0,611	+ 0,111	0,563	0,647	+ 0,084
B-RS	0,737	0,870	+ 0,133	0,467	0,667	+ 0,200	0,571	0,755	+ 0,184
I-RS	0,800	0,846	+ 0,046	0,381	0,524	+ 0,143	0,516	0,647	+ 0,131
*RM	0,765	0,733	− 0,032	0,542	0,688	+ 0,146	0,634	0,710	+ 0,076
*RS	0,759	0,861	+ 0,102	0,431	0,608	+ 0,177	0,550	0,713	+ 0,163

Agent zur Disfluenz-Erkennung einmalig ausgeführt[12]. Es wurden zwei verschiedene Konfigurationen getestet: die Standard-Version des Agenten ohne Nachverarbeitung (*Std.*) und mit Nachverarbeitung (*Opt.*). Die Ergebnisse werden anhand der Metriken Präzision, Ausbeute und F_1 bemessen (siehe Tabelle 7.6).

Die Standardkonfiguration ohne Nachverarbeitung erzielt deutlich schlechtere Ergebnisse als auf dem Testdatensatz des *Switchboard*-Korpus. Sowohl die Präzision, aber besonders die Ausbeute sind unzureichend, sodass die erreichten Werte für das F_1-Maß je nach Klasse nur zwischen 0,516 und 0,600 liegen. Dieses Ergebnis kann höchstwahrscheinlich auf die Unterschiede zwischen Trainings- und Evaluationsdatensatz zurückgeführt werden. Während es sich beim *Switchboard*-Korpus um Konversationen zwischen zwei Sprechern handelt, bei denen die einzelnen Äußerungen recht kurz sind, sind die Äußerungen im ProNat-Korpus lange Beschreibungsfolgen, die nur von einem Sprecher getätigt werden. Hinzu kommt, dass alle Sprecher des Evaluationsdatensatzes keine Muttersprachler sind, der Klassifikator aber auf Äußerungen von Muttersprachlern trainiert wurde. Gerade bei intuitiven Sprechsituationen, wie gefüllten Pausen oder Reparaturen, zeigen sich deutliche Unterschiede zwischen Muttersprachlern und Nicht-Muttersprachlern. Letztere verfallen bei Reparaturen in Konstrukte ihrer Muttersprache oder verwenden andere Verzögerungslaute.

Werden die Klassifikationsergebnisse nachverarbeitet (*Opt.*), zeigt sich, dass sich die Ausbeute in allen Fällen und die Präzision mit Ausnahme der Klasse *B-RM* (und damit auch **RM*) verbessert. Vor allem die Ausbeute steigt deutlich, teilweise um bis zu 30% (Klasse *B-RS*). Mit Werten von im Mittel 0,710 (Reparandum) bzw. 0,713 (Reparans) für das F_1-Maß werden mithilfe der Nachverarbeitung akzeptable Ergebnisse erzielt. Zukünftig sollte aber insbesondere die Präzision der Erkennung von Disfluenzen in Äußerungen, wie sie im Kontext von ProNat üblich sind, verbessert werden, indem zum Beispiel das trainierte Modell mithilfe des ProNat-Korpus verfeinert wird.

[12] Der Agent benötigt keine Informationen anderer Agenten und verfeinert die eigenen Analysen auch nicht weiter; eine einmalige Ausführung ist daher ausreichend.

7.2 Disambiguierung von Wortbedeutungen

Die inhärente Mehrdeutigkeit natürlicher Sprachen ist der wesentlichste Unterschied zu formalen Sprachen (siehe Abschnitt 2.3.8). Die einfachste Form semantischer Ambiguität ist die Mehrdeutigkeit von Bedeutungen auf Wortebene. Jedes Wort kann je nach Kontext eine andere Bedeutung tragen. Wie bereits in Abschnitt 2.3.8 dargelegt, ist die Disambiguierung von Wortbedeutungen eines der ältesten Probleme der Computerlinguistik. Da die Bestimmung der Bedeutung der Einzelwörter einer natürlichsprachlichen Sequenz die Grundlage vieler weiterer Analysen ist, ist die Korrektheit von besonderer Bedeutung.

Auch im Kontext von ProNat stellt die Auflösung von Mehrdeutigkeiten eine wesentliche Herausforderung dar. Die eindeutige Interpretation einzelner Wörter erleichtert später die Erkennung der in einer Aussage behandelten Themen und den Aufbau des sprachlichen situativen Kontextes (siehe Abschnitt 7.3 und 7.4). Ebenso können nur für disambiguierte Wörter passende Synonyme ermittelt werden (siehe Abschnitt 7.4). Synonyme wiederum erhöhen die Flexibilität der Quelltexterzeugung (siehe Abschnitt 7.8 und 8.1).

Für ProNat wurden zwei PARSE-Agenten zur Auflösung von Wortmehrdeutigkeiten implementiert. Der erste Agent dient als Adapter zur Web-API von *Babelfy*, das *WordNet* zur Disambiguierung verwendet [MRN14], der zweite implementiert eine angepasste Variante des von Mihalcea und Csomai vorgestellten Verfahrens zur Disambiguierung basierend auf *Wikipedia* [Mih07; MC07]. Diese Mehrfachlösung für dasselbe Problem wurde umgesetzt, da beide Ansätze Vor- und Nachteile mit sich bringen. *Babelfy* verwendet als diskretes Bedeutungsinventar *Synsets* aus *WordNet* (siehe Abschnitt 2.3.15.1). Das bedeutet, jedem zu disambiguierenden Wort wird ein *Synset* aus *WordNet* zugewiesen, welches die Bedeutung im aktuellen Kontext beschreibt. Die Verwendung von *WordNet-Synsets* ermöglicht die Disambiguierung von Nomen, Verben, Adjektiven und Adverbien, wohingegen sich eine Disambiguierung auf Grundlage von *Wikipedia* nur für Nomen eignet. Zusätzlich basieren die meisten Verfahren zur Auflösung von Wortmehrdeutigkeiten auf *WordNet*; damit lassen sich die für ProNat erzielten Ergebnisse besser mit anderen Ansätzen vergleichen. Hinsichtlich der Projekt-Organisation von ProNat bedeutet dies zusätzlich, dass *Babelfy* gegebenenfalls zu einem späteren Zeitpunkt durch einen anderen Ansatz ersetzt werden kann, ohne dass andere Agenten, die Wortbedeutungen verwenden, davon beeinflusst werden.

Ein spezialisierter Ansatz unter Verwendung von *Wikipedia* hat trotz der mannigfaltigen Möglichkeiten von *Babelfy* seine Daseinsberechtigung. Zunächst bietet *Wikipedia*, verwendet man die einzelnen Artikel(-Bezeichnungen) als Bedeutungsinventar, einen deutlich größeren Umfang, wenn auch nur für Nomen. Außerdem wird die *Wikipedia* ständig überprüft und erweitert; somit steigern sich sowohl Qualität als auch Quantität des Bedeutungsinventars stetig. Da Disambiguierung heutzutage meist durch maschinelles Lernen gelöst wird, ist die Qualität des gelernten Modells abhängig von der zur Verfügung stehenden Datengrundlage. Üblicherweise werden hierfür Korpora händisch annotiert; in diesem Fall bedeutet das, ein oder mehrere menschliche Annotatoren markieren einzelne Wörter mit der jeweils passenden Bedeutung. Dies ist aufgrund der Größe des Bedeutungsinventars und häufig auftretender verwandter Bedeutungen eine aufwendige und fehleranfällige Aufgabe. Aus diesem Grund werden die meisten Verfahren nur auf vergleichsweise kleinen Datensätzen trainiert. Nutzt

man die Artikelbezeichnungen aus *Wikipedia* als Bedeutungsinventar und Verknüpfungen (engl. *link*) als Instanzen der jeweiligen Bedeutungen in einem sprachlichen Kontext, umgeht man dieses Problem. So kann die gesamte *Wikipedia* als Trainingsdatensatz für einen Klassifikator genutzt werden.

7.2.1 Implementierung des Babelfy-Agenten zur Disambiguierung

Wie zuvor erwähnt, dient der Agent als Adapter für die von *Babelfy* bereitgestellte Web-API. *Babelfy* verwendet *WordNet-Synsets* zur Beschreibung von Wortbedeutung; das bedeutet, die Menge aller *Synsets* bilden das Bedeutungsinventar bzw. den Etikettensatz. *Babelfy* basiert auf *BabelNet*, einem semantischen Graphen, der aus einer automatischen Verschmelzung von *WordNet* und *Wikipedia* entstanden ist. Konzepte, das heißt *Synsets* und *Wikipedia*-Artikel, bilden die Knoten des Graphen, die Kanten werden anhand der Relationen zwischen den Konzepten erzeugt (beispielsweise Oberbegriff-Beziehungen oder Teil-Ganzes-Beziehungen).

Darüber hinaus sind die Konzepte mit Konzepten anderer lexikalischer Datenbanken verknüpft, unter anderem *FrameNet* und *VerbNet* (siehe Abschnitt 2.3.15). Natürlichsprachliche Sequenzen können mithilfe des *BabelNet*-Graphen wie folgt disambiguiert werden: Zunächst werden innerhalb einer Sequenz zusammenhänge Phrasen extrahiert; diese entsprechen im Wesentlichen den in Abschnitt 6.2 beschriebenen *Chunks*. Die Phrasen werden anschießend auf Konzepte des *BabelNet*-Graphen abgebildet, wobei Mehrfachabbildungen möglich sind. Das bedeutet, jede Phrase wird zunächst mit einer (potenziell großen) Menge von Bedeutungskandidaten verknüpft. Die Bedeutungskandidaten für alle Phrasen werden anschließend im Graph verbunden, indem die kürzesten Pfade zwischen jeweils zwei Konzepten bestimmt werden. Zuletzt wird heuristisch der dichteste Graph approximiert, wobei zu jeder Phrase nur ein Bedeutungskandidat enthalten sein darf. Dieser Graph enthält die wahrscheinlichsten Bedeutungen für die Phrasen der natürlichsprachlichen Sequenz; den Bedeutungen wird zudem jeweils eine Konfidenz zugeordnet.

Der `ProNat`-Agent konvertiert die im `ProNat`-Graphen vorhandenen sprachlichen Informationen zunächst in das von *Babelfy* akzeptierte Format; das Ergebnis (ein *Wordnet-Synset* samt Konfidenz) wird wiederum in den `ProNat`-Graphen eingepflegt. Der Agent entnimmt dem Graphen alle Knoten des Typs `Token`, was den einzelnen Wörtern der natürlichsprachlichen Eingabe entspricht. Anschließend wird die Tokensequenz in das Datenformat von *Babelfy* überführt und an die Web-API übermittelt. Die Rückgabe wird entgegengenommen und anhand der von *Babelfy* mitgelieferten Konfidenz entschieden, ob das Ergebnis in die Knoten zurückgeschrieben wird oder nicht[13]. Im Knoten wird die Kennung (ID) und die Glosse des Synsets gespeichert. Zusätzlich wird die Konfidenz der *Babelfy*-Antwort sowie die gesamte Antwort als Objekt hinterlegt.

[13] Der Schwellenwert lässt sich konfigurieren und ist mit *0,3* voreingestellt.

7.2.2 Implementierung des Agenten für Disambiguierung basierend auf Wikipedia

Die Implementierung des Agenten ist eine Adaption des von Mihalcea und Csomai vorstellten Verfahrens [Wei+19]. Die zugrundeliegende Idee ist wie folgt. Treten Begriffe in *Wikipedia*-Artikeln das erste Mal auf, erhalten sie vom Verfasser eine Verknüpfung zum entsprechenden erklärenden Artikel in der *Wikipedia*; das bedeutet, die Bedeutung eines Begriffs im sprachlichen Kontext wird durch das Hinzufügen der Verknüpfung zu einem erklärenden Artikel eindeutig beschrieben. Die einzelnen Verknüpfungen bilden somit die Instanzen eines Bedeutungsinventars, das aus allen *Wikipedia*-Artikeln besteht. Da die Verknüpfungen manuell von den Erstellern (oder Bearbeitern) des Artikels eingefügt werden, können diese als korrekt angesehen werden[14].

Die Bedeutungsinstanzen und der sprachliche Kontext können anschließend genutzt werden, um ein beliebiges überwachtes Lernverfahren einzusetzen; verwendet wurde, wie im Original-Ansatz, ein Naïve-Bayes-Klassifikator. Dieser zeichnet sich dadurch aus, dass er gut mit einer Vielzahl unterschiedlich geartete oder ähnlich relevanter Eingabe-Eigenschaften (engl. *input features*) umgehen und auch auf großen Datenmengen eingesetzt werden kann – sowohl hinsichtlich des Trainingsaufwands als auch etwaiger Überanpassungseffekte [MN98]. Als Eingabe-Eigenschaften werden, wie auch bei Mihalcea und Csomai, jeweils die drei Wörter vor und nach dem zu disambiguierenden Wort samt ihrer Wortart verwendet[15], sowie je das erste Nomen und das erste Verb links und rechts des Wortes. Zusätzlich wird das eigentlich zu disambiguierende Wort verwendet. Dieses wird im Gegensatz zum Original-Ansatz allerdings zehnfach gewichtet. Mihalcea und Csomai verwenden in ihrem Ansatz zusätzlich sogenannte Kontext-Wörter; diese sind lediglich die am häufigsten auftretenden Wörter in einem Paragraphen. Da in gesprochener Sprache keine Paragraphen vorhanden sind, wurde diese Art von Eingabe-Eigenschaft ausgelassen. Abbildung 7.1 zeigt einen Ausschnitt aus der *Wikipedia*, inklusive dem zu disambiguierenden Wort, dem Zieletikett, den extrahierten Eigenschaften und dem daraus resultierenden Vektor.

Um den Klassifikator zu trainieren, wurde ein vollständiger Auszug der englischsprachigen *Wikipedia* (Stand August 2017) verwendet. Der Auszug wurde wie folgt vorverarbeitet. Zunächst wurden alle Begriffserklärungsseiten entfernt. Diese erfüllen zwar in der Theorie genau die Aufgabe der Disambiguierung; allerdings werden diese nicht konsequent eingesetzt. Häufig fehlen Begriffserklärungsseiten, obwohl mehrere Artikel für den gleichen Begriff existieren, oder Verknüpfungen in Artikeln werden direkt zu den Zielartikeln vorgenommen, ohne auf die entsprechende Begriffserklärungsseite zu verweisen. Außerdem enthalten Begriffserklärungsseiten ausschließlich die jeweiligen Begriffe und eine Kurzbeschreibung. Das bedeutet, dass der sprachliche Kontext kaum nützlich ist, um einen Klassifikator zu trainieren. Aus demselben Grund wurden auch Listen und Info-Boxen entfernt. Listen enthalten zudem nur selten Verknüpfungen, weshalb sie für den Ansatz ohnehin ungeeignet sind; dasselbe gilt auch für Zitate. Zuletzt werden alle *Wikipedia*-Artikel (und zugehörige Verknüpfungen) entfernt, die einen Eigennamen (engl. *Named Entity*) beschreiben

[14] Zumindest ist anzunehmen, dass durch Domänenexperten hinzugefügte Verknüpfungen eher korrekt sind, als beispielsweise das Annotieren von Synsets durch beliebige Annotatoren.

[15] Stoppwörter werden hierbei ignoriert.

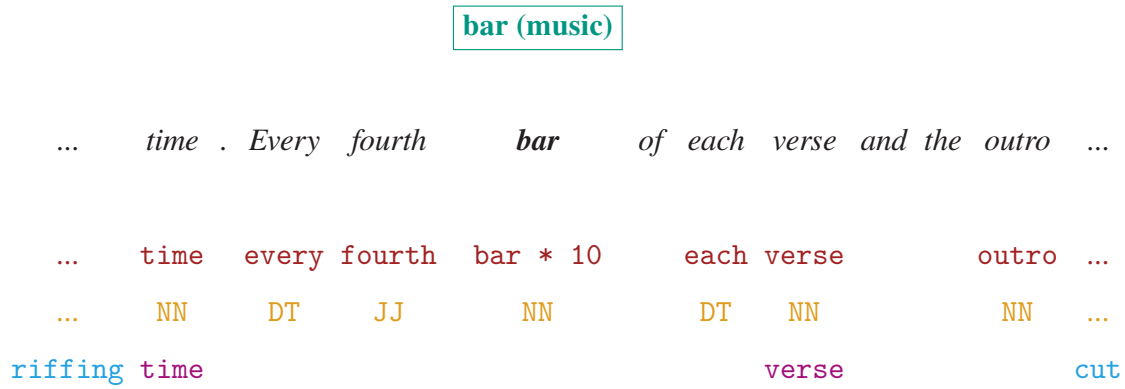


Abbildung 7.1: Beispielhafter Wortvektor zum Wort *bar*: Die Abbildung zeigt einen Auszug aus Wikipedia (kursiv), anhand dessen der Wortvektor erzeugt wird. Die Umgebungsmerkmale (in Schreibmaschinenschrift) setzen sich zusammen aus dem Wort *bar* an sich mit zehnfacher Gewichtung, den jeweils drei Wörtern links und rechts des Wortes (rot hervorgehoben) mit Ausnahme von Stopwörtern und Satzzeichen, den Wortarten der Umgebungswörter (gelb), sowie jeweils den ersten Nomen (lila) und Verben (blau) links und rechts des Wortes. Im Beispiel befinden sich die nächsten Verben außerhalb des Auszugs und sind daher am Rand abgedruckt. Das Wortbedeutungs-Etikett zum Wort *bar* ist fett gedruckt und in einem Kasten dargestellt.

(siehe Abschnitt 2.3.4.5). Eigennamen sind für gewöhnlich eindeutig und müssen daher nicht disambiguiert werden. Eine Hinzunahme dieser Artikel würde zu einer deutlichen Vergrößerung des Bedeutungsinventars führen, was wiederum einen erhöhtem Trainigsaufwand und potenziell mehr Fehlklassifikationen nach sich zieht.

Nach der Vorverarbeitung verbleiben 5.188.470 Trainingsinstanzen (Verknüpfungen zu *Wikipedia*-Artikeln). Davon sind 283.173 unterschiedlich. Die Menge dieser einzigartigen Bedeutungen bilden das Bedeutungsinventar, welches somit eine Mächtigkeit von 283.173 besitzt. 136.964 Instanzen treten nur genau einmal als Verknüpfung in einem Artikel auf. Diese einmalig auftretenden Instanzen machen damit 2,64% aller Instanzen und 46,37% der Bedeutungen aus. Insbesondere letztere Beobachtung ist problematisch für überwachte Lernverfahren; schließlich bedeutet sie, dass fast die Hälfte der Klassen des Bedeutungsinventars nur mit einer Instanz im Trainingskorpus vertreten sind. Der Klassifikator kann also nicht über unterschiedliche Instanzen (und deren unterschiedliche Kontexte) generalisieren. Zusätzlich muss bei der Evaluation davon ausgegangen werden, dass bei einer Aufteilung des Korpus in Trainings- und Testmenge die Testmenge zuvor unbekannte Instanzen enthält, die folglich nicht klassifiziert werden können.

Mit den Instanzen kann ein Klassifikator, in diesem Fall Naïve-Bayes, trainiert werden. Im Kontext von ProNat passiert dies einmalig offline mithilfe der Rahmenarchitektur *Weka* [HDW94; Hal+09]. Der zugehörige PARSE-Agent lädt ein so erzeugtes Modell. Anschließend wird die Eingabe ausgelesen – wiederum anhand von Knoten des Typs `Token`. Die vollständige Eingabe wird dem Klassifikator übergeben und die wahrscheinlichste Bedeutung anschließend in einem neuen Attribut im jeweiligen Knoten hinterlegt. Klassifiziert werden jedoch ausschließlich Nomen, die keine Eigennamen darstellen; beide Informationen werden während der Vorverarbeitung erzeugt (siehe Abschnitt 6.2 und Abschnitt 6.3).

7.2.3 Evaluation des Agenten für Disambiguierung basierend auf Wikipedia

Der Agent zur Disambiguierung basierend auf *Wikipedia* bzw. der verwendete Klassifikator wurde auf zwei unterschiedliche Arten evaluiert. Zunächst wurde der Ansatz auf dem *Wikipedia*-Auszug evaluiert, indem ein Großteil zum Training und der Rest zum Testen verwendet wurde. Um die Anwendbarkeit für ProNat zu prüfen, wurde zusätzlich auf einem Teil des in Abschnitt 5.5.2 vorgestellten Korpus evaluiert.

Die Evaluation auf dem *Wikipedia*-Auszug wurde als angepasste Zehnfach-Kreuzvalidierung durchgeführt. Bei der üblichen Zehnfach-Kreuzvalidierung wird der gesamte Datenbestand in zehn gleiche Teile geteilt und anschließend auf neun dieser Teile trainiert und auf dem zehnten getestet. Dieses Vorgehen wird zehnmal wiederholt und der Testteil getauscht, bis jeder der zehn Teile einmal als Testmenge verwendet wurde. Da eine Aufteilung in zehn gleichgroße Teile bei fast 5,2 Millionen Instanzen je Durchlauf 520000 Testinstanzen und in Summe 5,2 Millionen Testinstanzen erzeugen würde, was wiederum zu erheblichen Aufwand bei der Bestimmung der korrekten Klassifizierungen führt, wurde eine randomisierte Variante der Zehnfach-Kreuzvalidierung verwendet. Hierbei werden für jeden Durchlauf 10000 Testinstanzen zufällig gezogen und der Rest für die Trainingsmenge verwendet. Die Ergebnisse werden wie üblich über die zehn Durchläufe gemittelt.

Da die zu klassifizierenden Instanzen bekannt sind, kann je klassifizierter Testinstanz ausschließlich zwischen *richtig* klassifiziert und *falsch* klassifiziert unterschieden werden. Eine Unterscheidung nach *falsch positiven* und *negativen* sowie nach *richtig positiven* und *negativen* Klassifizierungen ist nicht sinnvoll. Mihalcea und Csomai unterscheiden diese trotzdem, indem sie ihren Klassifikator instrumentieren und keine Klassifizierungen für Wörter mit (zum Trainingszeitpunkt) unbekannter Oberflächenform zulassen. Dies betrifft vor allem die oben diskutierten nur einmalig auftretenden Instanzen, die Teil der Testmenge sind. Durch den Ausschluss dieser Instanzen wird das Klassifikationsergebnis verbessert, da diese immer falsch klassifiziert werden würden. Gleichzeitig kann auch wieder zwischen *richtig positiven*, *richtig negativen* und *falsch negativen* Klassifikationen unterschieden werden, weshalb die Verwendung von Präzision, Ausbeute und F_1 in der Original-Veröffentlichung sinnvoll ist. Allerdings erfordert diese Art der Auszählung manuellen Aufwand. Dies könnte ein Grund dafür sein, warum Mihalcea und Csomai ihren Ansatz nur auf 85 zufällig gezogenen *Wikipedia*-Artikeln evaluieren. Dieser Auszug enthielt 7286 Instanzen. Der hier vorgestellte Ansatz wurde hingegen auf zehnmal 10000 Instanzen getestet. Außerdem wird nur die Genauigkeit der Klassifikationsergebnisse angegeben, weshalb ein Vergleich mit der Original-Veröffentlichung nur bedingt möglich ist.

Tabelle 7.7 fasst die Ergebnisse der randomisierten Zehnfach-Kreuzvalidierung zusammen. Die Werte schwanken über die unterschiedlichen Durchläufe nur geringfügig (min: 0,789, max: 0,807). Insgesamt werden knapp 80% aller Testinstanzen korrekt klassifiziert. Dies entspricht ungefähr der Qualität aktueller Ansätze [RCN17]. Mihalcea und Csomai geben eine Ausbeute von 0,831 an[16].

[16] Die Ausbeute, wie von Mihalcea und Csomai beschrieben, entspricht am ehesten der hier verwendeten Genauigkeit [MC07].

Tabelle 7.7: Evaluationsergebnisse für den Agenten zur Disambiguierung auf dem *Wikipedia*-Datensatz: Als Datengrundlage wurde der *Wikipedia*-Auszug verwendet und zehn (Test-)Läufe auf je 10000 Testinstanzen durchgeführt.

Lauf	1	2	3	4	5	6	7	8	9	10	ø
Genauigkeit	0,802	0,802	0,797	0,807	0,794	0,797	0,807	0,800	0,795	0,789	0,799

Tabelle 7.8: Evaluationsergebnisse für den Agenten zur Disambiguierung auf dem ProNat-Datensatz.

Präzision	Ausbeute	F_1
0,894	0,876	0,885

Die hier vorgestellte Adaption erzielt somit ähnliche Ergebnisse wie in der Original-Veröffentlichung angegeben, obwohl der Testdatensatz deutlich größer ist und die Zählweise von Mihalcea und Csomai deutlich bessere Werte für die Ausbeute erzeugt.

Nachdem gezeigt werden konnte, dass der Klassifikator grundsätzlich funktioniert, wird nachfolgend die Anwendbarkeit für ProNat evaluiert. Hierfür wurde der Klassifikator auf dem vollständigen *Wikipedia*-Auszug trainiert und anschließend auf einem Teil des in Abschnitt 5.5.2 vorgestellten Korpus getestet. Die Testmenge umfasst Transkriptionen zu den ersten acht Szenarien, in Summe 168 Transkriptionen[17]. Für jede Transkription wurde eine Musterlösung erstellt; das bedeutet, dass jedem Nomen die passende Bedeutung aus dem Bedeutungsinventar (einem Artikel aus *Wikipedia*) zugewiesen wurde. Die Transkriptionen enthalten insgesamt 1060 Nomen und damit ebenso viele zu disambiguierende Instanzen.

Um die Performanz des Klassifikators im Kontext von ProNat bemessen zu können, wurde das vollständige Vorverarbeitungsfließband, wie in Kapitel 6 beschrieben mit Ausnahme der automatischen Spracherkennung, durchlaufen. Benötigt werden für die Klassifikation jedoch ausschließlich die Wortarten, die von der Stufe für seichte Sprachverarbeitung (siehe Abschnitt 6.2) emittiert werden, sowie die erkannten Eigennamen[18] (siehe Abschnitt 6.3). Anschließend wurde der Agent ausgeführt. Da dieser alle Nomen klassifiziert und die korrekte Erkennung der Wortarten von der Vorverarbeitung abhängt, ist die Menge der zu klassifizierenden Instanzen unbekannt. Aufgrund fehlerhaft klassifizierter Wortarten kann es somit zu fehlenden (*falsch negativ*) und überflüssigen Klassifikationen (*falsch positiv*) kommen. Zusätzlich erzeugen falsche Klassifikationen an einem (korrekt erkanntem) Nomen sowohl eine falsch positive als auch eine falsch negative Etikette.

Die Ergebnisse für die Messung auf dem ProNat-Datensatz sind in Tabelle 7.8 dargestellt. Die Ergebnisse erscheinen zunächst unerwartet. Eigentlich wäre zu erwarten gewesen, dass die Qualität der Klassifikation abnimmt, da das Modell zum einen auf einer anderen Domäne (*Wikipedia*-Artikel) trainiert wurde und zum anderen die Klassifikationsaufgabe komplexer ist (zusätzliche Bestimmung der zu klassifizierenden Instanzen anhand der Wortart). Stattdessen attestiert die Messung dem Agenten sehr gute Werte für Präzision (0,894), Ausbeute (0,876) und F_1-Maß (0,885).

[17] Wenn für die Beschreibung eines Probanden zu einem Szenario mehrere Aufnahmen im Korpus existieren (z.B. durch Bitte um einen erneuten Versuch), so wurde für die hiesige Evaluation nur die beste Transkription verwendet. Ebenso wurden aus den Transkriptionen zu Szenario 8 (Dialog-Szenario) nur die Ersteingaben der Probanden verwendet.

[18] Der einzige in den Transkriptionen auftretende Eigenname ist „Armar“, der in allen Fällen erkannt wurde.

Lediglich 21 Nomen konnten nicht klassifiziert werden, weil die Wortarterkennung fehlerhaft war. Die Eigennamenerkennung funktionierte in allen Fällen korrekt und führte somit zu keinen fehlenden Instanzen. Weder die Wortarterkennung, noch die Eigennamenerkennung haben darüber hinaus falsch positive Klassifikationen erzeugt. An diesen Ergebnissen lässt sich erkennen, dass etwaige Fehler aus der Vorverarbeitung nur geringen Einfluss auf die Ergebnisse des Agenten für Disambiguierung basierend auf *Wikipedia* haben.

Ein Großteil der Fehlklassifikationen kann auf fehlende Bedeutungen im Bedeutungsinventar zurückgeführt werden. Beispielsweise gibt es keinen *Wikipedia*-Artikel zum Konzept *front* (deutsch: *Vorderseite*) im Sinne von *the side that is forward or prominent* (deutsch: *die zugewandte oder markante Seite*). Der Klassifikator vergibt in diesen Fällen eine falsche Etikette. Abgesehen von diesen seltenen Fällen funktioniert die Disambiguierung basierend auf *Wikipedia* auf dem ProNat-Datensatz sehr gut. Die starken sprachlichen Unterschiede (deskriptive Texte in *Wikipedia* gegenüber potenziell grammatikalisch fehlerhaften Äußerungen im Imperativ im ProNat-Datensatz) scheinen die Klassifikation nur geringfügig negativ zu beeinflussen. Vielmehr können die vergleichsweise einfachen natürlichsprachlichen Äußerungen, die nur weit verbreitete Konzepte wie *Kühlschrank*, *Tisch* oder *Glas*, die zudem in ihrer natürlichen Umgebung (einer Küche) auftreten, sehr gut disambiguiert werden.

7.3 Modellierung und Etikettierung von Diskurs-Themen

Das Verständnis der behandelten Themen in einem Diskurs zwischen Mensch und Maschine ist ein wichtiger Baustein, um die Intention einer Aussage interpretieren zu können. Themen-Extraktion, -Modellierung und -Etikettierung sind seit jeher Felder der Computerlinguistik, denen große Beachtung geschenkt wird (siehe Abschnitt 2.3.9). Die meisten Ansätze betrachten jedoch geschriebene Dokumente oder Dokumentmengen, da Themen-Modellierung hauptsächlich zur Verarbeitung großer, geschriebener Datensätze eingesetzt wird. Dort können beispielsweise automatische Zusammenfassungen generiert oder der Datensatz strukturiert werden. Für gesprochene Sprache sind diese Themenfelder kaum erforscht, dabei ist gerade hier das Verständnis des Themenbereichs von besonderer Bedeutung. Mithilfe eines Themen-Modells könnten zum Beispiel automatisch Wortfehler automatischer Spracherkenner korrigiert werden oder ein Dialog-System könnte gezieltere Rückfragen an den Nutzer stellen.

Im Kontext von ProNat können Themen-Etiketten vor allem genutzt werden, um externe Datenquellen präziser nach Informationen zu durchsuchen. So können erkannte Themenbereiche beispielsweise auf Mikro-Theorien in *Cyc* abgebildet werden, um Anfragen auf diesen Teilraum von Entitäten und Regeln zu beschränken. Eine konkrete Anwendung für die Themen-Etikettierung im Rahmen von ProNat wird mit der Ontologie-Auswahl in Abschnitt 7.9 vorgestellt.

7.3.1 Implementierung des Agenten zur Modellierung und Etikettierung von Diskurs-Themen

Für ProNat wurde ein PARSE-Agent entwickelt, der Themen implizit modelliert und anschließend mit Artikeln aus Wikipedia (als Stellvertreter für ein Themen-Konzept) etikettiert [Wei+19; Wei+20a]. Neben den bekannten Herausforderungen bei der Verarbeitung gesprochener Sprache, wie Wortfehlern, grammatikalisch falsche Formulierungen, Disfluenzen, etc., ergibt sich für die Modellierung von Themen im Kontext von ProNat eine weitere: die natürlichsprachlichen Äußerungen sind vergleichsweise kurz. Gängige Verfahren zur Modellierung von Themen setzen dagegen lange Dokumente bzw. Dokumentmengen voraus.

Die meisten modernen Ansätze stützen sich dabei auf die von Blei et al. entwickelte *Latent Dirichlet Allocation* (kurz *LDA*) [BNJ03; MSZ07; Mag+09; Ble12; Hul+13; Hin+13]. *LDA* ist ein generatives probabilistisches Modell für Bestände diskreter Datenpunkte wie Textdokumentsammlungen. Angewendet auf ebendiese wird ein Themen-Modell anhand von Wort-Vorkommen über alle Dokumente der Sammlung bestimmt. Das Verfahren erzeugt daraus eine feste Menge an Themen, die aus Wörtern bestehen, welche häufig zusammen auftreten[19]. Um *LDA* sinnvoll verwenden zu können, sind dementsprechend längere Dokumente als die im Kontext von ProNat üblichen Äußerungen nötig. Zusätzlich soll die Anzahl an Themen je Äußerung variabel gestaltet werden, was mit *LDA* ebenfalls nicht möglich ist. Ansätze, die auf gesprochene Sprache spezialisiert sind, verwenden meist phonetische Informationen, um ein Themen-Modell zu erzeugen, vergeben aber keine Etiketten [Cer09; Haz+11; Siu+14].

Der vorgestellte Ansatz modelliert die Themen implizit über Graphzentralitätsmetriken auf einem Thememgraphen, der aus *DBpedia* extrahiert wurde (siehe Abschnitt 2.3.15.5). Dieses Vorgehen hat folgende Vorteile: Durch die Verwendung der Graphzentralität kann eine variable Anzahl an Themen auch für kurze natürlichsprachliche Äußerungen erzeugt werden. Zusätzlich können die als zentral identifizierten Konzepte aus *DBpedia* als Etiketten für die modellierten Themen verwendet werden.

Zur impliziten Modellierung der Themen wird zunächst davon ausgegangen, dass alle Nomen in einer Äußerung miteinander in Verbindung stehen[20]. Ausgehend von den Nomen werden sogenannte *Bedeutungsgraphen* aufgebaut. Dazu wird zunächst für jedes Nomen seine Bedeutung im sprachlichen Kontext (in Form eines zugehörigen *Wikipedia*-Artikels) bestimmt (siehe Abschnitt 7.2); diese Wortbedeutung wird als *initiale Bedeutung* des Bedeutungsgraphen bezeichnet. Anschließend wird *DBpedia* als Graph aufgefasst, wobei die Knoten den Konzepten bzw. Artikeln und die Kanten den Relationen entsprechen. Aus diesem *DBpedia*-Graphen werden anschließend Teilgraphen extrahiert, indem ausgehend von der initialen Bedeutung Relationen bis zu einer Tiefe von zwei verfolgt werden[21]. Die verwendeten Relationen sind in Tabelle 7.9 aufgeführt.

[19] *LDA* kann dementsprechend als Wort-Clustering-Verfahren aufgefasst werden.

[20] Diese offensichtlich falsche Annahme wird später verworfen und das Analysergebnis entsprechend interpretiert.

[21] Um möglichst spezifische Themen zu erhalten, wäre es besser nur je eine Relation zu traversieren. Allerdings entstehen so sehr kleine Graphen und es wird anschließend schwierig, überhaupt gemeinsame Konzepte unter den Bedeutungen zu finden.

Tabelle 7.9: *DBpedia*-Relationen zum Aufbau des Bedeutungsgraphen.

Die Relation	verbindet ein Konzept mit
dcterms:subject	seiner Wikipedia-Kategorie
skos:broader	einem allgemeineren Konzept
skos:narrower	einem spezielleren Konzept
purlg:hypernym	einem übergeordneten Konzept (Hyperonym)
purlg:meronym	einem Konzept, das einen zugehörigen Teil ausmacht (Meronym)
purlg:synonym	einem synonymen Konzept
rdfs:type	seiner zugehörigen DBpedia-Entität
rdfs:subClassOf	seiner in DBpedia definierten Unterklasse
rdfs:seeAlso	einem verwandten Konzept

Die einzelnen Bedeutungsgraphen werden anschließend zu einem sogenannten Themengraph vereinigt, indem Knoten, die das gleiche Konzept repräsentieren, vereinigt werden. Der Themengraph kann aus nicht verbundene Teilgraphen bestehen. Statt wie in verwandten Arbeiten vorgeschlagen, nur den größten Teilgraph weiterzuverarbeiten [Hul+13], wird diese Eigenschaft genutzt, um unterschiedliche Themen-Bereiche abzudecken. Hierzu wird davon ausgegangen, dass jeder Teilgraph ein oder mehrere Themen der ursprünglichen Äußerung beinhaltet. Gemeinsam mit der im Folgenden beschriebenen Bestimmung von Themen-Etiketten werden damit die Themen impliziert modelliert.

Die Auswahl der Etiketten hängt von der Menge *n* solcher Etiketten ab, die pro natürlichsprachlicher Äußerung erzeugt werden sollen. Dabei muss folgende Abwägung getroffen werden: Einerseits werden Themen sehr allgemein oder semantisch unpassend, wenn mehr Themen ausgewählt werden. Andererseits führt die Erzeugung von nur wenigen Themen-Etiketten dazu, dass gegebenenfalls Bedeutungen (von Nomen) der natürlichsprachlichen Äußerung durch kein Thema repräsentiert werden. Aus diesem Grund kann der Agent unterschiedlich konfiguriert werden, entweder mit einer festen Anzahl an zu erzeugenden Themen-Etiketten oder mit einem Vielfachen der initialen Bedeutungsknoten. Die Standard-Konfiguration ist so gewählt, dass doppelt so viele Themen-Etiketten erzeugt werden wie es (unterschiedliche) initiale Bedeutungsknoten im Themengraph gibt. Dieser Wert wurde empirisch ermittelt, wobei der Fokus auf die Maximierung der Abdeckung der Themen-Bereiche gelegt wurde (siehe hierzu auch Abschnitt 7.3.2).

Zur Auswahl der Themen-Etiketten kann der Agent mit einer von zwei Strategien konfiguriert werden: *Höchste Konnektivität* oder *Maximale Abdeckung*. Die Strategie *Höchste Konnektivität* wählt Knoten aus, die eng mit den initialen Bedeutungsknoten verbunden sind. Hierzu wird gezählt, in wie vielen Bedeutungsgraphen (*nicht* Themengraphen) jeder Knoten auftritt; dieser Wert wird im Folgenden als *Konnektivität* bezeichnet. Da die Konnektivität in den meisten Fällen für eine große Menge von Knoten gleich ist, wird als zweites Kriterium ein Graphzentralitätswert für jeden Knoten bestimmt. Da viele Metriken zur Bestimmung der Graphzentralität nicht mit unverbundenen Teilgraphen umgehen können, wird eine Abwandlung des von Brin und Page entwickelten *PageRank*-Algorithmus mit Ausrichtungsfaktor als Bewertungsfunktion $\Omega(V_i)$ verwendet[BP98; CMM09; CM09]. Die Funktion $\Omega(V_i)$ bewertet alle Knoten V und Kanten E des Themengraphen $T = (V, E)$ anhand der

eingehenden Kanten $E_{ein}(V_i)$ je Knoten (V_i) sowie der Bewertung $\Omega(V_j)$ und ausgehenden Kanten $E_{aus}(V_j)$ der benachbarten Knoten:

$$\Omega(V_i) = (1-d) * B(V_i) + d * \sum_{j \in E_{ein}(V_i)} \frac{\Omega(V_j)}{|E_{aus}(V_j)|} \tag{7.1}$$

Die Konstante d ist der sogenannte *Dämpfungsfaktor* (engl. *damping factor*); dieser wird im Agenten (wie auch von Brin und Page empfohlen) auf den Wert 0.85 gesetzt, ist aber konfigurierbar [BP98]. Zur Berechnung des Ausrichtungsfaktor $B(V_i)$ wird die Menge der initialen Bedeutungsknoten Γ_{init} verwendet. Diese besteht aus allen Knoten, die die initialen Bedeutungen der Nomen der natürlichsprachlichen Äußerung beschreiben (siehe oben). Der Ausrichtungsfaktor $B(V_i)$ berechnet sich wie folgt:

$$B(V_i) = \begin{cases} 0 & , V_i \notin \Gamma_{init} \\ \dfrac{1}{|\Gamma_{init}|} & , V_i \in \Gamma_{init} \end{cases} \tag{7.2}$$

Der Ausrichtungsfaktor sorgt dafür, dass Knoten umso stärker gewichtet werden, je näher sie den initialen Bedeutungsknoten des Thememgraphen sind. Die Strategie *Höchste Konnektivität* ordnet die Knoten anschließend gemäß ihrer Konnektivität und bei gleichen Konnektivitätswerten anhand der Bewertungsfunktion Ω. Aus dieser sortierten Menge werden die *n* besten als Themen-Etiketten ausgewählt.

Die Strategie *Höchste Konnektivität* hat allerdings den Nachteil, dass Themen, die in der natürlichsprachlichen Äußerung nur durch wenige Wörter repräsentiert werden, von anderen Themen dominiert werden, da sowohl die Konnektivität als auch Ω in diesem Fall geringe Werte annehmen. Dieser Nachteil ist die Motivation für die zweite Strategie: *Maximale Abdeckung*. Diese wählt das erste Thema nach dem gleichen Verfahren wie die Strategie *Höchste Konnektivität* aus. Die weiteren Themen werden jedoch iterativ wie folgt bestimmt: Solange noch nicht alle initialen Bedeutungsknoten durch ein Thema repräsentiert werden, wird als nächstes Thema dasjenige gewählt, welches die meisten der noch nicht repräsentierten Bedeutungsknoten repräsentiert. Bei Gleichstand wird wie zuvor Ω als zweites Auswahlkriterium verwendet. Sobald alle initialen Bedeutungsknoten abgedeckt sind, wird mit den höchstbewerteten Themen-Etiketten (gemäß der Strategie *Höchste Konnektivität*) aufgefüllt bis *n* erreicht ist.

Der Agent hinterlegt die Ergebnisse anschließend folgendermaßen im Graphen: Es wird ein neuer Knotentyp `topics` angelegt und für jede Eingabe ein Knoten diesen Typs erzeugt. Im Knoten werden die Themen-Etiketten mit zugehöriger Konfidenz als Attribute hinterlegt. Die Konfidenz entspricht dabei dem Ω-Wert des zugehörigen Knotens im Themengraph.

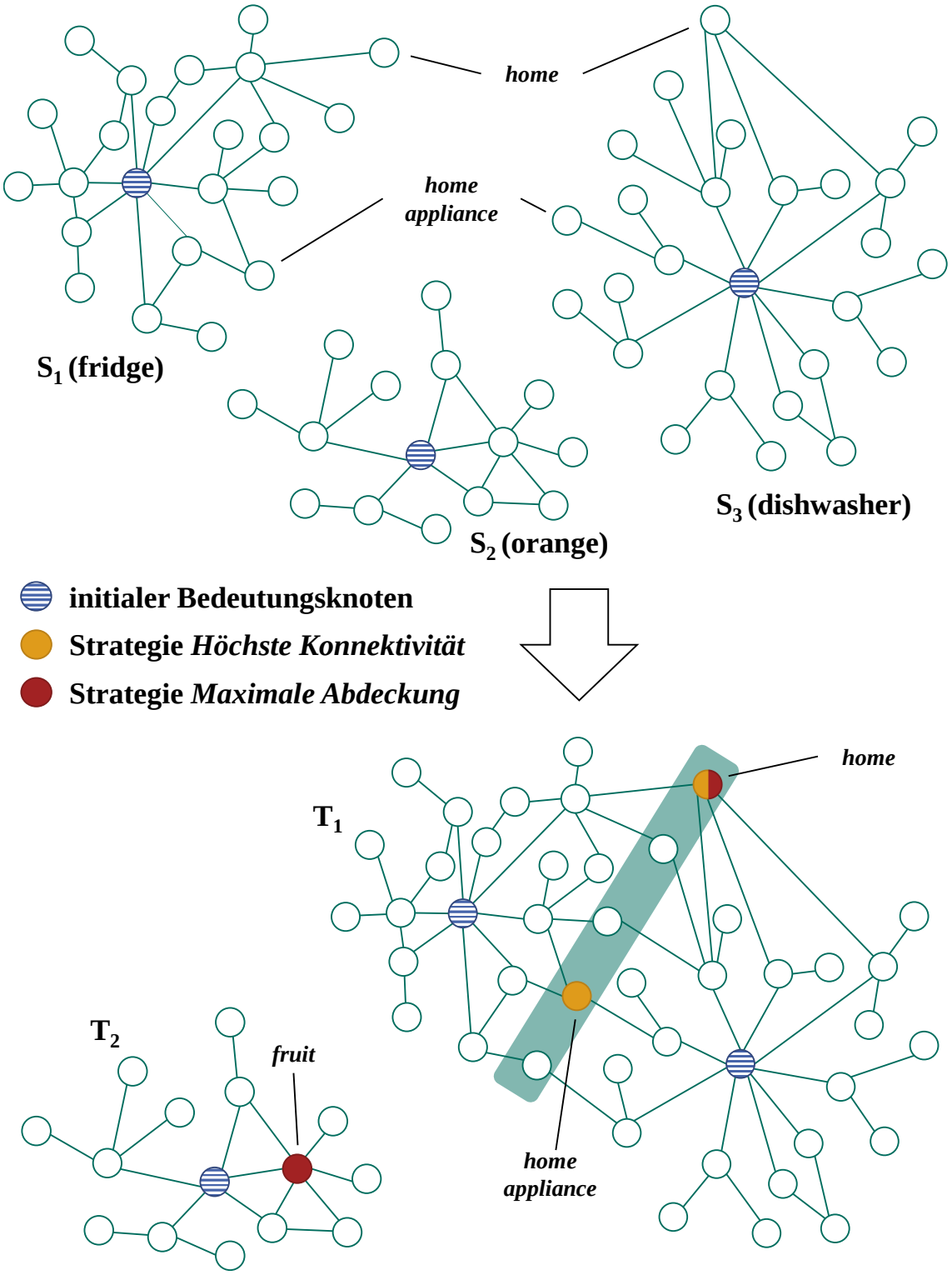


Abbildung 7.2: Exemplarische Bedeutungs- und Themengraphen: Dargestellt sind die Bedeutungs- (S_i) und Themen-(Teil-)Graphen (T_i) zur natürlichsprachlichen Sequenz „take the orange from the fridge and close the dishwasher afterwards“. Die initialen Bedeutungsgrahen sind blau-gestreift dargestellt. Knoten, die mittels der Strategie *Höchste Konnektivität* als Themen-Etiketten ausgewählt wurden, sind orange eingefärbt und Knoten, die anhand der Strategie *Maximale Abdeckung* ausgewählt wurden, sind rot eingefärbt. Die Menge der (Knoten-)Kandidaten mit gleicher Konnektivität in T_1 sind grün hinterlegt.

Beispiel:
Modellierung und Etikettierung von Diskurs-Themen

In diesem Beispiel soll die Erzeugung von Themen-Etiketten mithilfe der beiden Strategien *Höchste Konnektivität* und *Maximale Abdeckung* anhand einer natürlichsprachlichen Sequenz demonstriert werden. Einfachheitshalber werden im Beispiel nur zwei Themen-Etiketten erzeugt ($n = 2$). Die Beschreibung nimmt dabei Bezug auf Abbildung 7.2 in der die einzelnen Schritte bzw. Auswahlstrategien nachvollzogen werden können.

Natürlichsprachliche Sequenz

take the orange from the fridge and close the dishwasher

Vorgehen

Zu Beginn müssen die Bedeutungen aller Nomen disambiguiert werden (siehe Abschnitt 7.2); beispielsweise wird für *orange* die Bedeutung *Orange(fruit)* anstatt der ebenfalls möglichen Bedeutungen *Orange(color)* oder *Orange(word)* gewählt. Darüber hinaus werden die Bedeutungen *Refrigerator* (für *fridge*) und *Dishwasher* (für *dishwasher*) gewählt. Anschließend wird ein Bedeutungsgraph für jede Bedeutung erzeugt (S_1, S_2 und S_3 in der oberen Hälfte von Abbildung 7.2). Als Nächstes werden alle Knoten aller Graphen, die das gleiche Konzept repräsentieren, vereinigt. Im Beispiel betrifft das die Konzepte *home* und *home appliance* in den Bedeutungsgraphen S_1 und S_3. So entsteht der Themengraph im unteren Teil der Abbildung 7.2. In diesem Fall besteht der Themengraph aus zwei nicht verbundenen Teilgraphen (T_1 und T_2). Zusammenhängende Themengraphen weisen darauf hin, dass Bedeutungen thematisch verwandt sind. Aus dem Themengraph werden anschließend die Themen-Etiketten extrahiert; je nach angewendeter Strategie führt dies zu unterschiedlichen Ergebnissen. Unter Verwendung der Strategie *Höchste Konnektivität* werden ausschließlich Knoten aus T_1 in Betracht gezogen. Die grün hinterlegten Knoten weisen einen Konnektivitätswert von *zwei* auf, während die Konnektivität für alle anderen *eins* beträgt. Aus der Kandidatenmenge werden nun mithilfe des Ω-Wertes (ausgerichteter *PageRank*) die besten Etiketten ausgewählt; im Beispiel sind dies *home* und *home appliance*. Wird die Strategie *Maximale Abdeckung* angewendet, ändert sich die Auswahl. Zwar wird auch hier anhand der Konnektivität und des Ω-Wertes das Themen-Etikett *home* bestimmt; anschließend wird jedoch der Themengraph T_2, der bisher nicht repräsentierten Bedeutung *Orange(fruit)*, betrachtet. Wieder wird anhand des Ω-Wertes das beste Etikett gewählt, hier *fruit*. Die Strategie *Maximale Abdeckung* erzeugt somit die beiden Etiketten *home* und *fruit*.

7.3.2 Evaluation des Agenten zur Modellierung und Etikettierung von Diskurs-Themen

Die Bewertung des Ansatzes ist schwerlich anhand eines objektiven Goldstandards möglich. In den meisten Fällen ist die Einschätzung, ob ein Themen-Etikett zu einer natürlichsprachlichen Sequenz passt und ob diese die einzig passende ist oder nicht, subjektiv. Aus diesem Grund wurde zur Bewertung der generierten Etiketten eine Nutzerstudie mit sechs Probanden durchgeführt. Alle Probanden waren Master-Studenten unterschiedlicher Fakultäten, vier männliche und zwei weibliche in einer Alterspanne von 22 bis 27 Jahren. Als Testdatensatz wurden aus den ersten acht Szenarien des ProNat-Korpus (siehe Abschnitt 5.5) je zwei zufällige Transkriptionen gezogen (das heißt in Summe sechzehn); zusätzlich wurden sechs weitere synthetische Äußerungen erstellt. Diese ähneln in Länge, Komplexität und Formulierung den Transkriptionen des ProNat-Korpus. Der Grund für die Erstellung der zusätzlichen Äußerungen ist, dass die Szenarien des ProNat-Korpus alle in einer Küchenumgebung angesiedelt sind; die synthetischen Äußerungen entsprechen inhaltlich andersgeartete Szenarien (Drohnensteuerung im Freien, Kommandos an einen *Lego*-Roboter in einem Parcours und Anfragen an ein intelligentes Assistenzsystem, siehe Abschnitt 5.4.3). Dadurch kann überprüft werden, ob der Ansatz auch für andere Themenbereiche funktioniert. Die für die Nutzerstudie verwendeten Transkriptionen sind im Anhang in Abschnitt D.1 aufgeführt.

Zur Erzeugung der Themen-Etiketten für die Nutzerstudie wurde das vollständige ProNat-Vorverarbeitungsfließband durchlaufen (siehe Kapitel 6). Anschließend wurde der Agent für Disambiguierung basierend auf *Wikipedia* einmal ausgeführt, um die initialen Bedeutungen für die Nomen zu erzeugen (siehe Abschnitt 7.2). Zuletzt wurde der hier beschriebene Agent ausgeführt und die generierten Themen-Etiketten je Äußerung gespeichert; zur Generierung wurden beide Strategien (*Maximale Abdeckung* und *Höchste Konnektivität*) verwendet. Für die ausgewählten Äußerungen wurden zwischen vier und zehn Themen-Etiketten erzeugt. Anschließend wurden (mehrstufige) Formulare zur Bewertung der Etiketten erstellt. Die Formulare bestehen aus der jeweiligen Äußerung, allen extrahierten Etiketten, absteigend geordnet nach ihrer Konfidenz (Ω-Wert), und einer mehrstufigen Bewertungsauswahl je Etikette. Ein Formular ist schematisch im Anhang in Abschnitt C.3 dargestellt. Im Verlauf der Auswertung konnte beobachtet werden, dass die in den Formularen verwendeten Bewertungsmöglichkeiten zu feingranular gewählt wurden; die Probanden verwendeten nur wenige der möglichen Bewertungsklassen. Aus diesem Grund werden für die Auswertung nur die vier folgenden Klassen verwendet:

- *passend*: Diese Bewertung soll vergeben werden, wenn das Themen-Etikett ein oder mehrere Konzept(e) zutreffend beschreibt.
  Beispiel: Etikett *home appliance* zu *fridge* und *dishwasher*
- *zu allgemein*: Diese Bewertung soll vergeben werden, wenn das Themen-Etikett ein oder mehrere Konzept(e) beschreibt, aber zu allgemein ist.
  Beispiel: Etikett *domestic implements* zu *fridge* und *dishwasher*
- *verwandt*: Diese Bewertung soll vergeben werden, wenn das Themen-Etikett zu einem oder mehreren Konzept(en) verwandt ist, diese(s) aber nicht direkt beschreibt.
  Beispiel: Etikett *tool* zu *fridge* und *dishwasher*

Tabelle 7.10: Verteilung der Einschätzungen der Probanden zur Qualität der top-k Themen-Etiketten, die mithilfe der Strategien *Maximale Abdeckung* (Abd.) und *Höchste Konnektivität* (Kon.) erzeugt wurden.

	passend		zu allgemein		verwandt		unpassend	
k	Abd.	Kon.	Abd.	Kon.	Abd.	Kon.	Abd.	Kon.
1	0,530	0,530	0,242	0,242	0,045	0,045	0,182	0,182
2	0,447	0,424	0,167	0,182	0,106	0,106	0,280	0,288
3	0,449	0,444	0,141	0,152	0,157	0,146	0,253	0,258
4	0,432	0,420	0,144	0,140	0,155	0,167	0,269	0,273
5	0,381	0,369	0,145	0,136	0,176	0,182	0,298	0,312
alle	0,368	0,340	0,138	0,132	0,179	0,200	0,315	0,329

- *unpassend*: Diese Bewertung soll vergeben werden, wenn das Themen-Etikett für alle Konzepte der Äußerung unpassend ist.
  Beispiel: Etikett *clothes valet* zu *fridge* und *dishwasher*

Die sechs Probanden wurden in zwei Gruppen eingeteilt. Jeder Gruppe wurden elf Formulare (zu elf Äußerungen) zur Bewertung vorgelegt; das bedeutet, die Etiketten wurden unabhängig von je drei Probanden bewertet. Dementsprechend kann die Urteilerübereinstimmung (Interannotatorübereinstimmung) in Form von *Fleiss Kappa* bestimmt werden (siehe Abschnitt 2.4.9). Der ermittelte κ-Wert ist 0,27; nach Landis und Koch entspricht dies einer mittelmäßigen aber ausreichenden Übereinstimmung [LK77].

Die Verteilung der Bewertungen der Probanden ist in Tabelle 7.10 gezeigt. Dargestellt sind die relativen Häufigkeiten der Bewertungen für die k bestbewerteten (und alle) Etiketten für alle oben definierten Klassen und die beiden Strategien. Bewertungen aus der Klasse *passend* können als genau zutreffend gewertet werden. Das bedeutet, die Strategie *Maximale Abdeckung* erreicht eine Genauigkeit von 0,368 über alle Etiketten und 0,530 für die höchst-eingestuften. Davon abgesehen, sind auch als *zu allgemein* bewertete Etiketten in den meisten Fällen eine gute Wahl. Zählt man diese hinzu, steigt die Genauigkeit aller Etiketten auf 0,506 und die höchst-bewerteten auf 0,772. Auffällig ist, dass es nur wenige Etiketten gibt, die als *verwandt aber nicht beschreibend* eingeschätzt wurden (0,045). Da aber gerade diese Kategorie besonders subjektiv ist, kann angenommen werden, dass viele Probanden statt dieser Bewertung eher *zu allgemein* oder *unpassend* wählten. Hinsichtlich der Klasse *unpassend* sei darauf hingewiesen, dass für diese (anders als für die übrigen) gilt, dass niedrige Werte ein besseres Ergebnis darstellen. Insgesamt sind die Ergebnisse für Etiketten, die mithilfe der Strategie *Maximale Abdeckung* erzeugt wurden, besser bewertet als solche, die mit der Strategie *Höchste Konnektivität* generiert wurden. Die Beurteilung der Themen-Etiketten durch jeweils drei Probanden erlaubt eine Bewertung nach Mehrheitsentscheid. Hierzu werden zwei weitere Metriken verwendet. Die erste ist die *Präzision@k*, wie von Hulpus et al. eingeführt [Hul+13]:

$$\textit{Präzision@}k = \frac{\#\text{Treffer mit Rang} \leq k}{k} \tag{7.3}$$

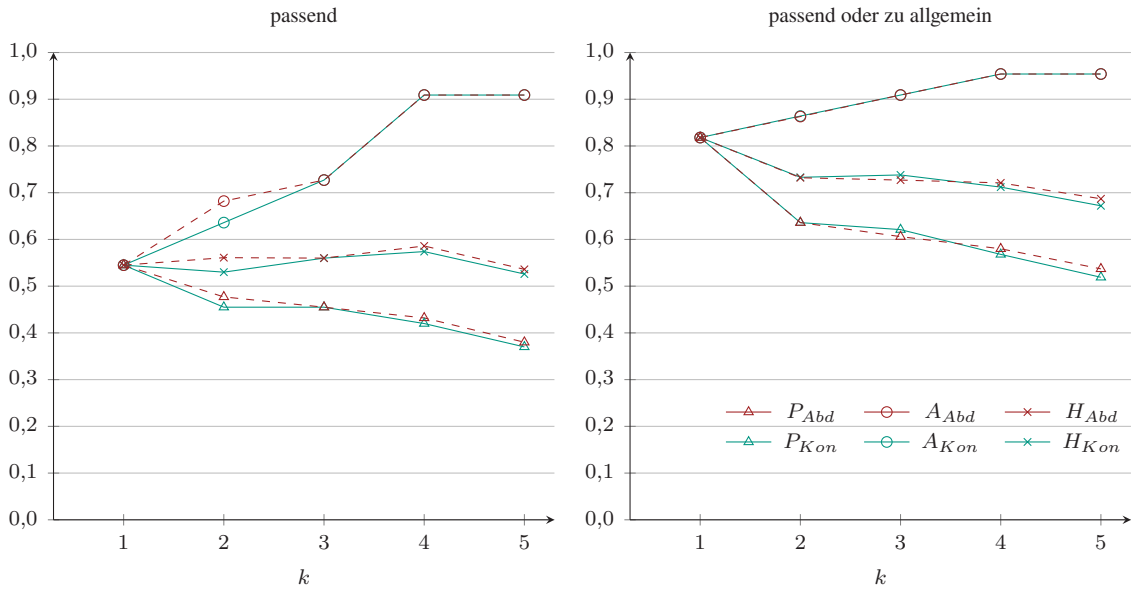


Abbildung 7.3: *Präzision@k* (*P*), *Abdeckung@k* (*A*) und *H@k* (*H*) für Themen-Etiketten, die per Mehrheitsentscheid als *passend* bzw. *passend* oder *zu allgemein* bewertet wurden: Dargestellt sind die Ergebnisse für beide Strategien, *Maximale Abdeckung* (Abd.) und *Höchste Konnektivität* (Kon.).

Die *Präzision@k* gibt an, wie viele Etiketten unter den ersten k Etiketten ($k = [1, 5]$) ein sogenannter *Treffer* sind. Ein Treffer ist eine Themen-Etikette, die durch die Probanden als *passend* (bzw. als *passend* oder *zu allgemein*) bewertet wurde. Die zweite Metrik ist eine angepasste Variante der von Hulpus et al. vorgeschlagenen *Abdeckung@k*[22]:

$$Abdeckung@k = \frac{\#\text{Äußerungen mit mind. 1 Treffer mit Rang} \leq k}{\#\text{Äußerungen}} \tag{7.4}$$

Die *Abdeckung@k* bestimmt den Anteil der Äußerungen, für die mindestens ein Etikett unter den ersten k Etiketten ein *Treffer* ist. Um bemessen zu können, an welchem Punkt der beste Kompromiss aus *Präzision@k* und *Abdeckung@k* erzielt werden kann, wird zusätzlich das harmonische Mittel ($H@k$), in Anlehnung an das F_1-Maß, bestimmt (siehe Abschnitt 2.4.3):

$$H@k = 2 * \frac{Präzision@k * Abdeckung@k}{Präzision@k + Abdeckung@k} \tag{7.5}$$

Da Präzision, Abdeckung und das harmonische Mittel nur für gute Etiketten bestimmt werden sollen, werden hier nur Etiketten betrachtet, die als *passend* oder *passend oder zu allgemein* eingestuft wurden. Abbildung 7.3 zeigt Liniendiagramme für beide Varianten. Dargestellt sind *Präzision@k*, *Abdeckung@k* und $H@k$ für die beiden Strategien *Maximale Abdeckung* und *Höchste Konnektivität*. Wie zu erwarten, steigt die Abdeckung mit steigenden Werten für k. Gleichzeitig sinkt aber die Präzision. Bei der Variante, bei der Etiketten, die als *passend oder zu allgemein* eingestuft wurden, als gut betrachtet werden, sind die Werte für die Abdeckung sehr gut (zwischen 0,801 und 0,954).

[22] Hulpus et al. erzeugen ein explizites Themen-Modell. Daher können sie die *Abdeckung@k* pro modelliertem Thema bestimmen. Da der hiesige Ansatz die Themen implizit modelliert, wird die *Abdeckung@k* pro Äußerung berechnet.

Selbst wenn nur *passende* Etiketten als gut angesehen werden, übersteigt die Abdeckung 0,909 bei $k \geq 4$. Natürlich wären höhere Werte für die Präzision wünschenswert. Immerhin sinkt die Präzision für *passende* Etiketten weniger stark als die Ausbeute steigt. Diese Beobachtung wird durch das harmonische Mittel bestätigt, das bei $k = 4$ mit 0,586 seinen höchsten Wert erreicht (Strategie *Maximale Abdeckung*). Bei Etiketten, die als *passend oder zu allgemein* eingestuft wurden, sinkt der Wert hingegen für das harmonische Mittel stetig (max. 0,818 bei $k = 1$, beide Strategien). Insgesamt lässt sich Folgendes festhalten: Zum einen sind die Werte für die Strategie *Maximale Abdeckung* in fast allen Fällen etwas besser als bei Verwendung der Strategie *Höchste Konnektivität*. Zum anderen ist die Verwendung von bis zu vier Themen-Etiketten je Äußerung sinnvoll, wenn man nur Etiketten zulassen möchte, die als *passend* bewertet wurden (legt man das harmonische Mittel als Kriterium zugrunde); betrachtet man hingegen *passende oder zu allgemeine* Etiketten ist nur ein Themen-Etikett je Äußerung sinnvoll, für $k > 1$ sinkt die Präzision stärker, als die Ausbeute steigt[23].

7.4 Modellierung des sprachlichen Kontextes

Die Interpretation einer natürlichsprachlichen Äußerung, wie sie nötig ist, um Programmieren mit natürlicher Sprache zu ermöglichen, bedarf einer tiefgehenden sprachlichen Analyse. Dazu genügt es nicht, nur die Bedeutung einzelner Wörter und die Themen des Diskurses zu verstehen (siehe Abschnitt 7.2 und 7.3). Die Bedeutungen der einzelnen Wörter stehen zu isoliert und die Diskurs-Themen sind zu grobgranular, um ein tiefes Verständnis der natürlichsprachlichen Äußerung zu generieren. Stattdessen muss der sprachliche Kontext einer Äußerung analysiert und formalisiert werden. Dazu gehören in der Äußerung genannte Entitäten, Konzepte und Ereignisse sowie Relationen zwischen diesen. Mithilfe eines solchen Modells können Zusammenhänge und Abläufe nachvollzogen werden. Die Modellierung des sprachlichen Kontextes ist hilfreich für jedwede weitere Analyse der Äußerung. Im Kontext von Pro$\mathcal{NAt}$ kann das Modell des sprachlichen Kontextes zur Unterstützung folgender Analysen verwendet werden: Korreferenz-Analyse, Korrektur von Wortfehlern des automatischen Spracherkenners und Erzeugung von Methodendefinitionen und Skripten (siehe Abschnitt 7.5 und 7.8).

Der Aufbau eines solchen Modells erfolgt über die Umgebung einzelner sprachlichen Einheiten (Wörter, Phrasen, etc.), ähnlich wie bei der Disambiguierung von Wortbedeutungen (siehe Abschnitt 7.2). Anhand der Umgebung können Zusammenhänge erkannt und auf bekanntes Wissen über Konzepte und ihrer Relationen zueinander zurückgeführt werden. Dieses Wissen kann aus Weltwissensdatenbanken oder -ontologien bezogen werden.

In der Literatur wird das Problem der Modellierung des sprachlichen Kontextes (im Kontext der Programmierung mit natürlicher Sprache) unterschiedlich, häufig aber nur implizit, betrachtet. Die Modellierung erfolgt nur soweit für die konkrete Lösung erforderlich, eine allgemeingültige und übertragbare Lösung wird nicht angestrebt. Auch wird der sprachliche Kontext zumeist nicht formal

[23] Der Agent ist daher so vorkonfiguriert, dass er die Strategie *Maximale Abdeckung* verwendet und bis zu fünf Themen-Etiketten ausgibt.

definiert oder überhaupt explizit modelliert. Ansätze, die auf rekurrenten neuronalen Netzen basieren, verwenden den sprachlichen Kontext schon aufgrund der verwendeten Architektur [XS14]. Da vorherige Eingaben immer wieder in das Netz eingespeist werden (Rekurrenz), sind Klassifikationen immer auch teilweise vom Kontext der aktuellen Eingabe abhängig. Ähnlich verhält es sich für Ansätze, die teilweise beobachtbare Markov-Entscheidungsprozesse (eng. *partially-observable markov decision process*, *POMDP*) verwenden [You+10]. Diese modellieren den Kontext implizit in ihrem Glaubenssystem (engl. *belief system*). Implizite Kontext-Modellierung wird auch in wissensbasierten Systemen verwendet. Bei *Aktiven Ontologien* (engl. *Active Ontology*, *AO*) beispielsweise verbleiben zuvor extrahierte Informationen im System bis sie von einer gleichartigen verdrängt werden [GCB06; GBC06; GBC07]. Auf diese Weise können Informationen aus einer vorangegangenen Äußerung verwendet werden, um die aktuelle zu interpretieren. Diese impliziten Modellierung, wie von den beschriebenen Ansätzen verwendet, ist beschränkt auf die jeweilige Anwendung und ist weder übertragbar noch direkt nutzbar. Auch der Einfluss auf die Ergebnisse ist schwer zu bemessen. Daher modellieren einige Ansätze den Kontext explizit. Misra et al. nutzen beispielsweise explizit modellierten situativen Kontext (der Systemumgebung), um die (inferierte) geteilte Information zwischen System und Nutzer zu validieren [Mis+15]. Für denselben Zweck verwenden Bordes et al. im sprachlichen Kontext erkannte Konzepte [Bor+10]. Fleischmann und Roy hingegen nutzen den sprachlichen Kontext von Aktionen, um ihren Ansatz zur Abbildung von natürlichsprachlichen Äußerungen auf Systemaktionen zu verbessern [FR05]. Den aufgeführten Ansätze ist allen gemein, dass sie entweder zum Ziel haben, die Umgebung zu modellieren oder eine Umgebungsmodellierung nutzen, um neues Wissen zu inferieren. Für Pro$\mathcal{N}at$ soll hingegen der sprachliche Kontext an sich modelliert werden. Das Modell bezieht zwar ebenso Umgebungs- und Weltwissen ein, modelliert werden aber in der natürlichsprachlichen Äußerung genannte Entitäten, Konzepte und Ereignisse sowie Relationen zwischen diesen.

7.4.1 Ein Model des sprachlichen Kontextes

Zunächst gilt es festzulegen, was genau unter sprachlichem Kontext zu verstehen ist. In der Literatur existieren unterschiedliche Definitionen für den Begriff *Kontext* im Bezug auf natürliche Sprache [Dur92; Sta99; Fet04]. Allen Definitionen ist jedoch gemein, dass Kontext eine Art Information beschreibt, die genutzt werden kann, um die Bedeutung eines (anderen) sprachlichen Artefakts zu erfassen. Diese Artefakte sind die Bestandteile des sprachlichen Diskurses. Der Kontext der Artefakte umfasst Informationen über die Sprachsituation (Umgebung) der Äußerung, wie den Ort, die Zeit, die Beziehung der Gesprächspartner und Annahmen über geteiltes Wissen (Weltwissen) der Gesprächspartner. Zusätzlich bilden vorangegangene Äußerungen eine sprachliche Umgebung für die jeweilige Äußerung.

Da für Pro$\mathcal{N}at$ einzig natürlichsprachliche Äußerungen als Eingabe verwendet werden sollen, muss zunächst analysiert werden, welche Kontextinformationen überhaupt erzeugt werden können. Neben der Äußerung als solcher stehen auch die vorangegangenen Äußerungen im Diskurs zur Verfügung. Einzelne Äußerungen enthalten Informationen über Aktionen (oder Ereignisse), die von Akteuren ausgeführt werden und Objekte behandeln. Andere Ausdrücke enthalten zeitliche oder örtliche Einordnungen und Bedingungen. Entitäten wie Subjekte, Objekte und Aktionen sind Instanzen

bestimmter Konzepte, die inferiert werden müssen. In natürlichsprachlichen Äußerungen gehören viele Entitäten zum selben Konzept. Gleichermaßen sind viele Konzepte Spezialisierungen von Oberkonzepten. Folglich bilden die in einer sprachlichen Äußerung enthaltenen Konzepte eine (Konzept-)Hierarchie.

Beispiel:
Konzepte und Oberkonzepte in sprachlichen Kontexten

In diesem Beispiel soll die grundlegende Idee von Konzepten und Oberkonzepten in einem sprachlichen Kontext demonstriert werden.

Natürlichsprachliche Sequenz

the fridge and dishwasher are running

Enthaltene Konzepte und Oberkonzepte

Der Begriff *fridge* ist eine Manifestation des Konzeptes `refrigerator`; gleiches gilt für *dishwasher* und `dishwasher`. Die Phrase *are running* bezieht sich auf das Konzept `run_operate` (und nicht etwas auf `run_move`, welches auch möglich, aber im Kontext nicht sinnvoll wäre). Letztlich findet sich zu den Konzepten `refrigerator` und `dishwasher` das gemeinsame Oberkonzept `white_goods`.

Ausgehend von den diskutierten Beobachtungen, wird im Folgenden ein dreischichtiges Modell für sprachlichen Kontext definiert. In der ersten Schicht (*Individuelle Information*) werden die individuellen Informationen je sprachlichem Artefakt modelliert. Die zweite Schicht (*Konzeptionelle Information*) beschreibt die Konzepte, die von den jeweiligen Entitäten der ersten Schicht geformt werden. Zuletzt modelliert die dritte Schicht (*Hierarchische Information*) die Generalisierungen der Konzepte (Oberkonzepte). Die Schichten bilden ein hierarchisches Modell zunehmender Abstraktion von der natürlichsprachlichen Äußerung. Tabelle 7.11 gibt einen Überblick über die Schichten und die jeweils enthalten Typen von Kontextinformation.

Die erste Schicht umfasst alle Informationen, die direkt aus dem geäußerten Handlungsverlauf entnommen werden können. Das bedeutet, in der ersten Schicht werden Aktionen, Entitäten, etc. modelliert, die in der natürlichsprachlichen Äußerung auftreten. Alle auftretenden Nomen sind entweder Subjekte oder Objekte eines Prädikates. Sowohl Subjekte als auch Objekte bilden Entitäten im Kontextmodell. Davon ausgehend werden mögliche Beziehungen zwischen den Entitäten betrachtet. Beschreibt ein Prädikat eine Handlung, wird diese Information als *Aktion* im Modell hinterlegt. Eine Aktion kann auf Entitäten Bezug nehmen, wie in der Äußerung „robot go to the table“: `go(who: robot, where: table)`. So kann über den Diskurs hinweg modelliert werden, welche Entitäten von Handlungen beeinflusst wurden. Eine Ausprägung einer solchen Einflussnahme ist die Änderung des Zustands einer Entität durch eine Handlung. Diese Art von Kontextinformation wird als *Zustandsübergang* im Modell hinterlegt. Beispielsweise erzeugt die Äußerung „open the fridge“ einen Zustandsübergang der Entität `refrigerator` in den Zustand `opened`. Zustandsübergänge liefern Informationen darüber, ob ein Handlungsablauf statthaft ist; es

Tabelle 7.11: Kontext-Schichten und -Arten.

Schicht	Art	Beschreibung
Individuell	Entität	Dinge in der realen Welt (Systemumgebung)
	Spatial Deixis	Räumliche Relationen zwischen Entitäten
	Aktion	Ereignisse beschrieben im sprachl. Kontext
	Zustandsübergang	Zustandswechsel, induziert durch Aktionen
Konzeptionell	Konzept	Abstraktion einer Entität oder Aktion
	Zustand	Zustände, in denen eine Entität sein kann
Hierarchisch	Überkonzept	Oberbegriff zu einem Konzept (Hyperonym)
	Teil-Ganzes-Beziehung	Entität ist ein Teil einer anderen (Meronym)

ist beispielsweise nur sinnvoll etwas zu *öffnen*, was zuvor *geschlossen* war. Da ProNat ausschließlich die natürlichsprachlichen Äußerungen zur Verfügung stehen, kann nur aus Zustandsübergängen auf den Ausgangszustand von Entitäten geschlossen werden. Eine weitere häufig verwendete Beziehung beschreibt örtliche Einordnungen, wie `cup : is_located_on : table` in Äußerungen wie „the cup on the table“. Diese Art Kontextinformation wird als *Spatial Deixis* bezeichnet.

Die zweite Schicht enthält Abstraktionen folgender Modell-Elemente der ersten Schicht: Entitäten, Aktionen und Zustandsübergänge. Menschen formen zum Verständnis ihres Gegenübers Konzept aus Begriffen des Diskurses. Hierzu nutzen sie erlerntes Wissen über die Welt, Dinge die existieren (können) und Handlungen, die ausgeführt werden können. Dieses Wissen umfasst unter anderem unterschiedliche Bedeutungen von Begrifflichkeiten, Synonyme oder Beziehungen zwischen Begriffen. Beispielsweise wissen Menschen (für gewöhnlich), dass die Begriffe *fridge* und *refrigerator* das gleiche Konzept beschreiben. Dementsprechend werden *Konzepte* als weitere Kontextart definiert. Konzepte können als Abstraktion sowohl für Entitäten als auch für Aktionen gebildet werden. Natürlich sollen Entitäten und Aktionen bei wiederholter Nennung im Diskurs immer auf dasselbe Konzept abgebildet werden. Aus einer Zustandsänderung, wie zuvor beschrieben, kann die Existenz bestimmter Zustände eines Konzeptes gefolgert werden. Während eine Zustandsänderung immer eine konkrete Entität betrifft, gelten mögliche Zustände für alle Entitäten und damit für das zugehörige Konzept. Aus diesem Grund werden Zustände der zweiten Schicht zugeordnet. Außerdem kann auf konzeptioneller Ebene ein Zustand zu mehreren Konzepten gehören; beispielsweise kann der Zustand `opened` sowohl zum Konzept `refrigerator` als auch zu `dishwasher` gehören.

Die dritte Schicht beschreibt Oberkonzept- und Teil-Ganzes-Beziehungen zwischen Konzepten. Menschen formen nicht nur wie zuvor beschrieben Konzepte aus Begrifflichkeiten, sie abstrahieren auch und stellen Verbindungen zwischen Konzepten her. Dies geschieht auf vielfältige Weise, von denen zwei im hier verwendeten Kontextmodell abgebildet werden. Zum einen bilden Menschen übergeordnete Konzepte zu den zuvor bekannten Konzepten. So können hierarchische Beziehungen zwischen Konzepten hergestellt werden. Diese Kontextart wird hier als *Oberkonzept* bezeichnet; beispielsweise haben die Konzepte `refrigerator` und `dishwasher` das gemeinsame Oberkonzept `white_goods`. Zum anderen können Menschen auch erkennen, in welchen Fällen ein Konzept ein Teil eines anderes ist. Ein solcher Zusammenhang wird als *Teil-Ganzes-Beziehung* bezeichnet;

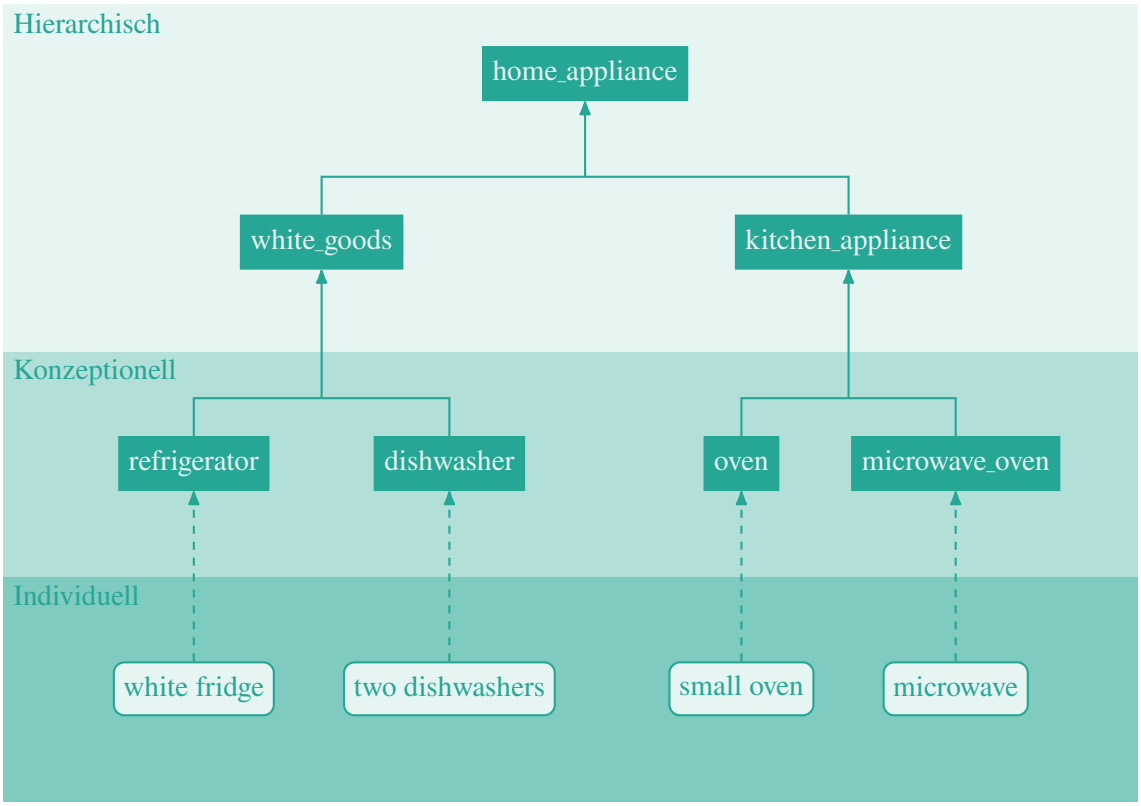


Abbildung 7.4: Entitäten und Konzepte eines konkreten Kontextmodells: Dargestellt sind beispielhafte Zusammenhänge zwischen Entitäten und zugehörigen Konzepten und Oberkonzepten über die verschiedenen Schichten des Kontextmodells.

beispielsweise kann anhand der Äußerung „go to the fridge and open its door“ gefolgert werden, dass das Konzept *refrigerator* ein Teil-Konzept *refrigerator.door* besitzt. Abbildung 7.4 zeigt beispielhaft für die Entitäten *white fridge*, *two diswashers*, *small oven* und *microwave* die inferierten Konzepte und Oberkonzepte.

7.4.2 Implementierung des Agenten zur Modellierung des sprachlichen Kontextes

Um die beschriebene Modellierung des sprachlichen Kontextes für natürlichsprachliche Äußerungen zu generieren, wurde ein `ProNat`-Agent implementiert [WHT17a; WHT17b]. Der Agent liest zunächst den aktuellen `ProNat`-Graph aus und betrachtet alle darin enthaltenen Knoten vom Typ `Token`, welche die ursprüngliche natürlichsprachliche Äußerung darstellen. Aus den Token liest der Agent die Informationen über die Wortart, das Lemma, die Chunk-Zugehörigkeit und die semantische Rolle sowie gegebenenfalls die Markierung als Eigenname aus. Damit sind die Ergebnisse des Agenten direkt abhängig von den drei entsprechenden Vorverarbeitungsstufen (siehe Abschnitt 6.2, 6.3 und 6.4). Zusätzlich wird – sofern vorhanden – die Bedeutungsmarkierung extrahiert, die durch den *Babelfy*-Agenten zur Disambiguierung von Wortbedeutungen erzeugt wurden.

Das Kontextmodell wird anschließend iterativ von unten nach oben aufgebaut. Zunächst werden die Modell-Elemente der ersten Schicht (*Individuelle Information*) erzeugt. Mithilfe der Chunk-Zugehörigkeit werden zunächst die Entitäten bestimmt, indem alle Nominalphrasen (Etikett *NP*) extrahiert werden. Diese werden heuristische unter Zuhilfenahme von Wortartmarkierungen und Chunks um Informationen bezüglich des Numerus, adjektivische Modifikatoren sowie zugehöriger Quantoren und Artikeln ergänzt. Anschließend werden die örtlichen Beziehungen (*Spatial Deixis*) modelliert. Hierzu wird eine Schlüsselwortsuche auf natürlichsprachlichen Ausdrücken zwischen zwei Entitäten durchgeführt. Dieser Ansatz ist statthaft, da im Englischen örtliche Beziehungen normalerweise mithilfe von Präpositionalphrasen (zwischen Nominalphrasen) ausgedrückt werden [Cla73]. Zuletzt werden die Aktionen modelliert. Um Aktionen in den Token zu erkennen, wird eine regelbasierte Analyse auf die semantischen Rollen angewandt. Da wie in Abschnitt 6.4 beschrieben nicht nur die reinen semantischen Rollen erzeugt werden, sondern zusätzlich eine Verknüpfung zu *PropBank* und *VerbNet* stattfindet, können die Argumente der Aktionen konkreten Entitäten zugeordnet werden. Zustandsübergänge werden erst gemeinsam mit den Modell-Elementen der zweiten Schicht modelliert, da hierfür die erkannten Konzepte notwendig sind.

Zur Erzeugung der Modell-Elemente der zweiten Schicht (*Konzeptionelle Information*) werden zunächst Konzepte modelliert. Hierzu wird versucht, alle Entitäten auf Individuen der eingebundenen Pro$\mathcal{N}at$-Domänenontologie abzubilden(siehe Abschnitt 5.4). Für die Abbildung wird die Jaro-Winkler-Ähnlichkeit mit einem Schwellenwert von 0,92 verwendet[24][25] (siehe Abschnitt 2.3.13.2). Zusätzlich werden Synonyme aus *WordNet* verwendet, um auch unscharfe Abbildungen zu ermöglichen. Steht die Wortbedeutung zur Verfügung, werden statt aller Synonyme nur die zur Wortbedeutung gehörenden verwendet[26] Kann auf diese Weise keine Abbildung ermittelt werden, wird mithilfe von *WordNet* ein Konzept synthetisiert. Hierzu wird zunächst für die vollständige Phrase, die die Entität repräsentiert, eine Entsprechung in *WordNet* gesucht. Falls dies erfolglos bleibt, wird die Suche mit Sub-Phrasen abnehmender Länge wiederholt, wobei der Kopf der Phrase immer enthalten sein muss. Für die Phrase „the small white fridge“ werden beispielsweise nacheinander die folgenden Anfrage-Zeichenketten erzeugt: *small white fridge*, *small fridge*, *white fridge* und *fridge*[27]. Kann für keinen Teil der Phrase ein *WordNet*-Eintrag bestimmt werden, wird kein Konzept erzeugt. Dasselbe zweischrittige Verfahren wird auch für die Erzeugung von Konzepten für Aktionen angewandt. Für alle Konzepte (egal ob direkt oder mithilfe von *WordNet* abgebildet), werden zusätzlich Synonyme aus *WordNet* hinterlegt. Anschließend werden für alle Konzepte die möglichen Zustände bestimmt; diese sind in der Pro$\mathcal{N}at$-Ontologie hinterlegt. Für jeden extrahierten Zustand wird ein Modell-Element erzeugt und mit den jeweiligen Konzepten verbunden. Als Nächstes werden mögliche Zustandsübergänge untersucht. Hierzu wird für jedes Aktionskonzept in der Pro$\mathcal{N}at$-Ontologie die Information abgefragt, ob dieses Konzept zu einem Zustandswechsel führen kann und welche Zustände potenziell involviert sind. Anschließend wird

[24] Für die Abbildung werden nur die Nomen der Nominalphrase, die die Entität beschreibt, verwendet.

[25] Der Schwellenwert wurde empirisch anhand von Beispielen aus den Szenarien eins bis fünf des Pro$\mathcal{N}at$-Korpus ermittelt.

[26] Da die Bedeutungsmarkierungen in Form von *WordNet-Synsets* vorliegen, werden die passenden Synonyme direkt mitgeliefert.

[27] Stoppwörter wie Artikel werden grundsätzlich von der Suche ausgeschlossen.

versucht, diese Information auf die Aktionsinstanzen und die in Verbindung stehenden Entitäten abzubilden. Beispielsweise enthält die Ontologie die Information, dass die Aktion `open` einen Zustandswechsel von `closed` zu `opened` hervorruft und dass `refrigerator` die Zustände `opened` und `closed` besitzt. Aus Äußerungen wie „open the fridge“ kann dann gefolgert werden, dass die Entität *fridge* (Konzept *refrigerator*) ihren Zustand von *closed* zu *opened* ändert. Eben diese Informationen werden auch im Modell hinterlegt. Für jede zustandsverändernde Aktion wird eine Verbindung zum entsprechenden Zustand hergestellt. Ebenso wird die entsprechende Entität mit dem Zustand verbunden. Alle folgenden Entitäten, die demselben Konzept entsprechen, werden ebenfalls mit dem Zustand verbunden, es sei denn, es tritt ein erneuter Zustandsübergang ein.

Ausgehend von den Konzepten können nun die Modell-Elemente der dritten Schicht (*Hierarchische Information*) erzeugt werden. Zunächst wird die Konzept-Hierarchie mithilfe von *WordNet* aufgebaut. Hierzu werden alle Konzepte paarweise betrachtet. Augenscheinlich ist das gemeinsame Oberkonzept zweier Konzepte, welches in der Hyperonym-Hierarchie von *WordNet* am weitesten unten steht, (engl. *least common subsumer*, *LCS*) ein guter Kandidat für ein Oberkonzept im Kontextmodell [WP94]. Allerdings würden mit diesem naiven Vorgehen viele unpräzise (bzw. zu allgemeine) Oberkonzepte wie *artifact* oder *entity* erzeugt werden. Aus diesem Grund verwendet der Ansatz das Ähnlichkeitsmaß von Wu und Palmer [WP94] mit einem Schwellenwert von 0,7 und einem zusätzlichen Tiefen-Filter (für die *WordNet*-Konzepthierarchie), um zu allgemeine Konzept auszuschließen. Falls eines der beiden Konzepte ein Oberkonzept des anderen in *WordNet* ist, wird diese Beziehung auch in das Kontextmodell übernommen. Ist ein Oberkonzept bereits vorhanden, werden die jeweiligen Konzepte mit diesem verbunden; es findet also keine Duplizierung von Oberkonzepten statt. Das Vorgehen zur Erzeugung von Oberkonzepten wird iterativ fortgesetzt. Das bedeutet, für erzeugte Oberkonzepte wird erneut versucht, ein neues gemeinsames Oberkonzept zu finden. Dies wird solange fortgesetzt bis keine Oberkonzepte mehr erzeugt werden können (gemäß der oben beschriebenen Einschränkungen). Zuletzt werden Teil-Ganzes-Beziehungen generiert. Einige davon sind in den ProNat-Domänenontologien hinterlegt und können direkt übernommen werden (siehe Abschnitt 5.4). Darüber hinaus werden Meronym-Relationen aus *WordNet* verwendet, um weitere Teil-Ganzes-Beziehungen zwischen bereits generierten (Ober-)Konzepten zu erzeugen. Bei den in der Ontologie hinterlegten Konzepten werden hingegen alle zugehörigen Teil- und Ganzes-Konzepte hinterlegt, unabhängig davon, ob sie in der Äußerung genannt werden oder nicht. Der Agent zur Modellierung des sprachlichen Kontextes nutzt zur Erzeugung der Modell-Elemente die Eigenschaft der PARSE-Rahmenarchitektur, dass Agenten mehrfach ausgeführt werden können (siehe Abschnitt 4.2.5). Die zuvor beschriebenen iterativen Schritte entsprechen Aufrufen des Agenten. Das bedeutet, wird der Agent zum ersten Mal aufgerufen, werden ausschließlich Modell-Elemente der ersten Schicht erzeugt, beim zweiten Aufruf die der zweiten und beim dritten Aufruf die der dritten. Bei jedem weiteren Aufruf werden weitere Elemente der dritten Schicht erzeugt, solange dies nach dem zuvor beschriebenen Verfahren möglich ist.

Das auf diese Weise erzeugte Kontextmodell besteht aus drei Schichten mit fünf Kontextarten, die als Knoten im ProNat-Graph hinterlegt werden (*Entität*, *Aktion*, *Konzept*, *Oberkonzept* und *Zustand*), und drei Arten, die als Kanten modelliert werden (*Spatial Deixis*, *Zustandsübergang* und *Teil-Ganzes-Beziehung*).

Beispiel:
Erzeugung des Kontextmodells

In diesem Beispiel soll die Erzeugung des Kontextmodells anhand einer synthetischen, natürlichsprachlichen Äußerung demonstriert werden. Das Beispiel nimmt Bezug auf Abbildung 7.5, welche das erzeugt Kontextmodell visualisiert.

Natürlichsprachliche Sequenz

Robo turn on the small oven between the oven and the fridge

Vorgehen

Beim ersten Aufruf des Agenten werden zunächst alle Nominal- und Verbalphrasen sowie zugehörige Adjektive, Adverbien etc. extrahiert. Aus diesen Informationen werden die Entitäten *Robo*, *small oven*, *oven* und *fridge* sowie die Aktion *turn on* erzeugt. Anschließend werden heuristisch die örtlichen Beziehungen modelliert, hier die Information, dass sich die Entität *small oven* zwischen den Entitäten *oven* und *fridge* befindet. Wird der Agent von der Rahmenarchitektur erneut aufgerufen, werden Konzepte der zweiten Schicht gebildet. Im Beispiel sind alle Konzepte Bestandteil der ProNat-Domänenontologie. Das Konzept *oven* subsumiert die Entitäten *small oven* und *oven*. Zusätzlich wird der Zustandsübergang der Entität *small oven* in den Zustand *on* geschlossen; für das Konzept *oven* werden außerdem die beiden möglichen Zustände *on* und *off* aus der Ontologie extrahiert. Beim nächsten Aufruf generiert der Agent mithilfe von *WordNet* das gemeinsame Oberkonzept *home_appliance* für die Konzepte *oven* und *refrigerator*. Außerdem wird dem Modell das Teil-Konzept *oven.door* hinzugefügt.

7.4.3 Evaluation des Agenten zur Modellierung des sprachlichen Kontextes

Zur Evaluation des Agenten wurden die Transkriptionen zu den Szenarien sechs und sieben der stationären Datensammlung des ProNat-Korpus verwendet (siehe Abschnitt 5.5.2). Der Datensatz enthält insgesamt 21 Äußerungen mit insgesamt 264 Instruktionen. Weitere Informationen zum verwendeten Datensatz können Tabelle 7.12 entnommen werden. Für alle Transkriptionen wurden Musterlösungen erstellt; das bedeutet, die erwarteten Modell-Elemente wurden phrasenweise händisch annotiert. Konzepte und Oberkonzepte werden an alle zugehörigen Entitäten bzw. Aktionen annotiert. Die Kontextarten, die als Kante im Graph modelliert werden, werden an der Zielphrase annotiert (inklusive eines Verweises auf die Quellphrase)[28]. Hinsichtlich der Zustände wird in der Evaluation zwischen erzeugten *Zustands*-Elementen und Verbindungen zu diesen unterschieden. Die Modell-Elemente werden erzeugt, wenn der Zustand (bzw. komplementäre Zustand) das

[28] Eine Ausnahme bilden die Teil-Ganzes-Beziehungen. Wird in der Äußerung nur das *Ganze* genannt, wird die Annotation an dieser Stelle hinzugefügt, da keine Zielphrase für die Beziehung existiert.

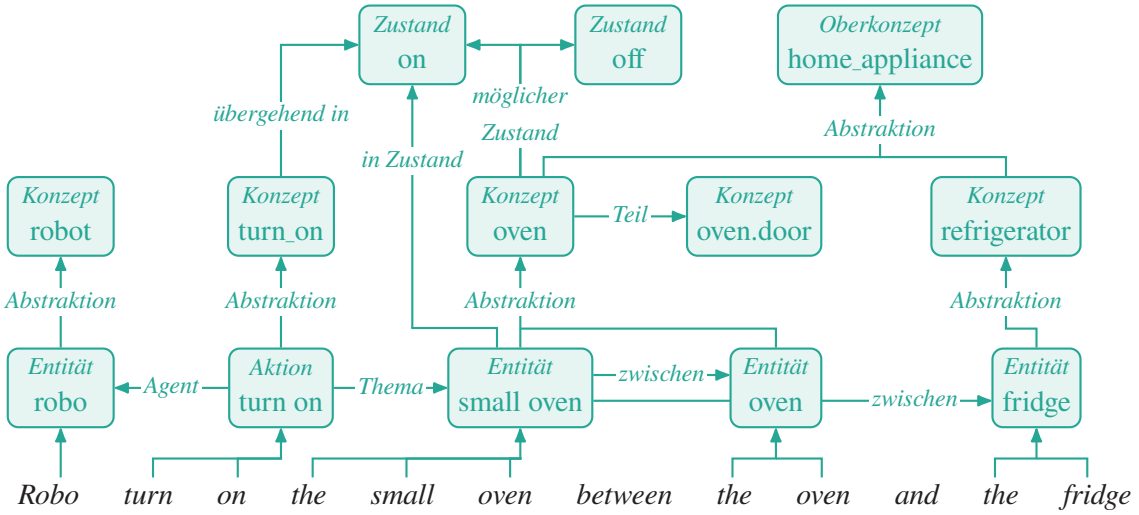


Abbildung 7.5: Modellierung des sprachlichen Kontextes zu einer synthetischen Beispiel-Äußerung.

erste Mal in der Äußerung genannt wird. Eine Verknüpfung zu einer *Entität* findet aber bei jeder Nennung der Entität statt, solange die Entität in diesem Zustand verbleibt. Durch die zusätzliche Vermessung der Zustandsverknüpfungen kann detaillierter bestimmt werden, für welche *Entitäten* die richtigen Zustände bzw. Zustandsübergänge gefolgert werden können. Eine Übersicht über die erstellten Modell-Elemente des Gold-Standards wird ebenfalls in Tabelle 7.12 gegeben[29]. Um das vom Agenten erzeugt Kontextmodell zur Evaluation zu erhalten, wurde zunächst das vollständige ProNat-Vorverarbeitungsfließband ausgeführt. Anschließend wurde der *Babelfy*-Agent zur Disambiguierung von Wortbedeutungen ausgeführt. Zuletzt wurde der Agent zur Modellierung des sprachlichen Kontextes solange wiederholt ausgeführt bis keine Veränderung mehr im ProNat-Graphen eintritt. Das erzeugte Modell wurde anschließend mit dem Gold-Standard verglichen.

