Karlsruhe Institute of Technology

Vollständigkeits- und Semantikprüfung für gesprochene Aussagen

Bachelorarbeit
von

Daniel Hamann

An der Fakultät für Informatik
Institut für Programmstrukturen
und Datenorganisation (IPD)

Erstgutachter:	Prof. Dr. Walter F. Tichy
Zweitgutachter:	Prof. Dr. Ralf H. Reussner
Betreuender Mitarbeiter:	Dipl.-Inform. Sebastian Weigelt
Zweiter betr. Mitarbeiter:	M. Sc. Tobias Hey

Bearbeitungszeit: 18.04.2018 – 17.08.2018

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Die Regeln zur Sicherung guter wissenschaftlicher Praxis im Karlsruher Institut für Technologie (KIT) habe ich befolgt.

Karlsruhe, 17.08.2018

. . . . . .

(**Daniel Hamann**)

Kurzfassung

Diese Arbeit betrachtet das Problem von unvollständigen Aussagen in natürlicher Sprache. Solche Aussagen enthalten nicht explizit alle Informationen, die für die vollständige Umsetzung einer Handlung benötigt werden. Menschen fällt es leicht, aus solchen Aussagen wieder einen Sinn zu extrahieren, indem sie sich auf ihr gelerntes Wissen beziehen. Für Sprachverarbeitungssysteme ist dieser Schritt jedoch nicht so einfach nachvollziehbar. Um diesen Prozess greifbar zu machen wird ein Ansatz entwickelt, der solchen Systemen menschliches, implizites, Wissen verfügbar macht. Damit wird eine durch Wissen begründete Entscheidung über die Vollständigkeit von Aussagen gemacht. Im Weiteren werden durch den Ansatz Kandidaten generiert, welche für die fehlenden Rollen infrage kommen könnten. Diese werden mithilfe von Wissen über die Welt bezüglich ihrer Eignung für die Rolle bewertet und die Aussage somit wieder vervollständigt. Um die Funktion des implementierten PARSE-Agenten zu prüfen wurde eine Evaluation der Ergebnisse im Vergleich zu einer händisch erstellten Lösung durchgeführt. Fehlende Rollen konnten mit einer Präzision von 51% bestimmt werden und eine Vervollständigung war in 25%(Im Falle des Subjektes sogar 100%) der Fälle korrekt möglich.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Die Steuerung von Softwaresystemen mit natürlichsprachlichen Anweisungen ist eine immer präsentere Form der Interaktion mit Software. Mobile Assistenten wie Googles Assistant oder Apples Siri bieten ein breites vorgefertigtes Spektrum an Funktonalität an. Doch gerade das Erlernen von neuen Funktionen ist ein aufwendiger Prozess, denn die Programmierung solcher Funktionen ist nicht einfach und nur mit einer großen Menge an Hintergrundwissen möglich. Um diesen Prozess auch für Laien zu vereinfachen soll mittels des PARSE-Projektes eine Schnittstelle zwischen Computersystemen und Menschen erstellt werden, welche natürlichsprachliche Anweisungen in, vom Zielsystem verständlichen, Quelltext übersetzt. Um eine natürliche Form der Interaktion zu bieten, muss solch ein System mit vielen verschiedenen Formen von Aussagen umgehen können. Da Menschen beim Erklären von Prozessen aber auch ungenaue oder unvollständige Anweisungen geben können, müssen auch dieser Fakt von einem Sprachverarbeitungssystem erfasst und sinnvoll behandelt werden. Unvollständige Aussagen können dabei auf verschiedene Weisen auftreten. Ein Sprecher könnte implizites Wissen nicht erneut genannt oder aber auch vergessen haben. Ein Beispiel für den ersten Fall ist:

> John, go and get the orange juice.

In diesem Beispiel ist zwar für „get“ ausreichend Information gegeben, für „go“ fehlt jedoch der Ort, zu dem sich John bewegen soll. Menschen fällt es leicht sich aus der aktuellen Umgebung und aus erlernten Regeln die Aussage zu vervollständigen. Das Wissen über den Aufbewahrungsort könnte zum Beispiel über die vorangehende Aktion und das damit verknüpfte Wissen „I have put the juice in the fridge earlier.“ inferiert werden. Während Menschen sich auf diese Umgebungsannahmen und Allgemeinwissen stützen können, ist ein solches Verständnis für Computersysteme nicht vorhanden. Ein System um dieses Verständnis zu erweitern, ist eine wichtige Voraussetzung um die Intention des Sprechers richtig zu deuten und Fehler zu vermeiden. Dazu muss verstanden werden, wann eine Aussage unvollständig ist und wie sich dies erkennen lässt. Die Zielsetzung der Arbeit ist es, ein Werkzeug zu entwickeln, welches in Eingaben in natürlicher Sprache erkennen kann, ob diese unvollständig sind. Ebenso soll das Werkzeug in der Lage sein diese Unvollständigkeiten wieder zu beheben. Die Arbeit ist wie folgt strukturiert: Zuerst werden in Kapitel 2 die wichtigsten Themen, die für das Verständnis der Arbeit essentiell sind vorgestellt. Kapitel 3 stellt relevante Arbeiten aus ähnlichen Gebieten vor. Darauffolgend findet sich eine ausführliche Analyse der Problemstellung in Kapitel 4, gefolgt von einem Entwurf einer Lösung in Kapitel 5 und der Dokumentation der Implementierung in Kapitel 6. Die Güte

des Ansatzes wird in Kapitel 7 ausführlich evaluiert und mögliche Fehlerquellen und Verbesserungen werden diskutiert. Das letzte Kapitel fasst die Arbeit zusammen, und nennt noch einige interessante Problemstellungen die in zukünftigen Arbeiten betrachtet werden können.

2. Grundlagen

In diesem Kapitel werden die grundlegenden Begriffe definiert und die verwendeten Werkzeuge vorgestellt.

2.1. Syntax

Syntax befasst sich in der Linguistik mit der Struktur von Sätzen. Durch die Syntax einer Sprache wird ein Regelsatz vorgegeben, welcher den Aufbau von grammatikalisch korrekten Sätzen beschreibt. Diese Regeln lassen sich formal mittels kontextfreier Grammatiken darstellen. In diesen sind Abbildungsregeln definiert.

NP	→	*Det Nominal*
NP	→	*ProperNoun*
Nominal	→	*Noun* | *Nominal Noun*
Det	→	a
Det	→	the
Noun	→	flight

Tabelle 2.1.: Grammatik-Regeln für simple Nominalphrasen [JM14]

Jede dieser Regeln gibt eine bestimmte Ableitung vor, nach denen Symbole ersetzt werden. Diese werden solange angewendet, bis alle Nichtterminalsymbole auf Terminalsymbole, meistens Wörter, abgebildet wurden. Mit den obigen Regeln lässt sich aus dem Grundbaustein *NP* wie folgt ein valider Satz abgeleitet werden [JM14]:

Startsymbol:	*NP*
Regel 2:	*Det Nominal*
Regel 3:	*Det Noun*
Regel 4:	a *Noun*
Regel 6:	a flight

Grammatiken, die jede Aussage einer Sprache ableiten können, sind jedoch deutlich komplexer.

In der Computerlinguistik ist die syntaktische Zerteilung (engl.: syntactic parsing), oder auch nur Zerteilung (engl.: parsing) der Prozess, der einen gegebenen Satz seine syntaktische Struktur zuweist. Dafür existieren verschiedene Algorithmen, zum Beispiel Bottom-Up, Top-Down oder CKY.

2.2. Semantik

Die Semantik befasst sich mit dem Inhalt und der Bedeutung von Sätzen. Syntaktisch wohlgeformte Sätze müssen nicht zwangsläufig semantisch sinnvoll sein. Beispielsweise erlaubt die Syntax der englischen Sprache einen Satz, der aus einer Nominalphrase gefolgt von einer Verbalphrase besteht. Nach dieser Regel lassen sich alle beliebigen Kombinationen, wie auch der Satz „I sing a letter“. Er ist zwar mit dieser Konstruktionsregel konform, semantisch jedoch ergibt er keinen Sinn. Diese Erkenntnis lässt sich jedoch nicht ausschließlich auf Grund der syntaktischen Regeln erlangen.

Die semantische Analyse von Sätzen befasst sich mit der Aufgabe, diesen eine eindeutige Repräsentation ihrer Bedeutung zuzuweisen. Dies erlaubt eine leichtere Analyse deren Inhalt, losgelöst von den zugrunde liegenden Sprachmerkmalen. Ein weiterer Vorteil einer solchen Darstellung ist, dass verschiedene Formulierungen, die inhaltlich gleich sind, auch auf dieselbe semantische Repräsentation abgebildet werden. Jurafsky et al. verwenden folgendes Beispiel: [JM14]

> Does Maharani have vegetarian dishes?
> Do they have vegetarian food at Maharani?
> Are vegetarian dishes served at Maharani?
> Does Maharani serve vegetarian fare?

Für jede dieser Fragen ist die Antwort dieselbe, da der semantische Inhalt der Frage, trotz der verschiedenen Formulierungen der Gleiche ist. Eine semantische Abbildung der Frage könnte zum Beispiel wie folgt aussehen:
`Serves(Maharani, VegtarianFood)`

2.3. Parse

Mittels des PARSE (Programming ARchitecture for Spoken Explanations)-Projekts [Wei] soll eine Schnittstelle zwischen Computersystemen und Menschen erstellt werden, welche natürlichsprachliche Anweisungen in - vom Zielsystem verständlichen - Quelltext übersetzt. Die natürlichsprachlichen Aussagen werden dazu in mehreren Schritten verarbeitet. PARSE setzt dabei auf eine Agenten-basierte Verarbeitung der Aussagen. Die Aussagen werden zuerst von einem Vorverarbeitungs-Fließband (engl.: Pipeline) verarbeitet, das die gesprochenen Aussagen transkribiert und über einen Zerteiler Beschriftungen wie Wortart oder Grundformen der Wörter hinzufügt. Die interne Repräsentation der Aussage erfolgt in Graphenform: Wörter in der Aussage werden als Knoten dargestellt, gerichtete Kanten stellen Beziehungen zwischen diesen dar. Für die Umsetzung der Teilschritte, die den Graphen in Quellcode umwandeln, kommt die Agentenstruktur zum Einsatz. Dabei erhält jeder Agent eine Kopie des Graphen auf welcher er mit regelbasierten oder statistischen Verfahren, sowie anderer Wissensquellen arbeitet um diesen um neue Informationen zu erweitern. Auch eine wiederholte Ausführung der Agenten ist möglich. Sollte ein Agent mit erst kürzlich in den Graphen eingefügtes Wissen in der Lage sein, seine Ergebnisse noch weiter zu präzisieren, oder zu erweitern, so wird er erneut ausgeführt. Einige für diese Arbeit wichtigen Agenten werden in den nachfolgenden Kapiteln genauer vorgestellt.

2.3.1. Vorverarbeitungsfließband

Für die Vorverarbeitung der gesprochenen Aussagen wurden in [Koc15] verschiedene Sprachverarbeitungswerkzeuge untersucht, um eine geeignete Verarbeitung für die Äußerungen

zu finden. Für die Wortartenerkennung sowie die Erkennung phrasaler Annotationen wurde eine Kombination der Werkzeuge SENNA[1] und der Stanford POS-Tagger[2] verwendet. Außerdem wird eine Heuristik angewandt, um Befehlsgrenzen in der Aussage zu finden. Bei diesem Prozess wird Wörtern desselben Befehls dieselbe Befehlsnummer zugewiesen. Das erlaubt eine getrennte Betrachtung von verschiedenen Teilaussagen in einem Satz.

In der Graphenrepräsentation der Aussage wird jedem Wort das Attribut „pos“ für die Wortart, „chunk“ für die phrasale Markierung, „position“ die Position des Wortes in der Aussage und „instructionNumber“ für die Befehlsnummer der Aussage hinzugefügt.

Für die Eingabe „Go to the fridge.“ sieht die Ausgabe der Vorverarbeitungspipeline wie folgt aus:

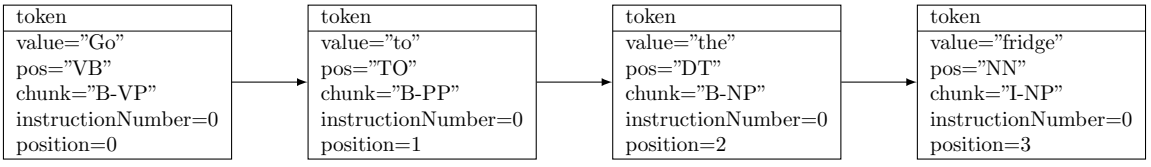

Abbildung 2.1.: Ausgabe der Vorverarbeitung

2.3.2. Aktionserkennung

Bei der Aktionserkennung [Ou16] wird ein Agent für das PARSE-Projekt implementiert, der in einem Eingabetext für erkannte Handlungen Attribute für die semantischen Rollen der Handlung einfügt. Dabei werden durch den Agenten das Attribute „Role“ für jedes Wort in den Graphen eingefügt, welches die semantische Rolle des Wortes beinhaltet. In der nachfolgenden Tabelle sind die möglichen Rollen beschrieben.

Rollenname	semantische Rolle
predicate	Markiert das Prädikat
what	Das Objekt der Handlung
who	Das Subjekt der Handlung
when	Eine zeitliche Beschreibung
where	Eine räumliche Beschreibung
why	kausaler Zusammenhang
null	Falls keine andere Rolle gefunden werden kann

Tabelle 2.2.: Semantische Rollenbeschriftungen und deren Erklärung

Nach der Ausführung des Agenten, sowie des Agenten der Vorverarbeitung sieht der Graph wie folgt aus:

[1] `https://ronan.collobert.com/senna/`
[2] `https://nlp.stanford.edu/software/tagger.shtml`

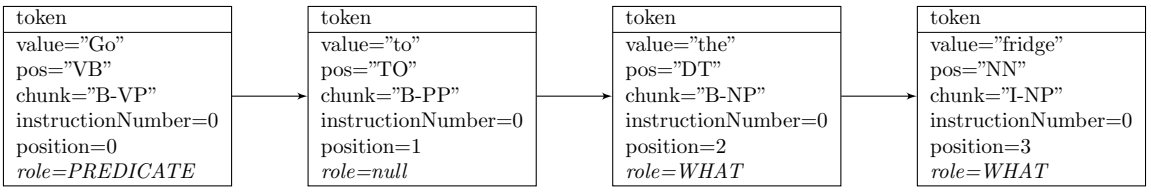

Abbildung 2.2.: Ergebnisgraph der Aktionserkennung

2.3.3. Kontext- und Korreferenzanalyse

In [Hey16] wurde ein Agent für PARSE erstellt, der aus gesprochenen Aussagen Kontextwissen extrahiert und Korreferenzen auflöst. Dieses Kontextwissen enthält Beziehungen zwischen Entitäten, wie die örtliche Beziehungen zwischen ihnen. Außerdem werden darin Entitäten Zustände zugewiesen. Zusätzlich wird das Modell der Situation mit Wissen aus einer Ontologie angereichert, das die Situation beschreibt. Lexikalische Relationen wie Synonyme und Antonyme werden für die gefundenen Entitäten aus WordNet eingefügt. Bei der Auflösung der Referenzen werden zwischen den Teilen der Aussage Kanten eingefügt, welche diese Beziehung repräsentiert.

Im folgenden ist ein Beispiel, das die erstellten Beschriftungen und Realtionen darstellt.

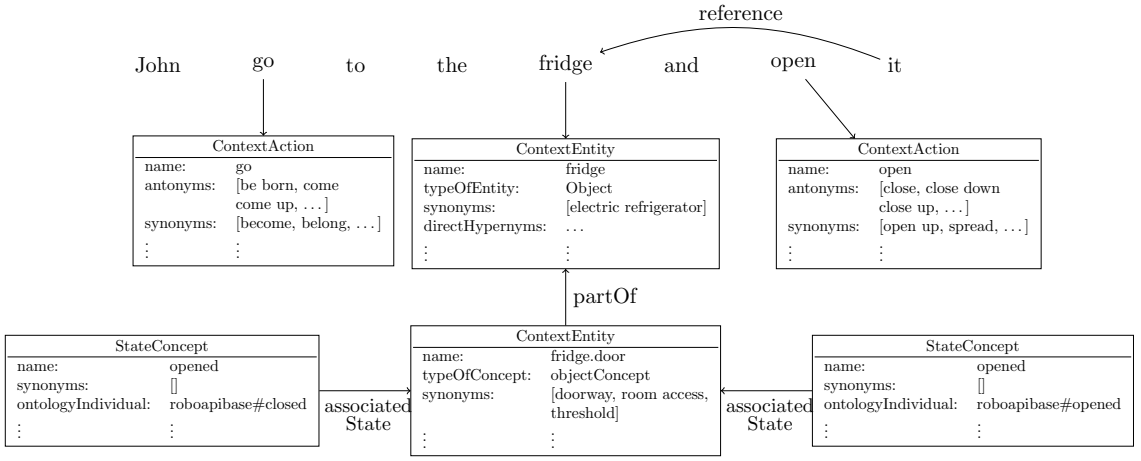

Abbildung 2.3.: Ausschnitt aus Ergebnisgraphen des Kontextanalysierers

2.4. Ontologie

Der Begriff Ontologie stammt aus dem Griechischen und bedeutet „Lehre des Seins“. In der Computerlinguistik ist eine Ontologie eine Sammlung von Wissen. Anders als Lexika, die zu Wörtern deren Bedeutung und eine geringe Anzahl an Relationen wie zum Beispiel Synonyme speichert, sammelt eine Ontologie vermehrt Relationen zwischen Konzepten. Das in Ontologien gespeicherte Wissen beinhaltet auch Wissen, das bei Menschen implizit verfügbar ist, oder sich ohne großen Aufwand aus dem Kontext der Situation inferieren lässt. Durch diese explizite Darstellung von für Menschen implizites Wissen, ist es für ein Computersystem möglich, auch auf dieses zuzugreifen und Aufgaben mit einer breiteren Wissensmenge zu bearbeiten. Da es unmöglich ist, jegliches Wissen aus allen Bereichen explizit zu speichern, behandeln Ontologien meist einen abgegrenzten Geltungsbereich, beziehungsweise eine spezifische Domäne. Des Weiteren unterscheiden sich Ontologien im

Detailgrad: Sogenannte „Upper Ontologies“ befassen sich mit der Sammlung von Wissen, welches in einer Vielzahl von Domänen gilt. Beispiele für diese sind WordNet[3], gist[4] und Cyc. Andere Ontologien behandeln spezifischere, meist kleinere Domänen, enthalten dann aber einen höheren Detailgrad.

2.5. Cyc

Cyc ist eine Ontologie, die seit 1984 entwickelt wird. Bis 1995 sind bereits einhundert Personenjahre an Aufwand in die Erstellung des umfangreichen Korpus geflossen [Len95]. Der Grundgedanke von Cyc ist es, die Welt so abzubilden, wie sie von einem erwachsenen Menschen wahr genommen wird. Dabei sollen auch die trivialsten Annahmen mit aufgenommen werden, die im Allgemeinen sogar als banal und nicht nennenswert angesehen werden. Dazu gehören Annahmen wie:

- You have to be awake to eat.
- You can usually see peoples noses, but not their hearts.
- If you cut a lump of peanut butter in half, each half is a lump of peanut butter; but if you cut a table in half, neither half is a table.

([Len95])

Cyc strukturiert seine Einträge in sogenannte *Mikrotheorien*, die einzelne Wissensgebiete abgrenzen. Sie kategorisieren dabei Wissen in verschiedene Domänen, die den Detailgrad des Wissens, seine thematische Grenzen, oder dessen Geltungszeitraum einschränken.

Die Notwendigkeit von verschiedenen Mikrotheorien folgt direkt aus der Annahme, dass Wissen nicht universell gilt, sondern von seinem Kontext abhängig ist. So ist die Aussage „You can usually see peoples noses, but not their hearts“ während einer Herzoperation faktisch falsch und muss der korrekten Mikrotheorie zugeordnet werden. Durch diese Abgrenzung kann sonst in Konflikt stehendes Wissen trotzdem gespeichert werden.

Cyc verfügt aktuell über tausende Mikrotheorien, über 500.000 Terme, darunter 17.000 Relationstypen und insgesamt 7.000.000 Wissensannahmen, zwischen diesen Konzepten [5].

Mittels Cycs *Inferenzmaschine* (engl.: inference engine) ist es möglich, Wissensannahmen zu prüfen und neues Wissen aus vorhandenen Einträgen zu generieren, oder Fragen an die Ontologie zu stellen.

Für die Repräsentation der Einträge und der Anfragen wird die Sprache *CycL* verwendet. Eine ausführliche Dokumentation der Sprache findet sich auf Cycorps Internetseite[6]. Die für diese Arbeit wichtigsten Bestandteile der Sprache werden nun im Weiteren erklärt. Diese Erläuterungen basieren auf den Beispielen und Erklärungen der offiziellen Dokumentation der Sprache.

2.5.1. Konstanten

Konstanten in Cyc beschreiben Dinge oder Konzepte in der Welt. In CycL erkennt man sie an einem vorangestelltem `#$`. Außerdem beginnen Konstanten mit einem Großbuchstaben.

Eine Kategorie von Konstanten sind Sammlungen von Konzepten oder Individuen, wie zum Beispiel `#$AnimalWalkingProcess` als Sammlung aller Prozesse, bei denen ein Tier läuft, oder `#$Author`.

[3] `https://wordnet.princeton.edu/`
[4] `https://semanticarts.com/gist/`
[5] `http://www.cyc.com/kb/`
[6] `http://www.cyc.com/cycl/syntax/`

Diese Sammlungen beinhalten wiederum Konstanten, welche einzelne Entitäten oder Konzepte beschreiben. Über das Prädikat`#$isa` werden diese in Relation gebracht. Der in Cyc gespeicherte Wissenseintrag `(isa Pythagoras Author)` besagt, dass `#$Pythagoras` Teil der Sammlung `#$Author` ist.

Eine weitere Möglichkeiten in Cyc Konstanten zu definieren, ist über Funktionen: Funktionen bieten die Möglichkeit neue Konzepte aus anderen zu generieren, ohne die benötigten Ergebnisse der Funktion explizit in der Ontologie zur Verfügung stellen zu müssen. Funktionen können an jeder Stelle verwendet werden, an der sonst eine Konstante erwartet wird. So kann an Stelle der Konstante `#$Apple` die Funktion `(#$FruitFn #$AppleTree` verwendet werden. Diese stellt inhaltlich gesehen, dasselbe Konzept dar. Funktionen beschränken Argumenttypen und Ergebnistypen auf bestimmte Sammlungen, die mit dem Prädikat `#$argIsa` (siehe Abschnitt 2.5.2) und `#$resultIsa` abgefragt werden können. Dadurch lassen sich ganze Sammlungen von ähnlichen Konzepten über eine Funktion darstellen. Statt für jeden Baum eine Frucht als eigene Konstante zu definieren, wird eine Funktion definiert, welche angibt, die Baumarten Früchte tragen.

2.5.2. Prädikate

Prädikate sind in Cyc jene Konzepte, die Wissen aus der Ontologie in Verbindung bringen. Einige Beispiele für Prädikate sind `#$isa` für die zweistellige Relation, die Dingen Sammlungen zuweist, oder `#$likesAsAFriend` die Relation die eine freundschaftliche Beziehung zwischen zwei Konstanten definiert. Prädikate können dabei verschiedene *Stelligkeiten* (engl. arity) haben, welche wiederum in dem zwei-stelligen Prädikat `#$arity` definiert ist.

Ein weiteres wichtiges Prädikat ist `#$genls`, das eine Oberklassenrelation beschreibt, und verwendet wird, um den Wissensgraphen hierarchisch aufzubauen. Ein Beispiel für `#$genls` ist der Eintrag `(genls Author Person)`, der definiert, dass jeder Autor auch eine Person ist.

Mit dem Prädikat `#$argIsa` werden Prädikaten (Argument 1) deren Argumenttypen (Argument 3) an der Position (Argument 2) zugewiesen.
Beispiel für das Prädikat `#$isa`:

```
(argIsa isa 1 Thing)
(argIsa isa 2 Collection)
```

Diese beiden Einträge besagen, dass das erste Argument der `#$isa`-Relation ein Ding und das zweite Argument eine Sammlung sein muss.

2.5.3. Formeln

Formeln sind Zusammenschlüsse aus verschiedenen Konstanten um Wissen zu Begriffen zu repräsentieren. Sie sind in Klammerform und beginnen mit einem Prädikat und einer Menge an Argumenten. Ein Eintrag der die Farbe von Gras als grün definiert, sieht wie folgt aus:

```
(#$typicalColorOfThing #$Grass-Plant #$Green-Color)
```

Mit den Prädikaten `#$not`, `#$and`, `#$or` und `#$implies` können mehrere Formeln zusammengefasst werden, um Bedingungen oder Folgen zu beschreiben. Diese Prädikate sind wie ihre Äquivalente in der formalen Logik definiert. Ein Beispiel für die Verwendung des `#$and` Prädikats ist:

```
(#$and
 (#$colorOfObject #$FredsBike #$RedColor)
 (#$objectFoundInLocation #$FredsBike #$FredsGarage))
```

Die Formel wertet genau dann zu wahr aus, wenn die beiden inneren Formeln wahr sind, also Freds Fahrrad rot ist *und* sich in der Garage von Fred befindet.

2.5.4. Variablen

Variablen können in CycL verwendet werden, um noch unidentifizierte Konzepte zu repräsentieren. Sie können an Stelle einer Konstante verwendet werden. Variablen beginnen mit einem oder zwei Fragezeichen und bestehen aus Großbuchstaben. Zwei Fragezeichen deuten auf eine sogenannte „don't-care“ Variable hin, deren genaue Belegung für die Anfrage nicht relevant ist.

2.5.5. Inferenzmaschine

Cyc verfügt über eine Inferenzmaschine (engl.: inference engine), diein der Lage ist Formeln zu beweisen, oder Variablen in diesen zu quantifizieren. Jede Anfrage wird im Kontext einer Mikrotheorie gestellt, in der sie evaluiert wird. Diese gibt vor, welches Wissen für die Annahme gelten soll. Dadurch können die selben Anfragen in verschiedenen Domänen verschiedene Antworten ergeben. Die Inferenzmaschine kann unterschiedliche Formen von Anfragen verarbeiten.

Bei Fragen an die Inferenzmaschine wird eine Formel übergeben, in der unquantifizierte Variablen vorkommen können, die dann von der Inferenzmaschine quantifiziert werden. Die folgende Formel stellt die Frage, aus welchen geopolitischen Untereinheiten Kanada besteht.

```
(geopoliticalSubdivision Canada ?WHAT)
```

Die Inferenzmaschine beantwortet diese Anfrage indem die Variable `?WHAT` mit verschiedenen möglichen Konstanten wie `#$CityOfVancouver` oder `(#$CityNamedFn "Waterloo" Ontario-CanadianProvince)` quantifiziert wird. Dabei kann die Variable auch mit einer Funktion gefüllt werden. In diesem Fall beschreibt die Funktion die kanadische Stadt Waterloo, die in der Provinz Ontario in Kanada liegt.

Anfragen an die Inferenzmaschine können auch nur mit einem Wahrheitswert beantwortet werden. Bei diesen Anfragen wird eine Quantifizierung und eine Menge daraus abgeleiteter Fakten gestellt. Diese werden dann im Kontext der gegebenen Mikrotheorie ausgewertet. Die genaue Belegung der Variablen ist für die Beantwortung nicht relevant, kann aber auf Anfrage als Beleg für die Antwort angefragt werden. So könnte das obige Beispiel mit den Städten in Kanada auch wie folgt als Frage formuliert werden:

```
(forAll ?WHAT
 (implies
  (isa ?WHAT CanadianProvince)
  (geopoliticalSubdivision Canada ?WHAT)))
```

Dabei wird geprüft, ob für jede Belegung der Variable „?WHAT“ die Implikation gilt. Die Inferenz ergibt, dass die Aussage wahr ist.

Für jede Inferenz wird von Cyc eine Begründung bereitgestellt, die die Herleitung der Antwort erläutert. In vielen Fällen können Ergebnisse auf verschiedenen Wegen bewiesen werden.

3. Verwandte Arbeiten

Im folgenden Kapitel werden verschiedene Arbeiten aus dem Bereich der Sprachverarbeitung vorgestellt. Sie beziehen sich auf Probleme aus verschiedenen Themengebieten der Sprachverarbeitung, wie Sprachverständnis oder Wissensextraktion. Dabei sind viele der Problemstellungen und Vorgehensweisen grundlegend wichtig für die in dieser Arbeit thematisierte Problematik.

3.1. RESI - A Natural Language Specification Improver

RESI(Requirements Engineering Specification Improver) [KB09] ist ein Werkzeug, das Softwarearchitekten bei der Anforderungserhebung unterstützen soll. Die Anforderungserhebung ist ein Teilprozess der Softwareentwicklung, der zu großen Teilen noch manuell erfolgt. Diese manuelle Arbeit ist teuer und fehleranfällig. Teilschritte der Anforderungserhebung sollen durch RESI automatisiert, beziehungsweise unterstützt werden [KL10]. RESI unterstützt den Nutzer bei der halbautomatischen Verbesserung von Anforderungstexten. Dafür wird eine Menge an möglichen Problemen in den Anforderungstexten gesucht und dann, wenn möglich, gelöst. Zu den von RESI behandelten Problemen gehören das Auflösen von Nominalisierungen, das Vervollständigen von Prozesswörtern, das Überprüfen von Artikeln und Quantoren, sowie die Auflösung von Mehrdeutigkeiten.

Nominalisierungen stellen in Anforderungen Probleme dar, weil ein komplexer Prozess mit einem Wort beschrieben wird. Möglicherweise fehlen wichtige Informationen. RESI findet Nominalisierungen, indem es für jedes Substantiv der Spezifikation mit einer Ontologieanfrage - an die bereits im Grundlagenkapitel vorgestellte Ontologie Cyc - ermittelt, ob es sich bei dem Wort um eine Nominalisierung handelt. In der Ontologie ist für Nominalisierungen auch das Verb gespeichert, das dem Nutzer als Änderungsvorschlag übergeben wird.

Mit der Auflösung von Nominalisierungen ist die Prüfung von Prozesswörtern möglich. Ein Prozesswort ist meist ein Verb oder Prädikat [KB09], das die Handlung in der Spezifikation beschreibt. Um diese zu prüfen, muss jedoch zuerst ein passendes Konzept in der Cyc-Ontologie gesucht werden, welches den Prozess beschreibt. Kandidaten werden über eine Ontologieanfrage generiert. Mögliche Wortbedeutungen werden dem Nutzer in einem Dialogfeld angezeigt, der dann die richtige Bedeutung auswählen kann. Eine automatische Disambiguierung erfolgt nicht.

Cyc besitzt ein Prädikat, das für ein Prozesswort eine Liste an benötigten Argumenten speichert. Dieses Prädikat wird mit dem vom Nutzer gewählten Sinn der Handlung angefragt. Außerdem besitzt jedes Konzept in Cyc eine Menge von Oberklassen. Diese werden in dem zweistelligen Prädikat `#$isa` kodiert. Ebenso sind für Prädikate die Argumenttypen über das Prädikat `#$argIsa ?PRED ?NUM ?IS` abfragbar. Mit diesem wird für die Rollen, welche in Cyc Prädikate sind, angefragt, welche Klassen ihre Argumente haben. Diese werden verwendet, um bereits im Satz vorhandene Konzepte als Kandidaten für die Rollen zu prüfen. Dazu wird über das `#$isa` Prädikat eine Verbindung der Konzepte gesucht.

Ein weiteres Problem, das zu Unklarheiten führen kann, sind Wörter derselben Bedeutung. Wenn an mehreren Stellen derselbe Prozess mit verschiedenen Wörtern beschrieben wird, die aber dasselbe Konzept beschreiben, ist eventuell nicht gleich ersichtlich, dass es sich um dasselbe Konzept handelt. Diese Problem wird bei RESI gelöst, indem Wortähnlichkeiten mit Hilfe einer Ontologie geprüft werden. Es werden Substantive aus derselben Spezifikation paarweise auf Ähnlichkeit geprüft. Der Nutzer kann im Anschluss auswählen, ob er ähnliche Wörter durch das gleiche Wort ersetzen möchte.

Der durch RESI verfolgte Ansatz wurde in einer Fallstudie getestet. Dazu wurden zwei verschiedene beispielhafte Anforderungstexte bereit gestellt. Diese sollten, durch RESI unterstützt, auf mögliche Probleme untersucht werden. Zum Vergleich wurden die Texte ohne das Werkzeug durchsucht. Die Bearbeitungsgruppen waren dabei in Informatik-Doktoranden, Informatiker und Nicht-Informatiker aufgeteilt.

Ein erstaunliches Phänomen war der Fund von möglichen Ambiguitätsproblemen, welche den Analysten aufgrund ihres Domänenwissens nicht aufgefallen wären. RESI hat diese oftmals erkannt, während die Analysten diese nicht gefunden hatten oder diese nicht als Unklarheit bezeichnet hätten.

Abseits von diesem Fund hat RESI insbesondere bei der Nicht-Informatiker Gruppe eine besonders starke Erhöhung der Ausbeute erzielt. Die Steigerung der Mängel ist abhängig vom Spezifikationstext: durchschnittlich 31,34%, beziehungsweise 88%. Mit einer ausreichend großen Bearbeitungszeit und unter Verwendung von RESI ist eine Ausbeute von 1 möglich. Die Autoren der Arbeit folgern, dass dieses Werkzeug für den praktischen Einsatz geeignet scheint.

3.2. SensiCal

SensiCal [Mue00] ist ein Kalender, der den Nutzer bei der Erstellung von Kalendereinträgen unterstützt. Kalendereinträge werden beim Erstellen auf Vollständigkeit und eventuelle Konflikte geprüft. Für diese Prozesse verwendet SensiCal die Wissensdatenbank ThoughtTreasure[1]. Diese verfügt über 25.000 hierarchisch organisierte Konzepte, 50.000 Wissensannahmen und etwa 100 Skripte, die eine typische Aktivität, deren Ziele und Voraussetzungen beschreiben.

Um ThoughtTreasure nutzen zu können, werden die Kalendereinträge in einem einheitlichen Datenformat angelegt. Dieses beinhaltet unter anderem das Datum und die Uhrzeit, sowie die Dauer, den Ort, die Teilnehmer, sowie den Typ des Termins. Diese Informationen werden aus den vom Nutzer eingegebenen Informationen extrahiert. So wird aus dem Kalendereintrag „lunch w/ lin at frank's steakhouse“ die Terminart „meal“, die Art des Essens als „lunch“, die Teilnehmer „Lin“, der Ort „frank's steakhouse“ und das dort servierte Essen als „steak“ extrahiert. Diese Extraktion wird von verschiedenen Agenten durchgeführt, die in dem Text agieren und mittels eines Lexikons und Regeln bestimmte

[1] `http://alumni.media.mit.edu/~mueller/papers/tt.html`

Informationen extrahieren. Der „restaurant text agent“ kann zum Beispiel über das Muster „X's Steak House | X Steakhouse | X Steak | X House | X Grill | X Grill → steakhouse“ in dem obigen Beispiel den richtigen Restauranttypen extrahieren.

Im zweiten Schritt werden fehlende Informationen automatisch vervollständigt. Beispielsweise wird für einen Essenstermin ohne Enduhrzeit die Dauer des Termins automatisch vervollständigt, indem die durchschnittliche Dauer eines solchen Termins in ThoughtTreasure angefragt wird.

Schlussendlich kann das Werkzeug nun Probleme in den Kalendereinträgen finden. So wird der Nutzer beim Erstellen des Kalendereintrages „breakfast w/ susan at 3am“ mit dem Hinweis „You are eating breakfast at 3am“ auf ein potentielles Problem hingewiesen. Dafür werden verschiedene Regeln angewendet, die teilweise aus ThoughtTreasure stammen und teilweise im Programmcode fest kodiert sind. Diese Regeln prüfen in verschiedenen Kategorien, wie Ort und Zeit der Veranstaltung und welche Annahmen jeweils gelten. Unter diesen Annahmen sind simple Allgemeinwissens-Fakten, wie „You cant't visit places that are not open“.

Um diese Annahmen zu prüfen, sind weitere Informationen über die Teilnehmer erforderlich. Dafür können in den Kontaktdaten erweiterte Einträge für Essenspräferenzen, oder die Beziehung zu der Person eingetragen werden. Im Beispiel „lunch w/ lin at frank's steakhouse“ ist in Lins Kontaktdaten hinterlegt, dass sie Vegetarierin ist. SensiCal folgert dann aus dem Fakt, dass „Frank's steakhouse“ ein Steakhouse ist und in diesen Fleisch serviert wird, sowie dem Fakt, dass Fleisch nicht vegetarisch ist, dass ein Problem vorliegen könnte und stellt eine Rückfrage an den Nutzer: „You are taking Lin who is vegetarian to a steakhouse?“

Die Autoren geben im Ausblick noch den Anstoß als weiteren Entwicklungsschritt die Wissensmenge von ThoughtTreasure weiter auszubauen, um SensiCal noch mehr für die Anwendung relevante Informationen zu liefern. Eine Evaluation erfolgte nicht, d.h. es bleibt unklar, wie gut die Applikation den Nutzer tatsächlich beim Erstellen von Kalendereinträgen unterstützt, oder wie gut die generierten Vervollständigungen sind. Eine Evaluation wurde auch von den Autoren vorgeschlagen, um häufige Fehler zu finden und aus diesen zu lernen und die Allgemeinwissens-Regeln um Sonderfälle zu erweitern.

3.3. On the Application of the Cyc Ontology to WordSense Disambiguation

In [CCB06] beschreiben Curtis et al. einen Ansatz zur Disambiguierung, der im Gegensatz zu lexikonbasierten Ansätzen, auf semantischen Informationen aufbaut. Dabei werden mit dem `Recursive Template Parser` und `Cyclifier` Sätze in natürlicher Sprache auf CycL-Ausdrücke übertragen. Diese CycL-Ausdrücke repräsentieren jeweils eine andere Bedeutung des Satzes. Abhängig von der Bedeutung, die gewählt wurde, werden dann unterschiedliche Annahmen der Ausdrücke gewählt. Diese Hypothesen werden in eigenen Mikrotheorien abgelegt, in der dann die Cyc-Inferenzmaschine Wissen prüft, und Annahmen generieren kann. Im folgenden werden diese verschiedenen, sich in der Bedeutung unterscheidenden Hypothesen gegeneinander abgewägt. Für den Satz „Gates founded Microsoft“ wurden die Sätze

```
(#$foundingAgent #$MicrosoftInc #$BillGates)
```

und

```
(#$thereExists ?X
 (#$and
```

```
  (#$isa ?X #$Gate)
  (#$foundingAgent #$MicrosoftInc ?X)
 )
)
```

aufgestellt. Die erste Interpretation beinhaltet genau den Inhalt des Satzes, während die zweite Interpretation sich auf ein `Tor` bezieht, das zur selben Zeit auch `MicrosoftInc` gegründet hat. Dabei können bereits Kandidaten verworfen werden, indem die Sätze mit der Inferenzmaschine überprüft werden. Für das obige Beispiel kann der zweite Fall verworfen werden, da das Prädikat `#$foundingAgent` nicht mit den Konstanten `#$MicrosoftInc` und einer Konstante aus der Sammlung `#$Gate` befüllt werden kann (`#$foundingAgent` gibt Beschränkungen der Argumenttypen vor). Diese Prüfung reduziert die Anzahl der möglichen Kandidaten auf circa drei pro Wort.

Im Weiteren wird auf die Wissenseinträge von Cyc zurückgegriffen, um die verschiedenen verbleibenden Theorien zu bewerten. Dieser Prozess findet in zwei Teilschritten statt: Dem Bestimmen der semantischen Nähe (engl. „semantic closeness“) und dem semantischen Beitrag (engl. „semantic contribution“).

Die semantische Nähe ist durch Beziehungen zwischen Konzepten definiert. Durch Prädikate sind Konzepte in Cyc verbunden. Existiert ein Pfad über ein oder mehrere Prädikate, der von einem Konzept zum anderen führt, wird eine semantische Nähe der Konzepte angenommen. So sind `#$FireTruck` und `#$RedColor` durch das Prädikat `#$typicalColorOf` verbunden und eine semantische Nähe der beiden Konzepte in der beinhaltenden Mikrotheorie wird angenommen. Zur Bestimmung der semantischen Nähe werden die Prädikate `(#$nearestIsa OBJ COL)` und `(#$nearestIsaOfType OBJ TYPE COL)`[2] verwendet. Diese können benutzt werden, wenn die semantische Distanz zwischen ganzen Sammlungen von Konzepten nicht bekannt ist. Außerdem wird das Prädikat `(#$conceptuallyRelated THING1 THING2)` verwendet, das bedeutet, dass es einen Pfad von `THING1` nach `THING2` durch die Ontologie (bzw. die ausgewählte Mikrotheorie) gibt. Diese Pfade sind oft nicht explizit deklariert, sondern müssen von der Inferezmaschine gefunden werden. Als Beispiel dient der Eintrag `(#$colorOfType #$FireTruck #$RedColor)`, der aussagt, dass Feuerwehrautos (zumindest in der ausgewählten Mikrotheorie) rot sind. Da `#$colorOfType` zu `#$conceptuallyRelated` generalisiert, gilt die Aussage auch für dieses übergeordnete Prädikat. Mit diesen Prädikaten wird nun eine Bewertung der Wortbedeutungen durchgeführt. Sie werden auf eindeutige Wörter aus dem Text im Satz-, Paragraph-, und Dokumentkontext angewendet. Semantische Nähe im Satzkontext wird besser bewertet, der Dokumentkontext wird am geringsten bewertet. Hierdurch erhält der unmittelbare Satzkontext einen deutlichen stärkeren Einfluss auf die Bedeutung des mehrdeutigen Wortes als Wörter, die an einer weit entfernten Stelle im Text stehen.

Zur Bestimmung des semantischen Beitrags werden für alle generierten CycL-Sätze die möglichen Bedeutungen von mehrdeutigen Wörtern geprüft. Trägt eine Bedeutung zum Wahrheitsgehalt eines Satzes bei, so wird die Bewertung dieser Bedeutung erhöht. Cyc unterscheidet zum Beispiel zwischen BookCopy, einer physischen Kopie eines Buchs und `AuthoredWork`, also dem abstrakten Konzept. In dem Ausdruck „worn book“ ist das Wort „book“ nicht sofort eindeutig. Da aber nur Kopien von Büchern Abnutzung erfahren können und dieser Fakt in der Ontologie gespeichert ist, hat diese Bedeutung einen stärkeren semantischen Beitrag.

Nach Verarbeitung der beiden Teilschritte wird für jedes Wort die Bedeutung mit der höchsten Bewertung ausgewählt.

[2] `(#$nearestIsa OBJ COL)` beschreibt, dass COL die „nächstliegendste“ Sammlung von Begriffen ist, in welcher sich auch OBJ befindet. `#$nearestIsaOfType` erweitert das `#$nearestIsa` um TYPE, welcher noch die Regel `(#$isa COL TYPE)` vorschreibt, also den Typ der Sammlung beschränkt.

Das Verfahren wurde mit fünf verschiedenen, händisch annotierten Wikipedia-Artikeln evaluiert. Im Vergleich zur zufälligen Wahl der Bedeutungen (33.49%) konnte die Disambiguierung in 55.61% der Fälle korrekt durchgeführt werden.

3.4. Modeling Human-Agent Interaction with Active Ontologies

Guzzoni et al. beschreiben in ihrer Publikation `Active`, ein Rahmenarchitektur (engl. `framework`), welches die Grundlage für intelligente Assistenten bildet. Active besteht aus dem `Active Editor`, welcher Modellierung, Einsatz und Testen von Anwendungen ermöglicht, dem `Active Server`, welche die Ausführung der Anwendung durchführt sowie der `Active Console` mit welcher eine laufende Instanz eines Servers beobachtet und überprüft werden kann.

Zur Verwaltung von Wissen wird eine `Active Ontology` verwendet. Diese besteht aus Konzepten, Domänenobjekten, Ereignissen, Aktionen, Prozessen und Regeln, die die Logik der Domäne beschreiben. Regeln bestehen aus einer Bedingung und Aktion.

Grundlegend besteht Active aus einer Regelmaschine und einer Faktenmenge. Diese Regelmaschine arbeite mit Ereignissen und Daten. Wird eine Änderung der Fakten in der Ontologie ausgelöst, werden zugehörige Regeln ausgewertet. Diese können Konsequenzen im Wissen oder folgende Aktionen definieren. Die Funktion einer Anwendung hängt dabei von den ausgewählten Regeln ab. Beim Prüfen der Regeln werden die Bedingungen dieser geprüft, und deren Folgen angewendet. Variablen in der Bedingung der Regel werden dabei mit passenden Fakten instantiiert. Wenn zwei Fakten vereint (engl. `unify`) werden, werden sie als äquivalent angesehen. Für bestätigte Regeln wird dann die an sie gebundene Aktion ausgeführt.

Eine Active Anwendung kann eine Hierarchie von Konzepten definieren. Für eine Raumbuchung könnten zum Beispiel die Konzepte „Datum“, „Person“, „Ort“, etc. als Unterkonzepte an die übergeordnete Aktion, also in diesem Fall die Raumbuchung, angehängt werden.

Active stellt zusätzlich noch zwei Sprachverarbeitungswerkzeuge zur Verfügung, um den Aufbau von Applikationen zu erleichtern. Einen aufsteigenden Zerteiler(engl. `bottom-up parser`), welcher eine Grammatik benutzt um Sätze auf Konzepte in Active zu übertragen [Ear70] und ein Modul, welches Muster in Texten erkennt.

Diese können verwendet werden um Objektknoten in der Ontologie beim Finden von für sie relevanten Informationen in der Eingabe zu unterstützen. Ein Knoten, welcher das Thema einer Sitzung in natürlicher Sprache finden soll, verwendet beispielsweise die Regel „Für zwei Eingabewörter: Wenn das erste Wort ‚about‘ ist, ist das folgende Wort das Thema der Besprechung“. Diese Vorgehensweise ermöglicht eine Stückweise Verarbeitung der Eingabe mit einem Aufwärtsverfahren.