Zur Bewertung des generierten Modells wurden pro Kontextart Präzision, Ausbeute und F_1-Maß bestimmt. Tabelle 7.13 zeigt eine Übersicht über die Evaluationsergebnisse. Die Werte für das F_1-Maß reichen je nach Kontextart von 0,723 bis 0,977, wobei ein Großteil einen Wert größer 0,9 erzielt. Die Ausbeute für die Kontextarten *Spatial Deixis*, *Zustandsübergang* und *Aktion* fällt deutlich niedriger aus als für die restlichen. Für letztere können viele Probleme auf Fehler in der Vorverarbeitungsphase zurückgeführt werden; insbesondere fehlerhafte Wortartetiketten und semantische Rollen führen dazu, dass Aktionen nicht korrekt erkannt werden. Der verwendete semantische Rollenerkenner *SENNA* erreicht einen Wert von 0,755 für das F_1-Maß für die Vergleichsaufgabe der *SIGNLL Conference on Computational Natural Language Learning 2005* (*CoNLL-2005 shared task*) [Col+11]. Dementsprechend ist das hier erzielte F_1-Maß von 0,804 vielversprechend. Für die beiden erstgenannten Kontextarten kann die niedrige Ausbeute damit begründet werden, dass die Erzeugung dieser Modell-Elemente heuristikbasiert erfolgt und die Heuristiken offensichtlich nicht alle auftretenden Fälle abdecken. Zusätzlich findet durch die inkrementelle Erzeugung des Modells eine Fehlerfortpflanzung statt. Fehler, die auf niedrigen

[29] Die große Menge der händisch zu annotierenden Modell-Elemente ist der Grund für die vergleichsweise geringe Anzahl an Transkriptionen als Evaluationsdatensatz.

Tabelle 7.12: Der für die Evaluation der Modellierung des sprachlichen Kontextes verwendete Datensatz.

	Szenario 6	Szenario 7	Gesamt
Äußerungen	10	11	21
enthaltene Wörter	734	811	1545
enthaltene Phrasen	467	543	1010
enthaltene Instruktionen	121	143	264
Entitäten	199	233	432
Spatial Deixes	41	43	84
Aktionen	120	154	274
Zustandsübergänge	33	48	81
Konzepte	274	320	594
Zustände (Verbindungen)	8 (306)	10 (338)	18 (644)
Oberkonzepte	88	56	144
Teil-Ganzes-Beziehungen	35	43	78

Tabelle 7.13: Ergebnisse der Evaluation der Modellierung des sprachlichen Kontextes nach Kontextarten.

Kontextart	Präzision	Ausbeute	F_1
Entität	0,972	0,975	0,973
Spatial Deixis	0,945	0,793	0,862
Aktion	0,852	0,762	0,804
Zustandsübergang	0,854	0,627	0,723
Konzept	0,986	0,974	0,981
Zustand	1,000	0,955	0,977
Oberkonzept	0,680	0,932	0,786
Teil-Ganzes-Beziehung	0,897	0,959	0,927

Abstraktionsebenen (oder bereits in der Vorverarbeitung) auftreten, führen zu weiteren Fehlern in den höheren Schichten. Wird beispielsweise ein Verb nicht als solches erkannt, wird keine *Aktion* erzeugt. In der Folge kann auch kein *Zustandsübergang* und somit auch kein *Zustand* inferiert werden. Der Fehler pflanzt sich sogar noch weiter fort. Da die Zustandsverbindungen einzeln bewertet werden, bedingt ein fehlender *Zustandsübergang* gegebenenfalls eine Vielzahl von fehlenden Zustandsverbindungen. Hinsichtlich der örtlichen Beziehungen kann ein Großteil der nicht erzeugten Modell-Elemente auf mehrdeutige Sequenzen in den Äußerungen zurückgeführt werden. Werden beispielsweise verschachtelte Präpositionalphrasen wie in „put the meal from the fridge on the table“ verwendet, ist eine eindeutige Interpretation nicht möglich. Der Agent entscheidet sich in diesen Fällen entweder für eine der möglichen Interpretationen (gegebenenfalls für eine falsche) oder keine. Die Generierung von *Konzepten* und die Verbindung von *Konzepten* und *Zuständen* funktioniert zuverlässig; *falsch positive* und *falsch negative* Ergebnisse lassen sich ausnahmslos auf Fehler in der Vorverarbeitung zurückführen. Auch die Erzeugung der Teil-Ganzes-Beziehungen erzielt ähnlich gute Ergebnisse. Hier können die wenigen Fehleinschätzungen auf Fehler bei der

Bestimmung der Wortbedeutung zurückgeführt werden. Wird beispielsweise dem Wort *plate* die Bedeutung *a sheet of metal or wood or glass or plastic* zugewiesen, obwohl im Kontext die Bedeutung *dish on which food is served or from which food is eaten* gemeint war, wird *plate* fälschlicherweise zu einem Teil-Konzept von *table*. Die Generierung von *Oberkonzepten* erzielt gute Werte für die Ausbeute, die Präzision weist hingegen die mit Abstand schlechtesten Werte auf. Wieder können falsche Zuordnungen auf Fehler bei der Disambiguierung zurückgeführt werden. Außerdem ist die *WordNet*-Hyperonym-Hierarchie weder vollständig noch in allen Fällen eindeutig, was zu weiteren Fehlern führt. Zusätzlich wirkt sich die Fehlerfortpflanzung bei der iterativen Erzeugung von *Oberkonzepten* deutlich stärker aus als bei anderen Kontextarten. Grundsätzlich ist eine höhere Ausbeute zulasten der Präzision bei der Erzeugung von *Oberkonzepten* aber gewünscht. Fehlerhafte *Oberkonzepte* können von anderen Analysen ignoriert werden (die vollständige Konzept-Hierarchie ist ohnehin in den seltensten Fällen nützlich); werden Informationen hingegen nie erzeugt, können sie auch nie genutzt werden.

Zusammenfassend lässt sich festhalten, dass der Ansatz für die Kontextarten *Entität*, *Spatial Deixis*, *Konzept*, *Zustand* und *Teil-Ganzes-Beziehung* sehr gute Ergebnisse erzielt. Die Kontextarten *Aktion* und *Zustandsübergang* würden hingegen von einer verbesserten Erkennung semantischer Rollen profitieren. *Oberkonzepte* können mit einer hohen Ausbeute, aber niedriger Präzision erzeugt werden; die Generierung hängt dabei stark von der Qualität der *WordNet*-Hyperonym-Hierarchie für die gegebenen *Konzepte* ab.

7.5 Korreferenzanalyse

Die Verwendung von sprachlichen Referenzen ist ein wesentlicher Unterschied zwischen natürlichen und formalen Sprachen. Durch Referenzen werden Sprachen ökonomischer und in vielen Fällen stilistisch schöner und damit für den Hörer (oder Leser) angenehmer. Allerdings setzen sprachliche Referenzen voraus, dass die Diskursteilnehmer wissen, wie diese aufzulösen sind. Je nach Art der Referenz setzt dieses Vermögen eine andere Art geteilten (Welt-)Wissens voraus. Diese Voraussetzung führt dazu, dass Referenzen grundsätzlich zu Mehrdeutigkeiten führen können. Viele der Mehrdeutigkeiten, die durch die Verwendung von natürlichsprachlichen Referenzen entstehen, können über Regelwissen (z. B. mithilfe von Grammatikregeln) aufgelöst werden; dieses Regelwissen kann vergleichsweise einfach auch Rechnersystem zur Verfügung gestellt werden. Andere Referenzen erschließen sich hingegen nur aus dem (sprachlichen) Kontext des Gesagten. Die Computerlinguistik beschäftigt sich mit der Auflösung Referenzen in natürlicher Sprache, insbesondere mit explizite Korreferenzen zwischen Nominalphrasen. Zwei oder mehr Phrasen, die sich auf dieselbe Entität beziehen, bezeichnet man als korreferent. Eine eingehende Diskussion dieser Problematik befindet sich in Abschnitt 2.3.4.7. Die Analyse von Korreferenzen ist die Grundlage vieler weiterführender Sprachanalysen. Sie ist beispielsweise von Bedeutung, um verstehen zu können, wie viele und welche Entitäten im Diskurs genannt werden und an welchen Stellen sie referenziert werden. Für die Programmierung mit natürlicher Sprache wird dieses Wissen unter anderem dafür benötigt, die natürlichsprachlichen Ausdrücke auf die richtigen Objekte (beispielsweise API-Klassen oder Objekte und Realwelt-Objekte) abzubilden.

7.5.1 Korreferenzanalyse von Nominalphrasen

Für ein Vorgehen zur Korreferenzanalyse von Nominalphrasen muss zunächst geklärt werden, welche Arten von Korreferenzen analysiert werden sollen. In Abschnitt 2.3.4.7 wurde bereits erläutert, dass man bei Korreferenzen zwischen Anaphern und Kataphern unterscheidet. Anaphern stellen einen Rückverweis dar; das bedeutet, der referenzierende Ausdruck folgt im Diskurs nach der referenzierten Entität (*Referent*), wie in „go to the door and open it“. Im Gegensatz dazu stellen Kataphern einen vorwärts gerichteten Bezug her, wie in „he was tired, so Tom went to bed“. Aus folgenden Gründen beschränken sich die meisten Ansätze zur Korreferenzanalyse auf die Analyse von Anaphern: Zum einen treten Anaphern in den meisten natürlichen Sprachen deutlich häufiger auf als Kataphern[30]. Zum anderen entstehen viele zusätzliche potenzielle Mehrdeutigkeiten, wenn für jeden referenzierenden Ausdruck angenommen würde, dass dieser eine Anapher oder eine Katapher darstellen kann. Analysen werden dadurch komplexer und fehleranfälliger.

Auch der für ProNat entwickelte Agent beschränkt sich daher auf Anaphern. Bei der Form des referenzierenden Ausdrucks gibt es in natürlicher Sprache viele Varianten. Die einfachste stellt die Verwendung von Pronomen dar, wie in den zuvor gegebenen Beispielen. Da diese Form zudem die am häufigsten verwendete darstellt, fokussieren sich die meisten Ansätze darauf. Im hier vorgestellten Ansatz soll jedoch zusätzlich eine weitere Form analysiert werden, die sogenannte *Identität*. Als Identität bezeichnet man Ausdrücke, die im Diskurs sowohl referenzierend wirken als auch als Referent verwendet werden können. Das bedeutet, Identitäten sind alle expliziten namentlichen Nennung derselben Entität. Beispielsweise könnte der frühere Präsident der Vereinigten Staaten von Amerika, George W. Bush im Diskurs unter seinem Namen oder unter der Bezeichnung *the former president of the US* genannt werden; wird hingegen mithilfe des Personalpronomens *he* Bezug genommen, handelt es sich um eine pronominale Anapher (oder Katapher). Viele Identitäten können nur anhand des Kontextes und mit Weltwissen als solche identifiziert werden (siehe Abschnitt 2.3.15). Der hier vorgestellte Agent soll daher Identitäten mit hoher Präzision erkennen, gegebenenfalls zulasten der Ausbeute; eine vollständige Referenzauflösung für Identitäten wird nicht angestrebt.

In der Literatur werden häufig nur einfach Identitäten, das heißt lexikalisch identische Ausdrucke, als solche erkannt. Zudem beschränken sich die meisten Ansätze wie zuvor erwähnt auf die Auflösung von Anaphern. Für die sprachlichen Analysen kommen häufig Zerteiler und Abhängigkeitsgraphen zum Einsatz. Da diese Werkzeuge nur für Textdokumente belastbare Ergebnisse erzeugen, beschränken sich die meisten Ansätze zur Korreferenzanalyse auf die Analyse geschriebener natürlicher Sprache. Ein bekannter Vertreter dieser Kategorie ist der Ansatz von Lee et al. [Lee+11]. Dieser stützt sich auf syntaktische Voranalysen der Sprache und verwendet eine konfigurierbare Sieb-Architektur. Durch die Einbindung in das Computerlinguistik-Fließband der *Stanford University* (*Stanford Core NLP*[31]) ist dieses Korreferenzanalyse-Werkzeug zudem weit verbreitet. Moderne Ansätze verwenden bestärkendes Lernen (engl. *reinforcement learning*), neuronale Netze und vortrainierte Sprachmodelle, wie das *BERT*-Modell [KG19; Fei+19]. Ansätze, die gezielt für gesprochene Sprache

[30] Da die Kataphorik häufig als Stilmittel eingesetzt wird, um beispielsweise dem Leser die handelnde Entität zunächst vorzuenthalten, gilt dies insbesondere für spontane gesprochene Sprache, wie sie im Kontext von ProNat angenommen wird.

[31] Stanford CoreNLP: `https://stanfordnlp.github.io/CoreNLP/`, zuletzt besucht am 24.02.2021.

entwickelt wurden, betrachten explizit Dialoge mit häufigen Sprecherwechseln und jeweils kurzen Äußerungen [SM03; TA04]. Dementsprechend beschäftigen sich diese Ansätze hauptsächlich mit der Auflösung der Referenzen, die sich auf die Dialogteilnehmer beziehen (z. B. *I*, *you*, etc.).

7.5.2 Implementierung des Agenten zur Korreferenzanalyse von Nominalphrasen

Der für ProNat entwickelte Agent verwendet Heuristiken und adaptiert die Stanford'sche Sieb-Architektur [Hey16]. Als Basis für die Heuristiken werden lexikalische und syntaktische Eigenschaften der natürlichsprachlichen Äußerung verwendet. Zusätzlich wird – sofern vorhanden – der situative Kontext, der durch den entsprechenden Agenten aufgebaut wird, verwendet (siehe Abschnitt 7.4). Neben pronominalen Anaphern löst der Agent Identitäten auf, wobei ein Großteil der nötigen Analyseschritte für die Identitätserkennung bereits vom Kontext-Agenten geleistet wird. Neben den Wörtern der Äußerung verwendet der Agent für seine Analysen Wortart- und Chunk-Etiketten sowie semantische Rollen. Es bestehen somit Abhängigkeiten von den Ergebnissen der Vorverarbeitungsstufen zur seichten Sprachverarbeitung und zur Erkennung von semantischen Rollen (siehe Abschnitt 6.2 und Abschnitt 6.4). Zusätzlich werden die Ergebnisse des Kontext-Agenten verwendet. Diese sind jedoch optional; viele der Analysen funktionieren auch ohne Kontext-Informationen. Stehen keine Kontext-Informationen zur Verfügung, werden Analyseschritte, die diese erfordern, automatisch übersprungen. Im Folgenden werden das Vorgehen des Agenten und die verwendeten Siebe genauer erläutert.

Zunächst werden alle Nominalphrasen anhand der Chunk-Etiketten aus der Äußerung ausgelesen. Jede Nominalphrase ist zunächst ein möglicher Kandidat für einen referenzierenden oder referenzierten Ausdruck (Referent). Für jeden Kandidaten wird anschließend überprüft, ob es sich um einen referenzierenden Ausdruck handelt und auf welchen Referenten er sich gegebenenfalls bezieht. Das bedeutet, pro Kandidat für einen referenzierenden Ausdruck bilden alle anderen Nominalphrasen die Kandidatenmenge der potenziellen Referenten. Die Kandidatenmenge wird anschließend mithilfe der Siebe schrittweise reduziert. Siebe bilden jeweils eine zu untersuchende Eigenschaft des betrachteten Kandidaten bzw. der Elemente der Kandidatenmenge, wie zum Beispiel Genus oder Numerus, ab. Es gibt allgemeine Siebe, die zur Analyse aller Kandidaten eingesetzt werden können und spezialisierte Siebe, deren Analysen sich nur für bestimmte Arten von Referenzen eignen (näheres hierzu folgt). Hinsichtlich der Durchlässigkeit gibt es zwei Arten von Sieben, *harte* und *weiche* Siebe. Harte Siebe schließen bestimmte Kandidaten aus, während weiche Siebe den Konfidenzwert eines Kandidaten anpassen (beispielsweise durch die Multiplikation mit einem konstanten Faktor, falls eine bestimmte Bedingung zutrifft). Alle harten Siebe können auch als weiche Siebe konfiguriert werden; die Konfiguration erlaubt auch die Anpassung der Konfidenz-Faktoren der weichen Siebe. Zusätzlich kann die Reihenfolge der Siebe variiert werden, Siebe können entfernt und neue hinzugefügt werden.

Die Kandidaten durchlaufen die Siebe; alle Kandidaten, die nach Durchlaufen aller Siebe nicht entfernt wurden und einen Konfidenzwert größer 0,1 aufweisen, werden im Graphen als neue Korreferenz hinterlegt. Hierzu wird eine neue Kante vom Typ `contextRelation` zwischen dem referenzierenden Ausdruck (Quelle) und dem referenzierten Ausdruck (Senke) erzeugt. In den

Attributen der Kante werden zudem die Art der Referenz und die Konfidenz hinterlegt. Durch die Verwendung von weichen Sieben ist es möglich, dass ein referenzierender Ausdruck mehrere Referenten referenziert. Anhand der Konfidenz ist trotzdem meist eine eindeutige Interpretation möglich. Gleichzeitig werden aber auch alternative Interpretationen der Referenz erzeugt. Andere Agenten können diese nutzen, falls die Korreferenz mit der höchsten Konfidenz nicht dem gewünschten Ergebnis entspricht oder das Analyseergebnis aufgrund weiterer Informationsquellen umgedeutet werden soll.

Wie zuvor beschrieben, gibt es unterschiedliche Arten von Korreferenzen; es wurde bereits zwischen pronominalen Anaphern und Identitäten unterschieden. Diese beiden Gruppen können je nach Referent und Art des referenzierenden Ausdrucks noch weiter unterschieden werden. Die Kategorisierung der Referenzen dient dazu, je nach Kategorie unterschiedliche Sieb-Kombinationen für eine möglichst präzise Analyse der natürlichsprachlichen Äußerung einsetzen zu können. Bei Identitäten unterscheidet der Agent zur Korreferenzanalyse von Nominalphrasen anhand ihrer grammatikalischen Funktion im Satz. Da Referenten von Nominalphrasen gebildet werden, können sie entweder die Funktion eines Subjektes oder eines Objektes einnehmen; unterschieden werden dementsprechend Subjekt- und Objekt-Identitäten. Auch bei pronominalen Anaphern wird zwischen solchen unterschieden, die auf Subjekte oder Objekte referenzieren[32]. Die referenzierenden Ausdrücke sind bei diesen beiden Kategorien Pronomen der zweiten Person oder dritten Person Singular. Die referenzierten Ausdrücke (Referenten) können wiederum entweder gleichartige Pronomen oder entsprechende Entitäten sein. Zusätzlich werden noch folgende Kategorien unterschieden: Gruppen und Sprecher. Pronomen der ersten Person Singular und Plural beziehen sich auf Referenten der Kategorie *Sprecher*. Im Kontext der Programmierung mit natürlicher Sprache ist darunter der menschliche Diskurspartner zu verstehen. Verwendet dieser in einer Äußerung einen Ausdruck, der sich auf sich selbst bezieht wie in *bring the cup to **me***, entspricht der Ausdruck der Kategorie *Sprecher*. Zugehörige Referenten sind entweder wieder Pronomen, die den Sprecher repräsentieren, oder eine potenziell virtuelle Sprecherentität. Da sich menschliche Diskurspartner normalerweise nicht selbst benennen – insbesondere in imperativen Anweisungen – bleibt der Referent einer Korreferenz, die mit einem Pronomen der ersten Person gebildet wurde, zumeist anonym, wie in der bereits zuvor diskutierten Äußerung *bring the cup to **me***. In solchen Fällen wird später eine virtuelle Sprecherentität durch den Agenten erzeugt. Die letzte Kategorie bilden referenzierende Ausdrücke, die mithilfe von Pronomen der dritten Person Plural gebildet werden und sich auf Gruppen beziehen. Die Referenten dieser Kategorie können gleichartige Gruppenpronomen oder mehrzahlige Subjekt- oder Objektentitäten sein. Ein Beispiel für eine Korreferenz dieser Kategorie ist die Äußerung *locate **the cup** and **the plate** and bring **them** to me*. Tabelle 7.14 gibt eine Übersicht über die Kategorien, zugehörige Referenten und referenzierende Ausdrücke.

[32] Diese Unterscheidung ist relevant, da Korreferenzen vorrangig zwischen Satzglieder mit derselben grammatikalischen Funktion gebildet werden [Hob78]. Das bedeutet, ein referenzierender Ausdruck, der als Satzobjekt fungiert, bezieht üblicherweise auch wieder auf einen Objekt. Daher werden Objekt- und Subjekt-Referenzen vom Agenten zur Korreferenzanalyse von Nominalphrasen unterschiedlich behandelt. Die Herstellung eines Bezuges zwischen referenzierenden Ausdruck und einem Referenten mit unterschiedlicher grammatikalischer Funktion ist dennoch möglich. Eine derartige Korreferenz wird nur als unwahrscheinlicher angesehen und erhält daher eine niedrigere Konfidenz.

Tabelle 7.14: Kategorisierung von Personalpronomen nach Bezugspunkten: Je Kategorie ist angegeben, welche Referenten potenziell referenziert werden können und wie die jeweiligen Pronomen gebildet werden. Zusätzlich sind alle referenzierenden Pronomen aufgelistet, die jeweils gebildet werden können.

Kategorie	potenzielle Referenten	Bildung	referenzierende Ausdrücke
Sprecher	Sprecherpronomen, (virtuelle) Sprecherentität	1. Per. Sing.:	*I*, *me*, *myself*, *mine*, *my*
		1. Per. Pl.:	*we*, *us*, *ourselves*, *ours*, *our*
Subjekt	Subjektpronomen (gl. Art), Subjektentitäten	2. Per.:	*you*, *yourself*, *yourselves* *your*, *yours*
		3. Per. Sing. Maskulin:	*he*, *him*, *himself*, *his*
		3. Per. Sing. Feminin:	*she*, *her*, *herself*, *hers*
Objekt	Objektpronomen, Objektentitäten	3. Per. Sing. Neutrum:	*it*, *itself*, *its*
Gruppen	Gruppenpronomen, mehrzahlige Subjekt- oder Objektentitäten	3. Per. Pl.:	*they*, *them*, *themself*, *themselves*, *their*, *theirs*

Wie zuvor erläutert, gibt es Siebe, die allgemeingültig bei allen Arten von Korreferenz eingesetzt werden können, und solche, die nur für bestimmte Kategorien geeignet sind. In Abbildung 7.6 ist diese Zuordnung dargestellt, wie sie in der Standardkonfiguration des Agenten vorgesehen ist[33]. Ein ausgefülltes Feld bedeutet, dass das entsprechende Sieb für Korreferenzen der jeweiligen Kategorie eingesetzt wird; Siebe, die bei allen Kategorien Verwendung finden, sind allgemeingültige Siebe. Kursiv geschriebene Siebe sind weiche Siebe, die übrigen harte (in der Standardkonfiguration). Siebe, die Kontext-Informationen verwenden, werden mit einem Stern (*) gekennzeichnet. Wie zuvor erläutert, werden diese nicht verwendet, wenn keine Kontext-Informationen verfügbar sind. Nachfolgend wird die Funktionsweise der einzelnen Siebe kurz erläutert.

Reihenfolge: Dieses harte Sieb entfernt alle Kandidaten aus der Kandidatenmenge, die in der Äußerung *nach* dem referenzierenden Ausdruck stehen. Der Grund hierfür ist, dass nur pronominale Anaphern aber keine Kataphern analysiert werden sollen. Die Identitätsbeziehung ist hingegen bidirektional. Falls eine Identitätsbeziehung zwischen dem betrachteten referenzierenden Ausdruck und einer nachfolgenden Nominalphrase besteht, so wird dieser zu einem späteren Zeitpunkt gefunden (wenn die nachfolgende Nominalphrase als referenzierender Ausdruck betrachtet wird).

Gleiche Kategorie: Dieses ebenfalls harte Sieb überprüft, ob die möglichen Referent-Kandidaten hinsichtlich der Kategorie (siehe Tabelle 7.14) mit dem betrachteten referenzierenden Ausdruck übereinstimmen; alle anderen werden aus der Kandidatenmenge entfernt. Wird beispielsweise das Personalpronomen *it* als referenzierender Ausdruck betrachtet, verbleiben nur Kandidaten in der Kandidatenmenge, die entweder ebenfalls ein Objektpronomen (*it*, *itself* und *its*) oder eine Objektentität darstellen.

[33] Der Agent erlaubt eine freie Konfiguration der Sieb-Zusammenstellung. Es können also prinzipiell alle Siebe für alle Kategorien verwendet werden.

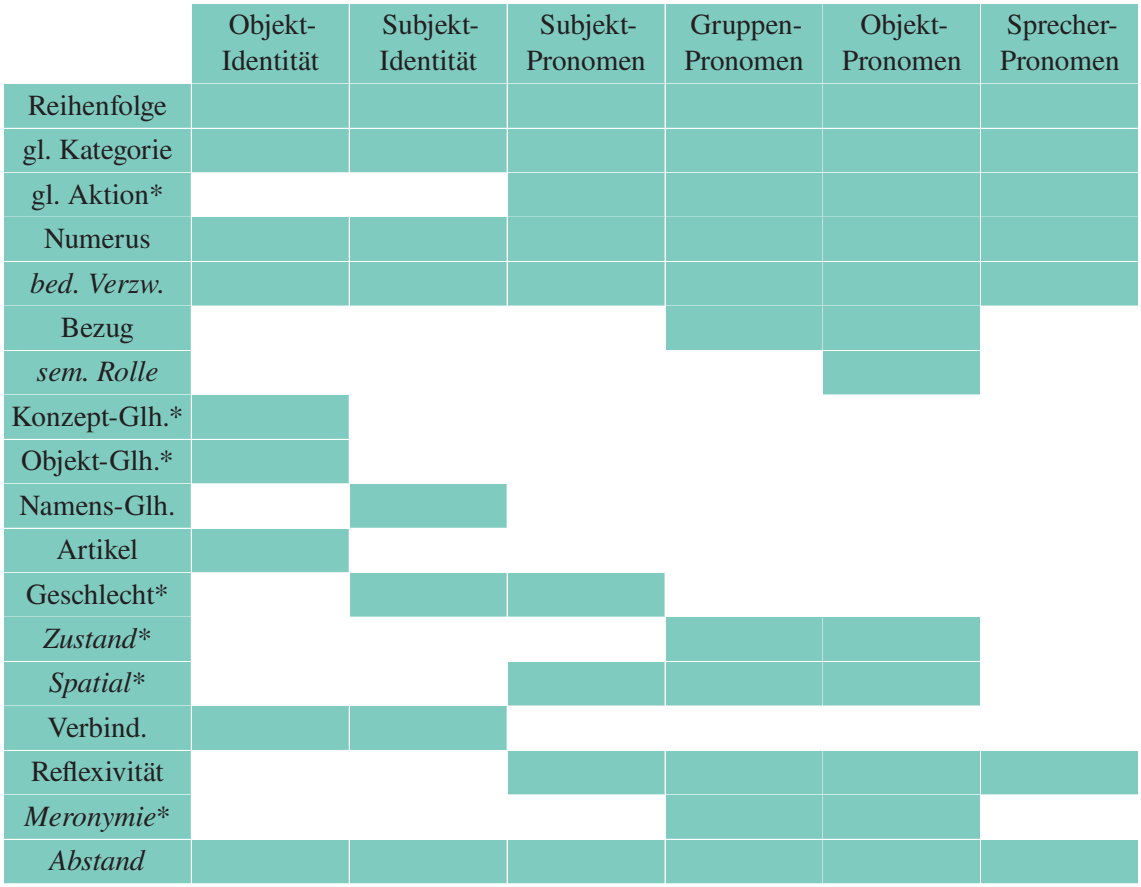


Abbildung 7.6: Zuordnung von Sieben zu Referenz-Arten: Die Darstellung zeigt den Zusammenhang zwischen Arten von Referenzen (Spalten) und Sieben (Zeilen), die in der Standardkonfiguration des Agenten verwendet werden. Ein ausgefülltes Feld symbolisiert die Verwendung eines Siebes für eine bestimmte Art. Siebe, die Kontext-Informationen verwenden, sind mit einem Stern (*) gekennzeichnet.

Gleiche Aktion: Dieses harte Sieb entfernt alle Kandidaten aus der Kandidatenmenge, die Teil derselben Aktion wie der betrachtete referenzierende Ausdruck sind. Das Sieb gilt nur für Pronomen, die als referenzierender Ausdruck verwendet werden. Unter einer Aktion ist hier ein Prädikat mit allen zugehörigen *A**-Rollen (Handelnder, behandeltes Objekt, etc., siehe Abschnitt 2.3.4.8) zu verstehen, wie durch den Erkenner semantischer Rollen annotiert (siehe Abschnitt 6.4). Das Sieb sorgt also dafür, dass referenzierende Ausdruck und der Referent nicht zwei unterschiedliche *A**-Rollen innerhalb derselben Aktion einnehmen können. In Äußerungen wie *get the cup and fill water in it* wird somit eine Korreferenz zwischen *it* und *water* ausgeschlossen. Eine Ausnahme bilden Reflexiv- und Possessivpronomen für die dieses Sieb nicht angewandt wird.

Numerus: Dieses harte Sieb, welches bei allen Kategorien referenzierender Ausdrücke angewendet wird, entfernt alle Kandidaten aus der Kandidatenmenge, deren Numerus nicht mit dem Numerus des betrachteten referenzierenden Ausdrucks übereinstimmt. In Äußerungen wie *get the cup and apples and slice them* wird somit eine Korreferenz zwischen *them* und *the cup* ausgeschlossen.

Bedingte Verzweigung: Dieses weiche Sieb sorgt dafür, dass Korreferenzen zwischen Nominalphrasen, die zu unterschiedlichen Gliedsätzen gehören (gemäß der in Abschnitt 7.6 gegebenen Definition), als eher unwahrscheinlich eingestuft werden; die entsprechenden Kandidaten erhalten

als neuen Konfidenzwert ihren vorherigen multipliziert mit dem Faktor 0,8. Das bedeutet, Korreferenzen zwischen einem Referenten der Teil eines *Dann*-Gliedsatzes und einem referenzierenden Ausdruck aus dem *Andernfalls*-Gliedsätze gelten als weniger wahrscheinlich. Da dieses Sieb von den Ergebnissen des Agenten zur Erkennung von bedingten Verzweigungen abhängt, kann es nur eingesetzt werden, falls die entsprechenden Informationen generiert werden. Falls der Agenten zur Erkennung von bedingten Verzweigungen nicht Teil der Konfiguration ist oder aus einem anderen Grund keine bedingten Verzweigungen extrahiert wurden, lässt dieses Sieb alle Kandidaten unverändert passieren.

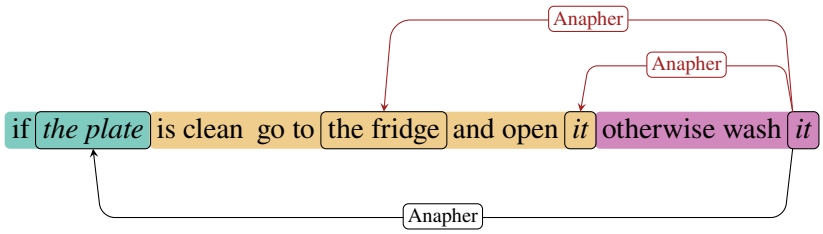
Beispiel:
Nutzung erkannter verbalisierter bedingter Verzweigungen zur Analyse von Korreferenzen

In diesem Beispiel soll demonstriert werden, wie die Struktur erkannter verbalisierter bedingter Verzweigungen genutzt wird, um die Wahrscheinlichkeit von Korreferenzkandidaten zu schätzen. Das Beispiel bezieht sich auf die in Abschnitt 7.6 beschriebenen Analysen des Agenten zur Erkennung von bedingten Verzweigungen.

Natürlichsprachliche Sequenz

if the plate is clean go to the fridge and open it otherwise wash it

Potenzielle Korreferenzen



Erläuterung

Für dieses Beispiel sei der betrachtete referenzierende Ausdruck das Personalpronomen *it* am Ende der Äußerung. Als Kandidaten für einen zugehörigen Referenten kommen die Nominalphrasen *the plate*, *the fridge* und *it* infrage. Die Erkennung verbalisierter bedingter Verzweigungen hat ergeben, dass es sich bei dem grün unterlegten Teil um den Konditionalsatz, beim orange unterlegten um den *Dann*-Gliedsatz und beim lila unterlegten um den *Andernfalls*-Gliedsatz handelt. Aus dem Kontext ist offensichtlich, dass die Korreferenz zwischen *it* und *the plate* korrekt ist. Ein Hinweis darauf liefert die Zugehörigkeit der beiden Auftreten des Pronomens *it* und der Nominalphrase *the fridge* in unterschiedlichen Gliedsätzen. Da sich aber auch Beispiele konstruieren lassen, bei denen eine andere Interpretation der Referenzen richtig ist, wurde das Sieb als weiches Sieb implementiert.

Bezug: Dieses harte Sieb entfernt alle Kandidaten aus der Kandidatenmenge, die in der Äußerung weit vor dem referenzierenden Ausdruck stehen und dementsprechend höchstwahrscheinlich in keinem Bezug stehen. Es folgt der Intuition, dass Korreferenzen üblicherweise zwischen nah beieinander stehenden Ausdrücken auftreten. Wird eine Entität im Diskurs an entfernten Stellen referenziert, verwenden Sprecher für gewöhnlich eine namentliche Nennung statt einer pronominalen Anapher. Um die Entfernung in gesprochenen Äußerungen bemessen zu können, verwendet dieses Sieb die von der Vorverarbeitungsstufe zur seichten Sprachverarbeitung erzeugte Instruktionsnummer (siehe Abschnitt 6.2). Wörter mit derselben Instruktionsnummer bilden einen Gliedsatz, in dem eine Aktion oder ein Ereignis beschrieben wird. In der Standardkonfiguration entfernt das Sieb alle Kandidaten, die eine Instruktionsnummer aufweisen, die um sechs oder mehr verschieden von der Instruktionsnummer ist, die dem referenzierenden Ausdruck zugeordnet wurde. Das Sieb wird nur bei Gruppen- und Objektpronomen angewendet, da nur bei diesen Kategorien nach Anwendung der anderen harten Siebe viele Kandidaten verbleiben, sodass eine Filterung hinsichtlich einer maximalen Entfernung nötig wird. Der Grund hierfür ist, dass im Englischen die meisten Entitäten sächlich sind und die beiden Kategorien referenzierende Pronomen der 3. Person (Neutrum) betrachten. Das bedeutet, für diese Pronomen gibt es besonders viele Kandidaten.

Semantische Rolle: Dieses weiche Sieb passt die Konfidenz der Kandidaten an, je nachdem welche *A**-Rolle sie einnehmen (siehe Abschnitt 2.3.4.8). Dabei wird die Konfidenz aller Kandidaten, die *nicht* dieselbe *A**-Rolle wie der betrachtete referenzierende Ausdruck einnehmen, mit dem Faktor 0,7 multipliziert (in der Standardkonfiguration). Die Intuition hinter diesem Vorgehen geht auf einen der ersten Ansätze zur Korreferenzanalyse von Jerry R. Hobbs zurück [Hob78]. Er beobachtete, dass Korreferenzen mit einer höheren Wahrscheinlichkeit zwischen Ausdrücken bestehen, die im Satz die gleiche grammatikalische Funktion bekleiden. Bei Äußerung wie *the dog chased the cat then it fell asleep* erachten Menschen intuitiv die Korreferenz zwischen *it* und *the dog* aufgrund der Satzstruktur als wahrscheinlicher als die zwischen *it* und *the cat*. Um die grammatikalische Funktion herzuleiten, analysierte Hobbs Syntaxbäume. Das hier verwendete Sieb verwendet semantische Rollen, das zugrundeliegende Prinzip ist jedoch dasselbe. Wie das Sieb *gleiche Aktion* wird das hier beschriebene Sieb nicht für Possessivpronomen angewandt.

Konzept-Gleichheit: Dieses harte Sieb wird bei der Analyse von Objekt-Identitäten eingesetzt und erfordert Kontext-Informationen. Das Sieb behält nur Kandidaten bei, die entweder dasselbe Konzept oder ein Unterkonzept des referenzierenden Ausdrucks darstellen. Auf diese Weise gelingt es in Äußerungen wie *get the popcorn and the juice and pass me the food* die Identitätsbeziehung zwischen *the food* und *the popcorn* herzustellen.

Objekt-Gleichheit: Dieses harte Sieb, welches für Objekt-Identitäten verwendet wird, prüft, ob zwei Kandidaten die gleiche Entität darstellen; alle anderen werden aus der Kandidatenmenge entfernt. Für jedes Kandidatenpaar wird überprüft, ob sie in alle zusätzlichen Eigenschaften (wie beispielsweise beschreibenden Adjektiven) übereinstimmen. Das Sieb verwendet hierzu die Entitätseigenschaften, die durch den Agenten zur Modellierung des sprachlichen Kontextes erzeugt wurden und ist somit von diesem abhängig. Ein referenzierender Ausdruck ist auch dann ein valider Kandidat wenn er nur eine Teilmenge der Eigenschaften des Referenten besitzt wie in *locate **the red cup** bring **the cup** to me*.

Namens-Gleichheit: Dieses harte Sieb prüft für Subjekt-Identitäten, ob die konkreten Benennungen der Kandidaten ähnlich sind, alle anderen Kandidaten werden aus der Kandidatenmenge entfernt. Ähnliche Nennungen stellen hierbei Teilphrasen dar, wie sie auftreten wenn bei Namen Vor- oder Nachnamen entfallen und Titel hinzugefügt werden.

Artikel: Dieses harte Sieb entfernt bei Objekt-Identitäten alle Kandidaten aus der Kandidatenmenge, die einen unbestimmten Artikel besitzen. Das Sieb folgt der Intuition, dass unbestimmte Artikel verwendet werden, um Entitäten in den Diskurs einzuführen, nicht aber um auf vorangegangene Nennungen zu referenzieren. Der Unterschied kann an den Äußerungen *get the cup* und *get a cup* nachvollzogen werden. Während erste eine Identitätsbeziehung zu einer vorangegangenen Äußerung wie *locate a cup* erzeugen könnte, ist dies für die zweite Äußerung nicht möglich.

Geschlecht: Dieses harte Sieb betrachtet das Geschlecht der potenziellen Referenten und entfernt alle Kandidaten, deren Genus nicht mit dem des referenzierenden Ausdrucks übereinstimmen. Das Sieb betrachtet nur das Femininum und Maskulinum; das bedeutet, es werden nur Korreferenzen der Kategorie Subjekt-Identität und Pronomen betrachtet. Die Information über das Geschlecht einer Entität entnimmt das Sieb den Entitätseigenschaften, die durch den Agenten zur Modellierung des sprachlichen Kontextes erzeugt wurden (siehe Abschnitt 7.4). Das Geschlechter-Sieb ist somit ebenfalls von Kontextinformationen abhängig und wird beim Fehlen dieser nicht verwendet.

Erfüllbarkeit des Zustands: Dieses weiche Sieb überprüft für Gruppen- und Objektpronomen, ob eine Zustandsänderung, die durch ein Ereignis, an dem der referenzierende Ausdruck beteiligt ist, für den Kandidaten-Referenten möglich ist. In der Äußerung *get me **the full cup** next to **the empty glass** and empty **it*** ist es beispielsweise wahrscheinlicher, dass sich das Pronomen *it* auf *the full cup* statt *the empty glass* bezieht, da das Glas im Gegensatz zur Tasse bereits leer ist. Für alle Kandidaten, auf die dies nicht zutrifft, wird die Konfidenz um den Faktor 0,66 verringert (in der Standardkonfiguration). Das Sieb verwendet für diese Analyse die Zustandsinformationen des Agenten zur Modellierung des sprachlichen Kontextes.

Spatial Deixis: Dieses weiche Sieb prüft bei Subjekt-, Gruppen- und Objektpronomen, ob Kandidaten-Referenten im Diskurs nur genannt werden, um die räumliche Ordnung anderer Entitäten zu beschreiben; die Konfidenz aller Kandidaten, auf die dies zutrifft, wird mit dem Faktor 0,25 multipliziert. In Äußerungen wie *take **the cup** next to **the popcorn** and pass **it** to me* wird die Entität *the popcorn* nur erwähnt, um *the cup* näher zu beschreiben; eine Korreferenz zwischen *it* und *the cup* ist somit deutlich wahrscheinlicher.

Entitäten stehen bereits in Verbindung: Dieses harte Sieb entfernt bei Objekt- und Subjekt-Identitäten alle Kandidaten aus der Liste, bei denen bereits eine Korreferenzbeziehung aus einem früheren Analyseschritt bekannt ist. Dies ist nötig, da Identitätsbeziehungen (wie zuvor diskutiert) bidirektional und transitiv sind und somit einer iterativen Analyse aller Kandidatenpaare Dopplungen auftreten können.

Reflexivität: Dieses harte Sieb prüft für Reflexivpronomen, ob potenzielle Referenten im unmittelbaren Kontext geäußert werden; alle anderen Kandidaten werden aus der Kandidatenmenge entfernt. Dieses Vorgehen kann darauf zurückgeführt werden, dass mithilfe von Reflexivpronomen bei einem Ereignis oder einer Aktion ein Rückbezug auf den Handlungsträger stattfindet wie beispielsweise

in ***John*** *washes* ***himself***. Der Bezugsrahmen für Reflexivpronomen wird daher auf Entitäten mit derselben Instruktionsnummer beschränkt (siehe Abschnitt 6.2).

Meronymie: Dieses weiche Sieb wird ausschließlich für Possessivpronomen eingesetzt. Possessivpronomen stellen einen Sonderfall dar, da sie immer an eine benannte Entität gebunden sind. Sind zusätzlich Teil-Ganzes-Beziehungen (Meronymie) zwischen den Entitäten bekannt (siehe Abschnitt 7.4), können die Possessivpronomen genutzt werden, um die Konfidenzen der potenziellen Referenten anzupassen. Korreferenzen zwischen einem Possessivpronomen (und der zugehörigen Entität), das einen *Teil* darstellt, und einem referenzierten Ausdruck, der das *Ganze* ist, gelten dabei als wahrscheinlicher. Die Konfidenzen aller anderen Kandidaten werden mit dem Faktor 0,75 multipliziert. In der Äußerung *open* ***the door*** *of* ***the fridge*** *take out* ***the milk*** *then close* ***its door*** kann so gefolgert, dass sich *its door* wahrscheinlicher auf *the fridge* als auf *the milk* bezieht.

Abstand: Dieses weiche Sieb wird für alle Kategorien von referenzierenden Ausdrücken angewandt und bewertet die potenziellen Referenten anhand des Abstands zum betrachteten referenzierenden Ausdruck. Die Intuition hinter diesem Vorgehen ist, dass die Wahrscheinlichkeit für eine Korreferenz zwischen zwei Ausdrücken mit dem Abstand voneinander abnimmt. Das Sieb betrachtet alle Kandidaten, die den höchsten (und gleichen) Wert für die Konfidenz teilen. Unter diesen Kandidaten (mit der höchsten Konfidenz), gilt derjenige als wahrscheinlichster, der dem referenzierenden Ausdruck am nächsten steht; alle übrigen Kandidaten werden um einen konstanten Faktor reduziert, der sich aus der Anzahl der verbliebenen Kandidaten ableitet. Da diese Berechnung von der Anzahl der verbleibenden Kandidaten abhängt und alle Konfidenzen bis auf eine angepasst werden müssen, empfiehlt es sich das Abstands-Sieb als letztes Sieb einzusetzen (so wie es auch die Standardkonfiguration vorsieht). Die Ausgabe des Abstands-Siebes entspricht dementsprechend dem finalen Analyseergebnis, das wie zuvor beschrieben im ProNat-Graphen hinterlegt wird.

Beispiel:
Zusammenspiel der Siebe zur Ermittlung von Korreferenzen

In diesem Beispiel soll das Zusammenspiel der Siebe zur Ermittlung der wahrscheinlichsten Korreferenz für einen betrachteten referenzierenden Ausdruck demonstriert werden. Das Beispiel nimmt Bezug auf die Abbildung 7.7, in welcher die Kandidaten nacheinander eine beispielhafte Sieb-Konfiguration für Objektpronomen durchlaufen. Für die nachfolgende Äußerung soll für den referenzierenden Ausdruck *it* (grün in der Abbildung) ermittelt werden, welcher der potenziellen Referenten (orange) am wahrscheinlichsten der korrekten Referenz entspricht.

Natürlichsprachliche Sequenz

Robo put the plates on the table go to the fridge open the fridge and take the water out of the fridge then go to the table if the cup on the table isn't empty bring the cup to me otherwise take the cup and fill the water in it then come to me

Erläuterung

Das erste Sieb in der Beispielkonfiguration ist das *Reihenfolge*-Sieb. Diese entfernt Nominalphrasen, die nach dem referenzierenden Ausdruck auftreten, im Beispiel das letzte vorkommende *me*. Das Sieb *Gleiche Kategorie* entfernt Nominalphrasen, die nicht derselben Kategorie angehören wie der referenzierende Ausdruck (siehe Tabelle 7.14), im Beispiel das andere vorkommende *me*. Anschließend wird im Sieb *bedingte Verzweigung* die Konfidenz aller Kandidaten, die in einem anderen Gliedsatz einer Bedingungsstruktur auftreten als der referenzierende Ausdruck, verringert. Im Beispiel trifft das nur auf *the cup* im *Dann*-Gliedsatz *bring the cup to me* zu. Das Sieb *Gleiche Aktion* schließt alle Kandidaten aus, die in derselben Aktion wie der referenzierende Ausdruck, aber in einer anderen *A**-Rolle auftreten (siehe Abschnitt 2.3.4.8). Im Beispiel betrifft dies das letzte vorkommende *the water*. Das *Numerus*-Sieb schließt im Beispiel alle Entitäten im Plural aus; das bedeutet, *the plates* wird aus der Kandidatenmenge entfernt. Anschließend werden durch das Sieb *Bezug* alle Kandidaten ausgeschlossen, die in der Äußerung zu weit entfernt vom referenzierenden Ausdruck auftreten (Unterschied der Instruktionsnummern größer sechs). Im Beispiel betrifft dies alle Kandidaten vor dem ersten Vorkommen von *the water*. Das Sieb *semantische Rolle* wertet alle Kandidaten ab, die eine andere semantische Rolle bekleiden als der referenzierende Ausdruck. Für die verbleibenden Kandidaten im Beispiel hat das den Effekt, dass nur die Konfidenzen von *the water* und der letzten beiden Vorkommen von *the cup* unverändert bleiben. Da in der Äußerungen zwar durch das Ereignis *fill* eine Zustandsänderung der Entität *the cup* herbeigeführt wird, diese Änderung aber im Kontext möglich ist, verändert das Sieb *Zustand* die vorhandenen Konfidenzen nicht. Das letzte Vorkommen *the table* wird im Beispiel nur verwendet, um die nachfolgende Nominalphrase *the cup* näher zu beschreiben. Das Sieb *Spatial Deixis* verringert die Konfidenz des Kandidaten entsprechend. Zuletzt wird das Sieb *Abstand* verwendet, um die finale Entscheidung für einen Kandidaten zu treffen. Nur zwei weisen einen Konfidenzwert von 1,0 auf, *the water* und das letzte Vorkommen von *the cup*. Letzterer weist einen geringeren Abstand zum referenzierenden Ausdruck auf, seine Konfidenz bleibt entsprechend unverändert; die Konfidenz aller anderen Kandidaten wird hingegen verringert. Letztlich wird also die Nominalphrase *the cup* (aus der Phrase *otherwise take the cup and fill water in it*) als wahrscheinlichster Referent ausgewählt. Diese Wahl ist für diese Beispieläußerung die richtige.

7.5.3 Evaluation des Agenten zur Korreferenzanalyse von Nominalphrasen

Die Evaluation des Agenten zur Korreferenzanalyse von Nominalphrasen wird anhand von Transkriptionen zu den Szenarien eins bis sieben des ProNat-Korpus durchgeführt. Einige der Äußerungen zu den ersten fünf Szenarien wurden während der Entwicklung des Agenten verwendet (siehe Abschnitt 7.5.2). Dies betrifft jeweils die ersten zehn Transkriptionen zu den Szenarien eins bis drei und jeweils die ersten neun zu den Szenarien vier und fünf; dieser Teil des Korpus wird gesondert

Robo put the plates on the table go to the fridge open the fridge and take the water out of the fridge then go to the table if the cup on the table isn't empty bring the cup to me otherwise take the cup and fill the water in it then come to me

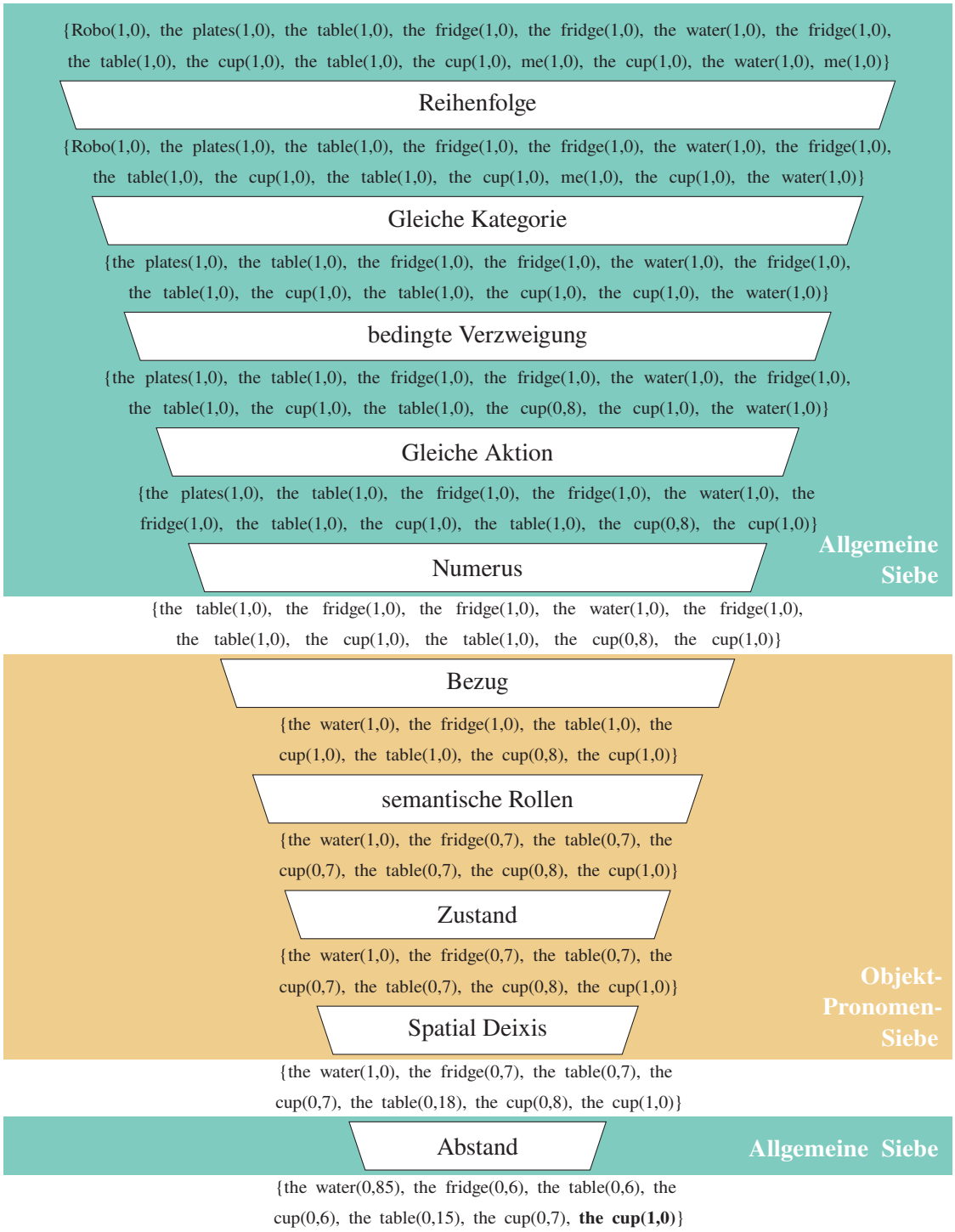


Abbildung 7.7: Beispiel zur Korreferenzanalyse von Nominalphrasen: Dargestellt ist eine beispielhafte Äußerung, welche die Sieb-Architektur durchläuft, um für die Nominal-Phrase *it* als referenzierenden Ausdruck den passenden Referenten zu identifizieren.

Tabelle 7.15: Ergebnisse der Evaluation des Agenten zur Korreferenzanalyse von Nominalphrasen: Die Tabelle zeigt die Ergebnisse für die Basisvariante (Basis) und bei Verwendung des sprachlichen Kontextes (Kont.). Außerdem werden die erzielten Ergebnisse des Korreferenzanalysewerkzeugs der *Stanford University* (Stan.) als Referenzwerte angegeben.

	Präzision			Ausbeute			F_1		
Korpus	Stan.	Basis	Kont.	Stan.	Basis	Kont.	Stan.	Basis	Kont.
S_{1-5} (Entw.)	0,743	0,922	0,936	0,491	0,848	0,920	0,591	0,884	0,928
S_{1-5} (Rest)	0,773	0,841	0,890	0,544	0,786	0,885	0,639	0,812	0,887
S_{6+7}	0,728	0,797	0,848	0,607	0,786	0,859	0,662	0,791	0,854
Gesamt	0,746	0,837	0,881	0,561	0,799	0,881	0,640	0,818	0,881

betrachtet. Die Äußerungen zu den Szenarien sechs und sieben wurden erst nach der Entwicklung des Agenten aufgezeichnet; daher werden auch diese in der Evaluation als gesonderter Datensatz betrachtet. Der Datensatz der bereits zur Entwicklung verwendeten Äußerungen umfasst 48, der Datensatz der übrigen Äußerungen zu den Szenarien eins bis drei umfasst 98 und der Datensatz der Äußerungen zu den Szenarien sechs und sieben umfasst 21 Transkriptionen.

Für alle Transkriptionen wurde ein Goldstandard erstellt, indem für jede Äußerung alle referenzierenden Ausdrücke und zugehörigen Referenten manuell annotiert wurden. Es wurden zwei Varianten des Agenten evaluiert: eine Basis-Variante und eine zweite Variante, die Kontext-Informationen verwendet. Um die Ergebnisse der Basis-Variante zu erzeugen, wurde zunächst das vollständige Vorverarbeitungsfließband durchlaufen und anschließend der Agent zur Korreferenzanalyse von Nominalphrasen genau einmal ausgeführt. Da auf diese Weise keine Informationen über den sprachlichen Kontext vorliegen, verwendet der Agent nur die Siebe, die in Abbildung Abbildung 7.6 nicht mit einem Stern gekennzeichnet sind[34]. Für die zweite Variante wurde ebenfalls das Vorverarbeitungsfließband ausgeführt, anschließend der Agent zur Erkennung von bedingten Verzweigungen. Zuletzt wurden die Agenten zur Modellierung des sprachlichen Kontexts und zur Korreferenzanalyse von Nominalphrasen so lange im Wechsel ausgeführt, bis (über zwei Iterationen) keine Änderung am Graphen festzustellen waren. Zum Vergleich wurde das Stanford'sche Korreferenzanalysewerkzeug auf die Transkriptionen angewendet [Lee+11]. Dieses ist auf textuelle Dokumente optimiert und verwendet syntaktische Analysen, wie Syntaxbäume und Abhängigkeitsgraphen, die nur schlecht auf gesprochene Äußerungen angewendet werden können. Der Vergleich ist damit nur bedingt aussagekräftig; es kann jedoch gezeigt werden, warum eine Sonderlösung für gesprochene Äußerungen nötig ist und nicht stattdessen ein Standardwerkzeug verwendet werden kann.

Die Ergebnisse der Evaluation sind in Tabelle 7.15 aufgeführt. Wie zu erwarten, übertrifft die für ProNat entwickelte Sonderlösung das Stanford'sche Standardwerkzeug auf allen Datensätzen sowohl hinsichtlich der Präzision als auch der Ausbeute. Insgesamt erzielt das Stanford'sche Standardwerkzeug für das F_1-Maß einen Wert von 0,640. Die Basis-Variante des Agenten erreicht hingegen 0,818 und die Variante, die Kontextinformationen verwendet, sogar 0,881. Dieses sehr gute Ergebnis ist im Fall der Basis-Variante vor allem auf eine sehr gute Präzision zurückzuführen.

[34] Die Basis-Variante verwendet darüber hinaus auch keine Informationen, die durch andere Agenten, wie etwa dem Agenten zur Erkennung von bedingten Verzweigungen, zur Verfügung gestellt werden.

Die Kontext-Variante steigert vor allem die Ausbeute deutlich; gleichzeitig gelingt ein weiterer Zuwachs an Präzision. Betrachtet man die Ergebnisse je Datensatz, zeigt sich, dass für den Entwicklungsdatensatz erwartungsgemäß die besten Ergebnisse erzielt wurden (F_1: 0,927). Für die restlichen Transkriptionen zu den Szenarien eins bis fünf zeigt sich eine leichtere Verschlechterung (F_1: 0,887). Die Ergebnisse für die Transkriptionen zu den Szenarien sechs und sieben sind nochmals etwas schlechter. Auch diese Entwicklung war zu erwarten, da diese Transkriptionen komplexere Beschreibungen enthalten als der Großteil zu den Szenarien eins bis fünf. Trotzdem wird auch hier für die Korreferenzanalyse für das F_1-Maß ein sehr guter Wert von 0,854 erreicht. Insgesamt zeigt die Evaluation, dass ein auf gesprochene Sprache angepasster Ansatz zur Korreferenzanalyse bessere Ergebnisse erzielen kann als ein Standardwerkzeug. Außerdem ist die Verwendung des sprachlichen Kontextes vorteilhaft.

7.6 Synthese von Kontrollstrukturen

Kontrollstrukturen sind neben Anweisung das wesentliche Konstrukt nahezu aller Programmiersprachen. Ohne Kontrollstrukturen würden Programme nur aus sequenziellen Anweisungsfolgen bestehen; der wirkliche Nutzen von Programmierung entsteht erst durch Kontrollstrukturen, wie bedingten Verzweigungen und Schleifen. Dasselbe gilt für die Programmierung mit natürlicher Sprache. Sollen Endnutzer befähigt werden, Programme in Alltagssprache zu formulieren, so müssen Kontrollstrukturen in natürlichsprachlichen Äußerungen erkannt und in ihre programmatischen Pendants überführt werden. Die in Programmiersprachen üblichen Kontrollstrukturen sind lediglich Formalisierungen menschlicher Denkprozessen. Schleifen sind beispielsweise nur die Formalisierung von Wiederholungen (z. B. von Ereignissen) und bedingte Verzweigungen formalisieren Fallunterscheidungen (die unter gewissen Voraussetzungen getroffen werden). Das legt die Vermutung nahe, dass auch in natürlicher Sprache Konstrukte existieren, die Kontrollstrukturen beschreiben. Tatsächlich existieren bestimmte Formulierungen und Satzstrukturen, die zur Formulierung von Kontrollstrukturen verwendet werden. Allerdings sind die Varianten in natürlicher Sprache vielfältiger. Folgen Kontrollstrukturen in Programmiersprachen der jeweils fest vorgegebenen Syntax, so können Kontrollstrukturen in natürlicher Sprache nahezu beliebig formuliert werden; häufig können sie nur anhand der Semantik (oder ggf. der Pragmatik) interpretiert werden.

Aufgrund dieser Komplexität ist der Umgang mit Kontrollstrukturen in der Literatur sehr unterschiedlich. Ansätze, die sich als Sprachschnittstelle zum Quelltext-Diktat verstehen, umgehen das Problem der semantischen Interpretation der Nutzereingabe [Pri+00; Beg04; BG05; DFN06]. Stattdessen erwarten diese Ansätze vom Nutzer syntaktisch korrekt diktierten Quelltext, einschließlich etwaiger Kontrollstrukturen. Bei Ansätzen, die das Ziel der Endnutzer-Programmierung mit natürlicher Sprache verfolgen, gibt es im Wesentlichen zwei Strömungen. Zur ersten gehören Ansätze, die bei jeder Eingabe die Verbalisierung einer Kontrollstruktur erwarten, normalerweise eine bedingte Verzweigung [LGS13; QMG15; BQ16]. Genauer gesagt erwarten Sie vom Nutzer, dass dieser zunächst eine Bedingung formuliert und anschließend die auszuführenden abhängigen Aktionen. Diese Art der fest vorgeschriebenen Formulierungen für bedingte Verzweigung werden als *Wenn-Dann-Rezepte* (engl. *If-Then Recipes*) bezeichnet. Ansätze der zweiten Strömung erlauben den

Einsatz von Kontrollstrukturen an beliebigen Stellen des verbalisierten Programms. Viele dieser Ansätze nutzen einen Sprach-Formalismus wie kombinatorische Kategorialgrammatiken (engl. *Combinatory Categorial Grammar*, kurz *CCG*), lexikalisch-funktionale Grammatiken (engl. *Lexikal-Functional Grammar*, kurz *LFG*) oder Phrasenkopf-getriebene Phrasenstruktur-Grammatiken (engl. *Head-driven Phrase Structure Grammar*, kurz HPSG), um die Semantik der natürlichsprachlichen Eingabe zu erfassen [KB82; PS94; ZC05; ZC07]. Die Formalismen bieten unter anderem auch Konstrukte zur Interpretation von Kontrollstrukturen. Ein Vertreter dieser Kategorie ist der Ansatz von Vadas und Curran [VC05]. Andere Ansätze verwenden Computerlinguistik-Fließbänder und interpretieren anschließend die natürlichsprachlichen Äußerungen anhand der Syntax [LL05a; MLL06; LH15]. Abgesehen von den Ansätzen zum Quelltext-Diktat, haben alle Ansätze gemein, dass sie ausschließlich geschriebene Sprache akzeptieren. Sowohl die *Wenn-Dann*-Rezepte, als auch die Sprach-Formalismen und die Syntax-basierten Ansätze sind auf Interpunktion und grammatikalisch korrekte Formulierungen angewiesen.

Für Pro$\mathcal{N}at$ wurden daher neue Ansätze zur Synthese von Kontrollstrukturen entwickelt [WHS18a; WHS18b]. Betrachtet wurden die Kontrollstrukturarten bedingte Verzweigung, Nebenläufigkeit und Schleife, für die jeweils ein eigener Pro$\mathcal{N}at$-Agent implementiert wurde. Die in den Agenten verwendeten Analysen sind zwar ebenfalls Syntax-basiert, jedoch leichtgewichtiger als vergleichbare Ansätze, da als Grundlage nur robuste Phrasen-*Chunks* verwendet werden. Zudem werden die natürlichsprachlichen Äußerungen nur partiell zerteilt, um nicht auf vollständig korrekte Eingaben angewiesen zu sein. Mithilfe von Korreferenzinformationen wird zudem eine einfache Interpretation der Semantik vorgenommen, um die Ergebnisse weiter zu verbessern. Im Folgenden werden die Agenten einzeln vorgestellt; dabei werden zunächst die linguistischen Hintergründe diskutiert, bevor die Implementierung und schließlich die Evaluation der Agenten beschrieben wird.

7.6.1 Bedingte Verzweigungen in natürlicher Sprache

Der Ansatz zur Erkennung von bedingten Verzweigungen basiert auf der Analyse wiederkehrender Satzstrukturen, die Bedingungen ausdrücken. Aus diesem Grund werden im Folgenden kurz die grundlegenden linguistischen Eigenschaften von Bedingungen in natürlicher Sprache diskutiert.

In der Linguistik gibt es keine einheitliche Definition für den Begriff *Bedingung*. Da die meisten Sprachen viele unterschiedliche Formen (und Formulierungen) zulassen, sind ohnedem nur generische Definitionen plausibel. Eine solche liefern Declerck und Reed für die englische Sprache [DR01]:

> „*A conditional is a two-clause structure in which one of the clauses is introduced by* ***if*** *(possibly preceded by* ***only***, ***even*** *or* ***except****) or by a word or phrase that has a meaning similar to* ***if***, ***only if*** *(e.g.* ***provided****) or* ***except if*** *(viz.* ***unless****).*“[35]

[35] Zu deutsch: Ein Bedingung ist eine Struktur aus zwei Gliedsätzen, in welcher einer der Gliedsätze durch das Schlüsselwort *if* (gegebenenfalls gefolgt von *only*, *even* oder *except*) oder durch ein Wort oder eine Phrase mit einer vergleichbaren Bedeutung wie *if* bzw. *only if* (z. B. *provided*) oder *except if* (nämlich *unless*) eingeleitet wird.

Bedingungen bestehen immer aus einem Konditionalsatz, in dem eine Bedingung formuliert wird, und einem Hauptsatz, in dem die von der Bedingung abhängige Aussage formuliert wird[36]. Zusätzlich wird in der Literatur zwischen zwei unterschiedlichen Arten von Bedingungen unterschieden: Ereignis-Bedingungen und voraussetzende Bedingungen [Hae03]. Ereignis-Bedingungen beschreiben Ereignisse oder (eintretende) Zustände in einem Konditionalsatz, die wiederum zu einem Ereignis (oder einer Zustandsänderung) führen, das in der Hauptphrase beschrieben wird. Voraussetzende Bedingungen hingegen gliedern den Diskurs, indem im Konditionalsatz Bezug auf eine vorangegangene Äußerung genommen wird. Für die Erkennung von Bedingungen im programmatischen Sinne sind damit lediglich Ereignis-Bedingungen relevant; dementsprechend fokussieren sich die nachfolgenden Betrachtungen auf diese Art von Bedingungen.

Wie zuvor beschrieben, bestehen Bedingungen in natürlicher Sprache immer aus einem Konditionalsatz, in dem die Bedingung formuliert wird und einem Hauptsatz. Der Hauptsatz kann weiter untergliedert werden, in einen *Dann*-Gliedsatz und einen optionalen *Ansonsten*-Gliedsatz. Im nachfolgenden Beispielsatz sind die unterschiedlichen Gliedsätze markiert:

[*If the kitchen is dirty,*]$_{\text{Wenn}}$ [*clean it*]$_{\text{Dann}}$ [*and otherwise just wait.*]$_{\text{Ansonsten}}$

Auf diese Weise entsteht eine *Wenn-Dann-Ansonsten*-Struktur, die der aus Programmiersprachen bekannten Struktur verzweigter Bedingungen in *if*, *then* und *else* stark ähnelt. Diese Ähnlichkeit besteht nicht zufällig; zwar sind Programmiersprachen formale Sprachen, beim Entwurf wurden allerdings viele Elemente natürlicher Sprachen in abgewandelter Form übernommen. Die Begründung hierfür liegt darin, dass Konstrukte wie Bedingungen in natürlicher Sprache auf bestimmte Denkmuster zurückgeführt werden können (in diesem Fall *Voraussetzungen und Fallunterscheidungen*). Diese Muster werden dann nur in natürlicher Sprache ausgedrückt. Beim Entwurf von Programmiersprachen wurde die Ausdrucksart übernommenen, um die vorhandenen Denkmuster zu unterstützen.

Betrachtet man näher, wie Bedingungen im Englischen formuliert werden, so fällt zunächst auf, dass im Wesentlichen zwei Reihenfolgemuster auftreten:

1. ein Konditionalsatz, gefolgt von einem *Dann*-Gliedsatz und einem optionalen *Andernfalls*-Gliedsatz: *Wenn-Dann-(Ansonsten)*
2. ein *Dann*-Gliedsatz, gefolgt von in Konditionalsatz und einem optionalen *Andernfalls*-Gliedsatz: *Dann-Wenn-(Ansonsten)*

Die erstgenannte Struktur ist jedoch die deutlich verbreiterte. Hinsichtlich der Formulierungen der Konditionalsätze bietet die englische Sprache eine Vielzahl von Variationen. Zwar beschreiben Declerck und Reed wie zuvor diskutiert, dass Konditionalsätze immer von einem Schlüsselwort oder einer Phrase eingeleitet werden. Allerdings gibt es eine Vielzahl von möglichen Synonymen und Umschreibungen. Ein Beispiel für letzteres ist die Äußerung *the longer you rinse it, the*

[36] Dies gilt zumindest für germanische Sprachen.

more it will get clean, die kein Schlüsselwort enthält und die Bedingung nur durch Erfassung der Semantik der gesamten Aussage erkennen lässt. Diese vielfältigen Ausprägungen der Semantik von Bedingungen erschwert die Formalisierung in einem allgemeingültigen Modell. Bedingungen können in natürlichsprachlichen Äußerungen in unterschiedlichen Kontexten verwendet werden und daher beliebige Bedeutungen tragen. Für den Agenten zur Erkennung von verzweigten Bedingungen wird angenommen, dass alle in natürlicher Sprache formulierten Bedingungen in programmatische *if-then-else*-Strukturen überführt werden können. Das bedeutet auch, dass eine bedingte Verzweigung immer mindestens aus einem Konditionalsatz und einem *Dann*-Gliedsatz bestehen muss. Andernfalls ist weder die Semantik der Bedingung seitens der natürlichen Sprache klar, noch lässt sie sich in ihr programmatisches Pendant überführen.

Die Aufgabe des Agenten soll es sein, die Satzglieder in die programmatische Struktur zu überführen. Es wird an dieser Stelle keine Überprüfung der Bedingung oder der Überführbarkeit der *Dann*- und *Andernfalls*-Gliedsätze in Quelltext vorgenommen. Die Herausforderungen bestehen dementsprechend hauptsächlichen darin, zum einen die oben beschriebenen Reihenfolge der Gliedsätze zu bestimmen sowie festzustellen, ob alle nötigen (Konditionalsatz und *Dann*-Gliedsatz) und optionalen Bestandteile (*Andernfalls*-Gliedsatz) vorhanden sind. Zum anderen muss der Umfang der jeweiligen Bestandteile bestimmt werden. In Programmiersprachen sind die Bedingung (*if*) sowie die *then*- und *else*-Blöcke eindeutig anhand der Syntax bestimmbar, beispielsweise durch eine entsprechende Klammerung. In natürlichen Sprachen hingegen ist eine solche Unterstützung durch die Syntax nicht gegeben; es muss stattdessen anhand der Semantik entschieden werden, auf welche nachfolgenden Aussagen sich ein Konditionalsatz bezieht. Im Folgenden wird der Umfang von *Dann*- und *Andernfalls*-Gliedsätze als *Bezugsrahmen* des Konditionalsatzes bezeichnet.

Die Bestimmung des Bezugsrahmens ist selbst für einfache Beispiele nicht eindeutig möglich; beispielsweise ist bei der Äußerung *if I'm still at work do the laundry then clean the kitchen* unklar, ob sich die Bedingung auf die beiden nachfolgenden Äußerungen oder nur auf die erste bezieht. Ist ein *Andernfalls*-Gliedsatz vorhanden, so begrenzt dieser die *Dann*-Gliedsätze (in der *Wenn-Dann-Ansonsten*-Struktur). Ist dieser jedoch nicht vorhanden, findet keine syntaktische Eingrenzung statt. Gleiches gilt für *Andernfalls*-Gliedsätze, die grundsätzlich syntaktisch unbegrenzt sind. Für die *Dann-Wenn-Ansonsten*-Struktur gilt, dass *Dann*-Gliedsätze syntaktisch durch den Konditionalsatz begrenzt sind. Allerdings ist in diesem Fall der Beginn des Bezugsrahmens unklar. Die Aufgabe der Bestimmung des Bezugsrahmens wird durch die Fokussierung von ProNat auf gesprochenen Sprache zusätzlich erschwert. Neben den grundsätzlichen Herausforderungen, wie falsch erkannten Wörtern und Disfluenzen bzw. Reparaturen, ist an dieser Stelle das Fehlen der Interpunktion von besonderer Bedeutung (siehe Abschnitt 2.3.12). Mithilfe der Interpunktion können in geschriebener Sprache viele mehrdeutige Fälle ausgeschlossen werden.

Beispiel:
Interpretierbarkeit von gesprochenen und geschriebenen Äußerungen

In diesem Beispiel soll der Unterschied in der Interpretierbarkeit von gesprochenen und geschriebenen Äußerungen demonstriert werden. Das Beispiel betrachtet ausschließlich die Unterschiede, die aufgrund der fehlenden Interpunktion gesprochener Äußerungen auftreten.

Natürlichsprachliche Sequenzen

Geschrieben:

(1) *Clean the kitchen. If I am still at work do the laundry.*

Gesprochen (transkribiert):

(2) *clean the kitchen if I am still at work do the laundry*

Erläuterung

Im Fall der geschriebenen Äußerung ist die Interpretation eindeutig. Da die erste Aussage in einem eigenen Satz formuliert wurde, bezieht sich der Konditionalsatz auf die nachfolgende Aussage. Im Fall der Transkription ist der Bezug jedoch unklar. Ohne Interpunktion kann die Äußerung entweder so gedeutet werden, dass sich der Konditionalsatz ebenfalls auf die nachfolgende Äußerung bezieht, alternativ ist aber auch ein Bezug auf die vorherige möglich[37].

Das Ziel von ProNat ist die Programmierung mit uneingeschränkter gesprochener Sprache. Für die Erkennung von verzweigten Bedingungen bedeutet das, eine Lösung muss neben den üblichen Herausforderungen natürlicher Sprache auch mit Äußerungen variabler Länge umgehen können, die zudem beliebig formuliert sind. Außerdem kann nicht davon ausgegangen werden, dass jede Aussage genau eine Bedingung und davon abhängige Aktionen enthält (eine Einschränkung, die von verwandten Arbeiten häufig getroffen wird). Vielmehr kann eine Äußerung eine beliebige Anzahl an verzweigten Bedingungen enthalten (oder auch keine) und von jeder Bedingung kann eine beliebige Menge von Aktionen (in den Gliedsätzen) abhängen.

Aufgrund dieser Rahmenbedingungen ist ein Entwurfsziel für den Agenten, möglichst viele richtige, vor allem aber auch nachvollziehbare und vorhersagbare Ergebnisse zu erzeugen. Daher ist der Ansatz regelbasiert; er extrahiert mithilfe von Heuristiken auf Basis der Syntax der Äußerungen und Schlüsselwörtern bedingte Verzweigungsstrukturen. Das grundsätzliche Vorgehen ist dabei konservativ. Nur Phrasen, die sicher einem Bestandteil zugeordnet werden können, werden extrahiert. Dies betrifft insbesondere den zuvor diskutierten Umfang der Bezugsrahmen einer Bedingung. Das bedeutet, der Ansatz legt mehr Wert auf eine hohe Präzision als auf eine hohe Ausbeute.

[37] Eine eindeutige Interpretation der gesprochenen Äußerung ist gegebenenfalls anhand der Intonation möglich. Eine Betrachtung dieser steht ProNat aufgrund der Projekt-Struktur jedoch nicht zur Verfügung. Außerdem gibt es für die Interpretation der Intonation noch keine zuverlässigen Lösungen.

Tabelle 7.16: Schlüsselwörter und -phrasen zur Erkennung der Bestandteile von bedingten Verzweigungen in natürlicher Sprache.

Bestandteil	Schlüsselwörter/-phrasen
Wenn (Konditionalsatz)	if, when, suppose(d) that, supposing that, whenever, in case, in the case that, unless, on condition that, providing that, provide(d) that, as long as, else if
Dann	then, please, if so, you can, you have to, could you, would you
Andernfalls	else, if not, otherwise, elseways, alternatively, instead, either, rather, oppositely

$$
\begin{aligned}
\text{Bedingung} &\rightarrow \text{Konditionalsatz } \textit{Dann}\text{-Gliedsatz} \\
\text{Bedingung} &\rightarrow \text{Konditionalsatz } \textit{Dann}\text{-Gliedsatz } \textit{Andernfalls}\text{-Gliedsatz} \\
\text{Konditionalsatz} &\rightarrow \textit{Wenn-Schlüsselwort NP VP} \\
\textit{Dann}\text{-Gliedsatz} &\rightarrow \textit{Dann-Schlüsselwort VP} \mid \textit{VP} \\
\textit{Dann}\text{-Gliedsatz} &\rightarrow \textit{Dann-Schlüsselwort NP VP} \mid \textit{NP VP} \\
\textit{Andernfalls}\text{-Gliedsatz} &\rightarrow \textit{Andernfalls-Schlüsselwort VP} \\
\textit{Andernfalls}\text{-Gliedsatz} &\rightarrow \textit{Andernfalls-Schlüsselwort NP VP}
\end{aligned}
$$

Abbildung 7.8: Kontextfreie Grammatik zur Erzeugung der *Minimalkonstrukte* für die Erkennung von verzweigten Bedingungen.

Implementierung des Agenten zur Erkennung von bedingten Verzweigungen

Als Grundlage für die Erkennung von verzweigten Bedingungen verwendet der Agent nur die Ergebnisse der Vorverarbeitungsstufe zur seichten Sprachverarbeitung, genauer gesagt die Wortart- und Chunk-Etiketten (siehe Abschnitt 6.2). Hierzu werden aus dem $\texttt{Pro}\mathcal{Nat}$-Graph alle Knoten vom Typ `Token` extrahiert, welche die ursprüngliche Äußerung, angereichert um die in der Vorverarbeitung erzeugten Informationen, darstellen. Optional können die Ergebnisse des Agenten zur Korreferenzanalyse von Nominalphrasen genutzt werden, um die Analyse der Bezugsrahmen zu verbessern (siehe Abschnitt 7.5). Der Ansatz verwendet anschließend Schlüsselwörter und spezialisierte Grammatiken zur Extraktion der verzweigten Bedingungen. Die Produktionsregeln der Grammatiken sind von den zuvor diskutierten linguistischen Charakteristiken von Bedingungen in natürlichsprachlichen Äußerungen abgeleitet. Der Agent verarbeitet die Äußerungen in zwei Phasen. Zunächst werden sogenannte *Minimalstrukturen* mithilfe von Schlüsselwörtern und einfachen, robusten Grammatiken extrahiert (siehe Tabelle 7.16 und Abbildung 7.8).

Falls mithilfe der Grammatiken keine *Minimalstrukturen* um das Schlüsselwort gebildet werden können, wird das Schlüsselwort verworfen. Das Ergebnis dieses ersten Schrittes sind Kandidatenmengen für Konditionalsätze sowie *Dann*- und *Andernfalls*-Gliedsätze. Die einzelnen Strukturen sind jedoch bisher nicht miteinander verbunden. Außerdem können mit diesem Vorgehen die meisten *Dann*-Gliedsätze nicht ermittelt werden, da diese häufig *nicht* mit einem Schlüsselwort eingeleitet

$$
\begin{aligned}
\text{Konditionalsatz} &\rightarrow \textit{Wenn-Schlüsselwort}\ \text{NBS} \\
\text{NBS} &\rightarrow \text{NPB VPB} \mid \text{NPB VPB}\ \textit{Conj}\ \text{NPB VPB} \\
\text{NPB} &\rightarrow \textit{NP}\ \text{CC NPB} \mid \textit{NP PP}\ \text{NPB} \mid \textit{NP} \\
\text{VPB} &\rightarrow \textit{VP}\ \text{CC VPB} \mid \textit{VP PP}\ \text{NPB} \mid \textit{VP PP}\ \text{VPB} \mid \textit{ADVP}\ \text{VPB} \\
&\quad \mid \textit{VP}\ \text{VMD} \mid \textit{VP} \\
\text{CC} &\rightarrow \textit{Conj} \mid \textit{Neg} \\
\text{VMD} &\rightarrow \textit{ADJP} \mid \textit{ADVP} \mid \textit{PRT}
\end{aligned}
$$

Abbildung 7.9: Ausschnitt der kontextfreien Grammatik zur Erzeugung von Konditionalsätzen.

werden. Aus diesem Grund werden *Dann*-Gliedsätze gesondert behandelt. Um fehlende *Dann*-Gliedsätze zu erkennen, werden alle Phrasen zwischen aufeinanderfolgenden Konditionalsätzen und *Andernfalls*-Gliedsätzen betrachtet. Auf diese Phrasen werden die in Abbildung 7.8 dargestellten Produktionsregeln für *Dann*-Gliedsätze *ohne* Verwendung von Schlüsselwörtern angewandt. Die so erkannten *Minimalstrukturen* werden zur Kandidatenmenge hinzugefügt. Zuletzt werden die Kandidaten mithilfe der ersten beiden Produktionsregeln zusammengesetzt und es entstehen elementare *Wenn-Dann(-Ansonsten)*-Strukturen. Anschließend wird versucht, den Umfang der Bezugsrahmen auszudehnen. Hierzu werden Minimalstrukturen inkrementell mithilfe von komplexeren Grammatiken (syntaktische Erweiterung) und Korreferenzinformationen (Referenz-basierte Erweiterung) erweitert. Für die syntaktische Erweiterung der *Minimalstrukturen* werden Grammatiken verwendeten, die eine teilweise Zerteilung (engl. *partial parse*) für die Äußerung erzeugen. Abbildung 7.9 zeigt einen Auszug der Grammatik, die verwendet wird, um Konditionalsätze zu extrahieren.

Die Produktionsregeln der Grammatiken sind von den zuvor diskutierten grammatikalischen Charakteristiken von Bedingungen in natürlicher Sprache abgeleitet. Im Fall der Grammatik für Konditionalsätze sind die Produktionsregeln anhand folgender Überlegungen entstanden. Zunächst gibt es syntaktische Strukturen, die Phrasen miteinander verbinden: Konjunktionen (*Conj*) und Negationen (*Neg*) bilden homogene Nominalphrasen- und Verbalphrasen-Blöcke (NPB und VPB). Verbalphrasen können zudem über Adverbialphrasen (*ADVP*) und Präpositionalphrasen (*PP*) verbunden werden. Letztere verbinden außerdem Verbal- und Nominalphrasen und erzeugen so inhomogene Verbalphrasen-Blöcke. Die Blöcke können zu sogenannten Nominal-Block- und Verbal-Block-Strukturen (NBS und VBS) verbunden werden (je nach Reihenfolge der Blöcke). Ein Konditionalsatz kann nur mithilfe der ersten Produktionsregel erzeugt werden, die besagt, dass ein Konditionalsatz *immer* aus einem entsprechenden Schlüsselwort (oder -phrase), gefolgt von einer Nominal-Block-Struktur, bestehen muss, da ein Konditionalsatz nie mit einer Verbalphrase beginnen kann. Die Grammatiken zur Erzeugung von *Dann*- und *Andernfalls*-Gliedsätzen bestehen aus einer größeren Menge an Produktionsregeln; die Definitionen der Blöcke unterscheidet sich zudem geringfügig. Falls ein Kandidat für einen Konditionalsatz bzw. *Dann*- oder *Andernfalls*-Gliedsatz nicht mithilfe der jeweiligen Grammatik erweitert werden kann, wird auf die entsprechende *Minimalstruktur* zurückgegriffen.

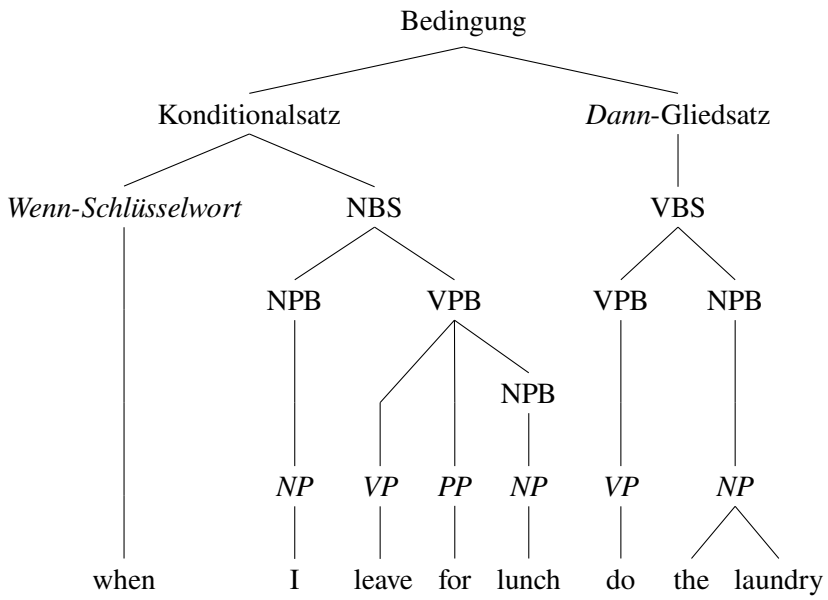
Beispiel:
Syntaktische Erweiterung des Bezugsrahmens

In diesem Beispiel soll die Zerteilung einer natürlichsprachlichen Äußerung mit den erweiterten Grammatiken zur Erkennung von Bedingungen in natürlicher Sprache demonstriert werden. Für die Zerteilung des Konditionalsatzes wurde die in Abbildung 7.9 dargestellte Grammatik und für den *Dann*-Gliedsatz eine entsprechende (hier nicht gezeigte) Grammatik verwendet.

Natürlichsprachliche Sequenz

when I leave for lunch do the laundry

Erzeugte Zerteilung



Die syntaktische Erweiterung der Bezugsrahmen ist konservativ entworfen. Das bedeutet, die erwartete Präzision ist hoch, die Ausbeute hingegen niedrig. Der Hauptgrund hierfür ist, dass mithilfe der Grammatiken zwar verkette Gliedsätze erkannt werden können, erstreckt sich eine vom Konditionalsatz abhängige Struktur (*Dann* oder *Andernfalls*) hingegen über mehrere Sätze (bzw. Instruktionen), genügen die Grammatiken nicht mehr. Aus diesem Grund werden die Bezugsrahmen zusätzlich mithilfe von Korreferenzinformationen ausgedehnt. Die grundlegende Idee ist dabei folgende: Wird eine Entität mehrfach in aufeinanderfolgenden Phrasen erwähnt, kann man davon ausgehen, dass diese Phrasen zusammengehören; das bedeutet, sie gehören zur gleichen Struktur. Um derartige Fälle zu erkennen, werden die Korreferenz- und Identitätsinformationen des Agenten zur Korreferenzanalyse von Nominalphrasen verwendet (siehe Abschnitt 7.5). Aus den einzelnen Referenzen werden sogenannten Korreferenzketten gebildet (siehe Abschnitt 2.3.4.7). Das bedeutet, für jede Entität wird eine (zusammenhängende) Reihe von Nennungen gebildet. Normalerweise unterscheidet man bei der Bildung von Korreferenzketten zwischen Inter- und Intra-Satz-Korreferenzen. Da bei gesprochenen Äußerungen jedoch keine Interpunktion vorhanden

ist, wird hier auf diese Unterscheidung verzichtet und alle Referenzen werden gleich behandelt. Der Agent fügt alle Phrasen, die Teil einer Korreferenzkette sind, zu derjenigen Struktur (*Dann* oder *Andernfalls*) hinzu, in der die Entität das erste Mal genannt wird. Die Korreferenzketten werden aufgeteilt, sobald sie unterschiedliche Arten von *Minimalstrukturen* überspannen. So kann gewährleistet werden, dass sich unterschiedliche Strukturarten nicht überlagern.

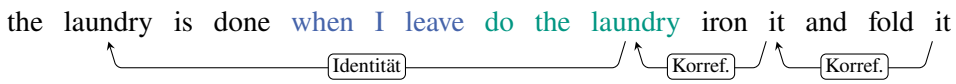
Beispiel:
Referenz-basierte Erweiterung des Bezugsrahmens

In diesem Beispiel soll die Referenz-basierte Erweiterung des Bezugsrahmens demonstriert werden.

Natürlichsprachliche Sequenz

the laundry is done when I leave do the laundry iron it and fold it

Erkannte Korreferenzkette



Erläuterung

Für die natürlichsprachliche Äußerung wurden mithilfe der Grammatiken *when I leave* als Konditionalsatz und *do the laundry* als *Dann*-Gliedsatz erkannt. Höchstwahrscheinlich ist die Äußerung aber so zu verstehen, dass auch *iron it* und *fold it* vom Konditionalsatz abhängen. Mithilfe der Korreferenzinformation wird für die Entität *the laundry* folgende Korrefenzkette gebildet:

the laundry → the laundry → it → it.

Aus dieser Kette kann abgeleitet werden, dass *iron it* und *fold it* ebenfalls zur *Dann*-Struktur gehören; der Bezugsrahmen des Konditionalsatzes wird entsprechend erweitert. Die Phrase *the laundry is done* wird nicht hinzugefügt, da sich die entsprechende Referenz (*Identität*) über eine Struktur anderen Typs (Konditionalsatz) erstreckt.

7.6.2 Nebenläufigkeit und Schleifen in natürlicher Sprache

Das Vorgehen zur Erkennung von Nebenläufigkeit und Schleifen in natürlicher Sprache kann nicht wie die Bedingungserkennung auf linguistische Theorien zurückgeführt werden, da in der Linguistik für diese Phänomene keine einheitliche Betrachtungsweise existiert. Dies liegt unter anderem daran, dass sowohl Wiederholungen als auch Nebenläufigkeit (von Ereignissen) in natürlicher Sprache sehr unterschiedlich ausgedrückt werden können. Insbesondere gibt es (im Allgemeinen) keine typischen syntaktischen Strukturen, wie die von Declerck und Reed für Bedingungen beschriebene [DR01]. Da im Kontext von ProNat das Problem der Erkennung von Nebenläufigkeit und Schleifen in

natürlicher Sprache jedoch nicht allgemein gelöst, sondern ausschließlich für die Programmierung mit natürlicher Sprache umgesetzt werden soll, kann pragmatisch vorgegangen werden.

Auch verwandte Arbeiten setzen auf zielsetzungsorientierte Lösungen. Landhäußer und Hug verwenden beispielsweise Schlüsselwörter und -phrasen [LH15]. Sie beschreiben weiterhin, dass diese gegebenenfalls Aufschluss über die Art der Kontrollstruktur geben. Ist dies nicht der Fall, so muss der Kontext untersucht werden. Die Autoren bestimmen anhand der Anzahl, der Reihenfolge und der Ordnung der abhängigen Gliedsätze die Art der Kontrollstruktur. Zusätzlich kann auch das Schlüsselwort an sich weitere Information tragen. Beispielsweise impliziert die Aussage *do A twice*, dass *A* zweimal durchgeführt werden soll. Dementsprechend weist das Schlüsselwort *twice* nicht nur auf die Verwendung einer Zählschleife hin, sondern auch auf die Anzahl der Iterationen (hier zwei).

Auch wenn die Arbeit von Landhäußer und Hug eine gute Orientierung liefert, kann der Ansatz nicht übernommen werden. Die erforderlichen Analysen verwenden Abhängigkeitsgraphen, die nur für geschriebene Sprache erzeugt werden können (siehe Abschnitt 2.3.4.6). Darüber hinaus scheint der Ansatz einige Schlüsselwörter und -phrasen nicht geeignet zu verwenden; außerdem erscheinen einige Annahmen hinsichtlich der Satzstruktur von Schleifen- und Nebenläufigkeitsbeschreibungen nicht allgemeingültig.

Die im Folgenden vorgestellten Implementierungen zur Erkennung von Nebenläufigkeit und Schleifen in natürlicher Sprache nutzen ebenfalls Schlüsselwörter bzw. -phrasen. Die Schlüsselwörter sind jedoch typisiert; die abhängigen Gliedsätze werden je nach Typ anders bestimmt. Um die Struktur der verbalisierten Kontrollstrukturen zu analysieren, werden semantische Rollen verwendet (siehe Abschnitt 6.4). Auch für Nebenläufigkeit und Schleifen werden zunächst *Minimalstrukten* erzeugt, die anschließend heuristisch erweitert werden.

7.6.3 Implementierung des Agenten zur Erkennung von Nebenläufigkeit

Der Agent zur Erkennung von Nebenläufigkeit verwendet für die Analyse lediglich die Wörter der natürlichsprachlichen Äußerungen und deren semantische Rollen. Damit besteht eine Abhängigkeit von den Ergebnissen der Vorverarbeitungsstufe von `Pro`$\mathcal{N}at$ zur Erzeugung semantischer Rollen (siehe Abschnitt 6.4) Zur Ausdehnung des Bezugsrahmens einer erkannten Nebenläufigkeit werden zusätzlich Korreferenzinformationen verwendet, sofern der entsprechende Agent diese generiert hat (siehe Abschnitt 7.5).

Da wie zuvor erläutert in der Linguistik keine Definition für die Struktur von Gliedsätzen, die nebenläufige Ereignisse beschreiben, existiert, wird (verbalisierte) Nebenläufigkeit im Kontext von `Pro`$\mathcal{N}at$ folgendermaßen definiert: Eine in natürlicher Sprache verbalisierte Nebenläufigkeit ist eine Struktur, die aus zwei oder mehr Gliedsätzen besteht, welche nebenläufige Ereignisse beschreiben (im Folgenden als *nebenläufige Gliedsätze* bezeichnet). Das programmatische Pendant sind Quelltextabschnitte, die parallel von mehreren Fäden (engl. *threads*) ausgeführt werden können, wie beispielsweise das *OpenMP-Pragma* `omp parallel for`. Die nebenläufigen Gliedsätze können in natürlichsprachlichen Äußerungen anhand bestimmter Schlüsselwörter bzw. -phrasen erkannt

Tabelle 7.17: Schlüsselwörter und -phrasen zur Erkennung von Nebenläufigkeit in natürlicher Sprache: Die Schlüsselwörter und -phrasen sind nach Typen unterteilt.

Typ	Schlüsselwörter/-phrasen
Umhüllend	at once, simultaneously, coevally, concurrently, synchronistically, during
Teilend	at the same time, while, whilst, meanwhile, in the meantime, while doing so
Öffnend	and while, while, and whilst, whilst
Schließend	at the same time, in the meantime, while doing so

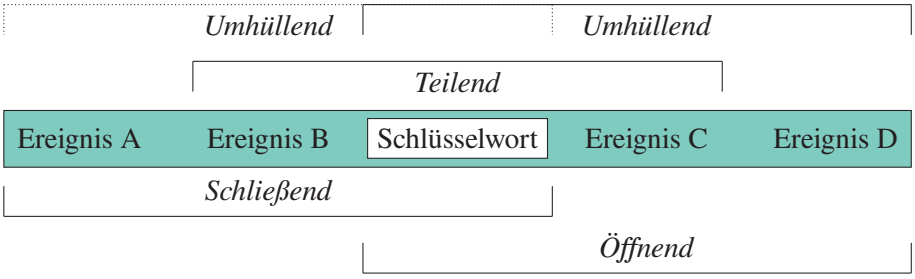


Abbildung 7.10: Spannen der Schlüsselwort-Typen zur Erkennung von Nebenläufigkeit.

werden. Die Struktur der nebenläufigen Gliedsätze hängt dabei vom jeweiligen Schlüsselwort ab; folgende Typen werden unterschieden:

- *Umhüllend*: Die nebenläufigen Gliedsätze treten in der Äußerung entweder vor oder nach dem Schlüsselwort auf, wobei die zweite Variante die üblichere ist.
- *Teilend*: Das Schlüsselwort tritt in der natürlichsprachlichen Äußerung zwischen den nebenläufigen Gliedsätzen auf.
- *Öffnend*: Das Schlüsselwort ist den nebenläufigen Gliedsätzen vorangestellt.
- *Schließend*: Das Schlüsselwort ist den nebenläufigen Gliedsätzen nachgestellt.