Beziehungen zwischen Knoten können mit verschiedenen Information erweitert werden. Sie können Attribute als optional, verpflichtend, einzigartig oder mehrfach markieren. Die Ontologie speichert den Kontext der aktuellen Situation, wodurch auch partielle Aussagen zum Verständnis der Situation beitragen. In Folgesätzen kann dann durch weitere Informationen die Situation vervollständigt werden. Bei der Regelauswertung wird ein Nutzer beim Nichteinhalten der Regeln darauf hingewiesen, und eine Resolution der Fehler ist möglich.

Mit diesen Bausteinen können einfach intelligente Assistenten erstellt werden. In der Veröffentlichung wird ein Anwendungsfall für das Planen von Treffen. Der Nutzer interagiert dabei mit über Texteingabe mit dem Assistenten. Möchte dieser ein Treffen erstellen, so

wird über die Eingabe eine Änderung der Fakten in Active ausgelöst. Werden wichtige Informationen für den Plan ausgelassen, so informiert Active den Nutzer darüber indem es einen Texthinweis hinterlässt. Das wird erkannt, indem auf die Liste der benötigten Informationen und deren Wichtigkeit (optional, verpflichtend, etc.) ausgelesen wird. Daraufhin wird mittels einem Zerteiler die Eingabe verarbeitet, und die zuständigen Knoten, bzw. derer Aktionen, in der Ontologie werden ausgeführt. Dazu gehören Knoten, die zum Beispiel eine Liste an freien Datumsvorschlägen sucht, oder ein Knoten, der E-Mails an Teilnehmer sendet und deren Bestätigungen abwartet. Je nach Ablauf der Aktion, werden dynamisch verschiedene Knoten für verschiedene Funktionen ausgewählt.

3.5. Methods of Rule Acquisition in the TextLearner System

Curtis et al. stellen in ihrem Paper TextLearner [CBW+09] - einen Prototypen zur Wissensaneignung aus Textquellen - vor. TextLearner setzt dabei auf Cyc's breite Wissensquelle, und seine Inferenzmaschine, um aus diversen Mikrotheorien neue Einträge zu generieren und schon vorhandene Regeln zu erweitern. Bei der Entwicklung des Modells für den Text wurde darauf geachtet, die Modellierung an die Wissensrepräsentation von Cyc anzupassen. In Cyc wird zum Beispiel zwischen abstrakten Konzepten und Instanzen mittels verschiedenen Konstanten unterschieden. Ein Buch kann demnach ein abstraktes Konstrukt ohne Bezug zur Realwelt, oder als eine genaue Instanz, also eine physische Kopie sein, welche die irgendwo in der Welt existiert. Für beide Fälle wird aber in der englischen Sprache dasselbe Wort verwendet, die Bedeutung ist also nicht ausschließlich durch das Wort identifizierbar. In Cyc existiert diese Ambiguität nicht, es wird zwischen konkreten Instanzen (`#$BookCopy`) und dem abstrakteren Konstrukt (`#$Book-CW`) unterschieden. Um sinnvolle Einträge für die Wissensdatenbank zu generieren, sollen alle im Text gefundenen Instanzen auf ihre richtige abstrakten Strukturen zurückgeführt werden. Die grundlegende Schwierigkeit besteht beim Versuch, korrekte Interpretationen für kleine Textstellen zu finden. Die richtige Interpretation lässt sich nur bei der Betrachtung des Kontextes finden. Aus diesem Grund wird nach der Bearbeitung eines Satzes weiter in einem Modell *gehalten*, um sich bei der Analyse von neuen Sätzen auf bereits gelerntes Kontextwissen zu stützen.

Zur Zeit verfügt TextLearner über verschiedene Regeln, die unterschiedliche Merkmale in Textkonstrukten auf CycL Terme abbilden. Zum Beispiel werden für zusammengesetzte Substantive, also mindestens zwei aufeinanderfolgende als Substantiv markierte Wörter, das Prädikat `#$candidateNounCompound` angewendet. Dieses prüft, ob die Kandidaten syntaktische und semantische Regeln, die schon für andere zusammengesetzte Substantive existieren, einhalten. Gilt das Prädikat, so wird ein neuer Eintrag in einer speziellen Mikrotheorie erstellt, in der das neue Substantiv, sowie die erfüllten Regeln gespeichert sind. Die Einträge in dieser Mikrotheorie werden zu einem späteren Zeitpunkt gesammelt einem Ontologisten übergeben, der die Einträge dann manuell auf ihre Richtigkeit überprüft. Ein Regeleintrag für die Phrase „Bantu word“, welche Cyc zu diesem Zeitpunkt unbekannt ist, sieht wie folgt aus:

```
(NCRuleWithTemplateFn
 (TheList SubcollectionOfWithRelationToFn)
 (TheList TheNCHead genls LexicalWord)
 wordInLanguage
 (TheList TheNCModifier isa NaturalLanguage))
```

Diese Regel bedeutet, dass zusammengesetzte Substantive aus einer natürlichen Sprache („Bantu“) und einem Worttypen („verb, word, name“) bestehen kann. Sie abstrahiert die ursprüngliche Phrase „Bantu Word“ und erlaubt noch weitere Konstruktionen für ähnliche

Sätze. Deren folge ist es, dass Wörter die auf dieses Wort folgen in einer Bantusprache sind.

Andere Regeln von TextLearner generieren Vorlagen für maschinelle Übersetzung von natürlicher Sprache auf CycL Terme. Lücken in diesen Vorlagen haben die Folge, dass eine Übersetzung in einigen Fällen nicht möglich ist. Um diese zu schließen werden Texte betrachtet, welche bei einer Übersetzung nicht alle Informationen beibehalten. Der Satz „The heart pumps the blood to the lungs“ würde mit den Informationen

```
(and
 (isa :ACTION PumpingFluid)
 (providerOfMotiveForce :ACTION :SUBJECT)
 (primaryObjectMoving :ACTION :OBJECT))
```

zwar in der Lage sein die Verb-Subjekt-Objekt Struktur zu erfüllen, kann dabei aber „to the lungs“ nicht zuordnen. Um dies zu ermöglichen wird in unübersetzten, fehlenden Wort `To-TheWord` in Erfahrung gebracht, dass dieses Inhalte der Form `to-UnderspecifiedLocation` beschreiben kann. Die Prädikate, welche unterspezifizierte Parameter beschreiben sind sehr allgemein gehalten und haben viele Unterklassen. Diese Unterprädikate werden dabei absteigend durchsucht, bis das passendste gefunden wurde. Dazu werden Allgemeinwissensannahmen aus `PumpingFluid` und `Pump-TheWord` verwendet. Aus `(genls Pumping-Fluid Translocation)` wird gefolgert, dass jeder Pumpvorgang auch eine Veränderung der Position ist. Aus `(requiredActorSlots Translocation toLocation)` folgt, dass jede Veränderung der Position eine Zielposition als benötigte Handlungsrolle trägt. Diese Informationen werden dann zusammengesetzt um eine neue Übersetzungsvorlage zu generieren:

```
(and
 (isa :ACTION PumpingFluid)
 (providerOfMotiveForce :ACTION :SUBJECT)
 (primaryObjectMoving :ACTION :OBJECT)
 (toLocation :ACTION :OBLIQUE–OBJECT))
```

Damit kann nun der komplexere Eingabesatz vollständig auf CycL übersetzt werden.

Dieses Verfahren wurde jedoch nicht evaluiert. Jegliches Wissen, das durch TextLearner generiert wird, muss noch von einem menschlichen Prüfer betrachtet werden. Es ist also unklar, wie gut die Ansätze dieses Prototypen in allgemeinen Fällen unbekanntes Wissen erschließen können.

3.6. Populating Cyc with Web-Knowledge

Der manuelle Aufbau von Ontologien ist ein zeitintensiver Prozess. Oftmals gibt es Textkorpora, die die benötigten Informationen bereits enthalten. Dabei werden aus diesen Textkorpora Fakten extrahiert, um zum Beispiel eine Ontologie zu befüllen. Ein besonders beliebter, frei zugänglicher Textkorpus ist das gesamte Internet, das gleichzeitig via Suchmaschinen effizient durchsucht werden kann. Frühe Ansätze für Informationsextraktion finden sich bereits in den 90er Jahren (Hearst Pattern: [Hea92]). Diese befassen sich primär mit der Satzstruktur und verwenden vorgegebene Muster (z.B.: „X is a Y“) um Relationen zu finden. In [MWK+05] verfolgen Matuszek et. al einen Ansatz, der im Internet Fakten sucht, um bereits in Cyc bestehende Prädikate um neue Informationen zu erweitern. Dabei werden für bereits bekannte Entitäten und bekannte Prädikate fehlende Argumente gesucht. In [SSM+06] werden mit einem ähnlichen System Fakten zu benannten Entitäten gesucht, um Cyc durch neues Wissen zu diesen zu erweitern. Der zweite Ansatz unterscheidet sich in der Vorgehensweise, da hier auch für noch unbekannte Entitäten neues Wissen hinzugefügt werden kann.

Beide Systeme arbeiten jedoch beim Suchen und Verifizieren der neuen Fakten ähnlich, und stützen sich stark auf bereits in Cyc vorhandenes Wissen.

Bei der Faktensuche für neue Entitäten wird in einem Vorverarbeitungsschritt zuerst der Typ der gefundenen benannten Entität bestimmt. Dafür werden mehrere Textstücke im Internet gesucht, welche die Entität nennen. Die Suchstrings ergeben sich dabei durch Cycs Sprachgenerierungsmodul. Dieses kann für Anfragen an die Inferenzmaschine eine natürlichsprachliche Repräsentation der Fakten generieren. So kann zum Beispiel aus dem unvollständigen Fakt `terroristOrgPoliticalWing LebanistHizballah ?X` der Suchstring „* , a|the political wing of Hezbollah“. Jeder im Internet gefundene Satz wird mit einem Eigennamenerkenner (engl. `named entity recognizer`, im folgenden NER) markiert. Die gefundenen Entitäten werden damit einer möglichst allgemeinen Oberklasse in Cyc zugewiesen. Dies bedeutet, dass diese Klasse in der Kette von Relationen, die von der `#$isa` Relation vorgegeben ist, möglichst nah an deren Ende ist. Bei inkonsistenter Typisierung wird die Entität verworfen und der nächste Kandidat wird überprüft. Die gefundenen Sätze werden mit Zerteilern markiert; Cyc überführt diese markierten Sätze auf eine CycL Repräsentation. Diese Repräsentation kann dann sowohl mit den Fakten aus Cyc auf Konsistenz geprüft werden, als auch wieder als Vorlage für die Generierung von natürlichsprachlichen Hypothesen sein, welche zur Validierung wieder an eine Internet-Suchmaschine übergeben werden.

Die Suche nach neuen Informationen im Internet erfolgt bei beiden Ansätzen mit Suchmustern. Neue Fakten werden immer im Bezug auf relevante Prädikate (anhand der Oberklasse) gesucht. Für jedes Prädikat wurden händisch Konstruktionsregeln erstellt, die den Aufbau der Suchmuster beschreiben. Dem Prädikat (`#$foundingAgent PalestineIslamicJihad ?WHO`) fehlt der Gründer-Eintrag. Aus den vorgegebenen Regeln für das Prädikat `#$foundingAgent` werden die Suchstrings

- PIJ, founded by
- Palestine Islamic Jihad founder

generiert.

Schon in diesem Schritt werden Informationen aus Cyc herangezogen, um alternative Suchanfragen zu generieren. Im obigen Beispiel wird etwa das hinterlegte Akronym anstelle des vollen Namens verwendet.

Mit diesen Mustern werden bei Google übereinstimmende Seiten gesucht. Für jeden Treffer wird der jeweilige Satz zerteilt, um eine passende Cyc-Entität zu finden. Dieser Prozess besteht aus der Suche nach passenden Entitäten in der Ontologie, sowie der Prüfung der Argumenttypen des Prädikats.

Für noch unbekannte Entitäten werden außerdem noch die Sätze der ursprünglichen Typisierung verwendet, um simple Fakten zu finden. Da aber keine einheitlichen Muster für diese Sätze bestehen, werden nur referenzierende, auf die Entität bezogene, Eigenschaften gesucht und als Prädikat übernommen (Bsp.: „`Palestinian` leader Yassir Arafat was laid to rest after memorials in Cairo and Ramallah.“ ergibt `(#$ethnicity #$YassirArafat #$EthnicGroupOfPalestinians)`).

Alle gefundenen Informationen werden verwendet, um verschiedene GAFs (Ground Atomic Formulæ), also einzelne Formeln, zu erstellen. Jede dieser Formeln wird dann an Cyc's Inferenzmaschine übergeben, um geltende Regeln zu den Prädikaten zu überprüfen und um nicht sinnvolles Wissen, sowie redundante Einträge zu verwerfen. Im obigen Beispiel konnten mit Hilfe der Suchergebnisse und des Zerteilens die folgenden GAFs generiert werden:

```
(foundingAgent PalestineIslamicJihad Terrorist-AlShikaki)
(foundingAgent PalestineIslamicJihad AugusteRodin)
```

Der erste Term ist bereits in der Ontologie enthalten und wird aus diesem Grund verworfen. Für den zweiten Satz muss die folgende Regeln gelten:

```
(implies
 (and
  (foundingAgent ?ORG ?AGENT)
  (foundingDate ?ORG ?DATE))
 (temporallySubsumes ?AGENT ?DATE))
```

Diese Regel besagt, dass für jeden Gründer `?AGENT` einer Organisation `?ORG` zu einem Datum `?DATE` gelten muss, dass das Gründungsdatum innerhalb der Lebenszeit des Gründers liegt. Cyc verfügt über das Todesdatum (1917), sowie das Gründungsdatum (1981) und folgert über die Verletzung der Implikation, dass der Gründer-Kandidat faktisch nicht korrekt sein kann.

Als weiterer Prüfungsschritt werden die generierten Terme über ein Cyc-Modul in natürliche Sprache umgewandelt und erneut mittels der Suchmaschine gesucht. Dabei werden nur exakt übereinstimmende Treffer gesucht, um sicherzustellen, dass die gefundenen Wissenseinträge an einer Stelle im Internet auch in dieser Form verwendet werden. Jeder Term, der keine exakte Treffer liefert wird als fehlerhaft markiert.

Alle generierten Fakten werden einer speziellen Mikrotheorie hinzugefügt, die später von einem Menschen überprüft wird. Dieser verwirft dann fälschlich als korrekt markierte Einträge, und markiert jene, die in die Ontologie übernommen werden sollen. Für beide Ansätze wurde eine Evaluation durchgeführt. Dabei wurde geprüft, wie viele generierte Fakten im Vergleich zu händischer Einstufung, korrekt waren.

Der erste Ansatz erreichte für 114 evaluierte Fakten eine Ausbeute von 41% und eine Präzision von 52%.

Fakten über benannte Entitäten konnten für verschiedene Prädikate unterschiedlich gut extrahiert werden. Durchschnittlich wurden dabei 31% der gefundenen Fakten mittels Google validiert. Von diesen 31% wurden 48% von einem menschlichen Annotierer als korrekt eingestuft. Viele fehlerhafte Fakten konnten auf Probleme beim Entitätenerkenner zurückgeführt werden. Dieser markiert oftmals zusammenhängende Entitäten als mehrere unabhängige Entitäten, wodurch gleich eine Vielzahl von inkorrekten Fakten produziert wurde. Außerdem sind durch Fehler in den Suchmustern Ergebnisse für gesamte Prädikate fehlerhaft geworden. Allgemein galt, dass für Prädikate mit vielen Einträgen, also einer großen Menge an verfügbarem Wissen, die Suche bessere Resultate liefert.

4. Analyse

Unvollständige Aussagen kommen insbesondere in natürlicher Sprache häufig vor. Denn die Situation, in der ein Sprecher sich befindet, ist deutlich komplexer, als der in Texten schriftlich genau definierte Kontext. Viele Annahmen sind implizit oft wird Bezug auf sichtbare Objekte genommen, und der Sprecher verlässt sich auf das Wissen über gegenseitige Annahmen. Betrachtet man beispielhaft die Aussage „Get the juice.“ ist ohne weiteres Wissen über den Kontext nicht genau klar, wer überhaupt angesprochen wurde, noch woher der Saft kommen soll. Im unmittelbaren Kontext der Situation und der vorangegangenen Konversation sind Informationen enthalten, die zu diesem Verständnis beitragen. Über solche Informationen kann „fehlendes“ nicht explizit geschildertes Wissen von Menschen inferiert werden. Doch bei natürlichsprachlichen Aussagen ist das Nutzen dieses Kontextes schwieriger. Als Mensch erkennt man, dass die Aussage an ihn gerichtet ist, wenn der Sprecher bei der Aussage in die eigene Richtung blickt. Solche Informationen müssten für Roboter zum Beispiel über Sensorsysteme verfügbar gemacht werden, oder auf andere Art explizit definiert sein.

Ebenso problematisch ist es, wenn weitere wichtige Teilaussagen nicht genannt werden, weil der Sprecher sie als implizit annimmt. In dem Beispiel wurde nicht genannt, an welcher Stelle sich der Saft befindet. Menschen können über ihr Weltwissen mögliche Kandidaten generieren, und diese Hypothesen dann abwägen. Dazu nutzen sie diverse Informationen, wie etwa zuletzt getätigte Aktionen („Ich habe den Orangensaft auf den Tisch gestellt“), visuelle Reize, oder Wissensannahmen („Normalerweise lagert man Orangensaft im Kühlschrank“). Wird eine solche Anfrage an ein naives Sprachverarbeitungssystem gestellt, so hat dieses keine Möglichkeit, auf Informationen dieser Art zurückzugreifen. Fehlen Informationen, wird die Aussage eventuell falsch interpretiert, was dazu führt, dass der Nutzer diesen Fehler durch Intervenieren beheben muss. Ein solches Verhalten von Sprachverarbeitungsystemen ist nicht wünschenswert. Für eine natürliche Interaktion ist es wichtig, dass ein Sprachverarbeitungssystem unvollständige Aussagen ähnlich wie ein Mensch erkennen kann und dann ebenso in der Lage ist, diese wieder zu vervollständigen. Auch sollte es die Möglichkeit besitzen, den Nutzer darauf hinzuweisen, sobald es eine Aussage nicht mehr verstehen kann.

In den folgenden Kapiteln werden mögliche Ansätze vorgestellt um dieses Problem anzugehen. Sie gliedern sich in das Erkennen von unvollständigen Aussagen in Abschnitt 4.1 und dem automatischen Vervollständigen von unvollständigen Aussagen in Abschnitt 4.2.

4.1. Unvollständige Aussagen

Um eine gute Lösung für das Problem von unvollständigen Aussagen zu finden, muss zuerst betrachtet werden, in welchen Fällen dieses auftritt und was die Gründe dafür sind. Ein Grund dafür ist, wenn ein Sprecher tatsächlich Teile der Aussage nicht nennt. Entweder wurde dabei ein Teil schlichtweg vergessen, oder der Sprecher hat die Informationen mit Absicht ausgelassen. Er denkt dabei, dass die Handlung klar genug definiert ist, um eventuelle Unklarheiten auch implizit zu inferieren. Dies führt direkt zu dem Problem, dass eine Aussage nicht vollständig ist, da sie bei der Eingabe schon unvollständig war.

Auch durch Fehler in Vorverarbeitungsschritten können Unvollständigkeiten auftreten. Dafür gibt es verschiedene Ursachen: gesprochene Sprache unterscheidet sich stark von geschriebener Sprache. Die zugrunde liegende Grammatik ist zwar die Gleiche, es kommt jedoch beim Sprechen zu Wiederholungen, Korrekturen, Unterbrechungen, Verzögerungsumlauten und Pausen. Insbesondere Verzögerungen führen dazu, dass Spracherkenner Laute, die keine Bedeutung haben, noch an vorherige erkannte Wörter anhängen [JM14]. Das Ergebnis der Spracherkennung wird somit verfälscht und die erkannte Aussage stimmt nun nicht mehr mit der eigentlichen Eingabe überein.

Ein anderes Problem mit gesprochener Sprache sind Nachtragungen: Dabei werden einem vorangegangenen Satz noch Informationen nachgereicht, beziehungsweise Informationen korrigiert. Die beiden einzelnen Sätze sind unvollständig, nur eine Betrachtung der gesamten Aussage erlaubt eine korrekte Interpretation, wodurch eine richtige satzweise Interpretation der Sätze schwieriger wird.

Außerdem ist es möglich, dass Fehler bei der Sprachverarbeitung von den NLP-Werkzeugen auftreten. Diese können Wörter falsch erkennen, oder entfernen. Das Microsoft-Spracherkennungswerkzeug [AR18] erzielt zum Beispiel eine Wortfehlerrate von 5.1%.

Auch bei dieser Sorte Fehler ist es wichtig, sie zu erkennen, bevor eine weitere Verarbeitung der Aussage stattfindet, um weitere Folgefehler oder ein falsches Verständnis der Aussage zu vermeiden.

Dieses Problem lässt sich im Kontext von unvollständigen Aussagen betrachten. Eine grundlegende Überlegung zur Bewertung der Eignung von Vervollständigungskandidaten muss erstellt werden. Diese könnte auch auf alle vorkommenden Entitäten angewendet werden, um solche Fehler zu finden.

4.1.1. Diskussion der verwandten Arbeiten

Um Unvollständigkeiten in gesprochenen Aussagen zu erkennen, lassen sich Ansätze aus verwandten Arbeiten anwenden. Insbesondere RESI, ein Untermodul von RECAA [Kör14] welches Probleme in Anforderungstexten findet. Die dazugehörenden, unvollständigen Prozesswörter beschreiben eine ähnliche Problematik, wie in diesem Kapitel. RESI verwendet einen wissensbasierten Ansatz um für Prozesswörter Argumentlisten anzufragen. Eine Liste von möglichen Entitäten wird vorgeschlagen. Die grundlegende Idee des Ontologiebasierten Prüfens der Vollständigkeit lässt sich übertragen. Ein entscheidender Unterschied ist jedoch, dass der in dieser Arbeit entwickelte Ansatz automatisch ablaufen soll. RESI generiert eine Liste von semantischen Rollen, ist aber nicht in der Lage zu entscheiden, welche dieser mit welchen Wörtern befüllt werden sollen.

Im Zuge dieser Arbeit muss also, wenn ein ähnlicher Ansatz verwendet werden soll, eine sinnvolle Beschriftung der Eingabe mit Rollenbeschriftungen erfolgen.

Um diese Beschriftungen zu erstellen, gibt es viele verschiedene Beschriftungswerkzeuge, die auf gegebenen Texten Worte mit ihren zugehörigen semantischen Rollen annotieren.

SensiCal [Mue00] erstellt aus Text Kalendereinträge und prüft diese gleichzeitig auf eventuelle Probleme. Zu diesen Problemen gehört es, wenn bestimmte benötigte Informationen, wie zum Beispiel das Datum für ein Kalenderereignis fehlen. Diese Probleme können relativ einfach gefunden werden, da das Format des Kalendereintrags bekannt ist. Dadurch kann bestimmt werden, welche Informationen vorliegen, und welche fehlen.