Die zugehörigen Schlüsselwörter bzw. -phrasen können Tabelle 7.17 entnommen werden. Zu beachten ist, dass einige davon primäre und sekundäre Typen haben. Die Ordnung der Schlüsselwort-Typen entspricht der Reihenfolge der Zeilen der Tabelle; beispielsweise hat *while* den primären Typ *Teilend* (zweite Zeile) und den sekundären *Öffnend* (dritte Zeile).

Der Agent detektiert zunächst die Schlüsselwörter in der natürlichsprachlichen Äußerung. Anschließend werden je nach Schlüsselwort-Typ die nebenläufigen Gliedsätze extrahiert. Abbildung 7.10 zeigt den Zusammenhang zwischen Schlüsselwörtern, Schlüsselwort-Typen und zugehörigen Gliedsätzen. Beispielsweise werden bei einem Schlüsselwort vom Typ *Öffnend* die nächsten zwei Gliedsätze extrahiert. Die Gliedsätze wiederum sind in diesem Kontext genau ein Ereignis, wobei ein Ereignis einer (vom Erkenner von semantischen Rollen) Verb-Struktur entspricht (das heißt, das erkannte Verb und die davon abhängigen Rollen, siehe Abschnitt 6.4). Wie schon bei der Erkennung von Bedingungen ist auch hier das Vorgehen konservativ. Daher wird pro Gliedsatz auch nur genau ein Ereignis extrahiert. Ohne Interpunktion ist es unmöglich, den Bezugsrahmen von Nebenläufigkeit genauer zu bestimmen. Beispielsweise kann bei Äußerungen wie „do A do B *while* you do C and D“ nicht entschieden werden, ob nur die Ereignisse *B* und *C* gleichzeitig oder *A* und *B* gleichzeitig

mit *C* und *D* stattfinden sollen. Andererseits ist es kein Problem, verschachtelte Nebenläufigkeit zu erkennen, solange jedes Paar von nebenläufigen Gliedsätzen über ein Schlüsselwort identifiziert werden kann, wie in „do A *while* you do B *in the meantime* do C“. Falls mithilfe des primären Typs des Schlüsselwortes keine nebenläufigen Gliedsätze ermittelt werden können, wird das Verfahren mit dem sekundären Typen wiederholt. Falls auch dann kein entsprechendes Ergebnis erzielt werden kann, wird das Schlüsselwort verworfen. Zuletzt wird versucht, den Umfang des Bezugsrahmens der ermittelten Nebenläufigkeit zu erweitern. Hierzu wird der Ansatz, der zum selben Zweck bei der Erkennung verzweigter Bedingungen eingesetzt wurde, adaptiert. Auch hier werden die Korreferenzinformationen des Agenten zur Korreferenzanalyse von Nominalphrasen genutzt, um Korreferenzketten zu bilden. Allerdings können nun (anders als bei der Analyse von verbalisierten Bedingungen) die Ereignisse (wie zuvor definiert) betrachtet werden; das bedeutet, die Kettenglieder werden von Ereignissen gebildet. Alle Ereignisse einer Kette werden dem nebenläufigen Gliedsatz zugeordnet in dem das erste Glied der Kette auftritt. Die Intuition hinter diesem Vorgehen ist, dass Ereignisse, welche dieselben Entitäten betreffen, stärker in Verbindung stehen als andere. Wie auch bei verzweigten Bedingungen gilt: überspannt die Korreferenzkette das Schlüsselwort oder einen anderen Gliedsatz, so wird die Kette aufgebrochen.

Der hier vorgestellte Agent erkennt selbst keine bedingte Nebenläufigkeit, da sich diese genauso wie eine verschachtelte Variante verhält: eine *Wenn-Dann-Andernfalls*-Struktur mit Nebenläufigkeit innerhalb der *Dann*- oder *Andernfalls*-Struktur. Dementsprechend kann bedingte Nebenläufigkeit erkannt werden, indem der hier vorgestellte Agent und der Agent zur Erkennung von verzweigten Bedingungen unabhängig voneinander die natürlichsprachliche Äußerung analysieren.

7.6.4 Implementierung des Agenten zur Erkennung von Schleifen

Der Agent zur Erkennung von Schleifen verwendet wie der Agent zur Erkennung von Nebenläufigkeit semantische Rollen zur Interpretation von Ereignissen, die in der natürlichsprachlichen Äußerung beschrieben werden. Zusätzlich werden Chunk-Etiketten verwendet. Dementsprechend sind die erzielten Ergebnisse abhängig von den Vorverarbeitungsstufen zur seichten Sprachverarbeitung und zur Erzeugung semantischer Rollen (siehe Abschnitt 6.2 und Abschnitt 6.4). Sofern verfügbar, verwendet auch dieser Agent zur Ausdehnung der Bezugsrahmen Korreferenzinformationen, die vom entsprechenden Agenten bereitgestellt werden (siehe Abschnitt 7.5).

In Programmiersprachen finden verschiedene Arten von Schleifen Verwendung. Im Kontext von ProNat werden nur solche linguistische Strukturen betrachtet, die auf *While*-, *Do-While*- und Zählschleifen (*For*-Schleifen) abgebildet werden können. Diese Schleifentypen werden typischerweise in gängigen Programmiersprachen verwendet; gleichzeitig existieren für diese natürlichsprachliche Pendants. Die Erkennung und Überführung von verbalisierten Schleifenkonstrukten ist nichtsdestotrotz eine Herausforderung, da Ereignisse, die wiederholt stattfinden sollen, in mannigfaltiger Weise formuliert werden können.

Beispiel:
Herausforderungen bei der Erkennung von verbalisierten Schleifen

In diesem Beispiel sollen unterschiedliche Formulierungen von verbalisierten Schleifen demonstriert werden. Das Beispiel zeigt zudem, wie relevante Informationen aus den Äußerungen extrahiert werden können.

Natürlichsprachliche Sequenzen

(1) *turn* ***twice***

(2) ***while the fridge is open*** *take out beverages*

Erläuterung

Das erste Beispiel kann als Zählschleife interpretiert werden; das Schlüsselwort *twice* weist darauf hin. Zusätzlich kann anhand des Schlüsselwortes erkannt werden, dass die Iterationszahl der Schleife zwei beträgt. Dasselbe gilt für Schlüsselphrase wie *four times*. Das abhängige Ereignis im ersten Beispiel ist *turn*. Das zweite Beispiel beinhaltet neben dem Schlüsselwort *while* die Bedingung *the fridge is open*. Dieses Konstrukt entspricht in seinem Aufbau einer Konditionalphrase und kann auch in Verbindung mit anderen Schlüsselwörtern bzw. -phrasen auftreten, beispielsweise *until* und *as long as*. Das Schlüsselwort *while* ist zudem mehrdeutig, da es auch bei der Beschreibung verbalisierter Nebenläufigkeit Verwendung findet (siehe Tabelle 7.17). Zur Auflösung der Mehrdeutigkeit kann der Kontext des Schlüsselwortes betrachtet werden. Folgt dem Schlüsselwort eine verbalisierte Bedingung, handelt sich höchstwahrscheinlich um eine Schleife, andernfalls um Nebenläufigkeit.

Da wie auch im Falle von Nebenläufigkeit in der Linguistik keine Definition hinsichtlich der Struktur von (verbalisierten) Schleifen existiert, werden diese im Kontext von ProNat wie folgt definiert: Eine in natürlicher Sprache verbalisierte Schleife ist eine Struktur, die aus einem Gliedsatz, der eine Bedingung enthält, und mindestens einem weiteren Gliedsatz, der ein oder mehrere Ereignisse enthält, besteht. Der Gliedsatz, der die Bedingung enthält (in der Folge als *Bedingungs*-Gliedsatz bezeichnet), kann wie ein Konditionalsatz aufgebaut sein; er kann aber auch nur aus einem Schlüsselwort bzw. einer Schlüsselphrase bestehen. *Bedingungs*-Gliedsätze enthalten die Abbruchbedingung im programmatischen Sinn. Die Gliedsätze, welche die abhängigen Ereignisse enthalten (im Folgenden als *Schleifenkörper*-Gliedsätze bezeichnet), entsprechen den Anweisungen innerhalb des Schleifenkörpers im programmatischen Sinn. Wie auch Nebenläufigkeit, können verbalisierte Schleifen in natürlicher Sprache anhand von Schlüsselwörtern bzw. -phrasen erkannt werden. Die Schlüsselwörter sind dabei Teil des *Bedingungs*-Gliedsatzes.

Um verbalisierte Schleifen aus natürlichsprachlichen Äußerungen zu extrahieren, sucht der Agent zunächst nach Schlüsselwörtern bzw. -phrasen. Die verwendeten Schlüsselwörter und -phrasen sind in Tabelle 7.18 aufgeführt. Falls das Schlüsselwort nicht implizit die zugehörige Abbruchbedingung enthält, wie bei *twice* oder *four times*, wird eine partielle Zerteilung der Äußerung, rund um das

Tabelle 7.18: Schlüsselwörter und -phrasen zur Erkennung von Schleifen in natürlicher Sprache: Die Schlüsselwörter und -phrasen sind nach Typen unterteilt. [CD] ist ein Stellvertretersymbol für ein beliebiges Wort mit der Wortartetikette *CD*.

Typ	Schlüsselwörter/-phrasen
Zählschleife	twice, thrice, [CD] times
Umhüllend	while, as long as, until, till
Öffnend	and while, and as long as
Schließend	repeat this until, repeat it until, do it until, do this until

$$
\begin{aligned}
\text{Bedingungs-Gliedsatz} &\rightarrow \textit{Schleifen-Schlüsselwort}\ \text{NBS} \\
\text{NBS} &\rightarrow \text{NPB VPB} \mid \text{NPB VPB}\ \textit{Conj}\ \text{NPB VPB} \\
\text{NPB} &\rightarrow \textit{NP}\ \text{CC NPB} \mid \textit{NP PP}\ \text{NPB} \mid \textit{NP} \\
\text{VPB} &\rightarrow \textit{VP}\ \text{VMD} \mid \textit{VP}\ \text{VMD CC VPB} \\
\text{CC} &\rightarrow \textit{Conj} \mid \textit{Neg} \\
\text{VMD} &\rightarrow \textit{ADJP} \mid \textit{ADJP}\ \text{CC VMD} \mid \textit{PRT} \mid \textit{PRT}\ \text{CC VMD}
\end{aligned}
$$

Abbildung 7.11: Kontextfreie Grammatik zur Erzeugung von *Bedingungs*-Gliedsätzen verbalisierter Schleifen.

Schlüsselwort, vorgenommen. Diese partielle Zerteilung ist ähnlich zu der für verbalisierte Bedingungen durchgeführten, jedoch enthält die hier verwendete Grammatik (siehe Abbildung 7.11) nur eine Teilmenge der Produktionsregeln. Für Konditionalsätze sind Aussagen und Ereignisbeschreibungen als Bedingungen gültig, für *Bedingungs*-Gliedsätze von verbalisierten Schleifen nur Aussagen. Der Grund hierfür sind die zuvor beschriebenen Mehrdeutigkeiten von Schlüsselwörtern.

Nach der Extraktion der Schlüsselwörter und verbalisierten Abbruchbedingungen werden die Strukturen typisiert. Die Typen sind dieselben wie bei der Erkennung der Nebenläufigkeit mit Ausnahme des Typs *Teilend*. Stattdessen wird zusätzlich der Typ *Zählschleife* eingeführt. Schlüsselwörter und -phasen dieses Typs weisen auf eine Zählschleife hin, wobei die Iterationszahl üblicherweise Teil des Schlüsselwortes (oder der Schlüsselphrase) ist. Die Typen werden wie zuvor anhand der Schlüsselwörter bestimmt; die Zuordnung kann Tabelle 7.18 entnommen werden. Im Gegensatz zur Erkennung von nebenläufigen Ereignissen haben alle Schlüsselwörter bzw. -phrasen zur Erkennung verbalisierter Schleifen einen eindeutigen Typen. Das Vorgehen zur Extraktion der abhängigen Ereignisse in den *Schleifenkörper*-Gliedsätzen gleicht dem Vorgehen bei der Nebenläufigkeit. Auch hier hängt die Auswahl, welche Phrasen (bzw. Ereignisse) betrachtet werden, vom Typ der Schleife ab. Abbildung 7.12 veranschaulicht diesen Zusammenhang. Ohne Interpunktionen kann wieder mit Sicherheit nur ein abhängiges Ereignis je Schleife bestimmt werden. Aus diesem Grund wird wie bei der Erkennung verzweigter Bedingungen und Nebenläufigkeit versucht den Umfang der Bezugsrahmen anhand von Korreferenzketten zu erweitern (sofern Korreferenzinformationen vorliegen).

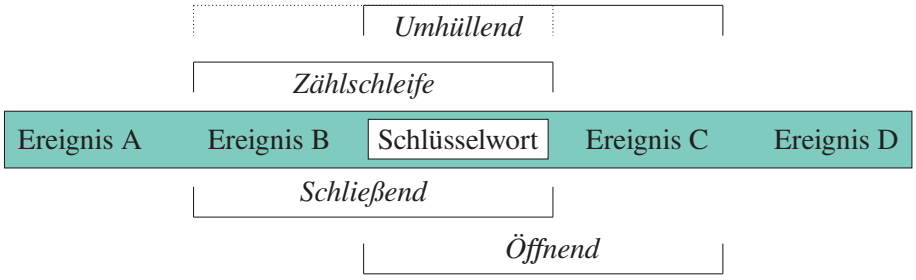


Abbildung 7.12: Spannen der Schlüsselwort-Typen zur Erkennung von Schleifen.

Beispiel:
Synthese von Kontrollstrukturen

In diesem Beispiel soll gezeigt werden, wie unter Verwendung aller drei Agenten zur Synthese von Kontrollstrukturen eine natürlichsprachliche Aussage in Quelltext überführt werden kann. Das Beispiel zeigt aus den Analysen der Agenten abgeleiteten Pseudo-Quelltext zur Steuerung eines fiktiven Haushaltsroboters.

Natürlichsprachliche Sequenz

after dinner go to the laundry room and start the washer wait until the laundry is done while you take the laundry from the washer check its condition if the laundry is dry iron the shirts and fold them twice otherwise put the laundry into the dryer

Erzeugtes Skript

```
robot.go(where: laundryRoom);
robot.start(what: washer);
repeat
    robot.wait();
until laundry.isDone() = true
do together
    robot.empty(what: washer);
    robot.checkState(what: laundry);
end do together
if laundry.isDry() = true then
    robot.iron(shirts);
    for int i = 0; i < 2; i++ do
        robot.fold(shirts);
    end for else
    robot.put(what: laundry, where: dryer);
end if
```

Tabelle 7.19: Datensatz für die Evaluation des Agenten zur Synthese von verzweigten Bedingungen.

	Szenario 4	Szenario 5	Gesamt
Transkriptionen	19	17	36
Transkriptionen ohne bedingte Verzweigungen	4	2	6
Wörter	556	538	1094
Bedingte Verzweigungen	28	19	47
Bed. Verzweigungen ohne *Andernfalls*-Gliedsatz	17	9	26

Tabelle 7.20: Datensatz für die Evaluation der Agenten zur Synthese von Nebenläufigkeit und Schleifen: Die Menge der Instruktionen umfasst alle *Bedingungs*-Gliedsätze und Ereignisse.

	Szenario 9	Szenario 10	Gesamt
Transkriptionen	10	10	20
Transkriptionen ohne Kontrollstrukturen	0	1	1
Wörter	282	287	569
Kontrollstrukturen	10	9	19
Instruktionen	20	17	37

7.6.5 Evaluation der Agenten zur Erkennung von Kontrollstrukturen

Zur Evaluation der Agenten zur Synthese von Kontrollstrukturen aus natürlichsprachlichen Äußerungen werden die Transkriptionen zu den Szenarien vier, fünf, neun und zehn (der stationären Datensammlung) des ProNat-Korpus verwendet (siehe Abschnitt 5.5.2). Die Szenarien vier und fünf enthalten Äußerungen, in denen viele bedingte Verzweigungen formuliert wurden und werden entsprechend für die Evaluation des Agenten zur Erkennung von bedingten Verzweigungen verwendet. Die Transkriptionen zu Szenario neun enthalten hingegen Beschreibungen von nebenläufigen Ereignissen und die zu Szenario zehn Beschreibungen zu sich wiederholenden Ereignissen. Für alle Transkriptionen wurden händisch Musterlösungen erstellt; das bedeutet, je Äußerung wurden die Konditional- bzw. Gliedsätze annotiert. Der Bezugsrahmen der jeweiligen Struktur wurde dabei bestmöglich anhand der Semantik der Äußerung und der zugehörigen Szenariobeschreibung bestimmt. Tabelle 7.19 gibt eine Übersicht über den verwendeten Datensatz zur Evaluation des Agenten zur Erkennung von bedingten Verzweigungen. Von den 36 enthaltenen Äußerungen enthalten vier keine bedingte Verzweigung, die restlichen enthalten in Summe 47 bedingte Verzweigungen. Das bedeutet, im Mittel wurden 1,31 bedingte Verzweigungen pro Äußerung beschrieben. Viele der bedingten Verzweigungen besitzen zwar einen *Dann*- aber keinen *Andernfalls*-Gliedsatz (26 in Summe).

Tabelle 7.20 fasst den Datensatz zusammen, der für die Evaluation der Agenten zur Erkennung von Nebenläufigkeit und Schleifen verwendet wurde. Beide Datensätze enthalten je zehn Transkriptionen, wobei eine der Äußerungen aus Szenario zehn keine verbalisierte Schleife enthält[38]. Die

[38] Der Proband beschreibt den Ablauf der Ereignisse sequenziell, ohne dass etwas wiederholt ausgeführt wird. Stattdessen werden Ereignisse mehrfach beschrieben, das heißt, die Schleife wurde in der Beschreibung sozusagen *ausgerollt*.

Tabelle 7.21: Ergebnisse der Evalution des Agenten zur Synthese von verzweigten Bedingungen: Werte ohne Klammern entsprechen den Ergebnissen *ohne* Verwendung, Werte in Klammern *mit* Verwendung des Agenten zur Korreferenzanalyse von Nominalphrasen.

Gliedsatz-Art	Präzision	Ausbeute	F_1
Konditionalsatz	0,960 (0,960)	0,932 (0,932)	0,946 (0,946)
Dann-Gliedsatz	0,900 (0,911)	0,795 (0,908)	0,844 (0,910)
Andernfalls-Gliedsatz	0,944 (0,951)	0,405 (0,464)	0,570 (0,624)
Gesamt	0,930 (0,934)	0,803 (0,864)	0,862 (0,898)

restlichen Transkriptionen enthalten je eine Kontrollstruktur. Damit die Agenten die Kontrollstrukturen erzeugen können, wurde aus dem ProNat-Vorverarbeitungsfließband der Agent für seichte Sprachverarbeitung ausgeführt. Zur Evaluation der Agenten zur Erkennung von Nebenläufigkeit und Schleifen wurde zusätzlich die Vorverarbeitungsstufe zur Erkennung semantischer Rollen ausgeführt. Um zusätzlich den Einfluss von Korreferenzinformationen bemessen zu können, wurde derselbe Evaluationsaufbau wie für den entsprechenden Agenten hergestellt. Das bedeutet, es wurde der *Babelfy*-Agent zur Disambiguierung von Wortbedeutungen, anschließend mehrfach der Kontext-Agent (bis keine Änderung des Kontext-Modells mehr festgestellt wird) und schließlich der Agent zur Korreferenzanalyse von Nominalphrasen ausgeführt. Zuletzt wurde der jeweils zu evaluierende Agent ausgeführt und die Ergebnisse mit den Musterlösungen verglichen. Um eine möglichst genaue Bemessung der Korrektheit der Lösungen zu ermöglichen und auch teilweise richtige Extraktionen zu erfassen, findet der Vergleich von Musterlösung und Ausgabe des Agenten auf Wortebene statt. Das bedeutet, wurden beispielsweise von einem Gliedsatz nur drei der vier erwarteten Wörter richtig zugeordnet, ist die Extraktion trotzdem zu drei Vierteln korrekt; bei einem blockweisem Vergleich (z. B. je Gliedsatz) würde ein solches Ergebnis als gänzlich falsch gewertet werden. Zur Bemessung der Ergebnisse werden die Metriken Präzision, Ausbeute und F_1-Maß verwendet (siehe Abschnitt 2.4).

7.6.5.1 Evaluation des Agenten zur Erkennung von bedingte Verzweigungen

Die Ergebnisse der Evaluation des Agenten zur Erkennung von verzweigten Bedingungen sind in Tabelle 7.21 abgebildet. Die *nicht* geklammerten Werte wurden *ohne* Verwendung von Korreferenzinformationen erzielt, die Werte in Klammern entsprechend *mit* Korreferenzinformationen. Ohne Korreferenzinformationen erreicht das Verfahren einen sehr guten Wert für das F_1-Maß von 0,862. Die Ergebnisse zeigen zudem, dass der Ansatz Präzision (0,930) gegenüber Ausbeute bevorzugt (0,803). Bei einer eingehenden Analyse der nicht korrekt erkannten verzweigten Bedingungen konnten drei Hauptfehlerquellen identifiziert werden. Zunächst erzeugen die Vorverarbeitungsstufen teilweise fehlerhafte Ergebnisse. Diese Fehler pflanzen sich fort und führen dazu, dass die zuvor beschriebenen Heuristiken nicht mehr greifen. Der häufigste Fehler in diesem Kontext sind fehlerhafte Chunk-Etiketten der Vorverarbeitungsstufe zur seichten Sprachverarbeitung. Fehlerhafte Chunk-Etiketten führen unmittelbar dazu, dass die Grammatiken zur Extraktion der Gliedsätze keine oder falsche Ergebnisse erzeugen. Das zweite Problem sind grammatikalisch fehlerhafte

Tabelle 7.22: Ergebnisse der Evalution der Agenten zur Synthese von Nebenläufigkeiten und Schleifen: Werte ohne Klammern entsprechen den Ergebnissen *ohne* Verwendung, Werte in Klammern denen *mit* Verwendung des Agenten zur Korreferenzanalyse von Nominalphrasen.

Kontrollstruktur-Art	Präzision	Ausbeute	F_1
Schleife	1,000 (1,000)	0,514 (0,686)	0,679 (0,814)
Nebenläufigkeit	0,889 (0,750)	0,800 (0,800)	0,842 (0,774)

Äußerungen. Einige Probanden lieferten (wahrscheinlich aufgrund unzureichender Englischkenntnisse) teilweise grammatikalisch fehlerhafte Beschreibungen. Dies führt ebenso zu fehlerhaften Extraktionen. Immerhin konnten die verwendeten Grammatiken in vielen Fällen die Konditional- und abhängigen Gliedsätze zumindest teilweise korrekt extrahieren. Zuletzt verwendeten einige Probanden in ihren Beschreibungen zwar grammatikalisch korrekte aber für den Ansatz unerwartete Ausdrücke. Diese konnten von den entworfenen Grammatiken nicht oder nur teilweise erfasst werden. Ein Beispiel hierfür sind Nominalphrasen, die direkt nach einem Adverb stehen. In den meisten Fällen weist ein solches Konstrukt darauf hin, dass mit der Nominalphrase ein neuer Gliedsatz beginnt (so auch in den Grammatiken erfasst), wie beispielsweise in der Äußerung *take the orange juice* [*now*]$_{\text{ADVP}}$ [*it*]$_{\text{NP}}$ *is located* [...]. In einigen Fällen kann die Nominalphrase aber auch dem vorherigen Gliedsatz zugeordnet werden, wie in folgendem Beispiel: *otherwise bring me* [*just*]$_{\text{ADVP}}$ [*the orange juice*]$_{\text{NP}}$. Die verwendeten Grammatiken erfassen bisher nur den ersten (gebräuchlicheren) Fall. Eine Erweiterung der Grammatiken könnte in diesen Fällen zu einer weiteren Verbesserung der Ergebnisse führen. Die Verwendung von Korreferenzinformationen zur Erweiterung des Umfangs der Bezugsrahmen verbessert die Ausbeute deutlich (um 7,6% von 0,803 auf 0,864). Die Ergebnisse je Gliedsatz-Art zeigen, dass vor allem die *Dann*-Gliedsätze profitieren (Verbesserung um 14,2% von 0,795 auf 0,908); aber auch die seltener auftretenden und schwieriger zu erkennenden *Andernfalls*-Gliedsätze zeigen eine Verbesserung; die relative Steigerung übersteigt sogar die der *Dann*-Gliedsätze (Verbesserung um 14,6% von 0,405 auf 0,464). Der Grund für die deutlichen Verbesserungen ist folgender: Viele Transkriptionen enthalten längere Sequenzen von abhängigen Gliedsätzen, die aus mehreren Phrasen zusammengesetzt sind. Der syntaktische Ansatz zur Extraktion der Gliedsätze kann nur einfache Kombinationen von Phrasen erzeugen. Mithilfe der Heuristik, die auf Korreferenzketten basiert, können längere Phrasenketten, die aus einer Vielzahl von Aussagen bestehen, extrahiert werden. Auch die Präzision verschlechtert sich durch die Hinzunahme der Heuristik nicht, wie zu erwarten gewesen wäre; stattdessen steigt die Präzision sogar leicht an (um 0,4% von 0,930 auf 0,934).

7.6.5.2 Evaluation der Agenten zur Erkennung von Nebenläufigkeit und Schleifen

Die Ergebnisse der Evaluation der Agenten zur Erkennung von Nebenläufigkeit und Schleifen sind in Tabelle 7.22 aufgeführt. Wieder entsprechen die Werte in Klammern den Ergebnissen unter Verwendung von Korreferenzinformationen. Werden *keine* Korreferenzinformationen verwendet, erreicht das Verfahren mit 0,842 einen ähnlich guten Wert wie der Ansatz zur Erkennung von verzweigten Bedingungen (0,862). Die Präzision erreicht auch hier höhere Werte als die Ausbeute

(0,889 gegenüber 0,800). Die Verwendung von sekundären Typen für Schlüsselwörter bzw. -phrasen verbessert die Ausbeute, ohne zusätzliche *falsch positive* Ergebnisse zu erzeugen. Insgesamt extrahiert das Verfahren neun von zehn Nebenläufigkeitsstrukturen; nur drei Ereignisse werden zusätzlich extrahiert (*falsch positive*) und fünf nicht erkannt (*falsch negative*). Die Verwendung von Korreferenzinformationen bringt für die Erkennung von Nebenläufigkeit keine Vorteile. Die Ausbeute bleibt gleich, während die Präzision sogar sinkt (um 15,6% von 0,889 auf 0,750).

Wie in Tabelle 7.20 nachzuvollziehen, beinhalten die Transkriptionen in Summe zwanzig beschriebene Ereignisse in zehn Nebenläufigkeitsstrukturen. Das bedeutet, die nebenläufigen Ereignisse treten immer paarweise auf und jeder nebenläufige Gliedsatz enthält nur genau ein Ereignis. Da die Heuristik zur Erweiterung des Umfangs der Bezugsrahmen anhand von Korreferenzketten darauf abzielt, bereits erkannten Gliedsätzen weitere Ereignisse zuzuordnen, ist es unmöglich, die Ausbeute für diese Transkriptionen zu verbessern. Stattdessen werden fünf *falsch positive* Ereignisse hinzugefügt und die Präzision sinkt entsprechend. Dieses Ergebnis kann teilweise auf den geringen Umfang des Evaluationsdatensatzes zurückgeführt werden. Eine Wiederholung mit mehr Äußerungen, die nebenläufige Gliedsätze beinhalten, welche aus mehr als einem Ereignis bestehen, könnte zu deutlich anderen Ergebnissen führen.

Für die Erkennung von Schleifen liegt der Wert für das F_1-Maß nur bei 0,679. Zwar erreicht die Präzision den optimalen Wert von 1,000, die Ausbeute ist mit 0,514 jedoch zu niedrig. Nur sechs der neun Schleifenstrukturen werden erkannt. Zusätzlich werden elf eigentlich abhängige Ereignisse (des *Schleifenkörper*-Gliedsatzes) nicht zugeordnet. Fünf dieser Fälle können darauf zurückgeführt werden, dass der Ansatz nicht nach Ereignissen gesucht hat, da bereits die Schlüsselwortsuche fehlgeschlagen ist. Eine eingehende Analyse dieser Fälle hat gezeigt, dass die Transkriptionen unerwartete Formulierungen enthielten, wie „for every piece you find in there“ und „if there are still any pieces left repeat the step“. Andere Formulierungen sind ungewöhnlich bzw. unüblich und können auf die unzureichenden Englischkenntnisse der Probanden zurückgeführt werden. Nichtsdestotrotz sollte eine höhere Ausbeute angestrebt werden, beispielsweise durch Erweiterungen der Grammatiken oder zusätzliche Schlüsselwörter und -phrasen. Die Verwendung von Korreferenzketten verbessert die Ausbeute bereits deutlich (um 33% von 0,514 auf 0,686). Damit steigt auch der Wert für das F_1-Maß auf 0,814. Mithilfe der Korreferenzketten können alle Ereignisse dem entsprechenden *Schleifenkörper*-Gliedsatzes zugeordnet werden, sofern überhaupt (mithilfe der Schlüsselwortsuche) eine Schleife erkannt wurde. Die Heuristik erzeugt zudem keine *falsch positiven* Ergebnisse, die Präzision bleibt dementsprechend bei 1,000.

Zusammenfassend lässt sich festhalten, dass alle Agenten zur Synthese von Kontrollstrukturen gute Ergebnisse erzielen. Bei optimaler Konfiguration liegen die Werte für das F_1-Maß bei allen Agenten über 0,8 (0,814 bis 0,898). Für alle Agenten gilt zudem, dass die Präzision höher ausfällt als die Ausbeute. Bei den Agenten zur Erkennung von verzweigten Bedingungen und Schleifen kann die Ausbeute mithilfe von Korreferenzinformationen deutlich gesteigert werden; für den Agenten zur Erkennung von Nebenläufigkeit bringen Korreferenzinformationen hingegen keine Verbesserung, stattdessen sinkt die Präzision.

7.7 Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache

Das Programmieren ist ein kreativer Prozess, in dem ein Programmierer einer Maschine schrittweise neue Fähigkeiten verleiht. Hierzu kombiniert der Programmierer vorhandene (zuvor programmierte oder grundlegende) Funktionalitäten, um neue zu erschaffen. Für die Programmierung in natürlicher Sprache gilt es, diesen kreativen Schaffensprozess zu übertragen. Der aktuelle Stand der Technik sieht aber hauptsächlich die Nutzung von Sprachschnittstellen zum Aufruf vorgefertigter Funktionalität vor. Im besten Fall können die Ergebnisse zweier Anfragen vereinigt werden. Der Nutzer des Sprach-gesteuerten Systems hat keine Möglichkeit, die Funktionalität zu erweitern. Neue Funktionen werden weiterhin nur durch Entwickler implementiert und über die Schnittstelle zur Verfügung gestellt. Zukünftige Sprach-gesteuerte Systeme sollten hingegen den Nutzern ermöglichen, eigene Funktionserweiterungen zu definieren – am besten auf die gleiche Art und Weise, wie die Systeme genutzt werden, durch natürlichsprachliche Anweisungen. Daher soll ProNat neben der Erzeugung skriptartiger Programme auch die Definition neuer Funktionalität ermöglichen. Um neue Funktionen aus natürlichsprachlichen Äußerungen ableiten zu können, muss zunächst erkannt werden, wann ein Nutzer nicht nur Anweisungen gibt, sondern eine neue Funktion lehren möchte. Wird eine solche Lehrsequenz erkannt, muss zudem ihre Semantik analysiert werden. Um zu verstehen, wie Laien das Lehren einer neuen Funktionalität verbalisieren, wurden mittels einer Studie zunächst Daten gesammelt und analysiert [WST20a; WST20b]. Die textuellen Beschreibungen bilden den in Abschnitt 5.5.3 beschriebenen Online-Datensatz. Aus den Erkenntnissen der Datenanalyse wird ein Ansatz zur Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache abgeleitet und ein ProNat-Agent implementiert [Wei+20c; Wei+20d]: Es wird ein hierarchische Klasifikationsaufgabe definiert, bei der die erste Stufe zwischen lehrenden und nicht lehrenden Äußerungen unterschieden werden soll. In der zweiten Stufe wird die semantische Struktur von lehrenden Äußerungen analysiert. Für beide Stufen wurden eine Reihe von Verfahren des maschinellen Lernens implementiert und vermessen. Mithilfe der so entstandenen Modelle kann der ProNat-Agent ungesehene Eingaben klassifizieren.

7.7.1 Analyse der Datensammlung

Die Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache ist bisher nicht Gegenstand der Forschung. Ebenso existieren keine linguistischen Theorien zur Form und Struktur, wie dies beispielsweise bei Konditionalsätzen der Fall ist, welche die Grundlage zur Erkennung von bedingten Verzweigungen in natürlichsprachlichen Äußerungen bilden (siehe Abschnitt 7.6.1). Daher wurde zu Analysezwecken zunächst eine Datensammlung angelegt, die natürlichsprachliche Lehrsequenzen enthält. Wie in Abschnitt 5.5.3.1 erläutert, sollten Probanden in vier Szenarien einem humanoiden Haushaltsroboter mittels textueller Beschreibungen vier neue Funktionen beibringen: *jemanden Grüßen*, *Kaffee zubereiten*, *Getränke servieren* und *einen Tisch für zwei eindecken*. Der Datensatz enthält 3168 Beschreibungen von 870 Probanden. Anhand des Datensatzes werden sprachliche und inhaltliche Charakteristika von natürlichsprachlichen Lehrsequenzen untersucht.

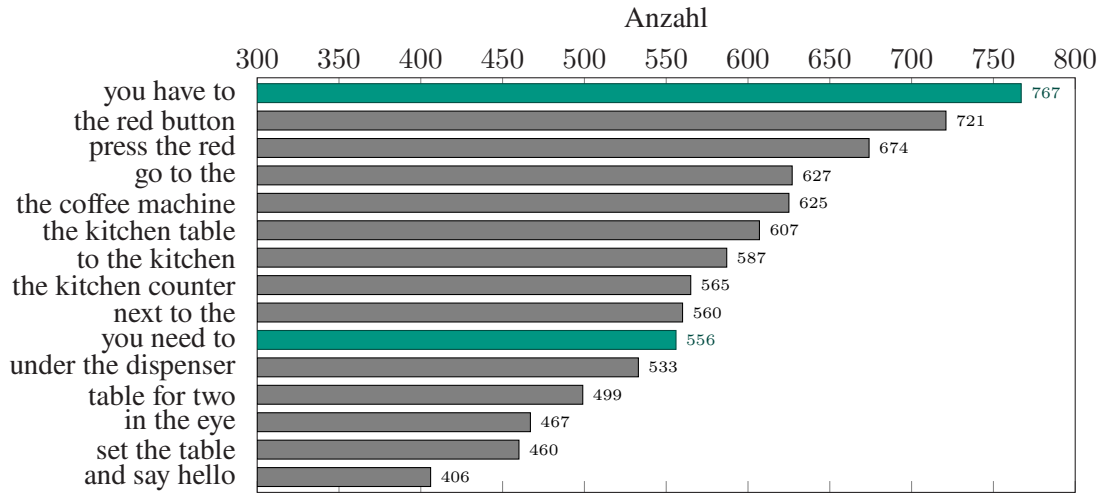


Abbildung 7.13: Häufigste Trigramme der Online-Datensammlung: Die Abbildung zeigt die absoluten Häufigkeiten der fünfzehn häufigsten Trigramme des Datensatzes (Beschreibungen zu den vier Szenarien der Online-Datensammlung, siehe Abschnitt 5.5.3).

Es zeigt sich, dass Probanden unterschiedliche erzählerische Perspektiven einnehmen. Die meisten geben ihre Beschreibungen aus Sicht eines Lehrers ab, der ein Kind unterrichtet. Viele Probanden wechseln in die Rolle eines naiven Endnutzers; nur wenige verwendeten technische Beschreibungen, wie es ein Software-Entwickler tun würde. Hinsichtlich des Inhalts der Beschreibungen fällt auf, dass etwa ein Drittel lediglich Anweisungsfolgen beschreiben und keine neue Funktion lehren. Diese Beschreibungen stellen keine Lehrsequenzen dar und können nicht dafür verwendet werden, ein System um neue Funktionen zu erweitern[39]. Folglich müssen Lehrsequenzen von bloßen Anweisungsfolgen unterschieden werden. Die Beschreibungen des Datensatzes, die lehrende Anweisungen enthalten, wurden näher untersucht. Unter anderem wurden häufige Wortfolgen (das heißt N-Gramme) untersucht.

Abbildung 7.13 zeigt die fünfzehn häufigsten Trigramme, die im Datensatz enthalten sind. Die Meisten sind Phrasen, die entweder ohne Bedeutung sind oder Anweisungen für bestimmte Szenarien enthalten. Zwei der Trigramme leiten jedoch eindeutig unabhängig von einem konkreten Szenario Lehrsequenzen ein: *you have to* und *you need to* (in der Abbildung grün eingefärbt). Weiterführende Betrachtungen fördern häufig verwendete Phrasen zur Formulierung einer Lehrabsicht zutage. Diese lauten wie folgt:

- „*in order to* [...] *you have to* [...]“
- „[...] *means you have to* [...]“
- „*if you want to* [...] *you need to* [...]“
- „*we are going to learn how to* [...]“

[39] Typische Beispiele führ derartige Anweisungsfolgen ohne Lehraspekt sind das erste und und fünfte Beispiel aus Tabelle 5.9 in Abschnitt 5.5.3.2.

Das bedeutet, es gibt Regularitäten in der Wortwahl und Struktur bei der Formulierung von Lehrsequenzen. Hinsichtlich des Aufbaus der Lehrsequenzen zeigen sich weitere Probanden-übergreifende Übereinstimmung. Die Beschreibungen enthalten häufig eine kurze Begrüßung, wie *Hi Armar* oder *Hey Robot*. Anschließend wird die Absicht eine neue Funktion zu lehren geäußert, beispielsweise mithilfe einer der zuvor erwähnten Phrasen, und die neue Funktion benannt. In der Folge werden zumeist die Teilschritte beschrieben, die der Roboter ausführen soll, wenn die neue Funktion aufgerufen wird. Zum Szenario *Kaffee zubereiten* lautet eine der Beschreibungen zum Beispiel wie folgt: „You have to place the cup under the dispenser and press the red button to make coffee.“. Die Reihenfolge dieser Bestandteile von Lehrsequenzen variiert jedoch stark. Häufig wird beispielsweise der Wunsch eine Funktion zu lehren erst zum Ende der Beschreibung geäußert. In anderen Beschreibungen wird die Bestrebung etwas zu lehren mehrfach formuliert, beispielsweise zu Beginn und bekräftigend zum Ende der Beschreibung. Die Beschreibung der Teilschritte erfolgt in einigen Fällen nicht zusammenhängend, sondern wird unterbrochen, beispielsweise durch Grußformeln (oder andere nicht relevante Äußerungen). Das bedeutet, Lehrsequenzen enthalten häufig ähnliche Strukturen; diese werden jedoch unterschiedlich zusammengefügt, sind in konkreten Beschreibungen mehrfach oder gar nicht enthalten und gegebenenfalls nicht zusammenhängend.

7.7.2 Generelles Vorgehen

Aus der Analyse der Datensammlung können zusammenfassend folgende Erkenntnisse gezogen werden: Erstens lassen sich die natürlichsprachlichen Äußerungen unterteilen in solche, die eine Lehrsequenz enthalten, und andere, die lediglich eine Handlungsanweisung darstellen. Zweitens bestehen Lehrsequenzen aus den folgenden semantischen Bestandteilen:

- **Deklaration:** Die Deklaration umfasst den explizit geäußerten Wunsch (Intention), eine neue Funktion zu lehren, einen Namen für die neue Funktionalität und gegebenenfalls Parameter. Im programmatischen Sinn entspricht der deklarative Teil der Methodensignaturdefinition. Ein Beispiel für einen solchen deklarativen Teil einer Lehrsequenz ist: „[In order to]$_{Intention}$ [set the table]$_{Name}$ [for two]$_{Parameter}$“
- **Spezifikation:** Die Spezifikation ist die Beschreibung der nötigen Arbeitsschritte (Aktionen), um die neue Funktionalität zu erlernen. Im programmatischen Sinn entspricht der spezifizierende Teil einem Methodenrumpf. Ein Beispiel für einen spezifizierenden Teil ist: „[go to the cupboard]$_{Aktion1}$ [open it]$_{Aktion2}$ [and take out two plates]$_{Aktion3}$“
- **Sonstiges:** Sonstige Aussagen sind alle anderen Arten von Aussagen, die für das Verständnis einer Lehrsequenz irrelevant sind. Dies schließt unter anderem Grußformeln, das Lehren von Weltwissen oder Feststellungen ein. Beispiele für in diesem Kontext irrelevante Aussagen sind: „hey robot“ und „wine is a beverage people like to drink“

Die jeweiligen Bestandteile treten an beliegen Stellen und in beliebiger Reihenfolge in den natürlichsprachlichen Äußerungen auf. Außerdem wird die Deklaration häufig in zwei oder mehr (nicht aufeinanderfolgende) Aussagen aufgeteilt oder an unterschiedlichen Stellen der Äußerung (in anderen Worten) wiederholt. Die Beschreibung der Arbeitsschritte (Spezifikation) ist beliebig

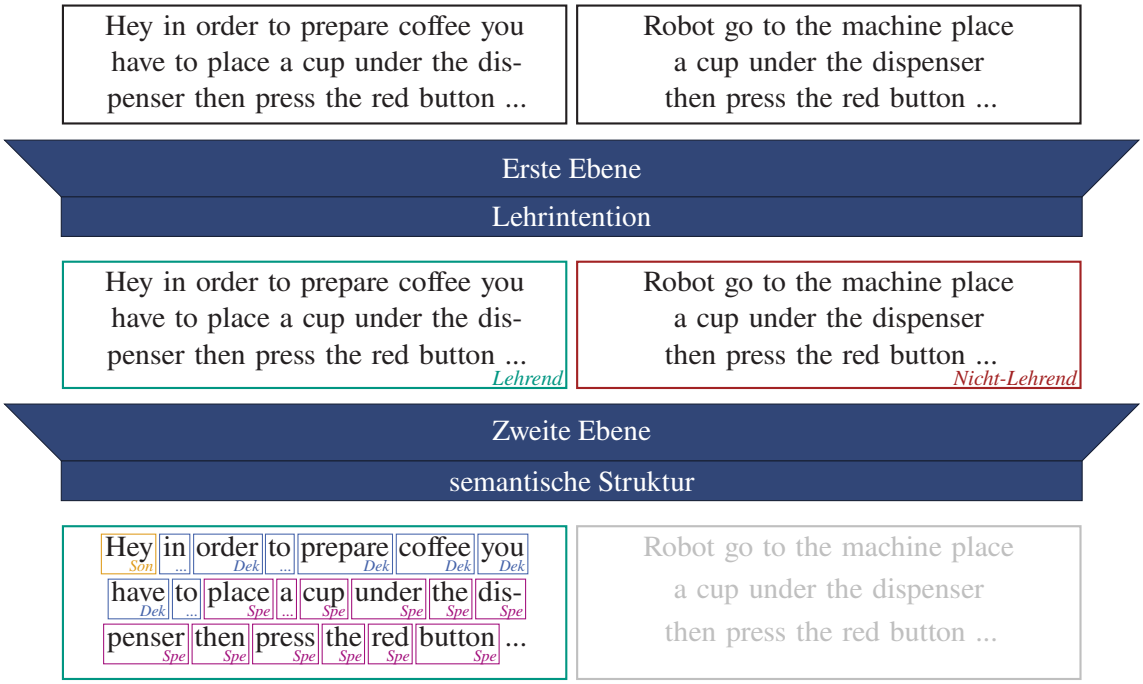


Abbildung 7.14: Schematische Darstellung des zweistufigen, hierarchischen Klassifikationsansatzes: Die abgebildeten Beispieläußerungen sind synthetisch, wurden jedoch echten nachempfunden.

lang und gegebenenfalls nicht zusammenhängend; häufig unterbrechen deklarative Aussagen oder Aussagen der Kategorie *Sonstiges* die Beschreibung.

Aus den Erkenntnissen der Analyse der Datensammlung wird das Vorgehen des Agenten zur Erkennung und Analyse von Lehrsequenzen in gesprochener Sprache abgeleitet. Da durch die Online-Datensammlung ein umfangreicher Datensatz gebildet wurde (siehe Abschnitt 5.5.3), kann als Ansatz maschinelles Lernen gewählt werden. Wie zuvor beschrieben, untergliedert sich die Aufgabe in zwei Teile: erstens die Unterscheidung von Lehrsequenzen und Handlungsbeschreibungen und zweitens die Analyse der semantischen Strukturen von Lehrsequenzen. Es bietet sich daher an, die Aufgabe als hierarchische Klassifikation anzusehen[40]. Im Agenten werden folgende Klassifikationsschritte verwendet:

- **Erste Ebene:** Auf der ersten Ebene wird klassifiziert, ob eine Äußerung die Intention enthält, eine neue Funktion zu lehren, oder nicht. Als Etiketten für diese binäre Klassifikation werden *Lehrend* und *Nicht-Lehrend* verwendet.

- **Zweite Ebene:** Auf der zweiten Ebene werden die semantischen Bestandteile von Lehrsequenzen (wie zuvor definiert) klassifiziert. Betrachtet werden ausschließlich Äußerungen, die auf der ersten Ebene als *Lehrend* klassifiziert wurden. Die verwendeten Etiketten für diese ternäre Klassifikation sind *Deklaration*, *Spezifikation* und *Sonstiges*.

[40] Neben der inhärenten logischen Teilung der Aufgabe in zwei aufeinanderfolgende Klassifikationsaufgaben, haben Forscher wie Cohen et al. nachgewiesen, dass hierarchische Klassifikationsansätze monolithischen bei ähnlich gearteten Aufgaben überlegen sind[CRM07].

Abbildung 7.14 zeigt schematisch die zweistufige Klassifikation. Um die Klassifikationsschritte in dieser Weise durchführen zu können, muss der Datensatz zunächst etikettiert werden. Die Annotation der Etiketten wurde gemeinsam vom Autor der Dissertation und einer Master-Absolventin des Lehrstuhls durchgeführt. Zunächst wurden die binären Etiketten der ersten Klassifikationsebene je Äußerung annotiert. Hierbei wurde ein strikter Ansatz verfolgt; nur Äußerungen, die eindeutig als Lehrsequenz zu erkennen sind, erhielten das Etikett *Lehrend*. Allerdings wurden auch Äußerungen zugelassen, bei denen die Lehrintention nur implizit formuliert wird. Ein Beispiel hierfür ist die Äußerung „to make coffee“ in der nur das Wort *to* darauf hinweist, dass hier eine Funktion (*make coffee*) gelehrt werden soll. Die Annotation der Etiketten der zweiten Klassifikationsebene erfolgt wortweise. Hierzu wurde zunächst versucht, alle deklarativen Teile der Lehrsequenz zu identifizieren. Anschließend wurden die spezifizierenden Teile annotiert; alle übrigen Wörter erhielten das Etikett *Sonstiges*. Bei der Annotation der Etiketten wurde darauf geachtet Wörter eher als relevant (*Deklaration* und *Spezifikation*) denn als irrelevant (*Sonstiges*) zu bewerten. Während der zweiten Annotationsphase konnten einige *falsch positive* Etiketten der ersten Ebene identifiziert werden; immer dann, wenn nicht mindestens ein Wort das Etikett *Deklaration* erhält, kann es sich nicht um eine Lehrsequenz handeln. Tabelle 7.23 gibt eine Übersicht über die Summe und Verteilung der angebrachten Etiketten. Die Etiketten beider Mengen sind ungleich verteilt; dies kann gegebenenfalls die Güte der gelernten Modelle beeinflussen. Einseitig verteilte Etikettenmengen führen häufig dazu, dass die Modelle während des Trainings auf die dominante Klasse überangepasst (engl. *over-fitting*) werden. Diese Problematik betrifft insbesondere die zweite Klassifikationsebene, bei der die Klasse *Spezifikation* mit einem Anteil von 76% der Etiketten die anderen Klassen deutlich dominiert. Ein weiterer Faktor, der beim Einsatz von Techniken aus dem Bereich des maschinellen Lernens beachtet werden muss, ist die Länge der Eingabe. Die Komplexität der meisten Verfahren steigt proportional zur (maximal zugelassenen) Eingabelänge. Dies gilt insbesondere für neuronale Netze, bei denen die Eingabelänge die Anzahl der Neuronen der Eingabeschicht bestimmt. Da die Anzahl der Neuronen der Eingabeschicht bei neuronalen Netzen zum Zeitpunkt der Konstruktion festgelegt wird, muss die maximale Eingabelänge vorab bestimmt werden. Eine Möglichkeit ist, die Eingabelänge so zu definieren, dass sie der Länge der längsten natürlichsprachlichen Äußerung des Datensatzes entspricht; wie Tabelle 7.24 zeigt, würde dies 312 entsprechen. Da aber nur wenige Datenpunkte derart lang sind, empfiehlt es sich stattdessen die oberen Quantile der Verteilung zu betrachten. Wie Tabelle 7.24 entnommen werden kann, stellt das 99,5%-Quantil einen guten Kompromiss zwischen Allgemeingültigkeit (Anzahl der Eingaben, die vollständig verarbeitet werden) und Ökonomie (Länge der Eingabe) des Ansatzes dar. Die Wahl für den ProNat-Agenten zur Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache fällt daher auf eine Eingabelänge von 135 Wörtern. Mit dieser Wahl können 99,5% der Eingaben vollständig verarbeitet werden (die übrigen werden nach 135 Wörtern abgeschnitten). Aus dem Datensatz müssen anschließend Trainingsinstanzen generiert werden; laut Mihalcea erfordert dies drei aufeinanderfolgende Schritte [Mih07]:

1. Sammlung und Vorverarbeitung eines Datensatzes

2. Extraktion der Trainingsinstanzen

3. Vorverarbeitung der Trainingsinstanzen

Tabelle 7.23: Absolute und relative Häufigkeiten (in Klammern) der Etiketten, die zur Klassifikation für die Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache verwendet wurden.

binär			ternär			
Lehrend	Nicht-Lehrend	Σ	Deklaration	Spezifikation	Sonstiges	Σ
1998 (0,63)	1170 (0,37)	3168	15559 (0,21)	57156 (0,76)	2219 (0,03)	74934

Tabelle 7.24: Datensatzstatistiken zu Wortverteilung der Online-Datensammlung: Neben der minimalen und maximalen Anzahl verwendeter Wörter, dem arithmetischen Mittel (ø) sowie der Standardabweichung (σ), sind auch die oberen Quantile der Verteilung angegeben.

				Quantile		
Min.	Max.	ø	σ	99,0%	99,5%	99,9%
1.0	312	35.43	22.48	117	135	232

Auch wenn der Datensatz bereits vorhanden ist, müssen die einzelnen Datenpunkte vorverarbeitet werden (erster Schritt), um diese optimal für Verfahren zum maschinellen Lernen vorzubereiten. Folgende Anpassungen werden vorgenommen:

- Umwandlung in Kleinbuchstaben, z. B. *Hello* zu *hello*
- Auflösung von Kontraktionen, z. B. *don't* zu *do not*
- Umwandlung von Ordnungszahlen, z. B. 1^{st} zu *first*
- Entfernung von Aufzählungszeichen, Interpunktion und Disfluenzen
- Korrektur von typographischen (nicht aber von grammatikalischen) Fehlern, z. B. *thng* zu *thing*

Um die Trainingsinstanzen zu extrahieren (zweiter Schritt), können einfach alle etikettierten Instanzen des Datensatzes (wie zuvor beschrieben) verwendet werden. Eine weitere Vorverarbeitung (dritter Schritt) ist nur für die Trainingsinstanzen der zweiten Klassifikationsebene notwendig. Zunächst werden alle Wörter lemmatisiert und ein Datensatz mit und einer ohne Lemmatisierung erstellt. Zusätzlich wird je ein Datensatz mit und einer ohne Stoppwörter erstellt (siehe Abschnitt 2.3.5).

Zuletzt müssen die Instanzen in eine numerische Repräsentation überführt werden. Hierzu werden auf Ausgabeseite die Etiketten als 1-aus-n-Vektor (engl. *one-hot vector*) dargestellt. Für die natürlichsprachlichen Wörter auf der Eingabeseite werden zwei Repräsentationen implementiert. Zum einen werden *Bag-of-Words*-Vektoren erzeugt (siehe Abschnitt 2.3.16). Zum anderen werden 300-dimensionale Worteinbettungen nach dem *fastText*-Verfahren generiert [Jou+17]; Zusätzlich wird das von *Facebook Research* vortrainierte *fastText*-Modell verwendet [Boj+17; Mik+18][41]. Das Modell umfasst zwei Millionen 300-dimensionale Worteinbettungen und wurde mithilfe eines

[41] Verwendet wurde das Modell `crawl-300d-2M.vec`: `https://fasttext.cc/docs/en/english-vectors.html`, zuletzt besucht am 24.02.2021.

Common-Crawl-Datensatz erzeugt[42]. Der Datensatz muss anschließend in Teilmengen aufgeteilt werden. Ein größerer Teil wird für gewöhnlich zum Training des Modells verwendet; ein kleinerer wird zurückgehalten und zum Testen verwendet. Üblicherweise teilt man einen Datensatz in einem Verhältnis von achtzig zu zwanzig in eine Trainings- und eine Testmenge. Der gesamte Datensatz wird hierfür randomisiert den jeweiligen Mengen zugeteilt. Da viele Verfahren des maschinellen Lernens neben der Trainings- eine zusätzliche Validierungsphase verwenden, in der Hyperparameter angepasst werden (siehe Abschnitt 2.2.2), wird die Trainingsmenge erneut randomisiert im Verhältnis achtzig zu zwanzig geteilt. Diese Art der Aufteilung, die auch hier Anwendung findet, wird im Folgenden als *zufällige Aufteilung* bezeichnet.

Leider bildet die zufällige Aufteilung die Realität nur schlecht ab. Im tatsächlichen Einsatz wird ein Modell immer auf historischen Daten gelernt und für neue, ungesehene und gegebenenfalls unerwartete angewandt. Um diese Situation besser nachzustellen, wurde eine zweite Aufteilung umgesetzt. Wie zuvor und in Abschnitt 5.5.3 beschrieben, besteht der Datensatz aus Äußerungen zu vier unterschiedlichen Szenarien. Für die zweite Art der Aufteilung des Datensatzes werden die Äußerungen eines Szenarios als Testmenge bestimmt und die übrigen für das Training bzw. die Validierung verwendet. Wie zuvor werden die Trainigs- und die Validerungsmengen im Verhältnis achtzig zu zwanzig aufgeteilt. Im Folgenden wird diese Art der Aufteilung als *Szenario-basierte Aufteilung* bezeichnet. Der Vorteil der Szenario-basierten Aufteilung besteht darin, dass besser bemessen werden kann, wie die gelernten Modelle auf zuvor ungesehene Eingaben reagieren, die sich konzeptionell von denen der Trainingsmenge unterscheiden. Zwar enthalten die natürlichsprachlichen Beschreibungen je Szenario viele Überschneidungen hinsichtlich der geäußerten Aktionen, Ereignissen und behandelten Objekten, zwischen den Szenarien unterscheiden sie sich aber deutlich. Dadurch kann mit dieser Art der Aufteilung bemessen werden, wie gut ein Klassifikator generalisiert und in der Lage ist, die generelle Struktur einer Lehrsequenz zu verstehen, anstatt nur bestimmte (gegebenenfalls Domänen-abhängige) Phrasen auswendig zu lernen.

Mit den so vorbereiteten Daten können Verfahren, die maschinelles Lernen verwenden, eingelernt und getestet werden. Als Vergleichsgrundlage für beide Klassifikationsebenen wurde der sogenannte *Keine-Regel-Klassifikator* (engl. *zero rule*, kurz *ZeroR*) verwendet. Dieser weist jeder Instanz die dominante Klasse der Trainingsmenge zu. So kann überprüft werden, ob die trainierten Modelle besser sind als eine naive Lösung. Gerade bei ungleichmäßig verteilten Klassen (wie es auch für beide Klassifikationsebenen der Fall ist) neigen viele Verfahren dazu, sich ähnlich wie der *ZeroR*-Klassifikator zu verhalten und das Modell zugunsten der dominanten Klasse überanzupassen. Dementsprechend ist der *ZeroR*-Klassifikator in diesen Fällen eine gute Vergleichsgrundlage. In den beiden nachfolgenden Abschnitten werden die Implementierungen der Klassifikationsebenen beschrieben. Die Ergebnisse der jeweiligen Verfahren auf den Testmengen wird zur Bemessung der Qualität genutzt. Daher werden die Ergebnisse direkt zusammen mit der Beschreibung der jeweiligen Klassifikationsebene anstatt in einem separaten Abschnitt zur Evaluation diskutiert.

[42] *Common Crawl* ist eine gemeinnützige Organisation, die mithilfe von *Web-Crawling*-Werkzeugen Texte aus dem Internet sammelt, zu unbearbeiteten Korpora zusammenführt und diese der Öffentlichkeit kostenlos zur Verfügung stellt: `https://commoncrawl.org/`, zuletzt besucht am 24.02.2021.

7.7.3 Unterscheidung von Lehrsequenzen und Handlungsbeschreibungen

Die erste Klassifikationsaufgabe ist eine sogenannte Sequenz-zu-Einzel-Etiketten-Aufgabe (engl. *sequence-to-single-label task*); das bedeutet, einer Eingabesequenz – hier die Wörter der Äußerung – wird in ihrer Gesamtheit ein Etikett zu gewiesen – hier entweder *Lehrend* oder *Nicht-Lehrend*. Die Vorstudie hat gezeigt, dass Lehrsequenzen sehr unterschiedlich formuliert werden. Oft kann die Intention der Probanden, eine neue Funktion zu lehren, nur anhand einzelner Wörter festgemacht werden, wie in *do A and B* ***to*** *prepare coffee*. Die Klassifikationsaufgabe ist dadurch alles andere als unkompliziert und die Literatur kann keine Empfehlungen für ein bestimmtes Verfahren liefern, da eine vergleichbare Aufgabe nicht identifiziert werden konnte. Aus diesem Grund wurden eine Reihe von klassischen und modernen Verfahren (basierend auf neuronalen Netzen) implementiert, um bestmöglich einschätzen zu können, welches Verfahren sich für die gestellte Aufgabe eignet. Für die neuronalen Netze wurden zudem unterschiedliche Konfigurationen getestet; das bedeutet, es wurden zusätzliche Schichten hinzugefügt und Hyperparmeter systematisch variiert.

Folgende klassische Verfahren wurden implementiert: Entscheidungsbaum (engl. *decision tree*), (zufällige) unkorrelierte Entscheidungsbäume (engl. *random forest*), Stützvektormethode (engl. *support vector maschine*), Naïve-Bayes und logistische Regression. Als Eingabe-Merkmale für die klassischen Verfahren wurden *Bag-of-Words*-Vektoren, sowie Tri- und Tetragramme verwendet (um den Wortkontext zu modellieren). Die Verfahren wurden anschließend auf den lemmatisierten und nicht lemmatisierten Datensätzen trainiert, validiert und getestet. Allerdings war die Genauigkeit aller Verfahren auf den lemmatisierten Datensätzen höher. Dasselbe gilt für Stoppwörter; alle Verfahren erzielten bessere Ergebnisse, wenn Stoppwörter aus den Äußerungen entfernt wurden. Daher werden in Tabelle 7.25 nur die Ergebnisse für den *lemmatisierten* Datensatz *ohne Stoppwörter* gezeigt. Für alle Verfahren wird die Genauigkeit für die zufällige Aufteilung des Datensatzes und die Szenario-basierte Aufteilung bemessen. Dabei wird jeweils die Genauigkeit auf der Validierungs- und der Testmenge angegeben[43]. Als Vergleichsgrundlage wird wie zuvor beschrieben der *ZeroR*-Klassifikator verwendet.

Die Werte für den *ZeroR*-Klassifikator sind für beide Aufteilungen ähnlich. Das spricht dafür, dass sich die Verteilung der beiden Klassen sich zwischen den Szenarien kaum unterscheidet. Die Ergebnisse der Verfahren unterscheiden sich jedoch deutlich je nach Aufteilung des Datensatzes. Auf den zufällig aufgeteilten Daten sind die Genauigkeiten auf der Testmenge bei allen Verfahren in derselben Größenordnung wie die Genauigkeiten auf den Validierungsmengen. Das beste Ergebnis wird dabei vom Klassifikator erzielt, der logistische Regression zur Problemmodellierung verwendet; eine Genauigkeit von 94,7% auf der Testmenge ist ein überraschend gutes Ergebnis für ein einfaches Lernverfahren. Die Ergebnisse für die Szenario-basierte Aufteilung sind weniger positiv. Die Klassifikationsqualität aller Verfahren nimmt deutlich ab, drei von fünf Verfahren fallen sogar unter

[43] Für die Bewertung der Verfahren ist die Genauigkeit auf der Testmenge ausschlaggebend. Die Genauigkeit auf der Validierungsmenge ist nichtsdestotrotz von Interesse, falls diese bereits zum Trainingszeitpunkt zur Verfügung steht. Das bedeutet, die Genauigkeit auf der Validierungsmenge kann genutzt werden, um die Genauigkeit des jeweiligen Verfahrens auf der Testmenge abzuschätzen. Betrachtet man also beide Genauigkeiten gemeinsam, kann bemessen werden, wie gut sich die Genauigkeit auf der Validierungsmenge für das jeweilige Verfahren als Schätzer eignet.

Tabelle 7.25: Erzielte Genauigkeiten der klassischen Verfahren des maschinellen Lernens: Unterschieden wird nach *zufälliger* und *Szenario-basierter* Aufteilung des Datensatzes sowie nach Validierungs- und Testmenge.

	Zufällig		Szenario	
Verfahren	Val.	Test	Val.	Test
Entscheidungsbaum	0,893	0,903	0,861	0,719
Zufällige Entscheidungsbäume	0,917	0,909	0,893	0,374
Stützvektormethode	0,848	0,861	0,870	0,426
Naïve-Bayes	0,771	0,801	0,765	0,300
Logistische Regression	0,927	0,947	0,891	0,719
Vergleichsgrundlage (*ZeroR*)	——	0,573	——	0,547

die Vergleichsgrundlage. Der Naïve-Bayes-Klassifikator etikettiert beispielsweise nur noch 30% der Instanzen richtig. Lediglich die Ergebnisse der Klassifikatoren, die einen Entscheidungsbaum bzw. logistische Regression verwenden, sind mit einer Genauigkeit von 71,9% zufriedenstellend. Die eigentliche Problematik wird aber am Beispiel des Klassifikators, der randomisierte Entscheidungsbäume verwendet, offenbar. Der Klassifikator liefert solide Ergebnisse bei zufälliger Aufteilung des Datensatzes und liefert sogar das beste Ergebnis auf der Validierungsmenge bei Szenario-basierter Aufteilung (89,3% Genauigkeit). Auf der Testmenge fällt die Genauigkeit allerdings auf 37,4%. Eine mögliche Erklärung hierfür ist eine mögliche Überanpassung des gelernten Modells auf bestimmte Wörter der Szenarien. Bei zufälliger Aufteilung der Daten sieht der Klassifikator zum Trainingszeitpunkt ausreichend Beispiele aus allen Szenarien, um die Wörter für das jeweilige Szenario auswendig zu lernen. Wird der Datensatz hingegen Szenario-basiert aufgeteilt, sieht der Klassifikator nur Beispiele aus den ersten drei Szenarien; die Überanpassung findet nur in Bezug auf die Wörter dieser Szenarien statt. Infolgedessen kann ein gutes Ergebnis auf der Validierungsmenge erzielt werden, deren Instanzen ebenfalls aus den ersten drei Szenarien stammen. Bei Anwendung auf die zuvor ungesehenen Formulierungen des Testszenarios versagt der Klassifikator. Diese Ergebnisse zeigen deutlich, dass klassische Verfahren des maschinellen Lernens für die Aufgabe unzureichend sind. Sie vereinfachen das Klassifikationsproblem zu stark und sind nicht in der Lage, allgemeingültige Modelle zu erzeugen.

Seitens der Verfahren, die neuronale Netze verwenden, wurden folgende Arten von Netzen verwendet: (vorwärts gerichtete) künstliche neuronale Netze (engl. *artificial neural network*, kurz *ANN*), faltende neuronale Netze (engl. *convolutional neural network*, kurz *CNN*) und rekurrente neuronale Netze (engl. *recurrent neural network*, kurz *RNN*). Außerdem wurden unterschiedliche Architekturen rekurrenter Netze implementiert. Zum einen solche, die gesteuerte rekurrente Einheiten (engl. *gated recurrent unit*, kurz *GRU*) verwenden und zum anderen solche, die ein sogenanntes langes Kurzzeitgedächtnis (engl. *long short-term memory*, kurz *LSTM*) darstellen (siehe Abschnitt 2.2.2). Beide Architekturen wurden zudem jeweils als uni- und bidirektionale Variante implementiert. Außerdem wurden zusätzliche Schichten hinzugefügt, um tiefere Netze zu erzeugen und gegebenenfalls bessere Ergebnisse zu erzielen. Tabelle 7.26 gibt eine Übersicht über die Typen implementierter neuronaler Netze, den implementierten Architekturen sowie den

Tabelle 7.26: Implementierte Arten von neuronalen Netzen sowie verwendete Architekturen und Schichten.

Art	Architekturen	Zusätzliche Schichten
ANN		Flatten (Flat), Global max pooling 1D (G_{Max}), Dense (D), Dropout(DO)
CNN		Max pooling 1D (Max), Global max pooling 1D (G_{Max}), Dense (D), Dropout(DO)
RNN	LSTM GRU Bi-LSTM Bi-GRU	Dense (D), Dropout (DO)

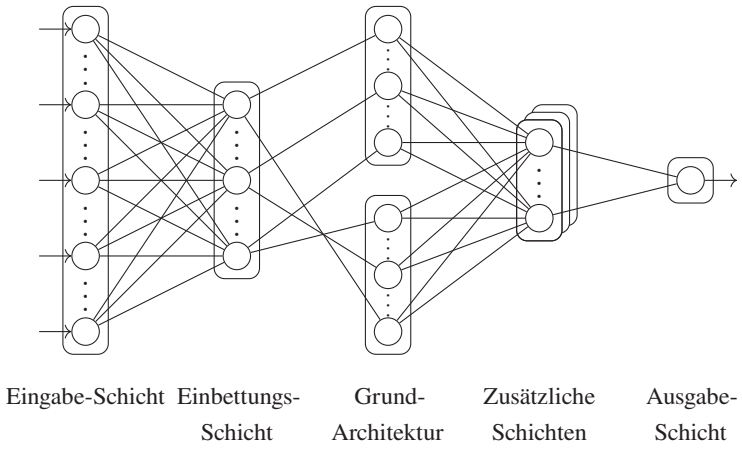


Abbildung 7.15: Schematische Illustration des Aufbaus der verwendeten neuronalen Netze.

jeweils verwendeten zusätzlichen Schichten. Abbildung 7.15 illustriert zudem den schematischen Aufbau der neuronalen Netze. Zunächst werden die einzelnen Wörter der Eingabe (kodiert als 1-aus-n-Vektor) eingelesen und anschließend in ihre jeweilige Repräsentation als Worteinbettung überführt. Die Worteinbettungsschicht enthält je nach Konfiguration entweder selbst trainierte Gewichte oder die von *Facebook Research* vortrainierten (siehe Abschnitt 7.7.2). Anschließend wird die Grund-Architektur durchlaufen, das heißt, entweder ein vorwärts gerichtetes (*ANN*), ein faltendes (*CNN*) oder ein rekurrentes Netz (*RNN*). Anschließend folgen – je nach Konfiguration – weitere Schichten, bevor in der Ausgabeschicht die eigentliche Klassifikation vorgenommen und das jeweilige Etikett emittiert wird. Für die einstellbaren Hyperparameter wurden alle Kombinationen üblicher Werte (der einzelnen Hyperparameter) getestet. Folgende Hyperparameter wurden variiert: Anzahl der Neuronen je Schicht, Anzahl der (Trainings-)Epochen, Stapel-Größe (engl. *batch size*),

Tabelle 7.27: Verwendete Werte zur Hyperparameter-Belegung der neuronalen Netze.

Hyperparameter	Getestete Werte
Anzahl Neuronen:	10; 20; 32; 40; 50; 64; 100; 128; 150; 250; 256; 512
Epochen (binär):	300; 500; 1000
Epochen (ternär):	50; 100; 300
Stapel-Größen (binär):	50; 100; 300; 400
Stapel-Größen (ternär):	32; 64; 100; 256; 300
Ausfall-Werte:	0,1; 0,2; 0,3
Lernraten:	0,001; 0,0005

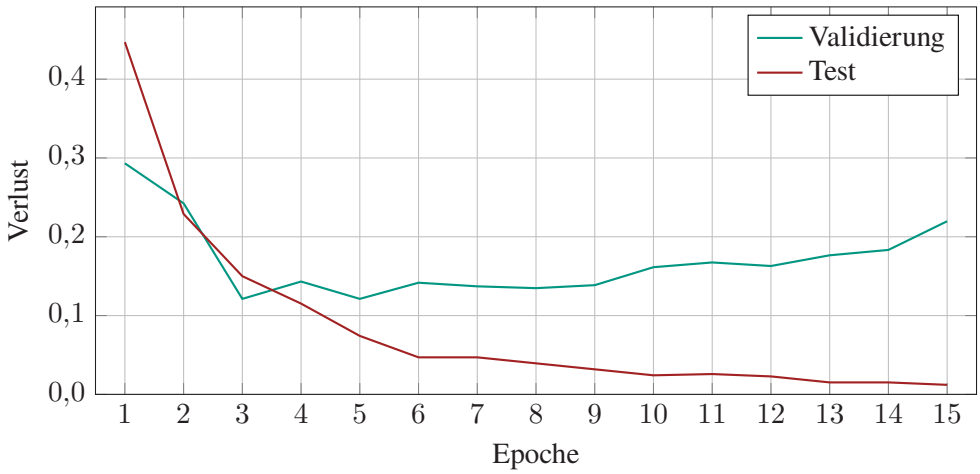


Abbildung 7.16: Beispielhafte Verlustfunktionen: Dargestellt sind die Verlustfunktionen für die Validierungs- und die Testmenge eines *Bi-LSTM* mit 128 Neuronen (Bi-LSTM(128)) in der rekurrenten Schicht und einer zusätzlich Verdichtungs-Schicht mit 64 Neuronen (D(64)).

Ausfall-Wert (engl. *dropout value*) und Lernrate[44]. Die getesteten Werte der Hyperparameter sind in Tabelle 7.27 gelistet. In der Tabelle nicht aufgeführt ist ein weiterer Hyperparameter, der nur bei faltenden Netzen relevant ist, der sogenannte Faltungsfaktor (engl. *convolution factor*). Dieser gibt die Dimension der quadratischen Faltungsmatrix an; getestet wurden die Werte 3, 5 und 7.

Das Training der Netze wurde in der Variante mit *frühem Abbruch* (engl. *early stopping*) durchgeführt. Das bedeutet, das Training kann auch vor dem Erreichen der maximalen Anzahl von Epochen unterbrochen werden und zwar dann, wenn die Verlustfunktion der Validierungsmenge (engl. *validation loss*) nicht mehr sinkt. Abbildung 7.16 zeigt die Verlustfunktionen der Test- und Validierungsmenge für ein Netz mit einem langen Kurzzeitgedächtnis, das bidirektional arbeitet (*Bi-LSTM*), und einer zusätzlichen *Verdichtungs*-Schicht (engl. *drop out layer*, kurz *DO*) mit 64 Neuronen. In diesem Fall sinkt die Verlustfunktion der Validierungsmenge nach fünf Epochen nicht mehr; das bedeutet, das Training wird an diesem Punkt abgebrochen und die bis dahin ermittelten

[44] In Abschnitt 2.2.2 befindet sich eine Einführung in die unterschiedlichen Arten von Schichten neuronaler Netze sowie eine Erläuterung zu den Hyperparmetern.

Gewichte der Neuronen werden verwendet. Dieses Kriterium wurde während des Trainings von allen Konfigurationen erfüllt, sodass die in Tabelle 7.27 geführten maximalen Epochenzahlen nie erreicht wurden[45]. Wie auch bei den klassischen Verfahren werden die besten Ergebnisse auf dem lemmatisierten Datensatz unter Ausschluss der Stoppwörter erzielt. Auch bei der Stapel-Größe hat sich während des Trainings 100 als bester Wert herausgestellt.

In Tabelle 7.28 und in Tabelle 7.29 sind die erzielten Genauigkeiten der neuronalen Netze mit ebendiesen Randbedingungen aufgeführt; die erste zeigt die Ergebnisse für die zufällige Aufteilung des Datensatzes und die zweite für die Szenario-basierte Aufteilung. Für alle weiteren Hyperparameter wurden alle möglichen Konfigurationen getestet. In den Tabellen sind jedoch nur die Konfigurationen je Netz-Art bzw. -Architektur aufgeführt, die auf der Validierungsmenge die besten Genauigkeiten erzielten[46] Die Konfigurationen können folgendermaßen gelesen werden. Für die vorletzte gezeigte Konfiguration, $\text{RNN}_{1.5.2}$, wurde beispielsweise als Architektur ein langes Kurzzeitgedächtnis verwendet, das zusätzlich bidirektional arbeitet (*Bi-LSTM*). Die rekurrente Schicht verfügt über 128 Neuronen. Darüber hinaus verfügt das Netz über folgende zusätzliche Schichten: eine *Verdichtungs*-Schicht mit 100 Neuronen, eine *Ausfall*-Schicht mit einem *Ausfall*-Wert von 0,3 und eine weitere *Verdichtungs*-Schicht mit 50 Neuronen. In den Tabellen werden die Ergebnisse nach Validierungs- und Trainingsmenge sowie nach der Art der verwendeten *fastText*-Worteinbettungen unterschieden: selbst trainiert oder vortrainiert. Als Vergleichsgrundlage wird das beste klassische Verfahren (logistische Regression) angegeben.

Auf dem zufällig aufgeteilten Datensatz (siehe Tabelle 7.28) erreichen die meisten Netze sehr gute Genauigkeiten. Lediglich die Konfigurationen $\text{RNN}_{1.1.*}$ und $\text{RNN}_{1.3.*}$ fallen deutlich ab und erzielen weit schlechtere Ergebnisse als der Klassifikator, der auf logistischer Regression basiert. Offensichtlich sind uni-direktionale rekurrente Netze (sowohl *GRU* als auch *LSTM*) ungeeignet für die vorliegende Aufgabe. Ihre bidirektionalen Pendants erzielen hingegen sehr hohe Genauigkeiten. Die meisten Konfigurationen erzielen unter Verwendung vortrainierter Worteinbettungen die besten Ergebnisse[47]. Dieses Resultat ist insofern positiv, als eine Anpassung der Worteinbettungen auf die Domäne keine Vorteile bringt; im Umkehrschluss wird so auch keine Überanpassung auf die Domäne erzeugt. Vergleicht man die Ergebnisse der abgebildeten Konfigurationen mit dem besten klassischen Verfahren, zeigt sich, dass lediglich faltende und bidirektionale, rekurrente Netze bessere Ergebnisse erzielen können (sofern die Hyperparameter richtig gewählt werden). Andererseits liegt die Genauigkeit des faltenden neuronalen Netzes $\text{CNN}_{1.2.1}$ auf der Testmenge mit 97,2% 2,5 Prozentpunkte über dem Ergebnis des Vergleichsverfahrens. Zusammenfassend lassen die Ergebnisse auf dem zufällige aufgeteilten Datensatz den Schluss zu, dass neuronale Netze sehr gute Ergebnisse für die vorliegende Aufgabe erzielen können, jedoch sorgfältig konfiguriert werden müssen.

[45] Das bedeutet, die Anzahl der trainierten Epochen variiert je Konfiguration. Allerdings konvergiert die Verlustfunktion in den meisten Fällen, bevor die zehnte Epoche erreicht wird.

[46] Da die Genauigkeit auf der Validierungsmenge für gewöhnlich als Schätzer für die Performanz des Netzes verwendet wird, ist dies ein übliches Vorgehen.

[47] Das insgesamt beste Ergebnis (auf dem Testdatensatz) wird mit 97,2% von der Konfiguration $\text{CNN}_{1.2.1}$ zwar unter Verwendung der selbst trainierten Worteinbettungen erzielt; die beste Konfiguration, die vortrainierte Einbettungen verwendet, $\text{CNN}_{1.2.0}$, liegt jedoch lediglich 0,1 Prozentpunkte dahinter.

Tabelle 7.28: Erzielte Genauigkeiten verschiedener Netzkonfigurationen auf dem *zufällig aufgeteilten Datensatz* für die *erste Klassifikationsebene*: Dargestellt sind jeweils ein Kurzname, die Netzkonfiguration und die erzielten Genauigkeiten auf den Validierungs- und Testmengen – jeweils unter Verwendung von selbst trainierten und vortrainierten Worteinbettungen.

		Selbst Train.		Vortrainiert	
Name	Konfiguration	Val.	Test	Val.	Test
$ANN_{1.1.0}$	Flat, D(10)	0,907	0,911	0,874	0,887
$ANN_{1.1.1}$	Flat, D(100)	0,916	0,914	0,846	0,867
$ANN_{1.2.0}$	G_{Max}, D(10)	0,876	0,887	0,872	0,902
$ANN_{1.2.1}$	G_{Max}, D(100)	0,899	0,896	0,879	0,896
$CNN_{1.1.0}$	C(128; 3), G_{Max}, D(10)	0,947	0,966	0,954	0,963
$CNN_{1.1.1}$	C(128; 5), G_{Max}, D(10)	0,947	0,971	0,930	0,965
$CNN_{1.1.2}$	C(128; 7), G_{Max}, D(10)	0,952	0,966	0,943	0,962
$CNN_{1.2.0}$	C(128; 3), Max(2), C(64; 3), G_{Max}, D(10)	0,952	0,959	0,952	0,971
$CNN_{1.2.1}$	C(128; 5), Max(2), C(64; 5), G_{Max}, D(10)	0,949	0,972	0,952	0,966
$CNN_{1.2.2}$	C(128; 5), Max(2), C(128; 5), G_{Max}, D(10)	0,952	0,964	0,954	0,966
$CNN_{1.2.3}$	C(128; 5), Max(5), C(128; 5), G_{Max}, D(10)	0,956	0,958	0,952	0,959
$RNN_{1.1.0}$	GRU(128)	0,560	0,625	0,562	0,625
$RNN_{1.1.1}$	GRU(128), D(100)	0,562	0,625	0,562	0,625
$RNN_{1.2.0}$	Bi-GRU(32), DO(0,2), D(64), DO(0,2)	0,947	0,944	0,952	0,959
$RNN_{1.3.0}$	LSTM(64)	0,566	0,631	0,568	0,638
$RNN_{1.3.1}$	LSTM(128)	0,570	0,625	0,654	0,738
$RNN_{1.3.2}$	LSTM(128), D(100)	0,562	0,625	0,562	0,625
$RNN_{1.4.0}$	Bi-LSTM(64), DO(0,2), D(64), DO(0,2)	0,947	0,955	0,949	0,955
$RNN_{1.4.1}$	Bi-LSTM(64), DO(0,3), D(200), D(100)	0,941	0,947	0,947	0,949
$RNN_{1.5.0}$	Bi-LSTM(128), D(64)	0,951	0,955	0,956	0,959
$RNN_{1.5.1}$	Bi-LSTM(128), D(64), D(32)	0,945	0,962	0,947	0,955
$RNN_{1.5.2}$	Bi-LSTM(128), D(100), DO(0,3), D(50)	0,936	0,937	0,945	0,941
$RNN_{1.6.0}$	Bi-LSTM(256), D(128)	0,952	0,944	0,945	0,952
Basis	Vergleichsgrundlage (*Logistische Regression*)			0,927	0,947

Die Betrachtung der Ergebnisse auf dem nach Szenarien aufgeteilten Datensatz (siehe Tabelle 7.29) sind für die Bewertung der Konfigurationen eher relevant, da dieser Aufbau, wie zuvor diskutiert, den Einsatz der Modelle realistischer nachbildet. Zunächst offenbaren die Ergebnisse, dass die Genauigkeiten aller Konfigurationen sinken. Dieses Resultat war zu erwarten, da sich das verwendete Vokabular in den Szenarien unterscheidet. Verfahren, die sich stärker an einzelnen Begriffen orientieren, sind von diesem Umstand besonders betroffen. Netze mit einfachem Aufbau neigen dazu lediglich bestimmte Formulierungen auswendig zu lernen; sie sind somit auf die Trainingsinstanzen überangepasst. Dementsprechend sinken die Genauigkeiten der einfachen, vorwärts gerichteten neuronalen Netze (*ANN*) besonders deutlich (bis zu minus 36 Prozentpunkte auf 54,2% für $ANN_{1.2.0}$

Tabelle 7.29: Erzielte Genauigkeiten verschiedener Netzkonfigurationen auf dem *Szenario-basiert aufgeteilten Datensatz* für die *erste Klassifikationsebene*: Dargestellt sind jeweils ein Kurzname, die Netzkonfiguration und die erzielten Genauigkeiten auf den Validierungs- und Testmengen – jeweils unter Verwendung von selbst trainierten und vortrainierten Worteinbettungen.

		Selbst Train.		Vortrainiert	
Name	Konfiguration	Val.	Test	Val.	Test
$ANN_{1.1.0}$	Flat, D(10)	0,918	0,759	0,897	0,722
$ANN_{1.1.1}$	Flat, D(100)	0,905	0,781	0,874	0,715
$ANN_{1.2.0}$	G_{Max}, D(10)	0,907	0,766	0,905	0,542
$ANN_{1.2.1}$	G_{Max}, D(100)	0,893	0,668	0,918	0,674
$CNN_{1.1.0}$	C(128; 3), G_{Max}, D(10)	0,962	0,765	0,966	0,854
$CNN_{1.1.1}$	C(128; 5), G_{Max}, D(10)	0,973	0,743	0,973	0,776
$CNN_{1.1.2}$	C(128; 7), G_{Max}, D(10)	0,973	0,775	0,970	0,897
$CNN_{1.2.0}$	C(128; 3), Max(2), C(64; 3), G_{Max}, D(10)	0,968	0,855	0,962	0,874
$CNN_{1.2.1}$	C(128; 5), Max(2), C(64; 5), G_{Max}, D(10)	0,969	0,850	0,975	0,859
$CNN_{1.2.2}$	C(128; 5), Max(2), C(128; 5), G_{Max}, D(10)	0,973	0,862	0,977	0,862
$CNN_{1.2.3}$	C(128; 5), Max(5), C(128; 5), G_{Max}, D(10)	0,962	0,901	0,973	0,801
$RNN_{1.1.0}$	GRU(128)	0,477	0,299	0,519	0,702
$RNN_{1.1.1}$	GRU(128), D(100)	0,519	0,702	0,519	0,702
$RNN_{1.2.0}$	Bi-GRU(32), DO(0,2), D(64), DO(0,2)	0,954	0,911	0,958	0,932
$RNN_{1.3.0}$	LSTM(64)	0,519	0,702	0,519	0,702
$RNN_{1.3.1}$	LSTM(128)	0,519	0,702	0,519	0,702
$RNN_{1.3.2}$	LSTM(128), D(100)	0,519	0,702	0,519	0,702
$RNN_{1.4.0}$	Bi-LSTM(64), DO(0,2), D(64), DO(0,2)	0,956	0,896	0,962	0,916
$RNN_{1.4.1}$	Bi-LSTM(64), DO(0,3), D(200), D(100)	0,947	0,884	0,956	0,911
$RNN_{1.5.0}$	Bi-LSTM(128), D(64)	0,960	0,927	0,962	0,919
$RNN_{1.5.1}$	Bi-LSTM(128), D(64), D(32)	0,950	0,919	0,966	0,898
$RNN_{1.5.2}$	Bi-LSTM(128), D(100), DO(0,3), D(50)	0,937	0,922	0,954	0,917
$RNN_{1.6.0}$	Bi-LSTM(256), D(128)	0,954	0,843	0,962	0,912
Basis	Vergleichsgrundlage (*Logistische Regression*)			0,891	0,719

bei Betrachtung der Genauigkeit auf der Testmenge mit vortrainierten Worteinbettungen). Das neuronale Netz, welches auf dem zufällig aufgeteilten Datensatz das beste Ergebnis erzielte, $CNN_{1.2.1}$, erfährt ebenso eine starke Verschlechterung. Trotz sehr guter Ergebnisse auf der Validierungsmenge, sinkt die Genauigkeit auf der Testmenge (je nach verwendeten Worteinbettungen) auf 85% bzw. 85,9%. Einzig die bidirektionalen rekurrenten Netze erzielen weiterhin gute Ergebnisse; die Konfiguration $RNN_{1.2.0}$ erreicht eine Genauigkeit von 93,2%. Damit liegt das Ergebnis einerseits vier Prozentpunkte unter dem besten Ergebnis auf dem zufällige aufgeteilten Datensatz ($CNN_{1.2.1}$: 97,2%), andererseits aber auch deutlich über dem Ergebnis des besten klassischen Verfahrens (Logistische Regression: 71,9%). Wie zu erwarten, wird diese hohe Genauigkeit unter Verwendung

der vortrainierten Worteinbettungen erzielt, da die Testmenge zuvor ungesehenes Vokabular enthält, welches dementsprechend bei Verwendung der selbst trainierten Einbettungen auf den Null-Vektor abgebildet wird (siehe Abschnitt 2.3.16). Dieser Intuition widersprechen die Ergebnisse der Konfigurationen $\text{RNN}_{1.5.*}$; diese erzielen ihre besten Ergebnisse mit den selbst trainierten Einbettungen. Eine mögliche Erklärung für dieses Resultat könnte sein, dass es diesen Netzen tatsächlich gelingt, ihre Klassifikation vorrangig von Formulierungen abhängig zu machen, die tatsächlich auf eine Lehrsequenz hinweisen, wie *means you have to* oder ähnliche. Überprüfen lässt sich diese Vermutung jedoch nicht, da die von neuronalen Netzen getroffenen Klassifikationsentscheidungen generell kaum interpretiert werden können. Zudem fällt auf, dass die Test-Genauigkeit nur noch schlecht anhand der auf der Validierungsmenge erzielten Genauigkeit geschätzt werden kann. Das Netz mit der besten Genauigkeit auf der Testmenge ($\text{RNN}_{1.2.0}$) erzielt auf der Validerungsmenge nur ein mittelmäßiges Ergebnis, mit knapp zwei Prozentpunkten Abstand zum besten Netz. Andersherum ist das Testergebnis des Netzes mit der besten Validierungs-Genauigkeit ($\text{CNN}_{1.2.2}$) schlechter als das der meisten anderen; die Genauigkeit liegt dabei sogar sieben Prozentpunkte hinter dem besten Ergebnis.

Als weitere Variante eines Verfahrens, welches neuronale Netze verwendet, wurde das vortrainierte Sprachmodell *BERT* (kurz für *Bidirectional Encoder Representations from Transformers*) verwendet [Dev+19]. *BERT* ist ein tiefes neuronales Netz, das strukturelle Zusammenhänge in natürlicher Sprache erlernt hat (siehe Abschnitt 2.3.17). Dieses vortrainierte Wissen kann genutzt werden, um spezielle Klassifikationsaufgaben zu lösen, hier die Unterscheidung von *lehrenden* und *nicht-lehrenden* Äußerungen. Bei der Verwendung des *BERT*-Sprachmodells ist es nicht nötig, Worteinbettungen zu verwenden; stattdessen wird eine eigene Eingabekonvertierung verwendet, die direkt im Modell geschieht. Es genügt also, eine einfache Eingabeschicht zu verwenden. Als Ausgabeschicht wurde eine flache binäre Schicht (engl. *flat layer*) gewählt. Für ProNat wurde das Modell $\textit{BERT}_{\textit{BASE}}$ genutzt (siehe Abschnitt 2.3.17). Der schematische Aufbau des auf *BERT* basierenden Klassifikators ist in Abbildung 7.17 dargestellt. Hinsichtlich der Trainingsepochen (neben der Modellgröße der einzige zu wählende Hyperparameter), wurden die Werte 5, 10 und 300 getestet. Auch für *BERT* wurde die Trainingsvariante mit *frühem Abbruch* verwendet; allerdings wurde das Abbruchkriterium (wie zuvor beschrieben) von keiner Konfiguration vor dem Erreichen der maximalen Epochenzahl erfüllt. Tabelle 7.30 zeigt die Ergebnisse der drei *BERT*-Konfigurationen. Wieder wird nach Aufteilung des Datensatzes und Validierungs- und Testmenge unterschieden. Als Vergleichsgrundlagen dienen neben dem *ZeroR*-Klassifikator und dem besten klassischen Verfahren (Logistische Regression) die beiden jeweils besten neuronalen Netze unter Verwendung der vortrainierten Worteinbettungen auf dem zufällig aufgeteilten und auf dem Szenario-basiert aufgeteilten Datensatz, $\text{CNN}_{1.2.0}$ und $\text{RNN}_{1.2.0}$.

Wie der Tabelle entnommen werden kann, übertreffen die *BERT*-Konfigurationen sowohl das beste klassische Verfahren als auch die beiden besten neuronalen Netze. Wenig überraschend erzielt das über 300 Epochen am längsten trainierte Modell BERT_3 die besten Genauigkeiten. Es erreicht eine Genauigkeit von 98,2% auf dem zufällig aufgeteilten Datensatz; dies entspricht einer Verbesserung um 1,1 Prozentpunkte gegenüber dem besten Vergleichsmodell ($\text{CNN}_{1.2.0}$: 97.1%). Auf dem Szenario-basiert aufgeteilten Datensatz beträgt die Verbesserung sogar 4,5 Prozentpunkte (gegenüber $\text{RNN}_{1.2.0}$: 93,2%). Die hier erzielte Genauigkeit von 97,7% stellt ein herausragendes

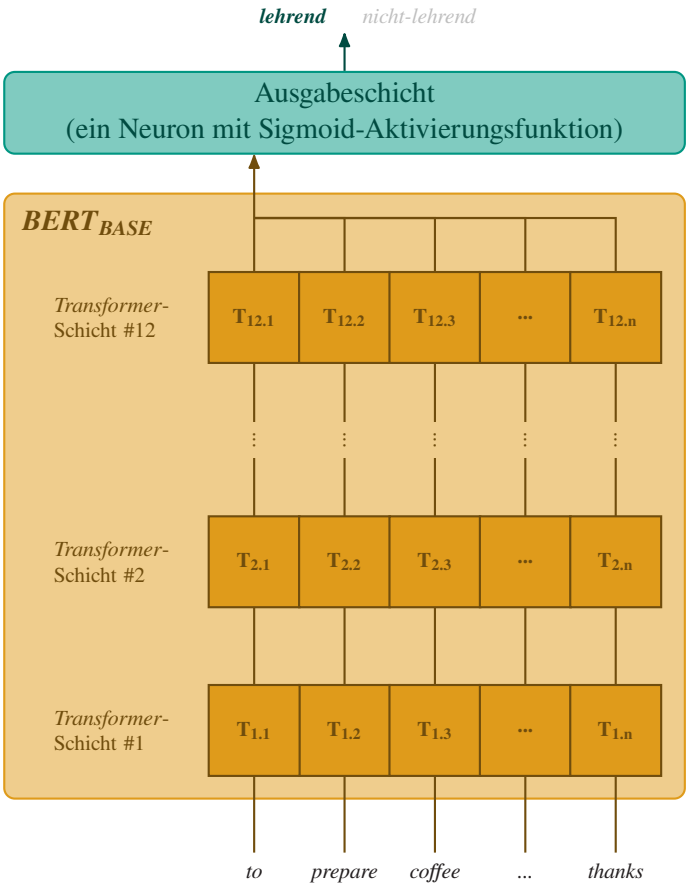


Abbildung 7.17: Schematische Darstellung der auf *BERT* basierenden Netzkonfigurationen für die *erste Klassifikationsebene*: Das Netz setzt sich aus dem vortrainierten Sprachmodell $BERT_{BASE}$, das aus 12 *Transformer*-Schichten besteht, und einer flachen Schicht zur Erzeugung der Ausgabe (binäre Klassifikation) zusammen.

Ergebnis dar. Offensichtlich gelingt es dem *BERT*-Sprachmodell unabhängig von den konkreten Formulierungen in den Szenarien, die generelle Struktur von Lehrsequenzen zu erlernen. Neben dem Modell $BERT_3$ stellt auch das Modell $BERT_2$ eine gute Wahl dar; die Qualität der Klassifikation ist ähnlich gut und der Trainingsaufwand um ein vielfaches geringer. Da das Training allerdings im besten Fall nur ein einziges Mal durchgeführt werden muss, fällt dieses Argument gegebenenfalls nicht ins Gewicht.

Zusammenfassend lässt sich festhalten, dass für die erste Klassifikationsebene, in der Lehrsequenzen von einfachen Befehlssequenzen unterschieden werden sollen (binäre Klassifikation, Klassen *Lehrend* und *Nicht-Lehrend*), klassische Verfahren des maschinellen Lernens unbrauchbar sind. Neuronale Netze erreichen hingegen sehr gute Ergebnisse; insbesondere rekurrente Netze erzielen auch bei einer Szenario-basierten Aufteilung des Datensatzes sehr hohe Genauigkeiten. Die beste Alternative stellt jedoch die Verwendung des *BERT*-Sprachmodells dar. Unabhängig von der Aufteilung des Datensatzes erreicht es die besten Ergebnisse. Das über 300 Epochen trainierte Modell $BERT_3$ erzielt auf der Szenario-basierten Aufteilung eine hervorragende Genauigkeit von 97,7%. Aufgrund des hohen Speicherbedarfs des Modells, der insbesondere aufgrund der Nebenläufigkeit der Agenten ein Problem darstellt, wird für den $\texttt{Pro}\mathscr{N}at$-Agenten als Standardkonfiguration das beste Modell eines neuronalen Netzes gewählt: $RNN_{1.2.0}$.

Tabelle 7.30: Erzielte Genauigkeiten der auf *BERT* basierenden Netzkonfigurationen für die *erste Klassifikationsebene*: Unterschieden wird nach *zufälliger* und *Szenario-basierter* Aufteilung des Datensatzes sowie nach Validierungs- und Testmenge.