Dieser Ansatz lässt sich jedoch nur sehr schwer auf diese Domäne übertragen. Dazu müsste ein Modell erstellt werden, das vorgibt, wie eine vollständige Aussage aussieht. Ein Modell dieser Art für natürliche Sprache zu erstellen wäre damit verbunden für jede natürlichsprachliche Aussage herauszufinden, wann sie vollständig ist oder nicht. Aufgrund der hohen Komplexität von Sprache scheint dies jedoch nach einer unpraktischen Art dieses Problem anzugehen. Trotzdem wäre es denkbar, lexikalische Ressourcen zu verwenden, um damit darauf schließen zu können, welche Aussagen vollständig sind.

SensiCal verfolgt regelbasiert aber auch noch einen weiteren Ansatz. Wenn bestimmte Informationen nicht eingegeben werden, fordert SensiCal diese aus einer Ontologie an. So wird die Enduhrzeit eines Termins automatisch festgelegt, wenn die Art des Termins bekannt ist indem die typische Länge dieses Termins in einer Wissensquelle gesucht wird. Mittels einiger dieser fest codierten Regeln können diverse Unvollständigkeiten automatisch mit Vorschlägen wieder vervollständigt werden.

Auch Active [GBC07] kann Unvollständigkeiten automatisch reparieren. In einer ActiveOntology werden dabei Kandidaten, zum Beispiel mögliche Tage für Termine, von Knoten in der ActiveOntology generiert. Diese setzten dabei simple Regeln um, die den Kalender des Nutzers durchsuchen.

Es ist denkbar solche Ansätze für die Vervollständigung von unvollständigen Aussagen zu verwenden, jedoch müssten dann sehr viele Regeln händisch erstellt werden. Dies scheint in Anbetracht der Tatsache, dass eine sehr große Anzahl an verschiedene unvollständiger Aussagen, jedoch unpraktikabel. Regelbasiert könnten jedoch allgemeine Fälle gelöst werden. Fehlen zum Beispiel (zum Beispiel das Subjekt im Imperativ 2. Person Singular), so könnten Regeln definiert werden, die diese Problemkategorie lösen. Für andere Rollen ist diese vermutlich eher schwer möglich. Keiner der Ansätze lässt sich gut auf die Domäne der natürlichen Sprache übertragen, da diese eine deutlich größere Komplexität aufweist.

4.1.2. Erkennen von unvollständigen Aussagen

In diesem Kapitel wird diskutiert, wie unvollständige Aussagen erkannt werden können. Mittels PARSE soll Computersystemen, per Eingabe in natürlicher Sprache, neue Aktionsabläufe beigebracht werden. Eine Anweisung in diesem Ablauf führt immer dann zu Problemen in der Übersetzung, sobald für die Umsetzung auf bereits bekannte Funktionen Informationen fehlen. Das bedeutet ein Parameter, der vom Roboter benötigt wird, wurde nicht gegeben. Dieses System benötigt dabei das Wissen über die Aussage in einer expliziten Form, d.h. selbst wenn die Aussage über den Kontext klar wäre, müssen relevante Informationen explizit gemacht werden. Denn ohne dieses Wissen ist ein korrekter Ablauf nicht möglich. Diese fehlenden Informationen sind die Bestandteile, die von einem System zum Erkennen von unvollständigen Aussagen gefunden werden müssen. Zwischen der Repräsentation des Computersystems und der Eingabe in natürlicher Sprache liegt jedoch eine große Distanz. Es ist nicht möglich, Eingaben direkt zu übersetzten, da die verwendeten Sprachen andere Grammatiken mit stark unterschiedlichen Komplexitäten verwenden. Natürliche Sprache bietet eine Vielzahl von verschiedenen Möglichkeiten eine Aussage mit gleichem Inhalt wiederzugeben. Das System, das diese verstehen soll, verwendet dafür aber nur eine Funktion. Die Übertragung auf die System-Sprache soll aber all diese möglichen Formen verstehen. Um die Sätze besser verstehen zu können ist eine Repräsentation der

Aussagen, die von dieser Komplexität abstrahiert wichtig. PARSE verwendet dazu in einer Graphenrepräsentation verschiedene Kantentypen, um die ursprünglichen Wörter der Eingabe auf einheitliche Relationen, wie etwa jene des Eigennamenerkenners, abzubilden.

Um das Problem der unvollständigen Aussagen betrachten zu können, müssen verschiedene Informationen bekannt sein. Zu einer gesprochenen Eingabe muss dazu bekannt sein, welche Informationen für deren Vollständigkeit überhaupt benötigt werden. Diese Informationen müssen in der Repräsentation der Aussage, also dem Graphen, explizit verfügbar sein. Ebenso ist es wichtig zu wissen, welche Bestandteile einer Aussage welche Rollen übernehmen. Das bedeutet, es muss bekannt sein, wie viele Handelnde welche Rollen in der Aussage spielen, damit sie korrekt umsetzbar ist. Sucht man die benötigten Handelnden für das Verb „laugh“ findet sich nur eine einzige Rolle: der Handelnde. Bei komplexeren Prozessen, zum Beispiel „get“ ist diese Rollenliste deutlich komplexer und beinhaltet mindestens ein Subjekt und Objekt.

Betrachtet man beispielsweise die First-Order-Logic Repräsentation von Wissen, so könnten Relationen wie `Restaurant(Maharani)` und `Serves(Maharani, VegetarianFood)` die Wissensrepräsentation von „Maharani serves vegetarian food.“ sein [JM14]. Aus dem Ausgangssatz ist es nicht ohne weiteres Wissen möglich zu wissen, dass eine Relation `serve` zweistellig sein muss. Diese Informationen zu den Prädikaten von Sätzen liegt nicht implizit vor, sondern existiert nur in expliziten Wissensquellen. Aus solchen Repräsentationen kann abgeleitet werden, ob einer Aussage ein wichtiger Bestandteil fehlt. Fehlt im obigen Beispiel der Restaurant-Name ist auch die Relation `serves(2)` unterbesetzt und es lässt sich folgern, dass die Relation `serves(2)` mit der gegebenen Aussage nicht vollständig ist. Diese Repräsentationen von Handlungsrollen unterschiedet sich jedoch stark, abhängig von der Quelle dieses Wissens. Einige Wissensquellen verwenden zur Beschreibung dieser Argumente semantische Rollen: Diese weisen Prädikaten eine Liste von Rollennamen zu, die beschreiben, welchen Teil der Handlung des Satzes von welchen Entitäten in diesem Satz getragen werden. Solche Rollen lassen sich über Fragen wie „who?“, „when?“, etc. erfragen. Wissensquellen, die diese Form der Repräsentation wählen, sind PropBank, FrameNet, VerbNet, Cyc oder NomBank und haben händisch erstellte Listen, welche Verben Rollennamen zuweisen. Außerdem finden sich in vielen dieser Wissensquellen, auch Ontologien, noch weitere Wissensannahmen über diese semantischen Rollen.

Um eine Aussage über die Vollständigkeit treffen zu können, muss außerdem ermittelt werden, welche Teile der Aussage zur Vollständigkeit beitragen. Das bedeutet, dass für die in der Eingabe vorkommenden Entitäten bestimmt werden muss, welche Rolle sie bereits übernehmen. Dafür gibt es verschiedene automatische Annotations-Werkzeuge. Eines von diesen, ist das in [Ou16] erstellte PARSE-Modul. Dieser Aktionserkenner erkennt in Sätzen Prädikate und weist den gefundenen Entitäten in der Aussage ihre semantischen Rollen aus verschiedenen Ontologien zu. Diese Informationen können dann verwendet werden, um zu entscheiden, welche Rollen tatsächlich noch fehlen, um die Aussage als vollständig zu betrachten.

Für diesen Vorgang ist die Wahl der Wissensquelle relevant. Die Informationen, die sich in den verschiedenen Ontologien findet haben einige Unterschiede. Während PropBank („A0, A1, A2, ...“) und VerbNet („Agent, Patient, Location, Theme“) eine festgelegte Liste von Rollen für alle Verben[1] verwenden, werden in Cyc („thing given, giveer, givee“ für „give“) und FrameNet („Student, Subject, Techer, Material“ für „teach“) verschiedene Rollennamen für jedes Verb verwendet. Dies muss beachtet werden, wenn bei der Verarbeitung von Aussagen semantische Rollen verwendet werden. Während die Rollen aus FrameNet und Cyc zwar durch den oft auf das Prädikat zugeschnittenen Namen eine genauere Be-

[1] Die vollständige Liste dieser Rollen und derer Beschreibungen findet sich in Abschnitt B

schreibung der Rolle liefern, führt die große Menge der Rollen dazu, dass zum Beispiel eine einfache Abbildung der Rollen auf andere Konzepte nicht mehr einfach möglich ist.

Eine vollständige Diskussion über die Eignung der verschiedenen Ontologien findet sich im Entwurf in Abschnitt 5.1.

Da für den weiteren Verlauf, insbesondere die automatische Vervollständigung, Aussagen über das Prüfen von Wissensannahmen erfolgen soll, ergibt sich noch eine weitere Anforderung an die verwendete Ontologie. Sie sollte in der Lage sein, Aussagen aufgrund ihrer Inhalte zu beweisen, bzw. das Gegenteil beweisen. Die meisten Entitätenbeschrifter liefern jedoch nur Beschriftungen aus Ontologien, in denen keine Inferenz möglich ist. Dies ist aus zwei Gründen der Fall: Entweder die Ontologien unterstützen keine Inferenz, oder sie sind in der Domäne zu speziell. VerbNet speichert zum Beispiel nur Informationen über Verben, deren Rollen und Beschreibungen der Rollen. Auch FrameNet speichert primär Informationen über lexikalische Frames. Zu diesen gehören zwar auch annotierte Beispieltexte zu einer bestimmten Aktion, diese Informationen reichen aber nicht aus, um allgemeinere Fragen - wie die Eignung von Kandidatenkonzepten - zu beantworten. Damit Ontologien mit Inferenzfähigkeit verwendet werden können, muss eine Möglichkeit gefunden werden die leicht zugänglichen Rollen von Standardwerkzeugen auf Ontologiekonzepte aus anderen Ontologien zu übertragen. Möglichkeiten um dies umzusetzen, finden sich in Abschnitt 4.1.4.

Um zu entscheiden, welche Aussagen vollständig sind, müssen gefundene Rollen in der Aussage auf eine aus einer Ontologie stammenden Liste von benötigten Rollen abgebildet werden. Durch die Verwendung einer externen Wissensquelle treten aber noch weitere Probleme auf, die nun im Weiteren noch genauer betrachtet werden.

4.1.3. Disambiguierung der Prädikate

Aus den Überlegungen des letzten Kapitels zur Erkennung von unvollständigen Aussagen folgt, dass eine Wissensquelle benötigt wird. Prädikate aus einer Aussage sollen dabei als Suchterm verwendet werden, um in diesem Lexikon einen Eintrag mit Wissen über bestimmte Rollen zu finden. Dabei entsteht ein Problem, das aufgrund der nicht eindeutigen Natur von Sprache auftritt: Viele Wörter in der englischen Sprache haben mehr als eine Bedeutung. Betrachtet man in einer Aussage ein solches polysemes Verb, so muss seine richtige Bedeutung zuerst bestimmt werden. Untersucht man die möglichen Bedeutungen des Wortes „get“, findet man die Bedeutungen des Bringendes („acquiring something“) aber auch das krank werden („stricken by illness“). Sofern man nur diese beiden Bedeutungen betrachtet, ist es schon nicht trivial zu bestimmen, welche Bedeutung im Kontext eines Satzes gemeint ist. Insgesamt definiert WordNet für das Wort „get“ 36 verschiedene Bedeutungen. Obwohl sich einige dieser Rollen ähneln, existieren viele Paare, deren Bedeutung sich sehr stark unterscheidet. Das Finden der richtigen Bedeutung ist für eine Verarbeitung von Sprache wichtig. Würde in einem Kontext die Bedeutung „AcquiringSomething“ korrekt sein, aber das krank werden als richtige Bedeutung gewählt, können Fehler auftreten. Denn die Vorbedingungen und Implikationen der beiden Bedeutungen unterscheiden sich stark. Ebenso unterschiedlich sind die semantischen Rollen, die für die beiden Bedeutungen vorgegeben sind. Sind diese aufgrund einer falschen Wahl des Wortsinns bereits verkehrt, können infolgedessen weitere Fehler auftreten. Eine Vervollständigung gestaltet sich als schwierig, da dann für die falsche Handlung nach geeigneten Konzepten gesucht wird. Eine möglichst gute Disambiguierung ist aus diesem Grund wichtig.

Um diese umzusetzen existieren verschiedene mögliche Herangehensweisen: Die simpelste Methode ist die Wahl der häufigsten Bedeutung (engl. „most frequent sense“). Dabei wird für jedes polyseme Wort bestimmt, welche Bedeutung des Wortes am häufigsten in einem Textkorpus vorkommt. Die Ergebnisse sind bei der Wahl dieser Methode bei etwa

50%-60% [MCK04]. Aufgrund der invers proportionalen Verteilung der Bedeutungen zu der Häufigkeit, ist das Auftreten der häufigsten Bedeutung relativ wahrscheinlich. Beim Betrachten von anderen, eventuell komplexeren Verfahren sollte deswegen geprüft werden, ob sich der Einsatz im Vergleich zu dieser recht simplen Methode lohnt.

Andere Möglichkeiten sind die Verwendung von Standardwerkzeugen, die zum Beispiel statistische Verfahren anwenden. In einer Evaluierung von mehreren verschiedenen Werkzeugen auf einem annotierten Textkorpus erzielten verschiedene Werkzeuge ein F-Maß zwischen etwa 50% und 70% [RCCN17]. Eine Implementierung mit einem statistischen Disambiguierungswerkzeug liefert zum Beispiel ein PARSE-Modul. Dieses beschriftet Wörter in den Aussagen mit ihren zugehörigen Sinnen in WordNet. Dieses Modul stützt sich auf Babelfy [MRN14]. WordNet selbst beinhaltet jedoch nicht die benötigten Rollen-Markierungen, und kann in dieser Form nicht für die Lösung des Problems verwendet werden.

Wird ein Disambiguierungswerkzeug verwendet, das die Bedeutung der Wörter auf WordNet abbildet, ist nicht klar, welche Rollen benötigt werden. Um diese Frage zu beantworten, müsste zusätzlich noch ein äquivalentes Konzept aus einer Ontologie extrahiert werden, das diese Informationen bereitstellt.

Dieser Prozess ist jedoch komplex, eine detaillierte Analyse dieses Problems und mögliche Lösungsansätze finden sich in Abschnitt 4.1.4.

Eine Alternative zu diesem Prozess, ist es, eine Disambiguierung, ähnlich wie in [CCB06] durchzuführen (siehe Abschnitt 3.3). Dabei werden die Konzepte in einer Ontologie - basierend auf Informationen über diese - ausgewählt. Dieser Ansatz funktioniert jedoch nur für Ontologien wie Cyc, eine Umsetzung auf PropBank, Frame- oder VerbNet ist in dieser Form aufgrund der fehlenden Inferenzmöglichkeit nicht möglich. Zuerst werden dafür Kandidaten sowie Vor- und Nachbedingungen für die Verben generiert. Diese Bedingungen stehen in den Ontologien zur Verfügung. Mit den im Satz bereits gefundenen syntaktischen Rollen werden dann die Bedingungen überprüft. Die Bedeutung die die meisten Bedingungen erfüllt, wird als korrekte ausgewählt. Der Vorteil des Ansatzes, ist das direkt, ohne weitere Abbildung ein geeignetes Konzept in der Ontologie verfügbar ist.

Dieses Verfahren hat jedoch zwei grundlegende Schwächen: Wie bereits die Evaluierung der Autoren von [CCB06] zeigt, ist die Präzision geringer, als die Wahl der häufigsten Wortbedeutung. Es ist also fraglich, wie gut die Ergebnisse mit einem ähnlichen Ansatz für diese Arbeit sein können, oder ob der zusätzliche Aufwand sich Lohnt.

Zudem hängt die Disambiguierung von den vorhandenen Informationen ab. Fehlen zum Beispiel die Hälfte der Rollen, können eventuell nicht ausreichend viele Bedingungen bewiesen werden, um eine eindeutige Aussage über die richtige Bedeutung zu treffen. Es ist aber für andere Systeme trotzdem möglich, ein gutes Disambiguierungsergebnis zu erzielen, da auch vorangegangene Sätze betrachtet werden. Dies würde bei diesem Verfahren nicht geschehen.

4.1.4. Übertragen von Ontologiekonzepten

Unter Heranziehung verschiedener Wissensquellen, erhält man mehr Informationen. Ein Problem, welches sich ergibt, ist die uneinheitliche Benennung, die sich in den verschiedenen Quellen findet. Oftmals haben inhaltlich deckungsgleiche Konzepte ähnliche Namen in den Ontologien. Um die Verwendung von mehreren Quellen zu ermöglichen, muss bekannt sein, welche der Konzepte gleich sind.

Dazu werden die Ontologien, beziehungsweise ihre Konzepte, verbunden. Verschiedene Ontologien zu verbinden ist eine Aufgabe, die von verschiedenen Forschergruppen aufgegriffen

wurde. Dabei gilt es aus zwei verschiedenen Ontologien Paare von Konzepten zu finden, die ausreichend gleich sind. Das bedeutet insbesondere nicht, dass diese beiden Konzepte den gleichen Namen tragen müssen. Auch synonyme Begriffe werden bei dieser Aufgabe gefunden. Insbesondere eine automatische Lösung für dieses Problem ist zentraler Gegenstand aktueller Forschung. In [GFAH17] wurde beispielsweise ein Algorithmus entwickelt, der Distanzen zwischen Ontologiekonzepten berechnet. Diese nimmt zwei Ontologien in Graphenform entgegen und berechnet aufgrund der Ähnlichkeiten eine Zuordnung der Konzepte. Insgesamt konnten Zuordnungspräzisionen von 87% bei fünf Antwortkonzepten erreicht werden. Solche Verfahren sind ausreichend präzise, um nutzbare Verknüpfungen zu finden.

Zusätzlich existieren auch einige Projekte, die eine manuelle Übertragung von Ontologien durchführen (z.B. SemLink [Pal09]). Sollte eine solche, händisch erstellte Übertragung von Konzepten für die gewählten Ontologien bestehen, ist eine Lösung einfach. Für jedes Konzept muss dann nur noch ein Eintrag in einer Liste abgefragt werden. Ein Vorteil gegenüber den automatischen Verfahren ist hierbei, dass die Präzision aufgrund der manuellen Erstellung 100% beträgt.

Sollten keine solche Listen für die verwendeten Ontologien verwendet werden, ist es auch denkbar, solche Übertragungslisten selbst zu erstellen. Dies würde die oben genanten Vorteile erbringen, jedoch ist diese Aufgabe - insbesondere aufgrund der Größe der Wissensquellen - zu groß, um realistisch im Rahmen dieser Arbeit erfüllt werden zu können.

4.2. Automatische Vervollständigung

Werden unvollständige Aussagen an ein Sprachverarbeitungssystem übergeben, ist das nicht zwangsläufig ein Grund, die Verarbeitung direkt abzubrechen. Es kann möglich sein, über Wissen zu der Situation der Sprecher und der Welt zu inferieren, wie die Aussage eigentlich aussehen sollte.

In einer Studie ([CBL77]) wurden verschiedenen Lesern komplexe, oftmals auf den ersten Blick schwer verständliche Texte gezeigt. Die Leser sollten dabei gedanklich ein Modell der Situation erstellen, und beschreiben, welche Schritte sie dabei getätigt haben. Daraus ergaben sich die Strategien die von den Lesern verfolgt wurden, um die genaue Bedeutung des Textes zu verstehen. Die Forscher entwickelten daraus eine Theorie zum Verständnis von Textaussagen. Sie folgerten, dass Leser ein mentales Modell erstellen und dieses beim Lesen und Verstehen des Textes konstant verändern. Dabei werden verschiedene Vorgehensweisen zum Verändern des Modells angewendet.

Die - für diese Arbeit - wichtigste Vorgehensweise ist es, eine neue Belegung der vorkommenden Variablen zu finden. Ziel dabei ist, jegliche Variablenbindungen zu verwerfen, die in irgendeiner Form zu einem Konflikt führen. Das bedeutet, dass zum Beispiel geltenden Grundsätze über die Welt verletzt werden.

Das führt zu neuen Interpretationen und weiteren Abwägungen der Korrektheit. In einem Beispieltext kommt der Satz „He plunked down $5 at the window“ vor. Ein Leser beschrieb, dass er auf Basis dieses Satzes entschieden hat, dass es sich um das Fenster eines Wettbüros handeln muss. Beim Lesen des nächsten Satzes „She tried to give him $2.50, but he refused to take it“ hat der Leser diese Theorie verworfen, da er es für unrealistisch hielt, dass er Rückgeld in einem Wettbüro ablehnen würde. Zu diesem Zeitpunkt hat er die Variablenbelegung von „she“ angezweifelt, da ein Konflikt entstanden ist. Erst dann entwickelte er die Theorie, dass die Situation in einem Kino stattfindet, und die Variablenbelegung von „she“ seine Verabredung sein muss.

Für diese Arbeit ist dieser mentale Schritt besonders interessant. Beim Lesen von Texten kann ein ähnliches Verständnis für unbekannte Konzepte, oder auch implizit angenommene Fakten bestehen. Ein Computersystem das eine Aussage wieder vervollständigen soll, muss auch Theorien aufstellen, diese bewerten und im Falle eines Konfliktes verwerfen. In den folgenden Kapiteln soll dazu genauer betrachtet werden, ob und wie es für ein Computersystem möglich sein kann, unvollständige Aussagen - ähnlich dem Inferenzprozess bei Menschen - wieder zu vervollständigen.

4.2.1. Vorhandenes Wissen

Unvollständige Aussagen können nach den Überlegungen aus dem letzten Kapitel nur dann komplettiert werden, wenn bereits klar ist, welche Teile der Aussage unvollständig sind. Ohne dieses Rollenbeschriftungen kann wissensbasiert keine Lösung gefunden werden.

Da, wie auch bei den Rolleninformationen, nicht explizit gegeben ist, welche Entitäten in der Situation noch existieren, müssen diese bestimmt werden. Abseits von den Rollenbeschriftungen müssen also noch weitere Informationen aus einer externen Quelle zur Verfügung stehen, um eine Aussage wieder zu vervollständigen.

Nach einer Verarbeitung der Aussagen mit den in Abschnitt 4.1 vorgestellten Schritten steht Wissen über die fehlenden Akteure zur Verfügung: Neben Rollenbeschriftungen für die bereits vorhandenen Satzteile steht eine Liste von fehlenden Rollen zur Verfügung. Mit dieser Liste können in diesem Schritt Rollen bearbeitet werden, für die noch geeignete Akteure generiert werden müssen.

Lexikalische Wissensquellen aus den vorherigen Schritten liefern aber keine Informationen über Weltannahmen, oder Entitäten in diesen. Die benötigten Wissensannahmen der Welt sollen später verwendet werden, um die Kandidaten für die jeweiligen Rollen auf ihre Tauglichkeit zu prüfen.

Genau diese werden aber benötigt, um Hypothesen zu erstellen. Damit diese ermittelt werden können, muss eine Wissensquelle verwendet werden, die diese Informationen verfügbar macht. Ontologien wie Cyc oder OMCS stellen allgemeine Informationen über die Welt dar. Sie enthalten eine große Menge von simplen Wissensannahmen. Damit können viele Fragen, die keine zu komplexen Domänen behandeln, beantwortet werden.