	Zufällig		Szenario	
Name (und Konfiguration)	Val.	Test	Val.	Test
$BERT_{1.1}$ (*5 Epochen*)	0,973	0,981	0,991	0,969
$BERT_{1.2}$ (*10 Epochen*)	0,976	0,982	0,992	0,973
$BERT_{1.3}$ (*300 Epochen*)	0,962	0,982	0,992	0,977
Logistische Regression	0,927	0,947	0,891	0,719
$CNN_{1.2.0}$ (Vortrainierte Worteinbettungen)	0,952	0,971	0,962	0,874
$RNN_{1.2.0}$ (Vortrainierte Worteinbettungen)	0,952	0,959	0,958	0,932
Vergleichsgrundlage (*ZeroR*)	——	0,573	——	0,547

7.7.4 Analyse der semantischen Struktur von Lehrsequenzen

Auf der zweiten Klassifikationsebene sollen die semantischen Strukturen von Lehrsequenzen bestimmt werden. Daher werden nur Instanzen betrachtet, die im ersten Klassifikationsschritt mit dem Etikett *Lehrend* versehen wurden. Jedes Wort einer Lehrsequenz soll nun einer der drei zuvor definierten Klassen *Deklaration*, *Spezifikation* oder *Sonstiges* zugeordnet werden. Es handelt sich somit um eine typische Sequenz-zu-Sequenz-Aufgabe (engl. *sequence-to-sequence task*). Für diese Art von Aufgabe sind neuronale Netze besonders geeignet. Aus diesem Grund werden für die zweite Klassifikationsebene nur neuronale Netze verwendet[48]. Es wurden dieselben Netztypen, Architekturen und zusätzlichen Schichten verwendet wie für die erste Klassifikationsebene (siehe Tabelle 7.26). Auch die getesteten Hyperparameter-Belegungen unterscheiden sich nur bei der Epochen-Anzahl und den Stapel-Größen (vergleiche Tabelle 7.27). Für letztere wurden die Werte 32, 64, 100, 256 und 300 verwendet. Die maximale Anzahl an Epochen wurde entweder mit 50, 100 oder 300 konfiguriert. Wieder wurde beim Training die Option *früher Abbruch* gewählt, um gegebenenfalls nicht die volle Anzahl an Epochen trainieren zu müssen. Wie auch schon bei der Erkennung von Lehrsequenzen konvergierten alle getesteten Konfigurationen, bevor die maximale Epochenzahl erreicht wurde. Ebenso wurden erneut auf einer Datensatz-Variante von allen Konfigurationen die besten Ergebnisse erzielt, dieses Mal die *nicht* lemmatisierte Variante unter Ausschluss von Stoppwörtern. Dasselbe gilt für die Stapel-Größe; hier stellte sich 32 als beste Wahl heraus. Alle anderen Hyperparameter wurden in allen Kombinationen getestet.

In Tabelle 7.31 sind die Ergebnisse der besten Konfigurationen auf dem zufällig aufgeteilten Datensatz aufgeführt, in Tabelle 7.32 entsprechend die Ergebnisse auf der Szenario-basierten Aufteilung. Beide Tabellen unterscheiden zudem wieder zwischen erreichten Genauigkeiten auf der Validierungs- und auf der Testmenge sowie zwischen der Verwendung von selbst trainierten und vortrainierten Worteinbettungen. Als Vergleichsgrundlage dient der *ZeroR*-Klassifikator. Der *ZeroR*-Klassifikator erreicht vergleichsweise hohe Genauigkeiten, 75,9% auf dem zufällig aufgeteilten und

[48] Vortests auf Teilen des Datensatzes haben zudem die Unbrauchbarkeit der klassischen Lernverfahren belegt.

Tabelle 7.31: Erzielte Genauigkeiten verschiedener Netzkonfigurationen auf dem *zufällig aufgeteilten Datensatz* für die *zweite Klassifikationsebene*: Dargestellt sind jeweils ein Kurzname, die Netzkonfiguration und die erzielten Genauigkeiten auf den Validierungs- und Testmengen – jeweils unter Verwendung von selbst trainierten und vortrainierten Worteinbettungen.

		Selbst Train.		Vortrainiert	
Name	Konfiguration	Val.	Test	Val.	Test
$ANN_{2.1.0}$	-	0,851	0,855	0,851	0,856
$ANN_{2.2.0}$	D(10)	0,848	0,857	0,852	0,849
$ANN_{2.2.1}$	D(100)	0,853	0,856	0,853	0,848
$RNN_{2.1.0}$	LSTM(64)	0,977	0,976	0,979	0,978
$RNN_{2.1.1}$	LSTM(128)	0,974	0,976	0,978	0,977
$RNN_{2.2.0}$	LSTM(128), DO(0,2)	0,976	0,977	0,977	0,977
$RNN_{2.2.1}$	LSTM(128), DO(0,4)	0,976	0,977	0,979	0,979
$RNN_{2.2.2}$	LSTM(128), D(64)	0,973	0,972	0,977	0,976
$RNN_{2.3.1}$	Bi-LSTM(128)	0,986	0,983	0,987	0,985
$RNN_{2.3.2}$	Bi-LSTM(128), D(64)	0,980	0,983	0,985	0,984
$RNN_{2.3.3}$	Bi-LSTM(128), D(100), DO(0,3), D(50)	0,982	0,982	0,982	0,985
$RNN_{2.3.4}$	Bi-LSTM(128), DO(0,2)	0,985	0,984	0,988	0,988
$RNN_{2.3.5}$	Bi-LSTM(128), DO(0,4)	0,985	0,986	0,986	0,986
$RNN_{2.4.0}$	Bi-LSTM(256), DO(0,2)	0,986	0,984	0,987	0,985
$RNN_{2.5.0}$	Bi-GRU(128)	0,984	0,984	0,985	0,985
Basis	Vergleichsgrundlage (*ZeroR*)				0,759

75,7% auf dem Szenario-basiert aufgeteilten Datensatz. Diese Werte werden jedoch von allen (hier aufgeführten) Konfigurationen übertroffen. Selbst das nur aus zwei vorwärts gerichteten Schichten bestehende $ANN_{2.2.1}$ erreicht Genauigkeiten auf den Testmengen zwischen 82,2% und 85,6%. Lediglich faltende Netze sind ungeeignet für die vorliegende Aufgabe; keine Konfiguration ist unter den besten fünfzehn der hier gezeigten. Besonders gut schneiden jedoch erneut bidirektionale, rekurrente, neuronale Netze ab; alle Konfigurationen erreichen Test-Genauigkeiten über 98% auf dem zufällig aufgeteilten und über 96% auf dem Szenario-basiert aufgeteilten Datensatz. Insbesondere die Konfigurationen $RNN_{2.3.4}$ und $RNN_{2.3.1}$ erzielen hervorragende Ergebnisse; während $RNN_{2.3.4}$ auf dem zufällig aufgeteilten Datensatz mit 98,8% Genauigkeit hervorsticht, erzielt $RNN_{2.3.1}$ auf der Szenario-basierten Aufteilung mit 97,6% das beste Ergebnis. Insgesamt sind die Unterschiede zwischen den rekurrenten neuronalen Netzen jedoch nur marginal. Zudem fällt positiv auf, dass die Validierungsgenauigkeit bei allen Konfigurationen und für beide Datensatzaufteilungen gut geeignet ist, um die Testgenauigkeit abzuschätzen. Beide Werte bewegen sich durchgängig in der gleichen Größenordnung und die Tendenzen zwischen den Konfigurationen sind übertragbar.

Auch für die zweite Klassifikationsebene wurden Konfigurationen erstellt, die *BERT* verwenden. Diese entsprechen weitestgehend den für die erste Ebene verwendeten Konfigurationen. Lediglich die Ausgabeschicht wird angepasst, da bei der zweiten Klassifikationsebene einzelne Wörter

Tabelle 7.32: Erzielte Genauigkeiten verschiedener Netzkonfigurationen auf dem *Szenario-basiert aufgeteilten Datensatz* für die *zweite Klassifikationsebene*: Dargestellt sind jeweils ein Kurzname, die Netzkonfiguration und die erzielten Genauigkeiten auf den Validierungs- und Testmengen – jeweils unter Verwendung von selbst trainierten und vortrainierten Worteinbettungen.

		Selbst Train.		Vortrainiert	
Name	Konfiguration	Val.	Test	Val.	Test
$ANN_{2.1.0}$	-	0,850	0,779	0,851	0,826
$ANN_{2.2.0}$	D(10)	0,850	0,825	0,851	0,826
$ANN_{2.2.1}$	D(100)	0,851	0,822	0,851	0,827
$RNN_{2.1.0}$	LSTM(64)	0,971	0,960	0,975	0,966
$RNN_{2.1.1}$	LSTM(128)	0,973	0,960	0,973	0,964
$RNN_{2.2.0}$	LSTM(128), DO(0,2)	0,970	0,960	0,973	0,966
$RNN_{2.2.1}$	LSTM(128), DO(0,4)	0,971	0,959	0,974	0,967
$RNN_{2.2.2}$	LSTM(128), D(64)	0,970	0,955	0,971	0,963
$RNN_{2.3.1}$	Bi-LSTM(128)	0,983	0,960	0,981	0,976
$RNN_{2.3.2}$	Bi-LSTM(128), D(64)	0,973	0,960	0,979	0,965
$RNN_{2.3.3}$	Bi-LSTM(128), D(100), DO(0,3), D(50)	0,978	0,955	0,981	0,968
$RNN_{2.3.4}$	Bi-LSTM(128), DO(0,2)	0,982	0,958	0,981	0,975
$RNN_{2.3.5}$	Bi-LSTM(128), DO(0,4)	0,980	0,961	0,980	0,973
$RNN_{2.4.0}$	Bi-LSTM(256), DO(0,2)	0,982	0,964	0,982	0,975
$RNN_{2.5.0}$	Bi-GRU(128)	0,976	0,955	0,982	0,968
Basis	Vergleichsgrundlage (*ZeroR*)				0,757

(anstatt ganzer Äußerungen) klassifiziert werden und die Klassifikationsaufgabe (nicht binär, sondern) ternär ist. Die Ergebnisse sind in Tabelle 7.33 aufgeführt. Erneut dienen neben dem *ZeroR*-Klassifikator die jeweils besten Konfigurationen für die zufällige und die Szenario-basierte Aufteilung als Vergleichsgrundlage, das bedeutet $RNN_{2.3.1}$ und $RNN_{2.3.4}$. Auch für die zweite Klassifikationsebene erzielen die *BERT*-basierten Konfigurationen sehr gute Testergebnisse, bis zu 98,5% Genauigkeit für die zufällige Aufteilung und bis zu 97,3% für die Szenario-basierte Aufteilung. Der Vergleich mit den besten neuronalen Netzen zeigt jedoch, dass keine Verbesserung erzielt werden kann. Stattdessen liegen die Ergebnisse der *BERT*-basierten Konfigurationen sogar leicht hinter dem jeweils besten neuronalen Netz (je 0,3 Prozentpunkte).

Zusammenfassend lässt sich für die Klassifikation von semantischen Strukturen in Lehrsequenzen festhalten, dass neuronale Netze diese Aufgabe überraschend gut bewältigen. Besonders rekurrente neuronale Netze erscheinen geeignet; sie erreichen durchgehen Genauigkeiten von über 96% bis nahezu 99%. Die Verwendung des vortrainierten Sprachmodells *BERT* führt bei dieser Klassifikationsaufgabe zu keinen Verbesserungen. Für den $\texttt{Pro}\mathcal{NL}$-Agenten wird $RNN_{2.3.1}$ ausgewählt, da diese Konfiguration auf dem Szenario-basiert aufgeteilten Datensatz die beste Test-Genauigkeit erzielt und auch auf dem zufällig aufgeteilten nur knapp hinter der besten Konfiguration liegt. $RNN_{2.3.1}$ hat zudem den Vorteil, dass es unter den besten Netzen das kleinste Modell erzeugt, damit Ressourcen spart und schneller klassifiziert als die übrigen.

Tabelle 7.33: Erzielte Genauigkeiten der auf *BERT* basierenden Netzkonfigurationen für die *zweite Klassifikationsebene*: Unterschieden wird nach *zufälliger* und *Szenario-basierter* Aufteilung des Datensatzes sowie nach Validierungs- und Testmenge.

	Zufällig		Szenario	
Name (und Konfiguration)	Val.	Test	Val.	Test
$BERT_{2.1}$ (*5 Epochen*)	0,979	0,982	0,979	0,965
$BERT_{2.2}$ (*10 Epochen*)	0,983	0,985	0,983	0,972
$BERT_{2.3}$ (*300 Epochen*)	0,981	0,983	0,985	0,973
$RNN_{2.3.1}$ (Vortrainierte Worteinbettungen)	0,987	0,985	0,981	0,976
$RNN_{2.3.4}$ (Vortrainierte Worteinbettungen)	0,988	0,988	0,981	0,975
Vergleichsgrundlage (*ZeroR*)	——	0,759	——	0,757

7.7.5 Optimierung der ersten Klassifikationsebene

Die Ergebnisse des besten neuronalen Netzes für die erste Klassifikationsebene ($RNN_{1.2.0}$) liegen auf dem Test-Datensatz 4,5 Prozentpunkte hinter der besten *BERT*-basierten Konfiguration. Da die *BERT*-Modelle jedoch sehr viel Speicher benötigen, wird in der Standardkonfiguration von ProNat trotzdem $RNN_{1.2.0}$ verwendet. Um Fehler, die durch die Verwendung des schlechteren Modells entstehen, zu verringern, wurden zusätzliche Optimierungen entwickelt. Diese nutzen zum einen den sogenannten Trennwert der Ausgabeschicht und zum anderen die Ergebnisse der zweiten Klassifikationsebene.

Die Optimierungen der Klassifikationsergebnisse von $RNN_{1.2.0}$ erfolgen zweistufig. Zunächst wird der Trennwert optimiert. Die Ausgabeschicht verwendet (wie bei binären Klassifikationsproblemen üblich) als Aktivierungsfunktion die Sigmoidfunktion (siehe Abschnitt 2.2.2). Diese bildet die Aktivierungen der vorherigen Schicht auf den Wertebereich [0; 1] ab. Für die binäre Klassifikation werden anschließend alle Ergebnisse im Bereich [0; 0,5] der Klasse *Nicht-Lehrend* und alle im Bereich (0,5; 1] der Klasse *Lehrend* zugeordnet. Das bedeutet, der Trennwert der beiden Klassen liegt bei 0,5. Auch wenn im Trainingsdatensatz die Klasse *Lehrend* häufiger vertreten ist (siehe Abschnitt 7.7.1), zeigt sich, dass der Klassifikator dazu neigt, die Klasse *Nicht-Lehrend* zu bevorzugen. Daher wird der Trennwert entsprechend verschoben. Hierzu wurden alle Trennwerte im Bereich [0; 0,5] mit einer Schrittweite von 0,05 getestet. In Abbildung 7.18 sind die Ergebnisse aufgetragen. Als Vergleich dient das Testresultat von $RNN_{1.2.0}$, welches ohne Anpassung erzielt wird. Die Genauigkeit verbessert sich kontinuierlich bei Verringerung des Trennwertes, bis bei 0,1 der beste Wert erzielt wird. Die Genauigkeit erreicht 94,1%; dies entspricht einer Verbesserung um 0,9 Prozentpunkten gegenüber dem Grundmodell. Im ersten Optimierungsschritt wird dementsprechend der Trennwert der Ausgabeschicht auf 0,1 (statt 0,5) gesetzt. Abbildung 7.18 legt nahe, dass auch im Bereich [0,1; 0) falsche Klassifikationsergebnisse korrigiert werden könnten. Daher wird dieser Bereich untersucht und zusätzlich als zweiter Indikator für eine Lehrintention (Klasse *Lehrend*) das Ergebnis der zweiten Klassifikationsebene verwendet. Hierzu wird die zweite Ebene auf alle Äußerungen angewandt (und nicht wie zuvor beschrieben nur auf Äußerungen, die auf der ersten Ebene der Klasse *Lehrend* zugeordnet wurden). Anschließend wird die Klassenzuordnung derjenigen Äußerungen hinsichtlich

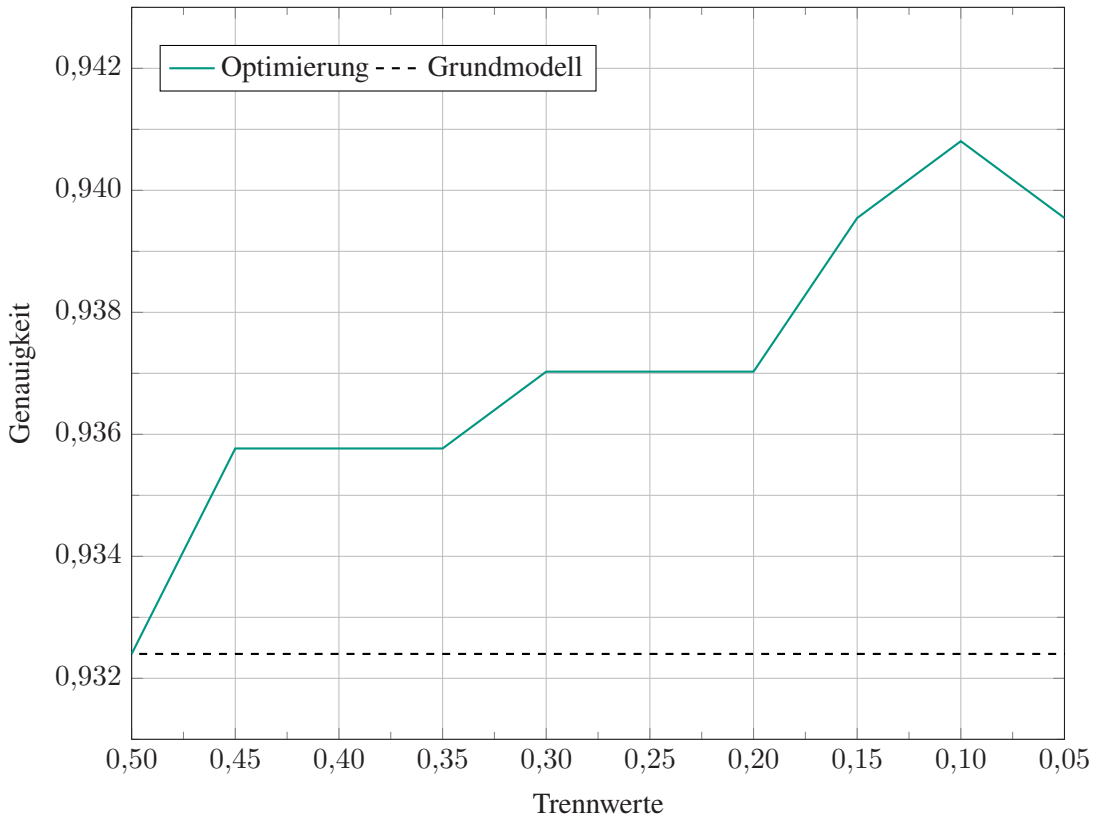


Abbildung 7.18: Optimierung der Trennwerte für die erste Klassifikationsebene: Dargestellt sind die Genauigkeiten, die unter Verwendung unterschiedlicher (optimierter) Trennwerte auf dem Testdatensatz erreicht werden.

der ersten Stufe abgeändert, deren numerische Repräsentation einen bestimmten Trennwert m unterschreitet und gleichzeitig eine gewisse Anzahl an Wörtern n aufweist, die auf der zweiten Stufe der Klasse *Deklaration* zugewiesen wurden. Die Grundidee hinter dieser Heuristik ist die Folgende: Wörter, die das Etikett *Deklaration* erhalten, beschreiben die Lehrintention; treten in einer Äußerung einige solche auf, liegt die Vermutung nahe, dass es sich um eine Lehrsequenz handelt (auch wenn die erste Klassifikationsebene keine solche erkannt hat).

Für den Trennwert m wurden Werte im Bereich [0,001; 0,1] mit einer Schrittweite von 0,001 getestet; für die Anzahl der *Deklaration*-Etiketten n wurde der Bereich [0; 6] untersucht. Abbildung 7.19 zeigt die Ergebnisse. Zum Vergleich wurde das Klassifikationsergebnis des Grundmodells aufgetragen[49]. Zwei Konfigurationen erreichen den besten Genauigkeitswert: der grüne Graph ($m = 0,008; n = 2$), und der lila Graph ($m = 0.006; n = 5$). Beide erreichen eine Genauigkeit von 95,5%; das entspricht einer Verbesserung von 2,2 Prozentpunkten gegenüber dem Grundmodell. Mit diesen Optimierungen liegt die Genauigkeit nur noch 1,8 Prozentpunkte hinter der besten *BERT*-Konfiguration. Im `ProNat`-Agenten kann die Optimierung optional verwendet werden. Als Standardwerte für m und n sind 0,008 bzw. 2 vorkonfiguriert; beide können jedoch frei gewählt werden.

[49] Die Abbildung zeigt die Ergebnisse im Bereich $m = [0,001; 0,03]$, da die besten Ergebnisse in diesem Bereich erzielt werden.

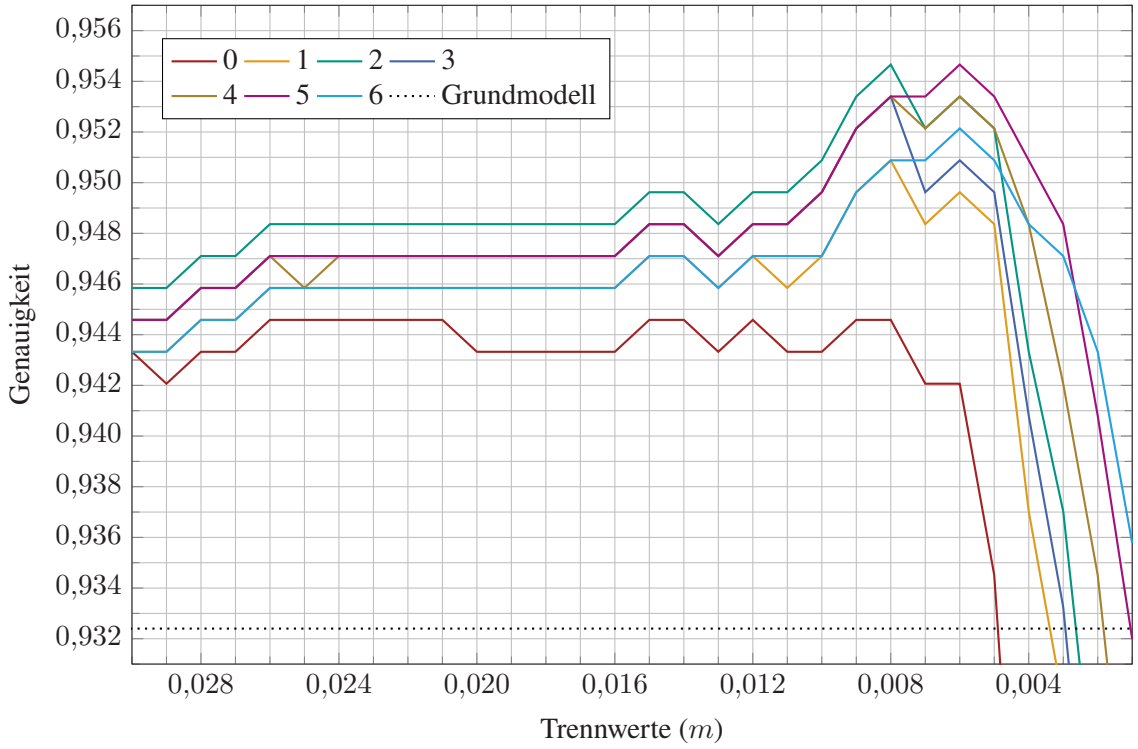


Abbildung 7.19: Kombinierte Optimierung für die erste Klassifikationsebene: Dargestellt sind die Genauigkeiten, die unter Verwendung unterschiedlicher Konfigurationen der kombinierten Optimierung der Klassifikation durch $\text{RNN}_{1.2.0}$ für die erste Klassifikationsebene erzielt werden. Eine Konfiguration besteht aus einem Trennwert m und der unteren Schranke beobachteter *Deklaration*-Etiketten n.

7.7.6 Implementierung des Agenten zur Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache

Der `Pro`$\mathscr{Nat}$-Agent zur Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache führt die zuvor beschriebene hierarchische Klassifikation durch. Hierzu werden die zuvor trainierten Modelle in den Agenten geladen. Welche Modelle genutzt werden sollen und ob die zweite Klassifikationsebene nur auf Lehrsequenzen oder alle Äußerungen angewendet werden soll, kann konfiguriert werden. Dasselbe gilt für die Verwendung von Wort-Lemmata, den Ausschluss von Stoppwörtern und die Verwendung der Optimierung des Klassifikationsergebnisses der ersten Klassifikationsebene. Der Agent ist wie folgt vorkonfiguriert: Für die erste Klassifikationsebene werden Wort-Lemmata verwendet und Stoppwörter ausgeschlossen; als Klassifikationsmodell dient $\text{RNN}_{1.2.0}$. Die Optimierung wird standardmäßig verwendet ($m = 0,008$ und $n = 2$). Für die zweite Klassifikationsebene werden keine Lemmata verwendet, Stoppwörter werden erneut ausgeschlossen. Zur Klassifikation werden nur Äußerungen zugelassen, die zuvor als *Lehrend* klassifiziert wurden und es wird das rekurrente Netz $\text{RNN}_{2.3.1}$ (BiLSTM(128)) verwendet.

Damit die Modelle die Klassifikation durchführen können, werden zunächst die natürlichsprachlichen Äußerungen aus dem `Pro`$\mathscr{Nat}$-Graphen extrahiert. Hierzu werden alle Knoten des Typs `Token` ausgelesen. Sollen Wort-Lemmata verwendet werden, liest der Agent diese zusätzlich aus den *Token* aus (Attribut *lemma*). Die Wort-Lemmata werden von der Vorverarbeitungsstufe zur seichten

Sprachverarbeitung erzeugt (siehe Abschnitt 6.2). Folglich besteht eine Abhängigkeit von den Ergebnissen dieser Vorverarbeitungsstufe. Sollen zudem Stoppwörter ausgeschlossen werden, geschieht dies mithilfe derselben Wortliste, die auch schon für das Training verwendet wurde. Anschließend wird die so vorbereitete Eingabe zunächst an die erste und dann an die zweite Klassifikationsebene übergeben. Je nach Konfiguration des Agenten wird die zweite Klassifikationsebene nur ausgeführt, wenn es sich gemäß der ersten Klassifikationsebene bei der Äußerung um eine Lehrsequenz handelt. Gegebenenfalls wird anschließend die Optimierung der Klassifikationsergebnisse der ersten Klassifikationsebene ausgeführt.

Zuletzt werden die Ergebnisse wie folgt im Graphen hinterlegt. Für das Ergebnis der ersten Klassifikationsebene wird ein neuer Knoten vom Typ `TeachingSequence` erzeugt; im booleschen Attribut *isTeachingSequence* wird hinterlegt, ob es sich bei der Äußerung um eine Lehrsequenz handelt oder nicht. Die Ergebnisse der zweiten Klassifikationsebene werden als neues Attribut *semanticPart* in den Knoten vom Typ `Token` gespeichert.

7.8 Synthese von Methodendefinitionen und Skripten

In nahezu allen Programmiersprachen ist es möglich, wiederverwendbare bzw. wiederholt auftretende Instruktionsfolgen zu definieren, beispielsweise in Form von Subroutinen, Methoden, Funktionen oder Prozeduren. Nur mithilfe dieser Konstrukte können Quelltexte effizient erstellt werden. Um Programmierung mit natürlicher Sprache zu ermöglichen, müssen Nutzer folglich in die Lage versetzt werden, Methoden mithilfe natürlichsprachlicher Beschreibungen zu definieren. Beschreibt ein Nutzer beispielsweise einem Heimautomationssystem, welche Aktionen für *einen Party-Modus* nötig sind oder einem Haushaltsroboter wie man *den Tisch deckt*, so möchte der Nutzer die nötigen Befehlsfolgen nicht jedes Mal wiederholen müssen. Stattdessen sollte das System diese erlernen und unter einem abrufbaren Namen abspeichern. Aus programmatischer Sicht entspricht der Name einer Methodensignatur und die zugehörige Befehlsfolge dem Methodenrumpf.

In der Forschung findet diese Aufgabe bisher wenig Beachtung. Die meisten intelligenten Systeme, die über eine Sprachschnittstelle verfügen, erlauben es nur einzelne Befehle bzw. einfache Befehlskombinationen auszuführen[50]. Einige Ansätze erlauben die Erzeugung von Skripten, wenige ermöglichen es, neue Funktionen zu definieren. Letztere erwarten aber entweder zusätzliche Eingaben (z. B. Zeige- bzw. Berührungsgesten) oder beschränken sich auf eine konkrete Zielanwendung (siehe Abschnitt 3.3). Das Ziel von ProNat ist es jedoch, Zielsystem- und Domänen-unabhängige Endnutzer-Programmierung ausschließlich mit natürlicher Sprache zu ermöglichen. Für ProNat wurde daher ein neuer Ansatz entwickelt [Wei+20b], der es erlaubt, aus natürlichsprachlichen Lehrsequenzen Methodendefinitionen zu synthetisieren (siehe Abschnitt 7.7). Da die Methodendefinitionen als Teil des ProNat-Graphen dargestellt werden, sind sie universell für unterschiedliche Programmiersprachen, Zielsysteme und Umgebungen einsetzbar.

[50] Dies gilt insbesondere für intelligente Assistenzsysteme (siehe Abschnitt 3.4).

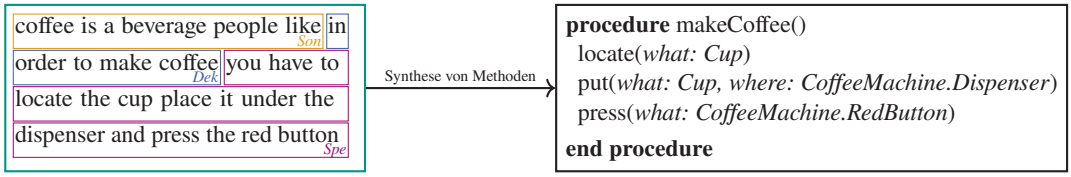


Abbildung 7.20: Schematische Darstellung des angestrebten Synthese-Prozesses: Zur Synthese von Methoden werden unter anderem die Etiketten verwendet, die von der zweiten Klassifikationsebene des Agenten zur Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache erzeugt werden.

7.8.1 Implementierung des Agenten zur Synthese von Methodendefinitionen und Skripten

Der Ansatz wurde als weiterer ProNat-Agent umgesetzt. der Agent stützt seine Analysen hauptsächlich auf die Ergebnisse des Agenten zur Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache (siehe Abschnitt 7.7). Dieser unterscheidet bereits Lehrsequenzen von Skripten. Zusätzlich wird die Struktur der Lehrsequenzen oberflächlich analysiert und in deklarative, spezifizierende und sonstige Abschnitte unterteilt. Der hier beschriebene Agent extrahiert diese Informationen aus dem ProNat-Graphen und erzeugt aus deklarativen Abschnitten Methodensignaturen und aus spezifizierenden Abschnitten die zugehörigen Methodenrümpfe. Handelt es sich bei einer Äußerung *nicht* um eine Lehrsequenz, wird aus den beschriebenen Schritten ein Skript erzeugt. Neben den Analyseergebnissen des Agenten zur Erkennung und Analyse von Lehrsequenzen verwendet der Agent zur Erzeugung der Methodendefinitionen Wortartmarkierungen, Chunk-Etiketten, Lemmata, Synonyme, semantische Rollen und Korreferenzinformationen. Dadurch hängt der Agent zusätzlich von den Vorverarbeitungsstufen zur seichten Sprachverarbeitung (Wortarten, Lemmata und Chunk-Etiketten) und zur Erkennung von semantischen Rollen sowie vom *Babelfy*-Agenten zur Disambiguierung von Wortbedeutungen (Grundlage für die Synonyme), dem Agenten zur Modellierung des sprachlichen Kontexts und dem Agenten zur Korreferenzanalyse von Nominalphrasen ab (siehe Abschnitte 6.2, 6.4, 7.2, 7.4 und 7.5).

Nachdem der Agent die benötigten Informationen aus dem ProNat-Graphen extrahiert hat, wird der Synthese-Prozess eingeleitet. Hierzu wird zunächst die Struktur der Äußerungen analysiert und in ein Modell überführt, in dem sowohl Methodendefinitionen als auch Skripte repräsentiert werden können. Anschließend werden zunächst die Methodensignaturen und dann die Methodenrümpfe bzw. Skripte synthetisiert. Während der Erzeugung der Rümpfe bzw. der Skripte erfolgt zudem eine Abbildung auf Funktionen des Zielsystems (API-Aufrufe), die in der Zielsystemontologie hinterlegt sind. Dadurch erzeugt der Agent bereits eine Art Pseudo-Code. In Abbildung 7.20 ist der angestrebte Synthese-Prozess anhand eines Beispiels schematisch dargestellt. Im Folgenden wird dieser genauer beschrieben.

Zunächst werden Äußerungen in ein semantisches Modell überführt; das Modell ist hierarchisch aufgebaut und repräsentiert die grundlegende Struktur von Methodendefinitionen. Die Grundbausteine sind jedoch keine Programmkonstrukte, wie Zuweisungen, Vergleiche oder Instruktionen, sondern Phrasen der natürlichsprachlichen Äußerung. Das Meta-Modell besagt, dass eine Methode immer aus einer Signatur und einem Rumpf besteht. Die Signatur wiederum besteht (in diesem

einfachen Modell) aus einem Namen und Parametern; der Rumpf der Methode besteht aus einer beliebigen Anzahl von Instruktionen oder Kontrollstrukturen. Kontrollstrukturen können wiederum Instruktionen enthalten. Eine Instruktion besteht, wie die Signatur, aus einem Namen und Parametern. Namen und Parameter werden im Modell durch Phrasen der Äußerung gebildet. Das bedeutet, Äußerungen müssen phrasenweise analysiert und auf Modellbestandteile abgebildet werden, um iterativ eine konkrete Modell-Instanz zu generieren. Prinzipiell werden Phrasen, die vom Agenten zur Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache als *deklarativ* erkannt wurden (siehe Abschnitt 7.7), auf den Signatur-Teil des Modells abgebildet. Phrasen, die dem *spezifizierenden* Teil der Äußerung zugeordnet wurden, werden hingegen auf Konstrukte des Rumpfes abgebildet.

Der Aufbau einer Modell-Instanz erfolgt aufwärts (engl. *bottom-up*); das bedeutet, das Modell wird ausgehend von den einzelnen Wörtern der natürlichsprachlichen Äußerung aufgebaut bis letztlich eine Methode (oder ein Skript) entsteht.

Zunächst wird die Äußerung gemäß der in Abschnitt 7.7 definierten Klassen in semantisch zusammenhängende Abschnitte unterteilt. Etwaige Phrasen, die der Klasse *Sonstige* zugeordnet wurden, werden nicht weiter betrachtet. Die weiteren Abschnitte (*Deklaration* und *Spezifikation*) werden anschließend syntaktisch und semantisch analysiert. Zunächst werden mithilfe semantischer Rollen einzelne Aktionen (semantische Rolle *V*) und zugehörige Argumente extrahiert (semantische Rollen *A**). Zusätzlich werden Korreferenzen aufgelöst[51].

Anschließend wird die Modell-Instanz schrittweise aufgebaut. Hierzu werden zunächst Aktionen auf Namen und die übrigen Rollen auf Parameter abgebildet[52]. Namen werden immer durch eine natürlichsprachliche Phrase gebildet, Parameter hingegen bestehen aus einer Liste von Phrasen. Zusätzlich zu den Phrasen werden jeweils Synonyme und Lemmata im Modell hinterlegt. Je nachdem, ob es sich bei der betrachteten Phrase um einen Teil der *Deklaration* oder der *Spezifikation* handelt, werden der Name und zugehörige Parameter wiederum zu einer Signatur (der neuen Methode) oder einer Instruktion (des Methodenrumpfes) zusammengefasst.

Der Rumpf kann aus beliebig vielen Instruktionen zusammengesetzt werden, die Signatur hingegen hat immer einen eindeutigen Namen und einen Satz Parameter[53]. Sobald mindestens eine Instruktion erzeugt werden konnte, wird der Methodenrumpf im Modell erzeugt. Wenn sowohl eine vollständige Signatur als auch ein Rumpf existieren, wird im Modell der Methoden-Knoten erzeugt und die Modell-Instanz ist vollständig. Handelt es sich bei der Äußerung nicht um eine Lehrsequenz, wird lediglich der Rumpf aufgebaut; die Modell-Instanz ist dann auch ohne Signatur und Methode vollständig.

[51] Das bedeutet, im Modell werden alle referenzierenden Ausdrücke durch die jeweiligen Referenten ersetzt.

[52] Während des Abbildungsprozesses werden Phrasen zusätzlich bereinigt. Unter anderem werden bestimmte Phrasenbestandteile, einige Rollen und Stoppwörter ausgeschlossen.

[53] Eine Betrachtung etwaiger Kontrollstrukturen innerhalb des Methodenrumpfes ist an dieser Stelle nicht nötig. Die von den entsprechenden Agenten erzeugten Graph-Strukturen sind ausreichend (siehe Abschnitt 7.6), um später Kontrollstrukturen in den synthetisierten Quelltext zu injizieren (siehe Abschnitt 8.1).

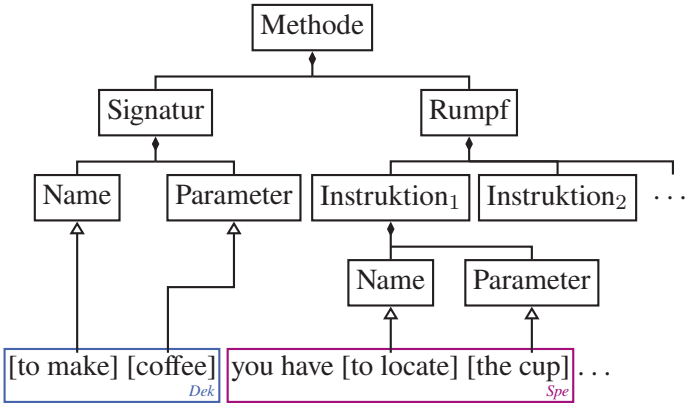


Abbildung 7.21: Ausschnitt einer Modell-Instanz des semantischen Meta-Modells für Methodendefinitionen in natürlicher Sprache: Die zugehörige Äußerung ist ein synthetisches Beispiel. Relevante Phrasen sind geklammert und die Etiketten *Dek* und *Spe* entsprechen den in Abschnitt 7.7 definierten Klassen zur Analyse der semantischen Struktur von Äußerungen.

Beispiel:
Erzeugung einer Modell-Instanz für natürlichsprachliche Methodendefinitionen

In diesem Beispiel soll die Struktur und Funktion des semantischen Meta-Modells für natürlichsprachliche Methodendefinitionen veranschaulicht werden. Das Beispiel nimmt Bezug auf die in Abbildung 7.21 dargestellte beispielhafte Modell-Instanz.

Natürlichsprachliche Sequenz

to make coffee you have to locate the cup [...]

Erläuterung

Der Ausschnitt der Beispieläußerung wurde bereits in einen deklarativen und einen spezifizierenden Abschnitt aufgeteilt (siehe Abschnitt 7.7). Davon ausgehend werden zunächst die semantischen Rollen ausgelesen, heuristisch bereinigt und in Chunks untergliedert (in der Abbildung durch eckige Klammern dargestellt). Aus den Phrasen *to make* und *to locate* werden anschließend Namensknoten für die Modell-Instanz erzeugt. Analog werden für *coffee* und *the cup* Parameterknoten angelegt. Der Namens- und Parameterknoten des deklarativen Abschnitts bilden im Modell zusammen die Signatur der Methode. Namen und Parameter werden im spezifizierenden Teil der Äußerung zu einer Instruktion zusammengesetzt; gegebenenfalls folgen weitere Instruktionen (im Beispiel nicht gezeigt). Alle extrahierten Instruktionen bilden den Methodenrumpf. Zuletzt wird aus der Signatur und dem Rumpf die gesamte Methode zusammengefügt.

Synthese von Methodensignaturen

Aus dem Modell wird anschließend zunächst die Methodensignatur synthetisiert. Der Methodenname wird erzeugt, indem alle Wörter, die im Modell-Knoten *Name* hinterlegt sind, in Binnenmajuskelschreibweise konkateniert werden[54] Aus den Phrasen des Parameter-Knotens werden entweder formale Parameter erzeugt oder sie werden dem zuvor erzeugten Namen angehängt (falls die Synthese nicht möglich ist). Um formale Parameter zu erzeugen, wird versucht, die jeweiligen Phrasen auf ein Individuum der Domänenontologie abzubilden[55]. Wird eine Abbildung gefunden, wird zusätzlich geprüft, ob das Konzept zu dem das Individuum gehört, ein gültiger Datentyp der API ist. Ist dies der Fall, wird ein formaler Parameter mit dem ermittelten Datentyp angelegt; der Name des Parameters wird aus dem Methodennamen und einem Suffix erzeugt. Das Suffix kann im zugehörigen Ontologie-Konzept hinterlegt werden. Wurde kein Suffix hinterlegt, wird das generische Suffix *.what<Ordnungsnummer des Parameter>* angehängt. Ist keine Abbildung möglich oder der Datentyp nicht bekannt, werden die Phrasenbestandteile dem Methodennamen hinzugefügt. Dieses Verfahren wird für alle Parameter des Modells durchgeführt.

Beispiel:
Synthese von Methodensignaturen

In diesem Beispiel soll veranschaulicht werden, wie Methodensignaturen synthetisiert werden. Das Beispiel geht auch auf die unterschiedlichen Varianten der Verwendung der Parameter des Modells ein.

Natürlichsprachliche Sequenz

serving wine means [...]

Erläuterung

Bei diesem Abschnitt einer Beispieläußerung handelt sich um eine deklarative Phrase. Der Agent erkennt in diesem Fall anhand der semantischen Rollen, sowohl *serving* als auch *means* als Verben. Zweiteres wird mithilfe einer Stoppwortliste ausgeschlossen. Aus dem Lemma von ersterem wird der erste Teil des Methodennamens gebildet: `serve`. Anschließend wird versucht, *wine* auf ein Ontologie-Individuum abzubilden. Angenommen der Agent findet das Individuum *RedWineBottle*, welches dem Konzept *Graspable* zugeordnet ist und *Graspable* ist ein zulässiger Datentyp, dann wird in diesem Fall folgende Signatur synthetisiert: `serve(serve.what : Graspable)`. Kann hingegen keine zulässige Abbildung gefunden werden, wird die Signatur `serveWine()` erzeugt.

[54] Wie zuvor beschrieben, wurden die Phrasen in den Modellknoten bereinigt, das bedeutet, sie enthalten beispielsweise keine Stoppwörter oder Hilfsverben.

[55] Die Abbildung von Phrasen auf Ontologie-Individuen wird im folgenden Paragraphen beschrieben.

Synthese von Methodenrümpfen und Skripten

Nachdem die Signatur der neuen Methode synthetisiert wurde, wird der Rumpf aufgebaut. Hierzu werden die Instruktionen im Modell auf API-Aufrufe abgebildet. In einem ersten Schritt werden Blätter des Modells, also alle Namen und Parameter, einzeln auf Ontologie-Individuen abgebildet. Um in der Ontologie nach passenden Individuen suchen zu können, werden die Phrasen vorverarbeitet und zu Suchwörtern konkateniert. Hierzu werden zunächst aus den Phrasen drei Wortmengen gebildet: die unveränderten Wörter der Phrase, die Lemmata aller Wörter und Synonyme. Aus den Mengen wird jeweils die Potenzmenge gebildet und anschließend jede mögliche Permutation erzeugt; zuletzt werden die Wörter konkateniert. Aus der Phrase *is closed* entstehen beispielsweise die folgenden Suchwörter: *isclosed*, *closedis*, *beclose*, *closebe*, *closed*, *is*, etc. Anschließend wird in der Ontologie nach Individuen gesucht, deren Bezeichner ähnlich zu den Suchwörtern sind. Als Ähnlichkeitsmetriken werden die Jaro-Winkler-Ähnlichkeit und die unscharfe Ähnlichkeitsbewertung (engl. *fuzzy score*), wie in Abschnitt 2.3.13 eingeführt, verwendet. Die Suchstrategien, wie in Abschnitt 5.4.2 beschrieben, sind so konfiguriert, dass alle Übereinstimmungen, die entweder eine Jaro-Winkler-Ähnlichkeit größer 0,4 oder eine unscharfe Ähnlichkeitsbewertung größer 0,15 erzielen, zurückgegeben werden[56]. Der Grund für die relativ niedrig angesetzten Schwellenwerte ist folgender: Diese erste Suche soll möglichst viele Kandidaten generieren.

Die so erzeugte Kandidatenmenge wird in der Folge reduziert; zuletzt werden die Kandidaten bewertet. Werden die Schwellenwerte höher angesetzt, besteht die Gefahr, dass Abbildungen, die sich in der Folge als richtig herausstellen, bereits an dieser Stelle ausgefiltert werden. Da für jede Phrase mehrere Suchwörter gebildet wurden, entstehen potenziell auch mehrere unterschiedlich bewertete Übereinstimmung zwischen einer Phrase und einem Ontologie-Individuum. In diesem Fall wird jeweils der höchste ermittelte Wert für beide Metriken (Jaro-Winkler und unscharfe Ähnlichkeitsbewertung) als Bewertung für die Abbildung übernommen.

Nachdem für die einzelnen Namen und Parameter Abbildungskandidaten erzeugt wurden, werden diese im zweiten Schritt zu vollständigen API-Aufrufen kombiniert und bewertet. Hierzu werden wiederum Namen und Parameter einer Instruktion (im Modell) gemeinsam betrachtet. Ausgehend von Abbildungskandidaten für Methodennamen werden alle möglichen Parameterbelegungen erzeugt. Hierzu wird für jeden Methodenkandidaten in der Ontologie geprüft, welche formalen Parameter die Signatur erfordert[57]. Der Agent befüllt die formalen Parameter anschließend mit allen gefundenen Parameterkandidaten. An dieser Stelle werden die Datentypen der Parameterkandidaten mit denen der formalen Parameter abgeglichen und nur die Parameter passenden Typs übernommen. Eine Methode inklusive Parameterbelegungen bildet einen API-Aufruf-Kandidaten. Kann für einen Methodenkandidaten keine gültige (oder nur eine teilweise gültige) Parameterbelegung ermittelt werden, wird dieser trotzdem in die Ergebnismenge übernommen. Die fehlenden Parameter werden gekennzeichnet.

[56] Beide Werte können über die Konfiguration des Agenten angepasst werden.

[57] Natürlich ist es möglich, dass die API formale Parameter primitiven Typs oder Standard-Typen (beispielsweise Zeichenketten oder Enumerationen) enthält. Diese werden mithilfe von Heuristiken erkannt und fließen ebenso in die Menge der Parameterkandidaten ein.

Beispiel:
Synthese von Kontrollstrukturen

In diesem Beispiel soll veranschaulicht werden, wie aus Methodennamen- und Parameterkandidaten Kandidaten für vollständige Aufrufe (Methoden inklusive Argumente) erzeugt werden. Diskutiert wird dabei auch die Überprüfung des Typs der Parameterkandidaten.

Natürlichsprachliche Sequenz

[*take*] [*the cup*] *from* [*the table*]

Abbildungskandidaten

take	→	`grasp(grasp.what Graspable)`
	→	`grasp(grasp.where Location, grasp.what Graspable)`
	→	`uncap(uncap.what Openable)`
the cup	→	`Mug`, vom Typ `Graspable`
	→	`Cupboard`, vom Typ `Location`
the table	→	`Kitchentable`, vom Typ `Location`

Erläuterung

Für den ersten Methodenkandidaten wird der Aufruf `grasp(Mug)` erzeugt, da die Methode nur einen formalen Parameter aufweist und nur `Mug` hinsichtlich des Typs passt. Der zweite Methodenkandidat erwartet zwei Parameter unterschiedlichen Typs. Es wird daher der Aufruf `grasp(Kitchentable, Mug)` erzeugt. Nicht erzeugt wird hingegen der Aufruf `grasp(Cupboard, Mug)`. Zwar stimmen die Typen der Parameter mit denen der formellen Parameter überein. Jedoch gehen beide Parameterkandidaten auf dieselbe Phrase in der natürlichsprachlichen Äußerung zurück (*the cup*). Eine solche Abbildung wäre semantisch nicht sinnvoll und wird daher ausgeschlossen. Für den letzten Kandidaten kann keine Parameterbelegung ermittelt werden: `uncap(?)`

Die API-Aufruf-Kandidaten werden in einem letzten Schritt bewertet. In diese Bewertung fließen die Bewertungen (Jaro-Winkler und unscharfe Ähnlichkeitsbewertung) der einzelnen Bestandteile (Methodennamen und Parameter) ein. Zusätzlich wird der Anteil der Wörter der natürlichsprachlichen Äußerung, die für die Abbildung verwendet wurden, und das Verhältnis der formalen zu den aktuellen Parametern einbezogen. Die Bewertung $S(c)$ eines Aufruf-Kandidaten c berechnet sich wie folgt:

$$S(c) = \phi * \Phi(c) * S_M(c) + (1 - \phi) * S_P(c) \tag{7.6}$$

Der erste Summand stellt die Bewertung des Methodenkandidaten dar, der zweite die der zugehörigen Parameterbelegung (Argumente). Das Verhältnis, mit dem beide Summanden in die Gesamtbewertung einfließen, kann über den Parameter ϕ angepasst werden. Die Bewertung des Methodenkandidaten wird zudem mithilfe eines Faktors für perfekte Abbildungen $\Phi(c)$ skaliert:

$$\Phi(c) = \begin{cases} \tau & M(c) > 0,9 \\ 1 & \text{andernfalls} \end{cases} \tag{7.7}$$

Durch die Verwendung von $\Phi(c)$, können Methodenkandidaten, deren Bezeichner besonders ähnlich zur natürlichsprachlichen Äußerung sind, bevorzugt werden. Hierzu wird $\Phi(c)$ für alle Methodenkandidaten mit einer Bewertung $S_M(c)$ größer 0,9 auf τ und für alle anderen auf 1 gesetzt. Der Skalierungsfaktor τ ist frei konfigurierbar; sinnhafte Werte liegen jedoch im Bereich (1; 1,5]. Die eigentliche Bewertung eines Methodenkandidaten $S_M(c)$ wird wie folgt ermittelt:

$$S_M(c) = M(c) - \frac{\beta}{|I_A(c)|} * \left(1 - \frac{|I_F(c)|}{|I_A(c)|}\right) \tag{7.8}$$

Die Methodennamenbewertung $M(c)$ ist lediglich das Maximum aus Jaro-Winkler und unscharfer Ähnlichkeitsbewertung, die zwischen der natürlichsprachlichen Phrase und dem Bezeichner der Methode bei der Abbildung ermittelt wurden. Diese reine Methodennamenbewertung kann durch den Subtrahenden reduziert werden. Der Subtrahend spiegelt wider, wie gut der Bezeichner die Wörter der ursprünglichen Phrase repräsentiert. Er ist zusammengesetzt aus zwei Faktoren. Der zweite ist eins minus der Anteil der Wörter der natürlichsprachlichen Phrase, die auf einen Bestandteil des Bezeichners abgebildet wurden ($I_F(c)$), im Verhältnis zu allen Wörtern der Phrase $I_A(c)$. Dieser Faktor stellt somit den Anteil der nicht abgebildeten Wörter der Phrase dar. Der andere Faktor skaliert diesen Wert um den Faktor β, welcher durch die Länge der Phrase geteilt wird. Der Grundgedanke hinter dieser Art von Skalierung im Verhältnis zur Phrasenlänge ist folgender. In kurzen Phrasen trägt jedes einzelne Wort verhältnismäßig mehr Informationen als in langen Phrasen. Das bedeutet, wurden in kurzen Phrasen einzelne Wörter nicht für die Abbildung verwendet, ist es eher wahrscheinlich, dass die Abbildung inkorrekt ist; es erfolgt eine stärkere Abwertung. Bei langen Phrasen hingegen ist es nahezu unmöglich alle Wörter in die Abbildung einfließen zu lassen. Gleichzeitig geht das Fehlen eines einzelnen Wortes im Verhältnis zur gesamten Phrase mit einem geringeren Informationsverlust einher. Mithilfe des zweiten Summanden der Gleichung 7.6 fließt die Bewertung der Parameterbelegung ($S_P(c)$) in die Gesamtbewertung des Aufruf-Kandidaten ($S(c)$) ein. Die Bewertung aller Parameter eines Kandidaten berechnet sich wie folgt:

$$S_P(c) = P(c) - \omega * \Gamma \tag{7.9}$$

Der Minuend $P(c)$ stellt die eigentliche Bewertung der einzelnen Parameter dar:

$$P(c) = \sum S_{p_i}(c) * \frac{|P_M|}{|P_O(c)|} \tag{7.10}$$

P_M ist die Menge aller Parameter p_i des Modells, die auf einen formalen Parameter des Aufruf-Kandidaten abgebildet werden konnten; $S_{p_i}(c)$ ist die Ähnlichkeitsbewertung eines (abgebildeten) Parameters p_i. Die Menge $P_O(c)$ umfasst alle formalen Parameter des betrachteten Aufruf-Kandidaten c (wie in der Domänenontologie hinterlegt). Anschaulich gesagt, berechnet $P(c)$ die Summe aller Ähnlichkeitsbewertung der abgebildeten Parameter multipliziert mit dem Verhältnis abgebildeter Parameter zu den (erforderlichen) formalen Parametern des Aufruf-Kandidaten. Der Subtrahend der Gleichung 7.9 ist ein gewichteter Strafterm (konfigurierbarer Faktor ω), der die Parameterbewertung verringert, falls nicht alle Parameter der Instruktion im Modell abgebildet werden konnten. Anschaulich erfolgt eine Abwertung, falls in der natürlichsprachlichen Äußerung mehr beschrieben als in der Parameterbelegung abgebildet wurde. Formell wird Γ wie folgt berechnet:

$$\Gamma = \frac{|P_E| - |P_M|}{|P_E|} \tag{7.11}$$

P_E ist dabei die Menge aller Parameter der Instruktion im Modell. Γ entspricht somit dem Verhältnis aller *nicht* abgebildeten Parameter des Modells ($|P_E| - |P_M|$) zur Anzahl der aus der natürlichsprachlichen Äußerung extrahierten Parameter ($|P_E|$). In der Standard-Konfiguration des Pro$\mathscr{N}_{at}$-Agenten sind folgende Werte für die konfigurierbaren Parameter voreingestellt:

- $\phi = 0,6$ (Verhältnis Methodennamenbewertung zu Parameterbelegungsbewertung)
- $\tau = 1,5$ (Skalierungsfaktor für perfekte Abbildungen des Methodennamens)
- $\beta = 0,5$ (Faktor des Strafterms für die Methodennamenbewertung)
- $\omega = 0,3$ (Faktor des Strafterms für die Parameterbelegungsbewertung)

Die Werte wurden empirisch anhand der Beschreibungen zum vierten Szenario der Online-Datensammlung ermittelt (siehe Abschnitt 5.5.3). Zuletzt hinterlegt der Agent seine Analyse-Ergebnisse im Pro$\mathscr{N}_{at}$-Graphen. Die Datenstruktur entspricht im Wesentlichen dem in Abbildung 7.21 dargestellten Modell. Je Modell-Element wird ein neuer Knotentyp angelegt. Für die Instruktionen werden zudem die synthetisierten API-Aufrufe gemäß ihres Ranges zusammen mit ihrer jeweiligen Konfidenz hinterlegt. Da für eine Instruktion potenziell eine große Anzahl an Aufruf-Kandidaten generiert wird, kann die maximale Anzahl der zu hinterlegenden API-Abbildungen konfiguriert werden (Standardwert 10).

7.8.2 Evaluation des Agenten zur Synthese von Methodendefinitionen und Skripten

Für die Evaluation des Pro$\mathscr{N}_{at}$-Agenten zur Synthese von Methodendefinitionen und Skripten wurden Beschreibungen zu den Szenarien eins bis drei der Online-Datensammlung verwendet (siehe Abschnitt 5.5.3)[58]. Da die Bewertung der Ergebnisse teilweise manuell erfolgt, konnte

[58] Die Beschreibungen zum vierten Szenario wurden nicht verwendet, da diese für die Entwicklung des Agenten verwendet wurden (siehe Abschnitt 7.8.1).

Tabelle 7.34: Datensatz für die Evaluation des Agenten zur Synthese von Methodendefinitionen und Skripten.

	Beschreibungen	lehrend	nicht-lehrend	API-Aufrufe
Szenario Online_1	25	18	7	77
Szenario Online_2	25	19	6	97
Szenario Online_3	25	15	10	123
Gesamt	75	52	23	297

nicht automatisch auf den vollständigen Datensätzen evaluiert werden. Stattdessen wurden aus jedem Datensatz zufällig 25 Äußerungen gezogen. Tabelle 7.34 gibt einen Überblick über den verwendeten Datensatz. Das Verhältnis von lehrenden zu nicht-lehrenden Äußerungen ist 69% zu 31% und ist damit ähnlich zum Verhältnis in den vollständigen Datensätzen (63% zu 37%, siehe Abschnitt 7.7.1). Der Datensatz enthält 52 Lehrsequenzen, das bedeutet, der Agent sollte 52 Methodensignaturen erzeugen. Darüber hinaus enthalten die spezifizierenden Abschnitte der Beschreibungen 297 Instruktionen. Dementsprechend müssen für die Methoden-Rümpfe bzw. Skripte ebenso viele API-Aufrufe synthetisiert werden. Um die synthetisierten API-Aufrufe zu evaluieren, wird eine Domänenontologie benötigt. Passend zu den Szenarien, wurde hierzu die API-Ontologie *Haushaltsroboter* (*ARMAR-III*) mit der Umgebungsontologie *Küche* verschmolzen (siehe Abschnitt 5.4.3). Die so erzeugte Ontologie umfasst

- 92 Methoden,
- 59 Parameter,
- 20 Datentypen (inklusive primitiver und Standard-Datentypen),
- 70 Umgebungsobjekte aus
- sechs Klassen (u.a. `Graspable` und `Location`), die
- sechs unterschiedliche Zustände einnehmen können (u.a. `opened` und `closed`).

Für die API-Aufrufe wurde ein Gold-Standard erstellt; zur Beschreibung passende Aufrufe können der Domänenontologie entnommen werden. Für die Methodensignaturen wurde hingegen auf einen Gold-Standard verzichtet, da zum einen für Methodennamen viele unterschiedliche, aber gleichwertige Varianten möglich sind und zum anderen manche Beschreibungen formelle Parameter erfordern, andere jedoch nicht. Stattdessen soll nachträglich entschieden werden, ob die synthetisierte Methodensignatur hinsichtlich der natürlichsprachlichen Äußerung sinnhaft ist.

Der Synthese-Prozess des Agenten fußt auf den Analyse-Ergebnissen anderer Agenten und Vorverarbeitungsstufen (siehe Abschnitt 7.8.1). Für die Etiketten des Agenten zur Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache, welche die Hauptgrundlage bilden, liegen händisch erstellte Lösungen vor, die verwendet werden können (siehe Abschnitt 7.7.2). Für die weiteren Abhängigkeiten liegen keine Musterlösungen vor. Dementsprechend wurden das vollständige Vorverarbeitungsfließband von ProNat sowie die entsprechenden Agenten ausgeführt,

Tabelle 7.35: Ergebnisse der Evaluation für die Abbildung natürlichsprachlicher Phrasen auf einzelne API-Elemente (Ontologie-Individuen): Phrasen werden entweder auf Methoden oder Parameter abgebildet. Die Ergebnisse in Klammern geben die Ergebnisse unter Ausschluss von fehlerhaften semantischen Rollen an.

	Einzelbestandteile (Methoden und Parameter)			
	Präzision	Ausbeute	F_1	ø Rang
Szenario Online_1	0,763	0,584 (0,776)	0,662 (0,769)	1,31
Szenario Online_2	0,783	0,742 (0,857)	0,762 (0,818)	1,16
Szenario Online_3	0,847	0,813 (0,893)	0,830 (0,870)	1,16
Gesamt	0,807	0,731 (0,854)	0,767 (0,830)	1,20

um die Analyse-Grundlage zu schaffen. Mögliche Fehlerquellen, die auf diese Abhängigkeiten zurückzuführen sind, können dementsprechend nur nachträglich analysiert werden. Zuletzt wurde der Agent ausgeführt, um Methoden bzw. Skripte für die 75 Äußerungen des Evaluationsdatensatzes zu erzeugen.

Der Agent synthetisiert für alle 52 Lehrsequenzen des Datensatzes eine Methodensignatur. Allerdings sind acht der erzeugten Methodennamen unpassend. Ein Name wird als unpassend gewertet, wenn er aus Termen zusammengesetzt ist, die keinen Bezug zur zu erlernenden Funktion haben, oder wenn zusätzliche (unnötige) Wörter angehängt wurden. Beispielsweise wurden für das Szenario *Kaffee zubereiten* (zweites Szenario der Online-Datensammlung) die Namen `askSpeaker`[59] (erster Fall) und `prepareCoffeeFriend`[60] (zweiter Fall) synthetisiert. Für die 52 Methodensignaturen wurden zudem 23 formalen Parametern synthetisiert, die alle als korrekt (hinsichtlich der zugehörigen Beschreibung) eingestuft werden (keine falsch positiven Ergebnisse). Ob darüber hinaus formale Parameter fehlen (falsch negative Ergebnisse), lässt sich aus dem Resultat nicht ableiten.

Die Evaluation der API-Aufrufe wurde zweistufig durchgeführt. Zunächst wurde die Abbildung auf einzelne API-Elemente (Ontologie-Individuen) betrachtet, also Methodennamen, Parameter, etc. Anschließend wurden die zusammengesetzten API-Aufrufe evaluiert. Die Tabelle 7.35 zeigt die Ergebnisse für die Betrachtung der Individuen, Tabelle 7.36 die Ergebnisse für die synthetisierten API-Aufrufe. Beide geben jeweils Präzision, Ausbeute und das F_1-Maß an. Der Agent erzeugt eine nach Konfidenzen sortierte Ergebnisliste; für die Berechnung der zuvor genannten Metriken wurde das (durch den Agenten) höchstbewertete Individuum (bzw. API-Aufruf) betrachtet. Zusätzlich wurde jedoch der durchschnittliche Rang bemessen, auf dem sich das korrekte Individuum (bzw. der korrekte API-Aufruf) befindet. Während der Evaluation konnten fehlerhafte (bzw. fehlende)

[59] Dieser Methodenname wurde aus der folgenden Äußerung generiert: „if I ask you to get me a drink [...]“. In diesem Fall hat der Agent zur Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache nur den ersten fünf Wörtern statt der vollständigen Phrase (wie es korrekt gewesen wäre) die Etikette *Deklaration* zugewiesen. Folglich werden auch nur diese fünf Wörter bei der Synthese des Methodennamens berücksichtigt. Die Bereinigung reduziert die Wortmenge auf die Wörter *I* und *ask*, wobei ersteres anschließend auf das Ontologie-Individuum `Speaker` abgebildet wird.

[60] Dieser Methodenname wurde aus der folgenden Äußerung generiert: „hi my friend preparing coffee means [...]“. In diesem Fall hat der Agent zur Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache das Wort *friend* fälschlich die Etikette *Deklaration* zugeordnet (siehe Abschnitt 7.7.4). Daher wird dieses Wort bei der Synthese des Methodennamens mitverwendet.

Tabelle 7.36: Ergebnisse der Evaluation für die Erzeugung zusammengesetzter API-Aufrufe: Die Ergebnisse in Klammern geben die Ergebnisse unter Ausschluss von fehlerhaften semantischen Rollen an.

	zusammengesetzte API-Aufrufe			
	Präzision	Ausbeute	F_1	ø Rang
Szenario Online_1	0,583	0,461 (0,614)	0,515 (0,598)	1,47
Szenario Online_2	0,674	0,620 (0,713)	0,646 (0,693)	1,19
Szenario Online_3	0,672	0,645 (0,708)	0,658 (0,690)	1,20
Gesamt	0,653	0,590 (0,689)	0,620 (0,670)	1,22

semantische Rollen als wesentliche Fehlerquelle identifiziert werden. Daher sind in den Tabellen in Klammern zusätzlich die Ergebnisse aufgeführt, die erzielt werden, wenn Phrasen einer Äußerung nicht betrachtet werden, die keine Annotation semantischer Rollen erhalten haben. Der mittlere Wert für das F_1-Maß für die Abbildung auf einzelne API-Elemente ist 0,767. Die Präzision übertrifft dabei die Ausbeute um 0,076. Werden Fehler bei der Erzeugung semantischer Rollen ausgeschlossen, erreicht der Wert für das F_1-Maß sogar 0,830. Diese Ergebnisse stellen auf den ersten Blick eine gute Grundlage für die Komposition der Aufrufe dar. Leider sinkt der Wert für das F_1-Maß bei Betrachtung dieser auf 0,620, wobei Ausbeute und Präzision ähnlich stark von der Verschlechterung betroffen sind.

Bei genauerer Betrachtung zeigt sich, dass die Ergebnisse für die API-Aufrufe unter den gegebenen Umständen nicht besser hätten ausfallen können. Die Aufrufe bestehen durchschnittlich aus drei Elementen (eine Methode mit zwei Argumenten). Stellt nur eines dieser Elemente eine fehlerhafte Abbildung dar oder fehlt, muss die Aufruf-Komposition als fehlerhaft gewertet werden. Die Abbildung der einzelnen API-Elemente sollte dementsprechend zukünftig noch verbessert werden. Hierzu empfiehlt sich zunächst eine Bestandsaufnahme der bestehenden Fehlerursachen. Neben fehlender semantischer Rollen sind unpassende und fehlende Synonyme eine wesentliche Fehlerquelle. Führt *WordNet* kein Synonym für ein wichtiges Wort einer Äußerung, kann der Agent gegebenenfalls keine Abbildung auf ein Individuum der Domänenontologie finden. Inferiert andererseits der *Babelfy*-Agent eine falsche Wortbedeutung, kann es vorkommen, dass für die Abbildung ein unpassendes Synonym verwendet wird. Folglich bildet der Agent gegebenenfalls auf ein unpassendes Ontologie-Individuum ab. Einige Fehler lassen sich wiederum auf den Synthese-Prozess des Agenten zurückführen. Das verwendete Modell natürlichsprachlicher Methodendefinitionen ist in manchen Fällen nicht in der Lage, die Semantik einer Äußerung sinngemäß abzubilden. Beispielsweise erzeugt der Agent für die Phrase *make sure you close it* zwei API-Aufrufe: `close(...)` und `make(...)`. Solche überflüssigen Abbildungen werden auch für erklärende Aussagen wie *the machine fills cups* generiert, sofern der entsprechende Abschnitt vom Agenten zur Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache *nicht* als irrelevant (Klasse *Sonstige*) gekennzeichnet wurde. Die bis hierher diskutierten Fehlerquellen treten bereits bei der Abbildung auf einzelne API-Elemente auf. Hinsichtlich der Komposition von API-Aufrufen zeigt sich, dass vor allem die Methodenargumente eine wesentliche Fehlerursache darstellen. Parameterbelegungen sind in vielen Fällen nicht eindeutig. Beispielsweise ist für die Phrase *open the door* unklar, um welche Tür

(engl. *door*) es sich handelt. Viele derartige Mehrdeutigkeiten, aber nicht alle, können mithilfe des Kontextmodells aufgelöst werden. Ein verwandtes Problem sind fehlerhaft aufgelöste Korreferenzen; jeder Fehler führt potenziell zu einem falschen Methodenargument. Auch wenn viele Fehler auf inkorrekte Analysen der abhängigen Module (die in Zukunft verbessert werden könnten) oder auf nicht beachtete Fälle im Agenten zurückgeführt werden können, haben ebenso viele ihren Ursprung in unpräzisen oder unvollständigen natürlichsprachlichen Beschreibungen. In vielen dieser Fälle könnte der Dialog-Agent in Zukunft Abhilfe schaffen (siehe Abschnitt 7.10). Eine weitere Möglichkeit wäre eine Prüfung der Plausibilität der Aufruffolgen. Für eine Umsetzung ist jedoch entscheidend, dass die korrekten API-Aufrufe eine hohe Konfidenz erhalten und dementsprechend zu Beginn der Kandidaten-Liste geführt werden. Das bedeutet, der durchschnittliche Rang des korrekten Elements ist von Interesse. Erfreulicherweise zeigen die Evaluationsergebnisse, dass der durchschnittliche Rang häufig nahe eins liegt; für die Individuen ist er im Mittel kleiner gleich 1,2 und für API-Aufrufe nur geringfügig schlechter (1,22). Mithilfe des Bewertungsmechanismus für die API-Aufruf-Kandidaten gelingt es also, die richtigen Elemente am Anfang der Liste zu platzieren.

7.9 Auswahl von Zielsystem- und Umgebungsontologien

Eines der Ziele beim Entwurf von ProNat ist die Domänenunabhängigkeit (siehe Abschnitt 5.1). Die starke Kapselung der Komponenten in PARSE ermöglicht die Umsetzung dieses Zieles. Lediglich die Verwendung der Ontologien zur Modellierung des Zielsystems und der Umgebung, mit denen ProNat konfiguriert werden muss, erzeugen eine (wenn auch nur geringfügige) Abhängigkeit von der Domäne. Besser wäre es die Ontologien zur Laufzeit auszuwählen. ProNat würde in seiner Verwendung noch flexibler; so könnten beispielsweise gleichzeitig mehrere Systeme (über unterschiedliche Zielsystemontologien) in unterschiedlichen Umgebungen (über verschiedene oder mehrere Umgebungsontologien) angesprochen werden, ohne dass die Konfiguration angepasst werden müsste. Gleichzeitig blieben die Vorteile kleiner Ontologien gegenüber einer gemeinsamen Ontologie erhalten (siehe Abschnitt 5.4). Mit einer derartigen Erweiterung müssten die entsprechenden Ontologien nur noch bereitgestellt werden, die Auswahl geschieht dann automatisch. Eine Möglichkeit zur Umsetzung dieser Idee stellt die Verwendung von Diskurs-Themen-Etiketten dar (siehe Abschnitt 7.3). Die erkannten Diskurs-Themen in natürlichsprachlichen Äußerungen können auf Ontologie-Themenbereiche abgebildet werden. Die besten Abbildungen können dann als Auswahl der Ontologien interpretiert werden.

Die Herausforderung, Ontologien für eine bestimmte Aufgabe auszuwählen, ist Gegenstand unterschiedlicher Forschungsbereiche, wie ontologiebasierte Fragebeantwortung (engl. *ontology-based question answering*) oder semantikangereichertes Web-Browsing (engl. *semantically enriched web browsing*) [LSM06; Sab+06; SLM06]. Üblicherweise verwenden Ansätze zur Auswahl von Ontologien (engl. *ontology selection*) lexikalische Informationen [POA11; KK12]. Kein bekannter Ansatz verwendet jedoch explizit Diskurs-Themen. Da ProNat zwischen Umgebungs- und Zielsystemontologien unterscheidet (siehe Abschnitt 5.4.1), PARSE aber nur die Verwendung einer Ontologie zulässt (siehe Abschnitt 4.2), müssen die ausgewählten Ontologien zudem verschmolzen

werden. Die für ProNat verwendeten Ontologien sind jedoch so entworfen, dass sie einfach verschmolzen werden können. Über die Konzepte `Objekt` und `State` werden De-Facto-Schnittstellen bereitgestellt, sodass die entsprechenden Konzepte der Umgebungsontologien an diesen Stellen in die Zielsystemontologien eingefügt werden können. Die Anwendung eines Ansatzes aus dem Forschungsgebiet der Ontologieverschmelzung (engl. *ontology merging*) ist somit nicht nötig. Ansätze aus diesem Forschungsgebiet betrachten die Verschmelzung von heterogenen Ontologien [HQ08; CAS09; SA09]. Das bedeutet, es gibt – anders als für ProNat vorgesehen – im Allgemeinen keine (völlig) übereinstimmenden Konzepte oder Individuen, sodass Ähnlichkeiten bestimmt werden müssen, um Verschmelzungen durchführen zu können.

Der Fokus des hier vorgestellten Ansatzes kann dementsprechend gänzlich auf die Auswahl der Ontologien gelegt werden. Bevor Ontologien automatisch ausgewählt werden können, müssen alle Individuen und Konzepte der Ontologien soweit möglich mit Bedeutungs-Etiketten (das heißt *Wikipedia*-Artikel-Bezeichnern) versehen werden[61]. Durch dieses Vorgehen werden sie (ebenso wie die Substantive der natürlichsprachlichen Äußerung) disambiguiert. Dieser Schritt muss derzeit manuell (aber nur einmal bei der Erstellung der Ontologie) durchgeführt werden. Das in Abschnitt 7.2 vorgestellte Verfahren ist aber prinzipiell auch auf Ontologien übertragbar, sodass zukünftig Themen-Etiketten auch für Ontologien automatisch generiert werden könnten.

7.9.1 Implementierung des Agenten zur automatischen Auswahl von Zielsystem- und Umgebungsontologien

Der für ProNat entwickelte Agent zur automatischen Auswahl von Zielsystem- und Umgebungsontologien [Wei+20a] nutzt die in der natürlichsprachlichen Äußerung erkannten Diskurs-Themen zur Auswahl passender Ontologien. Das bedeutet der Agent ist abhängig von den Ergebnissen des Agenten zur Modellierung und Etikettierung von Diskurs-Themen (siehe Abschnitt 7.3). Sobald die erkannten Themen der natürlichsprachlichen Äußerung in Form eines neuen Knotens vom Typ `topics` zur Verfügung stehen, liest der Agent die Themen-Etiketten samt Konfidenzen aus. Anschließend wird das in Abschnitt 7.3 vorgestellte Verfahren zur Generierung von Themen-Etiketten auf alle zur Verfügung stehenden Ontologien angewandt. Da das Verfahren ausschließlich auf den disambiguierten Bedeutungen beruht, ist es ohne Anpassung auch auf Ontologien anwendbar [62].

Anschließend werden die extrahierten Themen der natürlichsprachlichen Äußerung mit den Themen der Ontologien verglichen. Das Ziel des Verfahrens ist es, möglichst viele Zusammenhänge in den behandelten Themen zu erfassen. Je mehr Zusammenhänge ermittelt werden können, desto besser passt eine Ontologie zu einer Äußerung. Um die Zusammenhänge zu ermitteln, werden erneut Themengraphen (ähnlich zu denen in Abschnitt 7.3 vorgestellten) aufgebaut. Daraufhin wird der Abstand zwischen den Themen der Äußerung und den Themen einer jeden Ontologie verwendet, um

[61] Für einige Bedeutungen existiert kein *Wikipedia*-Artikel (siehe Abschnitt 7.2). Individuen und Konzepte, für die keine Bedeutungs-Etiketten gefunden werden können, werden ausgelassen. Die Vollständigkeit der Etiketten ist keine Voraussetzung für die Anwendbarkeit der hier vorgestellten Lösung.

[62] Im Gegensatz zur Standard-Konfiguration wird zur Disambiguierung von Ontologien die Strategie *Höchste Konnektivität* verwendet, da Ontologien – wie in ProNat verwendet – abgeschlossene Themen-Bereiche umfassen.

die Ähnlichkeit zu ermitteln. Formal bedeutet dies, dass die Ähnlichkeit $\Psi_T(t, T_o)$ eines Themas der natürlichsprachlichen Äußerung t zu einer Menge von Themen einer Ontologie T_o wie folgt berechnet wird (die Menge aller generierten Themen der natürlichsprachlichen Äußerung wird als T_u bezeichnet):

$$\Psi_T(t, T_o) = \frac{1}{min_{t_o \in T_o}(dist(t, t_o)) + 1},\ t \in T_u \tag{7.12}$$

Der Abstand $dist(t, t_o)$ ist definiert als der kürzeste Pfad zwischen einem Thema der Äußerung t und einem Thema der Ontologie t_o. Der minimale Abstand ist 0, wenn t einem der Themen der Ontologie entspricht ($t \in T_u \wedge T_o$). In diesem Fall ist die Ähnlichkeit maximal, Ψ_T gleich 1. Kann kein Pfad zwischen t und allen Elementen von T_o bestimmt werden, wird Ψ_T gleich 0 gesetzt. Das bedeutet, der Wertebereich von Ψ_T ist 0 bis 1. Die Ähnlichkeiten je Thema werden zu einem gemeinsamen Ähnlichkeitswert zusammengefasst. Die gemeinsame Ähnlichkeit Ψ_{Acc} gibt an, wie ähnliche alle Themen einer Äußerungen T_u den Themen einer Ontologie T_o sind:

$$\Psi_{Acc}(T_u, T_o) = \frac{\sum_{t \in T_u} \Psi_T(t, T_o)}{|T_u|} \tag{7.13}$$

Auf diese Weise werden die Ähnlichkeiten der natürlichsprachlichen Äußerung zu allen Ontologien bestimmt. Anschließend können die passendsten Ontologien ausgewählt werden. Für die Auswahl gibt es unterschiedliche Möglichkeiten. Eine Möglichkeit besteht darin, die n besten Ontologien auszuwählen. Dieses Vorgehen birgt allerdings die Gefahr, zu viele oder zu wenige Ontologien auszuwählen. Gegebenenfalls könnten Ontologien ausgewählt werden, die nur einen geringen Ähnlichkeitswert Ψ_{Acc} aufweisen (um bis zu n aufzufüllen), oder aber eigentlich passende Ontologien mit einem hohen Ähnlichkeitswert nicht ausgewählt werden (da bereits n Ontologien gewählt wurden). Daher ist eine zweite Möglichkeit, die Ontologien anhand eines Schwellenwertes für Ψ_{Acc} auszuwählen. Ein fest gewählter Schwellenwert wiederum birgt andere Risiken. Ist der Schwellenwert zu hoch gewählt, besteht die Möglichkeit, dass keine Ontologie gewählt wird. Ist andererseits der Schwellenwert zu niedrig, könnten viele unpassende Ontologien gewählt werden. Letzteres ist zwar nicht direkt schädlich, widerspricht aber der Entwurfsmaxime von ProMat, nur kleine, präzise Ontologien zu verwenden (siehe Abschnitt 5.4).

Daher wurde zur Auswahl ein flexibleres Modell entwickelt, das im Wesentlichen eine Kombination der beiden zuvor genannten Möglichkeiten darstellt. Um Ontologien o aus der Menge aller Ontologien O auszuwählen, wird ein flexibler Schwellenwert berechnet, der sich an dem Ψ_{Acc}-Wert der ähnlichsten Ontologie ausrichtet:

$$\Theta(T_u, O) = \{j \in O | \Psi_{Acc}(T_u, T_j) > \theta * \arg\max_{i \in O} \Psi_{Acc}(T_u, T_i)\},\ \theta \in \mathbb{R} \mid \theta = [0, 1] \tag{7.14}$$

Der Schwellenwert wird bestimmt, indem der höchste Ψ_{Acc}-Wert mit dem Auswahl-Faktor θ multipliziert (und somit verringert) wird.

Beispiel:
Berechnung des flexiblen Schwellenwertes

In diesem Beispiel soll die Berechnung des flexiblen Schwellenwertes zur Auswahl passender Ontologien zu einer natürlichsprachlichen Äußerung demonstriert werden.

Annahmen

Sei

- der beste Ähnlichkeitswert zwischen der Äußerung und einer Ontologie $\arg\max_{i \in O} \Psi_{Acc}(T_u, T_i) = 0,7$ und
- der Auswahl-Faktor $\theta = 0,9$.

Ergebnis

In diesem Fall wird der hintere Teil (flexibler Schwellenwert) der Gleichung 7.14 zu $0,63$ ausgewertet. Das bedeutet, es werden alle Ontologien ausgewählt, die der folgenden Gleichung genügen: $\Theta(T_u, O) = \{j \in O | \Psi_{Acc}(T_u, T_j)\} > 0,63$.

Der bis hierher vorgestellte Ansatz zur Auswahl von Ontologien bevorzugt Umgebungsontologien über Zielsystemontologien. Die natürlichsprachlichen Äußerungen (zumindest die des ProNat-Korpus) enthalten selten Begriffe, die sich auf das verwendete System beziehen. Zwar werden häufig Funktionen des Systems genannt, allerdings zumeist nur in ihrer Verbform. Der Agent zur Modellierung und Etikettierung von Diskurs-Themen betrachtet nur Substantive (siehe Abschnitt 7.3); daher ist das Zielsystem hinsichtlich der generierten Themen-Etiketten unterrepräsentiert. Infolgedessen können auch keine Zielsystemontologien bestimmt werden, die ähnlich zur natürlichsprachlichen Äußerung sind. Aus diesem Grund wird die Auswahl der Zielsystemontologien angepasst.

Hierzu werden zunächst nach dem zuvor beschriebenen Verfahren die Umgebungsontologien ausgewählt. Anschließend wird bestimmt, wie gut ein System jeweils zu den ausgewählten Umgebungen passt. Diese Kompatibilität wird als Ähnlichkeit interpretiert. Formal wird die Ähnlichkeit Ψ_{Sys} einer Zielsystemontologie o_s zu den Themen einer natürlichsprachlichen Aussage T_u anhand der Menge der zuvor ausgewählten Umgebungsontologien O_{sel} wie folgt berechnet:

$$\Psi_{Sys}(o_s, T_u, O_{sel}) = \frac{|DT_{o_s} \cap \bigcup_{i \in O_{sel}} Objects_i|}{3 * |DT_{o_s}|} + \frac{2 * \Psi_{Acc}(T_u, T_{o_s})}{3} \tag{7.15}$$

Die Kompatibilität einer Umgebungsontologie o_e mit einer Zielsystemontologie o_s hängt von den verwendeten Datentypen ab. Wie in Abschnitt 5.4 beschrieben, kann ein Zielsystem in ProNat komplexe Datentypen über seine Schnittstellen anbieten. Diese Datentypen können wiederum als Konzepte in den Umgebungsontologien angegeben werden. Anhand der Übereinstimmung der Datentypen einer Zielsystemontologie o_s mit den `Object`-Unterkonzepten der Umgebungsontologie

o_e kann die Ähnlichkeit Ψ_{Sys} bestimmt werden. Die Ähnlichkeit Ψ_{Sys} berechnet sich dann zu zwei Dritteln aus dem $\Psi_{Acc}(T_u, T_o)$-Wertes der Zielsystemontologie (siehe Gleichung 2.16) und einem Drittel aus dem soeben beschrieben Kompatibilitätswert. Formal ist Kompatibilität als Schnittmenge der Datentypen des Zielsystems (DT_{o_s}) und der Vereinigung aller Objekte-Konzepte der Umgebung ($\bigcup_{i \in O_{sel}} Objects_i$) definiert.

Beispiel:
Berechnung der Ähnlichkeit einer Zielsystemontologie zu einer natürlichsprachlichen Äußerung

In diesem Beispiel soll die Berechnung der Ähnlichkeit einer Zielsystemontologie zu einer natürlichsprachlichen Äußerung mithilfe der zuvor ausgewählten Umgebungsontologien demonstriert werden.

Annahmen

Sei

- der ermittelte Ähnlichkeitswert zwischen der Äußerung und der Zielsystemontologie $\Psi_{Acc}(T_u, T_i) = 0,6$,
- die Zielsystemontologie o_s mit den Datentypen DT_{o_s} `Locateable`, `Graspable` sowie `Openable` versehen und
- nur eine Umgebungsontologie ausgewählt, die zudem die folgenden `Object`-Konzepte beinhaltet: `Graspable`, `Openable` und `Closeable`.

Ergebnis

In diesem Fall ergibt sich für den hinteren Teil der Gleichung 7.15 (reine Ähnlichkeit) ein Wert von $\frac{2*0,6}{3} = 0,4$. Da die Mächtigkeit der Schnittmenge der Datentypen und `Object`-Konzepte 2 beträgt, wird der vordere Teil (Kompatibilität) zu $\frac{2}{3*3} = 0,22$ ausgewertet. Insgesamt ergibt sich damit folgender Ähnlichkeitswert: $\Psi_{Sys}(o_s, T_u, O_{sel}) = 0,22 + 0,4 = 0,62$.

Die Ontologie-Auswahl wird dementsprechend zweimal durchgeführt, zunächst um die Umgebungsontologien zu bestimmen und anschließend um die passende Zielsystemontologie zu bestimmen. So wird auch sichergestellt, dass immer mindestens eine Ontologie je Typ (Zielsystem und Umgebung) ausgewählt wird. Zuletzt werden die ausgewählten Ontologien zusammengeführt. Dieser Schritt ist wie oben beschrieben aufgrund der für Pro$\mathcal{NAT}$ festgeschriebenen Ontologiestrukturen trivial (siehe Abschnitt 5.4.1).

Tabelle 7.37: Übersicht über die in der Evaluation der Ontologie-Auswahl verwendeten Ontologien: Die Spalte *Typ* gibt an, ob es sich um eine Zielsystemontologie (Z) oder eine Umgebungsontologie (U) handelt.

Typ	Bezeichnung	Beschreibung
Z	Haushaltsroboter	Ein humanoider Haushaltsroboter wie ARMAR-III [Asf+06]
Z	Assistenzsystem	Ein intelligentes Assistenzsystem wie Amazons Alexa
Z	Drohne	Eine zivile Drohne, z.B. ein Quadrocopter
Z	Lego Mindstorm	Ein Lego-Mindstorm-Roboter mit Kettenantrieb und Greifer
U	Küche	Eine Küche, inklusive Möbeln, Utensilien, Lebensmitteln etc.
U	Bar	Eine Bar, inklusive Möbeln, Cocktail etc.
U	Garten	Ein Garten, inklusive Möbeln, Pflanzen, Werkzeugen etc.
U	Schlafzimmer	Ein Schlafzimmer mit Möbeln
U	Spielzimmer	Ein Kinderzimmer, inklusive Möbeln, Spielzeuge etc.
U	Musik	Mit Musik in Verbindung stehende Konzepte wie Genres, Instrumente etc.
U	Heizung	Eine Heizungsanlage und ähnliches, u. a. Heizungen, Klimaanlagen etc.
U	Waschküche	Eine Waschküche, inklusive Waschmaschine, Trockner, Waschmittel etc.

7.9.2 Evaluation des Agenten zur automatischen Auswahl von Zielsystem- und Umgebungsontologien

Für die Evaluation wurde der Datensatz der Evaluation des Agenten zur Modellierung und Etikettierung von Diskurs-Themen verwendet (siehe Abschnitt 7.3). Da der Agent zur Ontologie-Auswahl direkt von den Ergebnissen des Themen-Agenten abhängt, können die hier erzielten Ergebnisse auf diese Weise besser bewertet werden. Zusätzlich wurden neun weitere, synthetische Äußerungen erstellt (siehe Abschnitt D.2 des Anhangs). Diese decken zwei weitere Domänen ab und erfordern die Kombination mehrerer Umgebungsontologien. Der gesamte für die Evaluation genutzte Datensatz umfasst dementsprechend 33 Äußerungen, fünf davon erfordern die Kombination mehrerer Umgebungsontologien (hier *Heizung* und *Bar*, *Musik* und *Bar* sowie *Küche* und *Garten*). Insgesamt werden zwölf Ontologien angeboten, vier sind Zielsystemontologien und acht Umgebungsontologien. Eine Übersicht der verwendeten Ontologien wird in Tabelle 7.37 gegeben.

Für alle Äußerungen wurde ein Goldstandard erstellt; das bedeutet, es wurde für jede Äußerung festgelegt, welche Ontologien für die Ausführung der in der Äußerung genannten Aktionen (bzw. Anweisungen) benötigt werden. Um die Evaluation durchzuführen, wurde zunächst das vollständige ProNat-Vorverarbeitungsfließband ausgeführt (siehe Kapitel 6). Anschließend wurden die beiden Agenten zur Disambiguierung basierend auf *Wikipedia* und zur Modellierung und Etikettierung von Diskurs-Themen genau ein Mal und in dieser Reihenfolge ausgeführt (siehe Abschnitt 7.2 und Abschnitt 7.3). Zuletzt wurden der Agent zur automatischen Auswahl von Zielsystem- und Umgebungsontologien ausgeführt und die Ergebnisse mit dem Goldstandard verglichen. Zunächst werden nur die Umgebungsontologien betrachtet. Jede richtig ausgewählte Ontologie gilt als *richtig positives* Ergebnis, jede falsche (oder zusätzliche) als *falsch positives* Ergebnis und jede ausgelassene (obwohl erwartete) als *falsch negatives* Ergebnis. Als Metriken dienen Präzision, Ausbeute, F_1-Maß und die Falsch-Positiv-Rate (f_p-Rate, siehe Abschnitt 2.4). Letztere wurde hier zusätzlich verwendet,

Tabelle 7.38: Evaluationsergebnisse der Umgebungsontologieauswahl: Dargestellt sind die Werte für Präzision, Ausbeute, F_1-Maß und f_p-Rate, die von verschiedenen Konfigurationen des Agenten zur Themen-Modellierung und -Etikettierung (*TME*) und des Verfahrens zur Ontologie-Auswahl erzielt werden.

TME	Ontologie-Auswahl					
#Themen	#Themen	θ	Präzision	Ausbeute	F_1	f_p-Rate
2*n	5	0,90	0,919	0,895	0,907	0,013
2*n	5	0,85	0,809	0,895	0,850	0,035
2*n	5	0,80	0,708	0,895	0,791	0,062
5	5	0,90	0,850	0,895	0,872	0,027
5	5	0,85	0,791	0,895	0,840	0,040
5	5	0,80	0,761	0,921	0,833	0,049
2*n	10	0,90	0,778	0,921	0,843	0,044
5	10	0,90	0,790	0,895	0,840	0,040

da (wie in Abschnitt 5.4 diskutiert) ProNat mit möglichst kleinen Ontologien konfiguriert werden soll. Mithilfe der Falsch-Positiv-Rate kann bestimmt werden, wie viele (Teil-)Ontologien unnötig zur Konfiguration hinzugefügt wurden.

Tabelle 7.38 zeigt die Ergebnisse der Evaluation für unterschiedliche Konfigurationen. Die erste Spalte gibt die Konfiguration des Agenten zur Modellierung und Etikettierung von Diskurs-Themen an, genauer gesagt, wie viele Themen-Etiketten dieser pro Äußerung generiert. Das n ist dabei die Anzahl an einzigartigen Bedeutungen der Substantive in der natürlichsprachlichen Äußerung (siehe Abschnitt 7.2). Die zweite Spalte gibt die Anzahl an generierten Themen für die Ontologien an und die dritte den verwendeten Auswahl-Faktor θ. Unabhängig von der gewählten Konfiguration erreicht das Verfahren gute Evaluationsergebnisse; das F_1-Maß liegt je nach Konfiguration zwischen 0,791 und 0,907. Auch die Falsch-Positiv-Rate ist durchgehend niedrig. Insgesamt werden die besten Ergebnisse für die Konfiguration in Zeile 1 erzielt ($2 * n$ Themen je Äußerung, 5 Themen je Ontologie und $\theta = 0,90$). Auffallend ist, dass diese Konfiguration die restriktivste unter den getesteten ist; es werden sowohl wenig Themen-Etiketten zur Bestimmung der Ähnlichkeit $\Psi_{Acc}(T_u, T_o)$ zwischen Äußerung und Ontologie als auch ein hoher θ-Wert verwendet. Letzteres führt zu hohen Schwellenwerten bei der Auswahl der Ontologien (siehe Gleichung 7.14). Dies führt wiederum zu insgesamt weniger ausgewählten Ontologien. Anscheinend führt eine präzise Auswahl der Ontologien gegenüber einer hohen Trefferrate zu insgesamt besseren Ergebnissen. Die ermittelten Werte für die beiden Metriken Präzision und Ausbeute bestätigen diese Vermutung. Die Ausbeute bleibt weitestgehend konstant und steigt nicht etwa für weniger restriktive Konfigurationen. Im Gegensatz dazu sinkt die Präzision deutlich, insbesondere für niedrige θ-Werte. Das bedeutet, werden mehr Ontologien ausgewählt sind diese zusätzlichen meist falsch, was auch die steigende Falsch-Positiv-Rate bestätigt.

Eine Analyse der *falsch positiven* Ergebnisse ergab, dass in den meisten Fällen die fälschlicherweise ausgewählten viel mit den eigentlich richtigen Ontologien des Goldstandards gemeinsam hatten. Das bedeutet, sie beinhalten ähnliche oder gleiche Konzepte bzw. Individuen. So enthalten beispielsweise sowohl die Ontologie *Bar* als auch die Ontologie *Küche* die Individuen *orange juice* und *fridge*.

Tabelle 7.39: Werte für die Ausbeute bei der Umgebungsontologieauswahl für Äußerungen, die die Verwendung von mehr als einer Ontologie erfordern.

Äußerung	Gold-Standard	tatsächlich ausgewählt	Ausbeute
Extra_3.1	Heizung + Musik	Heizung	0,500
Extra_3.2	Heizung + Musik	Heizung	0,500
Extra_8.1	Bar + Musik	Bar	0,500
Extra_8.2	Bar + Musik	Bar + Musik	1,000
Extra_9.1	Garten + Küche	Garten	0,500

Auch die *falsch negativen* Ergebnisse wurden untersucht. Diese lassen sich mehrheitlich auf Äußerungen zurückführen, die die Verwendung mehrerer Umgebungsontologien erfordern. Tabelle 7.39 zeigt eine Analyse dieser Äußerungen. Wie zu erkennen ist, wird in vier von fünf Fällen nur eine statt zwei Ontologien ausgewählt. Dies kann darauf zurückgeführt werden, dass bei der Generierung der Themen-Etiketten ein Themenbereich den anderen dominiert. Infolgedessen kann anschließend auch nur die Ähnlichkeit eines Themas der Äußerung zu den Ontologien ermittelt werden, weshalb auch immer eine passende Ontologie, aber eben nur diese ermittelt wird. Konkret können für diese Äußerungen nur 65,90% der Bedeutungen mit einem Thema der Ontologie in Verbindung gebracht werden. Den übrigen wird auf Sprachseite erst gar kein Thema zugewiesen. Im Vergleich dazu kann über alle Äußerungen hinweg für 95,79% der Bedeutungen eine Verbindung zu einem Ontologie-Thema hergestellt werden.

Hinsichtlich der Auswahl von Zielsystemontologien besteht noch Verbesserungspotenzial; ohne die Adaption des Verfahrens kann keine Zielsystemontologie ermittelt werden (siehe Gleichung 7.15). Wie bereits zuvor diskutiert, ist es fast unmöglich, mit dem vorgestellten Verfahren Zielsystemontologien zu ermitteln, da systemrelevante Begriffe in natürlichsprachlichen Äußerungen kaum auftreten. Mit der beschriebenen Adaption hingegen kann immer eine Zielsystemontologie bestimmt werden. Allerdings ist dies in 94% der Fälle der Haushaltsroboter. Auch wenn tatsächlich 25 der 33 Äußerungen diesen als Zielsystem erfordern, ist die Auswahl anhand der Datentypen anscheinend unzureichend. Im konkreten Fall ist dies darin begründet, dass der Roboter in nahezu allen Umgebungen operieren kann; das bedeutet, er ist mit fast allen Umgebungsontologien kompatibel und wird daher bevorzugt ausgewählt. Hier könnte ein verbesserte Evaluation (Zielsysteme mit unterschiedlichen Datentypen und eine balancierte Menge von Äußerungen, die unterschiedliche Zielsysteme in ähnlicher Verteilung erfordern) Aufschluss darüber geben, ob es sich um ein konzeptionelles Problem handelt oder ob lediglich die Evaluationsdaten ungünstig gewählt wurden.

Zusammenfassend lässt sich festhalten, dass der Ansatz zur automatischen Auswahl von Ontologien gute Ergebnisse liefert. Dies gilt insbesondere für Umgebungsontologien. Besonders positiv ist zu bewerten, dass die Auswahl in den meisten Fällen korrekt funktioniert, obwohl das Verfahren direkt von den nicht immer präzisen Ergebnissen des Agenten zur Modellierung und Etikettierung von Diskurs-Themen abhängt. Das bedeutet, dass der Ansatz auch mit unpräzisen Themen-Etiketten als Basis funktioniert, solange Zusammenhänge zwischen den Themen der natürlichsprachlichen Äußerung und denen der jeweiligen Ontologie hergestellt werden können. Erfordern Äußerungen mehrere Umgebungsontologien, kann bisher nur eine der Ontologien zuverlässig bestimmt werden. Auch die

Auswahl der Zielsystemontologien anhand der Datentypen ist ausbaufähig. Zwar funktioniert das Verfahren prinzipiell; sind allerdings Systeme vorhanden, die mit vielen Umgebungen kompatibel sind, werden diese bevorzugt ausgewählt, selbst wenn ein anderes passender wäre.