Analog zu den Überlegungen in der Einleitung sollen neben Allgemeinwissen aber auch wissen über die Situation verwendet werden. Denn insbesondere in der unmittelbaren Umgebung finden sich wichtige Informationen, auf die sich die Handlung beziehen kann. Auch in dem Anwendungsfall für Robotersysteme ist es denkbar, dass ein Sprecher, welcher einem Roboter neue Aktionen beibringt, sich dabei nur auf Objekte und Entitäten in seinem Umfeld bezieht.

Der im Beispielsatz „Get the orange juice“ fehlende Ort könnte vermutlich nicht über eine „upper ontology“ inferiert werden, da die Information über den Lagerort zu spezifisch ist. Aus einer Wissensquelle, die ein solches Szenario genau beschreibt, könnte diese Informationen jedoch generiert werden. Dazu könnten noch weitere genauere Eigenschaften über diese Konzepte verwendet werden, die ebenfalls in einer solchen Wissensquelle gespeichert sind.

Solches Wissen könnte auf verschiedene Weisen bereitgestellt werden. Eine Möglichkeit ist es, eine weitere maschinenverständliche Wissensquelle zu verwenden, die deutlich spezifischere, an die Situation angepasste Informationen enthält. Diese sollte dann alle Informationen über die aktuelle Situation beinhalten.

Alternativ könnten Sensorsysteme Bilder aus der Umgebung wahrnehmen, und diese auf ein eigenes Modell der Situation übertragen. Aufgrund der Komplexität solcher Systeme ist jedoch die Wahl einer einfachen Ontologie sinnvoller.

4.2.2. Generieren von Kandidaten

Für unvollständige Rollen sollen in einem ersten Schritt eine Menge an geeigneten Kandidaten generiert werden. Grundlegend ist die Überlegung, dass eine vergessene, oder als implizit angenommene Entität, bereits bekannt ist. Das bedeutet, dass dem System diese Entität bereits genannt wurde, oder dass sie sichtbar ist. Wenn die Entität genannt wurde, befindet sie sich im Modell des situativen Kontextes, ist sie in der Umgebung sichtbar, so sollte sie als Konzept in der Umgebungsontologie gespeichert sein.

Schwieriger wird es, wenn eine Rolle mit einer Entität gefüllt werden soll, welche dem System noch unbekannt ist. Dies kann nach den Überlegungen in Abschnitt 4.1 auch auftreten, wenn Aussagen in der Vorverarbeitung falsch verarbeitet wurden und dabei Teile von der Aussage vernichtet wurden.

In diesem Fall können Kandidaten mittels Inferenz aus Ontologien generiert werden, welche den Anforderungen der Rolle, und weiteren Bedingungen der Situation genügen. Für diese ist aber eine Überprüfung deutlich schwieriger. Da nicht bekannt ist, welche Objekte und Handlungsträger in der Situation vorhanden sind, ist jeder Kandidat, welcher sich in diesem Umfeld finden könnte, geeignet. So könnte für die Aussage „Get _ from the fridge“ Kandidaten generiert werden, die Inhalte des Kühlschranks sind. Diese Menge ist jedoch relativ groß, und auch für einen Menschen ist es schwer, nur anhand dieser Aussage das richtige Objekt zu inferieren. Es gilt also auch zu betrachten, ob es überhaupt realistisch ist, eine bestimmte Rolle zu vervollständigen, oder in einem Fall dieser Art die Unsicherheit auch zu markieren.

Vertauscht man das verlorene Wort im obigen Beispielsatz, ist die Situation jedoch eine andere: Bei „Get the butter from the _“ könnte über geeignetes Allgemeinwissen inferiert werden, dass sich Butter typischerweise in einem Kühlschrank befindet, und dass dieser in ein Küchenszenario gefunden werden kann. Um im Voraus keine eventuell richtigen Kandidaten zu verwerfen, sollen alle Möglichkeiten geprüft werden. Das bedeutet, dass möglichst viele Entitäten aus dem Kontext und der Umgebungsontologie extrahiert werden, um im Weiteren geprüft zu werden. Aufgrund der hohen Zahl der möglichen generierbaren Kandidaten, kann dieser Ansatz langsam sein. Insbesondere, wenn die Umsetzung der Überprüfung ressourcen-aufwendig ist, kosten viele Kandidaten eine große Menge Zeit.

Eine andere Möglichkeit, das Problem zu lösen ist es die Vervollständigung regelbasiert durchzuführen. Dabei wird, ähnlich zu SensiCal (siehe Abschnitt 3.2), auf eine Liste von möglichen Problemen zurückgegriffen. In dieser Liste sollen dann Problemkategorien erfasst werden, die bei einer fehlerhaften Eingabe auftreten sollen. Ebenso enthält sie Reparaturmöglichkeiten, bzw. die Informationen, die zur Behebung benötigt werden. Dadurch müssen nicht, wie oben beschrieben, mehrere Kandidaten generiert werden, sondern eine direkte Suche in einer Liste ist möglich. In SensiCal sieht eine beispielhafter Ablauf wie folgt aus: Bei der Anfrage „Dinner w/ Susanna at 2“ wurde die Länge des Termins nicht angegeben. SensiCal befragt in diesem Fall eine Wissensquelle nach der Länge eines „dinner“ und fügt diese Information ein.

Solche Regeln lassen sich auch auf andere Domänen übertragen. Für bestimmte - häufig auftretende Fälle - wäre es sinnvoll, diese zu verwenden. Die alleinige Verwendung erscheint jedoch eher unpraktikabel, da eine vollständige Liste möglicher Fehler einen zu großen Umfang hätte, und schier unerstellbar wäre.

4.2.3. Überprüfen der Kandidaten

Für eine geeignete Repräsentation dieser Formulierung lässt sich dieses Problem auch als Klassifizierungsproblem darstellen, bei welchem die Klassen „sinnvoll“ und „nicht sinnvoll“

existieren. Um Klassifizierungsprobleme zu lösen, eigenen sich maschinelle Lernverfahren besonders gut. Diese trainieren einen Klassifizierer auf eine Menge von (beschrifteten) Eingabedaten. Das Problem müsste dann so formuliert werden, dass für jede bestehende Rolle ein Ergebnis vorliegt. Ein Kandidat kann, je nach Aussage, auch mehrere Rollen verwenden, weswegen eine solche Klassifikation für jede Rolle und alle Kandidaten durchgeführt werden muss. Hier würde ein maschinelles Lernverfahren aber zu zwei Problemen führen: Zuerst ist es schwierig, sinnvolle Merkmale in der Eingabe zu finden. Es ist unklar, wie gut natürlichsprachliche Merkmale für eine solche Klassifizierung funktionieren würden. Dazu müssten Experimente durchgeführt werden. Eine vorherige Abbildung auf eine weniger komplexe Sprache, wie zum Beispiel *First Order Logic* würde dieses Problem vermutlich reduzieren. Dann könnten auch andere Merkmale verwendet werden, doch selbst dann ist es schwierig vorherzusagen, ob dieser Ansatz gute Ergebnisse liefert. Ebenso ist es fraglich, welche Daten als Trainingsdaten verwendet werden sollen. Es existieren zwar Textkorpora, die für andere PARSE-Module zur Evaluation verwendet werden. Diese sind nur nicht für eine Klassifizierung dieser Art beschriftet und müssten noch händisch beschriftet werden. Aufgrund dieser Unklarheiten ist eine Lösung mit maschinellen Lernverfahren vermutlich unpraktikabel.

Alternativ könnte ein Inferenzprozess analog zum menschlichen entwickelt werden. Es sollen also so lange neue Kandidaten für Variablen verwendet werden, bis die Aussage in ihrer Gesamtheit Sinn ergibt. Daraus ergibt sich, dass es eine Möglichkeit geben muss, die produzierten Aussagen inhaltlich zu überprüfen. Dazu gehört, Folgen der Annahmen bestimmen zu können und diese auszuwerten.

Inferenz schildert die Möglichkeit eines Systems, aus bereits bekannten Fakten weiteres Wissen oder Konsequenzen zu folgern [JM14]. Um Inferenz durchzuführen, müssen einem System Inferenzregeln bekannt sein. Formal funktioniert Inferenz in Ontologien, indem auf einer bestimmten Wissensmenge bestimmte Nachschlageregeln angewendet werden. In diesen stehen Ableitungen der Form `antecedent` → `consequent`. Wenn der linke Teil der Regel gilt, lässt sich der rechte Teil der Regel inferieren. Solch eine beispielhafte Regel sieht in CycL wie folgt aus:

```
(implies
 (and
  (isa ?INST1 Vegetarian)
  (isa ?INST2 VegetarianDish)
 )
 (eatsWillingly ?INST1 ?INST2)
)
```

Diese Regel besagt, wenn die beiden ersten Prädikate wahr sind, ist auch die implizierte Regel wahr. Genauer bedeutet dies, dass aus dem Wissen über die Existenz eines Vegetariers `INST1` und eines vegetarischen Gerichts `INST2` inferiert werden kann, dass dieser das Gericht auch essen würde. Diese Regeln generieren neues Wissen, das Prädikat `#$eatsWillingly` muss dafür nicht mit Wissen gefüllt werden, diese Einträge können automatisch generiert werden. Mit diesen Regeln muss nicht zwangsläufig neues Wissen generiert werden. Sie können auch verwendet werden, um spezifische Anfragen (engl. „query“) zu beweisen.

Dazu wird die gesuchte Anfrage in der Wissensquelle gesucht. Wird sie nicht gefunden, werden Inferenzregeln gesucht, bei welchen der Konsequenzteil auf die Anfrage passt. Die Anfrage kann dann bewiesen werden, wenn die Vorbedingung von einer gefundenen Inferenzregel bewiesen wird. Dieses Verfahren kann rekursiv angewendet werden, bis keine weiteren Regeln mehr gefunden werden.

Um generierte Hypothesen - im Kontext von unvollständigen Aussagen - zu prüfen, sollte

ein Eintrag in der verwendeten Wissensquelle vorhanden sein, der die übernommene Rolle beschreibt. Dieser Eintrag kann als Inferenzanfrage verwendet werden, um zu überprüfen, ob diese Belegung mit dem bekannten Wissen der Wissensquelle beweisbar ist.

Wenn dies für Kandidaten für Rollen nicht möglich ist, kann die Theorie verworfen werden, und eine neue Belegung der Variablen kann überprüft werden. Dafür müssen jedoch die Kandidaten, sowie die benötigten Prädikate in der verwendeten Ontologie bereit stehen. Ansonsten müssen diese auf geeignete Ontologie-Konzepte übertragen werden (vgl. Abschnitt 5.3.1)

Außerdem muss für die geprüften Prädikate noch bestimmt werden, an welcher Stelle die Entitäten als Argument passen. Für das vorherige Beispiel `#$eatsWillingly ?INST1 ?INST2` muss klar sein, welcher Kandidat das Gericht und welches der Vegetarier sein muss. Dieses Problem lässt sich aber bei vielen Prädikaten relativ einfach lösen. Da die Inferenz Regeln beachtet, die den Argumenten Typen zuweist, bricht eine Inferenz sehr schnell ab, wenn ein Vegetarier anstelle eines vegetarischen Gerichts als Variablenbindung übergeben wird. Typfehler erzeugen also keinen sonderlich großen Mehraufwand bei der Berechnung des Ergebnisses. Deswegen können für Anfragen, bei denen nicht klar ist, welche Variable mit welchen Kandidaten instantiiert werden sollen, einfach alle Möglichkeiten geprüft werden. Inferenzen, welche aufgrund falscher Typen nicht beweisbar sind, brechen dabei einfach ab und geben „falsch“ zurück. Die Antwort auf die eigentlich gestellte Frage ist dann „wahr“, wenn mindestens eine Antwort „wahr“ war.

Im Folgenden ist eine Beispielinferenz für das Prädikat `#$eatsWillingly` aufgeführt: Ist etwa nicht klar, für wen das vegetarische Gericht `#$pancakes` gekocht werden soll, werden Kandidaten generiert. Darunter könnte zum Beispiel die Person `#$John` sein. Unter der Annahme, dass eine Inferenzregel diese Anfrage auf `#$eatsWillingly` ableitet ergeben sich die folgenden beiden Anfragen.

```
(eatsWillingly John Pancakes)
(eatsWillingly Pancakes John)
```

Mit den Wissensannahmen `(isa John Vegetarian)` und `(isa Pancakes VegetarianDish)` beweist eine Inferenz die zweite Aussage. Die erste Aussage kann nicht bewiesen werden, da keine der gefundenen Inferenzregeln beweisbar waren.

Das Ergebnis dieser Anfrage ist also „wahr“, John könnte der „Esser“ der Pfannenkuchen sein.

5. Entwurf

In diesem Kapitel wird der grundlegende Entwurf eines Werkzeugs geschildert, das nach den Überlegungen aus Kapitel 4 in gesprochenen Aussagen Unvollständigkeiten erkennt und automatisch behebt.

Zuerst wird entschieden, welche Ontologie, beziehungsweise welche Kombination von Ontologien zum Bezug des Weltwissens verwendet werden soll. Aufgrund dieser Wahl folgt ein - konkret auf die Ontologie bezogener - Entwurf der Teilschritte für die Erkennung von unvollständigen Aussagen in Abschnitt 5.2 und der automatischen Vervollständigung in Abschnitt 5.3.

5.1. Wahl der Ontologie

Die Wahl der besten Ontologie ist nicht nur von diesem Teilschritt abhängig. Ein Wechsel der Wissensquellen zwischen den Bearbeitungsschritten ist mit hohem Aufwand verbunden, da die Konzepte zwischen diesen dann übertragen werden müssten. Um diesen komplexen und nicht fehlerfreien Prozess zu vermeiden, soll eine Ontologie gewählt werden, die Funktionalität für alle Arbeitsschritte verfügt. Gesucht wird also eine Ontologie, die Rollenbeschriftungen für Aktionen besitzt und eine Möglichkeit der Inferenz auf diesem Wissen besitzt.

In Abschnitt 4.2.1 wurden die Ontologien ConceptNet und Cyc als mögliche Kandidaten vorgestellt. ConceptNet beinhaltet viele Konzepte, und verbindet diese über Kanten mit Relationsnamen. Für Konzepte, welche als Knoten repräsentiert werden, können über eine Anfrage an die Ontologie alle ausgehenden Kanten angefragt werden. So erhält man eine nach Kategorien sortierte Sammlung von Relationen. Unter diesen befinden sich viele Wissensrelationen, wie etwa ähnliche Begriffe, Antonyme, aber auch Unteraktionen. Diese beinhalten aber nur selten Informationen über Handlungsträger. Aus diesem Grund ist die Verwendung von ConceptNet eher ungeeignet.

Die Cyc Ontologie enthält in der `UniversalEnglishVocabularyMt` Mikrotheorie ein Prädikat, das einer Konstante, die eine Aktion repräsentiert, eine Liste von Handlungsträgern zuweist. Damit erfüllt sie die Anforderung der Rollenbeschriftungen. Außerdem verfügt sie über eine eigene Inferenzmaschine, welche das Inferieren von neuem Wissen und das Beweisen von Aussagen ermöglicht. Folglich muss kein externes Inferenzwerkzeug verwendet werden. Cyc enthält neben dem Wissen über die Rollen eine große Menge an Allgemeinwissen, das für das Vervollständigen der Rollen sinnvoll sein kann.

Eine Übersicht der Syntax der im Folgenden verwendeten CycL-Ausdrücke befindet sich in Abschnitt 2.5.

Von den verwendeten PARSE-Agenten werden bereits andere Ontologien verwendet. Der Rollenbeschrifter fügt Annotationen aus VerbNet, FrameNet und PropBank hinzu. Auch diese Informationen sollen für den hier entworfenen Agenten nutzbar sein. Um diese Verwendung einfacher zu machen, wird SemLink [Pal09] verwendet. SemLink besteht aus manuell erstellten Zuordnungen zwischen VerbNet, FrameNet und PropBank. Durch diese Zuordnung können bereits bekannte Konzepte, und Rollennamen aus einer dieser Ontologien auf die anderen abgebildet werden. Damit soll auch eine Übertragung von Konzepten auf Cyc, durch Bezug von weiterem Wissen aus jeder dieser Quellen, einfacher werden.

5.2. Unvollständige Aussagen

Um unvollständige Aussagen in gesprochener Sprache zu erkennen, muss zunächst das Prädikat in einem Satz bekannt sein. Im Bezug darauf kann dann über Ontologieanfragen eine Liste an möglichen Handlungsträgern verwendet werden. Zur Erkennung der Prädikate, und einer Beschriftung der vorhandenen Rollen wird der PARSE-Agent Aktionserkenner [Ou16] verwendet. Dieses Werkzeug arbeitet auf dem Graphen der Aussage und fügt dabei Knotenbeschriftungen für die Rollenbeschriftungen der einzelnen Wörter ein. Ebenfalls wird der PARSE-Agent Rollenbeschrifter verwendet, welcher in der Aussage semantische Rollen in Form von neuen Kanten beschriftet.

Für die Beispielaussage „Bring me the orange juice“ sieht der Graph nach der Eingabe wie folgt aus.

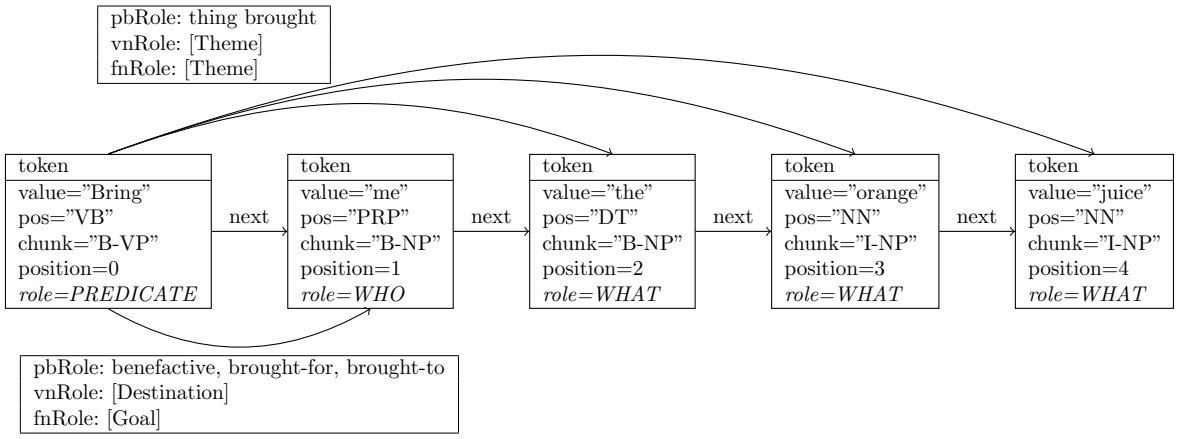

Abbildung 5.1.: Ergebnisse des Aktionserkenners und des Rollenbeschrifters

Der Rollenerkenner erzeugt dabei für Entitäten, die durch mehrere Wörter beschrieben werden je eine Kante pro Wort. Diese tragen dieselbe Kantenbeschriftung, die im obigen Bild aus Platzgründen ausgelassen wurde.

Neben den hier dargestellten Eigenschaften sind noch Einträge zum zugehörigen Verb und den Namen der Einträge in PropBank, VerbNet und FrameNet zur Verfügung.

5.2.1. Disambiguierung der Prädikate

Um geeignete Kandidaten in Cyc zu finden muss zuerst eine Konstante gefunden werden, die die Handlung repräsentiert.

Für diese Disambiguierung der Verben wurde der Ansatz aus Abschnitt 3.3 angepasst getestet. Dabei wurde für jedes Verb, das in der Eingabe vorkommt, eine Liste an möglichen Bedeutungen angefragt. Um das Prozesswort zu extrahieren, wird die Eingabe nach der Rollenbeschriftung `PREDICATE` durchsucht. Besteht dieses aus mehreren Wörtern, so werden diese konkateniert. Ist eine solche Beschriftung nach Ablauf des Aktionserkenners nicht verfügbar, wird der Ablauf des Agenten abgebrochen. Dieser Fall tritt jedoch nur sehr selten auf, da sich auch der Ablauf des Aktionserkenners auf der korrekten Erkennung des Prädikates basiert. Es wäre zwar nicht schwer, dieses nun zu finden, da aber in diesem Fall auch die Beschriftungen der Rollen fehlen, wird eine - auf diesen basierende - Verarbeitung der Aussage schwer möglich.

Für den Beispielsatz „Get the orange juice“ ergibt sich daraus die Anfrage:

```
(verbSemTrans Get-TheWord ??SENSE ??FRAMETYPE ?FRAME)
```

`verbSemTrans` ist ein Cyc-Prädikat, welches ein Wort (Argument 1) auf den Zähler der Bedeutung (Argument 2), die Art des Frames (Argument 3), und einen Frame (Argument 4) abbildet. Für die Auswertung ist nur der Inhalt des Frames relevant. Die FrameNummer könnte noch verwendet werden, um bei Unklarheiten die erste Bedeutung auszuwählen. Die Variablen `SENSE` und `FRAMETYPE` werden deswegen mit dem doppelten Fragezeichen als für die Antwort irrelevant erklärt und somit nicht belegt. Für die obige Anfrage liefert Cyc 13 verschiedene Ergebnisse. In der nachfolgenden Tabelle befindet sich eine Auswahl der Antworten.

?FRAMETYPE	?FRAME
TransitiveADJPFrame	(and (isa :ACTION IntrinsicStateChangeEvent) (objectOfStateChange :ACTION :SUBJECT) (toState :SUBJECT :OBJ-COL))
TransitiveNPFrame	(and (isa :ACTION AcquiringAnObject) (performedBy :ACTION :SUBJECT) (objectActedOn :ACTION :OBJECT))
TransitiveNPFrame	(and (isa :ACTION DevelopingAilment) (developsAilment :ACTION :SUBJECT) (ailmentDeveloping :ACTION :OBJECT))

Diese Frames bestehen aus verschiedenen Annahmen, welche für die jeweiligen Bedeutungen des Wortes gelten müssen. So wird in Cyc angenommen, dass für eine `ACTION`, welche das krank werden (`DevelopingAilment`) ist, zwei weitere Bedingungen für `ACTION`, `SUBJECT` und `OBJECT` gelten müssen. Das Subjekt des Satzes muss dabei die Krankheit entwickeln, und das Objekt muss die entwickelte Krankheit darstellen.

Die zweite Antwort enthält die in diesem Kontext korrekte Bedeutung (`AcquiringAnObject`). Dabei gilt, dass eine Aktion vom Subjekt ausgeführt wird und dabei das Objekt bewegt, bzw. behandelt wird.

Für jede Bedeutung gibt es eine Menge an Annahmen, die alle wahr sein müssen, damit diese Bedeutung sinnvoll ist. Der Ansatz, welcher entwickelt wurde verwendet diese Annahmen und überprüft für jede Bedeutung, wie viele dieser Annahmen im Kontext der Aussage bewiesen werden können. Schlussendlich wird die Bedeutung ausgewählt, der die meisten beweisbaren Annahmen enthält.

Dazu müssen die in den Annahmen vorkommenden Variablen noch gebunden werden. Dafür werden die Rollenbeschriftungen des Aktionserkenners verwendet. So werden zum Beispiel „WHAT“ auf `:OBJECT` übertragen. Um Konzepte in der Ontologie zu finden, die die Entitäten aus dem Satz wiedergeben werden die Überlegungen aus Abschnitt 5.3.1 verwendet.

Der intensivste Ansatz ist, die Annahmen mit den quantifizierten Variablen an die Inferenzmaschine zu übergeben, um zu prüfen, ob sie mit diesen Variablenbindungen gelten kann. Ein Problem dabei ist, dass die Menge der Relationen in der Ontologie nicht ausreichend gefüllt ist, um eine Aussage in jeder Domäne treffen zu können. Oft schlägt eine Inferenz fehl, da das benötigte Wissen nicht vorhanden ist. Das bedeutet aber nicht, dass die Aussage dadurch falsch ist. Um dieses Problem zu umgehen, werden nicht die Annahmen direkt ausgewertet, sondern ihre Argumenttypen mit den Typen der Konzepte aus der Aussage verglichen.

Dabei wird das Prädikat `(#$argIsa ?Predicate ?Num ?Collection)` verwendet. Dieses verbindet Prädikate mit ihren Argumenttypen und der jeweiligen Argumentnummer. Die Wissenseintragungen für das Prädikat `developsAilment` sehen wie folgt aus:

```
(argIsa developsAilment 1 DevelopingAilment)
(argIsa developsAilment 2 Organism-Whole)
```

Dabei wird beschrieben, dass für etwas, das eine Krankheit entwickelt, ein ganzheitlich betrachteter Organismus sein muss. Für Konzepte, wie etwa `OrangeJuice`, lässt sich die Oberklasse mit einer einzigen Anfrage bestimmen:

```
(genls OrangeJuice ?X)
```

Antworten für `?X` beinhalten `Drink`, `Mixture` und `Juice`. Für diese Antworten kann geprüft werden, ob eine Vererbungsbeziehung zwischen ihnen und der Klasse `OrganismWhole` existiert. Genauer kann dieser erste Schritt sogar übersprungen werden und direkt eine Verebungsbeziehung zwischen `OrangeJuice` und `OrganismWhole` geprüft werden. Dafür wird das Prädikat `genls ?SUB ?SUPER` verwendet. Ist die Aussage beweisbar, kann man annehmen, dass es möglich ist, den Kandidaten an der bestimmten Stelle zu verwenden. Der Beispielfall schlägt fehlt, da `OrangeJuice` keine Unterklasse der Sammlung `Organism-Whole` ist. Folglich wird dieser Teil der Annahme verworfen. Dieses Verfahren wird für alle anderen Annahmen angewendet, und der beste Kandidat wird ausgewählt.

Dieses Verfahren ist jedoch nicht sonderlich präzise. Sobald sich Annahmen zu stark ähneln, wird oft ein falsches Prädikat gewählt. Außerdem wurden bei einigen Testfällen aufgrund fehlender oder falscher Markierungen des Aktionserkenners die falschen Prädikate gewählt. Letztlich wurde dieses Verfahren nicht verwendet, da es nur funktioniert, solange die Handlungsträger richtig annotiert sind und vor alledem vorhanden sind. Da sich diese Arbeit insbesondere mit unvollständigen Aussagen befasst, ist dieser Ansatz bei dieser Problemmenge besonders problematisch. Eine vollständige Evaluation dieser Methode wäre in diesem Kontext jedoch interessant.

Aus diesem Grund soll ein anderes Verfahren für die Auswahl des richtigen Wortsinns gewählt werden. Cyc verfügt über eine spezielle Mikrotheorie, die ein Prädikat beinhaltet, das Konzepten in Cyc ihre äquivalenten Konzepte in WordNet zuweisen. Dann kann ein Standardwerkzeug zur Disambiguierung auf WordNet-Bedeutungen verwendet werden. Diese können dann in einem einfachen Schritt auf Prädikate in Cyc übertragen werden. Der Vorteil ist, dass man die deutlich bessere Präzision der Standardwerkzeuge ausnutzen kann. Zusätzlich reduziert diese Vorgehensweise die Komplexität der Implementierung und sorgt dafür, dass es weniger Stellen bei der Ausführung gibt, an der Fehler auftreten. Hierzu wurde das Werkzeug Babelfy [MCN14] auf Tauglichkeit geprüft, das eine Schnittstelle

für Disambiguierung bietet. Beim Verwenden dieses Werkzeuges ist jedoch aufgefallen, dass die Disambiguierung hauptsächlich für Substantive und benannte Entitäten erfolgt. Obwohl Handlungen auch als Nominalisierungen auftreten können, ist das fehlende Disambiguierungspotential für Verben ein Ausschlusskriterium für dieses Werkzeug.

Für einige Beispieltexte wurde die Präzision von MFS getestet. Die Ergebnisse waren für viele Prädikate gut. Zur Umsetzung werden Frames für das Prädikat, wie in 5.2.1 angefragt. Der erste Frame, ist in Cyc der häufigste, alle darauffolgenden Frames haben eine kleinere Auftretenswahrscheinlichkeit. Um die häufigste Bedeutung aus Cyc anzufragen muss nur die erste Antwort gewählt werden. Bei Wörtern, bei denen sich Konzepte aus verschiedenen Frames ähneln, sind diese auch direkt aufeinanderfolgend. In solchen Fällen ist es für das Erzeugen von Rollenlisten irrelevant, welcher der beiden Frames gewählt wird, da nur die `#$isa` Zeile ausschlaggebend ist. Die Rollen werden dann später direkt aus dem Ontologiekonzept generiert.

Daraufhin werden die verschiedenen Argumente des `and` Prädikates ausgewertet, und das letzte Argument des `isa` Terms wird extrahiert. Dies beinhaltet immer die Aktion. Sobald diese bekannt ist, können die Rollen aus Cyc extrahiert werden.

5.2.2. Extraktion der semantischen Rollen

In diesem Schritt steht nun ein Prädikat in Cyc zur Verfügung, das die Handlung aus der Orginalaussage beschreibt. Für das Beispiel „Get the orange juice“ ist die Konstante `AcquiringAnObject` selektiert.

Es gilt nun eine Liste von Rollen zu extrahieren, welche die benötigten Rollen der Handlung beschreibt. Dafür wird ein Prädikat verwendet, das einer Handlung eine Liste von Rollen zuweist: `(requiredActorSlots ?PRED ?ACTOR)` liefert alle für die Handlung und derer übergeordneten Handlungen die Handlungsträger.

Die Anfrage

```
(requiredActorSlots AcquiringAnObject ?ACTOR)
```

liefert, bei Standardeinstellung der Inferenzmaschine, 20 mögliche Handlungsträger in der Mikrotheorie `CurrentWorldDataCollectorMt-NonHomocentric`, welche den aktuellen Zustand der Teile der Welt modelliert, die nicht fiktiv sind. Darunter befinden sich die Rollen `eventOccursAt`, `doneBy`, `objectActedOn`, `actors`, `performedBy` und `takesControl`. Selbst unter dieser kleinen Auswahl an Rollen befinden sich Paare, welche eine starke Ähnlichkeit aufweisen. Die Rollen `doneBy` und `performedBy` beschreiben zum Beispiel dieselbe Rolle, den Handlungsträger der Aktion, der das Objekt bewegt. Diese Duplikate tauchen aufgrund der mehrfachen Oberklassenzugehörigkeit auf. Die Konstante `AcquiringAnObject` besitzt die Oberklassen `PurposefulAction` und `GainingControl` die jeweils ihre eigenen benötigten Rollen definieren. Diese Hierarchie wird vererbt, dabei wird aber nicht beachtet, ob Rollen aus den verschiedenen Oberklassen sich ähneln. Um weiterhin sinnvolle Ergebnisse zu erzielen, müssen diese ähnlichen Rollen automatisch erkannt und auf eine Rolle reduziert werden.

Eine Möglichkeit zur Bestimmung von Ähnlichkeiten zwischen verschiedenen Rollen ist es, ein Cyc-Prädikat für diesen Zweck abzufragen. In Cyc existiert `(#$similarTo ?OBJ1 ?OBJ2)`. Dieses Prädikat ist wahr, wenn zwei Konstanten in der Ontologie sich ähnlich sind. Die Ähnlichkeit ist durch verschiedene Inferenzregeln definiert. Wird dieses Prädikat verwendet, könnte paarweise für ähnliche Rollen entschieden werden, ob eine der beiden verworfen werden soll. Da Cyc aber für dieses Prädikat nicht ausreichend viele Informationen speichert, ist eine Inferenz auf Rollenkonzepten fast nie möglich. Trotzdem soll es

angefragt werden, denn sobald es wahr ist, sind die Ergebnisse nutzbar und haben einen positiven Einfluss auf die Güte der Rollenliste.

Zusätzlich zu diesem Prädikat wird zwischen den gefundenen Rollen je paarweise geprüft, ob es sich bei den Rollen um Verallgemeinerungen handelt. Dafür wird das `#$genlPreds` Prädikat verwendet, welches Vererbungsbeziehungen zwischen Prädikaten, analog zu `#$genls` mit Konstanten speichert. Für jedes Paar, das eine solche Beziehung besitzt, wird die allgemeinere der beiden Rollen verworfen. Dadurch wird sichergestellt, dass keine Annahmen zu den Rollen, die in der spezifischen Konstante gespeichert sind verworfen werden.

Ein weiterer Ansatz ist, es in der vollständigen Liste Ähnlichkeiten in den Rollenbeschreibungen zu suchen. Dazu werden die Kommentare der Rollen paarweise verglichen. Als Vergleichsmetrik wurde die Lesk-Ähnlichkeit [Les86] gewählt. Diese vergibt für zwei Konzepte eine Punktzahl, die die Ähnlichkeit der Texte beschreibt. Je größer diese Zahl, desto ähnlicher sind die Konzepte. Für alle Wortsequenzen wird geprüft, ob diese in dem anderen Text auch vorkommen. Ist dies der Fall, wird das Quadrat der Länge des Segments auf die Punktzahl addiert. Längere Passagen, die in Texten übereinstimmen werden dadurch stärker gewichtet. Über einen experimentell bestimmten Schwellwert sollen dann übereinstimmende Rollen verschmolzen werden, sobald dieser überschritten ist. So konnte beispielsweise bestimmt werden, dass es sich bei den Rollen `objectMoving` und `primaryObjectMoving` um dieselbe Rolle handelt. Auch diese Metrik soll noch verwendet, um gleiche, bis jetzt unauffindbare Paare zu verschmelzen.

5.2.3. Übertragen von semantischen Rollen und Erkennen von unvollständigen Aussagen

Um zu erkennen, welche Aussagen unvollständig sind, muss überprüft werden, welche der im letzten Schritt gefundenen Rollen in der Aussage bereits vorkommen. Dafür sollen nun die verschiedenen erstellten Rollenbeschriftungen des Aktionserkenners verwendet werden. Um zu bestimmen, welche Rollen in Cyc fehlen muss geklärt werden, welche der gefundenen Rollen bereits vorhanden sind. Dazu müssen die Rollen aus FrameNet, VerbNet und PropBank auf die gefundenen Konzepte aus Cyc übetragen werden.

In vorherigen Kapitel wurde ein Lesk-basiertes Verfahren vorgestellt, welches Ähnlichkeiten zwischen Rollen findet. Dieses soll nun auch verwendet werden, um Rollen aus verschiedenen Ontologien Ähnlichkeitswerte zuzuweisen. Ein großes Problem bei der Verwendung besteht jedoch bei den bestehenden Beschriftungen, beziehungsweise den Ontologien, aus denen sie stammen. Während VerbNet und Cyc Glossen zu den Rollen angeben, gibt es solche Informationen in PropBank nicht. VerbNet- und FrameNet-Rollen könnten also mit Cyc-Rollen auf Ähnlichkeit geprüft werden, ohne eine manuelle Erstellung dieser Beschreibungen für PropBank lässt sich dieser Ansatz nicht auf dieser Ontologie anwenden. Sinnvoller wäre die händische Erstellung einer Tabelle, welche diese Rollenübertragung liefert. Da aber viele Handlungen in Cyc ihre eigenen Rollen[1] besitzen, ist diese Erstellung unrealistisch.

Das Lesk-Verfahren wird zwei mal angewendet. Ein mal zwischen den Rollen von VerbNet und Cyc und zwischen den Rollen von FrameNet und Cyc. Es ist bekannt, welche Konzepte in FrameNet und VerbNet verwendet werden. Ebenso sind die Rollen bekannt, zu welchen die Beschreibungen extrahiert werden. Für den Cyc-Kommentar zu der Rolle, und den Glossen aus den beiden Ontologien wird dann mit dem Lesk-Algorithmus die Ähnlichkeit bestimmt, und die beiden Ergebniswerte addiert.

[1] Die Anfrage `(requiredActorsSlots ??ACTION ?ROLE)` liefert ca. 500 mögliche Antworten für `?ROLE`. Die Sammlung `actors` enthält über 1200 Unterkonzepte)

Die Cyc-Rolle, welche die beste Bewertung erhält, wird dann als Beschriftung für diese Rolle verwendet.

Der Beispielsatz „Bring me the orange juice“ und der Aktionserkenner liefern den FrameNet-Frame `[Bringing]`, und die gefundenen Rollen `Theme` und `Goal`. In VerbNet wird der Frame `[bring-11.3-1]` geliefert, der wiederum die Rollen `Theme` und `Destination` liefert. In der nachfolgenden Tabelle sind die Ergebnisse der Ähnlichkeiten mit dem Lesk-Algorithmus für die möglichen Cyc-Rollen aufgelistet. Außerdem ist die Summe der Ähnlichkeiten von gleichen Konzepten aus verschiedenen Lexika in einer eigenen Spalte aufgezeigt.

	Theme(FN)	Theme(VN)	**Sum**	Goal(FN)	Destination(VN)	**Sum**
giver	14	13	27	14	3	17
to-Generic	16	22	38	16	13	**29**
patient-Generic	11	29	**41**	11	17	28

Abbildung 5.2.: Ähnlichkeitswerte verschiedener Rollen aus Cyc, FrameNet (FN) und VerbNet (VN).

Das Maximum der Summe beider Ähnlichkeiten ist hervor gehoben. Diese maximalen Einträge werden für die jeweilige vorhandene Rolle aus Verb-/FrameNet übernommen. Für die `Theme`-Rolle wird also die Cyc-Rolle `patient-Generic`, für die `Goal`-Rolle wird die `to-Generic` Rolle übernommen.

Die hier nicht übertragene Rolle ist jene, die in der Aussage fehlt. In diesem Beispiel fehlt also die Rolle `giver`, es ist also unklar, wer die Handlung ausführt.

Wie sich schon in diesem Beispiel zeigt, ist dies Übertragung nicht perfekt. Der Unterschied der Ähnlichkeiten zwischen den Rollen `patient-Generic` und `to-Generic` ist nur sehr klein. Es ist denkbar, dass bei anderen Rollen für andere Prädikate die Ähnlichkeiten zum Verbinden von falschen Rollen führt. Dies ist eine Fehlerquelle, die zu großen Problemen führen kann, da dann im nächsten Schritt auch für bereits vorhandene Rollen noch neue Handlungsträger gesucht werden. Bei der Evaluation des Ansatzes gilt es zu klären, wie gut dieser Ansatz tatsächlich funktioniert, beziehungsweise wie viele Fehler produziert werden.

5.3. Automatische Vervollständigung

In diesem Kapitel werden die Entwurfsentscheidungen zur automatischen Vervollständigung von vorher unvollständigen Aussagen aufgezeigt. Der Ablauf dieses Prozesses ist aufgeteilt in das Generieren der Kandidaten, einer ersten Vervollständigung über fest definierte Regeln, einer Prüfung der generierten Kandidaten und dem Einfügen der Ergebnisse in den Graphen.

5.3.1. Finden von Ontologiekonzepten

Später sollen alle möglichen Kandidaten über Ontologieanfragen auf ihre Tauglichkeit geprüft werden. Dazu wird, wie bereits in Abschnitt 5.1 diskutiert, die Cyc-Ontologie verwendet.

Es sollen nun für Entitäten aus dem Kontext, anderen Ontologie oder anderen Quellen (z.B.: Konfigurationsdateien) auf eine Konstante in Cyc überführt werden. Dazu werden verschiedene Suchschritte nacheinander angewendet.

Vorerst werden aus jeder zu verarbeitende Phrase, die aus mehreren Worten besteht, sämtliche vorkommende Artikel entfernt. In den Wissensquellen, welche zur Suche verwendet werden, kommen diese auch nicht vor.

Zuerst werden bei der Suche vom Nutzer konfigurierbare Ersetzungsregeln verwendet. Diese können häufig vorkommende Unvollständigkeiten, die auf dasselbe Problem zurück zu führen sind, direkt lösen. Außerdem kann ein Nutzer Abbildungen für Phrasen definieren, die von dem Agenten sonst immer falsch abgebildet werden. Diese Definitionen werden in einer Textdatei im `properties`-Dateiformat gespeichert. Für den Suchbegriff wird diese Datei durchsucht, und ein Treffer wird angewendet. Phrasen, die in dieser Liste nicht gefunden werden, werden nun in der Ontologie gesucht.

Dabei werden dann die Wörter konkateniert und als Binnenmajuskel direkt in Cyc gesucht. Wenn ein Konzept auf diese Weise gefunden wird, so wird dieses direkt übernommen.

Diese Methode funktioniert für alle Wörter, welche nur eine einzige Bedeutung haben. Für die Wortbeschreibungen finden sich in Cyc eigene Konzepteinträge, die aber auf „the-Word“ enden und somit nicht im Konflikt zu der Suche stehen.

Sollten verschiedene Bedeutungen des Wortes in Cyc verfügbar sein, so sind diese im Titel bereits beschrieben: Zum Beispiel findet man in Cyc für das englische Wort „bank“ die verschiedenen Konzepte `#$BankOrganization`,`#$BankAccount`, `#$BankBuilding` und noch einige spezifischere Bankinstanzen. In solchen Fällen riskiert man also mittels einer direkten Suche nicht, ein falsches Konzept zu wählen, da diese unterschiedliche Namen tragen.

Für polyseme Wörter wird in dieser Arbeit der erste Treffer in einer Suche durch die Cyc-Ontologie verwendet. Jedes Suchergebnis für eine Anfrage in natürlicher Sprache ist eine Liste aus Tupeln. Diese Tupel bestehen aus einem Konzeptnamen in Cyc sowie einer Beschreibung des Konzepts in natürlicher Sprache. Die Ergebnisliste wird dann nach ihrer Ähnlichkeit zu der Eingabe sortiert und der erste Eintrag der Liste gewählt.

Diese Suche liefert ein Konzept, das in natürlicher Sprache möglichst ähnlich ist; dennoch ist es denkbar, dass dabei falsche Konzepte ausgewählt werden. Auf diese Weise wird nur ein möglichst passendes Konzept ohne vollständige Disambiguierung gesucht. Für eine richtige Wahl der Konzepte müsste ein Disambiguierungswerkzeug verwendet werden, das Substantive auf geeignete Weise auf WordNet oder Cyc abbildet. Wie sehr die Genauigkeit dieser Wahl die Güte der Ergebnisse beeinflusst, muss in der Evaluation überprüft werden.

Ein letztes Problem, das während Implementierungstests aufkam, sind Kurzformen von Wörtern, die insbesondere in der gesprochener Sprache verwendet werden. Das Konzept „Refridgerator“ lässt sich problemlos direkt in der Ontologie finden, sucht man aber nach der Kurzform „fridge“ erhält man keine Ergebnisse. Um dieses Problem zu lösen, werden diese Kurzformen aufgelöst, und wieder auf ihre Ursprungsform zurückgeführt.

Dazu werden Wikipedia-Artikelseiten verwendet. Für diverse Wörter und deren Kurzformen existiert nur eine einzige Seite zum Thema. Alle anderen Kurzformen sind lediglich Weiterleitungsseiten, die auf die korrekte Seite verweisen. Diese werden hier verwendet, um aus diesen Kurzformen ihre Grundform wieder herzuleiten. Für noch unbekannte Wörter werden Wikipedia-Seiten mit deren Namen gesucht. Handelt es sich um eine Weiterleitungsseite, wird der verlinkte Artikel aufgerufen, und der Artikelname dieser Seite verwendet, um erneut eine Suche durchzuführen. Außerdem denkbar wäre eine Suche in WordNet, welche zusätzlich zu der Wikipedia-Suche noch sogenannte Synsets findet. Für die Suche „fridge“ wird auch hier das richtige Ergebnisset „electric refridgerator“ gefunden. Diese Suche ist auch bereits im Modul des Kontextanalysierers implementiert und wird verwendet, wenn weiterhin keine Ergebnisse vorliegen.

Sollte der Suchbegriff nun immer noch keinem Cyc-Konzept zugewiesen sein, ist es wahrscheinlich, dass kein Konzept in Cyc oder WordNet existiert, das eine natürlichsprachliche Nähe aufweist. Außerdem handelt es sich um einen Ausdruck, der eine Kurzform sein

könnte, die aber auch weder durch Wikipedia noch WordNet zurückgeführt werden konnte. Die Konzepte, welche dann eigentlich gebraucht würden, sind wahrscheinlich Dinge, die für diese Ontologien zu spezifisch sind. Es könnte sich auch um komplexe zusammengesetzte Wörter handeln, die praktisch im Sprachgebrauch eher selten verwendet werden. Fraglich ist deshalb auch, inwiefern mit weiteren Schritten noch Konzepte hierfür gefunden werden sollen. Vorerst werden diese Terme für die Vervollständigung verworfen. Die Implementierung des Verfahrens soll aber die Möglichkeit geben, später auf diese Information zurückzugreifen. Damit soll, falls keine Möglichkeit gefunden wird eine Aussage zu vervollständigen, dem Nutzer eine mögliche Fehlerquelle genannt werden. Dieser kann dann für diese Phrasen die korrekten Konzepte der Ontologie in einer Konfigurationsdatei angeben, sofern sie ihm bekannt sind.

5.3.2. Generieren und Prüfen von Kandidaten

Die Kandidaten sollen, wie in Abschnitt 4.2.1 beschrieben, über den situativen Kontext und eine Umgebungsontologie erzeugt werden.

Um keine Kandidaten zu überspringen, sollen in der Umgebungsontologie und dem Kontext alle Entitäten betrachtet werden. Damit aber die Menge der Ontologieanfragen die Laufzeit der Implementierung nicht zu stark beeinflusst, wird eine grobe Prüfung der Kandidaten durchgeführt. Diese wird im folgenden Unterkapitel beschrieben. Um Entitäten aus der Ontologie zu extrahieren, wird die im OWL-Format bestehende Datenbank durchsucht und alle gefundenen Entitätsknoten sowie ihre Eigenschaften kopiert. Diese Eigenschaften sollen im Folgenden verwendet werden, um die Eignung der Kandidaten zu prüfen. In der Umgebungsontologie sind etwa Eigenschaften wie „openable“ und „closeable“ für die Entität „fridge“ gespeichert.