7.10 Dialog-Agent

Die Erstellung von Quelltext ist ein kreativer und komplexer Schaffensprozess. Programmierende sind daher auf Rückmeldung angewiesen; selbst kleine Programme gelingen selten auf Anhieb. Je besser die Rückmeldungen sind, die der Programmierende während der Erstellung seines Programms erhält, desto besser gelingt die Programmierung. Normalerweise erhält man bei der Entwicklung von Software Rückmeldung in Form von Warnungen oder Fehlermeldungen des Übersetzers (engl. *compiler*). Für die Programmierung mit gesprochener Sprache, wie sie in ProNat angestrebt wird, muss hingegen auch die Rückmeldung über die Sprachschnittstelle erfolgen. Das bedeutet, das System und der Nutzer treten in einen Dialog. Hierfür müssen alternative Rückmeldungsmechanismen entwickelt werden, denn klassische Fehlermeldungen können nicht (bzw. nur sehr schlecht) über eine Sprachschnittstelle kommuniziert werden. Zudem richtet sich ProNat an Programmierlaien, die herkömmliche Übersetzerwarnungen bzw. -fehlermeldungen nicht oder nur schwer interpretieren können. Die Rückmeldungen im Kontext der Programmierung mit gesprochener Sprache müssen also leicht verständlich sein. Außerdem sollten Rückfragen möglichst gezielt zur Lösung eines konkreten Problems gestellt werden.

7.10.1 Eine Dialog-Komponente für ProNat

Die zugrundeliegende Prämisse bei der Entwicklung der Dialog-Komponente für ProNat ist, dass die ursprüngliche Beschreibung des Nutzers zur Erzeugung des Programms genügt. Rückfragen sollen – soweit möglich – vermieden und nur dann gestellt werden, wenn Mehrdeutigkeiten oder Lücken nicht über den Kontext erschlossen werden können. Das bedeutet, die Dialog-Komponente soll dem Prinzip der *Nutzerinitiative* folgen (siehe Abschnitt 2.3.10).

Die größte Herausforderung bei der Entwicklung einer Dialog-Komponente für die Programmierung mit gesprochener Sprache ist es, programmatische Strukturen für Laien verständlich zu verbalisieren. Zusätzlich muss die Komponente so entworfen werden, dass sie einfach erweiterbar ist; schließlich erlaubt ProNat jederzeit das Hinzufügen neuer Agenten. Neue Agenten erzeugen wiederum neue Analyseergebnisse, die potenziell neuartige Rückfragen erfordern. Daher bietet sich im Kontext von ProNat eine regelbasierter Ansatz für die Dialog-Komponente an. In Abschnitt 2.3.10 wurde diskutiert, dass Forschungsprojekte vorrangig statistikbasierte Ansätze wählen, während industrielle Systeme zumeist regelbasiert sind. Auch wenn hier eine Dialog-Komponente für ein Forschungsprojekt entwickelt werden soll, ähneln die Randbedingung eher denen einer industriellen Lösung. Zunächst steht die hier entwickelte Dialog-Komponente nicht für sich, sondern muss in eine Rahmenarchitektur eingepasst werden. Das bedeutet, die Dialog-Komponente muss als Fließbandstufe oder Agent umgesetzt werden und Informationen aus dem ProNat-Graphen entnehmen und

zurückschreiben können. Die Dialog-Komponente muss zudem einfach erweiterbar sein, da ProNat jederzeit neue Agenten oder Fließbandstufen hinzugefügt werden können. Jedes neue Modul kann potenziell mehrere neue Dialogsituationen erzeugen. Gleichzeitig sollen so wenige Rückfragen wie möglich gestellt und dem Nutzer die Initiative überlassen werden. Beides spricht eher für einen konservativen, robusten Entwurf. Zuletzt müssen die anderen Agenten die Ergebnisse eines Dialogs verwerten können; das bedeutet, die Ergebnisse müssen vorhersag- und nachvollziehbar sein.

In Abschnitt 2.3.10 wurden bereits die grundlegende Architektur der meisten Dialogsysteme vorgestellt. Dialogsysteme bestehen üblicherweise aus den folgenden Modulen: (automatische) Spracherkennung, Sprachverständnis, Dialog-Management, Antworterzeugung und Sprachsynthese. Diese Module müssen auch in der Dialog-Komponente für ProNat umgesetzt werden. Einige Module sind bereits Bestandteil von ProNat, wie beispielsweise die automatische Spracherkennung (siehe Abschnitt 6.1), andere müssen geeignet in die Rahmenarchitektur integriert werden.

7.10.2 Implementierung des Dialog-Agenten

Die Dialog-Komponente für ProNat wurde als Agent implementiert [WHL18][63]. Durch die Implementierung als Agent kann die Dialog-Komponente die Analysen der anderen Agenten nutzen. Gleichzeitig kann sie diesen neue Informationen zur Verfügung stellen, die gegebenenfalls neue Analyseschritte ermöglichen.

Aufbau

Für die Implementierung des Agenten sollen soweit wie möglich bereits vorhandene Ressourcen verwendet werden. Die ersten beiden der in Abbildung 2.15 dargestellten Module werden bereits von ProNat zur Verfügung gestellt. ProNat verfügt über eine Vorverarbeitungsstufe zur automatischen Spracherkennung (siehe Abschnitt 6.1). Ein eigenes Spracherkennungsmodul für den Dialog-Agenten muss somit nicht implementiert werden. Das Verständnis natürlicher Sprache wird in ProNat durch das Zusammenspiel der Agenten bewerkstelligt; eine gesonderte Lösung für den Dialog-Agenten ist nicht nötig. Der Dialog-Agent kann das Vorverarbeitungsfließband und alle durch die Agenten erzeugten Informationen verwenden, und zwar nicht nur zur Interpretation der ursprünglichen Äußerung, sondern auch für etwaige Antworten. Je Nutzerantwort wird außerdem ein neuer (Antwort-) Graph erzeugt, der jedoch nur vom Dialog-Agenten interpretiert wird. Extrahierte Informationen werden dann wiederum im eigentlichen ProNat-Graphen hinterlegt.

Das Dialog-Management wird zweiteilig umgesetzt. Zum einen beobachtet der Dialog-Agent dauerhaft den ProNat-Graphen, um mehrdeutige oder lückenhafte Analyseergebnisse der anderen Agenten zu identifizieren und Gesprächssituationen daraus abzuleiten. Können neue Informationen infolge eines Dialogs gewonnen werden, schreibt der Agent diese wieder in den Graphen zurück. Der Graph dient also als *globales* Dialog-Modell. Diese Art des Dialog-Managements allein ist allerdings unzureichend. Je nach Gesprächssituation kann es nötig sein, mehrere aufeinanderfolgende

[63] Tatsächlich wird zusätzlich ein modifiziertes Vorverarbeitungsfließband verwendet, die eigentliche Anwendungslogik befindet sich jedoch im Agenten. Der genau Aufbau der einzelnen Bestandteile der Dialog-Komponente wird später erläutert.

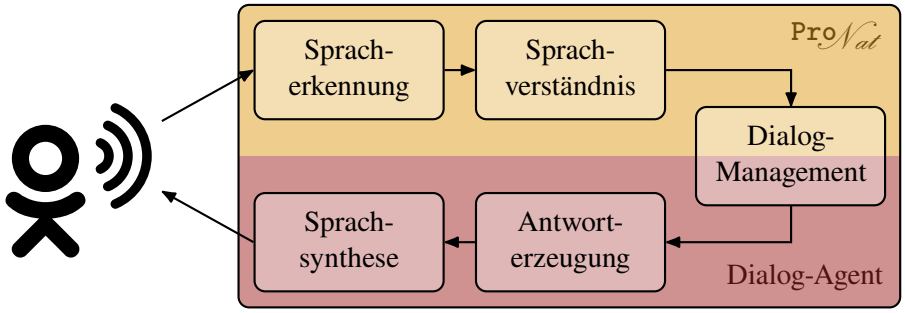


Abbildung 7.22: Aufteilung der Module der Dialog-Komponente in bereits durch ProNat zur Verfügung gestellte und im Dialog-Agenten implementierte.

Fragen zu stellen. In diesen Situationen muss der Zustand des Agenten bzw. der Gesprächssituation zusätzlich modelliert werden, um die Wiederholung gleichartiger Fragen und Gesprächssackgassen zu vermeiden. Daher verwaltet der Agent zusätzlich einen *lokalen* Zustand des Dialogs bis die jeweilige Gesprächssituation abgeschlossen ist und die jeweiligen Ergebnisse im *globalen* Modell hinterlegt wurden.

Die Antworterzeugung und die Sprachsynthese sind Bestandteil der Implementierung des Dialog-Agenten. Erstere verwendet Satzschablonen, die mit aktuellen Werten befüllt werden. Um ein abwechslungsreiches Nutzungserlebnis zu ermöglichen, wurden für gleiche Gesprächssituationen mehrere Schablonen implementiert. Für die Sprachsynthese wird eine von IBM zur Verfügung gestellte Web-API verwendet: *Watson – Text to Speech*[64]. Die Verteilung der Module der Dialog-Komponente auf bereits von ProNat zur Verfügung gestellte und im Dialog-Agenten implementierte ist in Abbildung 7.22 dargestellt.

Die wesentlichen Bestandteile des Dialog-Agenten sind das lokale Dialog-Management und die Antworterzeugung, die folgendermaßen umgesetzt werden. Es werden sogenannte *Gesprächssituationen* (engl. *dialog acts*) definiert, die auf bestimmte Muster im ProNat-Graphen reagieren. Jede Gesprächssituation betrachtet eine potenzielle Fehlerquelle, Unvollständigkeit oder Unsicherheit in den Analyseergebnissen der anderen Agenten. Beispielsweise sind niedrige Konfidenzen bei der automatischen Spracherkennung ein Indiz für mögliche Wortfehler. Gesprächssituationen bestehen aus Indikatoren (Graphmuster, auf die sie reagieren sollen), sowie Informationen über das jeweilige *lokale* Dialog-Management und die Antworterzeugung. Durch die Kapselung in Gesprächssituationen ist der Dialog-Agent einfach erweiterbar. Wird eine neue potenzielle Fehlerquelle bekannt, beispielsweise durch das Hinzufügen eines neuen Agenten, kann einfach eine neue Gesprächssituation implementiert werden, während der Rest der Logik des Dialog-Agenten unverändert bleibt. Die einzelnen Gesprächssituationen werden in einer Zuständigkeitskette (engl. *chain of responsibility*) organisiert [Gam+94]. Beginnend mit der ersten Gesprächssituation der Kette, überprüft jede Gesprächssituation den Graphen auf die definierten Indikatoren. Falls keins der Muster im Graph gefunden wird, übernimmt das nächste Kettenglied die Kontrolle. Wird jedoch eine Übereinstimmung detektiert, wird ein Dialog initiiert und die Gesprächssituation behält die

[64] Watson – Text to Speech: `https://www.ibm.com/cloud/watson-text-to-speech`, zuletzt besucht am 24.02.2021.

Kontrolle, bis diese abgeschlossen ist (lokales Dialog-Management). Kann eine Gesprächssituation erfolgreich abgeschlossen werden, beginnt die Verarbeitung in der Zuständigkeitskette von vorne. Dabei kann es vorkommen, dass der erfolgreiche Abschluss einer Gesprächssituation eine andere ebenfalls löst. Zum Beispiel könnte ein im Dialog bereinigter Wortfehler (verursacht von der automatischen Spracherkennung) dazu führen, dass eine zuvor mehrdeutige Korreferenz in der Folge korrekt analysiert werden kann, ohne dass weitere Rückfragen nötig wären. Falls kein Indikator aller Gesprächssituationen erfüllt ist, wird davon ausgegangen, dass kein weiterer Dialog nötig ist.

Gesprächssituationen

Wie zuvor diskutiert, soll der Dialog-Agent reaktiv agieren. Hierzu wird das Konzept der *Gesprächssituation* eingeführt, um mögliche Problemquellen im ProNat-Graphen zu identifizieren, einen entsprechenden Dialog zu initiieren und die erhaltenen Informationen in den Graphen zurückzuschreiben. Das Konzept *Gesprächssituation* ist aus verwandten Arbeiten entliehen[65]. Dort bezeichnet eine Gesprächssituation (engl. *dialog act*) eine Äußerung, die eine Systemreaktion erfordert. Für den Dialog-Agenten wird diese Idee adaptiert und auf Graphmuster übertragen (die eine Reaktion erfordern). Beispielsweise weist ein niedriger Wert des Attributs `asrConfidence` in einem Knoten vom Typ `Token` darauf hin, dass an dieser Stelle ein potenzieller Wortfehler vorliegt (siehe Abschnitt 6.1). Ein anderes Muster, das auf ein mögliches Problem hinweist, besteht aus einem Knoten vom Typ `Token` mit mehr als einer ausgehenden Kante vom Typ `contextRelation`, die zudem ähnliche Konfidenzen aufweisen. Hier handelt es sich um eine mehrdeutige Korreferenz (siehe Abschnitt 7.5). Graphmuster, die eine Systemreaktion erfordern, werden im Folgenden als *Indikatoren* bezeichnet. Gesprächssituationen können mehrere Indikatoren zugeordnet werden. Zu jeder Gesprächssituation wurde zudem eine Dialog-Strategie entworfen, sodass dem Nutzer zur Situation passende Fragen gestellt werden. Die Dialog-Strategien sind als endlicher Zustandsautomat implementiert; jeder Zustandsübergang löst eine Systemreaktion aus; das bedeutet, dem Benutzer wird eine neue Frage gestellt. Innerhalb der Zustände wird jeweils die letzte Nutzeräußerung interpretiert. Die Eingangsaktion des Endzustands führt dazu, dass die aus dem Dialog extrahierten Informationen in den ProNat-Graphen zurückgeschrieben werden. Die konkreten Umsetzungen der Strategien werden im Folgenden als *Dialog-Management* bezeichnet. Die in den Zuständen definierten Fragen werden mithilfe von Schablonen erzeugt. Je Zustand kann es mehrere Schablonen geben, die zwar semantisch gleich, aber unterschiedlich formuliert sind. Die implementierten Schablonen werden im Folgenden unter dem Begriff *Antworterzeugung* diskutiert.

Wie zuvor erläutert, impliziert jeder Agent potenziell mehrere Gesprächssituationen. Nicht alle lassen sich jedoch mit Indikatoren, wie oben beschrieben, fassen. Um die Umsetzbarkeit des entworfenen Konzepts zu belegen, wurden exemplarisch sechs Gesprächssituationen implementiert, die sich den Analyseergebnissen der Agenten zur Korreferenzanalyse und Erkennung von bedingten Verzweigungen sowie der Vorverarbeitungsstufe zur automatischen Spracherkennung zuordnen lassen. Tabelle 7.40 listet die implementierten Gesprächssituationen und die jeweils zugehörigen Indikatoren. Im Folgenden werden die einzelnen Gesprächssituationen genauer erläutert.

[65] Siehe hierzu beispielsweise die Arbeiten von Jokinen und McTear [JM09a] sowie Tur und De Mori [TD11].

Tabelle 7.40: Implementierte Gesprächssituationen und zugehörige Indikatoren: Der Indikator *viele Wortalternativen* ist ausgegraut, da dieser zwar implementiert wurde, aber nicht verwendet werden sollte.

Gesprächssituation	**Indikator**
Spracherkennung unsicher	geringe Wortkonfidenz
	viele Wortalternativen
Korreferenz mehrdeutig,	ähnliche Konfidenzen unterschiedl. Referenzen
unvollständig,	Pronomen ohne Referenten
oder unsicher	geringe Konfidenz der einzigen Korreferenz
Verzweigte Bedingung unvollständig	bedingte Verzweigung ohne *Dann*-Gliedsatz

Spracherkennungsunsicherheit

Die automatische Spracherkennung bildet den ersten Schritt im Verarbeitungsprozesses von ProNat. Alle nachfolgenden Analysen werden direkt oder indirekt von Wortfehlern in der Transkription beeinflusst. Daher ist die Korrektur dieser nicht nur besonders wichtig, sondern erübrigt gegebenenfalls weitere Korrekturen, da nachfolgende Analysen nach der Verbesserung eines Wortfehlers bessere Ergebnisse erzielen.

Indikatoren – Wie Tabelle 7.40 zeigt, gibt es zwei Indikatoren für potenzielle Wortfehler der automatischen Spracherkennung: eine geringe Konfidenz, die der Erkenner dem Wort auf dem ersten Rang zugeordnet hat (primär erkanntes Wort), und viele durch den Erkenner erzeugte Alternativen zu einem Wort. Die Anzahl der erzeugten Alternativen hängt stark vom eingesetzten automatischen Spracherkenner, dessen Konfiguration[66] und vom Wort selbst ab[67]. Aus diesem Grund wurde dieser Indikator zwar implementiert, sollte aber nicht verwendet werden. Für den Indikator, der sich auf die Konfidenz bezieht, wurde ein Schwellenwert festgelegt. Wann immer dieser konfigurierbare Schwellenwert unterschritten wird, soll ein Dialog initiiert werden. Der Einfluss des festgelegten Wertes auf das Ergebnis wird in Abschnitt 7.10.3 diskutiert.

Dialog-Management – Für das Dialog-Management gilt es, geeignet nach potenziellen Wortfehlern zu fragen. Eine direkte Frage nach dem Wort selbst ist nicht möglich, da das Wort mehrfach in einer Äußerung auftreten kann und die Frage entsprechend mehrdeutig wäre. Auch die vom Spracherkenner erzeugten Alternativen sind nicht hilfreich, da es sich inhärent um Homophone handelt; eine Fragestellung, wie „Do you mean write or right?“, führt höchstwahrscheinlich zu keinem Erkenntnisgewinn. Daher wurde folgende Strategie entworfen: Der Nutzer wird gebeten, den Teil der Äußerung, der den potenziellen Wortfehler enthält, zu wiederholen. Anschließend wird im ersten Zustand des Dialog-Managements versucht, die Antwort an der ursprünglichen Aussage auszurichten. Die beiden Äußerungen werden anhand von Wörtern mit hohen Konfidenzen, die zudem gleich sind und an derselben Stelle stehen, ausgerichtet. Anschließend werden die fehlerhaften Wörter der ursprünglichen Äußerungen mit denen der Nutzerantwort ersetzt, sofern diese eine Konfidenz aufweisen, die über dem festgelegten Schwellenwert liegt. Falls die Antwort nicht ausgerichtet werden kann, wird in einen zweiten Zustand übergegangen und der Nutzer wird

[66] ProNat erlaubt die Verwendung unterschiedlicher Spracherkenner, die zudem konfigurierbar sind (siehe Abschnitt 6.1)
[67] Besitzt ein Wort viele Homophone, steigt die Wahrscheinlichkeit eines Wortfehlers.

erneut aufgefordert, die ursprüngliche Äußerung exakt zu wiederholen. Falls auch dies scheitert, wird die Kontrolle an die nächste Gesprächssituation der Zuständigkeitskette abgegeben.

Beispiel:
Ausrichtung der Nutzerantwort und Austausch fehlerhafter Wörter

In diesem Beispiel soll die Ausrichtung der Nutzerantwort an der ursprünglichen Äußerung und der Austausch der fehlerhaften Wörter demonstriert werden. In den Äußerungen sind pro Wort die Konfidenzen, die durch einen hypothetischen Spracherkenner erzeugt wurden, tiefgestellt angegeben. Als Schwellenwert für den Indikator wird im Beispiel 0,8 verwendet.

Ursprüngliche Äußerung

... $take_{0,91}$ $the_{0,82}$ $right_{0,72}$ $french_{0,52}$ ...

Nutzerantwort

... $take_{0,94}$ $the_{0,92}$ $white_{0,85}$ $fridge_{0,81}$ ...

Erläuterung

In der ursprünglichen Äußerung unterschreiten die Wörter *right* und *french* den Schwellenwert. Die wiederholte Äußerung des Nutzers lässt sich anhand der ersten beiden Wörter mit hoher Konfidenz an der ursprünglichen Äußerung ausrichten. Die letzten beiden Wörter unterscheiden sich von denen der ursprünglichen Äußerung und überschreiten den Schwellenwert. Dementsprechend nehmen sie den Platz der ursprünglichen Wörter ein.

Antworterzeugung – Wie zuvor beschrieben, soll der Nutzer dazu angeregt werden, den Teil der Aussage, für den ein Wortfehler vermutet wird, zu wiederholen. Hierzu werden Satzschablonen wie die folgende verwendet: „Please repeat the following part of your statement: `[PART]`“. Der zu wiederholende Abschnitt (`[PART]`) besteht aus allen Wörtern zwischen dem Anfang der Verbalphrase vor und dem Ende der Nominalphrase nach dem potenziell fehlerhaften Wort. Für die zweite Nachfrage wird lediglich der statische Teil der Satzschablone angepasst und der Nutzer wird darauf hingewiesen, die gleiche Formulierung, wie in der ursprünglichen Äußerung zu verwenden.

Mehrdeutige, unvollständige oder unsichere Korreferenz

Die Auflösung von Korreferenzen ist seit jeher eines der schwierigsten Probleme der klassischen Computerlinguistik und inhärent mit Unsicherheiten behaftet. Daher verwendet auch der zugehörige `Pro`$\mathcal{N}_{at}$-Agent ein mehrstufiges Bewertungssytem und erzeugt eine sortierte Kandidatenliste statt ein eindeutiges Ergebnis auszugeben. Daher können zweifelhafte Korreferenzen durch den Dialog-Agenten korrigiert werden.

Indikatoren – Wie in Abschnitt 7.5 beschrieben, werden Korreferenzen in ProNat wie folgt dargestellt: Eine Kante vom Typ `contextRelation` verbindet den Knoten, der den referenzierenden Ausdruck darstellt, mit einem potenziellen Referenten. Dabei kann es vorkommen, dass mehrere Kanten vom selben referenzierenden Ausdruck ausgehen. Jede dieser Kanten verfügt über ein Attribut, das die Konfidenz für die jeweilige Korreferenzbeziehung angibt. Dieses Analyseergebnis birgt drei mögliche Fehlerquellen. Zunächst kann eine Korreferenz fehlen. Ein Indikator hierfür ist ein Personalpronomen (als referenzierender Ausdruck), das keine ausgehende Kante vom Typ `contextRelation` besitzt. Des Weiteren können Korreferenzbeziehungen uneindeutig sein. Dies ist der Fall, wenn es mehr als eine ausgehende Kante vom Typ `contextRelation` gibt, deren Konfidenzen ähnlich hoch sind[68]. Zuletzt kann eine Korreferenzbeziehung unsicher sein, und zwar dann, wenn es nur eine Kante vom Typ `contextRelation` gibt und die zugehörige Konfidenz niedrig ist[69].

Dialog-Management – Laien ist das Konzept *Korreferenz* üblicherweise unbekannt; eine Abfrage potenzieller Korreferenz-Kandidaten ist somit nicht möglich. Direkte Fragen, wie „Does *it* refer to *X*?“, sind auch nicht möglich, da *it* und *X* unter Umständen mehrfach in der Äußerung auftreten. Stattdessen wird wie folgt verfahren: Es wird der Teil der Äußerung wiederholt, der sowohl den referenzierenden Ausdruck als auch die möglichen Referenten enthält. Anschließend wird der Nutzer gefragt, auf welchen Referenten sich der referenzierende Ausdruck bezieht. Der Vorteil dieses Vorgehens ist, dass so direkt alle zuvor beschriebenen Problemquellen abgedeckt werden. Zur Interpretation der Nutzerantwort wird diese wiederum an der ursprünglichen Äußerung ausgerichtet. Falls dies nicht möglich ist, wird in den nächsten Zustand übergegangen. Hier werden dem Nutzer erneut alle potenziellen Referenten genannt. Bleibt auch dies erfolglos, wird im letzten Zustand einzeln nach den Kandidaten gefragt.

Antworterzeugung – Um den Nutzer nach der korrekten Korreferenz zu fragen, werden zunächst der zugehörige referenzierende Ausdruck (`[REF_EX]`) und die potenziellen Referenten (`[LIST_RT]`) extrahiert. Anschließend werden alle Phrasen zwischen diesen Ausdrücken sowie die Phrasen, zu denen die Ausdrücke gehören, bestimmt (`[ENCL_PHRASES]`). Daraufhin werden Schablonen wie die folgende verwendet, um die Information zu erfragen: „In the following, what does the word `[REF_EX]` refer to? `[ENCL_PHRASES]`.“ Im zweiten Zustand werden andersartige Schablonen verwendet: „I don’t understand. You’ve mentioned |`[LIST_RT]`| entities. Now, please tell me what does the `[REF_EX]` refer to, $\texttt{[LIST\_RT]}_{0,N-1}$, or $\texttt{[LIST\_RT]}_N$?“ Falls keine Liste potenzieller Referent-Kandidaten existiert (im Falle eines Personalpronomens ohne Korreferenz), wird nur die erste Frage inklusive bis zu zwei einschließenden Phrasen gestellt.

Unvollständige bedingte Verzweigung

Die Erkennung bedingter Verzweigungen wird im zugehörigen Agenten auf das grammatikalische Konstrukt der Bedingungssätze zurückgeführt (siehe Abschnitt 7.6). Diese folgen jedoch keiner einheitlichen Syntax; der beschriebene Ansatz funktioniert dementsprechend nicht in allen Fällen, gegebenenfalls werden Konditional- oder abhängige Gliedsätze nicht (bzw. nicht vollständig)

[68] Welche Differenz (der Konfidenzen) als Indikator wirkt, kann konfiguriert werden.
[69] Auch dieser Schwellenwert kann konfiguriert werden.

erkannt. Prinzipiell könnten hier mehrere Gesprächssituationen zu potenziellen Fehlerquellen abgeleitet werden. Wie in Abschnitt 7.6.1 diskutiert, ist die Länge der Gliedsätze unbeschränkt; die Vollständigkeit lässt sich daher kaum prüfen. Ebenso genügt für einen natürlichsprachlichen Bedingungssatz das Vorhandensein eines *Dann*-Gliedsatzes, *Andernfalls*-Gliedsätze sind optional. Das bedeutet, die einzig überprüfbare Fehlerquelle ist ein Konditionalsatz dem kein *Dann*-Gliedsatz zugeordnet werden kann.

Indikatoren – Der einzige Indikator für diese Gesprächssituation ist das Fehlen eines *Dann*-Gliedsatzes. Wurden in einer Äußerung ein oder mehrere Konditionalsätze erkannt, muss jedem genau ein *Dann*-Gliedsatz zugeordnet sein (zu erkennen an einer Kante vom Typ *statement*, siehe Abschnitt 7.6.1). Ist dies nicht der Fall, wird für jede dieser Fehlerstellen ein Dialog initiiert.

Dialog-Management – Das Dialog-Management soll den Nutzer dazu veranlassen, den fehlenden *Dann*-Gliedsatz anzugeben; es werden die folgenden Schritte durchgeführt: Zunächst wird der Nutzer gebeten, die Instruktionen, die den *Dann*-Gliedsatz bilden, zu wiederholen. Falls die Antwort an der ursprünglichen Äußerung ausgerichtet werden kann, werden die entsprechenden Instruktionen in den ProNat-Graphen eingefügt. Falls nicht, wird in den nächsten Zustand übergegangen. Nun wird der Konditionalsatz wiederholt und der Nutzer wird direkt gefragt, welche Aktionen durchgeführt werden sollen, wenn diese Bedingung zutrifft. Falls die entsprechenden Informationen erneut nicht aus der Antwort gewonnen werden können, wird in einen weiteren Zustand übergegangen, indem dieselbe Frage mit einer anderen Formulierung wiederholt wird. Schlägt auch das fehl, kann die Gesprächssituation nicht erfolgreich abgeschlossen werden und die Kontrolle wird an das nächste Glied der Zuständigkeitskette übergeben.

Antworterzeugung – Für die Antworterzeugung wird der Konditionalsatz, dem kein *Dann*-Gliedsatz zugeordnet werden konnte, aus der Äußerung extrahiert (`[IF_PART]`). Zusätzlich werden alle nachfolgenden Phrasen (bis hin zum nächsten Konditionalsatz) extrahiert; diese gehören potenziell zum *Dann*-Gliedsatz (`[POT_THEN_PART]`). Letztere werden in die Schablonen, die für die Fragen im ersten Zustand des Dialog-Managements verwendet werden, eingesetzt, wie zum Beispiel: „Please just say the then condition of the following part again: `[POT_THEN_PART]`." Schablonen, die im zweiten Zustand verwendet werden, sind wie folgt formuliert: „Please just repeat the part, which tells me what I have to do if the condition is true. Your words were: `[POT_THEN_PART]`." Für den letzten Zustand werden Schablonen folgender Art verwendet: „Please repeat the instruction in your statement, if the following condition is satisfied: `[IF_PART]`."

7.10.3 Evaluation des Dialog-Agenten

Die Evaluation des Dialog-Agenten basiert auf den Dialog-Aufzeichnungen des ProNat-Korpus (siehe Abschnitt 5.5.2). Zum einen existieren Dialog-Aufzeichnungen zu je einer Äußerung zu den Szenarien vier und sechs. Hier wurden bereits aufgezeichnete Äußerungen zur Evaluation des Agenten genutzt, indem zu diesen Nachfragen formuliert wurden. Das Szenario acht hingegen besteht ausschließlich aus Dialog-Aufzeichnungen. Hier wurden neue Gesprächssituationen initiiert. Zu allen Szenarien existieren je zehn Dialog-Aufzeichnungen, das heißt, in Summe 30.

Die Äußerungen zu Szenario vier enthalten verbalisierte bedingte Verzweigungen. Für die Evaluation des Dialog-Agenten wurde eine Äußerung verwendet für die der Agent zur Erkennung von bedingten Verzweigungen keinen *Dann*-Gliedsatz zu einem Konditionalsatz zuweisen konnte[70]:

> *Hi robo, please look at the table, if there are dirty dishes please put it in the dishwasher and if there are clean dishes put it in the cupboard.*

Ähnlich verhält es sich für die Auswahl der Äußerung zu Szenario sechs. Hier wurde eine Äußerung verwendet, bei der eine Korreferenz falsch und eine weitere nicht eindeutig aufgelöst wurde:

> *Hey Armar, can you please get the green cup from the table, it is located quite in the middle, please fill it afterwards with water from the fridge, the water is in the fridge right next to the orange juice, then you can bring the cup to me. Afterwards empty the dishwasher, there are red cups in there, so you have to open it and then put them into the cupboard.*

Der Vorteil der Verwendung bestehender Aufnahmen gegenüber einem freien Dialog besteht darin, dass die Fehlerquellen und die erwarteten Lösungen vorab bekannt sind. Somit kann der Erfolg eines geführten Dialogs objektiv bemessen und über unterschiedliche Dialog-Aufzeichnungen verglichen werden. Nichtsdestotrotz soll zusätzlich bemessen werden, wie sich der Agent im freien Dialog verhält. Hierfür dienen die Aufzeichnungen zu Szenario acht[71]. Auch diese enthalten potenziell unvollständige bedingte Verzweigungen und mehrdeutige oder fehlende Korreferenzen. Zur Bemessung der Auflösung von Spracherkennungsunsicherheiten werden die Aufzeichnungen zu den Szenarien sechs und acht verwendet; die gewählte Aufzeichnung zu Szenario vier enthält schlicht keine Spracherkennungsunsicherheiten.

Um die Evaluation durchzuführen, wird der Dialog-Agent mit den Analyseergebnissen der anderen Pro*Nat*-Agenten konfrontiert. Für die (bereits vorhandenen) Aufnahmen zu den Szenarien vier und sechs bedeutet dies, dass die aufgezeichneten Analyseergebnisse (in Form von Pro*Nat*-Graphen) der jeweiligen Evaluationen verwendet werden (siehe Abschnitt 7.5.3 und Abschnitt 7.6.5). Im Fall von Szenario acht, beschreiben die Probanden zunächst ihre initialen Befehle, Pro*Nat*[72] analysiert die Äußerungen und der Dialog-Agent stellt Rückfragen, falls einer der implementierten Indikatoren anschlägt.

Um die Qualität des Dialog-Agenten möglichst realistisch bemessen zu können, wurden für alle Szenarien auch alle implementierten *Gesprächssituationen* verwendet und nicht nur die jeweils passenden[73]. Sie werden in folgender Reihenfolge zur Zuständigkeitskette hinzugefügt: *Spracherkennungsunsicherheit*, *mehrdeutige, unvollständige oder unsichere Korreferenz* und *unvollständige bedingte Verzweigung*. Wie zuvor beschrieben, kann die Verwendung der Zuständigkeitskette dazu führen, dass Gesprächssituationen zu Beginn der Kette, die erfolgreich abgeschlossen werden, spätere

[70] In die Transkriptionen wurden Satzzeichen injiziert, um den Studienteilnehmern das Textverständnis zu erleichtern.
[71] Die Szenariobeschreibung für das achte szenario befindet sich im Anhang in Abschnitt C.1.6.
[72] Es werden das vollständige Vorverarbeitungsfließband und alle zuvor beschriebenen Agenten verwendet.
[73] Dies kann gegebenenfalls dazu führen, dass Indikatoren anschlagen, obwohl die zugehörige Gesprächssituation gar nicht erforderlich wäre. Im praktischen Einsatz können potenzielle Fehler jedoch auch nicht vorab ausgeschlossen werden, weshalb immer alle Gesprächssituationen verwendet werden sollten.

Tabelle 7.41: Einfluss unterschiedlicher Schwellenwerte auf die Treffsicherheit des Indikators zur Erkennung von Spracherkennungsunsicherheiten.

Schwellenwert	0,70	0,75	0,80	0,85	0,90	0,95
Ausbeute	0,714	0,730	0,722	0,655	0,624	0,566
Präzision	0,476	0,548	0,619	0,679	0,750	0,821
F_1	0,571	0,626	0,667	0,667	0,681	0,670
Genauigkeit	0,847	0,860	0,868	0,855	0,850	0,827

Gesprächssituationen obsolet werden lassen. Dieses Verhalten ist zwar erwünscht, führt aber auch dazu, dass die Anzahl der initiierten Gesprächssituationen je Dialog-Aufzeichnung unterschiedlich ist. Dies gilt für alle Szenarien. Für die Gesprächssituation *Spracherkennungsunsicherheit* wurde der Einfluss des Schwellenwertes, sowie der Erfolg der geführten Dialoge bemessen. Bei den *mehrdeutigen, unvollständigen oder unsicheren Korreferenzen* wurde hingegen nur der Erfolg der Dialogführung bewertet; dasselbe gilt für *unvollständige bedingte Verzweigung*. Im Folgenden werden die Evaluationsergebnisse für die einzelnen Gesprächssituationen diskutiert.

Spracherkennungsunsicherheit

Für die Evaluation der Gesprächssituation *Spracherkennungsunsicherheit* werden Äußerungen aus den Szenarien sechs und acht evaluiert; die Äußerung aus Szenario vier wird nicht betrachtet, da diese keine Spracherkennungsunsicherheit enthält. Zur Evaluation wird zum einen der Schwellenwert systematisch variiert ([0,70; 0,95] mit einer Schrittweite von 0,05) und der Einfluss auf den erfolgreichen Abschluss des Dialogs bemessen. Zum anderen wird bemessen, wie viele Unsicherheiten aufgelöst (im Falle eines Wortfehlers) oder verifiziert werden konnten (falls das Wort zwar korrekt erkannt wurde, die Konfidenz aber unter dem Schwellenwert lag). Zusätzlich wird bestimmt, wie viele neue Fehler durch die Gesprächssituation hinzugefügt wurden und wie viele Dialogschritte, das bedeutet, Rückfragen des Agenten, nötig sind, um die Unsicherheit aufzulösen.

Um den Einfluss des Schwellenwertes als Indikator zu bemessen, wurden zunächst alle Wortfehler in den Äußerungen händisch annotiert. Anhand dieser Musterlösung können die Präzision, Ausbeute und das F_1-Maß und Genauigkeit für den Indikator bestimmt werden[74]. Die Ergebnisse sind in Tabelle 7.41 gelistet. Wie zu erwarten war, beeinflusst die Wahl des Schwellenwertes die Präzision und Ausbeute deutlich; wie üblich, steigt die Präzision bei einer höher gewählten Schwelle, gleichzeitig sinkt aber die Ausbeute. Die Genauigkeit hingegen verändert sich kaum und erreicht durchgängig hohe Werte (ca. 85%). Dieses Ergebnis ist insofern positiv zu bewerten, als in die Berechnung der Genauigkeit auch die *richtig negativen* einfließen (siehe Abschnitt 2.4.4). Für einen Indikator, der potenzielle Fehlerfälle anzeigen soll, ist eine hohe Anzahl *richtig negativer* Ergebnisse wünschenswert. Dementsprechend ist die Genauigkeit das entscheidende Auswahlkriterium für den Schwellenwert. Die beste Genauigkeit wird mit 0,868 bei einem Schwellenwert von 0,80 erreicht; dieser wird als Standardkonfiguration gewählt und auch in den nachfolgenden Evaluationsbetrachtungen verwendet.

[74] Ergebnisklassen: *richtig-positiv* – Indikator schlägt aufgrund des Schwellenwertes an und es handelt sich um einen Wortfehler; *falsch-positiv* – Indikator schlägt an, es handelt sich aber nicht um einen Wortfehler; *richtig-negativ* – Indikator schlägt nicht an und es handelt sich um keinen Wortfehler; *falsch-negativ*– Indikator schlägt nicht an, es handelt sich aber um einen Wortfehler

Tabelle 7.42: Evaluationsergebnisse für die Gesprächssituation *Spracherkennungsunsicherheit*.

	Szenario acht	Szenario sechs
# Gesprächssituationen begonnen	44	33
# Gesprächssituationen erfolgreich abgeschl.	11	12
# gestellte Fragen	80	58
# verifizierte Aussagen	6	11
# gelöste Fehler	4	1
# neu erzeugte Fehler	1	0
Fehlerrate	0,02	0,00
Erfolgsrate	0,25	0,36
Lösungsrate	0,23	0,36

Der Erfolg von initiierten Gesprächssituationen wird wiederum anhand der Äußerungen aus den Szenarien sechs und acht bemessen. Folgende Metriken werden verwendet: die Erfolgsrate (engl. *success rate*), welche angibt, wie viele Gesprächssituationen mit einer Änderung des $\texttt{Pro}\mathcal{Nat}$-Graphen abgeschlossen wurden (unabhängig davon, ob diese Änderung korrekt ist oder nicht), die Lösungsrate (engl. *resolution rate*), welche angibt, wie viele tatsächliche Fehler oder Unsicherheiten korrekt gelöst werden konnten und die Fehlerrate (engl. *error rate*), welche angibt, wie viele zusätzliche Fehler erzeugt wurden. Die erzielten Ergebnisse sind in Tabelle 7.42 aufgeführt.

In Summe liegen die Konfidenzen von 77 Wörtern unter dem Schwellenwert von 0,80; ebenso viele Gesprächssituationen wurden dementsprechend initiiert. Von diesen konnten 23 erfolgreich abgeschlossen werden, siebzehn führten zur Bestätigung eines bereits zuvor richtig erkannten Wortes, fünf zu einer Korrektur eines Wortfehlers. In Summe wurden vom Agenten 58 Fragen zu Spracherkennungsunsicherheiten in Szenario sechs gestellt, 80 in Szenario acht; das bedeutet, durchschnittlich 1,8 pro Gesprächssituation. Allerdings wurden alle erfolgreich abgeschlossenen Gesprächssituationen mit nur einer Rückfrage gelöst. Einerseits konnten so zwar nur 29% der vorhandenen potenziellen Fehlerstellen korrigiert oder verifiziert werden. Andererseits wurde auch nur ein zusätzlicher Fehler erzeugt. Diese Ergebnisse zeigen, wie schwierig es ist, Wortkorrekturen direkt zu erfragen. In einigen Fällen interpretierten die Probanden die Rückfragen des Agenten falsch und formulierten die gesamte Äußerung um, was dazu führt, dass die Antwort nicht an der ursprünglichen Äußerung ausgerichtet werden und der Dialog nicht erfolgreich beendet werden kann. In Zukunft sollte dementsprechend das Dialog-Management und die Antworterzeugung dieser Gesprächssituation überarbeitet werden, um die Erfolgsrate zu erhöhen.

Mehrdeutige, unvollständige oder unsichere Korreferenz

Diese Gesprächssituation wird anhand der Äußerung aus Szenario sechs evaluiert. Die Transkription enthält zwei uneindeutige Korreferenzen. Die erste ist eine Anapher, bestehend aus dem referenzierenden Ausdruck *you* und zwei potenziellen Referenten: einem vorangegangen Vorkommen von *you* und dem Eigennamen *Philip*. Letzteres ist lediglich ein durch den automatischen Spracherkenner erzeugter Wortfehler; eigentlich gemeint war an dieser Stelle *fill it*. Dementsprechend ist der korrekte Referent *you*. Diese Art von Fehler ist ein gutes Beispiel, um den Vorteil der Verwendung

Tabelle 7.43: Evaluationsergebnisse für die Gesprächssituation *mehrdeutige, unvollständige oder unsichere Korreferenz*.

	you → you	it → green cup	∑
Auflösungsquote	0,40	0,60	0,50
Ø Fragen	2,00	1,50	1,75

Tabelle 7.44: Evaluationsergebnisse für die Gesprächssituation *unvollständige bedingte Verzweigung*.

	Quote	Ø Fragen
Erfolg	1,00	1,10
Auflösung	0,30	1,33

der Zuständigkeitskette zu verdeutlichen. Wird der Wortfehler bereits durch die vorausgehende Gesprächssituation *Spracherkennungsunsicherheit* korrigiert, kann die Korreferenz bereits eindeutig aufgelöst werden und es muss kein zusätzlicher Dialog initiiert werden. Die zweite Korreferenz besteht aus dem referenzierenden Ausdruck *it* und den potenziellen Referenten *green cup*, *table* und *Armar*, wobei *green cup* die korrekte Lösung darstellt.

Die Ergebnisse dieser Evaluation sind in Tabelle 7.43 aufgeführt. Erneut wird jeweils die Lösungsrate angegeben. Für die erste mehrdeutige Korreferenz beträgt diese 40%[75]. Zur Auflösung wurden vom Agenten im Mittel zwei Rückfragen gestellt. Die Lösungsrate für die zweite Korreferenz beträgt sogar 60%; hier stellte der Agent im Schnitt lediglich 1,5 Rückfragen an die Probanden. Diese Ergebnisse sind als gut zu bewerten, da es sich bei der Auflösung von Korreferenzen um einen komplexen Sachverhalt handelt, der nur schwierig im Rahmen eines Dialogs vermittelt werden kann.

Unvollständige bedingte Verzweigung

Die Gesprächssituation *unvollständige bedingte Verzweigung* wird anhand einer Äußerung aus Szenario vier evaluiert. Diese enthält eine unvollständige Verzweigungsstruktur; der Agent zur Erkennung von bedingte Verzweigungen hat lediglich die Bedingung erkannt, der zugehörige *Dann*-Gliedsatz konnte nicht ermittelt werden.

Tabelle 7.44 zeigt die erzielten Ergebnisse; es werden dieselben Metriken wie zuvor verwendet. Die Erfolgsrate von 100% zeigt, dass die Probanden alle Rückfragen des Agenten beantworten und alle Gesprächssituationen somit erfolgreich abgeschlossen werden konnten. Allerdings führten nur 30% der Antworten zu einer korrekten Lösung der Situation. Im Großteil der übrigen Fälle wiederholten die Probanden die vollständige Kontrollstruktur in ihren Antworten. In der Folge wurde die komplette Antwort vom Agenten als *Dann*-Gliedsatz in den ProNat-Graphen eingesetzt. Dieser Ausgang ist unerwünscht, kann zukünftig aber durch ein verbessertes Dialog-Management für diese Gesprächssituation vermieden werden; beispielsweise könnte beim Ausrichten der Antwort an die ursprüngliche Äußerung der Konditionalsatz ausgeschlossen werden. Auch wenn die Erfolgsrate für diese Gesprächssituation schlechter ausfällt, so konnte doch gezeigt werden, dass der Ansatz erfolgversprechend ist, immerhin handelt sich um eine programmatische Struktur, die im natürlichsprachlichen Dialog mit Laien vervollständigt werden musste.

[75] In zehn Dialogen konnten insgesamt acht von zwanzig fehlerhaften Korreferenzen korrigiert werden.

7.11 Funktionswächter-Agenten

Die Architektur PARSE sieht vor, dass Agenten innerhalb der Einheit zur Analyse der Semantik parallel und wiederholt ausgeführt werden (siehe Abschnitt 4.2.4). Das bedeutet, die Analysen der in den vorangegangen Abschnitten dieses Kapitels beschriebenen Agenten werden theoretisch beliebig oft wiederholt. Damit die semantische Analyse dennoch zu einem endgültigen Ergebnis gelangt, muss die Verarbeitung durch die Agenten kontrolliert beendet werden. Die Rahmenarchitektur Luna, die auch für die Ausführung der Agenten zuständig ist, verfügt über einen Mechanismus, um den Agenten mitzuteilen, dass sie ihre Ausführung beenden sollen. Hierzu versendet Luna ein *Terminierungs*-Event an alle Agenten (siehe Abschnitt 4.2.5). Neben diesem Mechanismus werden Kriterien benötigt, anhand derer entschieden wird, wann die Verarbeitung beendet wird. Statt Abbruchkriterien direkt innerhalb der Rahmenarchitektur zu hinterlegen, wird auch hierfür der agentenbasierte Ansatz verfolgt. PARSE ermöglicht die Verwendung sogenannter *Funktionswächter-Agenten* (engl. *watchdog agent*). Diese entsprechen in ihrer Funktionsweise normalen Agenten; allerdings können die Funktionswächter-Agenten einen speziellen Knotentypen erzeugen[76]. Das Vorhandensein des Knotentyps signalisiert der Rahmenarchitektur, dass ein *Terminierungs*-Event ausgelöst werden soll. Durch diesen Mechanismus können konkrete Funktionswächter-Agenten dafür genutzt werden, jeweils unabhängig voneinander ein bestimmtes Abbruchkriterium zu prüfen; sobald das Kriterium erfüllt ist, erzeugt der Agent den Knotentypen. Die Verwendung von Funktionswächter-Agenten erlaubt es, beliebige Kombinationen von Abbruchbedingungen zu verwenden.

Prinzipiell können in den Agenten beliebige Abbruchbedingungen implementiert werden. Es bietet sich jedoch an, Abbruchbedingungen zu definieren, die vom Zustand des ProNat-Graphen abhängen, da dieser den aktuellen Fortschritt bei der Analyse der Semantik darstellt. Für ProNat wurden zunächst drei Funktionswächter-Agenten entwickelt, die folgende Kriterien prüfen:

- *Zeitüberschreitung:* Der Agent prüft, ob die Gesamtausführungszeit von ProNat eine konfigurierbare Schranke überschreitet. Es empfiehlt sich, diesen Funktionswächter-Agenten in jeder Konfiguration zu verwenden, damit die Terminierung der Einheit zur Analyse der Semantik sichergestellt ist. Außerdem erzeugt die Berechnung des Abbruchkriteriums nur einen geringen Mehraufwand.
- *Keine Änderung:* Der Agent prüft, wie viel Zeit seit der letzten Änderung des ProNat-Graphen vergangen ist. Überschreitet dieser Wert eine konfigurierbare Schranke, wird der Knotentyp zur Signalisierung der Terminierung erzeugt. Die Schranke dieses Agenten muss sorgfältig gewählt werden (und gegebenenfalls auf die ausführende Plattform angepasst werden). Einerseits sollte die Schranke größer als die maximale Ausführungszeit des Agenten mit der längsten Ausführungszeit gewählt werden, da andernfalls der entsprechende Agent seine Analysen nie vollständig durchführen kann[77]. Anderseits sollte die Schranke kleiner als

[76] Der zu verwendende Knotentyp wird vorab per Konfiguration systemweit festgelegt.

[77] Die Ausführungszeit eines Agenten kann abgeschätzt werden, indem dieser separat mit üblichen bzw. erwartbaren Eingaben (das heißt natürlichsprachlichen Äußerungen) ausgeführt wird.

die Schranke des Funktionswächter-Agenten zur Prüfung einer Zeitüberschreitung gewählt werden (falls dieser verwendet wird), da andernfalls kein Mehrwert gegenüber diesem (einfacheren) Abbruchkriterium entsteht.

- *Zyklische Änderung:* Der Agent prüft, ob die Änderungen, welche die Agenten im ProNat-Graphen hinterlegen, zyklisch sind[78]. Durch zyklische Änderungen wird ein Pseudo-Fortschritt erzeugt, da sich der Graph kontinuierlich ändert, allerdings keine neuen Informationen hinterlegt werden. Zur Erkennung von zyklischen Änderungen vergleicht der Agent die *n* letzten Graph-Zustände, wobei *n* frei konfiguriert werden kann. Sind zwei der letzten *n* Graphen identisch[79], liegt ein Zyklus vor und der Knotentyp zur Signalisierung der Terminierung wird erzeugt. Dementsprechend sollte *n* größer gleich drei gewählt werden, da mindestens der Vergleich der letzten drei Graph-Zustände nötig ist, um einen Zyklus zu erkennen.

Wird eines der Kriterien erfüllt, erzeugt der Agent den Knotentypen, der die Beendigung signalisiert. Durch diese Änderung im Graphen wird dieser auch als aktueller Graph in der Rahmenarchitektur hinterlegt. Diese erkennt, dass der signalisierende Knotentyp enthalten ist und versendet wiederum das *Terminierungs*-Events an alle Agenten. Diese beenden daraufhin eigenständig ihre Ausführung. Sobald alle Agenten beendet wurden, wird der ProNat-Graph an die erste Stufe des Nachverarbeitungsfließbands weitergereicht und die Nachverarbeitung begonnen.

[78] Beispielsweise könnte ein Agent *A* eine neue Kante im Graphen hinterlegen. Ein Agent *B* entfernt ebendiese Kante aus dem Graphen, *A* fügt diese im nächsten Aufrufzyklus erneut hinzu und so weiter.

[79] Aus Effizienzgründen erfolgt der Vergleich der Graphen anhand ihres Hash-Codes. Dieser wird analog zur Gleichheit berechnet (siehe Abschnitt 4.2.2.2).

8 Nachverarbeitung

„Großes wird durch eine Reihe von kleinen Dingen erreicht,
die zusammenkommen.“

– Vincent Van Gogh

Mit der Beendigung der Agenten durch ein Signal eines Wächteragenten beginnt die Nachverarbeitungsphase von ProNat. Wie in Abschnitt 4.2.3 erläutert, sieht PARSE für die Nachverarbeitung die Verwendung eines Fließbandes vor. Die Stufen sollen dazu genutzt werden, die Analyseergebnisse, das heißt den Graphen, zu interpretieren. Für ProNat bedeutet das, der Graph wird als Programm interpretiert und es soll Quelltext für ein Zielsystem erzeugt werden. Zur Nachverarbeitung des ProNat-Graphen wurden vier Fließbandstufen entwickelt, die den Graphen schrittweise in ausführbaren Quelltext überführen:

1. Generierung eines abstrakten Syntaxbaumes (Abschnitt 8.1)
2. Extraktion des abstrakten Syntaxbaumes (Abschnitt 8.2)
3. Erzeugung von Quelltext (Abschnitt 8.3)
4. Injektion des Quelltexts in Schablonen (Abschnitt 8.4)

Zunächst wird ein abstrakter Syntaxbaum (engl. *abstract syntax tree*, kurz *AST*) [Aho+06] anhand der im Graphen enthaltenen Informationen generiert und in den ProNat-Graphen eingepflegt. Anschließend wird der abstrakte Syntaxbaum aus dem Graphen ausgelesen[1]. In der dritten Stufe wird Quelltext in einer konkreten Programmiersprache erzeugt und das Generat schließlich in der vierten Stufe in eine Zielsystem-spezifische Schablone eingefügt.

Die Aufgliederung der Nachverarbeitung in diese vier Schritte hat den Vorteil, dass das Hinzufügen neuer Zielsysteme mit geringem Aufwand verbunden ist. Nur die letzte Stufe hängt direkt vom Zielsystem ab; für jedes Zielsystem muss vorab eine Schablone definiert werden. Die Generierung (und Extraktion) des AST ist völlig unabhängig von der verwendeten Programmiersprache und dem Zielsystem. Auch für die dritte Stufe muss nur je Programmiersprache ein Generator bereitgestellt werden.

[1] Durch die Trennung von Generierung und Extraktion des AST kann die erste Fließbandstufe auch verwendet werden, falls anschließend in einer zusätzlichen Stufe weitere Analysen anhand des vollständigen Graphen durchgeführt werden sollen.

8.1 Generierung des abstrakten Syntaxbaumes

In der ersten Stufe des Nachverarbeitungsfließbandes soll ein abstrakter Syntaxbaum generiert und in den ProNat-Graphen integriert werden. Die Generierung erfolgt anhand der durch die Agenten erzeugten Informationen (siehe Kapitel 7). Für ProNat werden abstrakte Syntaxbäume als Zwischenrepräsentation genutzt, da diese eine universelle Darstellung des Programms bieten, die unabhängig von der verwendeten Programmiersprache und dem Zielsystem ist.

Nur zwei der in Kapitel 3 diskutierten verwandten Ansätze thematisieren abstrakte Syntaxbäume explizit. Rabinovich et al. entwerfen sogenannte *abstrakte Syntaxnetze*, die konzeptionell an abstrakte Syntaxbäume angelehnt sind [RSK17]. Ihr Ansatz sieht vor, ein *LSTM* so zu untergliedern, dass für jede Teilstruktur eines abstrakten Syntaxbaumes ein eigenes Dekodierer-Modul verwendet wird. Yin und Neubig generieren hingegen gezielt abstrakte Syntaxbäume, indem sie eine probabilistische Grammatik (zur Beschreibung der Programmkonstrukte) und ein neuronales Netz (zur Analyse der natürlichen Sprache) kombinieren [YN17]. Beide Ansätze verwenden probabilistische Ansätze, um abstrakte Syntaxbäume zu erzeugen. Für ProNat wird hingegen ein regelbasiertes, generatives Vorgehen umgesetzt. Der Grund hierfür ist, dass viele AST-Bestandteile (verzweigte Bedingungen, Schleifen, Methodenaufrufe etc.) bereits als baumartige Strukturen im ProNat-Graphen hinterlegt sind. Diese müssen nur noch auf geeignete Weise zusammengeführt werden, um einen abstrakten Syntaxbaum zu bilden.

Die Fließbandstufe erzeugt den abstrakten Syntaxbaum in drei Schritten. Zunächst wird der ProNat-Graph auf Strukturen untersucht, die sich in AST-Bestandteile überführen lassen; die Überführung erfolgt Muster-basiert (ähnlich dem Vorgehen des Dialog-Agenten, wie in Abschnitt 7.10 beschrieben) und ist dementsprechend erweiterbar. Durch die Mustersuche entstehen Teilstrukturen (bzw. Teilbäume), die Schleifen, Bedingungen, Aufrufe usw. repräsentieren. Im zweiten Schritt werden diese Strukturen zu einem AST verbunden. Da der so entstandene AST die Ausführungsreihenfolge noch nicht berücksichtigt, werden die Knoten im dritten Schritt, gemäß der Reihenfolge der Aussagen in der natürlichsprachlichen Äußerung, geordnet.

8.1.1 Erkennung von Teilstrukturen

Im ersten Schritt werden Teilstrukturen von abstrakten Syntaxbäumen generiert. Dabei handelt es sich nicht nur um einzelne Knoten, sondern in den meisten Fällen um Teilbäume. Folgende Strukturen sollen erzeugt werden:

- Anweisungen (inkl. Methodenaufrufen)
- Blöcke
- Bedingte Verzweigungen
- Schleifen (*For*, *While* und *Do-While*)
- Parallele Abschnitte (Nebenläufigkeit)

Diese enthalten wiederum weitere Bestandteile, wie Deklarationen, Ausdrücke, Komparatoren etc. Zusätzlich wird für jede Äußerung ein Wurzelknoten erzeugt; in diesem wird auch hinterlegt, ob es sich bei der Äußerung um eine Methodendefinition oder ein Skript handelt (siehe Abschnitt 7.8).

Die oben genannten Strukturen werden mithilfe sogenannter *Strukturerkenner* generiert. Ein Strukturerkenner besteht aus einem Graphmuster und einer zu erzeugenden Zielstruktur, die jedes Mal erzeugt wird, wenn das Muster im Graphen detektiert wird. Etwaige aktuelle Werte werden in der Zielstruktur mithilfe von Schablonen hinterlegt. Die Strukturerkenner sind in einer Zustandigkeitskette (engl. *chain of responsibility*) organisiert [Gam+94]. Die Menge der Strukturerkenner (und damit auch der erkennbaren Teilstrukturen) ist erweiterbar. Zudem kann konfiguriert werden, welche Strukturerkenner zur Zuständigkeitskette hinzugefügt und somit zur Analyse verwendet werden[2].

Beispiel:
Erzeugung der AST-Teilstruktur *Bedingte Verzweigung*

In diesem Beispiel soll veranschaulicht werden, wie anhand eines Graphmusters die Teilstruktur *Bedingte Verzweigung* erzeugt wird. Das Beispiel nimmt Bezug auf das in Abbildung 8.1 dargestellte Schema.

Natürlichsprachliche Sequenz

Robo if the cup is dirty wash it

Erläuterung

Die Knoten vom Typ `Token` des `Pro`*Nat*-Graphen enthalten unter anderem ein Attribut, das angibt, ob das jeweilige Wort Bestandteil einer verbalisierten bedingten Verzweigung ist. Token mit dem Attributwert *IF* gehören zum Konditional-, solche mit dem Wert *THEN* zum *Dann*- und die mit dem Wert *ELSE* zum *Andernfalls*-Gliedsatz (siehe Abschnitt 7.6). Der Strukturerkenner zur Erzeugung von bedingten Verzweigungen reagiert auf eine zusammenhängende Folge von Token, welche die obenstehenden Attributwerte tragen, wobei mindestens ein *IF* und ein *THEN* oder *ELSE* enthalten sein muss. Da dieses Muster im Beispiel erkannt wird, erzeugt der Strukturerkenner die entsprechende AST-Teilstruktur. Zusätzlich wird hinterlegt, welche Token (der natürlichsprachlichen Äußerung) die Teilstruktur umfasst.

8.1.2 Verbindung der Teilstrukturen

Im zweiten Schritt werden die bisher unzusammenhängenden Teilstrukturen zu einem Baum verbunden. Hierzu wird zunächst für jedes Token der natürlichsprachlichen Äußerung ein Pfad aufgebaut, der alle Teilstrukturen beinhaltet, in denen das Token enthalten ist. Die Teilstrukturen werden dabei gemäß ihrer Größe (Anzahl umspannter Token) sortiert. Jeder Pfad beginnt mit einem

[2] In der Standardkonfiguration werden alle Strukturerkenner für die zuvor genannten Teilstrukturen verwendet.

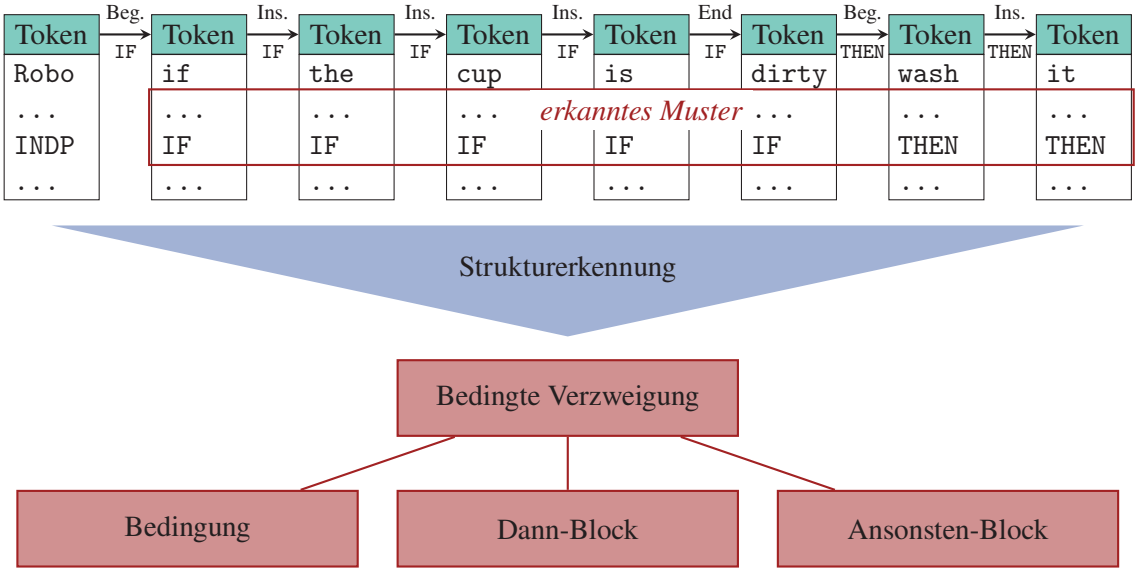
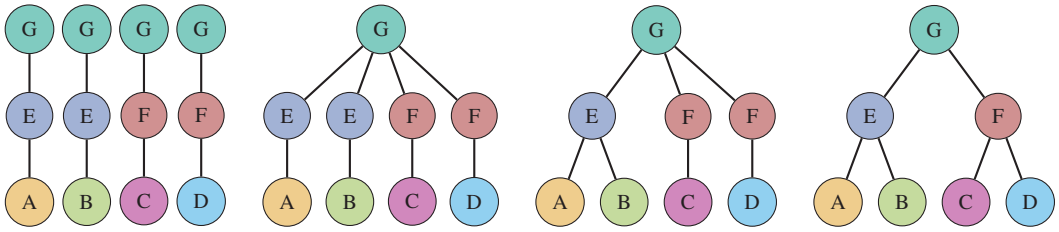


Abbildung 8.1: Erzeugung der Teilstruktur *Bedingte Verzweigung* anhand eines Graphmusters.

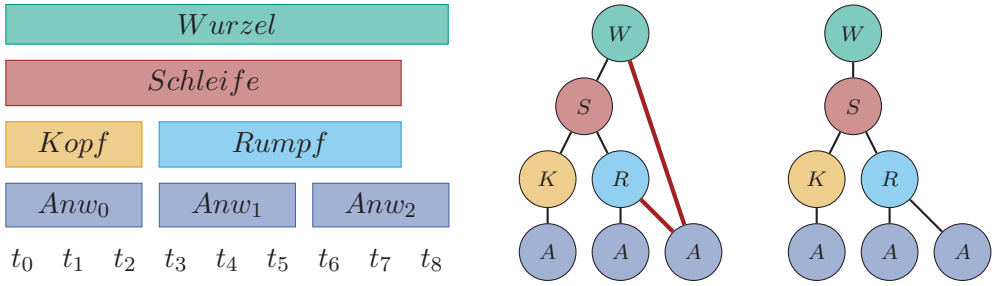


(a) Schritt 1. **(b)** Schritt 2. **(c)** Schritt 3. **(d)** Schritt 4.

Abbildung 8.2: Schrittweise Verschmelzung gleicher AST-Knoten: Die Verschmelzung wird ausgehend von der Wurzel zu den Blättern durchgeführt.

Token und endet beim Wurzelknoten. Für die Äußerung *if the cup is dirty wash it* könnte der Pfad des Tokens *wash* beispielsweise wie folgt lauten: $Token \rightarrow Anweisung/Methodenaufruf \rightarrow BedingteVerzweigung \rightarrow Wurzel$.

Alle Pfade werden anschließend schrittweise von oben nach unten (engl. *top-down*) verschmolzen, indem gleiche Knoten zusammengefasst werden. Das Ergebnis ist ein abstrakter Syntaxbaum, dessen Knotensortierung noch nicht der Reihenfolge in der natürlichsprachlichen Äußerung entspricht. Der Verschmelzungsprozess ist schematisch in Abbildung 8.2 dargestellt. Bei der Verschmelzung können Mehrdeutigkeiten auftreten; nicht in allen Fällen kann der Elter-Knoten eines Knotens eindeutig bestimmt werden. Beispielsweise kann eine Anweisung nur teilweise innerhalb eines Schleifenkörpers liegen. Derartige Ambiguitäten können auftreten, da Anweisungen (bzw. Methodenaufrufe) und Kontrollstrukturen unabhängig voneinander von unterschiedlichen Agenten erzeugt werden. Für die Erzeugung des abstrakten Syntaxbaumes hätte eine solche Situation zur Folge, dass der Knoten, der die Anweisung darstellt, zwei unterschiedliche Elter-Knoten hätte. Da der Baum dadurch nicht mehr minimal zusammenhängend wäre, muss einer der Pfade gewählt und die andere Kante entfernt werden. Um die Auswahl zu treffen, wird jeweils die Schnittmenge der überspannten



(a) Token und Teilstrukturen. **(b)** mehrd. AST. **(c)** erz. AST.

Abbildung 8.3: Auflösung einer mehrdeutigen Elter-Kind-Beziehung bei der Erzeugung des AST: Die Teilabbildung (a) zeigt die Ausgangssituation mit den Token t_0 bis t_8 und den zugehörigen Teilstrukturen. In Teilabbildung (b) ist der mehrdeutige AST dargestellt, der entsteht, da Anw_2 nur teilweise dem Rumpf der Schleife zugeordnet wurde. Die Teilabbildung (c) zeigt den letztlich erzeugten AST. Gemäß Gleichung 8.1 erhält der Knoten R die höhere Bewertung gegenüber W und wird daher ausgewählt ($A(Anw_2, Rumpf) = |\{t_6, t_7\}|/|\{t_3, ..., t_7\}| = 2/5 > A(Anw_2, Wurzel) = |\{t_6, t_7, t_8\}|/|\{t_1, ..., t_8\}| = 3/9$).

Token der übergeordneten Teilstrukturen (Elter-Knoten) mit der untergeordneten Struktur (Kind-Knoten) gebildet. Die Mächtigkeit der Mengen wird jeweils mit der Mächtigkeit der Token-Mengen der übergeordneten Teilstrukturen normiert. Die Auswahlbewertung A für Kombination eines Kind-Knoten k und mit einem Elter-Knoten e berechnet sich dementsprechend wie folgt:

$$A(k, e) = \frac{|T_k \cap T_e|}{|T_e|} \tag{8.1}$$

Dabei entspricht T_k der Menge der überspannten Token der untergeordneten Teilstruktur und T_e der entsprechenden Menge der übergeordneten Struktur. Die Auswahl eines Elter-Knotens in einer uneindeutigen Hierarchie ist in Abbildung 8.3 schematisch dargestellt. In dem gezeigten Beispiel muss entschieden werden, ob die dritte Anweisung (Anw_2) dem Schleifenrumpf (*Rumpf*) zugeordnet oder direkt unter dem Wurzelknoten ($Wurzel$) eingehängt wird.

8.1.3 Sortierung der Teilbäume

Die erzeugten Syntaxbäume sind bisher noch unsortiert. Die Reihenfolge der Knoten ist zufällig und entspricht damit insbesondere nicht der Reihenfolge der Aussagen der natürlichsprachlichen Äußerung. Diese Sortierung wird hergestellt, indem ausgehend von den Token die Position im Baum nach oben propagiert wird. Ein Elter-Knoten e erhält als Position P die mittlere Position seiner Kind-Knoten k_i zugewiesen (aus der Memge der Kind-Knoten K):

$$P(e, K) = \frac{\sum_{i=1}^{|K|} P(k_i)}{|K|} \tag{8.2}$$

Mit der Sortierung der Knoten ist die Generierung des abstrakten Syntaxbaumes abgeschlossen. Zuletzt wird der Syntaxbaum in den ProNat-Graphen eingegliedert. Hierzu werden zunächst für

alle Knoten des Syntaxbaums äquivalenten Knoten im ProNat-Graphen erzeugt[3] und anschließend alle Knoten vom Typ `Token` mit ihren jeweiligen direkten Elter-Knoten verbunden.

8.2 Extraktion des Syntaxbaumes

Die zweite Fließbandstufe des Nachverarbeitungsfließbandes von ProNat ist dafür zuständig, den zuvor erzeugten abstrakten Syntaxbaum aus dem Graphen zu extrahieren (siehe Abschnitt 8.1). Da die Knoten- und Kantentypen der AST-Bestandteile im Graphen bekannt sind, kann der AST extrahiert werden, indem beginnend bei der Wurzel die jeweiligen Kanten verfolgt werden. Der extrahierte AST wird anschließend in eine interne Graph-Darstellung überführt, die nur Informationen enthält, die zur Erzeugung von Quelltext erforderlich sind; das bedeutet, an dieser Stelle findet eine Reduktion der vollständigen Analyseergebnisse auf einen abstrakten Syntaxbaum statt.

8.3 Erzeugung von Quelltext

Die Fließbandstufe zur Erzeugung von Quelltext nimmt den zuvor generierten AST entgegen und transformiert diesen in Quelltext einer konkreten Programmiersprache. Abstrakte Syntaxbäume stellen eine hierarchische Repräsentation von Quelltext dar [Aho+06]. Daher kann Quelltext aus einem AST erzeugt werden, indem die Knoten des Baums ausgehend von der Wurzel über die Kanten verfolgt werden. Um die Erzeugung von Quelltext für verschiedene Programmiersprachen zu ermöglichen, wird das Entwurfsmuster *Besucher* (engl. *visitor*) [Gam+94] verwendet[4]. Ein Besucher sucht einen Knoten des AST auf und setzt das jeweilige abstrakte Programmkonstrukt in ein konkretes Konstrukt der entsprechenden Programmiersprache um. Da die meisten Programmiersprachen Programmkonstrukte syntaktisch unterschiedlich umsetzen, wird je Programmiersprache ein Besucher benötigt. Für ProNat wurden Besucher für folgende Programmiersprachen umgesetzt:

- Java
- C
- Python
- PlantUML[5] (domänenspezifische Sprache (engl. *domain-specific language*, kurz *DSL*) zur Erzeugung von *UML*-Diagrammen[6])
- PseudoCode (Pseudo-Quelltext)

[3] Je AST-Bestandteil (also Wurzel, Schleife, Schleifenkopf usw.) wird ein neuer Knotentyp erzeugt; alle Knotentypen haben das Präfix `AST`. Zudem werden unterschiedliche Kantentypen (ebenfalls mit Präfix `AST`) erzeugt, je nachdem, ob es sich um Kanten innerhalb einer Teilstruktur, zwischen Teilstrukturen oder zu Token handelt.

[4] Viele Kompilierer verwenden das Besucher-Muster, um aus abstrakten Syntaxbäumen (die zuvor aus konkretem Quelltext generiert wurden) Zwischen- oder Maschinensprache zu generieren (siehe z. B. [NCM03]). Für ProNat wurde dieser Ansatz umgekehrt, um aus einem AST konkreten Quelltext zu generieren.

[5] *PlantUML*: `https://plantuml.com/`, zuletzt besucht am 24.02.2021.

[6] Im Kontext von ProNat wird *PlantUML* zur Erzeugung von *UML*-Aktivitätsdiagrammen verwendet.

Die Auswahl der Sprachen erfolgte anhand folgender Überlegungen: Zunächst sollten populäre Programmiersprachen unterstützt werden. Daher wurden mit *Java*, *C* und *Python* die drei derzeit populärsten Programmiersprachen nach dem *TIOBE*-Index ausgewählt[7]. Durch diese Wahl werden zudem unterschiedliche Paradigmen abgedeckt: *C* ist eine imperative, prozedurale Programmiersprache, *Java* unter anderem Objekt-orientiert und Python zudem funktional. Somit kann gezeigt werden, dass ProNat Quelltext für unterschiedliche Typen von Programmiersprachen generieren kann. Zusätzlich wurde mit *PlantUML* eine domänenspezifische Sprache gewählt, mit der *UML*-Diagramme generiert werden können. Im Kontext von ProNat wird *PlantUML* verwendet, um Aktivitätsdiagramme zu generieren[8]. Zuletzt kann mithilfe eines weiteren Besuchers Pseudo-Quelltext generiert werden. Dadurch kann die Qualität des durch ProNat erzeugten Generats unabhängig von den Eigenheiten der jeweiligen Programmiersprachen bewertet werden.

Die Besucher können für unterschiedliche Zielsysteme verwendet werden. Ein neuer Besucher muss nur hinzugefügt werden, falls ein Zielsystem angebunden werden soll, das eine bisher nicht betrachtete Programmiersprache verwendet[9].

Beispiel:
Quelltextgenerierung für unterschiedliche Programmiersprachen

In diesem Beispiel soll die Quelltextgenerierung für unterschiedliche Programmiersprachen (hier *Java* und *C*) veranschaulicht werden. Das Beispiel nimmt Bezug auf Abbildung 8.4, welche die Generate für beide Programmiersprachen zur nachfolgenden natürlichsprachlichen Sequenz darstellt.

Natürlichsprachliche Sequenz

hey armar check if the dish is dirty then wash the dish twice after that get me orange juice while reading the news for me

Erläuterung

Das Beispiel enthält eine bedingte Verzweigung, eine Zählschleife, sowie zwei parallele Abschnitte. Die Verzweigung und die Zählschleife werden in beiden Programmiersprachen syntaktisch identisch umgesetzt. Die Quelltextabschnitte zur Umsetzung der parallelen Abschnitte unterscheiden sich hingegen deutlich. Während in *Java* Fäden (engl. *Thread*) erzeugt, gestartet und nach der Abarbeitung des Abschnitts zusammengeführt werden, genügt in *C* die Definition sogenannter *parallel sections* (bei Verwendung von *OpenMP* [Cha+01]).

[7] *TIOBE*-Index: `https://www.tiobe.com/tiobe-index/`, zuletzt besucht am 24.02.2021.

[8] Aktivitätsdiagramme eignen sich zur Darstellung von Abläufen und sind auch für Laien verständlich. Dadurch eignen sie sich, um ProNat in einer Nutzerstudie zu evaluieren (siehe Abschnitt 9.2). Zudem gibt die Erzeugung von Aktivitätsdiagrammen einen Ausblick darauf, wie ProNat abseits von reiner Quelltexterzeugung verwendet werden kann (siehe hierzu Abschnitt 10.2)

[9] Der Aufwand bei der Erstellung eines Besuchers für eine neue Programmiersprache ist gering. Die bisher entwickelten Besucher haben einen Umfang von lediglich 175 bis 245 Quelltextzeilen.

```java
if (isDirty(Dishes)) {
    for(int i = 0; i < 2; i++)
        {
        wash(Dishes);
    }
} else { }

Thread t0 = new Thread() {
    public void run() {
        get1(OrangeJuice);
    }
};

t0.start();

Thread t1 = new Thread() {
    public void run() {
        read(News);
    }
};

t1.start();
t0.join();
t1.join();
```

(a) Java.

```c
if (isDirty(Dishes)) {
    for(int i = 0; i < 2; i++)
        {
        wash(Dishes);
    }
} else { }

#pragma omp parallel sections
{

    #pragma omp section
    {
        get1(OrangeJuice);
    }

    #pragma omp section
    {
        read(News);
    }
}
```

(b) C.

Abbildung 8.4: Zwei Quelltextgenerate für die Programmiersprachen *Java* (a) und *C* (b).

8.4 Injektion des Quelltext in Schablonen

Zuletzt muss der generierte Quelltext noch auf einem Zielsystem ausführbar gemacht werden. Die letzte Stufe des Nachverarbeitungsfließbands von ProNat setzt diese Aufgabe um, indem der Quelltext in Quelltextschablonen (engl. *code templates*) injiziert wird[10]. Die verwendeten Quelltextschablonen bestehen aus Zielsystem-spezifischen Implementierungsdetails, beispielsweise zur Umsetzung eines Einschubs (engl. *Plug-In*), und Platzhaltern[11]. Als Platzhalter sind folgende Felder definiert:

- `{{ header }}`: Dateikopf (z. B. Modul- oder Klassenimporte)
- `{{ classHead }}`: Modul- oder Klassenkopf (z. B. Klassenattribute)
- `{{ newMethod }}`: private Methoden, von ProNat generiert (Abschnitt 7.8)
- `{{ topComment }}`: Hauptkommentarfeld
- `{{ code }}`: generierter Quelltext

[10] Die Verwendung von Quelltextschablonen ist nur eine Möglichkeit, um den Quelltext ausführbar zu machen. Alternativ könnten beispielsweise auch Web-API-Aufrufe generiert werden. Hierzu müsste nur eine neue letzte Fließbandstufe implementiert werden; das übrige ProNat-System könnte trotz dieses Paradigmenwechsels unverändert verwendet werden.

[11] Zum Befüllen der Schablonen wird das Werkzeug *jtwig* verwendet: `https://github.com/jtwig/jtwig`, zuletzt besucht am 24.02.2021.

```
// {{ header }}

class MyLegoRobot implements ISystem  {

// {{ classHead }}

    public static void main(String[] args) {
        ISystem sys = new LegoRobot();
        try {
            sys.execute();
        } catch (Exception e) {
            System.err.println("Error");
        }
        sys.close();
    }

    @Override
    public void execute(){
    // {{ topComment }}
    // {{ code }}
    }
    // {{newMethod}}
}
```

(a) Java (*Lego Mindstorms*-API).

```
@startuml

#HotPink:
'{{ topComment }}
;
detach

start

'{{ code }}

stop

'{{ newMethod }}

@enduml
```

(b) PlantUML.

Abbildung 8.5: Zwei Schablonen zur Quelltextinjektion: Die Schablone in Teilabbildung (a) kann verwendet werden, um einen *Lego Mindstorms*-Roboter über eine *Java*-API zu steuern; die Schablone in Teilabbildung (b) dient zur Erzeugung von PlantUML-Quelltext.

Für jedes anzubindende Zielsystem muss eine Schablone bereitgestellt werden. Der Inhalt der Schablonen kann beliebig gestaltet werden; die Schablonen können, müssen aber nicht alle zuvor genannten Platzhalter enthalten. Außerdem können auch zusätzliche Informationen kodiert werden, wie beispielsweise Zielsystem-interne Darstellungen (bzw. Implementierungen) der Umgebungsobjekte. Abbildung 8.5 zeigt zwei für ProNat erstellte Schablonen.

Die Fließbandstufe sorgt dafür, dass der zuvor erzeugte Quelltext (und gegebenenfalls die weiteren Bestandteile) an den jeweiligen Stellen eingefügt werden[12]. Dadurch entsteht ausführbarer Quelltext für ein Zielsystem. Mit der letzten Nachverarbeitungsfließbandstufe endet die Verarbeitung durch ProNat. Die Erzeugung eines Programms aus einer gesprochenen Äußerung ist damit abgeschlossen.

[12] Für welches Zielsystem Quelltext generiert werden soll, das heißt welche Schablone verwendet wird, kann konfiguriert werden. Die Standardkonfiguration sieht die Erzeugung von Pseudoquelltext in einer einfachen Umgebungsschablone vor.

9 Evaluation des Gesamtsystems

„Die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt.“

– Ludwig Wittgenstein

Für das System ProNat wurde in Kapitel 5 die Zielstellung formuliert, die Endnutzer-Programmierung mit gesprochener Sprache zu ermöglichen. Nachdem in den anschließenden Kapiteln die einzelnen Systembestandteile zur Umsetzung dieses Ziels beschrieben wurden, gilt es die Leistungsfähigkeit des Gesamtsystems zu bewerten. In diesem Kapitel soll folglich analysiert werden, wie gut es ProNat gelingt, aus natürlichsprachlichen Anweisungen Quelltext zu generieren.

In den vorangegangen Kapiteln wurden bereits einzelne Bestandteile von ProNat evaluiert. Diese Evaluationen geben einerseits Aufschluss darüber, welche Sprachanalysen bereits zufriedenstellende Ergebnisse liefern und welche zukünftig verbessert werden sollten, um die Performanz des Gesamtsystems zu steigern. Andererseits treffen einige Evaluationen auch bereits Aussagen zur Leistungsfähigkeit des Gesamtsystems hinsichtlich ausgewählter Aspekte. Die Evaluationen der Agenten zur Synthese von Kontrollstrukturen zeigen bereits, wie gut ProNat Kontrollstrukturen aus natürlichsprachlichen Äußerungen synthetisieren kann (siehe Abschnitt 7.6.5). Dasselbe gilt für den Agenten zur Synthese von Methodendefinitionen und Skripten (siehe Abschnitt 7.8); dieser erzeugt nicht nur Methodensignaturen und -rümpfe bzw. Skripte, sondern bildet zusätzlich auch natürlichsprachliche Anweisungen auf Zielsystemfunktionen ab. Das bedeutet, an dieser Stelle entstehen bereits Skripte bzw. Methoden, bestehend aus Pseudo-Quelltext. Dementsprechend zeigt die zugehörige Evaluation bereits, wie gut ProNat Anweisungen (bzw. API-Aufrufe) synthetisieren kann. Zuletzt konnte im Zuge der Beschreibung des Nachverarbeitungsfließbandes bereits exemplarisch gezeigt werden, dass Quelltext in unterschiedlichen Programmiersprachen für verschiedene Anwendungsdomänen generiert werden kann (siehe Kapitel 8).

Anhand der Evaluationen zu den Einzelbestandteilen kann jedoch keine Aussage darüber getroffen werden, wie gut das Zusammenspiel der Komponenten funktioniert. In diesem Kapitel soll daher die Leistungsfähigkeit des Gesamtsystems bemessen werden. Hierzu werden verschiedene Evaluationen durchgeführt, die ProNat jeweils aus unterschiedlichen Blickwinkeln bewerten bzw. unterschiedliche Teilaspekte evaluieren. Alle Evaluationen werden anhand von manuellen Transkriptionen durchgeführt. Dadurch wird der Einfluss der Spracherkennung kontrolliert und die Evaluationen sind wiederholbar. Dementsprechend zeigen die Evaluationen die Leistungsfähigkeit des Gesamtsystems bei Verwendung eines optimalen Spracherkenners[1]. Um den Einfluss der

[1] Wie in Abschnitt 6.1 beschrieben, verwendet die Vorverarbeitungsfließbandstufe zur Mehrfach-Automatischen-Spracherkennung ausschließlich Online-Spracherkennungssysteme. Diese werden stetig weiterentwickelt, sodass sich die für eine Audioaufzeichnung zurückgelieferten Transkriptionen prinzipiell bei jeder Ausführung ändern können.

automatischen Spracherkennung (zum aktuellen Stand der Technik) bemessen zu können, werden für eine der Evaluationen zusätzlich zum Vergleich automatisch erzeugte Transkriptionen eines Systems zur Spracherkennung verwendet. Die folgenden drei Evaluationen werden durchgeführt:

- eine Evaluation anhand einer Online-Studie, in der Probanden Pseudo-Quelltext-Generate bewerten,
- eine Evaluation konkreter *Java*-Quelltext-Generate und
- eine Evaluation der Fähigkeit zur Erzeugung von Methodendefinitionen.

Die erste Evaluation ermöglicht die Bemessung der Qualität der von ProNat erzeugten Ausgaben anhand unabhängiger (und mehrheitlicher) Bewertungen durch Laien. Die zweite analysiert hingegen Quelltext-Generate für eine konkrete Anwendungsdomäne und ermöglicht die detaillierte Bewertung der Generate auf Anweisungsebene. Als Datengrundlage für die zweite Evaluationen dienen sowohl händische Transkriptionen als auch Audioaufzeichnungen, die durch ein System zur automatischen Spracherkennung transkribiert werden. Damit ist diese Evaluation eine vollumfängliche Ende-zu-Ende-Evaluation des Gesamtsystems: Anhand (von Audioaufzeichnungen) gesprochener Handlungsanweisungen wird ausführbarer Quelltext für ein konkretes Zielsystem erzeugt. Zusätzlich wird bei dieser Evaluation für beide Varianten (manuelle und automatische Transkriptionen) die Gesamtlaufzeit des Systems pro Eingabe gemessen. Mithilfe der dritten Evaluation kann schließlich bewertet werden, wie gut ProNat das Lehren neuer Funktionalität von Handlungsanweisungen unterscheiden und Methodensignaturen und -rümpfe synthetisieren kann.

Nachfolgend wird zunächst erläutert, wie ProNat zur Durchführung der drei Evaluationen konfiguriert wird (Abschnitt 9.1). In den drei darauffolgenden Abschnitten werden die Durchführungen und Ergebnisse der Evaluationen diskutiert (Abschnitte 9.2, 9.3 und 9.4). Abschließend wird das durch die Summe der einzelnen Evaluationen gewonnene Gesamtbild der Leistungsfähigkeit des Gesamtsystems bewertet (Abschnitt 9.5).

9.1 Systemkonfiguration für die Evaluationen

Für die Durchführung aller drei Evaluationen wird ProNat im Wesentlichen einheitlich konfiguriert. Zur Verarbeitung der jeweiligen Evaluationsdaten werden alle in Kapitel 6 beschriebenen Vorverarbeitungsstufen mit Ausnahme der Mehrfach-Automatischen-Spracherkennung verwendet (siehe Abschnitt 6.1); letztere wird wie zuvor beschrieben nur für die zweite Variante der zweiten Evaluation verwendet. Alle anderen Evaluationen verwenden als Datengrundlage manuelle Transkriptionen (bzw. textuelle Beschreibungen), weshalb keine automatische Spracherkennung durchgeführt werden muss. Von den in Kapitel 7 beschriebenen Agenten werden alle bis auf vier verwendet. Die in den Evaluationen *nicht* eingesetzten Agenten sind folgende:

- Der Agent für Disambiguierung basierend auf Wikipedia (siehe Abschnitt 7.2.2)
- Der Agent zur Modellierung und Etikettierung von Diskurs-Themen (siehe Abschnitt 7.3)
- Der Agent zur automatische Auswahl von Zielsystem- und Umgebungsontologien (siehe Abschnitt 7.9)
- Der Dialog-Agent (siehe Abschnitt 7.10)

Die Fähigkeit von ProNat, Zielsystem- und Umgebungsontologien anhand von natürlichsprachlichen Äußerungen automatisch auszuwählen, wurde bereits hinlänglich im Zuge der Evaluation des Agenten bemessen (siehe Abschnitt 7.9.2). Daher wurden für die drei nachfolgend beschriebenen Evaluationen die Datensätze so gewählt, dass alle Beschreibungen an einen Haushaltsroboter als Zielsystem gerichtet werden; als zugehörige Umgebung wird in allen Szenarien die Küchenumgebung verwendet (siehe Abschnitt 5.4.3). Auf diese Weise kann die automatische Domänenauswahl als Fehlerquelle ausgeschlossen werden und der entsprechende Agent muss nicht verwendet werden. Durch diese Entscheidung kann auch auf den Agenten zur Modellierung und Etikettierung von Diskurs-Themen verzichtet werden, da derzeit lediglich der Agent zur automatischen Auswahl von Zielsystem- und Umgebungsontologien die erzeugten Themen-Etiketten verwendet. In der Folge wird auch der Agent für Disambiguierung basierend auf *Wikipedia* verzichtbar, da die *Wikipedia*-Bedeutungsetiketten bisher nur vom Agenten zur Modellierung und Etikettierung von Diskurs-Themen genutzt werden. Außerdem wird der Dialog-Agent nicht verwendet, da die Evaluationen kontrolliert und wiederholbar durchgeführt werden sollen. Daher werden ausschließlich Transkriptionen (bzw. textuelle Beschreibungen) und Audioaufzeichnung als Datengrundlage verwendet und die Ergebnisse, die ohne etwaige Korrekturen durch Dialoge entstehen, bewertet. Als Funktionswächter-Agent wird zum einen der Agent verwendet, der eine mögliche Zeitüberschreitung prüft; der Schwellenwert für die zulässige Gesamtverarbeitungszeit (durch die Agenten) wird auf fünfzehn Sekunden gesetzt. Zum anderen wird der Agent verwendet, der prüft, wie lange keine Änderung mehr am Graphen durchgeführt wurde; der Schwellenwert wird auf eine Sekunde gesetzt. Das bedeutet, die Verarbeitung einer einzelnen natürlichsprachlichen Äußerung durch die *EAS* wird spätestens nach fünfzehn Sekunden oder wenn über eine Sekunde keine Änderung am Graphen registriert wird terminiert. Diese vergleichsweise hohen Schwellenwerte wurden gewählt, um sicherzustellen, dass für alle Eingaben vollständige Analyseergebnisse erzeugt werden.

Zur Nachverarbeitung der Analyseergebnisse der Agenten werden alle in Kapitel 8 beschriebenen Fließbandstufen verwendet. Innerhalb der Stufe zur Erzeugung von Quelltext werden die Besucher zur Generierung von *Java*- und *PlantUML*-Quelltext verwendet[2] (siehe Abschnitt 8.3). Welcher Besucher für die jeweilige Evaluation verwendet wird und weitere Abweichungen von der hier festgelegten Konfigurationen, werden in den jeweiligen Abschnitten dieses Kapitels zu den einzelnen Evaluationen beschrieben.

[2] Mit *PlantUML* generierte Aktivitätsdiagramme werden als Laien-konforme Darreichungsform von Quelltext in einer Studie verwendet (siehe Abschnitt 9.2.1)

9.2 Studie zur Bewertung von Pseudo-Quelltext

Die erste Evaluation des Gesamtsystems bewertet das Zusammenspiel der Sprachanalysen von ProNat unabhängig von einer konkreten Anwendungsdomäne. Diese Herangehensweise wird verfolgt, da nur ein geringer Anteil der Sprachanalysen von der konkreten Anwendungsdomäne abhängt. Andererseits kann die Anweisungsdomäne das Evaluationsergebnis überproportional beeinflussen, beispielsweise wenn das gewählte Zielsystem eine Funktion, die der Nutzer erwartet und beschreibt, nicht anbietet. In diesem Fall würde kein Quelltext erzeugt werden, selbst wenn ProNat zuvor eine korrekte Anweisung synthetisiert hätte. Dementsprechend kann die Qualität der Sprachanalysen im Zusammenspiel ohne Anwendungsdomäne optimal bemessen werden. Daher wird für diese Evaluation Pseudo-Quelltext erzeugt.

Die Quelltext-Generate werden anschließend in einer Online-Studie von Probanden bewertet. Auf diese Weise wird eine neutrale Bewertung der Generate gewährleistet. Die Probanden sind nicht in die Auswertung der Ergebnisse involviert und haben zudem (aller Voraussicht nach) kein persönliches Interesse am Ausgang der Evaluation; somit unterliegen sie bei der Bewertung der Generate nicht dem Experimentator-Effekt. Außerdem kann jedes Generat von mehreren Probanden bewertet werden. Dadurch sind Mehrheitsentscheide (das heißt die Betrachtung mehrheitlicher Bewertungen) möglich, wodurch der Einfluss einzelner Fehleinschätzungen minimiert wird.

Als Datengrundlage dienen die Transkriptionen zu den Aufnahmen der stationären Datensammlung (siehe Abschnitt 5.5.2). Die Transkriptionen werden von ProNat verarbeitet; als Ausgabe werden *UML*-Aktivitätsdiagramme erzeugt. Durch die Verwendung von Aktivitätsdiagrammen können programmatische Strukturen in einer für Laien verständlichen Form dargestellt werden; durch die intuitive Repräsentation von Abläufen ist nur eine kurze Einführung nötig. Aktivitätsdiagramme eignen sich zudem zur Darstellung von Pseudo-Quelltext. Anweisungen bilden die Aktivitäten; auch die Darstellung von Kontrollstrukturen (bedingte Verzweigungen, Schleifen und nebenläufige Abschnitte) ist möglich.

Die von ProNat erzeugten Diagramme werden den Probanden zur Bewertung vorgelegt; als Vergleichsgrundlage erhalten die Probanden die zugehörige Transkription. Die Probanden sollen bewerten, ob ein Diagramm vollständig korrekt ist (das heißt die Modellierung entspricht der Transkription) oder ob Fehler enthalten sind. Falls die Probanden Fehler entdecken, sollen sie die Art des Fehlers benennen. Durch diesen Studienaufbau kann nicht erfasst werden, wie viele Fehler in einem Diagramm enthalten sind oder ob ein Diagramm beinahe richtig ist (wie im nachfolgenden Beispiel) oder überhaupt keine Ähnlichkeit zur Transkription aufweist. Diese Informationen von den Probanden zu erfragen könnte diese überfordern und obliegt zudem zu sehr persönlichen Einschätzungen. Da die Bewertung der Diagramme auch ohnedies subjektiv ist, sollen in der Studie alle Diagramme von mehreren Probanden bewertet werden, um mehrheitliche Bewertungen zu erhalten.

Nachfolgend wird zunächst der Studienaufbau präzisiert (Abschnitt 9.2.1). Anschließend wird die Durchführung der Online-Studie beschrieben (Abschnitt 9.2.2); zuletzt erfolgt die Auswertung der Ergebnisse (Abschnitt 9.2.3)

Beispiel:
Bewertung von synthetisierten Aktivitätsdiagrammen

In diesem Beispiel soll demonstriert werden, wie die zu bewertenden Diagramme geartet sind und welche Fehlerquellen es gibt. Das Beispiel nimmt Bezug auf das in Abbildung 9.1 dargestellte Diagramm. Als Vergleichsgrundlage dient die nachfolgende Transkription, die aus der stationären Datensammlung stammt (Szenario neun).

Transkription

hey armar check if the dish is dirty then wash the dish twice after that get me orange juice while reading the news for me

Erläuterung

Das generierte Diagramm scheint die Transkription vollständig und korrekt umzusetzen. Tatsächlich wurden alle notwendigen Aktivitäten korrekt erzeugt; auch die Kontrollstrukturen wurden korrekt umgesetzt. Sowohl die bedingte Verzweigung als auch die Schleife und der nebenläufige Abschnitt sind korrekt platziert und beinhalten die richtigen Aktivitäten. Allerdings ist fraglich, ob die erste Aktivität *hey armar check* tatsächlich notwendig ist (zumal diese auch nicht wohlgeformt erscheint). Außerdem wurde die Korreferenz innerhalb der Aktivität *while reading the news for me(me)* nicht korrekt aufgelöst. Das Diagramm enthält also – obwohl es zunächst vollständig korrekt erscheint – zwei unterschiedliche Fehlerarten.

9.2.1 Aufbau

Für die Online-Studie werden – wie auch schon zur Durchführung der Online-Datensammlung (siehe Abschnitt 5.5.3) – Web-Formulare verwendet; diese repräsentieren einen Studienbogen. Die für die Online-Studie verwendeten Web-Formulare sind schematisch im Anhang in Abschnitt C.4 abgedruckt. Sie bestehen aus den folgenden Bestandteilen. Zunächst erfolgt eine kurze Einführung. Den Probanden wird die Aufgabenstellung erläutert: Vergleich einer natürlichsprachlichen Beschreibung mit einem daraus generierten *UML*-Aktivitätsdiagramm. Anschließend werden die Probanden mit *UML*-Aktivitätsdiagrammen vertraut gemacht; es wird kurz erläutert, welche Diagramm-Elemente es gibt und wie Kontrollstrukturen dargestellt werden. Nach der Einführung werden die Probanden zur eigentlichen Aufgabe weitergeleitet; sie werden aufgefordert, nacheinander mehrere Aktivitätsdiagramme mit der zugrundeliegenden Transkription zu vergleichen und zu bewerten. Hierzu können die Probanden Haken im Formular setzen; folgende Bewertungsmöglichkeiten werden angeboten:

- **Vollständig korrekt**: Das Aktivitätsdiagramm (inklusive aller enthaltenen Aktivitäten und Kontrollstrukturen) entspricht vollumfänglich der zugrundeliegenden Transkription.
- **Korreferenz fehlerhaft**: Das Aktivitätsdiagramm enthält mindestens eine Aktivität, bei der eine Korreferenz nicht oder fehlerhaft aufgelöst wurde.