Entitäten aus dem situativen Kontext werden generiert, indem das PARSE-Modul des Kontextanalysierers ausgeführt wird. Aus dem Modell, das während des Ablaufes entsteht und in dem PARSE-Graphen zur Aussage vorliegt, werden alle Entitäten extrahiert. Dazu werden alle Knoten mit dem Typ `contextEntity` betrachtet. In den Entitätsknoten ist der Name der Entität, wie er in WordNet benannt ist und lexikalische Eigenschaften - wie Synonyme - gespeichert. Für die Verarbeitung wird nur der Name der Entität extrahiert. Relationskanten des Kontextanalysierers werden vorerst nicht betrachtet. Diese drücken Teil-Ganzes-, Zustands- oder Gleichheitsbeziehungen aus. Aus diesen könnten in Zukunft noch weitere Informationen extrahiert werden, um noch eine weitere Wissensquelle an die Prüfung anzufügen. Besonders die Teil-Ganzesbeziehungen könnten verwendet werden, um die Intention des Benutzers noch genauer zu verstehen. Dazu müssten jedoch die Konzepte der Ontologie noch feiner bekannt sein. Der Kontextanalysierer bestimmt zum Beispiel, aus Informationen der Umgebungsontologie in einem Küchenszenario, dass ein „Kühlschrank“ das Teilkonzept „Kühlschranktür“ besitzt. Für die Anweisung „open“ wäre natürlich präziser, dass man nicht den Kühlschrank, sondern die Kühlschranktüre öffnet. Im Kontext einer Cyc-Inferenz ist es jedoch nicht für das Ergebnis ausschlaggebend, welche der beiden Varianten angefragt wird. Dort ist sogar nur das Konzept des Kühlschranks direkt gespeichert. Obwohl eine Relation in Cyc besteht, welche einem Kühlschrank sein Türkonzept zuweist, ist es unwahrscheinlich, dass dadurch die Güte der Inferenz verbessert wird. Gerade für Konzepte, welche in Cyc nicht in diesem detaillierten Ausmaß existieren, ist eine solche Vorgehensweise eher schädlich.

Jedes Konzept aus Ontologie und dem situativen Kontext wird über die Verfahren des letzten Kapitels auf ein Cyc-Konzept übertragen, bzw. verworfen, sollte dies nicht möglich sein. Dadurch kann es vorkommen, dass einige Konzepte nicht auf Cyc-Konzepte übertragen werden. Diese werden dann in einer geeigneten Liste gespeichert, um den Nutzer auf dieses Problem bei der Übertragung hinzuweisen. Er kann dann entscheiden, ob er in der Konfiguration eine Regel für dieses Szenario anlegt.

5.3.3. Grobe Prüfung der Kandidaten

Mit den gefundenen Ontologiekonstanten wird nun eine simple Überprüfung durchgeführt werden. Dafür wird der Argumenttyp der gesuchten Rolle und die Klasse des gefundenen Konzepts auf eine Oberklassenrelation geprüft.

Dafür wird in Cyc für das Beispielkonzept „Refridgerator“ und die Rolle „providerOfMotiveForce“ die folgende Anfrage gestellt.

```
(and
 (argIsa 1 providerOfMotiveForce ?ARG)
 (genls Refridgerator ?ARG)
)
```

Diese Aussage ist dann wahr, wenn das gesuchte Konzept eine Unterklasse des Argumenttyps des Handlungsprädikats ist.

Um die Effizienz zu erhöhen, wird die Anfrage aufgeteilt. Denn die Menge der Rollen ist gering, aber die Anzahl der Anfragen für jede dieser Rollen ist hoch. In einem ersten Schritt wird der Argumenttyp für das Prädikat mittels des `argIsa` Prädikats angefragt und zwischengespeichert. Für jeden Kandidat, muss dann nur noch die `genls` Relation geprüft werden.

Alle Kandidaten, mit denen diese Aussage nicht beweisen werden kann, werden für die Vervollständigung dieser Rolle nicht weiter betrachtet und entsprechend markiert.

Für alle anderen Kandidaten wird dann eine genaue Prüfung durchgeführt.

5.3.4. Genaue Prüfung der Kandidaten

Nachdem über die grobe Prüfung der Kandidaten einige, garantiert unpassenden Entitäten verworfen wurden, können in einem nächsten Schritt komplexere, Rechenintensivere Prüfmetriken angewandt werden. Dafür werden die Frame-Informationen verwendet, die in Abschnitt 5.2.1 angefragt wurden. In diesen Frames stehen mehrere Annahmen, die gelten sollen, wenn das zugehörige Prädikat die Handlung beschreibt. Diese werden aufgetrennt. Alle vorkommenden Variablen werden in zweistelligen Prädikaten gesucht. Diese haben immer eine Form, in welcher ein Argument die Handlung, und das andere Argument die Entität mit der richtigen Rolle ist. Das lässt sich anhand des Variablennamens erkennen: „?ACTION“ beschreibt das Handlungsprädikat, die zweite Variable hat den Namen einer semantischen Rolle.

Nun wird bestimmt, welche gefundenen semantischen Rollen die Variablen belegen könnten. Mit den Informationen aus dem letzten Kapitel kann dann die Anzahl der zu prüfenden Kandidaten beschränkt werden. Alle Entitäten, welche die jeweiligen Rollen aufgrund ihrer Klasse nicht sein können, werden auch nicht für die Anfrage im Satz verwendet, sondern direkt als falsch markiert.

Beispielhaft sieht ein Ablauf für das Wort „open“ wie folgt aus: Der Inhalt des angefragten Frames ist der Nachstehende:

?FRAME
(and (isa :ACTION OpeningSomething) (performedBy :ACTION :SUBJECT) (objectActedOn :ACTION :OBJECT))

Um zu bestimmen, ob der Kandidat sinnvoll ist, werden diese Sätze mit dem Aktions-Prädikat und Kandidaten instantiiert. Jede generierte Anfrage wird unabhängig von den anderen Anfragen durchgeführt. Für jede, von der Ontologie, bewiesene Aussage, wird gespeichert, dass der Kandidat das Prädikat mit dieser Aktion erfüllen kann. Das Prädikat kann jedoch nicht direkt abgefragt werden, da es sich beim Typen der Aktion um eine Instanz und keine Sammlung handelt. Um automatisch aus der Ontologie Instanzen der Sammlung zu generieren werden die Anfragen wie folgt umgebaut:

```
(#$relationAllExists #$objectActedOn ?X #$OrangeJuice)
```

Cyc wandelt diese Anfrage so um, dass anstelle von `OpeningSomething` Instanzen aus der Klasse für die Suche verwendet werden. Damit wird gleichzeitig noch ein sehr breites Spektrum an Wissen geprüft, das diese Anfrage bestätigen könnte.

Sollte für eine Rolle kein Kandidat verbleiben, so muss diese Rolle als leer und fehlend markiert werden. Dieser Fehler kann dann durch die Funktionen dieser Anwendung nicht mehr korrigiert werden. Anderen PARSE-Modulen, wie etwa einer Dialogeinheit, soll es dann möglich sein, auch mit diesen Informationen weiter zu arbeiten.

Wenn für eine Rolle mehrere Möglichkeiten verbleiben, gibt es verschiedene Möglichkeiten zu verfahren. Da nach der Prüfung jedes Ergebnis gleich geeignet ist, lässt sich kein bester Kandidat bestimmen. Trotzdem könnte eine der Möglichkeiten übernommen werden, um dann experimentell zu bestimmen, ob durch die Wahl des ersten Kandidaten eine gute Lösung produziert wird. Es wird jedoch riskiert, dass einige Aussagen mit nicht garantiert richtigen Entitäten vervollständigt werden. Um dieses Risiko zu minimieren, sollten noch weitere Möglichkeiten betrachtet werden, diese neu erstellten Hypothesen auf ihren Inhalt zu überprüfen. Die Cyc-Ontologie plant in kommenden Versionen der Ontologie ein Modul zu bereitzustellen, welches natürliche Sprache auf CycL-Ausdrücke überträgt. Diese Hypothesen, wie sie auch schon in [MWK+05] generiert wurden, könnten dann mit den Inhalten aus verschiedenen Mikrotheorien auf ihren Sinngehalt geprüft werden. Eventuell sind auch andere Wissensquellen - wie OMCS - verwendbar, um weitere Aussagen über den Inhalt von Aussagen zu treffen.

6. Implementierung

Im den folgenden Kapiteln werden einige wichtige Ausschnitte aus der Implementierung des PARSE-Agenten aufgeführt. Es wird erläutert, welche Funktionen sie erfüllen, und welche Entwurfsentscheidungen sie implementieren. Außerdem wird in Abschnitt 6.2 Beispielcode für die Verwendung des Agenten gezeigt.

6.1. Implementierte Klassen

Im Folgendenen steht eine Übersicht der wichtigsten Klassen und deren Funktionalität, sowie deren wichtigste Schnittstellen. Die Reihenfolge der aufgeführten Klassen richtet sich nach der Reihenfolge, in der sie während des Ablaufes des Agenten benötigt werden. Es werden zuerst die Klassen für die Vollständigkeitsüberprüfung vorgestellt und daraufhin die Klassen für die automatische Vervollständigung

6.1.1. CompletenessAgent

Diese Klasse bietet den Startpunkt für die Ausführung der Vollständigkeitsprüfung und der automatischen Vervollständigung. Sie implementiert die Schnittstelle `AbstractAgent`, welche die `init()` und `exec()` Methoden vorgibt.

Die Initialisierungsmethode erstellt eine Verbindung zu der Cyc-Ontologie. Damit diese Initialisierung funktioniert, muss der Server-Ort von Cyc bekannt sein. Dieser kann auf verschiedene Weisen gesetzt werden, es ist jedoch am einfachsten vor der Ausführung die Java-VM-Umgebungsvariable `cyc.session.server` auf die URL des Cyc-Servers zu setzten. Ist die Adresse nicht vorhanden, wird der Nutzer aufgefordert, den Serverpfad in einer GBO (Grafische Benutzeroberfläche) anzugeben.

Die Ausführung des Agenten kann nur dann erfolgen, wenn die folgenden Agenten bereits die Ausführung ihrer Tätigkeit abgeschlossen haben.

- Action Recognizer
- Semantic Role Labeler
- Context Analyzer

Sollte `exec()` zuvor aufgerufen werden, geschieht nichts.

Die Ergebnisse nach der Ausführung des Agenten stehen im (über die Methode `getGraph()` verfügbaren) Graph zur Verfügung.

6.1.2. Word und Utterance

In der Implementierung existieren die Helferklassen `Word` und `Utterance`. Diese extrahieren aus dem Eingabegraphen wichtige, häufig beim Ablauf benötigte Informationen und speichern diese in einer leicht zugänglichen Datenstruktur. Eine Instanz der `Word` Klasse beinhaltet dabei immer genau die Informationen, die zu einem Wort der Eingabe bekannt sind. Außerdem wird um eine spätere Übertragung der Ergebnisse auf den Graphen zu ermöglichen, in jedem `Word`-Objekt eine Referenz auf den zugehörigen Knoten des Eingabegraphs gehalten. Dazu gehören Name, POS-Tag, aber auch bereits gefundene Rollenmarkierungen und die Ergebnisse, die im Laufe der Verarbeitung der Aussage mit dem Agenten anfallen.

Eine Instanz der Klasse `Utterance` speichert eine Liste von Wörtern in Form von `Word`-Instanzen, um diese zu einer Aussage zusammenzufassen. Außerdem existiert eine statische Fabrikmethode, welche aus einem `ParseGraph` eine Menge von `Utterance`-Instanzen erzeugt. Diese werden als Array zurückgegeben. Für jede Teilaussage, die eine neue Handlung beschreibt, wird ein eigenes `Utterance`-Objekt erstellt. Folglich sind Handelnde aus anderen Teilaussagen aus der Aussage herausgelöst, eine fehlerhafte Zuweisung ist damit nicht mehr möglich. Denn jede Teilaussage wird abgetrennt vom Rest der Aussage behandelt, das bedeutet, der Agent wird mehrfach ausgeführt, bevor die Ergebnisse wieder übertragen werden.

6.1.3. CycQueryHelper

Die Klasse `CycQueryHelper` stellt wichtige Funktionalität zur Verfügung, welche verwendet wird, um Anfragen an Cyc einfacher zu gestalten. Häufige, ähnliche Anfragen, werden hier über verschiedene Methoden bereitgestellt. Zu diesen Methoden gehören die Folgenden:

Methodensignatur	Beschreibung
`List<Sentence> getFrames(KbTerm cycAction)`	Ermittelt für ein Wort seine Cyc-Frames (siehe Abschnitt 5.2.1)
`boolean genlsQuery(String lower, String upper)`	Ist wahr, wenn eine Unterklassenbeziehung von `lower` zu `upper` besteht.
`List<KbTerm> query(String queryString, String varBinding, int stepCutoff)`	Stellt eine Anfrage an die Inferenzmaschine. Rückgabe ist eine Liste von Belegungen der übergebenen Variable.
`boolean genlsPredQuery(String one, String two)`	`wahr`, wenn Prädikat `one` eine Verallgemeinerung von `two` ist

Außerdem ist diese Klasse für das Öffnen und Schließen der Verbindung zu Cyc zuständig. Zum Ende der Ausführung des Agenten wird die Verbindung zu Cyc in der `close()` Methode geschlossen.

6.1.4. CycConstantFinder

Die Klasse `CycConstantFinder` beinhaltet Methoden, welche die in Abschnitt 5.3.1 vorgestellten Entwurfsentscheidungen umsetzt. Dabei sucht sie für eine Eingabephrase eine geeignete Konstante in der Ontologie, die den Eingabestring beschreibt. Die Methodensignatur dieser Methode ist `KbTerm findConstant(List<Word> words)`. Sie erhält eine Liste von `Word`-Objekten, da ein Konzept auch aus mehreren Wörtern bestehen könnte (z.B.: „the orange juice“). Vorerst wird die Eingabe auf eine einzige Zeichenkette reduziert. Dazu werden alle Wörter, durch Leerzeichen getrennt, konkateniert und Artikel werden entfernt. Beim Erstellen werden nicht die Wörter aus der Eingabe verwendet, sondern die von dem Sprachfließband erstellten Lemmas.

Eine vom Nutzer bestimmbare Konfigurationsdatei wird beim ersten Zugriff auf die Methode geladen. Einträge dieser Liste werden in einer `Map` repräsentiert, die Zeichenketten als Schlüssel und Werte hat. Damit können zügig Einträge gesucht werden. Die `Map` wird so konfiguriert, dass eine Suche nach dem Schlüssel unabhängig von der Groß- und Kleinschreibung erfolgt. Für die regelbasierte Übertragung der Terme wird das gesuchte Wort als Schlüssel in der `Map` angefragt. Außerdem werden noch einige Alternativen generiert, die etwa keine Leerstellen besitzen, oder die Artikel nicht verwerfen. Dadurch kann der Nutzer in der Konfigurationsdatei Objekte auf verschiedene Weisen beschreibe und sie werden trotzdem gefunden.

Wie im Entwurf diskutiert, werden danach noch weitere Suchschritte angewendet, bis ein Ergebnis gefunden wird. Dazu wird zunächst direkt ein Konzept in der Ontologie gesucht. Dann werden die Beschriftungen des Erkenners für benannte Entitäten betrachtet, daraufhin werden Personalpronomen ersetzt. Letztlich werden Wikipedia-Artikel gesucht, um Kurzformen auf ihren Ursprung zurückzuführen. Wird durch alle diese Methoden keine Konstante in der Ontologie gefunden, wird `null` als Ergebnis zurückgegeben.

Eine Helfermethode `findConstants(Utterance utterance)` führt automatisch die Suche von Konzepten für eine Eingabeaussage durch. Sie trennt die vorkommenden Wörter auf, indem sie zusammenhängende Nominalphrasen an die oben beschriebene Methode übergibt. Außerdem trägt sie in den jeweiligen `Word`-Instanzen die gefundenen Konzepte oder `null` ein.

6.1.5. RoleMatcher

Der `RoleMatcher` implementiert die in Abschnitt 5.2.3 definierten Entwurfsentscheidungen. Zu diesen Anforderungen gehören:

- Implementierung eines Verfahrens zur Bestimmung von Textähnlichkeit
- Erkennen und Entfernen von ähnlichen Cyc-Rollen.
- Finden von fehlenden Rollen in Propbank, Frame- und VerbNet.
- Das Erstellen eines Matchings zwischen Cyc-Rollen und Rollen aus anderen Ontologien

Zusätzlich zu den genannten - im Weiteren auch noch genauer beschriebenen Anforderungen - werden vom `RoleMatcher` noch einige kleine Funktionen übernommen. Wie die oben genannten Anforderungen konkret implementiert wurden ist in den folgenden Kapiteln beschrieben.

In Cyc existieren über 500 verschiedene Rollen für verschiedene Verben. Deren Oberklassen sind jedoch oft sehr allgemein. Gerade das Cyc-Prädikat `#$actors` ist die Oberklasse jeder Rolle. Trotzdem liefert eine Anfrage zu Rollenlisten deren Rollen bei jeder Suche. Da einige diese Rollen aber keine inhaltliche Relevanz haben, werden sie direkt zu Beginn der Ausführung des `RoleMatcher` entfernt. Dazu werden aus der Liste von benötigten Rollen einige gestrichen.

Ebenso werden als Vorbereitung für das Suchen der Rollen die Namen der Handlung in VerbNet, FrameNet und PropBank gesucht. Diese werden in den `contextAction`-Knoten im Graphen und in den Markierungen des Aktionserkenners gesucht. Falls für bis zu zwei Ontologien kein Eintrag verfügbar ist, werden diese über Einträge in SemLink [Pal09] gesucht.

6.1.5.1. Textähnlichkeit

Um die Ähnlichkeit von zwei Texten zu bestimmen, wurde die Methode `int lesk(String one, String two)` implementiert. Diese Methode berechnet für zwei Texte einen Ähnlichkeitswert mittels des Lesk-Algorithmus. Dafür werden die Texte wortweise verglichen: Für jede vorkommende, maximal lange Wortsequenz, welche in beiden Texten vorkommt, wird das Quadrat der Länge auf den Ähnlichkeitswert addiert. Dadurch werden Texte, die lange übereinstimmende Sequenzen haben, besser bewertet als kurze Sequenzen.

6.1.5.2. Erkennen und Entfernen von ähnlichen Cyc-Rollen

Oftmals wurden, trotz reduzierter Suchtiefe der Anfragen, viele Rollen gefunden. Viele dieser Rollen glichen sich sehr. Um diese ähnlichen Rollen zu verschmelzen, wurden zwei Methoden zum Finden dieser implementiert.

Zuerst wurde eine Methode implementiert, die für eine Liste von Cyc-Prädikaten bestimmt, welche Paare aus dieser Liste gleich sind. Dazu wurde für jedes Paar von Rollen zwei Anfragen an Cyc gestellt, um die Relation zu überprüfen. Dabei wird das Cyc-Prädikat `#$genls` für beide Permutationen des Paares geprüft. Ist eine der beiden Anfragen wahr, wird die speziellere Rolle übernommen und die allgemeinere verworfen.

Die Methode `int[][] calcSimilarityMatrix(String[] glosses)` liefert eine Matrix, die an Stelle `i,j` die Ähnlichkeit der i-ten und j-ten Kommentare speichert. Dann wird für jede Zeile der Matrix geprüft, ob der Eintrag einen bestimmten Schwellwert übersteigt. Dieser wurde experimentell bestimmt, indem die Liste von einigen Prädikaten manuell verglichen wurde und der Parameter solange geändert wurde, bis die Ergebnisse zufriedenstellend, also ohne gleiche Rollen, waren. Rollenpaare, die über diesem Wert lagen, wurden verschmolzen, d.h. die Rolle mit weniger hinterlegten Informationen wurde verworfen.

6.1.5.3. Finden von fehlenden Rollen in Propbank, Frame- und VerbNet

Da eine Rollenbeschriftung für Propbank, Frame- und VerbNet bereits durch den SRL verfügbar ist, ist es am einfachsten, die Differenz dort zu berechnen und die fehlenden Rollen im Anschluss auf Cyc zu übertragen. Dazu wurden die Verb-Konzepte über die Helferklassen `PropBankEntry`, `VerbNetEntry` und `FrameNetEntry` geladen und ihre Rollenlisten ausgelesen. Diese Klassen lesen lediglich eine Text-Datei ein und zerteilen die Eingabe und stellen sie in einer Objektstruktur zur Verfügung. Außerdem bieten sie eine lineare Suche, um zügig einen Eintrag in den teils langen Lexikon-Listen zu finden.

Das Finden von fehlenden Rollen wird in der Methode `findMissingVnPbRoles()` und ähnlichen Methoden für PropBank und FrameNet übernommen. Dabei wird die Differenz der benötigten und der vorhandenen Rollen auf einer Liste gebildet und diese Liste zurückgegeben.

6.1.5.4. Erstellen einer Übertragungsliste für Cyc-Rollen

Aus den erstellten Listen von Rollen aus den anderen Ontologien, wird nun eine Übertragungsliste für Cyc-Rollen erstellt. In dieser stehen die Rollennamen aus Frame- und VerbNet und der Name der Cyc-Rolle, welche inhaltlich am ähnlichsten ist. In der Methode `generateMatching()` wird für jede Cyc-Rolle ihre Ähnlichkeit zu allen Rollen in Frame- und VerbNet berechnet. Bei der Implementierung wurden zuerst ausschließlich VerbNet-Labels verwendet, um die Ähnlichkeiten zu berechnen. Da dieser Ansatz aber aufgrund der sehr kurzen Beschreibungen der Rollen nicht sonderlich ergiebig war, wurden noch die Beschreibungen der Rollen aus FrameNet hinzugezogen. Für jede VerbNet-Rolle wird dabei die FrameNet-Rolle aus dem SemLink-Matching angefragt und aus dem passenden

`FrameNetEntry`-Objekt wird dann der Kommentar zu dieser Rolle extrahiert. Mittels der Textähnlichkeitsmethode (siehe Abschnitt 6.1.5.1) werden dann für den Cyc-Kommentar und den Verb- und FrameNet Kommentar die zwei Ähnlichkeiten errechnet. Die Summe aus beiden Werten ergibt den finalen Ähnlichkeitswert. Letztlich wird die maximale Ähnlichkeit für die Cyc-Rolle berechnet und die zugehörige Rolle auf die jeweiligen `Word`-Objekte übertragen. Die Liste von verfügbaren Rollen wird dabei nicht verändert, da es möglich sein sollte mehreren Entitäten verschiedene Rollen zuzuweisen. Dies ist zum Beispiel im Satz „Armar grabs the popcorn“ wichtig, bei der ARMAR der Handelnde, aber auch der Empfänger ist.

Das erstellte Matching wird direkt auf die Aussage in dem `Utterance`-Objekt übertragen. Außerdem ist eine Abfragemethoden (engl.: „getter“) verfügbar, die eine Liste der fehlenden Rollen als Cyc-Termen zurückgibt.

6.1.6. OntologyConnector

Die Klasse `OntologyConnector` extrahiert aus einer Umgebungsontologie und dem Kontextmodell des Kontextanalysierers Entitäten, um mögliche Vervollständigungs-Kandidaten zu generieren.

Dazu liest sie eine Ontologie im OWL-Format[1] ein und sucht dort alle Entitätsknoten und gibt eine Liste derer Namen zurück. Für jeden dieser Namen wird mittels des `CycConstantFinder`(siehe Abschnitt 6.1.4) eine Konstante in Cyc gesucht.