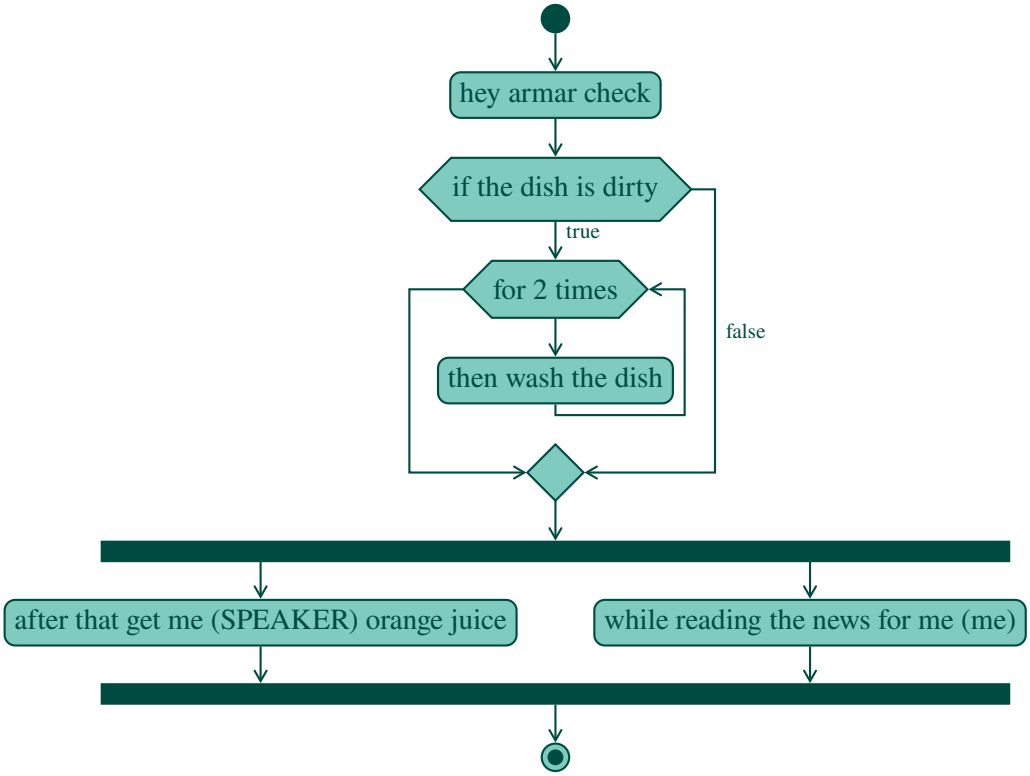


Abbildung 9.1: Ein von ProMat synthetisiertes *UML*-Aktivitätsdiagramm: Das Diagramm enthält neben Aktivitäten eine bedingte Verzweigung sowie eine Zählschleife und einen nebenläufigen Abschnitt.

- **Aktivität fehlerhaft**: Das Aktivitätsdiagramm enthält mindestens eine fehlerhafte Aktivität. Eine Aktivität ist fehlerhaft, wenn die beschriebene Aktivität entweder nicht wohlgeformt ist, oder semantisch falsch ist (z. B. können Wörter enthalten sein, die eigentlich zu einer anderen Aktivität gehören).

- **Aktivität fehlt**: Im Aktivitätsdiagramm fehlen eine oder mehrere Aktivitäten, die eigentlich aus der Transkription hervorgehen müssten. Auch Aktivitäten, die fälschlicherweise zu einer Aktivität verschmolzen wurden, gehören zu dieser Kategorie.

- **Aktivität überzählig**: Das Aktivitätsdiagramm enthält mindestens eine überzählige Aktivität, die nicht auf die Transkription zurückgeführt werden kann. Auch Aktivitäten, die fälschlicherweise in mehrere Aktivitäten aufgetrennt werden, gehören zu dieser Kategorie.

- **Reihenfolge der Aktivitäten fehlerhaft**: Das Aktivitätsdiagramm enthält zwar alle (gemäß der zugehörigen natürlichsprachlichen Äußerung notwendigen) Aktivitäten; diese sind jedoch in der falschen Reihenfolge.

- **Bedingte Verzweigung fehlerhaft**: Das Aktivitätsdiagramm enthält mindestens eine fehlerhafte bedingte Verzweigung. Eine bedingte Verzweigung ist fehlerhaft, wenn entweder die Bedingung semantisch falsch ist oder die zugehörigen Blöcke fehlerhaft sind (das heißt zu viele oder zu wenige Aktivitäten enthalten). Auch fehlende oder überzählige Verzweigungen fallen in diese Kategorie.

- **Schleife fehlerhaft**: Das Aktivitätsdiagramm enthält mindestens eine fehlerhafte Schleife; eine Schleife ist fehlerhaft, wenn entweder der Kopf (z. B. fehlerhafte Bedingung) oder der Rumpf (z. B. fehlende Aktivitäten) Fehler enthalten. Auch fehlende oder überzählige Schleifen fallen in diese Kategorie.

- **Nebenläufigkeit fehlerhaft**: Das Aktivitätsdiagramm enthält mindestens eine fehlerhafte Nebenläufigkeit; nebenläufige Abschnitte sind dann fehlerhaft, wenn ihnen falsche Aktivitäten zugeordnet wurden. Auch fehlende oder überzählige Nebenläufigkeiten fallen in diese Kategorie.

- **Sonstiges**: Das Aktivitätsdiagramm enthält andere (nicht durch die vorangegangen Fehlerklassen erfassten) Fehler. Diese können in Form von Freitext beschrieben werden.

Bei der Bewertung der Diagramme durch die Probanden sind Mehrfachnennungen gestattet, sofern nicht die Bewertung *vollständig korrekt* gewählt wurde. Das bedeutet, Probanden können ein Diagramm entweder als vollständig korrekt bewerten oder eine oder mehrere Fehlerarten wählen, die ihrer Meinung nach enthalten sind.

9.2.2 Durchführung

Als Datengrundlage zur Durchführung der Online Studie dienen vorwiegend die Transkriptionen der stationären Datensammlung. Die Dialog-Mitschnitte werden nicht verwendet. Dementsprechend stehen 201 Transkriptionen zur Verfügung (siehe Abschnitt 5.5.2.2). Von diesen wurden jedoch drei nicht verwendet. Wie in Abschnitt 5.5.2.1 beschrieben, konnte auf Wunsch der Probanden eine Aufnahme wiederholt werden, wobei sowohl die erste als auch die wiederholte Aufnahme in das Korpus aufgenommen wurden. In drei Fällen unterscheiden sich die Aufzeichnung inhaltlich nicht. Daher wurde in diesen Fällen nur eine der Aufnahmen (bzw. Transkriptionen) verwendet. Hinzu kommen die 39 Transkriptionen aus der Online-Datensammlung zu den ersten drei Szenarien der stationären Datensammlung (siehe Abschnitt 5.5.3). In Summe stehen somit 237 Transkriptionen für die Evaluation zur Verfügung. Für diese werden durch ProNat Aktivitätsdiagramme erzeugt. Hierzu wird die in Abschnitt 9.1 beschriebene Konfiguration von ProNat verwendet, wobei keine Anwendungsdomäne definiert wird. Dementsprechend können Sprachanalysen, die auf Ontologie-Informationen zugreifen, wie beispielsweise der Agent zur Modellierung des sprachlichen Kontextes (siehe Abschnitt 7.4), diesen Teil ihrer Analysen nicht anwenden. Weiterhin wird die Fließbandstufe zur Erzeugung von Quelltext mit dem Besucher zur Erzeugung von *PlantUML*-Quelltext konfiguriert (siehe Abschnitt 8.3). Auf diese Weise werden durch ProNat aus den Transkriptionen 237 Aktivitätsdiagramme erzeugt, die durch Probanden bewertet werden sollen.

Die Online-Studie wird – wie die Online-Datensammlung – über die Plattform *Prolific* durchgeführt (siehe Abschnitt 5.5.3.1). Zur Bewertung der 237 Aktivitätsdiagramme werden 24 Einzelstudien angelegt; pro Einzelstudie werden neun oder zehn Transkriptionen zufällig aus den 237 zur Verfügung

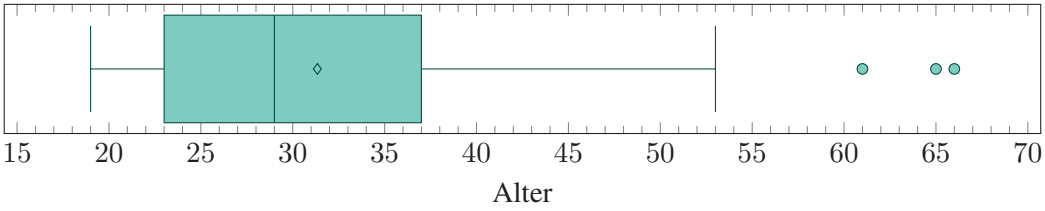


Abbildung 9.2: Kastengrafik zur Altersverteilung der Probanden der Online-Studie zur Bewertung von Pseudo-Quelltext: Neben den üblichen Bestandteilen (Median, Quartile, Antennen und Ausreißern) wird zusätzlich der Mittelwert als Raute dargestellt.

stehenden ausgewählt und daraus die Studienbögen generiert[3]. Bei der zufälligen Auswahl der Diagramme wird sichergestellt, dass jedes Diagramm genau einer Studie zugewiesen wird (das heißt es gibt keine Dopplung). Jede Einzelstudie wird für fünf Teilnehmer freigegeben; das bedeutet, jedes Diagramm wird von fünf Probanden bewertet. Hinsichtlich der Teilnehmerauswahl werden keine Einschränkungen getroffen, außer dass jeder Nutzer nur an einer der Einzelstudien teilnehmen darf. An der so konzipierten Studie nahmen 120 Teilnehmer teil[4]. 49% der Probanden sind weiblich, 51% männlich. Das Alter liegt zwischen neunzehn und 66 Jahren, wobei das durchschnittliche Alter 31,34 Jahre beträgt. Abbildung 9.2 zeigt die Kastengrafik (engl. *box plot*) zur Altersverteilung. 53% der 120 Teilnehmer gaben als Muttersprache Englisch an. Auch bei der Herkunft der Teilnehmer dominieren englischsprachige Länder; die drei am häufigsten vertretenen Nationalitäten sind britisch (42), portugiesisch (16) und US-amerikanisch (13).

9.2.3 Auswertung

In der Online-Studie wurden – alle Einzelstudien zusammengenommen – 237 Aktivitätsdiagramme von jeweils fünf Probanden bewertet; in Summe wurden dementsprechend 1185 Bewertungen abgegeben. Abbildung 9.3 zeigt die Verteilung der Einzelbewertungen[5]. Insgesamt vergaben die Probanden 580 Mal die Bewertung *vollständig korrekt*; das entspricht 49% der Diagramme. In allen anderen Diagrammen war folglich mindestens ein Fehler enthalten. Unter den angegebenen Fehlerarten macht die Bewertung *Aktivität fehlerhaft* mit 199 Stimmen den größten Anteil aus; es folgen fehlerhaft aufgelöste Korreferenzen mit 128 Nennungen[6]. Außerdem gaben die Probanden 210 mal an, dass eine Aktivität entweder fehlt oder überzählig ist; die meisten Fehler dieser Art können entweder auf eine inkorrekte Aufteilungen in Instruktionen (siehe Abschnitt 6.2) oder eine fehlerhafte Zuweisung von semantischen Rollen zurückgeführt werden (siehe Abschnitt 6.4). Derartige Fehler können auch Auswirkung auf die Benennung der Aktivitäten haben und somit

[3] Die Zuteilung erfolgt zufällig, damit nicht zu viele gleichartige Diagramme in den Einzelstudien enthalten sind. Außerdem sind die Diagramme je nach Szenario unterschiedlich umfangreich und komplex; durch die zufällige Zuteilung wird der Aufwand für die Bewertung besser verteilt als bei einer sequenziellen Zuteilung.

[4] Die nachfolgenden Werte entstammen den Nutzerprofilen der Probanden und sind daher nicht zwingend vollständig (siehe Abschnitt 5.5.3.2).

[5] Die Werte summieren sich nicht zu 1185, da bei der Auswahl der Fehlerarten Mehrfachnennung möglich waren.

[6] Es ist nicht auszuschließen, dass Probanden bei fehlerhaft aufgelösten Korreferenzen beide Möglichkeiten gewählt haben, da eine fehlerhafte Korreferenz gegebenenfalls auch die Benennung der Aktivität negativ beeinflusst (beispielsweise ist bei die Aktivität *open it(it)* unklar, welches Objekt geöffnet werden soll).

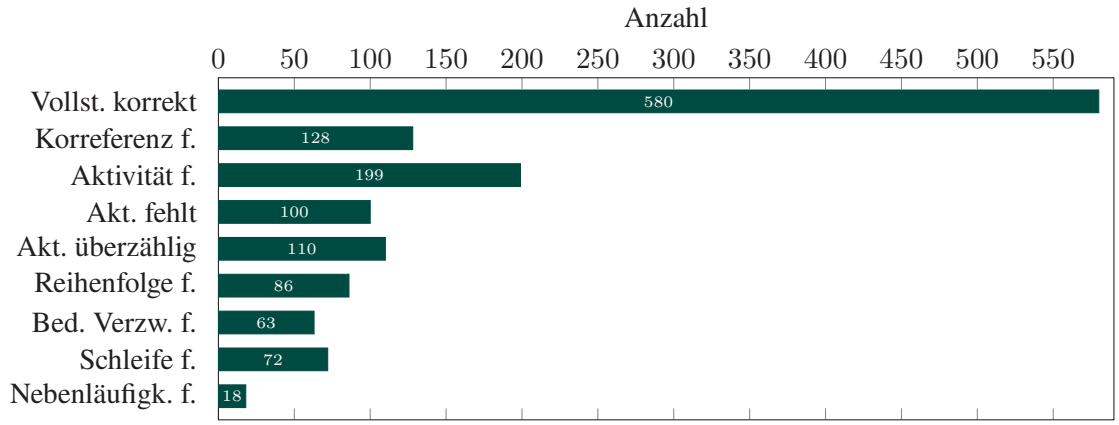


Abbildung 9.3: Verteilung der Einzelbewertungen durch die Probanden.

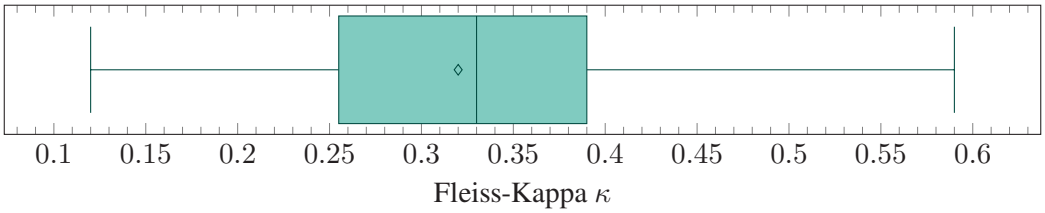


Abbildung 9.4: Kastengrafik zur Verteilung der Werte des Fleiss-Kappa κ über die Einzelstudien: Neben den üblichen Bestandteilen (Median, Quartile, Antennen und Ausreißern) wird zusätzlich der Mittelwert als Raute dargestellt.

zur Bewertung *Aktivität fehlerhaft* führen. Ein Beispiel hierzu folgt am Ende des Abschnitts. Eine inkorrekte Reihenfolge der Aktivitäten wurde 86 mal als Fehlerart gewählt. Für diese Fehlerart können keine eindeutigen Ursachen ausgemacht werden. Es kann aber nicht ausgeschlossen werden, dass Reihenfolgefehler eigentlich verdeckte Kontrollstrukturfehler sind (beispielsweise wenn eine Aktivität vor statt innerhalb eines nebenläufigen Bereich platziert wurde). Probanden könnten in solchen Fällen entweder nur *Reihenfolge fehlerhaft* oder zusätzlich einen Kontrollstrukturfehler als Bewertung gewählt haben. Fehler in Kontrollstrukturen wurden vergleichsweise selten entdeckt, die Bewertung *Nebenläufigkeit fehlerhaft* wurde sogar nur achtzehn mal gewählt (möglicherweise aufgrund des zuvor beschriebenen Effekts). Allerdings erfordern auch nicht alle Beschreibungen bzw. Transkriptionen die Verwendung von Kontrollstrukturen[7]. Außerdem ist die Bewertungen von Kontrollstrukturen noch stärker von der persönlichen Deutung der Probanden abhängig als die Auswahl der anderen Fehlerarten. Beispielsweise liegt es mitunter im Auge des Betrachters, ob eine Aktivität Teil eines Schleifenrumpfs sein sollte oder nicht. Ebenso können die Diagramme statt einer Zählschleife mit zwei Iterationen zwei aufeinanderfolgende, gleiche Aktivitäten enthalten; es obliegt dem Bewerter, dies als Fehler anzusehen oder nicht. Wurden Kontrollstrukturen tatsächlich (eindeutig) fehlerhaft synthetisiert, sind hierfür vorrangig durch die in den jeweiligen Agenten implementierten Regelsätze nicht erfasste grammatikalische Strukturen ursächlich (siehe Abschnitt 7.6).

[7] Insbesondere die Transkriptionen zu den ersten drei Szenarien enthalten kaum Beschreibungen, welche die Verwendung von Kontrollstrukturen erfordern.

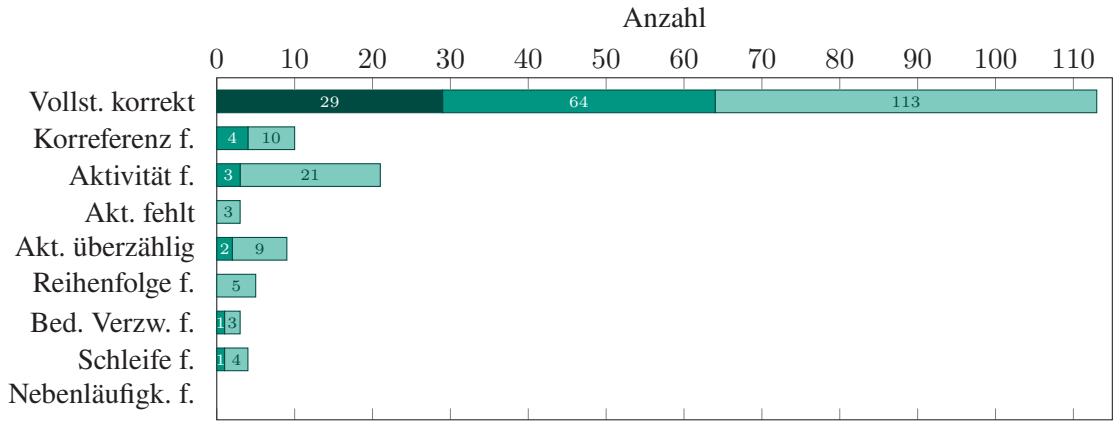


Abbildung 9.5: Bewertung je Diagramm unter Berücksichtigung des Mehrheitsentscheids durch je fünf Probanden pro Diagramm: Die dunkelgrünen Balken geben die Ergebnisse bei völliger Übereinstimmung, die mittelgrünen die Ergebnisse bei Übereinstimmung von mindestens vier Probanden und die hellgrünen bei Übereinstimmung von mindestens drei Probanden an (einfache Mehrheit).

Bis hierhin wurden nur die Bewertungen aller Probanden einzeln über alle Diagramme betrachtet. Aussagekräftiger ist hingegen die Betrachtung der gemeinsamen Bewertungen (durch jeweils fünf Probanden) pro Diagramm. Das bedeutet, für jedes Diagramm wird bestimmt, ob es mehrheitliche Bewertungen gibt. Um einschätzen zu können, ob es ausreichende Übereinstimmungen bei den Bewertungen der Probanden gibt, wird zunächst das Fleiss-Kappa für alle Einzelstudien bestimmt[8] (siehe Abschnitt 2.4.9). Abbildung 9.4 zeigt die Kastengrafik zur Verteilung der κ-Werte über die Einzelstudien. Der Mittelwert liegt bei 0,32; dies entspricht laut Landis und Koch einer mittelmäßigen, aber ausreichenden Übereinstimmung [LK77].Allerdings zeigt die Kastengrafik auch eine vergleichsweise weite Streuung der Werte und dass es auch Einzelstudien mit sehr geringer Übereinstimmung gibt. Dementsprechend steht zu erwarten, dass nicht für alle Diagramme mehrheitliche Bewertungen abgegeben wurden. Andererseits liegt der Median der κ-Werte sogar über dem Mittelwert; folglich gibt es mehr Einzelstudien mit einer ausreichenden als solche mit einer geringer Übereinstimmung.

Für alle Diagramme wurden die mehrheitlichen Bewertungen bestimmt; Abbildung 9.5 zeigt die Verteilung der Mehrheiten über die Bewertungsmöglichkeiten. Das abgebildete Balkendiagramm unterscheidet zwischen eindeutigen Bewertungen (Übereinstimmung aller fünf Probanden), starker Übereinstimmung (mindestens vier Probanden stimmen überein) und einfacher Mehrheit (mindestens drei Probanden stimmen überein). Wie die Abbildung zeigt, sind sich nur bei 29 Bewertungen alle fünf Probanden einig; diese Fälle entfallen zudem sämtlich auf die Wahl der Bewertung *vollständig korrekt*. Bei 91 Bewertungen sind sich vier Probanden einig und einfache Mehrheiten bilden sich für 168 Bewertungen. Betrachtet man einfache Mehrheiten je Diagramm, so zeigt sich, dass sich für 163 eine mehrheitliche Bewertung ergibt; das entspricht 69% aller Diagramme. 113 der Diagramme werden als *vollständig korrekt* eingeschätzt; das entspricht 69% der Diagramme für die es eine mehrheitliche Bewertung gibt und 48% aller Diagramme. 50 Diagramme enthalten hingegen eine oder mehrere Fehlerarten (31% bzw. 21%).

[8] Die Berechnung je Einzelstudie ist ausreichend, da alle Diagramme von denselben fünf Probanden bewertet wurden.

Gegenüber der Betrachtung der Einzelbewertung ergeben sich damit die folgenden Trends: Der Anteil der als vollständig korrekt angesehenen Diagramme gegenüber den Fehlerfällen ist höher bei den Einzelbewertungen, unabhängig davon welchen Grad der Übereinstimmung man zugrunde legt. Besonders deutlich wird der Unterschied jedoch bei Betrachtung der Übereinstimmung von fünf oder vier Bewertern. Bei der Wahl von Fehlerarten herrscht weniger Einigkeit, erst bei der Betrachtung von einfachen Mehrheiten zeigen sich nennenswerte Übereinstimmungen. Betrachtet man die Fehlerarten im Detail, zeigt sich, dass sich deutlich weniger Mehrheiten für fehlerhafte Korreferenzen bilden (10) als die Einzelbewertungen vermuten lassen. Gegenüber der Fehlerart *Aktivität fehlerhaft* (21) geht der Anteil deutlich zurück (48% gegenüber 64% bei Betrachtung der Einzelbewertungen); dies ist ein weiteres Indiz dafür, dass Probanden mitunter fehlerhafte Korreferenzen als fehlerhafte Aktivität deuten. Weiterhin fällt auf, dass die beiden Fehlerklassen *Aktivität fehlt* und *Aktivität überzählig*, die bei den Einzelbewertungen nahe beieinander liegen, sich bei Betrachtung des Mehrheitsentscheids deutlich unterscheiden; es gibt kaum Mehrheiten für die Fehlerart *Aktivität fehlt* (3), während immerhin in neun Diagrammen eine Mehrheit eine überzählige Aktivität ausmacht. Eine mögliche Erklärung hierfür ist, dass Probanden vereinzelt Aktivitäten als fehlend ansehen, die ihrer Meinung nach in das Szenario passen, aber keine Entsprechung in der Transkription haben, weshalb es keine Mehrheit für diese Einschätzung gibt. Auch hinsichtlich der Erkennung von Fehlern in der Reihenfolge der Aktivitäten herrscht wenig Einigkeit, lediglich bei fünf Diagrammen wird eine Übereinstimmung erzielt und dies auch nur bei Betrachtung einfacher Mehrheiten. Bei der Erkennung von fehlerhaften Kontrollstrukturen gibt es ebenfalls wenig Übereinstimmung; lediglich drei bedingte Verzweigungen und vier Schleifen werden von einer Mehrheit als fehlerhaft eingeschätzt, für fehlerhafte nebenläufige Abschnitte bilden sich keine Mehrheiten, obwohl innerhalb der Einzelbewertungen immerhin achtzehn identifiziert wurden.

Zusammenfassend deuten die Ergebnisse der Betrachtung von Mehrheitsbewertungen darauf hin, dass ein Großteil der Diagramme (69%) tatsächlich vollständig korrekt sind. Die geringe Übereinstimmung bei der Wahl von Fehlerarten kann vermutlich auf einzelne Fehleinschätzungen der Probanden zurückgeführt werden.

Um diese Ergebnisse zu bestätigen, werden in einer abschließenden Betrachtung der Ergebnisse einzelne Fehlerarten zu Fehlerkategorien zusammengefasst. Wie zuvor beschrieben, sind einige Fehlerarten verwandt, weitere können je nach Diagramm potenziell nur schwer unterschieden werden. Eine Folge ist, dass Probanden möglicherweise eine Fehlerart auswählen, obwohl sie eine andere meinen, wodurch Übereinstimmungen weniger wahrscheinlich werden. Durch die Bildung von Kategorien verwandter Fehlerarten kann dieser Effekt teilweise eliminiert werden. Ist die Anzahl der Übereinstimmungen bei Betrachtung der Kategorien größer als die Summe der Werte für die enthaltenen Fehlerarten, ist ein möglicher Grund der zuvor beschriebene Effekt. Andererseits werden potenziell auch mehr zufällige Übereinstimmungen erzeugt. Die Betrachtung von Kategorien erzeugt also gegebenenfalls mehr Übereinstimmungen, ist aber zugleich weniger scharf und dementsprechend fehlerbehaftet. Die folgenden Kategorien werden gebildet: Die ersten beiden Fehlerarten (*Korreferenz fehlerhaft* und *Aktivität fehlerhaft*) bilden die Kategorie *Aktivitätsfehler*. Reihenfolgefehler, sowie überzählige und fehlerhafte Aktivitäten bilden Kategorie *Aktivitäten-Sequenz fehlerhaft*. Die übrigen Fehlerarten werden zur Kategorie *Kontrollstruktur fehlerhaft* zusammengefasst.

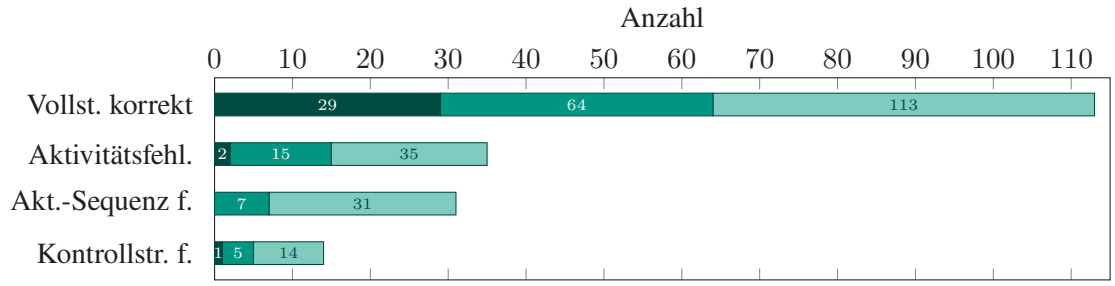


Abbildung 9.6: Zuordnung zu den verallgemeinernden Bewertungskategorien unter Berücksichtigung des Mehrheitsentscheids durch je fünf Probanden pro Diagramm: Die dunkelgrünen Balken geben die Ergebnisse bei völliger Übereinstimmung, die mittelgrünen die Ergebnisse bei Übereinstimmung von mindestens vier Probanden und die hellgrünen bei Übereinstimmung von mindestens drei Probanden an.

Eine erneute Auszählung der Bewertungen der Probanden nach diesem Schema erzeugt die in Abbildung 9.6 dargestellten Ergebnisse. Zum Vergleich enthält das Diagramm ebenfalls die Werte für die als *vollständig korrekt* eingestuften Diagramme. Wie zu erwarten, gibt es mehr Übereinstimmungen, zumindest hinsichtlich einfacher Mehrheiten. Bei Betrachtung von Fehlerkategorien erhalten 181 Diagramme eine mehrheitliche Bewertung (gegenüber 163 bei Betrachtung der einzelnen Fehlerarten). Dieser geringfügige Anstieg deutet darauf hin, dass der zuvor beschrieben Verdeckungseffekt nur selten auftritt. Die Steigerung der Anzahl an Diagrammen mit einer mehrheitlichen Bewertung hat zur Folge, dass der Anteil der vollständig korrekten Diagramme mit 62% etwas geringer ausfällt. Betrachtet man die Kategorien im Detail, zeigt sich, dass vor allem die ersten beiden Kategorien gegenüber den einzelnen Fehlerarten einen Zuwachs (auch im Grad der Übereinstimmung) verzeichnen: 35 der Diagramme werden der Kategorie *Aktivitätsfehler* zugeordnet. Etwas geringer fällt der Wert für die Kategorie *Aktivitäten-Sequenz fehlerhaft* (31); hier ist der Grad der Übereinstimmung zudem geringer. Nur vierzehn Diagramme enthalten eine fehlerhafte Kontrollstruktur, was einer Steigerung um lediglich zwei Anteile gegenüber der Summe der einzelnen Fehlerarten entspricht. Der Grund hierfür ist vermutlich, dass Kontrollstrukturen (bzw. in diesen enthaltene Fehler) von den Probanden besser unterschieden werden können als die übrigen Fehlerarten. Insgesamt bestätigt die Betrachtung der Kategorien die zuvor gewonnen Erkenntnisse. Der Zuwachs an Übereinstimmung bei den Fehlerkategorien gegenüber den einzelnen Fehlerarten ist nur gering. Folglich ergibt sich in der Bewertung der Ergebnisse ein ähnliches Bild; der Großteil der Diagramme wird als vollständig korrekt eingestuft. Zwar ist der relative Anteil zwangsweise niedriger als bei Betrachtung der Fehlerarten, allerdings dürfen auch zufällige Übereinstimmungen bei der Zusammenfassung zu Kategorien nicht außer Acht gelassen werden. Auch die Verteilung der Fehlerarten verändert sich nicht. Einzelne fehlerhafte Aktivitäten machen den Großteil aus, gefolgt von Fehlern in der Sequenz der Aktivitäten; Fehler in Kontrollstrukturen wurden hingegen kaum von den Probanden ausgemacht.

Neben den vorgegebenen Bewertungsmöglichkeiten konnten die Probanden auch für jedes Diagramm Freitextantworten abgeben. Insgesamt wurde diese Möglichkeit 49 mal genutzt. Im Folgenden werden einzelne, repräsentative Beispiele diskutiert. Einige Probanden nutzen Freitextantworten, um Verbesserungsvorschläge für die natürlichsprachlichen Beschreibungen abzugeben oder um auf potenzielle Probleme bei der Umsetzbarkeit der Anweisungen auf dem Zielsystem hinzudeuten.

Diese Freitextantworten können Hinweise zu weiterführenden Fragestellungen liefern; für die Evaluation des Gesamtsystems sind sie jedoch ohne Wert. Zwei Beispiele dieser Art sind die nachfolgenden:

(1) *everything seems correct, as ordered, but if I can add some details, it should close the fridge after taking the orange juice*

(2) *Elaboration: Armar might not know where the popcorn bag is.*

Andere nutzen die Möglichkeit zu Freitextantworten, um ihre Auswahl zu bekräftigen, oder zu präzisieren, wie folgenden beiden Beispiele zeigen:

(3) *should add an activity*

(4) *while loop missing*

Auch diese Art der Freitextantworten bietet keinen Mehrwert. Andere Probanden fügen ihrer Auswahl hingegen weitere Details oder Erläuterungen hinzu, die hilfreich sind, um Fehlerursachen nachzuvollziehen. Die folgenden drei Freitextantworten erläutern die gewählte Fehlerart *Aktivität fehlerhaft*:

(5) *Word „too“ included in wrong activity also.*

(6) *„put the plate“ „inside activate the microwave“ => „put the plate inside“ „activate the microwave“*

(7) *3rd activity should remove the first word, „out“ and place at end of 2nd activity.*

Alle drei weisen darauf hin, dass die Erkennung der Einzelinstruktionen in den zugrundeliegenden Äußerungen fehlerhaft ist (siehe Abschnitt 6.2). Die Tokensequenz wurde (teilweise) fehlerhaft in Einzelinstruktion getrennt, was dazu führt, dass Aktivitäten Wörter enthalten, die eigentlich anderen Aktivitäten zugeordnet werden sollten. Andere Freitextantworten erläutern, warum eine Kontrollstruktur als fehlerhaft angesehen wird:

(8) *when „if there are no fresh oranges“ is false, it means there are fresh oranges. It already checks once at the beginning of the task, so False check shouldn't be necessary*

(9) *I believe this needs a True/False check for „finding every piece“, because even if there aren't any pieces of dishware it should still close the dishwasher*

(10) *I think that „then please wash the green cup“ should be with „and while you wash the cup“*

In Beispiel (8) beschreibt der Proband, warum er eine Kontrollstruktur als überflüssig ansieht. Im konkreten Fall ist eine zweite verzweigte Bedingung redundant, die invers zur ersten und damit obsolet ist. Diese Fehlerart könnte behoben werden, indem der Agent zur Synthese bedingter Verzweigung um eine Überprüfung auf Verzweigungen mit inverser Bedingung erweitert werden würde (siehe Abschnitt 7.6.1). Das Beispiel (9) beschreibt eine gänzlich überflüssige Kontrollstruktur; eine mögliche Ursache ist eine ungenaue Regel in den Regelsätzen der Agenten zur Synthese von Kontrollstrukturen (siehe Abschnitt 7.6).

Das letzte Beispiel weist darauf hin, dass eine eigentlich nebenläufige Aktivität nicht dem entsprechenden Abschnitt zugeordnet wurde. Dieser Fehler lässt sich auf eine fehlerhafte Bestimmung des Bezugsrahmens der Kontrollstruktur zurückführen. Wie bereits in Abschnitt 7.6.1 diskutiert, kann der Bezugsrahmen selbst von Menschen mitunter nicht eindeutig bestimmt werden; zukünftig könnte die Erkennung aber durch die Verwendung eines statistikbasierten Ansatzes optimiert werden.

Zusammenfassend können aus der Online-Studie folgende Erkenntnisse gezogen werden: Die Auswertung der Bewertungen der Probanden hat gezeigt, dass ProNat in der Lage ist, für einen Großteil der Transkriptionen vollständig korrekten Pseudo-Quelltext (in Form von Aktivitätsdiagrammen) zu synthetisieren[9]. Dies gelingt, obwohl die Beschreibungen teilweise die Erzeugung umfangreicher und komplexer Aktivitätsdiagramme erfordern; Durchschnittlich enthalten die Diagramme etwas mehr als fünf Aktivitäten; das längste umfasst jedoch 24. Zudem erfordern die Beschreibungen die Erzeugung von 80 Kontrollstrukturen; dass mehr als eine erforderlich ist (wie im zu Beginn des Abschnitts diskutierten, beispielhaften Aktivitätsdiagramm), ist hingegen die Ausnahme (vier Fälle). Einige Aktivitätsdiagramme enthalten zudem marginale Fehler; der Grad der Fehlerhaftigkeit konnte jedoch in der Studie nicht bemessen werden.

Als hauptsächliche Fehlerart haben sich fehlerhafte Aktivitäten herausgestellt (die Anweisungen im Quelltext entsprechen). Je nachdem, wie die Aktivitäten im konkreten Fall geartet sind, haben diese Fehler jedoch nicht unbedingt Auswirkungen auf die Erzeugung von Quelltext in einer konkreten Anwendungsdomäne. Werden nur einzelne Wörter falsch zugeordnet (wie in den Beispielen (5), (6) und (7)), spielt dies bei der Abbildung auf Zielsystemfunktionen (bzw. Umgebungsobjekte) in den meisten Fällen keine Rolle (siehe Abschnitt 7.8). Die nachfolgende Evaluation, welche die Synthese von *Java*-Quelltext für eine konkrete Anwendungsdomäne betrachtet, wird diesbezüglich weitere Einsichten liefern. Fehler in der Reihenfolge der Aktivitäten, sowie fehlende Aktivitäten stellen hingegen kritische Fehler dar, die auch bei der Erzeugung von konkretem Quelltext nahezu zwangsläufig zu Fehlern führen. Überzählige Aktivitäten können, müssen aber nicht problematisch sein; gibt es keine Entsprechung in der Anwendungsdomäne, wird für diese Anweisungen auch kein Quelltext erzeugt. Ein weiterer positiver Aspekt: nur wenige der synthetisierten Aktivitätsdiagramme enthalten fehlerhafte Kontrollstrukturen.

[9] 69% oder 62% der Transkriptionen, je nachdem, ob man die einzelnen Fehlerarten, oder Kategorien zum Vergleich heranzieht.

9.3 Evaluation zur Erzeugung von Java-Quelltext

Die zweite Evaluation bemisst, wie gut ProNat Quelltext für eine konkrete Anwendungsdomäne erzeugen kann. Dementsprechend wird kein Pseudo-Quelltext betrachtet; stattdessen wird *Java*-Quelltext für eine konkrete Anwendungsdomäne generiert und anschließend bewertet. Als Anwendungsdomäne dient eine Anwendungsschnittstelle für einen Haushaltsroboter und eine Küchenumgebung. In Kapitel 8 wurde bereits gezeigt, dass ProNat Quelltext in unterschiedlichen Programmiersprachen (Abschnitt 8.3) und für unterschiedliche Anwendungsdomänen (Abschnitt 8.4) erzeugen kann. Durch die hier vorgestellte Evaluation soll zusätzlich die Güte des erzeugten Quelltextes quantitativ zu bemessen.

9.3.1 Durchführung

Als Datengrundlage für die zweite Evaluation dienen dieselben 237 Transkriptionen, die auch für die erste Evaluation verwendet wurden. Der Datensatz setzt sich aus allen 198 Transkriptionen der stationären Datensammlung und 39 Transkriptionen der Online-Datensammlung zu den ersten drei Szenarien zusammen (siehe Abschnitt 9.2.2). Zusätzlich werden zum Vergleich die Audioaufzeichnung zu ebendiesen 237 Transkriptionen als Eingabe verwendet. Damit diese verarbeitet werden können, ist die zusätzliche Verwendung der Vorverarbeitungsfließbandstufe zur Mehrfach-Automatischen-Spracherkennung nötig (siehe Abschnitt 6.1). Die Fließbandstufe wird so konfiguriert, dass ausschließlich die *Google Speech API*, nachfolgend als GS_{API} bezeichnet, zur automatischen Erzeugung von Transkriptionen verwendet wird[10]. Im Gegensatz zur ersten Studie wird aus den Transkriptionen und Audioaufzeichnungen konkreter Quelltext generiert und mit einer Musterlösung verglichen. Auf diese Weise kann die Fähigkeit des Gesamtsystems, Quelltext für eine konkrete Anwendungsdomäne zu erzeugen, quantitativ bemessen werden. Gleichzeitig ermöglicht diese Betrachtung, den wesentlichen Nachteil der ersten Evaluation auszugleichen. In der Online-Studie konnte nur bestimmt werden, ob ein Pseudo-Quelltext-Generat vollständig richtig ist oder Fehler enthält; der Anteil der fehlerhaften Bestandteile konnte nicht bestimmt werden. Durch die Betrachtung von Quelltext in der zweiten Evaluation kann auf Anweisungsebene die Korrektheit des Generats bestimmt werden. Hierzu wird für jede (händische) Transkription des Datensatzes eine Musterlösung erstellt[11]. Dieses Vorgehen ist nötig, da die Probanden die Szenarien sehr unterschiedlich beschreiben und dementsprechend auch verschiedenes Systemverhalten erwarten. Die Musterlösungen bestehen aus den erwarteten Anweisungen (bzw. API-Aufrufe). Diese werden bestimmt, indem die zur Beschreibung passenden Funktionen der Anwendungsschnittstelle und die zugehörigen Parameter bestimmt werden; Parameter können dabei Objekte der Systemumgebung

[10] Auf die Verwendung eines Ensembles von Systemen zur automatischen Spracherkennung wird verzichtet, da die Transkriptionen durch eine einzelne Web-API deutlich schneller erzeugt werden. Dies ist relevant, da im Zuge der zweiten Evaluation auch die Laufzeit des Gesamtsystem bestimmt werden soll. Außerdem wurde bereits in Abschnitt 6.1 gezeigt, dass das Ensemble nur geringfügig bessere Ergebnisse liefert als das beste Einzelsystem, die *Google Speech API*.

[11] Die jeweilige Musterlösung gilt selbstverständlich sowohl für die händische Transkription als auch für die zugehörige Audioaufzeichnung. Die Erstellung von Musterlösungen zu händischen Transkriptionen ist jedoch einfacher, da keine Wortfehler enthalten sind.

sein. Die Abfolge der Anweisungen erzeugt das (voraussichtlich vom Probanden) erwartete Systemverhalten. Zur Erzeugung einer sinnhaften Anweisungsfolge kann es nötig sein, Anweisungen einzufügen, die vom Nutzer nicht beschrieben wurden oder beschriebene Anweisungen zu ignorieren[12]. Dasselbe gilt für Parameter; die Anwendungsschnittstelle erwartet gegebenenfalls mehr Parameter als der Nutzer beschreibt, oder vom Nutzer beschriebene Parameter können nicht auf Zielsystem-Parameter abgebildet werden. Für die Musterlösungen werden nur Aufrufe erstellt, die konform zur Anwendungsschnittstelle sind.

Beispiel:
Erstellung von Anweisungen für Musterlösungen

In diesem Beispiel soll das Vorgehen bei der Erstellung von Anweisungen für die Musterlösungen der Evaluation demonstriert werden. Für dieses Beispiel wird eine synthetische natürlichsprachliche Sequenz verwendet, um mit einer kurzen Beschreibung möglichst viele Fälle abzudecken.

Natürlichsprachliche Sequenz

hey robot move to the table and stop before it then bring me the cup

Anweisungen der Musterlösung (Pseudo-Code)

```
move(Table);
fetch(Cup, Table);
move(Speaker);
```

Erläuterung

Zur Erstellung der Musterlösung wird der Beginn der Äußerung ignoriert, da dieser lediglich eine Grußformel darstellt (*hey robot*). Für die zweite Phrase (*move to the table*) kann unmittelbar eine Funktion der Anwendungsschnittstelle bestimmt und der Parameter abgebildet werden. Die nächste Phrase (*and stop before it*) ist für das Szenario nicht relevant, da der Roboter bereits beim Aufruf der `move`-Funktion stoppt, sobald er sein Ziel erreicht. Für die letzte Phrase (*then bring me the cup*) werden zwei Muster-Anweisungen erstellt. Die erste (`fetch`) hebt das Objekt `CUP` an, erwartet aber einen zweiten Parameter, der beschreibt, wo das Objekt steht, hier `TABLE`. Dieser zweite Parameter wird nicht beschrieben und müsste von `Pro`$\mathcal{Nat}$ inferiert werden. Zuletzt wird die `move`-Funktion hinzugefügt, da die Intension der Beschreibung ist, das Objekt zum Sprecher zu bringen.

[12] Vom Nutzer beschriebene Anweisungen werden ausgelassen, wenn die Anweisung nicht zur Umsetzung einer zum Szenario passenden Anweisungsfolge benötigt wird oder wenn sich hinter der korrespondieren Funktion der Anwendungsschnittstelle ein anderes Verhalten verbirgt als der Proband erzeugen wollte.

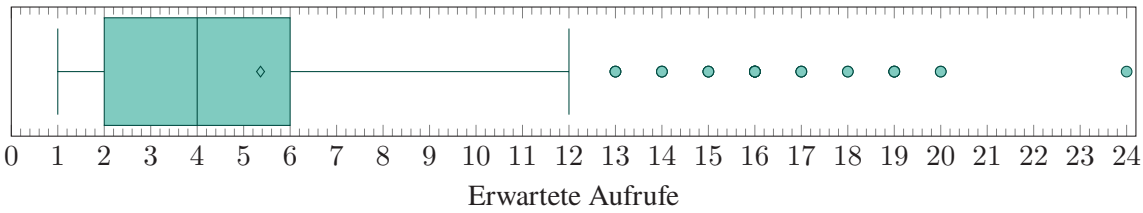


Abbildung 9.7: Kastengrafik zur Verteilung der zu erzeugenden API-Aufrufe pro Musterlösung: Neben den üblichen Bestandteilen (Median, Quartile, Antennen und Ausreißern) wird zusätzlich der Mittelwert als Raute dargestellt.

Neben den Anweisungen werden auch Musterlösungen für die Kontrollstrukturen erstellt. Das Vorgehen entspricht dem für die Anweisungen verfolgten; es wird versucht die Intension des Nutzers zu erfassen und Kontrollstrukturen dementsprechend zu gestalten. Sofern eine Kontrollstruktur die Angabe einer Bedingung erfordert, wird versucht für diese eine passende boolesche Funktion der Anwendungsschnittstelle zu ermitteln. Ein besonderes Augenmerk bei der Erstellung der Musterlösung für Kontrollstrukturen liegt auf dem Bezugsrahmen. Der Bezugsrahmen ist die natürlichsprachliche Entsprechung von Blöcken, die einer Kontrollstruktur zuzuordnen sind, wie beispielsweise ein Schleifenrumpf (siehe Abschnitt 7.6). Der Bezugsrahmen kann häufig nicht eindeutig bestimmt werden und ist vom Kontext abhängig; für die Erstellung der Musterlösung wurden einerseits die Intension der Beschreibung des Nutzers, anderseits aber auch die Szenariobeschreibung betrachtet, um sinnhafte Blöcke zu bilden.

Unter Anwendung dieser Richtlinien wurden für alle 237 Transkriptionen des Datensatzes Musterlösungen erstellt. Diese umfassen zwischen einem und 24 Aufrufe; Abbildung 9.7 zeigt die Verteilung der erwarteten Aufrufe je Musterlösung als Kastengrafik. Wie der Abbildung entnommen werden kann, erfordern die meisten natürlichsprachlichen Handlungsanweisungen des Datensatzes die Erzeugung von zwei bis sechs Aufrufen; der Mittelwert liegt bei 5,36. Viele Äußerungen erfordern allerdings auch die Erzeugung von deutlich mehr Aufrufen (Ausreißer): Die Musterlösungen zu 29 Handlungsanweisungen enthalten mehr als zwölf Aufrufe. Die Aufrufe beinhalten jeweils null bis drei Parameter. In Summe enthalten die Musterlösungen 1271 API-Aufrufe mit insgesamt 1640 Parametern, das heißt im Schnitt 6,92 Parameter pro Transkription und 1,29 Parameter pro Aufruf. Außerdem wurden zur Erstellung der Musterlösungen 80 Kontrollstrukturen verwendet.

Für die Evaluation wurde Pro$\mathcal{NAt}$ wie in Abschnitt 9.1 beschrieben konfiguriert; lediglich die Stufe zur Mehrfach-Automatischen-Spracherkennung wurde für die zweite Evaluationsvariante wie zuvor beschrieben dem Vorverarbeitungsfließband hinzugefügt. In der Nachverarbeitungsfließbandstufe zur Erzeugung von Quelltext wird der Besucher zur Generierung von *Java*-Quelltext genutzt. Als Anwendungsdomäne wird *der Haushaltsroboter* als Zielsystem in Kombination mit der zugehörigen Küchenumgebung als Systemumgebung verwendet (siehe Abschnitt 5.4.3). Die Anwendungsschnittstelle bietet 92 Funktionen und die Umgebungsmodellierung umfasst 70 Objekte und sechs Objekttypen.

Anschließend werden die Eingabedatensätze (händische und automatisch erzeugte Transkriptionen) durch Pro$\mathcal{NAt}$ verarbeitet und Quelltext-Generate erzeugt. Zunächst werden die händischen Transkriptionen an Pro$\mathcal{NAt}$ zur Verarbeitung übergeben und die Ausgaben gesammelt. Anschließend wird dem System die Vorverarbeitungsfließband zur Mehrfach-Automatischen-Spracherkennung

hinzugefügt. In der Folge können auch die 237 Audioaufzeichnung durch ProNat verarbeitet werden; auch hier werden alle generierten Quelltexte gesammelt. Für zwei Audioaufzeichnungen konnte kein Quelltext generiert werden, da kein gültiger Syntaxbaum erzeugt werden konnte; in beiden Fällen konnte aufgrund von Wortfehlern (in den automatisch erzeugten Transkriptionen) keine der natürlichsprachlichen Anweisungen auf eine Zielsystemfunktion abgebildet werden (siehe Abschnitt 7.8). Somit müssen sämtliche erwarteten Aufrufe dieser Datenpunkte als fehlend (*false negatives*) gewertet werden. Die übrigen 472 Quelltext-Generate (237 zu händischen und 235 zu automatisch erzeugten Transkriptionen) werden anschließend mit den Musterlösungen verglichen. Im nachfolgenden Abschnitt wird aber zunächst auf die Gesamtlaufzeit des Systems eingegangen, die zur Erzeugung der Generate aus den natürlichsprachlichen Äußerungen benötigt wird.

9.3.2 Analyse der Systemlaufzeit

Bei Systemen, die mit menschlichen Nutzern interagieren, ist ein wesentlicher Faktor für die Akzeptanz des Systems die Antwortzeit (engl. *response time*). Muss ein Nutzer zu lange auf eine Antwort des Systems warten, sinkt die Bereitschaft zur Nutzung [Mye94]. Um für ProNat Abschätzen zu können, wie hoch die Antwortzeit des Systems ist, wurde im Zuge der im vorangegangen Abschnitt beschriebenen Durchführung der zweiten Evaluation die Gesamtlaufzeit des Systems gemessen, die benötigt wird, um aus einer Eingabe (natürlichsprachliche Äußerung) eine Ausgabe (Quelltext-Generat) zu erzeugen. Die Laufzeit wurde sowohl für die Verarbeitung der händischen Transkriptionen also auch der Audioaufzeichnungen (das heißt mit zusätzlicher automatischer Spracherkennung durch die *Google Speech API*) gemessen; das bedeutet, insgesamt wurden 474 Laufzeitmessung durchgeführt. In die Messungen geht die Zeit zur Initialisierung des Systems, der Fließbandstufen und der Agenten *nicht* ein; diese wird einmalig vorab ausgeführt.

Die Laufzeitmessung wurde auf einem Computersystem mit folgender Konfiguration durchgeführt: Die verwendete CPU ist vom Typ *AMD Ryzen 5 3600*; diese besteht aus sechs physischen und zwölf virtuellen Prozessorkernen. Die Taktung beträgt für alle Kerne 3,6 GHz. Weiterhin verfügt die CPU über einen 384 KB großen L1-Cache, sowie einem 3-MB-L2- und einem 32-MB-L3-Cache. Der Hauptspeicher ist ein 16 GB großer RAM-Speicher vom Typ DDR4 mit einer Taktung von 2133 MHz[13]. Das verwendete Betriebssystem ist *Ubuntu Linux* in der Version 20.10.

Die Ergebnisse der Messungen sind in Abbildung 9.8 dargestellt; die Kastengrafik zeigt die Verteilung der Laufzeiten je Eingabe, einerseits für die händischen Transkription und anderseits bei Verwendung der GS_{API} zur automatischen Spracherkenner[14]. Wie der Abbildung entnommen werden kann, beträgt die Laufzeit für die Verarbeitung einer Eingabe bis zur Erzeugung des Quelltext-Generats bei

[13] Die GPU des Systems ist für die Laufzeitmessung nicht relevant, da ProNat in der aktuellen Konfiguration diese nicht nutzt.

[14] Die Laufzeiten liegen teilweise über den für die Evaluation festgelegten Schwellwert (fünfzehn Sekunden) für den Funktionswächter-Agenten, der auf eine Zeitüberschreitung prüft (siehe Abschnitt 7.11). Der Funktionswächter-Agenten überwacht allerdings nur die Ausführung der Agenten innerhalb *EAS* (siehe Abschnitt 4.2.4); das bedeutet, die Laufzeiten der Vor- und Nachverarbeitungsfließbänder sind nicht beschränkt und addieren sich zur (auf fünfzehn Sekunden beschränkten) Laufzeit der *EAS*. Wie nachfolgend diskutiert nimmt insbesondere bei Verwendung der GS_{API} das Vorverarbeitungsfließband einen Großteil der Laufzeit in Anspruch.

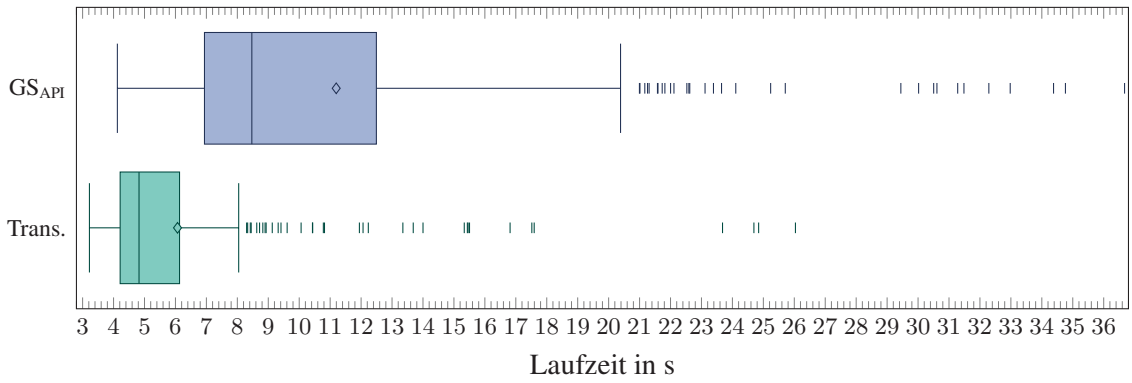


Abbildung 9.8: Kastengrafiken zur Verteilung der Gesamtlaufzeit von ProNat pro Eingabe: Die obere Grafik (GS_{API}) veranschaulicht die Verteilung der Laufzeiten für Audioaufzeichnungen (als Eingabe), die mithilfe der GS_{API} automatisch transkribiert werden; die untere zeigt die Verteilung für händische Transkriptionen (*Trans.*). Neben den üblichen Bestandteilen (Median, Quartile, Antennen und Ausreißern) wird zusätzlich der Mittelwert als Raute dargestellt.

händischen Transkriptionen in den meisten Fällen zwischen vier und sechs Sekunden; der Mittelwert liegt bei 6,1 Sekunden. Für 75% der Eingaben beträgt die Laufzeit gut 8 Sekunden oder weniger. Es gibt allerdings auch Ausreißer nach oben: Einige wenige Eingaben benötigen über zwanzig Sekunden; die höchste Laufzeit einer händischen Transkription beträgt 26 Sekunden. Insgesamt gelingt die Verarbeitung der Transkriptionen aber ausreichend schnell. Wird hingegen die GS_{API} (als zusätzliche Vorverarbeitungsfließbandstufe) für die automatische Spracherkennung zur Verarbeitung von Audioaufzeichnung verwendet, steigt die Laufzeit deutlich: Die meisten Audioaufzeichnungen können jedoch innerhalb von knapp sieben bis zwölfeinhalb Sekunden verarbeitet werden; die mittlere Laufzeit beträgt 11,2 Sekunden. Diese Werte zeigen, dass ProNat prinzipiell in der Lage ist, Ergebnisse nahezu in Echtzeit zu liefern. Allerdings endet das obere Quartil bei Verarbeitung von Audioaufzeichnung erst bei einer Laufzeit von zwanzig Sekunden; einige wenige Ausreißer benötigen sogar über 30 Sekunden bishin zu einem Maximum von 36,7 Sekunden. Die Laufzeit für diese Eingaben ist (noch) zu hoch.

Insgesamt zeigt sich, dass die Laufzeit von der Länge der Eingaben abhängt. Dies gilt insbesondere für die Verarbeitung von Audioaufzeichnung. Je länger eine Aufzeichnung, desto länger benötigt die GS_{API} zur Transkription. Da die GS_{API} bei der Verarbeitung von Audioaufzeichnung ohnehin den größten Anteil an der Laufzeit hat, fällt diese Beobachtung besonders ins Gewicht und erklärt, warum die Laufzeiten deutlich über denen für händische Transkriptionen liegen. Aber auch andere Analysen haben erhebliche Anteile an der Laufzeit: Die Erkennung semantischer Rollen benötigt vergleichsweise viel Zeit, da zur Verwendung von *SENNA* (siehe Abschnitt 6.4) zunächst ein externer Prozess gestartet werden muss; die Disambiguierung von Wortbedeutungen mittels *Babelfy* ist hingegen aufwendig, da Anfragen an eine Web-API gestellt werden müssen. Ein Agent, dessen Sprachanalysen für längere Eingaben deutlich an Komplexität gewinnen, ist der Agent zur Modellierung des sprachlichen Kontextes (siehe Abschnitt 7.4). Je länger die Eingabe, desto mehr Hierarchieebenen umfasst das erzeugte Kontextmodell. Das Kontextmodell erfasst dadurch zwar Zusammenhänge zwischen einzelnen Entitäten umfassender als bei kurzen Eingaben; gleichzeitig steigt die Laufzeit aber auch überproportional zur Länge der Eingabe.

Tabelle 9.1: Evaluationsergebnisse für die Abbildung von natürlichsprachlichen Handlungsanweisungen auf Quelltext: Die Tabelle zeigt die Ergebnisse für die Abbildung von Handlungsanweisungen auf Einzelbestandteile der Anwendungsdomäne, das heißt Funktionsaufrufe (Fkt.) und Parameter (Par.) sowie für die Erzeugung von vollständigen API-Aufrufen (Auf.). Angegeben sind jeweils die Ergebnisse für die händischen Transkriptionen (Trans.) und für die automatisch durch die *Google Speech API* erzeugten (GS_{API}) sowie die absolute Differenz zwischen beiden Ergebnissen (Δ).

	Präzision			Ausbeute			F_1		
	Trans.	GS_{API}	Δ	Trans.	GS_{API}	Δ	Trans.	GS_{API}	Δ
Fkt.	0,916	0,918	+ 0,002	0,944	0,834	− 0,110	0,930	0,874	− 0,056
Par.	0,827	0,830	+ 0,003	0,809	0,688	− 0,121	0,818	0,752	− 0,066
Auf.	0,736	0,720	− 0,016	0,757	0,656	− 0,101	0,746	0,687	− 0,059

Zusammenfassend zeigt die Analyse der Systemlaufzeit einerseits, dass händische Transkriptionen bereits überwiegend ausreichend schnell verarbeitet werden können, um eine angemessene Antwortzeit des Systems zu gewährleisten. Andererseits ist die Laufzeit zur Verarbeitung von Audioaufzeichnungen bei langen Eingagen noch zu hoch, als dass ProNat als (nahezu) echtzeitfähig gelten könnte. Problematisch ist in diesem Zusammenhang vor allem, dass ProNat in der aktuellen Systemkonfiguration Äußerungen zunächst vollständig aufzeichnet und erst danach mit der weiteren Verarbeitung beginnt. Zukünftig könnten die Sprachanalysen bereits mit einem Teil der Eingabe begonnen werden. Die meisten Web-APIs zur automatischen Spracherkennung unterstützen asynchrone Erkennungen, auch die *GS_API*. Die Architektur PARSE und das System ProNat unterstützen diesen Ansatz ebenfalls prinzipiell; nachträglich durch ein Spracherkennungssystem gelieferte Teile der Eingabe könnten einfach im ProNat Graphen als neue Knoten angehängt werden.

9.3.3 Auswertung

Die durch ProNat generierten Quelltexte werden folgendermaßen ausgewertet. Zunächst werden die einzelnen synthetisierten Aufrufe betrachtet; die Trefferquote bei der Abbildung von natürlichsprachlichen Anweisungen auf Funktionen des Zielsystems wird bewertet. Dabei wird einerseits die Korrektheit der Einzelbestandteile der Aufrufe untersucht; das bedeutet, es wird jeweils separat betrachtet, ob die richtigen Funktionen ausgewählt wurden und ob Phrasen der natürlichsprachlichen Äußerung richtig auf potenzielle Parameter abgebildet wurden. Andererseits wird auch die Korrektheit vollständiger Aufrufe, die aus den Einzelbestandteilen zusammengesetzt werden, untersucht. Sowohl für die Einzelbestandteile als auch für die vollständigen Aufrufe werden jeweils die Präzision, die Ausbeute und das F_1-Maß bestimmt (siehe Abschnitt 2.4). Anschließend wird analysiert, ob die Kontrollstrukturen an den richtigen Stellen eingesetzt wurden und ob – falls erforderlich bzw. durch die Musterlösung gefordert – korrekte Bedingungen formuliert wurden.

Zunächst werden die synthetisierten API-Aufrufe der Quelltext-Generate betrachtet. Durch Vergleich mit den Musterlösungen ergeben sich die in Tabelle 9.1 gezeigten Werte für Präzision, Ausbeute und F_1-Maß. Die Bestimmung der passenden Funktion zu einer natürlichsprachlichen Äußerung gelingt für die händischen Transkriptionen (Spalte *Trans.*) sehr präzise (0,916); die erreichte Ausbeute ist mit einem Wert von 0,944 sogar noch besser. Nur 71 Handlungsanweisungen konnten nicht auf

einen Funktionsaufruf abgebildet werden (*falsch negative*) und es wurden lediglich 110 überzählige Aufrufe ermittelt (*falsch positive*). Insgesamt ergibt sich damit für das F_1-Maß ein sehr guter Wert von 0,930. Die Ergebnisse für die Audioaufzeichnungen (Spalte GS_{API}) zeigen, dass auch bei Verwendung eines Systems zur automatischen Spracherkennung passende Funktionen sehr präzise ermittelt werden können (0,918). Allerdings kann die gegenüber den händischen Transkriptionen sogar etwas höhere Präzision darauf zurückgeführt werden, dass insgesamt deutlich weniger Aufrufe erzeugt werden. Dadurch werden zwar nur 94 fehlerhafte Funktionen generiert (*false positives*); allerdings fehlen auch deutlich mehr der erwarteten Aufrufe (*false negatives*), insgesamt 211. Folglich fällt die Ausbeute mit 0,834 (im Vergleich zu den händischen Transkriptionen) deutlich geringer aus. Dadurch wird für das F_1-Maß mit 0,874 ein guter aber dennoch im Vergleich um 0,056 geringer Wert erzielt.

Als nächstes wird die Zuordnung von natürlichsprachlichen Phrasen zu Elementen der Anwendungsdomäne betrachtet, die potenziell als Parameter verwendet werden können. Die Bestimmung von Parameter-Kandidaten funktioniert insgesamt ähnlich gut wie die Bestimmung der Funktionen. Für die händischen Transkriptionen wird eine gut Präzision bei der Abbildung erzielt (0,827; die Ausbeute ist auch nur geringfügig schlechter (0,809). Das zeigt sich auch in den absoluten Werten: 313 der 1639 erwarteten Parameter wurden nicht abgebildet (*falsch negative*), wohingegen 277 überzählige erzeugt wurden (*falsch positive*). Dadurch ergibt sich mit 0,818 ein um 0,112 geringer (als bei der Betrachtung der Funktionsaufrufe) aber nichtsdestotrotz guter Wert für das F_1-Maß. Die Ergebnisse für die Audioaufzeichnungen zeigen für die Parameter-Kandidaten einen ähnlichen Trend für die Funktionen. Auch hier ist die Präzision gegenüber den händischen Transkriptionen leicht besser (0,830); allerdings geht dies erneut zu Lasten der Ausbeute, die mit einem Wert von 0,688 deutlich geringer ausfällt. Insgesamt wurden nur 231 überzählige Parameter-Kandidaten erzeugt, allerdings auch 511 nicht gefunden.

Aus den Einzelbestandteilen setzt ProNat vollständige API-Aufrufe zusammen, die konform zur Schnittstelle des Zielsystems sind (siehe Abschnitt 7.8.1). Dementsprechend sind bei der Evaluation der vollständigen API-Aufrufe die erreichbaren Werte zwangsweise niedriger als die Werte bei Betrachtung der Einzelbestandteile; sobald ein Bestandteil falsch ist, ist der gesamte API-Aufruf falsch. Daher ist es ein sehr gutes Ergebnis, dass die in Tabelle 9.1 gezeigten Werte für API-Aufrufe sowohl für die händischen Transkriptionen als auch für die Audioaufzeichnungen so nah bei den Werten für die Einzelbestandteile liegen. Für die händischen Transkriptionen bewegen sich Präzision (0,736) und Ausbeute (0,757) auf einem ähnlichen Niveau. Insgesamt werden 964 der 1271 API-Aufrufe vollständig korrekt erzeugt. Dementsprechend wird für das F_1-Maß mit 0,746 auch ein guter Wert erzielt. Bei den Audioaufzeichnung fällt die Ausbeute (0,656) erneut deutlich gegenüber der Präzision ab (0,720). Trotzdem werden auch bei Verwendung der GS_{API} ein Großteil der erwarteten Aufrufe generiert: 834 von 1271. Das F_1-Maß erreicht aufgrund der geringen Ausbeute allerdings nur einen Wert von 0,687. Dass dieser jedoch nur um 0,059 geringer ausfällt als für die händischen Transkriptionen zeigt, dass ProNat zwar unter optimalen Bedingungen (händische Transkriptionen) die besten Ergebnisse erzielt, bei Verwendung der GS_{API} (als System zur automatischen Spracherkennung) aber bereits zum derzeitigen Stand der Technik als Gesamtsystem in der Lage ist, API-Aufrufe mit guter Präzision und ordentlicher Ausbeute zu generieren.

Die fehlerhaften und fehlenden API-Aufrufe wurden näher untersucht; dabei konnten die nachfolgenden Aspekte als hauptsächliche Fehlerquellen ausgemacht werden. Das wesentliche Problem bei der Generierung von API-Aufrufen anhand von gesprochenen Äußerungen, die mithilfe eines Systems zur automatischen Spracherkennung transkribiert wurden, sind die enthaltenen Wortfehler. Wird ein Wort von der GS_{API} falsch transkribiert, kann es in vielen Fällen nicht mehr korrekt auf eine Funktion oder einen Parameter abgebildet werden. Da mindestens ein Bestandteil fehlt, kann dementsprechend auch der erwartete vollständige API-Aufruf nicht generiert werden. Es zeigt sich jedoch, dass aufgrund von Wortfehlern nur vergleichsweise wenige *falsch positive* Abbildungen erzeugt werden. Dieses Resultat kann auf die im Agent zur Erzeugung von Methodendefinitionen und Skripten eingesetzten Schwellwerte bei der Abbildung zurückgeführt werden (siehe Abschnitt 7.8.1). Andererseits gelingt es in einigen Fällen trotz Wortfehlern, eine passende Abbildung zu finden. Beispielsweise wird das Wort *cupboard* vereinzelt durch die GS_{API} als *carport* oder *cardboard* transkribiert. Aufgrund der (durch die Ontologie-Anbindung angebotenen) unscharfen Suchstrategien (siehe Abschnitt 5.4.2) erhält das Ontologie-Individuum `Cupboard` trotzdem in diesen Fällen die höchste Abbildungsbewertung. Alle weiteren Fehlerquellen betreffen sowohl händische als auch automatisch erzeugte Transkriptionen, wobei viele durch etwaige Wortfehler verstärkt werden. Eine solche Fehlerquelle ist, wie auch schon bei der vorangegangenen Evaluation identifiziert, die teilweise fehlerhafte Aufteilung der natürlichsprachlichen Äußerungen in einzelne Instruktionen (siehe Abschnitt 6.2). Werden die Äußerungen falsch geteilt, werden entweder eigentlich zusammenhängende Phrasen getrennt oder eine Instruktion enthält mehrere Handlungsanweisungen; in beiden Fällen ist eine korrekte Abbildung auf Zielsystemfunktionen nahezu ausgeschlossen. Eine weitere bereits aus der vorangegangenen Evaluation bekannte Fehlerquelle sind Korreferenzen; fehlerhafte und fehlende Korreferenzen führen vor allem dazu, dass Parameter nicht korrekt bestimmt werden können. Zuletzt zeigt sich, dass einige Probanden ihre Beschreibungen so formulierten, dass eine Abbildung auf Zielsystemfunktion nicht möglich ist; unter anderem nutzen einige Probanden eine ungewöhnliche Wortwahl, verwendeten Synonyme, die nicht aufgelöst werden konnten, oder tätigten grammatikalisch falsche Äußerungen[15]. Die nachfolgenden Beispiele zeigen drei solche ungewöhnlichen (bzw. fehlerhaften) Formulierungen aus der stationären Datensammlung, die durch `Pro`$\mathcal{NL}$ nicht (oder nicht vollständig) auf Zielsystemfunktion bzw. -parameter abgebildet werden konnten. Die fehlerhaften Bestandteile sind jeweils fettgedruckt:

(1) *take the **dirty dishes** and put **it** into the dishwasher*

(2) ***take** all the dirty dishes **into** the dishwasher*

(3) ***searched** the orange juice*

Von den 80 in den Musterlösungen definierten Kontrollstrukturen wurden für die händischen Transkriptionen alle von `Pro`$\mathcal{NL}$ erzeugt; es wurden zudem keine überzähligen Kontrollstrukturen generiert. Auch bei Verwendung der Audioaufzeichnungen als Eingabe werden keine überzähligen

[15] Eine Ursache für diese Art von Fehler ist höchstwahrscheinliche mangelnde Sprachkompetenz; die Probanden, die zur stationären Datensammlung beitrugen, sind alle Nicht-Muttersprachler (siehe Abschnitt 5.5.2).

Kontrollstrukturen erzeugt. Allerdings werden fünf laut Musterlösung erwartete Kontrollstrukturen nicht erzeugt: die GS_{API} transkribiert in diesen Fällen Wörter falsch, die als Schlüsselwörter für die Erkennung der Kontrollstrukturen verwendet werden (siehe Abschnitt 7.6). Wird kein Schlüsselwort gefunden, wird durch die Agenten auch keine Kontrollstruktur erzeugt. Auch die erzeugten Kontrollstrukturen sowohl für die händischen als auch die automatischen Transkriptionen nicht frei von Fehlern; teilweise wurde der Bezugsrahmen falsch bestimmt und dadurch fehlerhafte Blöcke gebildet (siehe Abschnitt 7.6). Die Blöcke enthalten entweder zu viele oder zu wenige Anweisungen. Teilweise können Fehler dieser Art auch auf nicht synthetisierte oder überzählige API-Aufrufe zurückgeführt werden; dementsprechend ist die Betrachtung der Kontrollstrukturen nicht unabhängig von der zuvor angestellten Betrachtung der API-Aufrufe. Für die händischen Transkriptionen sind 22 von 144 Bezugsrahmen fehlerhaft, was einer Genauigkeit von 84,7% entspricht. Bei Verwendung der Audioaufzeichnungen als Eingabe sind hingegen 57 Bezugsrahmen falsch (60,4% Genauigkeit). Auch die synthetisierten Bedingungen sind teilweise fehlerhaft. In einigen Fällen konnte keine passende (boolesche) Funktion der Anwendungsschnittstelle bestimmt werden[16][17]. Außerdem scheinen Wortfehler bei der Erkennung der Bedingungen einen großen Einfluss zu haben, denn insgesamt können für die Audioaufzeichnungen nur 37 von 69 Bedingungen generiert werden (53,6% Genauigkeit). Für die händischen Transkriptionen sind immerhin 55 korrekt, was einer Genauigkeit von 79,7% entspricht. Ein fehlerhafter Bezugsrahmen und eine falsche Bedingung treten häufig in derselben Kontrollstruktur auf; die Kontrollstruktur *Nebenläufigkeit* erfordert zudem keine Bedingung. Daher ergibt sich die Korrektheit vollständiger Bedingung (bestehend aus Bedingung und Bezugsrahmen) nicht aus dem Produkt der Genauigkeiten der zuvor bestimmten Werte. Für händische Transkriptionen können 57 von 80 Kontrollstrukturen vollständig korrekt erzeugt werden (71,3%); für Audioaufzeichnungen sind es hingegen nur 41 (51,3%).

Zusammenfassend konnte die zweite Evaluation zeigen, dass ProVal auch für eine konkrete Anwendungsdomäne Quelltext (in diesem Fall *Java*) generieren kann. Insbesondere die Erzeugung einzelner Anweisungen (das heißt Aufrufe an eine Zielsystem-API) funktioniert in den meisten Fällen zuverlässig: für das F_1-Maß wird bei der Betrachtung einzelner Aufrufe ein Wert von 0,746 für händische Transkriptionen erreicht; für die Audioaufzeichnung wird ein nur geringfügig schlechterer Wert von 0,687 erzielt. Wie zu erwarten, haben Wortfehler, die durch die automatische Spracherkennung entstehen, den stärksten negativen Einfluss auf die Ergebnisse (im Vergleich zu den händischen Transkriptionen). Allerdings hat sich auch gezeigt, dass durch die bei der Abbildung (von natürlichsprachlichen Phrasen auf Zielsystemfunktionen und -parameter) verwendeten unscharfen Suchstrategien dazu führen, dass trotz vereinzelten Wortfehlern korrekte API-Aufrufe erzeugt werden können. Bei der Erzeugung von Kontrollstrukturstrukturen wirken sich Wortfehler hingegen stärker aus, da die angewendeten Verfahren auf bestimmte Schlüsselwörter bzw. -phrasen angewiesen sind. Daher können anhand von Audioaufzeichnung nur 51,3% der Kontrollstrukturen vollständig korrekt erzeugt werden. Für händische Transkriptionen werden hingegen immerhin 71.3% der

[16] ProVal kann zum aktuellen Stand nur Bedingungen synthetisieren, die auf eine boolesche Funktion der Zielsystemschnittstelle abgebildet werden können.

[17] Um dieses Problem zu lösen, könnte zukünftig ein ProVal-Agent entwickelt werden, der dediziert boolesche Bedingungen prüft bzw. aus gesprochenen Äußerungen prüfbare Bedingungen erzeugt. Beispielsweise könnte ein solcher Agent anhand der in den Umgebungsontologien modellierten Zustände von Objekten Bedingungen ableiten.

Kontrollstrukturen richtig generiert. Die Injektion von Kontrollstrukturen in Skripte ist dementsprechend generell verbesserungswürdig. Insgesamt sind 84 (35,4%) der 237 anhand händischer Transkriptionen erzeugter Quelltext-Generate vollständig fehlerfrei, für die Audioaufzeichnungen sind es immerhin 75 (31,6%). Vor allem Skripte, die aus weniger als sieben Anweisungen bestehen, sind häufiger korrekt und tragen dementsprechend viel zu diesem Ergebnis bei[18].

9.4 Evaluation der Synthese von Methodendefinitionen

In der dritten und abschließenden Evaluation des Gesamtsystems wird die Fähigkeit von ProNat untersucht, neue Funktionalität anhand von natürlichsprachlichen Beschreibungen zu erlernen. Innerhalb der intrinsischen Evaluation des Agenten zur Synthese von Methodendefinitionen und Skripten konnte zwar bereits gezeigt werden, dass dies möglich ist (siehe Abschnitt 7.8.2). Allerdings wurde dieser Aspekt bisher nur unter Verwendung von Mustereingaben für einen Agenten und noch nicht unter Verwendung des Gesamtsystems untersucht. Zur weiteren Abgrenzung von der Evaluation des Agenten wird für die hiesige Evaluation ein neuer Datensatz erstellt, der aus Beschreibungen zu zwei neuartigen Szenarien besteht, die über eine Online-Studie gesammelt werden. Als Anwendungsdomäne dient wie zuvor der *Haushaltsroboter* in seiner *Küchenumgebung* (siehe Abschnitt 5.4.3). Auf dieser Grundlage wird zunächst bestimmt, wie gut ProNat das Lehren neuer Funktionalität von einfachen Handlungsanweisungen unterscheiden kann. Außerdem wird die Qualität der synthetisierten Methodennamen bewertet. Zuletzt wird die Treffergenauigkeit bei der Auswahl der API-Aufrufe (und Parameter) zur Bildung der Methodenrümpfe bemessen.

9.4.1 Durchführung

Die Datengrundlage für die Evaluation wird durch eine Online-Studie gebildet. Für die Durchführung der Studie wurde wiederum die Plattform *Prolific* verwendet. Der Studienaufbau entspricht im Wesentlichen dem der Online-Datensammlung (siehe Abschnitt 5.5.3). Die Studienteilnehmer sollen natürlichsprachliche Lehrbeschreibungen für den Haushaltsroboter zu Szenarien abgeben. Der für die Evaluation verwendete Studienbogen umfasst zwei Szenarien, die nicht Teil der Online-Datensammlung waren[19]:

- *Szenario A*: Spülmaschine einschalten[20]
- *Szenario B*: Müsli zubereiten

[18] Dass kurze Skripte eher korrekt sind als längere folgt aus einer einfachen wahrscheinlichkeitstheoretischen Betrachtung. Für jede synthetisierte Anweisung ist die Wahrscheinlichkeit (nahezu) gleich groß, einen Fehler zu produzieren. Daher steigt die Wahrscheinlichkeit, dass ein Quelltext-Generat mindestens einen Fehler aufweist, mit jeder weiteren Anweisung. Dasselbe gilt für Kontrollstrukturen. Allerdings gibt es bei deren Erzeugung drei potenzielle Fehlerquellen: die Bedingung, die Blockgröße und die Positionierung.

[19] Zur Unterscheidung von den Szenarien der Online-Datensammlung werden die Szenarien mit *A* und *B* benannt.

[20] Das Szenario entspricht dem Beispielszenario der Online-Datensammlung.

Wie auch schon im Studienbogen der Online-Datensammlung wird den Studienteilnehmern die Aufgabe anhand eines Beispielszenarios erläutert[21]. Nach der Einführung, die aus einer kurzen Beschreibung der Zielstellung der Studie, einer Einführung in den Studienaufbau und dem Beispielszenario besteht, werden die Probanden aufgefordert natürlichsprachliche Beschreibungen zu den Szenarien abzugeben. Die Beschreibungen können in ein dafür vorgesehenes Textfeld eingegeben werden und sollen möglichst spontan verfasst werden. Inhaltlich soll einem Haushaltsroboter eine neue Funktion gelehrt werden; die Probanden sollen die Zwischenschritte beschreiben, die nötig sind, um die Fähigkeit zu erlernen, und die neue Funktionalität benennen.

Die Studie wurde für 110 Teilnehmer freigegeben; die Einreichungen von neun Teilnehmern wurden nachträglich aussortiert, da sie keine sinnhaften Beschreibungen enthielten. Die so entstandene Datensammlung umfasst dementsprechend je 101 Beschreibungen pro Szenario. Tabelle 9.2 gibt einen Überblick über den Umfang der Datensammlung. Die Beschreibungen enthalten in Summe 8375 Wörter, wobei ein deutlicher Unterschied zwischen den beiden Szenarien besteht. Während die Beschreibungen zu Szenario A im Durchschnitt lediglich aus gut 29 Wörtern bestehen, sind es bei Szenario B knapp 54. Ein Großteil der Probanden beschreibt zum Lehren der Funktionalität *Müsli zubereiten* in Szenario B mehr Arbeitsschritte. Die Beschreibungen beider Szenarien umfassen zusammengenommen im Durchschnitt gut 41 Wörter und damit deutlich mehr als die Beschreibungen der Online-Datensammlung, die durchschnittlich nur aus 31 Wörtern bestehen.

Neben den Beschreibungen wurden (wie auch bei den anderen Online-Studien) anonymisiert persönliche Daten aus den Profilen der Nutzer von *Prolific* übermittelt[22]. In der Studie überwiegt der Anteil der weiblichen Teilnehmer (57%) leicht gegenüber den männlichen (43%). Die Alterspanne reicht von neunzehn bis 72 Jahren; das durchschnittliche Alter liegt mit 32,11 etwas über denen der anderen Online-Studien (siehe Abschnitt 5.5.3.2 und Abschnitt 9.2.2). Abbildung 9.9 zeigt die Kastengrafik zur Altersverteilung der Probanden. Diese offenbart, dass der Großteil der Probanden zum Zeitpunkt der Teilnahme zwischen 23 und 37 Jahre alt waren. 50% der Probanden haben in ihrem Profil Englisch als Muttersprache hinterlegt. Dementsprechend ist auch die am häufigsten angegebene Nationalität britisch (33), gefolgt von US-amerikanisch (16) und portugiesisch (11).

Für die Evaluationen werden aus dem Datensatz je 50 Beschreibungen pro Szenario ausgewählt, das heißt in Summe 100[23]. Zu jeder Beschreibung werden Musterlösungen erstellt. Hierzu wird zunächst bestimmt, ob es sich bei der Beschreibung um einen Versuch, eine Funktion zu lehren, handelt oder ob der Proband lediglich eine Reihe von auszuführenden Handlungsanweisungen beschrieben hat. Nur für die lehrenden Beschreibungen sollte ProNat eine neue Methode synthetisieren, wohingegen aus den nicht-lehrenden Beschreibungen Skripte generiert werden sollten. Außerdem werden die erwarteten Anweisungen (das heißt API-Aufrufe), die Bestandteil des Methodenrumpfs (bzw. der Skripte) sein sollten, bestimmt. Für die Methodensignatur wird vorab keine Musterlösung bestimmt; die synthetisierten Signaturen werden stattdessen nachträglich manuell bewertet. Auch

[21] Als Beispielszenario wird das zweite Szenario der Online-Datensammlung *Kaffee zubereiten* verwendet.

[22] Da es sich um freiwillige Angaben handelt, sind diese teilweise unvollständig.

[23] Durch diese Auswahl, kann der Umfang der Evaluation eingeschränkt werden. Dies ist nötig, da einerseits ein Teil der Evaluation partiell manuell erfolgt; die Bewertung der synthetisierten Methodensignaturen, kann nur nachträglich beurteilt werden (siehe Abschnitt 7.8.2). Anderseits ist auch die Erstellung der Musterlösung für die Evaluation der Methodenrümpfe aufwendig.

Tabelle 9.2: Statistiken zur Datensammlung zur Evaluation der Methodenerzeugung: Die Tabelle gibt pro Szenario der Online-Studie an, wie viele Probanden jeweils teilgenommen haben und wie viele Beschreibungen dabei entstanden sind. Zudem sind jeweils die Gesamtanzahl und die Durchschnittswerte der Wörter der Beschreibungen angegeben.

Szenario	Probanden	Beschreibungen	Wörter (gesamt)	Wörter (durchschnittlich)
A	110	101	2930	29,010
B	110	101	5448	53,941
Alle	110	202	8378	41,475

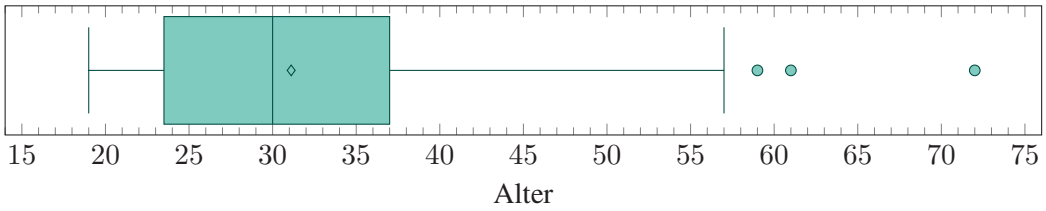


Abbildung 9.9: Kastengrafik zur Altersverteilung der Probanden der Online-Studie zur Evaluation der Methodensynthese: Neben den üblichen Bestandteilen (Median, Quartile, Antennen und Ausreißern) wird zusätzlich der Mittelwert als Raute dargestellt.

Tabelle 9.3: Statistiken zu den für die Evaluation der Methodenerzeugung verwendeten Beschreibungen: Die Tabelle gibt pro Szenario und in Summe an, wie viele Beschreibungen für die Evaluation ausgewählt wurden, wie viele von diesen die Definition einer neuen Funktion beschreiben (*lehrend*) und wie viele API-Aufrufe für die Methodenrümpfe (bzw. Skripte) erwartet werden (laut Musterlösung).

	Beschreibungen	lehrend	nicht-lehrend	API-Aufrufe
Szenario A	50	44	6	158
Szenario B	50	34	16	315
Alle	100	78	22	473

Kontrollstrukturen werden nicht betrachtet, da keine in den Beschreibungen der Probanden enthalten sind[24]. Tabelle 9.3 zeigt den Umfang des Datensatzes sowie die zu erwarteten Methodendefinitionen (lehrende Beschreibungen) und API-Aufrufe (gemäß der Musterlösung). Die Beschreibungen zu Szenario A sind im Vergleich zu denen zu Szenario B zu einem größeren Anteil lehrend; allerdings sollten zu den Beschreibungen zu Szenario B fast doppelt so viele API-Aufrufe generiert werden[25].

Die 100 ausgewählten Beschreibungen werden an Pro$\mathcal{NL}$ zur Verarbeitung übergeben. Das System ist wie in Abschnitt 9.1 beschrieben konfiguriert. Als Anwendungsdomäne dient – passend zu den Szenarien – der *Haushaltsroboter* (*ARMAR-III*) und die zugehörige *Küchenumgebung*. Die von Pro$\mathcal{NL}$ erzeugten Methodendefinitionen und Skripte können anschließend mit den Musterlösungen verglichen und bewertet werden.

[24] Der Grund hierfür liegt in den Szenarien, die sich als Anweisungsfolge beschreiben lassen, ohne dass Verzweigungen nötig wären. Da sich keine Handlungen wiederholen sollen, ist es auch nicht nötig Schleifen zu beschreiben; auch die nebenläufige Handlungen werden nicht erwartet.

[25] Diese Beobachtung korrespondiert mit dem in Tabelle 9.2 dokumentierten Wortumfang der Beschreibungen zu den Szenarien.

9.4.2 Auswertung

Die Qualität der von ProNat erzeugten Methodendefinitionen und Skripte wird folgendermaßen bemessen. Zunächst wird untersucht, in wie vielen Fällen ProNat in der Lage war, den Wunsch eine neue Funktion zu lehren (Methodendefinition) von der Beschreibung einfacher Handlungsanweisungen (Skripterzeugung) zu unterscheiden. Anschließend werden die synthetisierten Methodennamen analysiert und bestimmt, wie häufig ein passender Name gewählt wurde. Zuletzt werden die in den Methodenrümpfen und Skripten enthaltenen Anweisungen untersucht. Dabei wird zwischen einzelnen Bestandteilen (Methodenname und Parametern) sowie vollständigen Aufrufen an die Anwendungsschnittstelle unterschieden. In beiden Fällen werden jeweils die Präzision, die Ausbeute und das F_1-Maß bestimmt.

Von den 100 Beschreibungen des Datensatzes stellen laut Musterlösung 78 eine Methodendefinition dar, für die übrigen 22 sollten hingegen Skripte erzeugt werden. ProNat ordnet in der Evaluation 85 der 100 Beschreibungen richtig einer der beiden Kategorien zu, wobei keine Tendenz bei der Art des Fehlers auszumachen ist; es werden anteilig ähnlich viele Äußerungen nicht als Beschreibung einer neuen Funktion erkannt, obwohl sie das laut Musterlösung sind, wie umgekehrt. Insgesamt erzeugt ProNat aus 73 Beschreibungen eine Methodendefinition, davon 39 für Szenario A und 34 für Szenario B. Für die 73 synthetisierten Methodennamen soll anschließend analysiert werden, ob ein passender Methodenname für die Methodensignatur gewählt wurde. Ein Methodenname gilt dabei dann als passend, wenn er die angebotene Funktionalität der neu geschaffenen Methode beschreibt und darüber hinaus keine weiteren überflüssigen oder unpassenden Wörter enthält. Die Wahl eines passenden Methodennamens ist deshalb wichtig, weil die neue Methode anhand dieses Namens später durch den Nutzer verwendet werden kann. Die Einordnung nach passenden und unpassenden Namen soll anhand einiger Beispiele verdeutlicht werden. Für Beschreibungen zu Szenario A wurden beispielsweise die nachfolgenden Methodennamen generiert:

Szenario A - *Spülmaschine starten*:

1. startDishwasher
2. startDishwasherHello
3. startDishwasherHelloFirst
4. turnOnDishwasher
5. useDishwasher
6. pushTheBlueButton

Die Methodennamen eins bis fünf beschreiben offensichtlich die Funktionalität der neuen Methode; der sechste Name ist hingegen völlig unpassend und beschreibt stattdessen einen Teilschritt (bzw. eine Anweisung) des Methodenrumpfs. Darüber hinaus enthalten die Beispiele zwei und drei

überflüssige Wörter (*Hello* und *HelloFirst*)[26]. Diese beiden Namen werden daher ebenfalls als unpassend angesehen. Unter Anwendung dieser Kriterien werden insgesamt 34 der 39 synthetisierten Methodennamen zu Szenario A als passend bewertet, fünf hingegen als unpassend. Dieselbe Betrachtung wird für Beispiele zu Szenario B angestellt:

Szenario B - *Müsli zubereiten*:

1. prepareCerealFirstThing
2. prepareSomeCereal
3. prepareABowlOfCereal
4. makeCereal
5. makeCereals
6. makeSomeCereals
7. makeBowlOfCereal
8. prepareCereals

Alle acht der hier gezeigten synthetisierten Methoden beschreiben die neue Funktionalität. Nur das erste Beispiel enthält offensichtlich falsche Wörter und gilt daher als unpassend. Die Beispiele zwei, drei, sechs und sieben zeigen Namen, die weitere Wörter enthalten, die nicht unbedingt zur Beschreibung der Funktionalität benötigt werden (*Some* und *BowlOf*). Allerdings sind diese auch nicht falsch und bilden gemeinsam mit den übrigen Wörtern sinnhafte Methodennamen; daher werden diese Namen als passend angesehen. Insgesamt sind lediglich zwei der 34 synthetisierten Methodennamen zu Beschreibungen zu Szenario B unpassend. Für beide Szenarien werden insgesamt 66 der 73 Methodennamen als passend bewertet, sieben sind hingegen unpassend. Damit ergibt sich eine Genauigkeit von 0,904 für die Synthese der Methodennamen in dieser Evaluation.

Für alle erzeugten Methoden wurde anschließend geprüft, ob diese in der Folge auch (durch eine natürlichsprachliche Handlungsanweisung) aufgerufen werden können. Hierzu wurde jede der 73 Beschreibungen, anhand derer `Pro`$\mathcal{N}at$ eine neue Funktion erzeugen konnte, einzeln betrachtet[27]. Zunächst wird die neue Funktion wie zuvor beschrieben automatisch durch `Pro`$\mathcal{N}at$ erzeugt und der Methodenname als neues Individuum der Klasse `Mehod` in der Zielsystemontologie hinterlegt[28].

[26] Die Fehlerquellen sind zum einen die Erkennung der Instruktionen, die in diesen Fällen nicht korrekt aufteilt (siehe Abschnitt 6.2). Zum anderen hat der Klassifikator zur Bestimmung der semantischen Funktion von Phrasen in Lehrsequenzen die Wörter fälschlicherweise dem *deklarativen* Teil der Beschreibung zugeordnet (siehe Abschnitt 7.7.4). Zuletzt könnten verbesserte Filtermechanismen bei der Synthese des Namens angewendet werden, um derartige Fälle auszuschließen (siehe Abschnitt 7.8.1).

[27] Die Aufrufbarkeit wird je Beschreibung bestimmt, da so die realistische Verwendung des Systems am besten abgebildet wird. Ein Nutzer wird dieselbe neue Funktion nur einmal definieren und nicht mehrfach hintereinander mithilfe unterschiedlicher Formulierungen. Würden alle neuen Methodendefinitionen gleichzeitig dem System hinzugefügt werden, würden eine Vielzahl gleicher (oder sehr ähnlicher) Methodendefinitionen und -namen entstehen. Die Aufrufbarkeit einer einzelnen dieser Funktionen wäre auf diese Weise kaum bestimmbar.

[28] Das bedeutet, die Klasse `Mehod` der Ontologie enthält anschließend neben den Individuen, welche die Anwendungschnittstellenfunktionen repräsentieren, genau ein weiteres Individuum, welches die neu definierte Funktion repräsentiert.

Tabelle 9.4: Natürlichsprachliche Handlungsanweisungen, die zum Test der Aufrufbarkeit neuer Methodendefinitionen verwendet wurden.

Szenario A	Szenario B
start the dishwasher please	*prepare some cereal for me please*
please turn on the dishwasher	*make some cereal please*
please start the dishwasher	*make a bowl of cereal*
switch on the dishwasher please	*I want a bowl of cereal*
turn on the dishwasher now	*please prepare me some cereal*

Tabelle 9.5: Erzielte Genauigkeiten bei der Erzeugung von Aufrufen neu definierter Methoden: Die Tabelle zeigt die Genauigkeiten je Szenario und insgesamt; unterschieden wird zudem zwischen allen (zuvor neu generierten) Methodennamen und wohlgeformten.

	alle	wohlgeformte
Szenario A	0,856	0,912
Szenario B	0,759	0,788
Gesamt	0,811	0,852

Um die Aufrufbarkeit prüfen zu können, wurden je Szenario fünf beispielhafte (textuelle) Äußerungen erstellt; diese sind in Tabelle 9.4 aufgeführt. Anschließend kann bestimmt werden, ob ProNat für die Äußerungen den korrekten (einzelnen) Methodenaufruf der neuen Funktion generiert. Auf diese Weise wird die Genauigkeit (siehe Abschnitt 2.4.4) der Abbildung einer Handlungsanweisung auf die neue Funktion ermittelt. Dieses Prozedere wurde für alle 73 generierten Methoden mit denselben fünf (jeweils zum Szenario passenden) Äußerungen wiederholt. Dementsprechend wurden in Summe 365 Tests durchgeführt. Die Ergebnisse sind in Tabelle 9.5 aufgeführt; unterschieden werden die Genauigkeiten bei Betrachtung *aller* generierten und *wohlgeformter* Methodennamen.

Für Szenario A wurden für 85,6% der Äußerungen korrekte Methodenaufrufe der neuen Funktionen erstellt; betrachtet man nur die wohlgeformten Methodennamen, wird eine Genauigkeit von 91,2% erzielt. Der Aufruf der neuen Funktionen zu Szenario A funktioniert somit weitestgehend zuverlässig. In vielen Fällen konnten sogar Aufrufe von Methoden mit *nicht* wohlgeformten Methodennamen generiert werden. Unter den wohlgeformten gelang die Generierung nur für die Methodennamen `useDishwasher` und `turnOnDishwasher` nicht in allen Fällen. Für Szenario B konnten nur in etwas mehr als dreiviertel der Fälle ein korrekter Aufruf generiert werden (75,8%); auch bei ausschließlicher Betrachtung der wohlgeformten Methodennamen ergibt sich ein ähnliches Bild (78,8%). Damit sind die Ergebnisse deutlich schlechter als für Szenario A. Ein wesentlicher Faktor ist, dass der Methodenname `prepareSomeCereals` bei Äußerungen, welche die Phrase *make cereal* enthalten, nicht gefunden wird; stattdessen erhält der Aufruf `make(CerealBox)` eine höhere Abbildungsbewertung (siehe Abschnitt 7.8.1). Insgesamt sind die Ergebnisse des Aufruftests jedoch positiv zu bewerten: In 81,1% der Fälle wurde der korrekte Methodenaufrufe zu einer Äußerungen bestimmt; für wohlgeformte Methodennamen beträgt die Genauigkeit sogar 85,2%.

Tabelle 9.6: Ergebnisse der Evaluation für die Synthese einzelner API- bzw. Umgebungs-Elemente (das heißt Methoden und Parameter).

	Präzision	Ausbeute	F_1
Szenario A	0,823	0,854	0,839
Szenario B	0,920	0,876	0,898
Gesamt	0,886	0,869	0,877

Tabelle 9.7: Ergebnisse der Evaluation der Erzeugung vollständiger API-Aufrufe durch Kombination der Einzelbestandteile.

	Präzision	Ausbeute	F_1
Szenario A	0,589	0,649	0,617
Szenario B	0,711	0,679	0,695
Gesamt	0,668	0,670	0,669

Zuletzt wird die Qualität der synthetisierten Methodenrümpfe (bzw. Skripte) bestimmt. Diese bestehen aus einzelnen Anweisungen (bzw. API-Aufrufen an das Zielsystem); dementsprechend wird die Korrektheit der erzeugten API-Aufrufe bemessen. Das Zielsystem, an das die Aufrufe gerichtet werden, ist ein Haushaltsroboter; es werden nur solche Aufrufe synthetisiert, die der API des Zielsystems entsprechen. Parameter können Objekt der Küchenumgebung sein. Evaluiert werden zwei unterschiedliche Aspekte. Einerseits wird bestimmt, wie gut die einzelnen Bestandteile eines Aufrufs, das heißt die aufzurufende Methode und die (potenziellen) aktuellen Parameter einzeln, synthetisiert werden können; diese Bestimmung der Aufrufbestandteile entspricht der Abbildung einzelner Phrasen der natürlichsprachlichen Beschreibungen auf Zielsystem-Elemente oder Umgebungsobjekte (siehe Abschnitt 7.8.1). Andererseits wird betrachtet, wie gut daraus vollständige API-Aufrufe erzeugt werden können; hierzu werden die zuvor bestimmten einzelnen Bestandteile geeignet kombiniert. Tabelle 9.6 zeigt zunächst die Ergebnisse für die Betrachtung der Einzelbestandteile. Für beide Szenarien werden hohe Werte für die Präzision und die Ausbeute erzielt, wobei die Werte für Beschreibungen zu Szenario B etwas höher sind. Nimmt man die Ergebnisse für beide Szenarien zusammen, wird für das F_1-Maß ein Wert von 0,877 erreicht; die Präzision ist dabei mit einem Wert von 0,886 geringfügig besser als die Ausbeute mit 0,869. Damit sind die hier erzielten Ergebnisse sogar deutlich besser als die in der intrinsischen Evaluation des Agenten zur Synthese von Methodendefinitionen und Skripten diskutierten (siehe Abschnitt 7.8.2).

Dieser Trend zeigt sich auch bei Betrachtung der zusammengesetzten (vollständigen) API-Aufrufe; Tabelle 9.7 zeigt die zugehörigen Werte: Die Ergebnisse fallen zwar zwangsweise schlechter aus als bei Betrachtung der einzelnen Elemente, da ein Aufruf als falsch gewertet wird, sobald ein Bestandteil (z. B. einer der Parameter) inkorrekt ist (siehe Abschnitt 7.8.2). Davon abgesehen können aber dieselben Tendenzen beobachtet werden. Die Generierung von Aufrufen zu Beschreibungen zu Szenario B gelingt besser; insbesondere für die Präzision werden deutlich höhere Werte erreicht (0,589 zu 0,711). Insgesamt wird bei Betrachtung aller Aufrufe für das F_1-Maß ein Wert von 0,669 erzielt, wobei Präzision und Ausbeute nahezu gleichauf sind (0,670 zu 0,668). Auch diese Werte

liegen über denen der intrinsischen Evaluation des Agenten (auch wenn der Unterschied geringer ausfällt). Dieser Ausgang ist insofern überraschend, als die Randbedingungen in der hier diskutierten Evaluation schwieriger sind. Zum einen wurde ein zuvor ungesehener Datensatz verwendet, der aus Beschreibungen zu zwei neuartigen Szenarien besteht. Zum anderen wurden die Ausgaben durch das Gesamtsystem ProNat erzeugt, statt basierend auf perfekten Mustereingaben für den Agenten zur Synthese von Methodendefinitionen; dementsprechend kann es vorkommen, dass andere Agenten oder Fließbandstufen fehlerbehaftete Analysegrundlagen liefern. Unter diesen Randbedingungen wäre eigentlich zu erwarten gewesen, dass die Ergebnisse in der hier vorgestellten Evaluation (zumindest etwas) schlechter ausfallen. Das gegenteilige Resultat ist äußerst positiv zu bewerten. Eine mögliche Erklärung sind die zwei neuen Szenarien; diese könnten so gewählt und beschrieben sein, dass die Probanden weniger Probleme damit hatten, gute Beschreibungen zu verfassen. Eine weitere Erklärung könnten die Auswahl der Probanden sein; möglicherweise waren diese motivierter, gute Beschreibungen zu erstellen. Eine Begründung, die (nahezu) ausgeschlossen werden kann, ist das Sprachvermögen: mit knapp 50% liegt der Anteil der Muttersprachler sogar 10 Prozentpunkte unter dem der zur intrinsischen Evaluation des Agenten verwendeten Online-Datensammlung (siehe Abschnitt 5.5.3.2). Auch wenn sich die verbesserten Ergebnisse möglicherweise nur auf den Versuchsaufbau zurückführen lassen, konnte zumindest gezeigt werden, dass sich die Qualität der erzeugten API-Aufrufe durch die Einbettung in das Gesamtsystem (trotz potenzieller Fehler anderer Sprachanalysen) nicht verschlechtert.

Zusammenfassend konnte die Evaluation zeigen, dass ProNat in der Lage ist, neue Funktionalität anhand von natürlichsprachlichen Beschreibungen zu erlernen; weiterhin konnte gezeigt werden, dass Methodendefinitionen auch durch das Gesamtsystem zuverlässig synthetisiert werden können. Die Erkennung, bei welchen Beschreibungen es sich um den Versuch des Lehrens neuer Funktionalität handelt, funktioniert in einem Großteil der Fälle (85% Genauigkeit). Auch die Synthese der Namen für die Signaturen neu erlernter Methoden erzielt eine sehr gute Genauigkeit von über 90%. Die Erzeugung von API-Aufrufen zur Bildung von Methodenrümpfen funktioniert ebenfalls prinzipiell; mit einem für das F_1-Maß erzielten Wert von 0,669 besteht hier jedoch das größte Verbesserungspotenzial.

9.5 Zusammenfassung und Bewertung

Zur Evaluation des Gesamtsystems wurden drei verschiedene Einzelevaluationen durchgeführt. In der ersten wurden im Zuge einer Online-Studie 237 Pseudo-Quelltext-Generate von 120 Probanden bewertet. Die Auswertung der Antworten hat gezeigt, dass die Mehrheit der Probanden ungefähr die Hälfte aller Generate als vollständig fehlerfrei einschätzen. Die am häufigsten genannte Fehlerart sind nicht wohlgeformte Anweisungen. Die zweite Evaluation lieferte detaillierte Einsichten in die Korrektheit von Quelltext-Generaten: analysiert wurde die Synthese von API-Aufrufen und Kontrollstrukturen auf Anweisungsebene in einer konkreten Anwendungsdomäne. Diese Evaluationen wurde in zwei Varianten durchgeführt. In der ersten wurden – wie bei anderen Evaluationen – händische Transkriptionen als Eingabe verwendet. Bei der zweiten Variante dienen hingegen die Audioaufzeichnungen der stationären Datensammlung als Datengrundlage; diese

Variante kommt somit dem angestrebten Einsatzzweck von ProNat am nächsten: gesprochene Äußerungen werden aufgezeichnet, automatisch transkribiert, interpretiert und schließlich Quelltext generiert. Der Vergleich mit Musterlösungen hat gezeigt, dass die Synthese von API-Aufrufen an ein Zielsystem zumeist gelingt; bei der Untersuchung wurde für das F_1-Maß ein Wert von 0,746 für händische Transkriptionen und 0,687 für Audioaufzeichnungen bestimmt. Kontrollstrukturen werden mit einer Genauigkeit von über 71,3% (51,3%) erzeugt. Im Zuge der zweiten Evaluation wurde zudem die Gesamtlaufzeit des Systems gemessen, die benötigt wird, um aus einer natürlichsprachlichen Äußerung (händische Transkription bzw. Audioaufzeichnung) Quelltext zu generieren: Der Großteil der Quelltexte aus händischen Transkriptionen wird in unter 6 Sekunden erzeugt. Die Verarbeitung von Audioaufzeichnung verlängert die Gesamtlaufzeit von ProNat erheblich, da gesprochene Äußerungen zunächst durch ein System zur automatischen Spracherkennung in Textform überführt werden müssen; die Laufzeit für Audioaufzeichnung beträgt im Mittel 11,2 Sekunden. Mithilfe der dritten Evaluation wurde die Fähigkeit untersucht, neue Funktionalität anhand natürlichsprachlicher Beschreibungen zu erlernen. Die Auswertung hat gezeigt, dass die Erkennung, ob eine Äußerung eine Methodendefinition enthält, zu 85% genau ist. In mehr als 90% der Fälle wird zudem ein passender Methodenname synthetisiert. Lediglich die Synthese von API-Aufrufen zur Bildung der Methodenrümpfe gelingt weniger gut als bei der zweiten Evaluation; für das F_1-Maß wird ein Wert von 0,669 erreicht.

Durch die Evaluationen konnten Fehlerquellen aufgedeckt werden, die vermehrt (und in allen Untersuchungen) auftreten. Unter anderem hat sich gezeigt, dass eine inkorrekte Instruktionserkennung eine wesentliche Ursache für Folgefehler darstellt (siehe Abschnitt 6.2). Werden Äußerungen nicht richtig in Einzelinstruktionen unterteilt, können nicht alle Handlungsanweisung korrekt erfasst werden. In der Folge werden häufig entweder zu viele oder zu wenige Anweisungen für das Quelltext-Generat synthetisiert oder es können nicht alle Parameter eines Aufrufs ermittelt werden. Eine weitere Ursache für fehlende oder falsche Parameter sind falsch aufgelöste Korreferenzen (siehe Abschnitt 7.5). Ein weiteres Problem stellen ungewöhnliche (bzw. unerwartete) Formulierungen in den natürlichsprachlichen Äußerungen dar; sie führen dazu, dass Kontrollstrukturen vereinzelt eine falsche Struktur aufweisen oder dass Handlungsanweisungen nicht auf Zielsystemfunktionen abgebildet werden können. Bei der Verarbeitung von Audioaufzeichnungen führen vor allem Wortfehler, die bei der automatischen Spracherkennung entstehen, zu Problemen, insbesondere bei der Erzeugung von Kontrollstrukturen. Die von ProNat verwendeten Web-APIs zur Spracherkennung werden von den Anbietern kontinuierlich weiterentwickelt, sodass sich zukünftig die Qualität der Generate, die anhand von Audioaufzeichnungen erzeugt werden, noch stärker der Qualität, die ProNat für händische Transkriptionen erzielt, annähern wird.

Betrachtet man die Ergebnisse der Evaluation zusammengenommen, bildet sich das folgende Gesamtbild. Die Evaluationen konnten zeigen, dass Endnutzer-Programmierung mit gesprochener Sprache mit ProNat grundsätzlich möglich ist. Zwar sind nur ungefähr ein Drittel bzw. die Hälfte (je nach Evaluation) aller Quelltext-Generate vollständig korrekt; dieses Ergebnis ist jedoch positiv zu bewerten, schließlich enthalten die Beschreibungen des verwendeten Datensatzes bis zu 24 Handlungsanweisungen, zu denen jeweils ein API-Aufruf synthetisiert werden muss. Verwandte Arbeiten aus dem Bereich der Programmierung mit (geschriebener) natürlicher Sprache ermöglichen meist nur die Erzeugung deutlich kürzerer Skripte (siehe Kapitel 3). Ansätze, die gesprochene

Sprache verarbeiten können, sind nur in der Lage, einzelne oder die Kombination von zwei Anweisungen zu generieren. Dass auch ProNat vor allem kurze Skripte fehlerfrei generiert, ist plausibel, schließlich birgt jede zusätzlich zu synthetisierende Anweisung Fehlerpotenzial. Die guten Ergebnisse bei Betrachtung der einzelnen (in den Skripten enthaltenen) Anweisungen legen aber nahe, dass auch in längeren Skripten meist nur einzelne Anweisungen fehlen oder fehlerhaft sind.

10 Fazit und Ausblick

„We can only see a short distance ahead,
but we can see plenty there that needs to be done.“

– Alan Turing

In dieser Arbeit wurde das System ProNat zur Endnutzer-Programmierung mit gesprochener Sprache entworfen und umgesetzt. Das Motiv für die Entwicklung eines solchen Systems liegt in der wachsenden Menge an Nutzern von Computersystemen, die befähigt werden sollen, diese nicht nur zu nutzen, sondern auch zu programmieren. Die Notwendigkeit, vorab eine Programmiersprache erlernen zu müssen, stellt für Laien eine wesentliche Hürde dar. Daher ermöglicht ProNat stattdessen die Verwendung alltäglicher Sprache zur Beschreibung von Handlungsanweisungen. Die Beschreibungen werden von ProNat als Programm für ein Zielsystem interpretiert, das eine Anwendungsschnittstelle zur Endnutzer-Programmierung anbietet. Um natürlichsprachliche Handlungsbeschreibungen als Programm interpretieren zu können, wird die Semantik der Eingaben analysiert und ein tiefes Verständnis der Sprache erzeugt. Dadurch ist ProNat in der Lage, aus gesprochenen Äußerungen Quelltext für unterschiedliche Zielsysteme zu erzeugen und kann zudem mit geringem Aufwand an neue Anwendungsschnittstellen angepasst werden.

Die Grundlage für das System ProNat bildet die Architektur PARSE. Diese wurde so entworfen, dass sie als Rahmenarchitektur für beliebige Systeme dienen kann, die zur Lösung einer Problemstellung natürliche Sprache verstehen müssen. Als Kernkomponente sieht PARSE die Verwendung unabhängiger Agenten vor, die je einen bestimmten Aspekt der natürlichen Sprache analysieren. Dadurch kann die abstrakte Aufgabe der Erzeugung von Sprachverständnis in handhabbare Teilprobleme unterglieder werden. Die Kapselung in unabhängige Agenten ermöglicht es zudem, je Problemstellung (bzw. Teilaspekt zum Verständnis der Sprache) eine andere Technik zu verwenden. Außerdem können die einzelnen Agenten unabhängig voneinander evaluiert werden.

Die Rahmenarchitektur sorgt dafür, dass die Agenten nebenläufig und wiederholt ausgeführt werden. Dadurch können die Agenten von Analyseergebnissen anderer Agenten profitieren und zyklische Abhängigkeiten können aufgelöst werden. Zur Wahrung der gegenseitigen Unabhängigkeit ist es den Agenten nicht möglich, direkt miteinander zu kommunizieren. Stattdessen kommunizieren sie indirekt über eine geteilte Graph-basierte Datenstruktur. Der Graph enthält einerseits das zu analysierende natürlichsprachliche Artefakt, andererseits hinterlegen die Agenten auch ihre Analyseergebnisse, indem sie den Graph modifizieren (vor allem durch Hinzufügen von Knoten und Kanten). Dadurch können Agenten die Ergebnisse anderer Agenten in ihre Analysen einbeziehen. Im Zusammenspiel erzeugen die Agenten eine semantische Interpretation des natürlichsprachlichen Artefakts. Neben den Agenten sieht PARSE die Verwendung von Fließbändern zur Vor- und

Nachverarbeitung vor. Architektur-seitig sollte die Vorverarbeitung dazu verwendet werden, das zu analysierende natürlichsprachliche Artefakt (Textdokument, Audiosignal etc.) soweit vorzubereiten, dass die Agenten eine Analysegrundlage erhalten. Die Nachverarbeitung dient wiederum der Erzeugung einer Ausgabe; welche Art von Ausgabe generiert wird, hängt von der Problemstellung ab, die ein auf PARSE basierendes System lösen soll. Das in dieser Arbeit entwickelte System ProNat soll beispielsweise Quelltext anhand von gesprochenen Äußerungen synthetisieren.

Zur Umsetzung von ProNat wurden konkrete Agenten und Fließbandstufen für die Rahmenarchitektur PARSE entwickelt. Außerdem werden mithilfe der Rahmenarchitektur Informationen über die Anwendungsdomäne in Form von Ontologien bereitgestellt. Für die Programmierung gegen Anwendungsschnittstellen sind relevante Informationen unter anderem die von Zielsystemen bereitgestellten Schnittstellenfunktionen und Objekte der Systemumgebung. In ProNat werden zur Vorverarbeitung gesprochener Äußerungen Fließbandstufen verwendet, die das Audiosignal zunächst in eine Tokensequenz umwandeln. Anschließend erfolgt eine grundlegende syntaktische Analyse, bevor ein initialer Graph als Analysegrundlage für die Agenten erzeugt wird.

Die für ProNat entwickelten Agenten sind dafür zuständig, eine Äußerung als Programm zu interpretieren. Hierzu wurden insgesamt sechzehn unterschiedliche Agenten entworfen und implementiert. Diese lassen sich grob in drei Kategorien unterteilen: Agenten der ersten Kategorie übernehmen allgemeingültige Semantik-Analyse-Aufgaben, wie beispielsweise Korreferenzanalyse oder Modellierung und Etikettierung von Diskurs-Themen. Weitere Agenten sind für die Erkennung programmatischer Strukturen zuständig; beispielsweise gibt es Agenten, die einzelne Anweisungen synthetisieren oder Methodendefinitionen generieren. Zuletzt gibt es Agenten, welche zur Funktionalität des Systems beitragen, wie beispielsweise der Dialog-Agent oder Funktionswächter-Agenten, die den Fortschritt der Verarbeitung sicherstellen. Die Architektur-seitig sichergestellte Unabhängigkeit der Agenten ermöglicht es, für die Lösung der jeweiligen Problemstellung unterschiedliche Verfahren zu verwenden. Die Agenten zur Synthese von Kontrollstrukturen nutzen beispielsweise händisch erstellte Spezial-Grammatiken. Neuronale Netze und übertragendes Lernen werden vom Agenten zur Erkennung und Analyse von Lehrsequenzen in natürlicher Sprache zur Klassifikation verwendet. Der Agent zur Modellierung und Etikettierung von Themen nutzt hingegen *DBpedia*-Konzepte und -Relationen als Ausgangspunkt für die Inferenz von Themen. Das tiefe Sprachverständnis wird durch ProNat dementsprechend mittels einer Kombination von regel- als auch statistik- und wissensbasierten Techniken verwirklicht. Die Agenten wurden jeweils intrinsisch evaluiert. Die Evaluationen erfolgten anhand von realistischen Eingabebeispielen, die parallel zur Entwicklung der Agenten fortwährend über Nutzerstudien gesammelt wurden. Auf diese Weise kann sichergestellt werden, dass jeder Agent einen Beitrag zur Analyse der Semantik leistet; ein kontinuierlicher Entwicklungsfortschritt wird somit garantiert.

In der Nachverarbeitung wird aus dem gemeinschaftlich erzeugten Analyseergebnis der Agenten zunächst ein abstrakter Syntaxbaum generiert. Der Baum wird anschließend in Quelltext übersetzt und das Generat in eine Zielsystem-spezifische Quelltext-Schablone injiziert. Um ein neues Zielsystem anzubinden, muss ProNat neben der entsprechenden Schablone lediglich die Ontologie bereitgestellt werden, welche die Funktionen des Zielsystems (und gegebenenfalls die Systemumgebung) beschreibt.

Das Zusammenspiel der Komponenten und damit die Funktionalität des Gesamtsystems ProNat wurde abschließend anhand von drei Evaluationen untersucht. In allen Evaluationen wurden von ProNat für eine Reihe von natürlichsprachlichen Programmbeschreibungen Quelltext-Generate erzeugt, die anschließend bewertet wurden.

Die erste Evaluation wurde in Form einer Online-Studie durchgeführt. Zunächst wurden mithilfe des Gesamtsystems zu knapp 240 Transkriptionen Pseudo-Quelltexte (unabhängig von einem Zielsystem) in Form von *UML*-Aktivitätsdiagrammen erzeugt. Die Diagramme wurden anschließend von 120 Probanden bewertet, wobei jedes Diagramm von fünf Probanden begutachtet wurde. Die Auswertung ergab, dass knapp 70% der Diagramme von der Mehrheit der Probanden als korrekt eingeschätzt wurden. Als wesentliche Fehlerquelle konnten nicht wohlgeformte Aktivitäten (die Quelltext-Anweisungen entsprechen) ermittelt werden; diese enthielten entweder überflüssige Phrasen oder wurden nicht korrekt voneinander getrennt.

Für die zweite Evaluation wurde dieselbe Menge an Transkriptionen verwendet. Zusätzlich wurden in einer zweiten Variante die zugehörigen Audioaufzeichnungen als Eingabe genutzt; diese werden im Vorverarbeitungsfließband von ProNat zunächst durch ein System zur automatischen Spracherkennung transkribiert[1]. In beiden Varianten der zweiten Evaluation wurde *Java*-Quelltext zur Steuerung eines Haushaltsroboters generiert und mit einer vorab erstellten Musterlösung verglichen. Der Vergleich ergab, dass für die Erzeugung von Anweisungen und die Abbildung dieser auf Funktionen der Anwendungsschnittstelle für das F_1-Maß ein Wert von 0,746 für händische Transkriptionen und 0,687 für die Audioaufzeichnungen erreicht wird. Außerdem wurden Kontrollstrukturen in gut 71% (bzw. 51% für bei Verwendung von Audioaufzeichnungen) der Fälle korrekt synthetisiert.

In der letzten Evaluation wurden 100 natürlichsprachliche Beschreibungen betrachtet, die potenziell eine neue Systemfunktionalität lehren. Zunächst wurde untersucht, wie gut ProNat einfache (nicht lehrende) Handlungsbeschreibungen von Lehrsequenzen unterscheiden kann; dies gelingt für 85 der 100 Beschreibungen. Anschließend wurden die Lehrsequenzen näher untersucht und ermittelt, dass ProNat in über 90% der Fälle in der Lage ist, einen sprechenden Methodennamen für die neue Funktionalität zu synthetisieren. Auch der Aufruf der neuen Funktionen anhand der generierten Methodennamen gelingt mittels natürlichsprachlicher Handlungsaufforderungen in gut 81% der Fälle. Die Abbildung von Anweisungen auf Zielsystemfunktionen zur Bildung von Methodenrümpfen erreichte für das F_1-Maß einen Wert von 0,669.

Auch wenn einzelne Bestandteile noch Verbesserungspotenzial aufweisen, konnte anhand der drei Evaluationen nachgewiesen werden, dass ProNat grundsätzlich in der Lage ist, Programme aus gesprochenen Äußerungen zu synthetisieren. Weitere Einsichten zur Statthaftigkeit des in dieser Arbeit verfolgten Ansatz zur Endnutzer-Programmierung mit gesprochener Sprache liefert die nachfolgende Diskussion der eingangs formulierten Thesen.

[1] Für die Evaluation wurde die *Google Speech API* verwendet (siehe Abschnitt 6.1).

10.1 Diskussion der Thesen

Zu Beginn der Arbeit wurden in Abschnitt 1.6 Thesen aufgestellt, anhand derer bewertet werden soll, inwieweit die Zielstellung, Laien zur Endnutzer-Programmierung mit gesprochen Sprache zu befähigen, mit dem in den vorangegangen Kapiteln beschriebenen Ansatz erfüllt wird. Im Folgenden werden die einzelnen Thesen diskutiert und jeweils Belege angeführt, die in der Arbeit gesammelt wurden. Die Teile der Arbeit, die vorrangig Belege liefern, sind einerseits die Evaluation des Gesamtsystems (siehe Kapitel 9), andererseits die intrinsischen Evaluationen der einzelnen ProNat-Agenten (siehe Kapitel 7), aber auch die Beschreibung des grundsätzlichen Systemaufbaus (siehe Kapitel 5) bzw. der Architektur (siehe Kapitel 4).

T_1: Aus gesprochenen Äußerungen kann Quelltext zur Programmierung von Zielsystemen mit einer Endnutzer-Anwendungsschnittstelle synthetisiert werden.

Das System ProNat ist in der Lage, natürlichsprachliche, gesprochene Handlungsbeschreibung derart zu interpretieren, dass programmatische Anweisungen erzeugt werden können (siehe Abschnitt 7.8). Der für diese Aufgabe zuständige Agent zur Synthese von Methodendefinitionen und Skripten (siehe Abschnitt 7.8) erzeugt eine Zwischenrepräsentation, die den strukturellen Aufbau von Anweisungen widerspiegelt. Anhand dieser Zwischenrepräsentation gelingt es einerseits, größtenteils korrekte Pseudo-Quelltext-Anweisungen zu generieren (siehe Abschnitt 9.2), andererseits gelingt in den meisten Fällen auch, eine Abbildung auf Funktionen (inklusive aktueller Parameter) einer Anwendungsschnittstelle zu finden. Letzteres konnte durch die intrinsische Evaluation des besagten Agenten (siehe Abschnitt 7.8.2) und der zweiten Evaluation des Gesamtsystems nachgewiesen werden (siehe Abschnitt 9.3). Nichtsdestotrotz birgt insbesondere das Verfahren zur Abbildung auf Zielsystemfunktionen Verbesserungspotenzial. Weiterhin konnte innerhalb der Beschreibung des Nachverarbeitungsfließbandes (siehe Kapitel 8) gezeigt werden, dass ProNat Quelltext für verschiedene Anwendungsschnittstellen generieren kann; auch unterschiedliche Programmiersprachen werden unterstützt.

$T_{1.1}$: Anhand gesprochener Handlungsbeschreibungen können skriptartige Programme, die aus Aufrufen von Funktionen einer Anwendungsschnittstelle bestehen, erzeugt werden.

ProNat gelingt es, in den meisten Fällen zu entscheiden, wann aus einer Handlungsbeschreibung ein Skript erzeugt werden soll (siehe Abschnitte 7.7.3 und 9.4). Die erzeugten Skripte bestehen – im Gegensatz zu den von verwandten Arbeiten erzeugten – aus einer Vielzahl von Anweisungen (siehe Abschnitt 9.3). Die Abbildung der Anweisungen auf konkrete Zielsystemfunktionen wurde bereits im Zuge der Diskussion der These T_1 thematisiert.

$T_{1.2}$: Anhand lehrender Beschreibungen können Methodendefinitionen synthetisiert und dadurch der Funktionsumfang der Anwendungsschnittstelle eines Zielsystems erweitert werden.

ProNat ist in den meisten Fällen in der Lage, zu erkennen, wann ein Nutzer beabsichtigt, dem System eine neue Funktion zu lehren (siehe Abschnitte 7.7.3 und 9.4). Aus erkannten

Lehrsequenzen gelingt es zudem, Methodendefinitionen zu generieren. Die Erzeugung von Methodendefinitionen gelingt zuverlässig (siehe siehe Abschnitte 7.8.2 und 9.4): es werden größtenteils sprechende Methodennamen generiert. Diese sind wiederum eine Voraussetzung dafür, dass neu definierte Funktionen in gesprochenen Äußerungen erkannt und aufgerufen werden können. Es konnte gezeigt werden, dass der Aufruf der neu definierte Funktionen anhand von gesprochenen Handlungsanweisungen zuverlässig gelingt. Die Synthese von Methodenrümpfen entspricht im Wesentlichen der von Skripten (siehe $T_{1.1}$).

$T_{1.3}$: Es können automatisch Strukturen in natürlicher Sprache erkannt werden, welche die Verwendung von Kontrollstrukturen in einem Programm erfordern: Schleifen, bedingte Verzweigungen und nebenläufige Abschnitte können in die erzeugten Quelltexte integriert werden.

Dem System gelingt es in vielen Fällen, Kontrollstrukturen innerhalb gesprochener Äußerungen zu erkennen und in die synthetisierten Programme zu integrieren. Dabei machen die verfolgten Ansätze keine Annahmen über das Vorhandensein oder die Position der Kontrollstrukturen. Zudem können Formulierungen zur Beschreibung von Kontrollstrukturen vielfältig gestaltet werden: Unterschiedliche sprachliche Konstruktionen, welche die Verwendung verschiedener Kontrollstrukturen erfordern, werden zuverlässig erkannt. Dies konnten sowohl die intrinsischen Evaluationen der Agenten zur Synthese von Kontrollstrukturen (siehe Abschnitt 7.6) als auch die zweite Evaluation des Gesamtsystems belegen (siehe Abschnitt 9.3). Eine Einschränkung des derzeitigen Ansatzes ist, dass Bedingungen auf Schnittstellenfunktionen des Zielsystems (oder auf Laufvariablen bei Zählschleifen) abbildbar sein müssen. Außerdem hat sich gezeigt, dass eine der wesentlichen Herausforderungen bei der Erkennung von Kontrollstrukturen in natürlicher Sprache die Bestimmung des Bezugsrahmens (das heißt die Zuweisung von Anweisen zu Blöcken) darstellt.

$T_{1.4}$: Die Interpretation einer gesprochenen Äußerung als Programm kann größtenteils unabhängig vom konkreten Zielsystem durchgeführt werden. Dadurch ist der Aufwand zur Adaption an ein neues Zielsystem gering.

Der beschriebene Systemaufbau von ProNat zeigt, dass die Vorverarbeitung, sowie die vollständige semantische Analyse einer Äußerung und die Interpretation als Programm durch die Agenten vollständig unabhängig von einem konkreten Zielsystem durchgeführt werden können. Die Sprachanalysen machen weder Annahmen hinsichtlich der Funktionalität des Zielsystems noch zu anderen Domänencharakteristiken. Dadurch ist ProNat in der Lage, ohne Angabe eines konkreten Zielsystems, bereits Pseudo-Quelltext zu generieren (siehe Abschnitt 9.2). Zudem konnte gezeigt werden, dass die Anbindung eines (neuen) Zielsystems mit geringem Aufwand bewerkstelligt werden kann: Für jedes Zielsystem (und jede Systemumgebung) muss lediglich eine Ontologie bereitgestellt werden, die als Schnittstelle zu Domäneninformationen dient. Wie bereits von Landhäußer et al. gezeigt, können Zielsystemontologien (teil-)automatisch generiert werden [LWT17b]. Zusätzlich muss den entsprechenden Stufen des Nachverarbeitungsfließbandes ein Besucher je Programmiersprache und eine Quelltext-Schablone pro Zielsystem zur Verfügung gestellt

werden. Das übrige System kann unverändert weiterverwendet werden. Zuletzt konnte gezeigt werden, dass ProNat in der Lage ist, bei Bereitstellung mehrerer Zielsystem- und Umgebungsontologien zur Laufzeit diejenigen zu bestimmen, welche im aktuellen Kontext passend sind; das bedeutet, unter einer Auswahl von Zielsystemen, kann ProNat anhand der natürlichsprachlichen Äußerung entscheiden, welches Zielsystem angesprochen werden sollte (siehe Abschnitt 7.9).

T_2: Die Verwendung unabhängiger Agenten für einzelne Sprachanalyseaufgaben, die nebenläufig und wiederholt ausgeführt werden, erzeugt bei geeigneter Auswahl der Agenten ein tiefes Verständnis gesprochener Sprache.

Wie soeben diskutiert, ist das von ProNat erzeugte Verständnis natürlicher Sprache ausreichend tief, um die Programmierung unterschiedlicher Zielsysteme anhand gesprochener Äußerungen zu ermöglichen. Davon abgesehen, ist nur schwer abzuschätzen, wie umfangreich das Sprachverständnis tatsächlich ist, das die in Kapitel 7 beschriebenen Analysen der Agenten im Zusammenspiel erzeugen. Die von der Architektur vorgesehene Untergliederung in unabhängige Agenten ermöglicht es allerdings, zu bemessen, wie gut einzelne (semantische) Aspekte der Sprache „verstanden“ werden. Außerdem können dem System jederzeit Agenten hinzugefügt werden, um das allgemeine Sprachverständnis zu vertiefen. Die Möglichkeit durch die nebenläufige Ausführung der Agenten zyklische Abhängigkeiten aufzulösen und dadurch das Sprachverständnis zu vertiefen, konnte vereinzelt gewinnbringend angewandt werden, beispielsweise im Zusammenspiel der Agenten zur Korreferenzanalyse, zur Modellierung des sprachlichen Kontexts und zur Synthese von Kontrollstrukturen (siehe Abschnitte 7.5, 7.4 und 7.6). Aufschluss über die allgemeinen Sprachverständnisfähigkeiten von ProNat könnte eine (teilweise) Übertragung des Ansatzes in Anwendungsdomänen abseits der Programmierung liefern, wie im anschließenden Kapitel diskutiert.

T_3: Durch gezielte Dialogführung können Unklarheiten bei der Interpretation einer gesprochenen Äußerungen beseitigt werden.

Das Potenzial gezielter Dialogführung bei der Interpretation von gesprochenen Äußerungen wurde in der Arbeit exemplarisch gezeigt. Betrachtet wurden Möglichkeiten zur Korrektur von Spracherkennungsfehlern, fehlerhaften Korreferenzen und unvollständigen bedingten Verzweigungen. Für alle drei Fehlerarten konnte gezeigt werden, dass Unklarheiten (und damit potenzielle Fehlerquellen) im Dialog mit dem Nutzer ausgeräumt werden können (siehe Abschnitt 7.10.3). Die wesentliche Herausforderung bei der Dialogführung besteht darin, Konzepte, die Laien im Allgemeinen unbekannt sind (wie bedingte Verzweigungen oder Korreferenzen) derart zu erfragen, dass Laien die Fragestellung verstehen können. Mit dem Konzept eines Dialog-Agenten, der sogenannte *Gesprächssitationen* verwendet, um Dialogstrategien für bestimme Fehlerarten darzustellen, konnte zudem eine Möglichkeit demonstriert werden, wie Dialog-Management offen und erweiterbar gestaltet werden kann.

10.2 Weiterführende Themen und Ausblick

Die abschließende Diskussion der Eignung des Systems ProNat zur Endnutzer-Programmierung mit gesprochener Sprache anhand der aufgestellten Thesen zieht konsequenterweise die Betrachtung weiterführenden Forschungsfragen nach sich. Diesbezüglich werden im Folgenden zunächst Möglichkeiten zur Optimierung des Systems ProNat bzw. der Architektur PARSE erörtert. Anschließend wird ein Ausblick auf weiterführende Fragestellungen zur Weiterentwicklung des Ansatzes gegeben. Zuletzt werden alternative Einsatzmöglichkeiten abseits der Programmierung diskutiert.

10.2.1 Optimierungen und Erweiterungen

Damit ProNat die Quelltext-Synthese aus gesprochenen Äußerungen zukünftig noch besser gelingt, könnten einzelne Agenten optimiert oder ausgetauscht werden. Zunächst könnten die vorgestellten Verfahren verbessert bzw. erweitert werden. Unter anderem könnten die zur Erkennung von Kontrollstrukturen eingesetzten Grammatiken erweitert werden, um weitere syntaktische Konstrukte, die auf eine Kontrollstruktur hinweisen, erkennen zu können. Außerdem wäre es möglich, die Ergebnisse anderer Agenten noch stärker in die Sprachanalysen einfließen zu lassen; beispielsweise könnte der Dialog-Agent Themen-Etiketten verwenden, um Rückfragen zu präzisieren. Ist zum Beispiel bekannt, dass eines der Diskurs-Themen *kitchen* ist, könnte bei einer unsicheren Worterkennung, wie in der beispielhaften Hypothese „$\text{go}_{0,9}$ $\text{to}_{1,0}$ $\text{the}_{0,9}$ $\textit{french}_{0,3}$“[2], direkt folgende Frage (anstelle einer generischen Frage, die Unverständnis ausdrückt) gestellt werden: „Do you mean *fridge*?“.

Darüber hinaus könnten Agenten auch gänzlich ausgetauscht und neue Verfahren zur Lösung der jeweiligen Problemstellung entwickelt werden[3]. Vor allem Verfahren, die überwachtes maschinelles Lernen verwenden, könnten für Problemstellungen, die sich als Klassifikationsaufgabe auffassen lassen, eine Qualitätssteigerung bringen. Hierzu zählen beispielsweise die Synthese von Kontrollstrukturen und die Erkennung von Instruktionen. Gerade letztere hat sich in den Evaluationen als wesentliche Fehlerquelle herausgestellt. Für beide Aufgaben stehen bisher keine annotierten Datensätze zur Verfügung, wie sie für überwachtes Lernen benötigt werden. Allerdings existieren durch die Problemdefinition bereits Annotationsschemata. Gemäß dieser müssten zunächst umfangreiche annotierte Korpora angelegt werden, bevor beispielsweise ein neuronales Netz auf diese Aufgabe trainiert werden kann. Um den Aufwand für die Annotation zu verringern, wäre eine Möglichkeit, die vorhandenen regelbasierten Ansätze zu verwenden, um eine Dokumentenmenge (beispielsweise das *Switchboard*-Korpus) mit Etiketten zu versehen und diese anschließend manuell zu überprüfen bzw. zu korrigieren.

[2] Die tiefgestellten Zahlen stellen (Wort-)Konfidenzen dar, die üblicherweise von Systemen zur automatischen Spracherkennung bei der Erzeugung einer Hypothese generiert werden (siehe Abschnitte 2.3.7 und 6.1)

[3] Sofern die Analyseergebnisse in gleicher Form dargestellt werden wie zuvor, können (durch die Architektur-seitig sichergestellte Unabhängigkeit) Agenten, die dieselbe Problemstellung mithilfe unterschiedlicher Ansätze lösen, beliebig ausgetauscht oder sogar gleichzeitig eingesetzt werden.

Auch die Entwicklung neuer Agenten, die weitere sprachliche oder programmatische Aspekte betrachten, könnten die Interpretation gesprochener Äußerungen verbessern. Hilfreich wäre beispielsweise eine Überprüfung, ob die beschriebenen Handlungsanweisungen ein sinnhaftes Skript für ein Zielsystem erzeugen, oder ob Anweisungen zur Erfüllung der Aufgabe fehlen. Falls eine solche Lücke entdeckt würde, könnte ein weiterer Agent mögliche fehlende Funktionsaufrufe automatisch inferieren. Ein derartiger Agent wurde bereits für ProNat entwickelt [Ham18]; dieser stützt seine Inferenz auf Konzepte, Fakten und Axiome der Weltwissenontologie *ResearchCyc*, funktioniert jedoch nur sehr eingeschränkt. Das in *ResearchCyc* enthaltene Wissen über Abläufe und Vor- und Nachbedingungen ist schlicht nicht umfangreich genug. In einem alternativen Ansatz könnten zukünftig stattdessen Vor- und Nachbedingung von Schnittstellenfunktionen des Zielsystems verwendet werden, sofern die Anwendungsschnittstelle solche anbietet.

Neben Verbesserungen der Bestandteile des Systems ProNat, könnten auch einzelne Aspekte der Architektur PARSE optimiert werden. Ein offensichtlicher Ansatzpunkt ist der nebenläufige Zugriff der Agenten auf den PARSE-Graphen als geteilte Datenstruktur. Derzeit erhalten die Agenten zu Beginn jedes Aufrufzyklus eine Kopie des Graphen und führen ihre Analysen auf dieser Kopie aus. Nur der Agent, der seine Analyse zuerst vollständig abschließt (und neue Informationen generiert), darf seine Ergebnisse im Haupt-Graphen hinterlegen. Zukünftig könnte ein echt paralleler Mechanismus für den Zugriff auf den Graphen entwickelt werden; dann könnten die Agenten direkt auf dem Haupt-Graph arbeiten und auch Ergebnisse direkt in diesem hinterlegen. Auf diese Weise könnte die Effizienz der Analyse der Semantik durch die Agenten deutlich gesteigert werden. Allerdings müssen (schreibende) Zugriffe auf den Graphen entsprechend geregelt werden, um Inkonsistenzen zu verhindern. Außerdem würden die Agenten auch während eines Analysezyklus über Änderungen am Graphen informiert werden und müssen demnach fortwährend entscheiden, ob die Analyse unter den neuen Voraussetzungen fortgesetzt oder neu begonnen werden sollte. Auch hierfür sollte die Architektur unterstützende Mechanismen anbieten.

Eine weitere Unterstützung zur Entwicklung von Agenten, die Architektur-seitig ergänzt werden könnte, ist die Prüfung auf im Graphen enthaltene Informationen, die der jeweilige Agent für seine Analysen benötigt. Bisher obliegt die Prüfung den Agenten und wurde dementsprechend vielseitig umgesetzt. Es wäre aber möglich, vorzugeben, dass eine Prüfung erfolgen muss und in welcher Form diese erfolgen soll. Beispielsweise könnte jeder Agent Graphmuster angeben, die beschreiben, welche Informationen er benötigt (ähnlich der Verwendung von Mustern im Dialog-Agenten oder in der Nachverarbeitungsstufe zur Erzeugung des abstrakten Syntaxbaums). Dieser Mechanismus könnte auch mehrstufig realisiert werden; es könnte zwischen *mindestens erforderlichen* und *optionalen* Information unterschieden werden.

10.2.2 Weiterführende Fragestellungen

Sowohl der System-Entwurf als auch die Architektur bieten Raum für weiterführender Untersuchungen als sie im Rahmen dieser Arbeit umgesetzt werden konnten. Zunächst sollte untersucht werden, ob bestimmte Sprachanalysen noch stärker von der nebenläufigen und wiederholten Ausführung

der Agenten profitieren könnten. Die bisher für ProNat entwickelten Agenten nutzen die Möglichkeiten, die sich aus diese Mechanik bieten, nur vereinzelt. Das liegt weniger daran, dass es nicht nötig wäre, bereits Teilergebnisse verwenden oder zyklische Abhängigkeiten auflösen zu können; vielmehr hat sich gezeigt, dass die Entwicklung der Agenten deutlich an Komplexität gewinnt, wenn kontinuierlich geprüft werden muss, ob neue Ergebnisse anderer Agenten vorliegen oder der Graph bereits Teilergebnisse der eigenen Problemstellung enthält. Um diesen Aufwand zu minimieren und eine einheitliche Lösung für diese Problematik anzubieten, könnte die Architektur zukünftig um unterstützende Mechanismen erweitert werden, wie beispielsweise der zuvor beschriebene Mechanismus zur Prüfung auf im Graphen enthaltene Informationen. Das Potenzial der nebenläufigen und wiederholten Ausführung der Agenten ist sicherlich größer als in dieser Arbeit gezeigt werden konnte. Möglicherweise können andere Systeme, die auf PARSE basieren (wie im nachfolgenden Paragraphen diskutiert), diesbezüglich zukünftig Hinweise liefern.

Weiterhin sollte untersucht werden, ob die Konfidenzen, die viele Agenten samt alternativen Analyseergebnissen erzeugen, gewinnbringend genutzt werden können. Im aktuellen Zustand verwendet ProNat zumeist nur die Analyseergebnisse mit der höchsten Konfidenz (das heißt die Haupthypothese der Agenten). Gerade im Zusammenspiel der Agenten und voneinander abhängigen Sprachanalysen könnte eine Exploration der Alternativergebnisse mit niedrigerer Konfidenz aber hilfreich sein. Eine erste Untersuchung dieser Möglichkeit konnte bereits Hinweise darauf liefern, dass eine Exploration der Alternativen im Zusammenspiel der Agenten zur Disambiguierung von Wortbedeutungen, zur Modellierung und Etikettierung von Diskurs-Themen und zur automatischen Auswahl von Zielsystem- und Umgebungsontologien die Güte der Ergebnisse der beiden letztgenannten Agenten steigern kann [Fuc20].

Auch die Möglichkeiten geschickter Dialogführung sollten eingehender untersucht werden. Bisher wurde nur gezeigt, dass bestimmte Arten von Unklarheiten im Dialog ausgeräumt werden können; es wäre interessant zu sehen, ob dieses Konzept auf alle potenziellen Fehlerquellen, die bei der Beschreibung eines Programms mit natürlicher Sprache auftreten können, erweitert werden kann. Darüber hinaus könnten auch weiterführende Konzepte der Dialogführung untersucht werden. Beispielsweise könnte die Dialogführung stärker System-initiativ (statt wie bisher Nutzer-initiativ) ausgelegt werden und evaluiert werden, ob sich die Akzeptanz der Nutzer dadurch verbessert. In diesem Zusammenhang sollte auch untersucht werden, ob häufigere Rückmeldungen des Systems einen positiven Effekt erzielen; beispielsweise könnte das System, nachdem es eine Äußerung vollständig analysiert hat, dem Nutzer Rückmeldung darüber geben, wie es die Handlungsanweisungen interpretiert hat, bevor das Programm tatsächlich ausgeführt wird. Ein ähnlicher Mechanismus könnte auch für das Erlernen neuer Funktionen eingesetzt werden.

10.2.3 Alternative Einsatzgebiete

Neben Verbesserung und weiterführenden Untersuchungen des Systems ProNat bzw. der Architektur PARSE, sollte untersucht werden, ob die Architektur (und möglicherweise Teile von ProNat) auch für andere Aufgaben abseits der Endnutzer-Programmierung mit gesprochener Sprache eingesetzt werden können. Wie in Kapitel 4 erwähnt, nutzen bereits zwei weitere Projekte die Architektur.

Beide Projekte betrachten Problemstellungen aus gänzlich unverwandten Forschungsgebieten: Das Projekt *INDIRECT* widmet sich der Generierung von Rückverfolgbarkeitsinformationen zwischen Anforderungsdokumenten und Quelltext [Hey19]. Hierzu soll basierend auf PARSE, eine semantische Interpretation der Anforderungstexte erzeugt und mit einem semantischen Modell von Quelltext, das ebenfalls als PARSE-Graph repräsentiert wird, abgeglichen werden; letztlich sollen Verbindungen zwischen beiden Modellen geknüpft und so die Rückverfolgbarkeit gewährleistet werden. Das zweite Projekt prüft hingegen die Konsistenz von Modellbeschreibungen und zugehörigen konkreten Softwarearchitekturen. Auch hier werden die Beschreibungen mit dem PARSE-Ansatz analysiert und die Interpretation mit Architekturbestandteilen verglichen [KK19; KSK19]. Beide Projekte basieren nicht nur auf der PARSE-Architektur, sondern verwenden auch einzelne Fließbandstufen und Agenten des Systems ProNat wieder. Unter anderem nutzen die Projekte die Vorverarbeitungsstufen zur seichten Sprachverarbeitung und zur Erkennung semantischer Rollen sowie die Agenten zur Modellierung des sprachlichen Kontextes und zur Korreferenzanalyse (teilweise in modifizierter Form).

ProNat könnte zukünftig außerdem beim interaktiven Lernen Verwendung finden. Unter anderem könnte ProNat dazu eingesetzt werden, Systemen ihre Umgebung zu beschreiben. Dies ist nützlich, wenn Systeme, die mit ihrer Umgebung interagieren, wie beispielsweise ein Robotersystem, in einer neuen Umgebung eingesetzt werden sollen (für die keine Modellierung vorhanden ist). Zum Beispiel könnte einem Haushaltsroboter erklärt werden, dass er sich in einer *Küche* befindet, dass es in der Küche *Haushaltsgeräte* gibt (und welche das sind). Außerdem könnte beschrieben werden, wo die jeweiligen Umgebungsobjekte platziert sind. Auf diese Weise könnte schrittweise eine Umgebungsontologie befüllt werden. Zur Umsetzung dieser Idee müssten die Sprachanalysen von ProNat mit Analyseergebnissen aus anderen Quellen, wie beispielsweise Bild- und Gestenerkennung, verknüpft werden. Entsprechende Analyseverfahren für die weiteren Informationsquellen müssten zunächst entwickelt werden. Die Verknüpfung multimodaler Informationen wird aber bereits von der Architektur PARSE unterstützt; wie in Abschnitt 4.2.2 diskutiert, wurden Graphen als Datenmodell unter anderem aus dem Grund gewählt, dass Informationen aus anderen Quellen einfach eingebunden werden können.

Einen Hinweis auf weitere mögliche Einsatzgebiete liefert auch die bereits in ProNat umgesetzte Erzeugung von *UML*-Aktivitätsdiagrammen. Diese Möglichkeit zeigt, dass mithilfe von ProNat die Inhalte natürlichsprachlicher Äußerungen als formale Modelle dargestellt werden können. Davon ausgehend ergeben sich zahlreiche Anwendungsmöglichkeiten. Beispielsweise könnte ProNat genutzt werden, um automatisch Zusammenfassungen von Besprechungen in Form eines semantischen Modells (z. B. einer Ontologie) zu erstellen; Konzepte, die wiederholt von Teilnehmern geäußert werden, könnten erkannt und Relationen hergestellt werden. Außerdem könnte ein solches Modell durch eine Verknüpfung mit einer Wissensdatenbank mit Weltwissen angereichert werden. Dadurch würde eine Art *Knowledge Graph* zum Thema der Besprechung entstehen. Dieser könnte auch über mehrere Besprechungen hinweg erweitert oder mit einem bereits vorhanden Modell verknüpft werden.

10.3 Abschließende Bemerkungen

Programmieren für alle! Systeme zur Programmierung mit Alltagssprache haben das Potenzial, dieses ambitionierte Ziel eines Tages zu erreichen. In dieser Arbeit konnte gezeigt werden, dass es grundsätzlich möglich ist, von Laien umgangssprachlich formulierte Beschreibungen als Programm zu interpretieren und Systeme dadurch nicht nur zu steuern, sondern auch deren Funktionsumfang zu erweitern. Auf diese Weise könnten zukünftig die unzähligen Endnutzer von Computersystemen dazu befähigt werden, zumindest einfache Programme für ihre Smartphones, Laptops oder Tablet-PCs zu erstellen. Der in dieser Arbeit verfolgte Ansatz ist nur ein möglicher unter vielen vielversprechenden, die derzeit und zukünftig entwickelt werden. Welcher schließlich dazu führt, dass wirklich jeder sein persönliches Computersystem programmieren kann, ist unerheblich. Wichtig ist in erster Linie, dass die Forschungsgemeinde dieses Ziel auch zukünftig verfolgt, oder – um es mit den Worten von Johann Wolfgang von Goethe zu sagen – das Spiel trotz fallender Spielfiguren zu Ende spielt:

> „*Es ist mit Meinungen, die man wagt, wie mit Steinen, die man voran im Brette bewegt: Sie können geschlagen werden, aber sie haben ein Spiel eingeleitet, das gewonnen wird.*“

Anhang

A Etikettensätze

In den nachfolgenden Abschnitten werden die vollständigen Etikettensätze aufgeführt, wie sie zur Erkennung von Wortarten, zur Bestimmung syntaktischer Abschnitte, zur syntaktischen Zerteilung und für die Erkennung semantischer Rollen verwendet werden.

A.1 Wortarten

Nachfolgend sind Wortartetiketten, wie in der *Penn Treebank* definiert, aufgelistet [TMS03].

Etikett	Bedeutung (englisch)	Bedeutung (deutsch)
CC	Coordinating conjunction	Nebenordnende Konjunktion
CD	Cardinal number	Kardinalzahl
DT	Determiner	Determinativ (Artikel)
EX	Existential there	Existenzielles *there*
FW	Foreign word	Fremdwort
IN	Preposition or subordinating conj.	Präposition oder unterordnende Konj.
JJ	Adjective, positive	Adjektiv, Positiv (Grundform)
JJR	Adjective, comparative	Adjektiv, Komparativ
JJS	Adjective, superlative	Adjektiv, Superlativ
LS	List item marker	Listeneintragsmarkierung
MD	Modal	Modalverb
NN	Noun, singular or mass	Nomen, Singular oder Kontinuativum
NNS	Noun, plural	Nomen, Plural
NNP	Proper noun, singular	Eigenname, Singular
NNPS	Proper noun, plural	Eigenname, Plural
PDT	Predeterminer	Prädeterminativ
POS	Possessive ending	Possessivendung
PRP	Personal pronoun	Personalpronomen
PRP$	Possessive pronoun	Possesivpronomen
RB	Adverb, positive	Adverb, Positiv (Grundform)

Etikett	Bedeutung (englisch)	Bedeutung (deutsch)
RBR	Adverb, comparative	Adverb, Komparativ
RBS	Adverb, superlative	Adverb, Superlativ
RP	Particle	Partikel
SYM	Symbol	Symbol
TO	infinitival *to*	Infinitivisches *to*
UH	Interjection	Interjektion
VB	Verb, base form	Verb, Grundform
VBD	Verb, past tense	Verb, Präteritum
VBG	Verb, gerund or present participle	Verb, Gerundium oder Partizip I
VBN	Verb, past participle	Verb, Partizip II
VBP	Verb, non-3rd person singular present	Verb, 1. o. 2. Person Singular Präsens
VBZ	Verb, 3rd person singular present	Verb, 3. Person Singular Präsens
WDT	*Wh*-determiner	*Wh*-Determinativ (Artikel)
WP	*Wh*-pronoun	*Wh*-Pronomen
WP$	Possessive *wh*-pronoun	Possessiv-*Wh*-Pronomen
WRB	*Wh*-adverb	*Wh*-Adverb
#	Pound sing	Pfund-Zeichen
$	Dollar sing	Dollar-Zeichen
.	Sentence-final punctuation	Satz-beendendes Satzzeichen
,	Comma	Komma
:	Colon or semi-colon	Doppelpunkt oder Semikolon
(	Left bracket character	Öffnendes Klammer-Zeichen
)	Right bracket character	Schließendes Klammer-Zeichen
"	Straight double quote	Gerade doppelte Anführungszeichen
‘	Left open single quote	Einfaches öffn. Anführungszeichen
“	Left open double quote	Doppeltes öffn. Anführungszeichen
’	Right close single quote	Einfaches schließ. Anführungszeichen
”	Right close double quote	Doppeltes schließ. Anführungszeichen

A.2 Chunks

Nachfolgend ist der Etikettensatz zur Darstellung syntaktischer *Chunks* aufgeführt [TB00].

Etikett	Bedeutung (englisch)	Bedeutung (deutsch)
ADJP	Adjective phrase	Adjektivphrase
ADVP	Adverb phrase	Adverbialphrase
CONJP	Conjunction phrase	Konjunktionsphrase
INJ	Interjection	Interjektion
LST	List marker	Listenmarkierungszeichen
NP	Noun phrase	Nominalphrase
PP	Prepositional phrase	Präpositionalphrase
PRT	Particles	Partikel
SBAR	Subordinate clause	Nebensatz
UCP	Unlike coordinated phrase	Ungleichartige nebengeord. Phrase
UNDEF	Undefined	Nicht definiert
VP	Verb phrase	Verbalphrase

A.3 Syntaktische Zerteilung

Nachfolgend ist der Etikettensatz zur Darstellung syntaktischer Kategorien, wie in der *Penn Treebank* definiert, aufgeführt [TMS03].

Etikett	Bedeutung (englisch)	Bedeutung (deutsch)
ADJP	Adjective phrase	Adjektivphrase
ADVP	Adverb phrase	Adverbialphrase
NP	Noun phrase	Nominalphrase
PP	Prepositional phrase	Präpositionalphrase
S	Simple declarative clause	Einfacher Deklarativsatz
SBAR	Subordinate clause	Nebensatz
SBARQ	Direct question introduced by *wh*-element	Direkte Frage, eingeleitet durch ein *wh*-Element
SINV	Declarative sentence with subject-aux inversion	Deklarativsatz mit Subjekt-Hilfsverb-Umkehrung
SQ	Yes/no questions and subconstituent of SBARQ excluding *wh*-element	Ja-Nein-Fragen und Teilkonstituent von SBARQ, ausgeschlossen *wh*-Elemente
VP	Verb phrase	Verbalphrase

Etikett	Bedeutung (englisch)	Bedeutung (deutsch)
WHADVP	*Wh*-adverb phrase	*Wh*-Adverbialphrase
WHNP	*Wh*-noun phrase	*Wh*-Nominalphrase
WHPP	*Wh*-prepositional phrase	*Wh*-Präpositionalphrase
X	Constituent of unknown or uncertain category	Konstituent unbekannter oder unklarer Kategorie
*	"Understood" subject of infinitive or imperative	Implizites Subjekt eines Infinitives oder Imperatives
0	Zero variant of that in subordinate clauses	Nebensatz mit Auslassung von *that*
T	Trace of *wh*-Constituent	Spur eines *wh*-Konstituenten

A.4 Semantische Rollen

Nachfolgend ist der Etikettensatz zur Repräsentation semantischer Rollen aufgeführt [CM04; CM05]. Die Etiketten werden in drei Typen unterteilt: eine *V*-Etikette zur Markierung von (Prädikats-)Verben, *A**-Etiketten zur Markierung von Phrasen, die von einem Prädikatsverb abhängen (bzw. diesem zugeordnet werden), und *R**-Etiketten zur Markierung von Referenzen auf Phrasen, die einer der *A**-Rollen zugeordnet wurden.

Etikett	Bedeutung (englisch)	Bedeutung (deutsch)
V	verb	Verb (Prädikat)
A0	agent	Handelnder (Agent)
A1	1st patient or theme	1. Patiens oder Thema
A2	2nd patient or theme	2. Patiens oder Thema
A3	3rd patient or theme	3. Patiens oder Thema
A4	4th patient or theme	4. Patiens oder Thema
A5	5th patient or theme	5. Patiens oder Thema
AA	further patient or theme	Weiteres Patiens oder Thema
AM	unspecified adjunct	unspezifischer Modifikator
AM-ADV	general-purpose	Universalmodifikator
AM-CAU	cause	Ursache (Grund)
AM-DIR	direction	Richtung
AM-DIS	discourse marker	Diskursmarker
AM-EXT	extent	Ausmaß
AM-LOC	location	Ort
AM-MNR	manner	Art und Weise

Etikett	Bedeutung (englisch)	Bedeutung (deutsch)
AM-MOD	modal verb	Modalverb
AM-NEG	negation marker	Negation (Negationsmarker)
AM-PNC	purpose	Absicht (Zweck oder Ziel)
AM-PRD	predication	Aussage (Überzeugung)
AM-REC	reciprocal	Umkehrung
AM-TMP	temporal	Zeitausdruck
R-A0	reference to A0	Referenz zu A0
R-A1	reference to A1	Referenz zu A1
R-A2	reference to A2	Referenz zu A2
R-A3	reference to A3	Referenz zu A3
R-A4	reference to A4	Referenz zu A4
R-AA	reference to AA	Referenz zu AA
R-AM-ADV	reference to AM-ADV	Referenz zu AM-ADV
R-AM-CAU	reference to AM-CAU	Referenz zu AM-CAU
R-AM-DIR	reference to AM-DIR	Referenz zu AM-DIR
R-AM-EXT	reference to AM-EXT	Referenz zu AM-EXT
R-AM-LOC	reference to AM-LOC	Referenz zu AM-LOC
R-AM-MNR	reference to AM-MNR	Referenz zu AM-MNR
R-AM-PNC	reference to AM-PNC	Referenz zu AM-PNC
R-AM-TMP	reference to AM-TMP	Referenz zu AM-TMP

B Typen und Attribute des initialen ProNat-Graphen

Typ/Attribut	Datentyp	Beschreibung
Knoten-Typen		
token	(String)	Token der Haupthypothese
alternative_token	(String)	Token alternativer Hypothesen
Kanten-Typen		
relation	(String)	Zeiger zum nächsten Token der Sequenz
alternative_token	(String)	Zeiger einer alternativen Hypothese
srl	(String)	Kanten zur Darstellung von semantischen Rollen
Attribute (Knoten-Typ *token*)		
position	int	Position des Tokens innerhalb der Sequenz
type	String	Typ des Tokens (Wort, Interpunktion, etc.)
value	String	Oberflächenform des Tokens
pos	int	Wortart des Tokens
stem	String	Wortstamm des Tokens
lemma	String	Wortlemma des Tokens
chunkName	String	Chunk-Zugehörigkeit des Tokens
successors	int	Anzahl der Tok. des Chunks, die Tok. vorangehen
predecessors	int	Anzahl der Tok. des Chunks, die Tok. nachfolgen
chunkIOB	String	Chunk-Zugehörigkeit des Tokens im IOB-Format
instructionNumber	int	Instruktionsnummer des Token
sentenceNumber	int	Nummer des Satzes zu dem das Token gehört
ner	String	Ermittelter Wert der Eigennamenerkennung
startTime	double	Startzeit des Tokens im Audiosignal
endTime	double	Endzeit des Tokens im Audiosignal

Typ/Attribut	Datentyp	Beschreibung
Attribute (Knoten-Typ *token*)		
asrConfidence	double	Konfidenz des ASR für dieses Token
alternativesCount	int	Anzahl der alternativen Token
verifiedByDialogAgent	boolean	Gibt an, ob Werte durch Dialog-Agenten verifiziert
Attribute (Knoten-Typ *alternative_token*)		
position	int	Position der Alternative innerhalb der Sequenz
type	String	Typ der Alternative (Wort, Interpunktion, etc.)
value	String	Oberflächenform der Alternative
startTime	double	Startzeit der Alternative im Audiosignal
endTime	double	Endzeit der Alternative im Audiosignal
asrConfidence	double	Konfidenz des ASR für diese Alternative
Attribute (Kanten-Typ *relation*)		
value	String	„NEXT“ (Zeiger auf nächstes Token)
Attribute (Kanten-Typ *alternative_token*)		
number	int	Fortlaufende Nummer der Alternative
Attribute (Kanten-Typ *srl*)		
role	String	Abstrakte *A**-Rolle
roleConfidence	double	Konfidenz des Erkenners für diese Rolle
correspondingVerb	String	Zugehöriges Verb (Wurzel der Rollenkombination)
pbRole	String	Zugehörige *PropBank*-Rolle
propBankRolesetID	String	ID des *PropBank*-Rollensatzes
propBankRolesetDescr	String	Beschreibung des *PropBank*-Rollensatzes
vnRole	String	Zugehörige *VerbNet*-Rolle
verbNetFrames	String	Zugehöriger *VerbNet*-Rahmen
fnRole	String	Zugehörige *FrameNet*-Rolle
frameNetFrames	String	Zugehöriger *FrameNet*-Rahmen
eventTypes	String	Typ des *FrameNet*-Events (statisch, dynamisch)

C Studienbögen

In den nachfolgenden Abschnitten sind die Studienbögen aufgeführt, wie sie für die stationäre und die Online-Datensammlung sowie für die Studien zur Evalution des Agenten zur Modellierung und Etikettierung von Diskurs-Themen und zur Evaluation des Gesamtsystems verwendet wurden.

C.1 Stationäre Datensammlung

C.1.1 Einführung (*General Information*)

This survey is part of a research project of the Karlsruhe Institute of Technology. The aim of this project is programming a humanoid robot with natural spoken language. Within the project we try to extract instructions, objects and locations from natural language descriptions to teach a robot new skills. During this study you are kindly requested to fill out a questionnaire and formulate instructions for a robot. Your instructions will be recorded via microphone. The recordings and your personal information will only be used for research purposes and handled anonymously.
The results of this research study may be presented at scientific or professional meetings or published in scientific journals.
All provided information and audiotaped content are kept anonymous. All personal data will be shared and stored using a pseudonym. This pseudonym is assigned randomly and is not related to any personal information.

Before you read this entire form and decide to participate in this project, please understand your **participation is voluntary** and you have the **right to withdraw** your consent or **discontinue participation** at any time **without penalty or loss of benefits** to which you are otherwise entitled. **The alternative is not to participate.** You have the right to refuse to answer particular questions.

If you have any questions, concerns or complaints about this research, its procedures, risks and benefits, contact the supervisor of the study:

Sebastian Weigelt
+49 721 608-44064
weigelt@kit.edu

C.1.2 Einverständniserklärung (*Declaration of Consent*)

I, the undersigned, confirm that (please tick box as appropriate):

1. I have read and understood the information about the study, as provided in the previous pages. ☐
2. I received the opportunity to ask questions about the study and my participation. ☐
3. I voluntarily agree to participate in the study. ☐
4. I understand, that I can withdraw at any time without giving reasons and that I will not be penalized for withdrawing nor will I be questioned on why I have withdrawn. ☐
5. The procedures regarding confidentiality have been clearly explained (e.g. use of names, pseudonyms, anonymization of data, etc.) to me. ☐
6. I give consent to be audiotaped during this study. ☐
7. The use of the data in research, publications, sharing, and archiving has been explained to me. ☐
8. I understand that other researchers will have access to this data only if they agree to preserve the confidentiality of the data and if they agree to the terms I have specified in this form. ☐
9. I, along with the Researcher, agree to sign and date this informed consent form. ☐

Participant:

Name of Participant	Signature	Date

Researcher:

Name of Researcher	Signature	Date

C.1.3 Einführung (*Introduction*)

Thank you for taking part in the KOPRO survey. During the survey you are kindly requested to formulate instructions for a robot. Your instruction will be recorded via a microphone, but only used for research purposes and handled anonymously. This survey is part of a research project (KOPRO) of the Karlsruhe Institute of Technology aiming to program a humanoid robot via spoken natural language. Within the project we try to extract instructions, objects and locations from natural language descriptions to teach a robot new skills.

A first step to accomplish this goal a set of descriptions formulated by native English speakers. This description set allows us to identify varieties in human expressions.

Your task within the survey is to instruct a robot using natural language (and spontaneous expressions). At the following pages different situation will be presented to you. In each of that situations a robot has to fulfil a certain task for you, like getting a cup from a desk and bring it to you. Unfortunately the robot is not familiar with the tasks, but it has a large set of basic skills, like going somewhere, locating something and grabbing something. Based on the basic skills you can instruct the robot and as a consequence teach it a new skill. For example: If you want the robot to serve you coffee, you might say, that it should go to the table (with the coffee machine), fill water into the tank, get a filter from cupboard and so on.

To avoid any influence we do not inform you about the basic skills. You will have to decide for your own, which instructions the robot needs to fulfil the task. While giving your instruction try to act as naturally as possible using your own way of formulating. Therefore instead of interacting with a technical device – what the robot actually is – try to imagine you are talking to real person and formulate your instruction accordingly.

Example description for the complex task of bringing a water bottle:

- *Go to the kitchen table.*
- *Look for the water bottle and take it.*
- *Come over here and hand it over.*

C.1.4 Fragebogen (*Questionnaire*)

1. Age:_____
2. Gender:

   ☐ male
   ☐ female

3. Nationality: _____
4. Have you ever lived in an English speaking country:

   ☐ Yes
      ☐ $\leq$ 6 months
      ☐ 6 months - 1 year
      ☐ $\geq$ 1 year
   ☐ No

5. Occupation: _____
6. Mother tongue (native language):

   ☐ English
   ☐ German
   ☐ other (please quote):_____

7. English experience level:

   ☐ elementary
   ☐ advanced
   ☐ experienced
   ☐ proficient
   ☐ native speaker

8. Programming experience level:

   ☐ none
   ☐ elementary
   ☐ advanced
   ☐ experienced
   ☐ proficient

9. Programming skills:

   ☐ functional programming(e.g. Haskel, Lisp, . . .)
   ☐ procedural/imperative programming (e.g. C, Pascal, . . .)
   ☐ objected-oriented programming (e.g. Java, C++, . . .)

Date and time*: _____

*are used to generate anonymous user id.

C.1.5 Szenario (*Scenario*)

All scenes take place in a kitchen (3D animation in Figure C.1). While expressing your instructions to the robot (Figure C.2a), imagine you are standing in one end of the kitchen looking into the room (Figure C.2b). Somewhere in front of you is the robot awaiting your orders. Your view on the scenery, the position of the robot and relevant objects and locations are displayed in a set of pictures. You do not have to describe any kind of object. Assume that the robot knows what "an orange juice" or a "fridge" is.

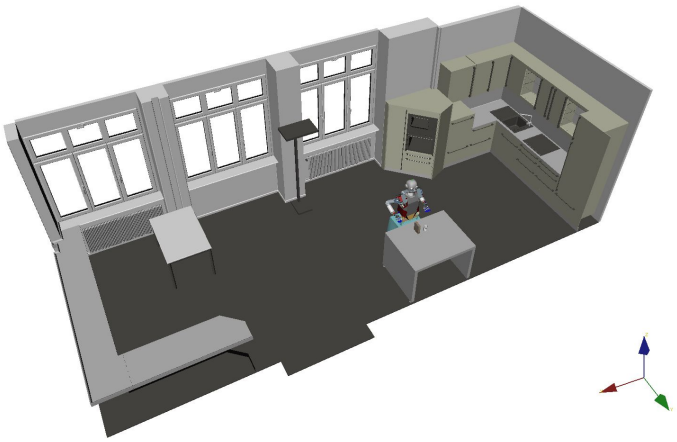
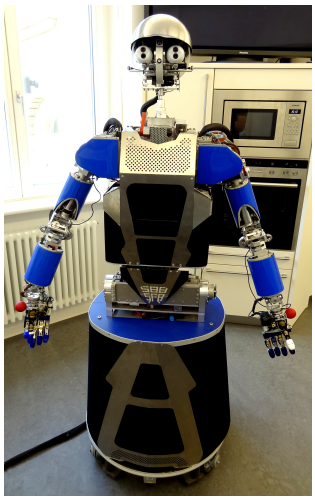


Figure C.1: Kitchen: 3D-Model



(a) ARMAR-III robot

(b) kitchen (subject view)

Figure C.2: Scenery (set-up)

Es folgen die jeweiligen Beschreibungen der konkreten Szenarien, wie in Abbildung 5.4 in Abschnitt 5.5.2.1 gezeigt. Von diesem Schema weit lediglich das achte Szenario ab. Dieses dient zur Generierung von Mitschnitten eines freien Dialogs, wie nachfolgend beschrieben.

C.1.6 Szenariobeschreibung Szenario 8: Freier Dialog

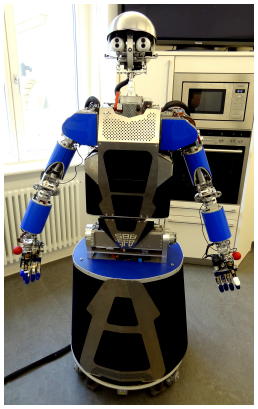
You have bought the new household robot Armar III. At the moment it is just able to fulfill basic tasks. The good news is the robot is able to learn more sophisticated tasks. Therefor you have to explain him a bunch of basic tasks in the correct order. In the case Armar understood everything it has learned a new skill. Otherwise the robot will ask you about the parts of your statement it did not understand. Since Armar is able to understand context, there might come up questions about the context and conditions in the text.

At the moment you are sitting in front of a laptop and have to do very interesting stuff. So you want to teach Armar to take out the laundry of the washing machine and put it into the dryer.

The robot is already aware that you will teach him a new instruction. So it will be fine if your statement just contains the instruction.

Some basic task examples:

- *Go to the cupboard and open it.*
- *Armar cut the orange into halves.*
- *Take the water bottle and bring it to me.*



(a) Armar, the robot

(b) closed washing machine

Advices:

- *Imagine the essential steps you would do putting the laundry into the dryer after the washing machine has finished.*
- *Say or imagine the complete instruction at least once before you start the record.*
- *The robot will ask you a lot of questions. Please be patient and keep calm.*

Thank you for participating.

C.2 Online-Datensammlung

C.2.1 Einführung auf der Plattform *Prolific*

Teach a robot new skills with natural language instructions

The study investigates how people would teach robots new skills with natural language. You have to provide 4 instructions. You will need your ProfilicID, please copy it in advance.
Please participate only once!

C.2.2 Web-Formular

Das für die Online-Datensammlung verwendete Web-Formular kann unter folgender URL abgerufen werden:
`https://docs.google.com/forms/d/e/1FAIpQLSfSaHflNtYN5mYwgeYka7PMxXchEc-wOa-dg1vgjibaPLFVAA/viewform`

Teach a robot new skills

Please enter your ProfilicID

ProlificIDs have 24 alphanumeric characters.

_____

Setting

Target system: Humanoid household robot (called Armar)
Setting: Kitchen
Goal: Teach the robot new skills
Your task: 4 natural language instructionsfor the robot to learn a specific skill

Es folgen zwei Abbildungen, die *ARMAR-III* in der Küchenumgebung zeigen.

Es folgt ein Beispielszenario, wie in Abbildung 5.8 in Abschnitt 5.5.3.1 gezeigt.

Important

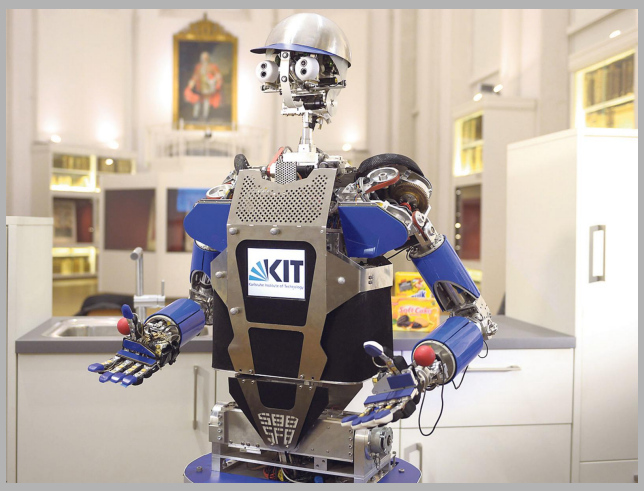
- Imagine the robot being a little child you want to teach something
- Do not forget to
  (a) **explicitly state** that you want to teach something (marked in red above)
  (b) give the skill some name(marked in blue above)
  (c) give a short description of the intermediate steps (marked in green above)
- Use **different wordings** as in the example and for each skill (if possible)

☐ OK

Skill 1

New skill: Greet someone
Intermediate steps:

- Look him or her in the eyes
- Wave robot hand
- Say 'Hello'



How would you the robot this new skill?
"Speak" as naturally as possible. Be creative!
Your instructions:

_____

Es folgen weitere Szenarien in denen eine neue Fähigkeit gelehrt werden soll.

English native speaker?

◯ Yes

◯ No

Skills in programming?

◯ Yes

◯ No

C.3 Evaluation des Agenten zur Modellierung und Etikettierung von Diskurs-Themen

Die für die Evaluation des Agenten zur Modellierung und Etikettierung von Diskurs-Themen verwendete Web-Formulare, können unter folgenden URLs abgerufen werden:
`https://docs.google.com/forms/d/e/1FAIpQLSfOAk4jWm1bIYOYq7vFRrfW7x-qB8_DBX8DfacI_4Bpu2j7tg/viewform`
`https://docs.google.com/forms/d/e/1FAIpQLSf6djsFDOEkY7yUT2hSX62MEFK25Abj9R_a2EDYBCcWkH2d8Q/viewform`

Themenextraktion

Im Folgenden werden elf verschiedene Texte aufgeführt, zu denen (mit verschiedenen Methoden) jeweils verschiedene Themen zugeordnet wurden. Texte sowie Themen sind in englischer Sprache. Die Texte stammen aus verschiedenen Szenarien zur Programmierung mit natürlicher Sprache, in der einem System neue Befehle bzw. Abfolgen mit Hilfe von Erklärungen beigebracht werden sollen.
Ihre Aufgabe ist die Bewertung der Themen zu den jeweiligen Texten. Dafür finden Sie einen Text mit verschiedenen Themenvorschlägen. Das Thema sollen Sie nacheinander daraufhin bewerten, ob das Thema passend, unpassend oder komplett unpassend ist, da es Ihrer Meinung nach zu einem anderen Themengebiet gehört. Dabei ist ein Thema passend, wenn es einen Teil oder den ganzen Text und seinen Inhalt beschreibt. Zum Beispiel wäre im Satz “Put the frying pan into the dishwasher” die Themen “kitchen” oder “kitchen utensil” passend. Das Thema “pan” könnte eher unpassend sein, da es zu spezifisch ist. Ebenso könnte das Thema “machine” eher unpassend sein, da es zu allgemein ist. Zuletzt könnte das Thema “iron mining” unpassend sein, da es (größtenteils) unverwandt mit der eigentlichen Thematik ist. Bei den ersten beiden Optionen kann im Anschluss Ihre Entscheidung mit verschiedenen Antwortmöglichkeiten näher erläutert werden (u.a. Thema war zu spezifisch oder zu allgemein). Beispielsweise kann zu einem Thema nach der Entscheidung ünpassendërklärt werden, dass das Thema zu allgemein oder zu spezifisch war. Bei Unsicherheiten zu den einzelnen Themen und ihren Bedeutungen, schlagen Sie bitte im Wörterbuch nach und informieren sich mit Hilfe der Wikipedia darüber, um Missverständnisse zu vermeiden. So ist unter anderem eine “Lazy Susan” nicht nur eine faule Person mit dem Namen Susan, sondern auch die Bezeichnung für einen Drehtisch (siehe https://en.wikipedia.org/wiki/Lazy_Susan).

Text 1, Thema 1

“Okay Armar, go to the table, grab popcorn, come to me, give me the popcorn, which is in your hand”

Das Thema “Plants used in Native American cuisine” ist

- ◯ Passend
- ◯ Unpassend
- ◯ Unpassend, da komplett unverwandt

Je nach getroffener Auswahl, wird der Proband zu einem anderen Formularfeld weitergeleitet.

Text 1, passendes Thema 1

“Okay Armar, go to the table, grab popcorn, come to me, give me the popcorn, which is in your hand”

Das Thema “Plants used in Native American cuisine” ist

- ◯ etwas zu allgemein
- ◯ in Ordnung
- ◯ etwas zu spezifisch

Text 1, unpassendes Thema 1

“Okay Armar, go to the table, grab popcorn, come to me, give me the popcorn, which is in your hand”

Das Thema “Plants used in Native American cuisine” ist

- ◯ zu allgemein
- ◯ zu spezifisch
- ◯ anderer Grund

Text 1, Thema 2

“Okay Armar, go to the table, grab popcorn, come to me, give me the popcorn, which is in your hand”

Das Thema “Furniture” ist

◯ Passend

◯ Unpassend

◯ Unpassend, da komplett unverwandt

Es folgen dieselben Auswahlmöglichkeiten wie zuvor und dann die weiteren Themen, die für den Text erzeugt wurden. Anschließend folgen die weiteren zu bewertenden Texte und Themen.

Hinweis

Die Studie wurde mithilfe der hier gezeigten Auswahlmöglichkeiten durchgeführt. In der Auswertung wurden jedoch die zweistufigen Bewertungen in einfachere nicht hierarchische Kategorien überführt, wie in Abschnitt 7.3.2 beschrieben. Auf diese Weise können die Ergebnisse besser dargestellt und es können mehr Übereinstimmungen zwischen den Probanden gefunden werden.

C.4 Evaluation des Gesamtsystems: Online-Studie

C.4.1 Einführung auf der Plattform *Prolific*

Compare english commands for a robot with activity diagrams

The study evaluates the similarity between commands given in English and computer generated activity diagrams.

The study begins with an Introduction to activity diagrams. After the introduction, you have to compare English command sequences with activity diagrams. If you identify differences between the commands and the activity diagram you check the appropriate boxes.

There will be 10 activity diagrams. Each diagram may differ in size and complexity.

You will need your ProlificID! Please copy the ID in advance. Please participate only once for the same study. However, you are welcome to participate in the other studies of the series.

C.4.2 Web-Formular

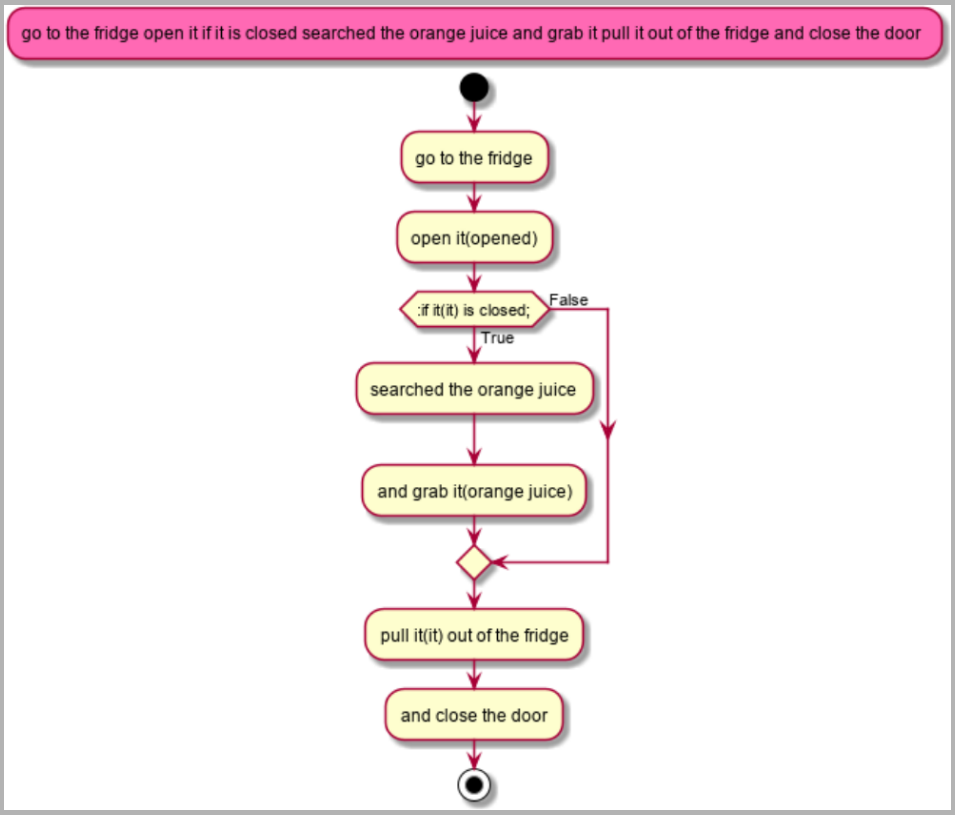
Eines der 24 Web-Formulare, die für die Online-Studie verwendet wurden, kann unter folgender URL abgerufen werden: `https://forms.gle/dQSWiVB9d85tdR1s6`

Comparing Activity Diagrams to Commands

Introduction to activity diagrams

Activity diagrams represent a work flow through activities. They're also known as flow charts. The activity diagrams and sentences in this questionnaire describe instructions to a kitchen robot called Armar. The activity diagrams where constructed automatically. Your task is to identify errors in the diagrams.

Example



Sentence

The full sentence is written atop in pink. This is the sequence of commands which the activity diagram is compared to. Both should ideally represent the same sequence of actions. The activity diagram starts with a full black dot. This is the start of the sequence of activities. The end of the activity diagram is symbolized by a black and white dot at the bottom.

Activities

An activity consists of a single action, command or condition. It is represented with a light yellow node. If a node contains activities that don't make sense, lack information or contain information which should be part of another node, then check the "Activity incorrect/incomprehensible" box. It is also possible that an activity is missing or that another activity should be split up. In that case check the "Activity missing" box. If one of the nodes should be removed, check the "Activity should be removed" box. Activities are connected by arrows. Arrows point to the next activity. If the sequence of activities is incorrect (i.e. an activity should be placed before another) check the "Activity sequence incorrect" box. Some Activities contain words in round brackets after pronouns like it/her/she. These refer to an entity. If they refer to the wrong entity check the "Reference to wrong entity" box.

Es folgt ein Diagramm, dass eine bedingte Verzweigung und eine Zählschleife enthält.

Conditions, Branches and Loops

In some cases a course of activities represents a conditional sequence. In such cases there may be multiple next activities and thus the path is split. Depending on the evaluation of the condition one of the next paths is chosen. Conditions are usually part of a loop or branch. Branches split the sequence in two separate paths. Only one path can be followed. Branches may contain multiple types of errors, such as missing or additional activities. It is also possible that a path doesn't contain activities at all. In these cases check the "Branches incorrect" box. In a loop one of the paths will be repeatedly followed until the condition is met to exit it. A special type of loop is a for-loop. Here the looping path will be followed for a number of times and then exited. If there are errors in any loop, check the "Loop incorrect" box.

Es folgt ein beispielhaftes Diagramm, dass zwei nebenläufige Abschnitte enthält.

Concurrency

Concurrency bars splits the path in a special way. Instead of following a single path, all paths are followed at the same time. Thus the activities in concurrency are done at the same time. The concurrency ends when all paths in the concurrency reach the ending bars. From now on, only a single path is followed.

No errors and other errors

Not all activity diagrams contain errors. It's possible that the diagram is entirely correct. In this case check "everything correct" only. In some cases you may identify errors that do not fit in one of the above categories. In cases, you may pick the "others" check box and describe the error textually.

Please enter your ProfilicID

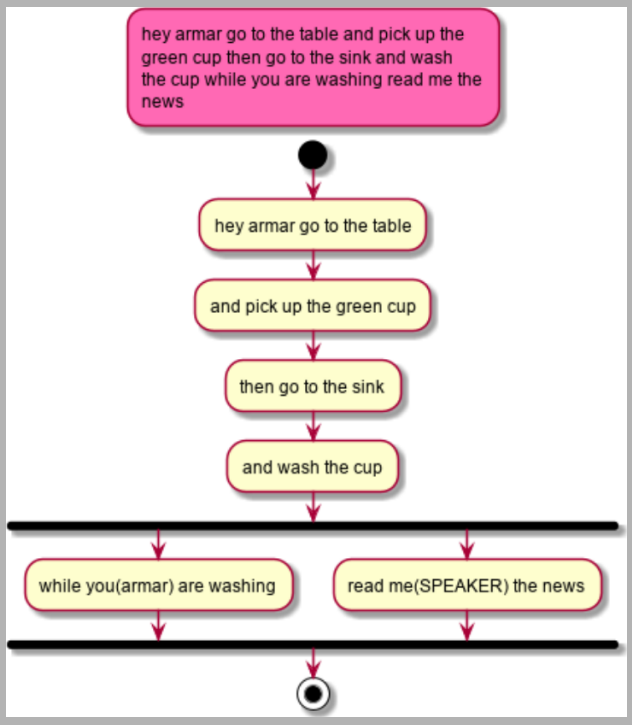
ProlificIDs have 24 alphanumeric characters.

_____

Activity diagrams:

Choose the check boxes, which fit the errors in the activity diagram

s09_088.txt.png



s09_088.txt.png

- ☐ Everything correct
- ☐ Activity incorrect/incomprehensible
- ☐ Activity missing
- ☐ Activity should be removed
- ☐ Activity sequence incorrect
- ☐ Reference to wrong entity
- ☐ Branch incorrect
- ☐ Loop incorrect
- ☐ Concurrency incorrect
- ☐ Sonstiges:_____

Es folgen die weiteren, zu bewertenden Aktivitätsdiagramme.

D Evaluationsdatensätze

Nachfolgend werden Datensätze aufgeführt, die für Evaluationen verschiedener Agenten verwendet wurden und von den in Abschnitt 5.5 beschriebenen Datensammlungen abweichen.

D.1 Evaluation des Agenten zur Modellierung und Etikettierung von Diskurs-Themen

Im Folgenden sind die Transkription zu verschiedenen Szenarien aufgeführt, die für die Evaluation der Modellierung und Etikettierung von Diskurs-Themen anhand einer Studie genutzt wurden (siehe Abschnitt 7.3.2). In die Transkriptionen wurden Satzzeichen injiziert, um den Studienteilnehmern das Textverständnis zu erleichtern. Die Bezeichner der Aufzählungselemente bestehen aus

- einem Suffix, das angibt, ob es sich bei dem Text, um eine Transkription der stationären Datensammlung handelt (ProNat) oder um einen synthetisch erzeugten (Extra),
- einer ersten Ziffer, die das Szenario angibt und
- einer zweiten Ziffer, die angibt, in welcher der beiden Umfragen der Text verwendet wurde.

Datensatz

ProNat_1.1 Okay Armar, go to the table, grab popcorn, come to me, give me the popcorn, which is in your hand

ProNat_1.2 Armar, can you please bring me the popcorn bag

ProNat_2.1 hello Armar, could you go to the table and take the green cup? Please put it in the dishwasher and close it

ProNat_2.2 hey Armar, please place the green cup from the kitchen table into the dishwasher

ProNat_3.1 Armar, would you please go to the fridge and open it, take out the orange juice and bring it to me?

ProNat_3.1 Armar, open the fridge and take the orange juice. Afterwards close the fridge and bring me the orange juice.

ProNat_4.1 hey Armar, could you please have a look at these dishes? If they are dirty put them into the dishwasher. If they are not dirty, put them into the cupboard

ProNat_4.2 robo, go to the table. If there are any dirty dishes, grab the dirty dishes and go to the dishwasher, open the dishwasher, and put the dirty dishes into the dishwasher. Close the dishwasher and return to the table. If there are any clean dishes, grab the clean dishes and go to the cupboard. Open the cupboard and put the clean dishes into the cupboard

ProNat_5.1 hello Armar, I want to make some drinks. Go to the fridge and if there fresh oranges, bring me the fresh oranges together with vodka. Otherwise bring me just orange juice and the vodka

ProNat_5.2 hello Armar, I would want to have some vodka with fresh orange, please. Go to the fridge and check if there are some fresh orange. If there are not any, please make me a vodka with orange juice

ProNat_6.1 Go to the table, take the green cup standing on the table, and go to the fridge. Open the fridge. Right in front of you there is a water bottle, take the bottle, open it, fill water in the cup, and put the bottle back in the fridge. Close the fridge and then bring me the cup. Afterwards go to the dishwasher, open the dishwasher, take two red cups em from inside, bring them to the cupboard, and open the cupboard. Put the cups on the shelf and close the cupboard again

ProNat_6.2 Go to the fridge, open the fridge door, then take the water bottle out. Close the fridge door, then go to the table and open the water bottle. Fill the green cup with the water and then bring the cup to me. Go to the dishwasher, ahm take the two red cups out of the dishwasher, and put them on the cupboard

ProNat_7.1 Hey Armar, I'm hungry. Please, take eh a plate from the dishwasher and rinse it with water at the sink. Afterwards, go to the fridge, open the fridge, and put the instant food you find into in there on to the plate. Close the fridge and warm the plate in the eh microwave. Therefore you have to open the door first. Afterwards, when it's warm, please, bring it eh to me at the table

ProNat_7.2 Go to the dishwasher and take one plate out. Wash this plate and then go to the fridge, put the em meal instant meal on the plate and bring it to the microwave. Put it into the microwave. Then, after it is warmed up, put the ehm meal on the plate and bring the plate to the table

ProNat_8.1 Go to the washing machine and open it. Take out the laundry, put the laundry into the dryer, start the dryer

ProNat_8.2 Go to the dryer. Open the dryer. Go to the front side of the washing machine. Grab the washing machine window handle. Put the handle to open the washing machine. Put your arms into the washing machine. Grab the laundry. Take the laundry out of the washing machine. Go to the dryer. Put the laundry into the dryer. Close the dryer. Push the start button of the dryer

Extra_1.1 Start and accelerate as fast as possible until you fly through the gate. Then slow down and turn left by sixty degree. Accelerate again and dodge the table by ascending first and descending afterwards. Fly through the greenhouse, then turn left and fly above the pond. If you crossed the pond, break and descend down to the lawn and finally turn off

Extra_1.2 Ascend and fly as fast as possible to the gate, turn left and ascend and start flying to the greenhouse. After you dodged the table by ascending, descend again. After passing through the greenhouse, turn left again and fly over the pond. Afterwards slow down and descend until you are on the lawn

Extra_2.1 Follow the line on the carpet. At the end of the carpet turn until you see the rattle. Grab the rattle and afterwards release it again

Extra_2.2 Move along the line until you are at the end of the carpet. Turn right until you see the rattle, grab it and then release it again

Extra_3.1 Alexa, turn up the temperature of the radiator by two degrees, then start playing my favorite playlist

Extra_3.2 Alexa, before you play my favorite playlist, turn on the radiator, because it is getting cold in here

D.2 Evaluation des Agenten zur automatischen Auswahl von Zielsystem- und Umgebungsontologien

Im Folgenden sind die zusätzlichen Texte zu finden, die für die Evaluation des Agenten zur automatischen Auswahl von Zielsystem- und Umgebungsontologien genutzt wurden. Neben den hier aufgeführten Texten, wurden auch die Beschreibungen aus der Evaluation des Agenten zur Modellierung und Etikettierung von Diskurs-Themen verwendet (siehe Abschnitt D.1). Daher wird die dort begonnene Nummerierung hier fortgeführt.

Extra_4.1 hey armar grab the lawn mower and use it to cut the grass

Extra_4.2 hey armar open the shed grab the mower and cut the lawn

Extra_5.1 Go to the Fridge take the tonic water and mix it in the glass with the gin that is on the counter

Extra_5.2 Mix a cuba libre by putting coke rum and some lime juice in a glass

Extra_6.1 go to the closet open it and grab the sweater and the trousers and bring them to me

Extra_6.2 move to the desk and clean it afterwards go the the nightstand and clean it as well if you finished these tasks take the book out of the shelf and bring it to me

Extra_7.1 alexa please play my metal playlist in a random order

Extra_7.2 alexa what are the most famous songwriters who use a piano

Extra_8.1 hey armar, Louis Armstrong is a really good artist, he plays the trumpet so well. turn up the volume. Then walk to the fridge and bring us two beers, afterwards get us each one whisky so we can honor him

Extra_8.2 armar, i think it is perfect to combine jazz with a nice glass of whisky. So please bring us our drinks

Extra_9.1 go to the fridge and grab the ketchup and the water and bring it outside to our patio table at the pond. be careful not to walk over the freshly sewn lawn

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[AA18] Atzeni, M. und Atzori, M. „Translating Natural Language to Code: An Unsupervised Ontology-Based Approach“. In: *2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*. Sep. 2018, S. 1–8. DOI: 10.1109/AIKE.2018.00009 (zitiert auf den Seiten 74, 96, und 131).

[Abb83] Abbott, R. J. „Program Design by Informal English Descriptions“. In: *Commun. ACM* 26.11 (Nov. 1983), S. 882–894. ISSN: 0001-0782. DOI: 10.1145/182.358441 (zitiert auf Seite 70).

[Abn92] Abney, S. P. „Parsing By Chunks“. In: *Principle-Based Parsing: Computation and Psycholinguistics*. Hrsg. von R. C. Berwick, S. P. Abney und C. Tenny. Studies in Linguistics and Philosophy. Dordrecht: Springer Netherlands, 1992, S. 257–278. ISBN: 978-94-011-3474-3. DOI: 10.1007/978-94-011-3474-3_10 (zitiert auf Seite 159).

[Aho+06] Aho, A. V., Lam, M. S., Sethi, R. und Ullman, J. D. *Compilers: Principles Techniques and Tools. 2007*. Second. Addison Wesley, 2006. ISBN: 0-321-48681-1 (zitiert auf den Seiten 291 und 296).

[All95] Allen, J. *Natural Language Understanding*. Pearson, 1995. ISBN: 978-0-8053-0334-6 (zitiert auf Seite 46).

[ALZ15] Artzi, Y., Lee, K. und Zettlemoyer, L. „Broad-Coverage CCG Semantic Parsing with AMR“. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, S. 1699–1710. DOI: 10.18653/v1/D15-1198 (zitiert auf Seite 75).

[Asf+06] Asfour, T., Regenstein, K., Azad, P., Schroder, J., Bierbaum, A., Vahrenkamp, N. und Dillmann, R. „ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control“. In: *2006 6th IEEE-RAS International Conference on Humanoid Robots*. Dez. 2006, S. 169–175. DOI: 10.1109/ICHR.2006.321380 (zitiert auf den Seiten 134 und 274).

[Aue+07] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R. und Ives, Z. „DBpedia: A Nucleus for a Web of Open Data“. In: *The Semantic Web*. Hrsg. von K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber und P. Cudré-Mauroux. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, S. 722–735. ISBN: 978-3-540-76298-0. DOI: 10.1007/978-3-540-76298-0_52 (zitiert auf Seite 56).

[Av04] Antoniou, G. und van Harmelen, F. „Web Ontology Language: OWL“. In: *Handbook on Ontologies*. Hrsg. von S. Staab und R. Studer. International Handbooks on Information Systems. Berlin, Heidelberg: Springer, 2004, S. 67–92. ISBN: 978-3-540-24750-0. DOI: 10.1007/978-3-540-24750-0_4 (zitiert auf Seite 53).

[Bas+14] Bastianelli, E., Castellucci, G., Croce, D., Iocchi, L., Basili, R. und Nardi, D. „HuRIC: A Human Robot Interaction Corpus“. In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. Reykjavik, Iceland: European Language Resources Association (ELRA), Mai 2014, S. 4519–4526 (zitiert auf Seite 139).

[BB79] Ballard, B. W. und Biermann, A. W. „Programming in Natural Language: NLC as a Prototype“. In: *Proceedings of the 1979 Annual Conference*. ACM '79. New York, NY, USA: ACM, 1979, S. 228–237. ISBN: 0-89791-008-7. DOI: 10.1145/800177.810072 (zitiert auf Seite 68).

[BB80] Biermann, A. W. und Ballard, B. W. „Toward Natural Language Computation“. In: *Computational Linguistics* 6.2 (Apr. 1980), S. 71–86. ISSN: 0891-2017 (zitiert auf Seite 68).

[BBR01] Bangalore, B., Bordel, G. und Riccardi, G. „Computing Consensus Translation from Multiple Machine Translation Systems“. In: *IEEE Workshop on Automatic Speech Recognition and Understanding, 2001. ASRU '01*. Dez. 2001, S. 351–354. DOI: 10.1109/ASRU.2001.1034659 (zitiert auf Seite 155).

[BBS83] Biermann, A. W., Ballard, B. W. und Sigmon, A. H. „An Experimental Study of Natural Language Programming“. In: *International Journal of Man-Machine Studies* 18.1 (1983), S. 71–87. ISSN: 0020-7373. DOI: 10.1016/S0020-7373(83)80005-4 (zitiert auf Seite 68).

[BCR98] Biber, D., Conrad, S. und Reppen, R. *Corpus Linguistics: Investigating Language Structure and Use*. Cambridge University Press, Apr. 1998. ISBN: 978-0-521-49957-6 (zitiert auf Seite 25).

[BDS92] Bear, J., Dowding, J. und Shriberg, E. „Integrating Multiple Knowledge Sources for Detection and Correction of Repairs in Human-Computer Dialog“. In: *Proceedings of the 30th Annual Meeting on Association for Computational Linguistics*. ACL '92. Stroudsburg, PA, USA: Association for Computational Linguistics, 1992, S. 56–63. DOI: 10.3115/981967.981975 (zitiert auf Seite 171).

[Beg04] Begel, A. „Spoken Language Support for Software Development“. In: *2004 IEEE Symposium on Visual Languages and Human Centric Computing*. Sep. 2004, S. 271–272. DOI: 10.1109/VLHCC.2004.49 (zitiert auf den Seiten 69, 95, und 216).

[BFL98] Baker, C. F., Fillmore, C. J. und Lowe, J. B. „The Berkeley FrameNet Project“. In: *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*. ACL '98/COLING '98. USA: Association for Computational Linguistics, Aug. 1998, S. 86–90. DOI: 10.3115/980845.980860 (zitiert auf Seite 54).

[BG05] Begel, A. und Graham, S. „Spoken Programs“. In: *2005 IEEE Symposium on Visual Languages and Human-Centric Computing*. Sep. 2005, S. 99–106. DOI: 10.1109/VLHCC.2005.58 (zitiert auf den Seiten 69, 95, und 216).

[BHS07] Buschmann, F., Henney, K. und Schmidt, D. *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2007. ISBN: 978-0-470-05902-9 (zitiert auf Seite 105).

[Bis06] Bishop, C. M. *Pattern Recognition and Machine Learning*. Information Science and Statistics. New York, NY: Springer, 2006 (zitiert auf den Seiten 16 und 19).

[Ble12] Blei, D. M. „Probabilistic Topic Models“. In: *Commun. ACM* 55.4 (Apr. 2012), S. 77–84. ISSN: 0001-0782. DOI: 10.1145/2133806.2133826 (zitiert auf Seite 185).

[BNJ03] Blei, D. M., Ng, A. Y. und Jordan, M. I. „Latent Dirichlet Allocation“. In: *Journal of Machine Learning Research* 3.4/5 (März 2003), S. 993–1022. ISSN: 1532-4435 (zitiert auf Seite 185).

[Bob+77] Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H. und Winograd, T. „GUS, a Frame-Driven Dialog System“. In: *Artificial Intelligence* 8.2 (Apr. 1977), S. 155–173. ISSN: 0004-3702. DOI: 10.1016/0004-3702(77)90018-2 (zitiert auf Seite 38).

[Boj+17] Bojanowski, P., Grave, E., Joulin, A. und Mikolov, T. „Enriching Word Vectors with Subword Information“. In: *Transactions of the Association for Computational Linguistics* 5 (Dez. 2017), S. 135–146. DOI: 10.1162/tacl_a_00051 (zitiert auf den Seiten 59 und 240).

[Bor+10] Bordes, A., Usunier, N., Collobert, R. und Weston, J. „Towards Understanding Situated Natural Language“. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, März 2010, S. 65–72 (zitiert auf Seite 194).

[BP98] Brin, S. und Page, L. „The Anatomy of a Large-Scale Hypertextual Web Search Engine“. In: *Computer Networks and ISDN Systems*. Proceedings of the Seventh International World Wide Web Conference 30.1 (Apr. 1998), S. 107–117. ISSN: 0169-7552. DOI: 10.1016/S0169-7552(98)00110-X (zitiert auf den Seiten 186 und 187).

[BQ16] Beltagy, I. und Quirk, C. „Improved Semantic Parsers For If-Then Statements“. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, S. 726–736. DOI: 10.18653/v1/P16-1069 (zitiert auf den Seiten 96, 102, und 216).

[BR03] Bohus, D. und Rudnicky, A. I. „RavenClaw : Dialog Management Using Hierarchical Task Decomposition and an Expectation Agenda“. In: *EUROSPEECH-2003* (2003), S. 597–600 (zitiert auf Seite 44).

[Bri92] Brill, E. „A Simple Rule-Based Part of Speech Tagger“. In: *Proceedings of the Workshop on Speech and Natural Language*. HLT ’91. Harriman, New York: Association for Computational Linguistics, 1992, S. 112–116. ISBN: 1-55860-272-0. DOI: 10.3115/1075527.1075553 (zitiert auf Seite 30).

[Bro+20] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. und Amodei, D. „Language Models Are Few-Shot Learners“. In: *arXiv:2005.14165 [cs]* (Juli 2020). arXiv: 2005.14165 [cs] (zitiert auf Seite 60).

[Cam+17] Campagna, G., Ramesh, R., Xu, S., Fischer, M. und Lam, M. S. „Almond: The Architecture of an Open, Crowdsourced, Privacy-Preserving, Programmable Virtual Assistant“. In: *Proceedings of the 26th International Conference on World Wide Web*. WWW ’17. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2017, S. 341–350. ISBN: 978-1-4503-4913-0. DOI: 10.1145/3038912.3052562 (zitiert auf den Seiten 91 und 97).

[Cam+19] Campagna, G., Xu, S., Moradshahi, M., Socher, R. und Lam, M. S. „Genie: A Generator of Natural Language Semantic Parsers for Virtual Assistant Commands“. In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2019. New York, NY, USA: Association for Computing Machinery, Juni 2019, S. 394–410. ISBN: 978-1-4503-6712-7. DOI: 10.1145/3314221.3314594 (zitiert auf den Seiten 92 und 97).

[Car+15] Carvalho, G., Barros, F., Carvalho, A., Cavalcanti, A., Mota, A. und Sampaio, A. „NAT2TEST Tool: From Natural Language Requirements to Test Cases Based on CSP“. In: *Software Engineering and Formal Methods*. Hrsg. von R. Calinescu und B. Rumpe. Lecture Notes in Computer Science 9276. Springer International Publishing, 2015, S. 283–290. ISBN: 978-3-319-22968-3 978-3-319-22969-0 (zitiert auf den Seiten 84 und 97).

[CAS09] Cruz, I. F., Antonelli, F. P. und Stroe, C. „AgreementMaker: Efficient Matching for Large Real-World Schemas and Ontologies“. In: *Proceedings of the VLDB Endowment* 2.2 (Aug. 2009), S. 1586–1589. ISSN: 2150-8097. DOI: 10.14778/1687553.1687598 (zitiert auf Seite 270).

[Cer09] Cerisara, C. „Automatic Discovery of Topics and Acoustic Morphemes from Speech“. In: *Computer Speech & Language* 23.2 (Apr. 2009), S. 220–239. ISSN: 0885-2308. DOI: 10.1016/j.csl.2008.06.004 (zitiert auf Seite 185).

[CG05] Cheyer, A. und Guzzoni, D. „United States Patent: 8677377 - Method and apparatus for building an intelligent automated assistant“. 8677377. Sep. 2005 (zitiert auf Seite 91).

[Cha+01] Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J. und Menon, R. *Parallel Programming in OpenMP*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001. ISBN: 978-1-55860-671-5 (zitiert auf Seite 297).

[Che+17] Chen, X., Shi, Z., Qiu, X. und Huang, X. „Adversarial Multi-Criteria Learning for Chinese Word Segmentation“. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, Juli 2017, S. 1193–1203. DOI: 10.18653/v1/P17-1110 (zitiert auf Seite 27).

[Cho+14] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. und Bengio, Y. „Learning Phrase Representations Using RNN Encoder–Decoder for Statistical Machine Translation“. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Okt. 2014, S. 1724–1734. DOI: 10.3115/v1/D14-1179 (zitiert auf Seite 22).

[CJ01] Charniak, E. und Johnson, M. „Edit Detection and Parsing for Transcribed Speech“. In: *Second Meeting of the North American Chapter of the Association for Computational Linguistics*. 2001 (zitiert auf Seite 171).

[Cla73] Clark, H. H. „Space, Time, Semantics, and the Child“. In: *Cognitive Development and Acquisition of Language*. Hrsg. von T. E. Moore. San Diego: Academic Press, Jan. 1973, S. 27–63. ISBN: 978-0-12-505850-6. DOI: 10.1016/B978-0-12-505850-6.50008-6 (zitiert auf Seite 198).

[CM04] Carreras, X. und Màrquez, L. „Introduction to the CoNLL-2004 Shared Task: Semantic Role Labeling“. In: *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL-2004) at HLT-NAACL 2004*. Boston, Massachusetts, USA: Association for Computational Linguistics, Mai 2004, S. 89–97 (zitiert auf den Seiten 37, 38, 164, und 352).

[CM05] Carreras, X. und Màrquez, L. „Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling“. In: *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*. Ann Arbor, Michigan: Association for Computational Linguistics, Juni 2005, S. 152–164 (zitiert auf den Seiten 37, 38, 164, und 352).

[CM09] Coursey, K. und Mihalcea, R. „Topic Identification Using Wikipedia Graph Centrality“. In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*. NAACL-Short ’09. Stroudsburg, PA, USA: Association for Computational Linguistics, 2009, S. 117–120 (zitiert auf Seite 186).

[CMM09] Coursey, K., Mihalcea, R. und Moen, W. „Using Encyclopedic Knowledge for Automatic Topic Identification“. In: *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*. CoNLL ’09. Stroudsburg, PA, USA:

Association for Computational Linguistics, 2009, S. 210–218. ISBN: 978-1-932432-29-9 (zitiert auf Seite 186).

[Coh60] Cohen, J. „A Coefficient of Agreement for Nominal Scales“. In: *Educational and Psychological Measurement* 20 (1960), S. 37–46. ISSN: 1552-3888(Electronic), 0013-1644(Print). DOI: 10.1177/001316446002000104 (zitiert auf Seite 65).

[Col+11] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K. und Kuksa, P. „Natural Language Processing (Almost) from Scratch“. In: *J. Mach. Learn. Res.* 12 (Nov. 2011), S. 2493–2537. ISSN: 1532-4435 (zitiert auf den Seiten 159, 163, 164, und 201).

[CR98] Chinchor, N. und Robinson, P. „Appendix E: MUC-7 Named Entity Task Definition (Version 3.5)“. In: *Proceedings of the Seventh Message Understanding Conference (MUC-7)*. Fairfax, Virginia, US, 1998 (zitiert auf Seite 32).

[CRA16] Corcoglioniti, F., Rospocher, M. und Aprosio, A. P. „Frame-Based Ontology Population with PIKES“. In: *IEEE Transactions on Knowledge and Data Engineering* 28.12 (Dez. 2016), S. 3261–3275. ISSN: 1558-2191. DOI: 10.1109/TKDE.2016.2602206 (zitiert auf Seite 164).

[CRF03] Cohen, W. W., Ravikumar, P. und Fienberg, S. E. „A Comparison of String Distance Metrics for Name-Matching Tasks“. In: *Proceedings of the 2003 International Conference on Information Integration on the Web*. IIWEB'03. Acapulco, Mexico: AAAI Press, Aug. 2003, S. 73–78 (zitiert auf Seite 49).

[CRM07] Cohen, S., Rokach, L. und Maimon, O. „Decision-Tree Instance-Space Decomposition with Grouped Gain-Ratio“. In: *Inf. Sci.* 177.17 (Sep. 2007), S. 3592–3612. ISSN: 0020-0255. DOI: 10.1016/j.ins.2007.01.016 (zitiert auf Seite 238).

[Cry11] Crystal, D. *A Dictionary of Linguistics and Phonetics*. John Wiley & Sons, Sep. 2011. ISBN: 978-1-4443-5675-5 (zitiert auf Seite 35).

[CSH18] Chen, B., Sun, L. und Han, X. „Sequence-to-Action: End-to-End Semantic Graph Generation for Semantic Parsing“. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, Juli 2018, S. 766–777 (zitiert auf den Seiten 78 und 96).

[Cut+92] Cutting, D., Kupiec, J., Pedersen, J. und Sibun, P. „A Practical Part-of-Speech Tagger“. In: *Proceedings of the Third Conference on Applied Natural Language Processing*. ANLC '92. USA: Association for Computational Linguistics, März 1992, S. 133–140. DOI: 10.3115/974499.974523 (zitiert auf Seite 29).

[Dah+94] Dahl, D. A., Bates, M., Brown, M., Fisher, W., Hunicke-Smith, K., Pallett, D., Pao, C., Rudnicky, A. und Shriberg, E. „Expanding the Scope of the ATIS Task: The ATIS-3 Corpus“. In: *Proceedings of the Workshop on Human Language Technology*. HLT '94. USA: Association for Computational Linguistics, März 1994, S. 43–48. ISBN: 978-1-55860-357-8. DOI: 10.3115/1075812.1075823 (zitiert auf den Seiten 73 und 138).

[DBB52] Davis, K. H., Biddulph, R. und Balashek, S. „Automatic Recognition of Spoken Digits“. In: *The Journal of the Acoustical Society of America* 24.6 (Nov. 1952), S. 637–642. ISSN: 0001-4966. DOI: 10.1121/1.1906946 (zitiert auf Seite 40).

[Des+16] Desai, A., Gulwani, S., Hingorani, V., Jain, N., Karkare, A., Marron, M., R, S. und Roy, S. „Program Synthesis Using Natural Language“. In: *Proceedings of the 38th International Conference on Software Engineering*. ICSE ’16. New York, NY, USA: ACM, 2016, S. 345–356. ISBN: 978-1-4503-3900-1. DOI: 10.1145/2884781.2884786 (zitiert auf den Seiten 73 und 96).

[Dev+19] Devlin, J., Chang, M.-W., Lee, K. und Toutanova, K. „BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding“. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Juni 2019, S. 4171–4186. DOI: 10.18653/v1/N19-1423 (zitiert auf den Seiten 60 und 249).

[DFN06] Désilets, A., Fox, D. C. und Norton, S. „VoiceCode: An Innovative Speech Interface for Programming-by-Voice“. In: *CHI ’06 Extended Abstracts on Human Factors in Computing Systems*. CHI EA ’06. New York, NY, USA: ACM, 2006, S. 239–242. ISBN: 978-1-59593-298-3. DOI: 10.1145/1125451.1125502 (zitiert auf den Seiten 69, 95, und 216).

[Dij63] Dijkstra, E. W. „On the Design of Machine Independent Programming Languages“. In: *Annual Review in Automatic Programming*. Annual Review in Automatic Programming 3 (1963), S. 27–42. ISSN: 0066-4138. DOI: 10.1016/S0066-4138(63)80003-8 (zitiert auf Seite 67).

[Dij64] Dijkstra, E. W. „Some Comments on the Aims of MIRFAC“. In: *Commun. ACM* 7.3 (März 1964), S. 190–. ISSN: 0001-0782. DOI: 10.1145/363958.364002 (zitiert auf Seite 67).

[Dij79] Dijkstra, E. W. „On the Foolishness of "Natural Language Programming"“. In: *Program Construction*. Hrsg. von P. D. D. h. c. F. L. Bauer, D.-M. M. Broy, E. W. Dijkstra, S. L. Gerhart, D. Gries, M. Griffiths, J. V. Guttag, J. J. Horning, S. S. Owicki, C. Pair, H. Partsch, P. Pepper, M. Wirsing und H. Wössner. Lecture Notes in Computer Science 69. Springer Berlin Heidelberg, 1979, S. 51–53. ISBN: 978-3-540-09251-3 978-3-540-35312-6. DOI: 10.1007/BFb0014656 (zitiert auf Seite 67).

[DL18] Dong, L. und Lapata, M. „Coarse-to-Fine Decoding for Neural Semantic Parsing“. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, Juli 2018, S. 731–742. DOI: 10.18653/v1/P18-1068 (zitiert auf den Seiten 79 und 96).

[dM08] de Marneffe, M.-C. und Manning, C. D. „The Stanford Typed Dependencies Representation“. In: *Coling 2008: Proceedings of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation - CrossParser ’08*. Manchester, United Kingdom:

Association for Computational Linguistics, 2008, S. 1–8. ISBN: 978-1-905593-50-7. DOI: 10.3115/1608858.1608859 (zitiert auf Seite 34).

[DMS00] Dale, R., Moisl, H. und Somers, H. *Handbook of Natural Language Processing*. CRC Press, Juli 2000. ISBN: 978-0-8247-9000-4 (zitiert auf den Seiten 27 und 28).

[DR01] Declerck, R. und Reed, S. *Conditionals: A Comprehensive Empirical Analysis*. Bd. 37. Berlin: De Gruyter Mouton, Jan. 2001, S. 1–538. ISBN: 978-3-11-017144-0 (zitiert auf den Seiten 217 und 224).

[Dur92] Duranti, A. *Rethinking Context: Language as an Interactive Phenomenon*. Cambridge University Press, Mai 1992. ISBN: 978-0-521-42288-8 (zitiert auf Seite 194).

[DVD10] De Smedt, T., Van Asch, V. und Daelemans, W. *Memory-Based Shallow Parser for Python*. Techn. Ber. 2. 2010 (zitiert auf Seite 159).

[EC01] Edmonds, P. und Cotton, S. „SENSEVAL-2: Overview“. In: *The Proceedings of the Second International Workshop on Evaluating Word Sense Disambiguation Systems*. SENSEVAL '01. USA: Association for Computational Linguistics, Juli 2001, S. 1–5 (zitiert auf Seite 42).

[FDK15] Ferguson, J., Durrett, G. und Klein, D. „Disfluency Detection with a Semi-Markov Model and Prosodic Features“. In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Denver, Colorado: Association for Computational Linguistics, Mai 2015, S. 257–262. DOI: 10.3115/v1/N15-1029 (zitiert auf Seite 171).

[Fei+19] Fei, H., Li, X., Li, D. und Li, P. „End-to-End Deep Reinforcement Learning Based Coreference Resolution“. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Juli 2019, S. 660–665. DOI: 10.18653/v1/P19-1064 (zitiert auf Seite 204).

[Fel10] Fellbaum, C. „WordNet“. In: *Theory and Applications of Ontology: Computer Applications*. Hrsg. von R. Poli, M. Healy und A. Kameas. Dordrecht: Springer Netherlands, 2010, S. 231–243. ISBN: 978-90-481-8847-5. DOI: 10.1007/978-90-481-8847-5_10 (zitiert auf Seite 53).

[Fel98] Fellbaum, C. *WordNet: An Electronic Lexical Database*. MIT Press, 1998. ISBN: 978-0-262-06197-1 (zitiert auf Seite 53).

[Fer+10] Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A. A., Lally, A., Murdock, J. W., Nyberg, E., Prager, J., Schlaefer, N. und Welty, C. „Building Watson: An Overview of the DeepQA Project“. In: *AI Magazine* 31.3 (2010), S. 59–79. ISSN: 0738-4602. DOI: 10.1609/aimag.v31i3.2303 (zitiert auf Seite 13).

[Fet04] Fetzer, A. *Recontextualizing Context: Grammaticality Meets Appropriateness*. John Benjamins Publishing, 2004. ISBN: 978-90-272-5363-7 (zitiert auf Seite 194).

[Fil66] Fillmore, C. J. „A Proposal Concerning English Prepositions“. In: *Monograph Series on Languages and Linguistics* 19 (1966), S. 19–34 (zitiert auf Seite 37).

[Fil76] Fillmore, C. J. „Frame Semantics and the Nature of Language*“. In: *Annals of the New York Academy of Sciences* 280.1 (1976), S. 20–32. ISSN: 1749-6632. DOI: 10.1111/j.1749-6632.1976.tb25467.x (zitiert auf Seite 54).

[Fir57] Firth, J. R. „A Synopsis of Linguistic Theory, 1930-1955“. In: *Studies in Linguistic Analysis* (1957) (zitiert auf Seite 41).

[Fis97] Fiscus, J. „A Post-Processing System to Yield Reduced Word Error Rates: Recognizer Output Voting Error Reduction (ROVER)“. In: *1997 IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*. Dez. 1997, S. 347–354. DOI: 10.1109/ASRU.1997.659110 (zitiert auf Seite 154).

[Fle71] Fleiss, J. L. „Measuring Nominal Scale Agreement among Many Raters“. In: *Psychological Bulletin* 76.5 (1971), S. 378–382. ISSN: 1939-1455(Electronic),0033-2909(Print). DOI: 10.1037/h0031619 (zitiert auf Seite 65).

[FR05] Fleischman, M. und Roy, D. „Intentional Context in Situated Natural Language Learning“. In: *Proceedings of the Ninth Conference on Computational Natural Language Learning*. CONLL '05. Stroudsburg, PA, USA: Association for Computational Linguistics, 2005, S. 104–111 (zitiert auf Seite 194).

[Fuc20] Fuchß, D. *Assessing Hypotheses in Multi-Agent Systems for Natural Language Processing*. https://publikationen.bibliothek.kit.edu/1000126806. 2020. DOI: 10.5445/IR/1000126806 (zitiert auf Seite 343).

[Gam+94] Gamma, E., Helm, R., Johnson, R. und Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, Okt. 1994. ISBN: 978-0-321-70069-8 (zitiert auf den Seiten 118, 279, 293, und 296).

[Gaš+14] Gašić, M., Kim, D., Tsiakoulis, P., Breslin, C., Henderson, M., Szummer, M., Thomson, B. und Young, S. „Incremental On-Line Adaptation of POMDP-Based Dialogue Managers to Extended Domains“. In: *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*. Jan. 2014, S. 140–144 (zitiert auf Seite 44).

[GB00] Graff, D. und Bird, S. „Many Uses, Many Annotations for Large Speech Corpora: Switchboard and TDT as Case Studies“. In: *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC'00)*. Athens, Greece: European Language Resources Association (ELRA), Mai 2000 (zitiert auf Seite 171).

[GBC06] Guzzoni, D., Baur, C. und Cheyer, A. „Active: A Unified Platform for Building Intelligent Web Interaction Assistants“. In: *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology - Workshops, Hong Kong, China, 18-22 December 2006*. IEEE Computer Society, Dez. 2006, S. 417–420. ISBN: 0-7695-2749-3. DOI: 10.1109/WI-IATW.2006.27 (zitiert auf den Seiten 91 und 194).

[GBC07] Guzzoni, D., Baur, C. und Cheyer, A. „Modeling Human-Agent Interaction with Active Ontologies.“ In: *AAAI Spring Symposium: Interaction Challenges for Intelligent Assistants*. 2007, S. 52–59 (zitiert auf den Seiten 91 und 194).

[GBC16] Goodfellow, I., Bengio, Y. und Courville, A. *Deep Learning*. The MIT Press, 2016. ISBN: 978-0-262-03561-3 (zitiert auf den Seiten 20, 57, und 59).

[GCB06] Guzzoni, D., Cheyer, A. und Baur, C. „Active, a Platform for Building Intelligent Software“. In: *Proceedings of the Second IASTED International Conference on Computational Intelligence, San Francisco, California, USA, November 20-22, 2006*. Hrsg. von B. Kovalerchuk. IASTED/ACTA Press, 2006, S. 126–131. ISBN: 0-88986-603-1 (zitiert auf den Seiten 91 und 194).

[GHM92] Godfrey, J. J., Holliman, E. C. und McDaniel, J. „SWITCHBOARD: Telephone Speech Corpus for Research and Development“. In: *Proceedings of the 1992 IEEE International Conference on Acoustics, Speech and Signal Processing - Volume 1*. ICASSP'92. San Francisco, California: IEEE Computer Society, März 1992, S. 517–520. ISBN: 978-0-7803-0532-8 (zitiert auf Seite 171).

[GJ02] Gildea, D. und Jurafsky, D. „Automatic Labeling of Semantic Roles“. In: *Computational Linguistics* 28.3 (Sep. 2002), S. 245–288. ISSN: 0891-2017. DOI: 10.1162/089120102760275983 (zitiert auf Seite 38).

[GK15] Gvero, T. und Kuncak, V. „Synthesizing Java Expressions from Free-Form Queries“. In: *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. OOPSLA 2015. New York, NY, USA: ACM, 2015, S. 416–432. ISBN: 978-1-4503-3689-5. DOI: 10.1145/2814270.2814295 (zitiert auf den Seiten 71 und 96).

[GM14] Gulwani, S. und Marron, M. „NLyze: Interactive Programming by Natural Language for Spreadsheet Data Analysis and Manipulation“. In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. SIGMOD '14. New York, NY, USA: ACM, 2014, S. 803–814. ISBN: 978-1-4503-2376-5. DOI: 10.1145/2588555.2612177 (zitiert auf den Seiten 71 und 96).

[GR71] Greene, B. B. und Rubin, G. M. *Automatic Grammatical Tagging of English*. Techn. Ber. Providence, Rhode Island: Department of Linguistics, Brown University, 1971, S. 306 (zitiert auf Seite 29).

[Gre+17] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R. und Schmidhuber, J. „LSTM: A Search Space Odyssey“. In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (Okt. 2017), S. 2222–2232. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2016.2582924 (zitiert auf Seite 22).

[Gru65] Gruber, J. S. „Studies in Lexical Relations.“ Thesis. Massachusetts Institute of Technology, 1965 (zitiert auf Seite 37).

[Gru93] Gruber, T. R. „A Translation Approach to Portable Ontology Specifications“. In: *Knowledge Acquisition* 5.2 (1993), S. 199–220. ISSN: 1042-8143. DOI: 10.1006/knac.1993.1008 (zitiert auf Seite 52).

[GS96] Grishman, R. und Sundheim, B. „Message Understanding Conference-6: A Brief History“. In: *Proceedings of the 16th Conference on Computational Linguistics - Volume 1*. COLING ’96. USA: Association for Computational Linguistics, Aug. 1996, S. 466–471. DOI: 10.3115/992628.992709 (zitiert auf Seite 31).

[GSC00] Gers, F. A., Schmidhuber, J. und Cummins, F. „Learning to Forget: Continual Prediction with LSTM“. In: *Neural Computation* 12.10 (Okt. 2000), S. 2451–2471. ISSN: 0899-7667. DOI: 10.1162/089976600300015015 (zitiert auf Seite 22).

[Gu+16] Gu, X., Zhang, H., Zhang, D. und Kim, S. „Deep API Learning“. In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2016. New York, NY, USA: ACM, 2016, S. 631–642. ISBN: 978-1-4503-4218-6. DOI: 10.1145/2950290.2950334 (zitiert auf den Seiten 71 und 96).

[Guu+17] Guu, K., Pasupat, P., Liu, E. und Liang, P. „From Language to Programs: Bridging Reinforcement Learning and Maximum Marginal Likelihood“. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, Juli 2017, S. 1051–1062. DOI: 10.18653/v1/P17-1097 (zitiert auf den Seiten 76 und 96).

[GWG10] Georgila, K., Wang, N. und Gratch, J. „Cross-Domain Speech Disfluency Detection“. In: *Proceedings of the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. SIGDIAL ’10. Tokyo, Japan: Association for Computational Linguistics, Sep. 2010, S. 237–240. ISBN: 978-1-932432-85-5 (zitiert auf Seite 171).

[Hae03] Haegeman, L. „Conditional Clauses: External and Internal Syntax“. In: *Mind & Language* 18.4 (Sep. 2003), S. 317–339. ISSN: 1468-0017. DOI: 10.1111/1468-0017.00230 (zitiert auf Seite 218).

[Hal+09] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. und Witten, I. H. „The WEKA Data Mining Software: An Update“. In: *ACM SIGKDD Explorations Newsletter* 11.1 (Nov. 2009), S. 10–18. ISSN: 1931-0145. DOI: 10.1145/1656274.1656278 (zitiert auf Seite 181).

[Ham18] Hamann, D. „Vollständigkeits- Und Semantiküberprüfung Für Gesprochene Aussagen“. Bachelor’s Thesis. Karlsruher Institut für Technologie (KIT) – IPD Tichy, Aug. 2018 (zitiert auf Seite 342).

[Haz+11] Hazen, T. J., Siu, M.-H., Gish, H., Lowe, S. und Chan, A. „Topic Modeling for Spoken Documents Using Only Phonetic Information“. In: *2011 IEEE Workshop on Automatic Speech Recognition Understanding*. Dez. 2011, S. 395–400. DOI: 10.1109/ASRU.2011.6163964 (zitiert auf Seite 185).

[HDW94] Holmes, G., Donkin, A. und Witten, I. H. „WEKA: A Machine Learning Workbench“. In: *Proceedings of ANZIIS ’94 - Australian New Zealnd Intelligent Information Systems Conference*. Nov. 1994, S. 357–361. DOI: 10.1109/ANZIIS.1994.396988 (zitiert auf Seite 181).

[Hey16] Hey, T. „Kontext- und Korreferenzanalyse für gesprochene Sprache“. Master's thesis. Karlsruher Institut für Technologie (KIT) – IPD Tichy, Sep. 2016. ISBN: 9781550657326 (zitiert auf den Seiten 164 und 205).

[Hey19] Hey, T. „INDIRECT: Intent-Driven Requirements-to-Code Traceability“. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. Mai 2019, S. 190–191. DOI: 10.1109/ICSE-Companion.2019.00078 (zitiert auf den Seiten 100 und 344).

[Hil72] Hill, I. D. „Wouldn't It Be Nice If We Could Write Computer Programs in Ordinary English - or Would It?“ In: *Computer Bulletin* 16.6 (1972), S. 306–312 (zitiert auf Seite 67).

[Hin+13] Hingmire, S., Chougule, S., Palshikar, G. K. und Chakraborti, S. „Document Classification by Topic Labeling“. In: *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '13. New York, NY, USA: ACM, 2013, S. 877–880. ISBN: 978-1-4503-2034-4. DOI: 10.1145/2484028.2484140 (zitiert auf Seite 185).

[Hin83] Hindle, D. „Deterministic Parsing of Syntactic Non-Fluencies“. In: *Proceedings of the 21st Annual Meeting on Association for Computational Linguistics*. ACL '83. Stroudsburg, PA, USA: Association for Computational Linguistics, 1983, S. 123–128. DOI: 10.3115/981311.981336 (zitiert auf Seite 171).

[HJ14] Honnibal, M. und Johnson, M. „Joint Incremental Disfluency Detection and Dependency Parsing“. In: *Transactions of the Association for Computational Linguistics* 2 (Dez. 2014), S. 131–142. DOI: 10.1162/tacl_a_00171 (zitiert auf Seite 171).

[Hob78] Hobbs, J. R. „Resolving Pronoun References“. In: *Lingua* 44.4 (Apr. 1978), S. 311–338. ISSN: 0024-3841. DOI: 10.1016/0024-3841(78)90006-2 (zitiert auf den Seiten 36, 206, und 210).

[HQ08] Hu, W. und Qu, Y. „Falcon-Ao: A Practical Ontology Matching System“. In: *Journal of Web Semantics*. World Wide Web Conference 2007 - Semantic Web Track 6.3 (Sep. 2008), S. 237–239. ISSN: 1570-8268. DOI: 10.1016/j.websem.2008.02.006 (zitiert auf Seite 270).

[HS97] Hochreiter, S. und Schmidhuber, J. „Long Short-Term Memory“. In: *Neural Computation* 9.8 (1997), S. 1735–1780. ISSN: 1530-888X(Electronic),0899-7667(Print). DOI: 10.1162/neco.1997.9.8.1735 (zitiert auf Seite 22).

[HTS73] Hendrix, G. G., Thompson, C. W. und Slocum, J. „Language Processing via Canonical Verbs and Semantic Models“. In: *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*. IJCAI'73. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Aug. 1973, S. 262–269 (zitiert auf Seite 38).

[Hul+13] Hulpus, I., Hayes, C., Karnstedt, M. und Greene, D. „Unsupervised Graph-Based Topic Labelling Using Dbpedia“. In: *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*. WSDM '13. New York, NY, USA: ACM, 2013, S. 465–474. ISBN: 978-1-4503-1869-3. DOI: 10.1145/2433396.2433454 (zitiert auf den Seiten 185, 186, und 191).

[IV95] Ide, N. und Véronis, J., Hrsg. *Text Encoding Initiative: Background and Context*. Springer Netherlands, 1995. ISBN: 978-0-7923-3689-1. DOI: 10.1007/978-94-011-0325-1 (zitiert auf Seite 109).

[Iye+17] Iyer, S., Konstas, I., Cheung, A., Krishnamurthy, J. und Zettlemoyer, L. „Learning a Neural Semantic Parser from User Feedback“. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Vancouver, Canada: Association for Computational Linguistics, 2017, S. 963–973. DOI: 10.18653/v1/P17-1089 (zitiert auf den Seiten 89 und 97).

[Jar89] Jaro, M. A. „Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida“. In: *Journal of the American Statistical Association* 84.406 (Juni 1989), S. 414–420. ISSN: 0162-1459. DOI: 10.1080/01621459.1989.10478785 (zitiert auf Seite 50).

[JJ17] Jamshid Lou, P. und Johnson, M. „Disfluency Detection Using a Noisy Channel Model and a Deep Neural Language Model“. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vancouver, Canada: Association for Computational Linguistics, Juli 2017, S. 547–553. DOI: 10.18653/v1/P17-2087 (zitiert auf Seite 171).

[JM09a] Jokinen, K. und McTear, M. „Spoken Dialogue Systems“. In: *Synthesis Lectures on Human Language Technologies* 2.1 (Jan. 2009), S. 1–151. ISSN: 1947-4040. DOI: 10.2200/S00204ED1V01Y200910HLT005 (zitiert auf den Seiten 44, 45, und 280).

[JM09b] Jurafsky, D. und Martin, J. H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. 2. ed., [Pearson International Edition]. Prentice Hall Series in Artificial Intelligence. Englewood Cliffs, NJ: Prentice Hall, Pearson Education International, 2009. ISBN: 0-13-504196-1 978-0-13-504196-3 (zitiert auf den Seiten 23, 24, 27, 29, 31, 33, 36, 37, 39, 40, 41, 42, 57, und 64).

[Jou+17] Joulin, A., Grave, E., Bojanowski, P. und Mikolov, T. „Bag of Tricks for Efficient Text Classification“. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, S. 427–431 (zitiert auf den Seiten 59 und 240).

[Kas66] Kasami, T. „An Efficient Recognition and Syntax-Analysis Algorithm for Context-Free Languages“. In: *Coordinated Science Laboratory Report no. R-257* (1966) (zitiert auf Seite 34).

[KB82] Kaplan, R. M. und Bresnan, J. „Lexical-Functional Grammar: A Formal System for Grammatical Representation“. In: *Formal Issues in Lexical-Functional Grammar* (1982), S. 29–130 (zitiert auf Seite 217).

[KG19] Kantor, B. und Globerson, A. „Coreference Resolution with Entity Equalization“. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Juli 2019, S. 673–677. DOI: 10.18653/v1/P19-1066 (zitiert auf Seite 204).

[Kip+06] Kipper, K., Korhonen, A., Ryant, N. und Palmer, M. „Extending VerbNet with Novel Verb Classes“. In: *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*. Genoa, Italy: European Language Resources Association (ELRA), Mai 2006 (zitiert auf Seite 54).

[KK12] Khan, Z. und Keet, C. M. „ONSET: Automated Foundational Ontology Selection and Explanation“. In: *Knowledge Engineering and Knowledge Management*. Hrsg. von A. ten Teije, J. Völker, S. Handschuh, H. Stuckenschmidt, M. d'Acquin, A. Nikolov, N. Aussenac-Gilles und N. Hernandez. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, S. 237–251. ISBN: 978-3-642-33876-2. DOI: 10.1007/978-3-642-33876-2_22 (zitiert auf Seite 269).

[KK19] Keim, J. und Koziolek, A. „Towards Consistency Checking Between Software Architecture and Informal Documentation“. In: *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. März 2019, S. 250–253. DOI: 10.1109/ICSA-C.2019.00052 (zitiert auf den Seiten 101 und 344).

[KM06] Knöll, R. und Mezini, M. „Pegasus: First Steps Toward a Naturalistic Programming Language“. In: *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications*. OOPSLA '06. Portland, Oregon, USA: ACM, 2006, S. 542–559. ISBN: 1-59593-491-X. DOI: 10.1145/1176617.1176628 (zitiert auf den Seiten 72, 96, 109, 110, und 112).

[Koc15] Kocybik, M. „Projektion von Gesprochener Sprache Auf Eine Handlungsrepräsentation“. Bachelor's Thesis. Karlsruher Institut für Technologie (KIT) – IPD Tichy, Juli 2015 (zitiert auf Seite 158).

[Kol+10] Kollar, T., Tellex, S., Roy, D. und Roy, N. „Toward Understanding Natural Language Directions“. In: *Proceedings of the 5th ACM/IEEE International Conference on Human-Robot Interaction*. HRI '10. Piscataway, NJ, USA: IEEE Press, 2010, S. 259–266. ISBN: 978-1-4244-4893-7. DOI: 10.1109/HRI.2010.5453186 (zitiert auf Seite 80).

[Kon+17] Konstas, I., Iyer, S., Yatskar, M., Choi, Y. und Zettlemoyer, L. „Neural AMR: Sequence-to-Sequence Models for Parsing and Generation“. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, Juli 2017, S. 146–157. DOI: 10.18653/v1/P17-1014 (zitiert auf Seite 76).

[Koz97] Kozen, D. C. „The Cocke—Kasami—Younger Algorithm“. In: *Automata and Computability*. Hrsg. von D. C. Kozen. Undergraduate Texts in Computer Science. New York, NY: Springer, 1997, S. 191–197. ISBN: 978-1-4612-1844-9. DOI: 10.1007/978-1-4612-1844-9_33 (zitiert auf Seite 34).

[KP02] Kingsbury, P. und Palmer, M. „From TreeBank to PropBank“. In: *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*. Las Palmas, Canary Islands - Spain: European Language Resources Association (ELRA), Mai 2002 (zitiert auf Seite 54).

[KS06] Kiss, T. und Strunk, J. „Unsupervised Multilingual Sentence Boundary Detection“. In: *Computational Linguistics* 32.4 (Nov. 2006), S. 485–525. ISSN: 0891-2017. DOI: 10.1162/coli.2006.32.4.485 (zitiert auf Seite 28).

[KSK19] Keim, J., Schneider, Y. und Koziolek, A. „Towards Consistency Analysis between Formal and Informal Software Architecture Artefacts“. In: *2019 IEEE/ACM 2nd International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE)*. Mai 2019, S. 6–12. DOI: 10.1109/ECASE.2019.00010 (zitiert auf den Seiten 101 und 344).

[LCL18] Liu, J., Cohen, S. B. und Lapata, M. „Discourse Representation Structure Parsing“. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, Juli 2018, S. 429–439. DOI: 10.18653/v1/P18-1040 (zitiert auf Seite 76).

[Lee+11] Lee, H., Peirsman, Y., Chang, A., Chambers, N., Surdeanu, M. und Jurafsky, D. „Stanford's Multi-Pass Sieve Coreference Resolution System at the CoNLL-2011 Shared Task“. In: *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*. CONLL Shared Task '11. Portland, Oregon: Association for Computational Linguistics, Juni 2011, S. 28–34. ISBN: 978-1-937284-08-4 (zitiert auf den Seiten 204 und 215).

[Lee+14] Lee, K., Artzi, Y., Dodge, J. und Zettlemoyer, L. „Context-Dependent Semantic Parsing for Time Expressions“. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, Juni 2014, S. 1437–1447. DOI: 10.3115/v1/P14-1135 (zitiert auf Seite 75).

[Leh+15] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S. und Bizer, C. „DBpedia – A Large-Scale, Multilingual Knowledge Base Extracted from Wikipedia“. In: *Semantic Web* 6.2 (Jan. 2015), S. 167–195. ISSN: 1570-0844. DOI: 10.3233/SW-140134 (zitiert auf Seite 56).

[Len95] Lenat, D. B. „CYC: A Large-Scale Investment in Knowledge Infrastructure“. In: *Commun. ACM* 38.11 (Nov. 1995), S. 33–38. ISSN: 0001-0782. DOI: 10.1145/219717.219745 (zitiert auf Seite 56).

[Lev66] Levenshtein, V. I. „Binary Codes Capable of Correcting Deletions, Insertions and Reversals“. In: *Soviet physics doklady* 10.8 (1966), S. 707–710 (zitiert auf Seite 49).

[LGS13] Le, V., Gulwani, S. und Su, Z. „SmartSynth: Synthesizing Smartphone Automation Scripts from Natural Language“. In: *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services - MobiSys '13*. Taipei, Taiwan: ACM Press, 2013, S. 193. ISBN: 978-1-4503-1672-9. DOI: 10.1145/2462456.2464443 (zitiert auf den Seiten 73, 96, 102, und 216).

[LH07] Litkowski, K. und Hargraves, O. „SemEval-2007 Task 06: Word-Sense Disambiguation of Prepositions“. In: *Proceedings of the 4th International Workshop on Semantic Evaluations*. SemEval '07. USA: Association for Computational Linguistics, Juni 2007, S. 24–29 (zitiert auf Seite 42).

[LH15] Landhäußer, M. und Hug, R. „Text Understanding for Programming in Natural Language: Control Structures“. In: *Proceedings of the 4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*. Mai 2015. DOI: 10.1109/RAISE.2015.9 (zitiert auf den Seiten 217 und 225).

[Li+16] Li, J., Monroe, W., Ritter, A., Galley, M., Gao, J. und Jurafsky, D. „Deep Reinforcement Learning for Dialogue Generation“. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, S. 1192–1202. DOI: 10.18653/v1/D16-1127 (zitiert auf Seite 44).

[Li+18] Li, T. J., Labutov, I., Li, X. N., Zhang, X., Shi, W., Ding, W., Mitchell, T. M. und Myers, B. A. „APPINITE: A Multi-Modal Interface for Specifying Data Descriptions in Programming by Demonstration Using Natural Language Instructions“. In: *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Okt. 2018, S. 105–114. DOI: 10.1109/VLHCC.2018.8506506 (zitiert auf den Seiten 83, 97, und 99).

[Lie+06] Lieberman, H., Paternò, F., Klann, M. und Wulf, V. „End-User Development: An Emerging Paradigm“. In: *End User Development*. Hrsg. von H. Lieberman, F. Paternò und V. Wulf. Human-Computer Interaction Series. Dordrecht: Springer Netherlands, 2006, S. 1–8. ISBN: 978-1-4020-5386-3. DOI: 10.1007/1-4020-5386-X_1 (zitiert auf Seite 11).

[Lin+16] Ling, W., Blunsom, P., Grefenstette, E., Hermann, K. M., Kočiský, T., Wang, F. und Senior, A. „Latent Predictor Networks for Code Generation“. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, S. 599–609. DOI: 10.18653/v1/P16-1057 (zitiert auf den Seiten 96 und 139).

[Lin+18] Lin, X. V., Wang, C., Zettlemoyer, L. und Ernst, M. D. „NL2Bash: A Corpus and Semantic Parser for Natural Language Interface to the Linux Operating System“. In: *Proceedings of the Eleventh International Conference on Language Resources and*

Evaluation (LREC 2018). Miyazaki, Japan: European Language Resources Association (ELRA), Mai 2018 (zitiert auf den Seiten 80, 96, und 138).

[Lin03] Lin, D. „Dependency-Based Evaluation of Minipar“. In: *Treebanks: Building and Using Parsed Corpora*. Hrsg. von A. Abeillé. Text, Speech and Language Technology. Dordrecht: Springer Netherlands, 2003, S. 317–329. ISBN: 978-94-010-0201-1. DOI: 10.1007/978-94-010-0201-1_18 (zitiert auf Seite 34).

[Lin98] Lin, D. „Automatic Retrieval and Clustering of Similar Words“. In: *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 2*. ACL '98/COLING '98. USA: Association for Computational Linguistics, Aug. 1998, S. 768–774. DOI: 10.3115/980691.980696 (zitiert auf Seite 42).

[LK77] Landis, J. R. und Koch, G. G. „The measurement of observer agreement for categorical data“. In: *Biometrics* 33.1 (Jan. 1977), S. 159–174. ISSN: 0006-341X. DOI: 10.2307/2529310 (zitiert auf den Seiten 191 und 310).

[LL04] Liu, H. und Lieberman, H. „Toward a Programmatic Semantics of Natural Language“. In: *Visual Languages and Human Centric Computing, 2004 IEEE Symposium On*. Sep. 2004, S. 281–282. DOI: 10.1109/VLHCC.2004.59 (zitiert auf Seite 70).

[LL05a] Liu, H. und Lieberman, H. „Metafor: Visualizing Stories as Code“. In: *IUI '05: Proceedings of the 10th International Conference on Intelligent User Interfaces*. San Diego, California, USA: ACM, 2005, S. 305–307. ISBN: 1-58113-894-6. DOI: 10.1145/1040830.1040908 (zitiert auf den Seiten 70, 95, und 217).

[LL05b] Liu, H. und Lieberman, H. „Programmatic Semantics for Natural Language Interfaces“. In: *CHI '05 Extended Abstracts on Human Factors in Computing Systems*. CHI '05. Portland, OR, USA: ACM, 2005, S. 1597–1600. ISBN: 1-59593-002-7. DOI: 10.1145/1056808.1056975 (zitiert auf den Seiten 70 und 95).

[LL06] Lieberman, H. und Liu, H. „Feasibility Studies for Programming in Natural Language“. In: *End User Development*. Hrsg. von H. Lieberman, F. Paternò und V. Wulf. Human-Computer Interaction Series 9. Dordrecht: Springer Netherlands, 2006, S. 459–473. ISBN: 978-1-4020-5386-3. DOI: 10.1007/1-4020-5386-X_20 (zitiert auf Seite 68).

[LM04] Lopez, V. und Motta, E. „Ontology-Driven Question Answering in AquaLog“. In: *Natural Language Processing and Information Systems, 9th International Conference on Applications of Natural Languages to Information Systems, NLDB 2004, Salford, UK, June 23-25, 2004, Proceedings*. 2004, S. 89–102. DOI: 10.1007/978-3-540-27779-8_8 (zitiert auf den Seiten 87 und 97).

[LM06] Little, G. und Miller, R. C. „Translating Keyword Commands into Executable Code“. In: *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*. UIST '06. New York, NY, USA: ACM, 2006, S. 135–144. ISBN: 978-1-59593-313-3. DOI: 10.1145/1166253.1166275 (zitiert auf den Seiten 73 und 96).

[LMU06] Lopez, V., Motta, E. und Uren, V. „PowerAqua: Fishing the Semantic Web“. In: *The Semantic Web: Research and Applications*. Hrsg. von Y. Sure und J. Domingue. Lecture Notes in Computer Science 4011. Springer Berlin Heidelberg, 2006, S. 393–410. ISBN: 978-3-540-34544-2 978-3-540-34545-9 (zitiert auf den Seiten 88 und 97).

[Lop+07] Lopez, V., Uren, V., Motta, E. und Pasin, M. „AquaLog: An Ontology-Driven Question Answering System for Organizational Semantic Intranets“. In: *Journal of Web Semantics*. Software Engineering and the Semantic Web 5.2 (Juni 2007), S. 72–105. ISSN: 1570-8268. DOI: 10.1016/j.websem.2007.03.003 (zitiert auf den Seiten 87 und 97).

[Lop+12] Lopez, V., Fernández, M., Motta, E. und Stieler, N. „PowerAqua: Supporting Users in Querying and Exploring the Semantic Web“. In: *Semantic Web* 3.3 (Jan. 2012), S. 249–265. DOI: 10.3233/SW-2011-0030 (zitiert auf den Seiten 88 und 97).

[LPL16] Long, R., Pasupat, P. und Liang, P. „Simpler Context-Dependent Logical Forms via Model Projections“. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, S. 1456–1465. DOI: 10.18653/v1/P16-1138 (zitiert auf den Seiten 76 und 96).

[LPM05] Lopez, V., Pasin, M. und Motta, E. „AquaLog: An Ontology-Portable Question Answering System for the Semantic Web“. In: *The Semantic Web: Research and Applications*. Hrsg. von A. Gómez-Pérez und J. Euzenat. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, S. 546–562. ISBN: 978-3-540-31547-6. DOI: 10.1007/11431053_37 (zitiert auf den Seiten 87 und 97).

[LQG18] López, G., Quesada, L. und Guerrero, L. A. „Alexa vs. Siri vs. Cortana vs. Google Assistant: A Comparison of Speech-Based Natural User Interfaces“. In: *Advances in Human Factors and Systems Interaction*. Hrsg. von I. L. Nunes. Advances in Intelligent Systems and Computing. Springer International Publishing, 2018, S. 241–250. ISBN: 978-3-319-60366-7 (zitiert auf Seite 91).

[LSM06] Lopez, V., Sabou, M. und Motta, E. „PowerMap: Mapping the Real Semantic Web on the Fly“. In: *The Semantic Web - ISWC 2006*. Hrsg. von I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold und L. M. Aroyo. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, S. 414–427. ISBN: 978-3-540-49055-5. DOI: 10.1007/11926078_30 (zitiert auf Seite 269).

[LV12] Lincoln, N. K. und Veres, S. M. „Natural Language Programming of Complex Robotic BDI Agents“. In: *Journal of Intelligent & Robotic Systems* 71.2 (Sep. 2012), S. 211–230. ISSN: 0921-0296, 1573-0409. DOI: 10.1007/s10846-012-9779-1 (zitiert auf den Seiten 81 und 97).

[LWT17a] Landhäußer, M., Weigelt, S. und Tichy, W. F. „NLCI - A Natural Language Command Interpreter“. In: *Software Engineering 2017, Fachtagung Des GI-Fachbereichs Softwaretechnik, 21.-24. Februar 2017, Hannover, Deutschland*. Hrsg. von J. Jürjens und K. Schneider. Bd. P-267. LNI. GI, 2017, S. 139–140 (zitiert auf Seite 74).

[LWT17b] Landhäußer, M., Weigelt, S. und Tichy, W. F. „NLCI: A Natural Language Command Interpreter“. In: *Automated Software Engineering* 24.4 (Dez. 2017), S. 839–861. ISSN: 0928-8910. DOI: 10.1007/s10515-016-0202-1 (zitiert auf den Seiten 74, 96, 100, 107, 140, und 339).

[MA16] Misra, D. K. und Artzi, Y. „Neural Shift-Reduce CCG Semantic Parsing“. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, S. 1775–1786. DOI: 10.18653/v1/D16-1183 (zitiert auf Seite 75).

[Mag+09] Magatti, D., Calegari, S., Ciucci, D. und Stella, F. „Automatic Labeling of Topics“. In: *2009 Ninth International Conference on Intelligent Systems Design and Applications*. Nov. 2009, S. 1227–1232. DOI: 10.1109/ISDA.2009.165 (zitiert auf Seite 185).

[Man+14] Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. und McClosky, D. „The Stanford CoreNLP Natural Language Processing Toolkit“. In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Baltimore, Maryland: Association for Computational Linguistics, 2014, S. 55–60. DOI: 10.3115/v1/P14-5010 (zitiert auf den Seiten 27 und 31).

[Man11] Manning, C. D. „Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics?“ In: *Proceedings of the 12th International Conference on Computational Linguistics and Intelligent Text Processing - Volume Part I*. CICLing'11. Berlin, Heidelberg: Springer-Verlag, 2011, S. 171–189. ISBN: 978-3-642-19399-6 (zitiert auf Seite 30).

[Mar+17] Markievicz, I., Tamosiunaite, M., Vitkute-Adzgauskiene, D., Kapociute-Dziki-ene, J., Valteryte, R. und Krilavicius, T. „Reading Comprehension of Natural Language Instructions by Robots“. In: *Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation*. Springer, Cham, Mai 2017, S. 288–301. DOI: 10.1007/978-3-319-58274-0_24 (zitiert auf den Seiten 82 und 97).

[Mar+99] Marcus, M. P., Santorini, B., Marcinkiewicz, M. A. und Taylor, A. „Treebank-3“. In: *Linguistic Data Consortium, Philadelphia* 14 (1999) (zitiert auf Seite 171).

[Mat+13] Matuszek, C., Herbst, E., Zettlemoyer, L. und Fox, D. „Learning to Parse Natural Language Commands to a Robot Control System“. In: *Experimental Robotics: The 13th International Symposium on Experimental Robotics*. Hrsg. von J. P. Desai, G. Dudek, O. Khatib und V. Kumar. Bd. 88. Springer Tracts in Advanced Robotics. Heidelberg: Springer International Publishing, 2013, S. 403–415. ISBN: 978-3-319-00065-7. DOI: 10.1007/978-3-319-00065-7_28 (zitiert auf den Seiten 81 und 97).

[MBS00] Mangu, L., Brill, E. und Stolcke, A. „Finding Consensus in Speech Recognition: Word Error Minimization and Other Applications of Confusion Networks“. In: *Computer Speech & Language* 4.14 (2000), S. 373–400. ISSN: 0885-2308. DOI: 10.1006/csla.2000.0152 (zitiert auf den Seiten 154 und 155).

[MC07] Mihalcea, R. und Csomai, A. „Wikify!: Linking Documents to Encyclopedic Knowledge“. In: *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*. CIKM '07. New York, NY, USA: ACM, 2007, S. 233–242. ISBN: 978-1-59593-803-9. DOI: 10.1145/1321440.1321475 (zitiert auf den Seiten 178 und 182).

[McE19] McElvenny, J. *Form and Formalism in Linguistics*. Language Science Press, Mai 2019. ISBN: 978-3-96110-182-5 978-3-96110-183-2. DOI: 10.5281/zenodo.2654375 (zitiert auf Seite 33).

[MCK04] Mihalcea, R., Chklovski, T. und Kilgarriff, A. „The Senseval-3 English Lexical Sample Task“. In: *Proceedings of SENSEVAL-3, the Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*. Barcelona, Spain: Association for Computational Linguistics, Juli 2004, S. 25–28 (zitiert auf Seite 42).

[McT98] McTear, M. F. „Modelling Spoken Dialogues with State Transition Diagrams: Experiences with the CSLU Toolkit“. In: *5th International Conference on Spoken Language Processing (ICSLP 98)*. Sydney, Australia, 1998 (zitiert auf Seite 44).

[MGA13] Manshadi, M., Gildea, D. und Allen, J. „Integrating Programming by Example and Natural Language Programming“. In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. Bd. 27. AAAI'13. Bellevue, Washington: AAAI Press, Juli 2013, S. 661–667 (zitiert auf den Seiten 83 und 97).

[Mih07] Mihalcea, R. „Using Wikipedia for Automatic Word Sense Disambiguation“. In: *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*. Rochester, New York: Association for Computational Linguistics, Apr. 2007, S. 196–203 (zitiert auf den Seiten 178 und 239).

[Mik+13a] Mikolov, T., Chen, K., Corrado, G. und Dean, J. „Efficient Estimation of Word Representations in Vector Space“. In: *arXiv:1301.3781 [cs]* (Sep. 2013). arXiv: 1301.3781 [cs] (zitiert auf Seite 59).

[Mik+13b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. und Dean, J. „Distributed Representations of Words and Phrases and Their Compositionality“. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Red Hook, NY, USA: Curran Associates Inc., Dez. 2013, S. 3111–3119 (zitiert auf Seite 59).

[Mik+18] Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C. und Joulin, A. „Advances in Pre-Training Distributed Word Representations“. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), Mai 2018 (zitiert auf Seite 240).

[Mil81] Miller, L. „Natural Language Programming: Styles, Strategies, and Contrasts“. In: *IBM Systems Journal* 20.2 (1981), S. 184–215. ISSN: 0018-8670. DOI: 10.1147/sj.202.0184 (zitiert auf Seite 140).

[Mis+15] Misra, D. K., Tao, K., Liang, P. und Saxena, A. „Environment-Driven Lexicon Induction for High-Level Instructions“. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, Juli 2015, S. 992–1002. DOI: 10.3115/v1/P15-1096 (zitiert auf den Seiten 82, 97, und 194).

[Mit97] Mitchell, T. M. *Machine Learning. 1997*. First. McGraw-Hill Science/Engineering/Math, 1997. ISBN: 0-07-042807-7 (zitiert auf den Seiten 12, 13, 16, 19, und 20).

[MLL06] Mihalcea, R., Liu, H. und Lieberman, H. „NLP (Natural Language Processing) for NLP (Natural Language Programming)“. In: *Proceedings of the 7th International Conference on Computational Linguistics and Intelligent Text Processing*. CICLing'06. Berlin, Heidelberg: Springer-Verlag, 2006, S. 319–330. ISBN: 978-3-540-32205-4. DOI: 10.1007/11671299_34 (zitiert auf den Seiten 70, 95, und 217).

[MN15] Moro, A. und Navigli, R. „SemEval-2015 Task 13: Multilingual All-Words Sense Disambiguation and Entity Linking“. In: *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*. Denver, Colorado: Association for Computational Linguistics, 2015, S. 288–297. DOI: 10.18653/v1/S15-2049 (zitiert auf Seite 42).

[MN98] McCallum, A. und Nigam, K. „A Comparison of Event Models for Naive Bayes Text Classification“. In: *AAAI-98 Workshop on Learning for Text Categorization*. 1998, S. 8 (zitiert auf Seite 180).

[MON08] Minock, M., Olofsson, P. und Näslund, A. „Towards Building Robust Natural Language Interfaces to Databases“. In: *International Conference on Application of Natural Language to Information Systems*. Springer. 2008, S. 187–198. DOI: 10.1007/978-3-540-69858-6_19 (zitiert auf den Seiten 86 und 97).

[Mor+14] Morbini, F., DeVault, D., Sagae, K., Gerten, J., Nazarian, A. und Traum, D. „FLoReS: A Forward Looking, Reward Seeking, Dialogue Manager“. In: *Natural Interaction with Robots, Knowbots and Smartphones*. Hrsg. von J. Mariani, S. Rosset, M. Garnier-Rizet und L. Devillers. New York, NY: Springer, 2014, S. 313–325. ISBN: 978-1-4614-8280-2. DOI: 10.1007/978-1-4614-8280-2_28 (zitiert auf Seite 44).

[MRN14] Moro, A., Raganato, A. und Navigli, R. „Entity Linking Meets Word Sense Disambiguation: A Unified Approach“. In: *Transactions of the Association for Computational Linguistics* 2 (Dez. 2014), S. 231–244. DOI: 10.1162/tacl_a_00179 (zitiert auf Seite 178).

[MSZ07] Mei, Q., Shen, X. und Zhai, C. „Automatic Labeling of Multinomial Topic Models“. In: *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '07. New York, NY, USA: ACM, 2007, S. 490–499. ISBN: 978-1-59593-609-7. DOI: 10.1145/1281192.1281246 (zitiert auf Seite 185).

[Mye94] Myers, B. „Challenges of HCI Design and Implementation“. In: *Interactions* 1.1 (Jan. 1994), S. 73–83. ISSN: 1072-5520. DOI: 10.1145/174800.174808 (zitiert auf Seite 318).

[Nas75] Nash-Webber, B. „The Role of Semantics in Automatic Speech Understanding“. In: *Representation and Understanding*. Hrsg. von D. G. Bobrow und A. Collins. San Diego: Morgan Kaufmann, Jan. 1975, S. 351–382. ISBN: 978-0-12-108550-6. DOI: 10.1016/B978-0-12-108550-6.50017-3 (zitiert auf Seite 38).

[NB12] Nyga, D. und Beetz, M. „Everything Robots Always Wanted to Know about Housework (but Were Afraid to Ask)“. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Okt. 2012, S. 243–250. DOI: 10.1109/IROS.2012.6385923 (zitiert auf Seite 81).

[NCM03] Nystrom, N., Clarkson, M. R. und Myers, A. C. „Polyglot: An Extensible Compiler Framework for Java“. In: *Compiler Construction*. Hrsg. von G. Hedin. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, S. 138–152. ISBN: 978-3-540-36579-2. DOI: 10.1007/3-540-36579-6_11 (zitiert auf Seite 296).

[Nid15] Nida, E. A. *A Componential Analysis of Meaning, An Introduction to Semantic Structures*. 2nd print. Reprint 2015. Berlin, Boston: De Gruyter Mouton, 2015. ISBN: 978-90-279-7927-8. DOI: 10.1515/9783110828696 (zitiert auf Seite 42).

[Nie+00] Nießen, S., Och, F. J., Leusch, G., Ney, H. u. a. „An Evaluation Tool for Machine Translation: Fast Evaluation for MT Research“. In: *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC'00)*. Athens, Greece: European Language Resources Association (ELRA), Mai 2000, S. 39–45 (zitiert auf Seite 64).

[Niv+16] Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajič, J., Manning, C. D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., Tsarfaty, R. und Zeman, D. „Universal Dependencies v1: A Multilingual Treebank Collection“. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. Portorož, Slovenia: European Language Resources Association (ELRA), Mai 2016, S. 1659–1666 (zitiert auf den Seiten 34 und 35).

[NJV13] Navigli, R., Jurgens, D. und Vannella, D. „SemEval-2013 Task 12: Multilingual Word Sense Disambiguation“. In: (2013), S. 10 (zitiert auf Seite 42).

[NS07] Nadeau, D. und Sekine, S. „A Survey of Named Entity Recognition and Classification“. In: *Lingvisticae Investigationes* 30.1 (Jan. 2007), S. 3–26. DOI: 10.1075/li.30.1.03nad (zitiert auf Seite 31).

[Oda+15] Oda, Y., Fudaba, H., Neubig, G., Hata, H., Sakti, S., Toda, T. und Nakamura, S. „Learning to Generate Pseudo-Code from Source Code Using Statistical Machine Translation“. In: *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Nov. 2015, S. 574–584. DOI: 10.1109/ASE.2015.36 (zitiert auf Seite 78).

[Oll72] Oller, J. W. „On the Relation Between Syntax, Semantics, and Pragmatics“. In: *Linguistics* 10.83 (Jan. 1972), S. 43–55. ISSN: 0024-3949, 1613-396X. DOI: 10.1515/ling.1972.10.83.43 (zitiert auf Seite 23).

[Orw49] Orwell, G. *Nineteen Eighty-Four*. London, UK: Secker & Warburg, 1949 (zitiert auf Seite 28).

[Pap+02] Papineni, K., Roukos, S., Ward, T. und Zhu, W.-J. „BLEU: A Method for Automatic Evaluation of Machine Translation“. In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, S. 311–318. DOI: 10.3115/1073083.1073135 (zitiert auf Seite 64).

[Pas15] Paskaran, D. „Evaluation Unterschiedlicher Spracherkennungssysteme in Der Domäne Humanoide Robotik“. Bachelor's Thesis. Karlsruher Institut für Technologie (KIT) – IPD Tichy, Nov. 2015 (zitiert auf Seite 148).

[Per76] Percival, W. K. „On the Historical Source of Immediate Constituent Analysis“. In: *Notes from the Linguistic Underground* (Dez. 1976), S. 229–242. DOI: 10.1163/9789004368859_015 (zitiert auf Seite 33).

[Pet+18] Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. und Zettlemoyer, L. „Deep Contextualized Word Representations“. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, Juni 2018, S. 2227–2237. DOI: 10.18653/v1/N18-1202 (zitiert auf Seite 59).

[PFS17] Porcheron, M., Fischer, J. E. und Sharples, S. „"Do Animals Have Accents?": Talking with Agents in Multi-Party Conversation“. In: *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*. CSCW '17. New York, NY, USA: Association for Computing Machinery, Feb. 2017, S. 207–219. ISBN: 978-1-4503-4335-0. DOI: 10.1145/2998181.2998298 (zitiert auf Seite 101).

[PGK05] Palmer, M., Gildea, D. und Kingsbury, P. „The Proposition Bank: An Annotated Corpus of Semantic Roles“. In: *Computational Linguistics* 31.1 (März 2005), S. 71–106. ISSN: 0891-2017. DOI: 10.1162/0891201053630264 (zitiert auf Seite 54).

[Pie+01] Pieraccini, R., Caskey, S., Dayanidhi, K., Carpenter, B. und Phillips, M. „ETUDE, a Recursive Dialog Manager with Embedded User Interface Patterns“. In: *IEEE Workshop on Automatic Speech Recognition and Understanding, 2001. ASRU '01.* 2001, S. 244–247. DOI: 10.1109/ASRU.2001.1034633 (zitiert auf Seite 44).

[Pit+05] Pitt, M. A., Johnson, K., Hume, E., Kiesling, S. und Raymond, W. „The Buckeye Corpus of Conversational Speech: Labeling Conventions and a Test of Transcriber Reliability“. In: *Speech Communication* 45.1 (Jan. 2005), S. 89–95. ISSN: 0167-6393. DOI: 10.1016/j.specom.2004.09.001 (zitiert auf Seite 144).

[PM06] Pane, J. F. und Myers, B. A. „More Natural Programming Languages and Environments“. In: *End User Development*. Hrsg. von H. Lieberman, F. Paternò und V. Wulf. Human-Computer Interaction Series. Dordrecht: Springer Netherlands, 2006, S. 31–50. ISBN: 978-1-4020-5386-3. DOI: 10.1007/1-4020-5386-X_3 (zitiert auf Seite 68).

[POA11] Park, J., Oh, S. und Ahn, J. „Ontology Selection Ranking Model for Knowledge Reuse“. In: *Expert Systems with Applications* 38.5 (Mai 2011), S. 5133–5144. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2010.10.002 (zitiert auf Seite 269).

[Por80] Porter, M. „An Algorithm for Suffix Stripping“. In: *Program* 14.3 (Jan. 1980), S. 130–137. ISSN: 0033-0337. DOI: 10.1108/eb046814 (zitiert auf Seite 30).

[Pra+11] Pradhan, S., Ramshaw, L., Marcus, M., Palmer, M., Weischedel, R. und Xue, N. „CoNLL-2011 Shared Task: Modeling Unrestricted Coreference in OntoNotes“. In: *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*. Portland, Oregon, USA: Association for Computational Linguistics, Juni 2011, S. 1–27 (zitiert auf Seite 37).

[Pra+12a] Pradhan, S., Moschitti, A., Xue, N., Uryupina, O. und Zhang, Y. „CoNLL-2012 Shared Task: Modeling Multilingual Unrestricted Coreference in OntoNotes“. In: *Joint Conference on EMNLP and CoNLL - Shared Task*. Jeju Island, Korea: Association for Computational Linguistics, Juli 2012, S. 1–40 (zitiert auf Seite 37).

[Pri+00] Price, D., Riloff, E., Zachary, J. und Harvey, B. „NaturalJava: A Natural Language Interface for Programming in Java“. In: *Proceedings of the 5th International Conference on Intelligent User Interfaces*. IUI ’00. New Orleans, Louisiana, USA: ACM, 2000, S. 207–211. ISBN: 1-58113-134-8. DOI: 10.1145/325737.325845 (zitiert auf den Seiten 69, 95, und 216).

[PRM01] Pane, J. F., Ratanamahatana, C. und Myers, B. A. „Studying the Language and Structure in Non-Programmers’ Solutions to Programming Problems“. In: *International Journal of Human-Computer Studies* 54.2 (2001), S. 237–264. ISSN: 1071-5819. DOI: 10.1006/ijhc.2000.0410 (zitiert auf den Seiten 68 und 140).

[PS94] Pollard, C. und Sag, I. A. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Aug. 1994. ISBN: 978-0-226-67447-6 (zitiert auf Seite 217).

[PSM14] Pennington, J., Socher, R. und Manning, C. „GloVe: Global Vectors for Word Representation“. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Okt. 2014, S. 1532–1543. DOI: 10.3115/v1/D14-1162 (zitiert auf den Seiten 59 und 173).

[QMG15] Quirk, C., Mooney, R. J. und Galley, M. „Language to Code: Learning Semantic Parsers for If-This-Then-That Recipes“. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing,

China: Association for Computational Linguistics, Juni 2015, S. 878–888. DOI: 10.3115/v1/P15-1085 (zitiert auf den Seiten 77, 96, 102, 139, und 216).

[Rao+10] Rao, G., Agarwal, C., Chaudhry, S., Kulkarni, N. und Patil, D. S. „Natural Language Query Processing Using Semantic Grammar“. In: *International journal on computer science and engineering* 2.2 (2010), S. 219–223 (zitiert auf den Seiten 86 und 97).

[RCN17] Raganato, A., Camacho-Collados, J. und Navigli, R. „Word Sense Disambiguation: A Unified Evaluation Framework and Empirical Comparison“. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, S. 99–110 (zitiert auf Seite 182).

[RK17] Richardson, K. und Kuhn, J. „Function Assistant: A Tool for NL Querying of APIs“. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Copenhagen, Denmark: Association for Computational Linguistics, 2017, S. 67–72. DOI: 10.18653/v1/D17-2012 (zitiert auf den Seiten 71 und 96).

[RL70] Rosenkrantz, D. J. und Lewis, P. M. „Deterministic Left Corner Parsing“. In: *11th Annual Symposium on Switching and Automata Theory (Swat 1970)*. Okt. 1970, S. 139–152. DOI: 10.1109/SWAT.1970.5 (zitiert auf Seite 34).

[RPT00] Roy, N., Pineau, J. und Thrun, S. „Spoken Dialogue Management Using Probabilistic Reasoning“. In: *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*. ACL '00. Stroudsburg, PA, USA: Association for Computational Linguistics, 2000, S. 93–100. DOI: 10.3115/1075218.1075231 (zitiert auf Seite 44).

[RSK17] Rabinovich, M., Stern, M. und Klein, D. „Abstract Syntax Networks for Code Generation and Semantic Parsing“. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, Juli 2017, S. 1139–1149. DOI: 10.18653/v1/P17-1105 (zitiert auf den Seiten 79, 96, und 292).

[RT13] Rasooli, M. S. und Tetreault, J. „Joint Parsing and Disfluency Detection in Linear Time“. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Okt. 2013, S. 124–129 (zitiert auf Seite 171).

[RWH16] Raghothaman, M., Wei, Y. und Hamadi, Y. „SWIM: Synthesizing What I Mean: Code Search and Idiomatic Snippet Synthesis“. In: *Proceedings of the 38th International Conference on Software Engineering*. ICSE '16. New York, NY, USA: ACM, 2016, S. 357–367. ISBN: 978-1-4503-3900-1. DOI: 10.1145/2884781.2884808 (zitiert auf den Seiten 71 und 96).

[RX99] Rudnicky, A. und Xu, W. „An Agenda-Based Dialog Management Architecture for Spoken Language Systems“. In: *IEEE Workshop on Automatic Speech Recognition and Understanding, 1999. ASRU '99*. 1999, S. 1–337 (zitiert auf Seite 44).

[RZ98] Roth, D. und Zelenko, D. „Part of Speech Tagging Using a Network of Linear Separators“. In: *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 2*. ACL '98/COLING '98. USA: Association for Computational Linguistics, Aug. 1998, S. 1136–1142. DOI: 10.3115/980691.980755 (zitiert auf Seite 159).

[SA09] Seddiqui, M. H. und Aono, M. „An Efficient and Scalable Algorithm for Segmented Alignment of Ontologies of Arbitrary Size“. In: *Journal of Web Semantics*. Semantic Web Challenge 2008 7.4 (Dez. 2009), S. 344–356. ISSN: 1570-8268. DOI: 10.1016/j.websem.2009.09.001 (zitiert auf Seite 270).

[SA18] Suhr, A. und Artzi, Y. „Situated Mapping of Sequential Instructions to Actions with Single-Step Reward Observation“. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, Juli 2018, S. 2072–2082. DOI: 10.18653/v1/P18-1193 (zitiert auf den Seiten 76 und 96).

[Sab+06] Sabou, M., Lopez, V., Motta, E. und Uren, V. „Ontology Selection: Ontology Evaluation on the Real Semantic Web“. In: *15th International World Wide Web Conference (WWW 2006)*. Edinburgh, Scotland, 2006 (zitiert auf Seite 269).

[Sam66] Sammet, J. E. „The Use of English As a Programming Language“. In: *Commun. ACM* 9.3 (März 1966), S. 228–230. ISSN: 0001-0782. DOI: 10.1145/365230.365274 (zitiert auf den Seiten 2 und 67).

[Sch05] Schuler, K. K. „Verbnet: A Broad-Coverage, Comprehensive Verb Lexicon“. Diss. USA: University of Pennsylvania, 2005 (zitiert auf Seite 54).

[Sch16] Scheu, S. „Aufbereitung von Spracherkennerausgaben“. Master's Thesis. Karlsruher Institut für Technologie (KIT) – IPD Tichy, Juli 2016 (zitiert auf Seite 153).

[Ser+16] Serban, I. V., Sordoni, A., Bengio, Y., Courville, A. und Pineau, J. „Building End-to-End Dialogue Systems Using Generative Hierarchical Neural Network Models“. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI'16. Phoenix, Arizona: AAAI Press, 2016, S. 3776–3783 (zitiert auf Seite 44).

[Sha48] Shannon, C. E. „A Mathematical Theory of Communication“. In: *The Bell System Technical Journal* 27.3 (Juli 1948), S. 379–423. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1948.tb01338.x (zitiert auf Seite 40).

[She+14] She, L., Cheng, Y., Chai, J. Y., Jia, Y., Yang, S. und Xi, N. „Teaching Robots New Actions through Natural Language Instructions“. In: *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*. Edinburgh, UK: IEEE, Aug. 2014, S. 868–873. ISBN: 978-1-4799-6765-0 978-1-4799-6763-6. DOI: 10.1109/ROMAN.2014.6926362 (zitiert auf den Seiten 81 und 97).

[SHF17] Sales, J., Handschuh, S. und Freitas, A. „SemEval-2017 Task 11: End-User Development Using Natural Language“. In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. 2017, S. 556–564 (zitiert auf den Seiten 68 und 139).

[Shr94] Shriberg, E. E. „Preliminaries to a Theory of Speech Disfluencies“. Dissertation. Berkeley: University of California at Berkeley, 1994 (zitiert auf den Seiten 169 und 170).

[Sim73] Simmons, R. F. „Semantic Networks : Their Computation and Use for Understanding English Sentences“. In: *Computer Models of Thought and Language* (1973), S. 63–113 (zitiert auf Seite 37).

[Siu+14] Siu, M.-h., Gish, H., Chan, A., Belfield, W. und Lowe, S. „Unsupervised Training of an HMM-Based Self-Organizing Unit Recognizer with Applications to Topic Classification and Keyword Discovery“. In: *Computer Speech & Language* 28.1 (Jan. 2014), S. 210–223. ISSN: 0885-2308. DOI: 10.1016/j.csl.2013.05.002 (zitiert auf Seite 185).

[SLM06] Sabou, M., Lopez, V. und Motta, E. „Ontology Selection for the Real Semantic Web: How to Cover the Queen’s Birthday Dinner?“ In: *Managing Knowledge in a World of Networks*. Hrsg. von S. Staab und V. Svátek. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, S. 96–111. ISBN: 978-3-540-46365-8. DOI: 10.1007/11891451_12 (zitiert auf Seite 269).

[SLM17] Srivastava, S., Labutov, I. und Mitchell, T. „Joint Concept Learning and Semantic Parsing from Natural Language Explanations“. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, S. 1527–1536. DOI: 10.18653/v1/D17-1161 (zitiert auf Seite 76).

[SM03] Strube, M. und Müller, C. „A Machine Learning Approach to Pronoun Resolution in Spoken Dialogue“. In: *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*. ACL ’03. Sapporo, Japan: Association for Computational Linguistics, Juli 2003, S. 168–175. DOI: 10.3115/1075096.1075118 (zitiert auf Seite 205).

[Sno+06] Snover, M., Dorr, B., Schwartz, R., Micciulla, L. und Makhoul, J. „A Study of Translation Edit Rate with Targeted Human Annotation“. In: *Proceedings of the 7th Biennial Conference of the Association for Machine Translation in the Americas (AMTA-2006)*. Cambridge, Massachusetts, 2006, S. 259–268 (zitiert auf Seite 65).

[Sno+09] Snover, M. G., Madnani, N., Dorr, B. und Schwartz, R. „TER-Plus: Paraphrase, Semantic, and Alignment Enhancements to Translation Edit Rate“. In: *Machine Translation* 23.2 (Sep. 2009), S. 117–127. ISSN: 1573-0573. DOI: 10.1007/s10590-009-9062-9 (zitiert auf Seite 65).

[SP00] Seneff, S. und Polifroni, J. „Dialogue Management in the Mercury Flight Reservation System“. In: *Proceedings of the 2000 ANLP/NAACL Workshop on Conversational Systems - Volume 3*. ANLP/NAACL-ConvSyst '00. Stroudsburg, PA, USA: Association for Computational Linguistics, 2000, S. 11–16. DOI: 10.3115/1117562.1117565 (zitiert auf Seite 44).

[ST93] Sleator, D. D. und Temperley, D. „Parsing English with a Link Grammar“. In: *Proceedings of the Third International Workshop on Parsing Technologies*. Tilburg, Netherlands and Durbuy, Belgium: Association for Computational Linguistics, Aug. 1993, S. 277–292 (zitiert auf Seite 34).

[Sta99] Stalnaker, R. *Context and Content: Essays on Intentionality in Speech and Thought*. Oxford University Press, 1999. ISBN: 978-0-19-823708-2 (zitiert auf Seite 194).

[Ste87] Steedman, M. „Combinatory Grammars and Parasitic Gaps“. In: *Natural Language & Linguistic Theory* 5.3 (1987), S. 403–439. ISSN: 0167-806X (zitiert auf den Seiten 75 und 109).

[TA04] Tetreault, J. R. und Allen, J. F. „Dialogue Structure and Pronoun Resolution“. In: *DAARC* (2004), S. 6 (zitiert auf Seite 205).

[TB00] Tjong Kim Sang, E. F. und Buchholz, S. „Introduction to the CoNLL-2000 Shared Task: Chunking“. In: *Proceedings of the 2nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning - Volume 7*. ConLL '00. USA: Association for Computational Linguistics, Sep. 2000, S. 127–132. DOI: 10.3115/1117601.1117631 (zitiert auf Seite 351).

[TD03] Tjong Kim Sang, E. F. und De Meulder, F. „Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition“. In: *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*. CONLL '03. Edmonton, Canada: Association for Computational Linguistics, Mai 2003, S. 142–147. DOI: 10.3115/1119176.1119195 (zitiert auf den Seiten 31, 32, und 163).

[TD11] Tur, G. und De Mori, R. *Spoken Language Understanding: Systems for Extracting Semantic Information from Speech*. John Wiley & Sons, Ltd, März 2011. ISBN: 978-0-470-68824-3 (zitiert auf den Seiten 44, 47, und 280).

[Tel+11] Tellex, S., Kollar, T., Dickerson, S., Walter, M. R., Banerjee, A. G., Teller, S. und Roy, N. „Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation“. In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*. AAAI'11. San Francisco, California: AAAI Press, 2011, S. 1507–1514 (zitiert auf den Seiten 80, 97, 139, und 140).

[Tho+15] Thomason, J., Zhang, S., Mooney, R. und Stone, P. „Learning to Interpret Natural Language Commands Through Human-Robot Dialog“. In: *Proceedings of the 24th International Conference on Artificial Intelligence*. IJCAI'15. Buenos Aires, Argentina: AAAI Press, 2015, S. 1923–1929. ISBN: 978-1-57735-738-4 (zitiert auf den Seiten 82 und 97).

[Thu+12] Thummalapenta, S., Sinha, S., Singhania, N. und Chandra, S. „Automating Test Automation“. In: *2012 34th International Conference on Software Engineering (ICSE)*. Juni 2012, S. 881–891. DOI: 10.1109/ICSE.2012.6227131 (zitiert auf den Seiten 84 und 97).

[Thu+13] Thummalapenta, S., Devaki, P., Sinha, S., Chandra, S., Gnanasundaram, S., Nagaraj, D., Kumar, S. und Kumar, S. „Efficient and Change-Resilient Test Automation: An Industrial Case Study“. In: *2013 35th International Conference on Software Engineering (ICSE)*. Mai 2013, S. 1002–1011. DOI: 10.1109/ICSE.2013.6606650 (zitiert auf den Seiten 84 und 97).

[TM00] Toutanova, K. und Manning, C. D. „Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger“. In: *Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora: Held in Conjunction with the 38th Annual Meeting of the Association for Computational Linguistics - Volume 13*. EMNLP ’00. USA: Association for Computational Linguistics, Okt. 2000, S. 63–70. DOI: 10.3115/1117794.1117802 (zitiert auf Seite 159).

[TMS03] Taylor, A., Marcus, M. und Santorini, B. „The Penn Treebank: An Overview“. In: *Treebanks: Building and Using Parsed Corpora*. Hrsg. von A. Abeillé. Text, Speech and Language Technology. Dordrecht: Springer Netherlands, 2003, S. 5–22. ISBN: 978-94-010-0201-1. DOI: 10.1007/978-94-010-0201-1_1 (zitiert auf den Seiten 29, 33, 34, 171, 349, und 351).

[Tou+03] Toutanova, K., Klein, D., Manning, C. D. und Singer, Y. „Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network“. In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*. NAACL ’03. USA: Association for Computational Linguistics, Mai 2003, S. 173–180. DOI: 10.3115/1073445.1073478 (zitiert auf Seite 159).

[TSY08] Thomson, B., Schatzmann, J. und Young, S. „Bayesian Update of Dialogue State for Robust Dialogue Systems“. In: *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*. März 2008, S. 4937–4940. DOI: 10.1109/ICASSP.2008.4518765 (zitiert auf Seite 44).

[Tur+19] Turc, I., Chang, M.-W., Lee, K. und Toutanova, K. „Well-Read Students Learn Better: On the Importance of Pre-Training Compact Models“. In: *arXiv:1908.08962 [cs]* (Sep. 2019). arXiv: 1908.08962 [cs] (zitiert auf Seite 60).

[Tur50] Turing, A. M. „Computing Machinery and Intelligence“. In: *Mind* 59.236 (1950), S. 433–460. ISSN: 0026-4423 (zitiert auf Seite 46).

[Vas+17] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. und Polosukhin, I. „Attention Is All You Need“. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Red

Hook, NY, USA: Curran Associates Inc., Dez. 2017, S. 6000–6010. ISBN: 978-1-5108-6096-4 (zitiert auf Seite 60).

[VC05] Vadas, D. und Curran, J. R. „Programming With Unrestricted Natural Language“. In: *Proceedings of the Australasian Language Technology Workshop*. Sydney, Australia, Dez. 2005, S. 191–199 (zitiert auf den Seiten 75, 96, und 217).

[Wan+07] Wang, C., Xiong, M., Zhou, Q. und Yu, Y. „PANTO: A Portable Natural Language Interface to Ontologies“. In: *The Semantic Web: Research and Applications*. Hrsg. von E. Franconi, M. Kifer und W. May. Bd. 4519. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, S. 473–487. ISBN: 978-3-540-72667-8. DOI: 10.1007/978-3-540-72667-8_34 (zitiert auf den Seiten 87 und 97).

[Wan+15] Wang, C., Pastore, F., Goknil, A., Briand, L. und Iqbal, Z. „Automatic Generation of System Test Cases from Use Case Specifications“. In: *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. ISSTA 2015. New York, NY, USA: ACM, 2015, S. 385–396. ISBN: 978-1-4503-3620-8. DOI: 10.1145/2771783.2771812 (zitiert auf den Seiten 84 und 97).

[Wan+16] Wang, S., Che, W., Liu, Y. und Liu, T. „Enhancing Neural Disfluency Detection with Hand-Crafted Features“. In: *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data*. Hrsg. von M. Sun, X. Huang, H. Lin, Z. Liu und Y. Liu. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, S. 336–347. ISBN: 978-3-319-47674-2. DOI: 10.1007/978-3-319-47674-2_28 (zitiert auf den Seiten 171, 173, 174, und 175).

[Wan+17] Wang, S. I., Ginn, S., Liang, P. und Manning, C. D. „Naturalizing a Programming Language via Interactive Learning“. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2017, S. 929–938. DOI: 10.18653/v1/P17-1086 (zitiert auf den Seiten 71 und 96).

[Wei+19] Weigelt, S., Keim, J., Hey, T. und Tichy, W. F. „Unsupervised Multi-Topic Labeling for Spoken Utterances“. In: *2019 IEEE International Conference on Humanized Computing and Communication (HCC)*. Sep. 2019, S. 38–45. DOI: 10.1109/HCC46620.2019.00014 (zitiert auf den Seiten 180 und 185).

[Wei+20a] Weigelt, S., Keim, J., Hey, T. und Tichy, W. F. „What's the Matter? Knowledge Acquisition by Unsupervised Multi-Topic Labeling for Spoken Utterances“. In: *International Journal of Humanized Computing and Communication* 1.1 (Aug. 2020), S. 43–66. ISSN: 2641-953X. DOI: 10.35708/HCC1868-126364 (zitiert auf den Seiten 185 und 270).

[Wei+20b] Weigelt, S., Steurer, V., Hey, T. und Tichy, W. F. „Programming in Natural Language with fuSE: Synthesizing Methods from Spoken Utterances Using Deep Natural Language Understanding“. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, Juli 2020, S. 4280–4295. DOI: http://dx.doi.org/10.18653/v1/2020.acl-main.395 (zitiert auf Seite 257).

[Wei+20c] Weigelt, S., Steurer, V., Hey, T. und Tichy, W. F. „Roger That! Learning How Laypersons Teach New Functions to Intelligent Systems“. In: *2020 IEEE 14th International Conference on Semantic Computing (ICSC)*. Feb. 2020, S. 93–100. DOI: 10.1109/ICSC.2020.00020 (zitiert auf Seite 235).

[Wei+20d] Weigelt, S., Steurer, V., Hey, T. und Tichy, W. F. „Towards Programming in Natural Language: Learning New Functions from Spoken Utterances“. In: *International Journal of Semantic Computing* 14.02 (Juni 2020), S. 249–272. ISSN: 1793-351X. DOI: 10.1142/S1793351X20400097 (zitiert auf Seite 235).

[Wel+13] Welke, K., Vahrenkamp, N., Wächter, M., Kröhnert, M. und Asfour, T. „The ArmarX Framework - Supporting High Level Robot Programming through State Disclosure“. In: *INFORMATIK 2013 – Informatik Angepasst an Mensch, Organisation Und Umwelt*. Hrsg. von M. Horbach. Bonn: Gesellschaft für Informatik e.V., 2013, S. 2823–2837 (zitiert auf Seite 134).

[Wen+17] Wen, T.-H., Vandyke, D., Mrkšić, N., Gašić, M., Rojas-Barahona, L. M., Su, P.-H., Ultes, S. und Young, S. „A Network-Based End-to-End Trainable Task-Oriented Dialogue System“. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, S. 438–449 (zitiert auf Seite 44).

[WHL18] Weigelt, S., Hey, T. und Landhäußer, M. „Integrating a Dialog Component into a Framework for Spoken Language Understanding“. In: *Proceedings of the 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*. RAISE ’18. New York, NY, USA: ACM, 2018, S. 1–7. DOI: 10.1145/3194104.3194105 (zitiert auf Seite 278).

[WHS18a] Weigelt, S., Hey, T. und Steurer, V. „Detection of Conditionals in Spoken Utterances“. In: *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*. Jan. 2018, S. 85–92. DOI: 10.1109/ICSC.2018.00021 (zitiert auf Seite 217).

[WHS18b] Weigelt, S., Hey, T. und Steurer, V. „Detection of Control Structures in Spoken Utterances“. In: *International Journal of Semantic Computing* 12.03 (Sep. 2018), S. 335–360. ISSN: 1793-351X. DOI: 10.1142/S1793351X18400159 (zitiert auf Seite 217).

[WHT17a] Weigelt, S., Hey, T. und Tichy, W. F. „Context Model Acquisition from Spoken Utterances“. In: *The 29th International Conference on Software Engineering & Knowledge Engineering,* Pittsburgh, PA, Juli 2017, S. 201–206. DOI: 10.18293/SEKE2017-083 (zitiert auf Seite 197).

[WHT17b] Weigelt, S., Hey, T. und Tichy, W. F. „Context Model Acquisition from Spoken Utterances“. In: *International Journal of Software Engineering and Knowledge Engineering* 27.09n10 (Nov. 2017), S. 1439–1453. ISSN: 0218-1940. DOI: 10.1142/S0218194017400058 (zitiert auf Seite 197).

[Win72] Winograd, T. „Understanding Natural Language“. In: *Cognitive Psychology* 3.1 (Jan. 1972), S. 1–191. ISSN: 0010-0285. DOI: 10.1016/0010-0285(72)90002-3 (zitiert auf Seite 68).

[Win90] Winkler, W. E. *String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage*. 1990 (zitiert auf Seite 50).

[WK92] Webster, J. J. und Kit, C. „Tokenization as the Initial Phase in NLP“. In: *Proceedings of the 14th Conference on Computational Linguistics - Volume 4*. COLING '92. USA: Association for Computational Linguistics, Aug. 1992, S. 1106–1110. DOI: 10.3115/992424.992434 (zitiert auf Seite 27).

[WLB19] Weigelt, S., Landhäußer, M. und Blersch, M. „How to Prepare an API for Programming in Natural Language“. In: *SEMPDS 2019 - Posters and Demos at SEMANTiCS 2019 - Proceedings of the Posters and Demo Track of the 15th International Conference on Semantic Systems Co-Located with 15th International Conference on Semantic Systems (SEMANTiCS 2019), Karlsruhe, Germany, September 9th to 12th, 2019. Ed.: A. Mehwish, R. Usbeck, T. Pellegrini, H. Sack, Y. Sure-Vetter*. Bd. 2451. Karlsruhe, Germany: RWTH, Aachen, 2019, S. 1–6. DOI: 10.5445/DIVA/2019-692 (zitiert auf Seite 132).

[WP94] Wu, Z. und Palmer, M. „Verbs Semantics and Lexical Selection“. In: *Proceedings of the 32Nd Annual Meeting on Association for Computational Linguistics*. ACL '94. Stroudsburg, PA, USA: Association for Computational Linguistics, 1994, S. 133–138. DOI: 10.3115/981732.981751 (zitiert auf Seite 199).

[WST20a] Weigelt, S., Steurer, V. und Tichy, W. F. „At Your Command! An Empirical Study on How Laypersons Teach Robots New Functions“. In: *2020 IEEE 14th International Conference on Semantic Computing (ICSC)*. Feb. 2020, S. 468–470. DOI: 10.1109/ICSC47212.2020.9309130 (zitiert auf den Seiten 148 und 235).

[WST20b] Weigelt, S., Steurer, V. und Tichy, W. F. „At Your Command! An Empirical Study on How Laypersons Teach Robots New Functions“. In: *arXiv:2009.06510 [Cs]*. Sep. 2020. arXiv: 2009.06510 [cs] (zitiert auf den Seiten 148 und 235).

[WT15] Weigelt, S. und Tichy, W. F. „Poster: ProNat: An Agent-Based System Design for Programming in Spoken Natural Language“. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Bd. 2. ICSE '15. Florence, Italy: IEEE Press, Mai 2015, S. 819–820. ISBN: 978-1-4799-1934-5. DOI: 10.1109/ICSE.2015.264 (zitiert auf Seite 99).

[XS14] Xu, P. und Sarikaya, R. „Contextual Domain Classification in Spoken Language Understanding Systems Using Recurrent Neural Network“. In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Mai 2014, S. 136–140. DOI: 10.1109/ICASSP.2014.6853573 (zitiert auf Seite 194).

[Yan+20] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. und Le, Q. V. „XLNet: Generalized Autoregressive Pretraining for Language Understanding“. In: *arXiv:1906.08237 [cs]* (Jan. 2020). arXiv: 1906.08237 [cs] (zitiert auf Seite 60).

[YN17] Yin, P. und Neubig, G. „A Syntactic Neural Model for General-Purpose Code Generation“. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, Juli 2017, S. 440–450. DOI: 10.18653/v1/P17-1041 (zitiert auf den Seiten 78, 96, und 292).

[You+10] Young, S., Gašić, M., Keizer, S., Mairesse, F., Schatzmann, J., Thomson, B. und Yu, K. „The Hidden Information State Model: A Practical Framework for POMDP-Based Spoken Dialogue Management“. In: *Computer Speech & Language* 24.2 (Apr. 2010), S. 150–174. ISSN: 0885-2308. DOI: 10.1016/j.csl.2009.04.001 (zitiert auf den Seiten 44 und 194).

[You67] Younger, D. H. „Recognition and Parsing of Context-Free Languages in Time N3“. In: *Information and Control* 10.2 (Feb. 1967), S. 189–208. ISSN: 0019-9958. DOI: 10.1016/S0019-9958(67)80007-X (zitiert auf Seite 34).

[ZC05] Zettlemoyer, L. S. und Collins, M. „Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars“. In: *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*. UAI'05. Arlington, Virginia, United States: AUAI Press, 2005, S. 658–666. ISBN: 978-0-9749039-1-0 (zitiert auf den Seiten 75, 109, 112, und 217).

[ZC07] Zettlemoyer, L. S. und Collins, M. „Online Learning of Relaxed CCG Grammars for Parsing to Logical Form“. In: *In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL-2007*. Prague, Czech Republic, Juni 2007, S. 678–687 (zitiert auf den Seiten 75 und 217).

[ZM96] Zelle, J. M. und Mooney, R. J. „Learning to Parse Database Queries Using Inductive Logic Programming“. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*. AAAI'96. Portland, Oregon: AAAI Press, Aug. 1996, S. 1050–1055. ISBN: 978-0-262-51091-2 (zitiert auf den Seiten 78, 85, 97, und 138).

[ZOH14] Zayats, V., Ostendorf, M. und Hajishirzi, H. „Multi-Domain Disfluency and Repair Detection“. In: *Fifteenth Annual Conference of the International Speech Communication Association*. 2014 (zitiert auf Seite 171).

[ZOH16] Zayats, V., Ostendorf, M. und Hajishirzi, H. „Disfluency Detection Using a Bidirectional LSTM“. In: *Interspeech 2016*. Sep. 2016, S. 2523–2527. DOI: 10.21437/Interspeech.2016-1247 (zitiert auf den Seiten 171, 172, und 173).

[ZXS17] Zhong, V., Xiong, C. und Socher, R. „Seq2SQL: Generating Structured Queries from Natural Language Using Reinforcement Learning“. In: *arXiv:1709.00103 [cs]* (Nov. 2017). arXiv: 1709.00103 [cs] (zitiert auf den Seiten 79, 88, 97, und 138).