Ebenso werden in dieser Klasse aus dem `ParseGraph` Knoten des Kontextanalysierers extrahiert, welche den Typen *contextEntity* besitzen. Jeder dieser Knoten enthält eine Entität aus dem Kontext der Situation. Da die Kontextverarbeitung vor dem Ablauf dieses Agenten vollständig abgeschlossen ist, stehen Entitäten aus allen vorangehenden Satzteilen zur Verfügung. Auch deren Namen werden als Liste zurückgegeben, für deren Einträge analog Cyc-Konstanten gesucht werden.

6.1.7. AutoCompleter

Die `AutoCompleter` Klasse implementiert die Entwurfsentscheidungen aus Abschnitt 5.2. Eine `AutoCompleter`-Instanz erhält eine `Utterance`-Instanz und berechnet mit dieser und einer Liste von Cyc-Termen mögliche automatische Vervollständigungen.

Dazu benötigt er Kandidaten, die aus dem situativen Kontext und der Umgebungsontologie stammen sollen. Diese können nach einer Initialisierung über eine Abfragemethode aus der Klasse `OntologyConnector` geholt werden.

Mit den gefundenen Konzepten wird dann die grobe Prüfung (siehe Abschnitt 5.3.3) der Kandidaten durchgeführt. Der Quellcode der zuständigen Methode ist der Folgende:

```
private void generateGenlsRelations() {
  generateArgIsas();
  genlsMatches = new boolean[missingRoles.size()][candidates.size()];

  for (int i = 0; i < missingRoles.size(); i++) {
        int j = 0;
        Iterator<KbTerm> it = candidates.iterator();
        while (it.hasNext()) {
                KbTerm currentTerm = it.next();
                boolean genlsResult = CycConnector.getInstance().
                    genlsQuery(currentTerm.toString(),
                    missingRoleArgumentTwo.get(i).cyclify());
```

[1] https://www.w3.org/TR/2012/REC-owl2-overview-20121211/

```
                genlsMatches[i][j] = genlsResult;
                j++;
            }
        }
    }
```

Die in Zeile zwei aufgerufene Methode generiert dabei über Anfragen an die Ontologie den Argumenttyp der Rollen, die dann in einem Feld gespeichert werden. Sie werden nicht als Rückgabeparameter übergeben, da sie eventuell für weitere Prüfmetriken noch relevant sein könnten.

In einer weiteren Methode werden die Kandidaten nach den Überlegungen in Abschnitt 5.3.4 mit Inhalten aus Cyc geprüft. Dazu wird für jeden Kandidaten und jede Rolle, die in `genlsResult[][]` noch nicht ausgeschlossen wurden die folgende Anfrage an Cyc gestellt:

```
(and
 (genls ?X cycAction)
 (relationAllExists currentRole ?X candidate)
)
```

Dabei wird `cycAction` durch das Prädikat der Handlung, `currentRole` durch die aktuell betrachtete Rolle und `candidate` durch den aktuellen Kandidat für die Vervollständigung ersetzt. Die Anfrage unterscheidet sich minimal von der im Entwurf vorgeschlagenen. Diese Änderung wurde vorgenommen, da die Inhalte der Prädikate direkt oft nicht in Cyc verfügbar waren. Eine Anfrage der Form `(objectActedOn AcquiringAnObject OrangeJuice)` ist so explizit in keiner Mikrotheorie aufgeführt. Alle Lösungen der neuen Anfrage, sind Handlungen, die Unterklassen der gesuchten Handlung sind, in denen der Kandidat die richtige Rolle erfüllt. Das bedeutet anstatt zu Prüfen, ob der Kandidat die Rolle direkt in der Handlung erfüllt, wird gesucht, in welchen spezifischeren Handlungen diese Rolle von dem Kandidaten getragen wird. Da Cyc explizit nach einer Belegung gefragt wird, ist die Antwort nicht wahr oder falsch, sondern eine Liste von Handlungen. Die Anzahl der Antworten wird dann für die Bewertung verwendet. Je mehr Antworten gefunden werden, desto besser wird der Kandidat bewertet, weil der Kandidat in vielen Kontexten richtig ist. Dadurch soll die Wahrscheinlichkeit eines richtigen Treffers erhöht werden. Der Code, der diese Funktion erfüllt ist im Folgenden aufgeführt:

```
private List<List<Pair<Integer, KbTerm>>> checkCandidatesWithCycRules
    () {
        List<List<Pair<Integer, KbTerm>>> results = new ArrayList<>();
        for (int i = 0; i < missingRoles.size(); i++) {
                List<Pair<Integer, KbTerm>> currentResult = new
                    ArrayList<>();
                results.add(currentResult);
                KbTerm currentRole = missingRoles.get(i);
                for (int j = 0; j < candidates.size(); j++) {
                        if (!genlsMatches[i][j])
                        continue; // don't waste computation time on
                            useless facts
                        KbTerm candidate = candidates.get(j);
                        String querySentence = "(and␣(genls␣?X␣" +
                            cycAction + ")␣(#$relationAllExists␣" +
                            currentRole + "␣?X␣" + candidate.toString()
                             + "))";
                        int answerCount = 0;
                        try (Query query = Query.get(querySentence)) {
                                answerCount = query.getAnswerCount();
                        } catch (QueryConstructionException e) {
                                //count should stay 0 on error
```

```
                        } finally {
                        //frees up allocated query ressources
                        }
                        currentResult.add(new Pair<Integer, KbTerm>(
                            answerCount, candidate));
                }
                currentResult.sort((p, q) -> q.first.compareTo(p.first)
                    ); //sort by number of results
        }
        return results;
}
```

6.2. Nutzung des Agenten

In diesem Kapitel findet sich ein Beispielablauf, in welchem der Agent verwendet wird, um eine Aussage zu untersuchen.

```
String transcription = "Get␣me␣some␣orange␣juice";

//create required agents
PrePipelineData ppd = new PrePipelineData();
ShallowNLP snlp = new ShallowNLP();
GraphBuilder graphBuilder = new GraphBuilder();
ActionRecognizer ar = new ActionRecognizer();
NERTagger ner = new NERTagger();
ContextAnalyzer ca = new ContextAnalyzer();
SRLabeler srl = new SRLabeler();

//set main hypothesis
ppd.setMainHypothesis(StringToHypothesis.stringToMainHypothesis(
    hypothesis, true));

//initialize required agents
snlp.init();
srl.init();
graphBuilder.init();
ca.init();
ar.init();

//run required agents
snlp.exec(ppd);
srl.exec(ppd);
graphBuilder.exec(ppd);

ar.setGraph(ppd.getGraph());
ar.exec();
ca.setGraph(ppd.getGraph());
ca.exec();

//run completenessAgent
CompletenessAgent agent = new CompletenessAgent();
agent.setGraph(ppd.getGraph());
agent.init();
agent.exec();
```

Nach der Ausführung von diesem Beispielcode werden in den Ergebnisgraphen neue Knoten und Kanten eingefügt. Für jede unvollständige Rolle wird ein neuer Knoten im Graph erstellt, welcher Informationen über die Cyc-Rolle und das Prädikat der Handlung tragen.

Konnte die Rolle vervollständigt werden, so enthält sie zusätzlich eine Liste von Kandidaten, die auch für die Rolle möglich gewesen wären. Zusätzlich besitzen diese Knoten ein Attribut mit dem Namen der Entität, die die Rolle vervollständigt. Zwischen dem Wort, das die Handlung beschreibt und den Kanten werden - analog zum Aktionserkenner - Kanten mit den Rollennamen eingefügt.

7. Evaluation

In diesem Kapitel wird die Evaluation des verwendeten Ansatzes vorgestellt. Dazu wird der erstellte Agent mit Aussagen aus einem Textkorpus ausgeführt und seine Ergebnisse mit der eigentlich korrekten Lösung verglichen. Daraufhin wird die Güte der einzelnen Teilschritte, Problemursachen, sowie mögliche Verbesserungen diskutiert.

7.1. Korpus

Die Grundlage der Eingabetexte bietet der Sprachkorpus, der in [Gün15] erstellt wurde. Er besteht aus drei verschiedenen Szenarien, in denen Sprecher einem Roboter erklären sollen, wie er eine Handlung ausführen soll. Diese drei Szenarien finden in einem Küchen-Umfeld statt. Damit eine ausreichen große Variation bei den Eingaben besteht, wurden diese Szenarien von verschiedenen Probanden gesprochen. Insgesamt 22 Sprecher haben solche Aufnahmen erstellt, die dann von einem Spracherkennungssystem auf eine Textrepräsentation übertragen wurde. Ebenso wurde für jede Aufnahme eine händische Transkription erstellt und die Texte mit einem Etikettierer (engl. tagger) mit „Part-of-speech“ Markierungen versehen.

Um diese Arbeit zu evaluieren werden Aussagen benötigt, in welchen Unvollständigkeiten vorkommen. In den gewählten Szenarien werden fast immer Anweisungen in einem, für den Sprecher klar definierten, Umfeld gegeben. Das bedeutet es ist möglich, dass Sprecher sich auf Umgebungswissen stützen. Unter diesen Bedingungen werden implizite Wissensannahmen sprachlich nicht genannt. Folglich lassen sich die Aussagen aus dem Korpus verwenden. Zusätzlich wurden noch einige Aussage hinzugefügt, die für Tests während der Implementierung verwendet wurden. Im folgenden Textblock ist eine Anweisung aus dem Testkorpus:

> Armar take the orange juice from the fridge and put it on the table and then take the green cup after you took the orange juice from the fridge close it please

Diese Anweisung besteht aus vielen Teilanweisungen. Während viele von diesen Teilaussagen vollständig sind, fehlt in dem Teil „and then take the green cup“ der Ort, an dem sich diese befinden. Um die weiteren verwendeten Aussagen ebenfalls für die Evaluierung nutzen zu können, wird der PARSE-Agent zur Auflösung von Korreferenzen nicht verwendet. Dadurch generiert jeder Satz mit Korreferenzen eine unvollständige Aussage, für die zusätzlich noch durch die Auflösung der Referenz die Musterlösung gefunden werden

kann. In dem Korpus werden jedoch sonst keine Angaben zur Vollständigkeit von Aussagen gemacht. Aus diesem Grund wird für eine Untermenge des Korpus eine Musterlösung erstellt. In dieser sind die folgenden Informationen enthalten:

- Cyc-Prädikat der (Teil-)Handlung
- Cyc-Rollen der (Teil-)Handlung
- Information über die Vollständigkeit der Aussage (ja/nein)
- Beschriftung der Entitäten mit Cyc-Rollen
- Eine Liste von unbesetzten Rollen
- Vervollständigungen der fehlenden Rollen, wenn dies möglich ist

Die Musterlösung wird mit Cyc-Konstanten erstellt, da die Lösung des Agenten (welche auch aus Cyc-Konstanten besteht) dadurch leichter vergleichbar ist.

Insgesamt wurden 20 Aussagen, bzw. deren 75 Teilaussagen, aus dem Textkorpus auf diese Weise annotiert. In komplexen Anweisungen wurde jede Teilaussage unabhängig von der restlichen Aussage annotiert. Insbesondere die Korreferenzen wurden dann als unvollständig markiert. Für die oben genannte Aussage sieht die erstellte Beschriftung wie folgt aus:

> and put it on the table
> `pred:PuttingSomethingSomewhere`
> `roles:objectTranslating=Cup, missing`
> `roles:doneBy=Armar, missing`
> `roles:toLocation=the table`
> `context:armar take the cup and put it on the table`

7.2. Evaluationsmetriken

Die Ergebnisse des Agenten und der Teilschritte werden mit den für die Sprachverarbeitung typischen Metriken *Präzision* (engl. precision), *Ausbeute* `engl. recall` und dem `F1-Maß` (engl. F1-Score) bewertet. Diese werden verwendet, um die Einschätzung von Systemen zu bewerten, deren Bewertungen entweder positiv oder negativ sein können. Dazu müssen die folgenden Begriffe bekannt sein.

- richtig positiv (tp): Bezeichnet ein Ergebnis, das als positiv markiert wurde, und in der Musterlösung als positiv markiert ist.
- richtig negativ (tn): Bezeichnet ein Ergebnis, das als negativ markiert wurde, und in der Musterlösung als negativ markiert ist.
- falsch positiv (fp): Bezeichnet ein Ergebnis, das als positiv markiert wurde, aber in Wirklichkeit negativ ist.
- falsch negativ (fn): Bezeichnet ein Ergebnis, das als negativ markiert wurde, aber in Wirklichkeit positiv ist.

Mit diesen lassen sich Präzision, Ausbeute und F1 wie folgt berechnen:

$$\text{Präzision} : \frac{\text{richtig positiv}}{\text{richtig positiv} + \text{falsch positiv}} = \frac{tp}{tp + fp}$$

Die Präzision gibt an, wie groß der Anteil der als positiv markierten Ergebnisse tatsächlich positiv war. Eine hohe Präzision bedeutet, dass viele der als positiv markierten Ergebnisse

auch in der Musterlösung positiv markiert ist. Eine geringe Präzision wiederum bedeutet, dass positiv markierte Ergebnisse nur mit einer geringen Wahrscheinlichkeit tatsächlich korrekt bewertet sind.

$$\text{Ausbeute} : \frac{\text{richtig positiv}}{\text{richtig positiv} + \text{falsch negativ}} = \frac{tp}{tp + fn}$$

Die Ausbeute gibt den Prozentsatz der überhaupt gefundenen Ergebnisse an. Eine geringe Ausbeute impliziert, dass aus der Menge der auffindbaren Ergebnisse nur wenige tatsächlich markiert wurden. Eine hohe Ausbeute bedeutet, dass die Menge der positiv markierten wahrscheinlich Ergebnisse vollständig ist.

$$\text{F1} : 2 * \frac{precision * recall}{precision + recall}$$

Das F1-Maß ist das harmonische Mittel zwischen Präzision und Ausbeute. Es bietet eine Kennzahl zur Bewertung der Genauigkeit, die Ausbeute und Präzision beachtet.

7.3. Evaluation des Vollständigkeitsprüfers

Um die Ergebnisse des Vollständigkeitsprüfers zu bewerten wird die händische Liste der fehlenden Rollen mit der Liste von fehlenden Rollen des Agenten verglichen.

Da diese aber von der richtigen Wahl eines Prädikates in Cyc abhängen wird erst noch die Güte der Disambiguierung der Prädikate untersucht.

7.3.1. Evaluation der Disambiguierung

Für jeden bearbeiteten Satz, muss der Agent für jede darin vorkommende Teilaussage ein Prädikat bestimmen, welches die Handlung in dieser möglichst gut beschriebt. Die Implementierung des Agenten setzt dabei auf die Wahl der wahrscheinlichsten Bedeutung (vgl. Abschnitt 5.2.1). In den händisch erstellten Beschriftungen für die Eingaben wurde für jeden Teil der Aussage eine passende Handlung in Cyc gesucht.

Insgesamt wurden für 97.8% der Teilaussagen korrekte Prädikate ausgewählt. Es wurde aber nicht für jeden Satz überhaupt ein Prädikat gefunden. Nur in 65.3% der Fälle konnte tatsächlich ein Cyc-Prädikat gefunden werden. Diese erstaunlich gute Präzision sollte aber nicht in jeder Domäne erwartet werden: Die verwendeten Anweisungen aus dem Korpus behandeln alle das gleiche Thema aus dem selben Anwendungsfall. Darin wird einem Roboter erklärt, wie er den Ablauf von verschiedenen Aufgaben durchführen soll. Viele verschiedene Aufnahmen dieser Anweisungen enthalten ähnliche Teilschritte. Besonders Phrasen wie „put the cup somehere“ oder „take the glass“ sind dabei häufig vorkommende Bausteine. Die Anzahl der verwendeten Grundbausteine ist in den Anweisungen - auch aufgrund der zu bewältigenden Aufgabe - relativ klein. Zusätzlich trägt zu diesem Phänomen noch die Sprachkenntnis der Sprecher bei. Unter den 22 Probanden, welche die Aufnahmen sprechen, befand sich nur ein englischer Muttersprachler. Der Wortschatz dieser ist denkbar kleiner als bei Muttersprachlern, weshalb vermutlich simple Wörter anstelle von komplexen Synonymen oder diffizilen Prozesswörtern verwendet werden.

Ebenso ist denkbar, dass die Sprecher eine „unnatürliche“, simple Art der Kommunikation wählten, da sie wussten, dass die Aufnahmen von einem Computer verstanden werden sollen.

Das würde auch die hohe Genauigkeit der Disambiguierung, im Vergleich zu der Wahl der häufigsten Bedeutung in anderen Sprachverarbeitungssystemen, erklären. Die Wahl

der ersten Bedeutung des Agenten spiegelt dann nämlich genau die Wahl des simpelsten Wortes des Sprechers wieder. Viele der Aussagen sind relativ einfach und präzise formuliert. Die Aussage „place it next to the green cup“ aus dem Korpus hätte auch als „leave it next to the green cup“ formuliert werden können. Eine Disambiguierung aufgrund der häufigsten Bedeutung hätte `go away from a place` - ein falsches Ergebnis - ergeben. Die Fälle in denen eine Disambiguierung ein falsches Ergebnis liefert, sind Fälle, in denen ein falsches Verb aus dem Satz ausgewählt wurde. Im Satz „Make sure to close the cupboard“ wurde zum Beispiel „make“ als Prädikat gewählt und somit ein falsches Ergebnis produziert.

7.3.2. Evaluierung der Vollständigkeitsprüfung

Aufgrund der Wahl der Prädikate wurden im nächsten Schritt die benötigten Rollen gesucht, und die Vollständigkeit dieser überprüft. Die Rollen für die Musterlösung wurden gewählt, indem manuell ein passendes Prädikat für die Handlung gewählt wurde. Eine Ontologieanfrage ermittelt dann die Rollen für die Handlung. Diese werden dann händisch reduziert, indem gleiche Rollen vereint, oder zu allgemeine Rollen verworfen werden. Dann wird für jede Rolle entschieden ob sie vorhanden, oder nicht vorhanden ist. Zusätzlich wurde für jede fehlende Rolle eine Liste von äquivalenten Rollen erstellt. Mittels diesen sollen unterschiedliche Rollennamen, welche aber die selbe semantische Rolle tragen nicht als falsche Lösung bewertet werden. Dadurch können nicht in der Musterlösung vorkommenden Rollen in diesen Listen gesucht werden. Insbesondere bei Rollennamen wie `doneBy` und `performedBy` gibt es keine definitiv bessere Rolle. Gerade in Cyc sind die mit diesen Rollen verbundenen Wissensannahmen ähnlich, was auch auf der Vererbungsrelation zwischen ihnen basiert. Inhaltlich ergibt sich für die Rolle keine Änderung, es ist also denkbar, dass mehrere richtige Lösungen für eine Rolle existieren können.

Mit den Beschriftungen einer Rolle als positiv (fehlt) oder negativ (fehlt nicht) lassen sich falsch positive, wahr positive, falsch negative und wahr negative Ergebnisse zählen. Die Ergebnisse dieser Zählung finden sich in der folgender Tabelle.

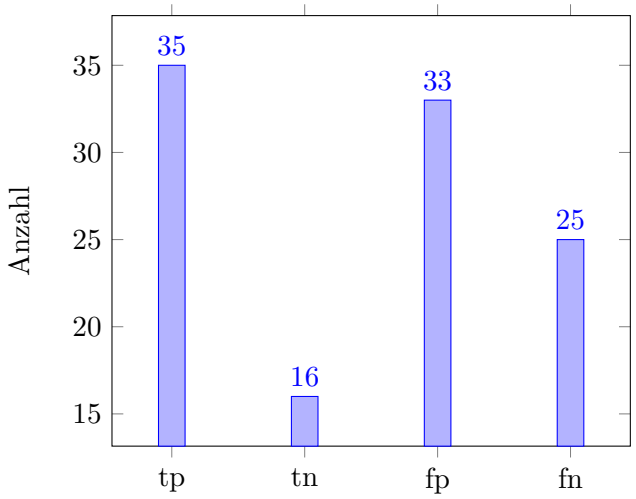

Abbildung 7.1.: Evaluationsergebnisse der Vollständigkeitsprüfung
true positive (tp), true negative (tn), false positive (fn), false negative (fn)

Aus diesen Werten lassen sich die folgenden Zahlen berechnen:

$$\text{Präzision} = \frac{35}{35 + 33} \approx 0.51 \tag{7.1}$$

$$\text{Ausbeute} = \frac{35}{35+25} \approx 0.58$$

$$\text{F1-Maß} = 2 \cdot \frac{\frac{35}{35+33} \cdot \frac{35}{35+25}}{\frac{35}{35+33} + \frac{35}{35+25}} \approx 0.55$$

In Sprache ausgedrückt bedeuten diese Werte: Von den gefundenen, unvollständigen Rollen, waren etwa 51 Prozent korrekt als unvollständig klassifiziert. Außerdem wurden von allen möglichen fehlenden Rollen etwa 59 Prozent gefunden. Diese relativ niedrige Präzision ist auf eine hohe Zahl, an fälschlich fehlenden markierter Rollen zurückzuführen. Diese werden von dem Agenten aus verschiedenen Gründen fälschlicherweise als fehlend markiert. Dazu muss noch genannt werden, dass vom Agenten alle nicht in der Aussage gefundenen Rollen als fehlend markiert werden. Das führt in den im Folgenden aufgeführten Fällen zu Problemen.

Eine Fehlerquelle ist eine schlechte Übertragung der Rollen von anderen Wissensquellen auf Cyc. Diese basiert auf den Beschriftungen des Rollenerkenners. Allen Nominalphrasen, welchen dieser keine Bedeutung zuweist, werden auch von diesem Agenten nicht weiter betrachtet. Kommt beispielsweise in einer Aussage die Nominalphrase „the fridge“ vor, welche aber vom Rollenerkenner nicht als solche beschriftet wird, kann der Agent dazu passende Rolle übertragen. Folglich wird eine Rolle als fehlend markiert, obwohl sie eigentlich in der Aussage vorkam, was zu einem *falsch positiven* Ergebnis führt.

Außerdem gab es Sätze, in denen die Kommentare einzelner Cyc-Rollen besser als alle anderen Kommentare der restlichen Rollen waren. Dadurch wurde für alle im Satz vorkommenden Entitäten die selbe Cyc-Rolle verwendet, und folglich jede weiter als fehlend markiert. Dies führt wiederum zu einer hohen Zahl an *falsch positiven* Ergebnissen. Ursprünglich wurde im Entwurf die Entscheidung getroffen, eine Mehrfachzuweisung von Cyc-Rollen zuzulassen. Damit sollte es möglich sein, Rollen wie `transportees` oder `preActors` mehreren Entitäten zuzuweisen. Semantische Rollen, welche in einer Aussage von mehreren Entitäten getragen werden scheinen jedoch relativ selten vorzukommen. Eine mögliche Konsequenz wäre es, diese Regeln nicht mehr zu verwenden, und eine Rolle nur noch ein mal pro Aussage zu vergeben, um die Präzision des Agenten zu erhöhen.

Ein weiteres Problem wird durch einige Prädikate in Cyc verursacht. Das Prädikat `FillingProcess` hat keine eigenen semantischen Rolleninformationen. Eine Anfrage via Inferenz liefert nur die Rollen, welche in den Oberklassen des Prädikats definiert sind. Diese sind jedoch sehr generisch und teilweise für die Aktion sogar unpassend, was dazu führt, dass die Liste von Rollen nun Namen wie `objectTranslating`, `motionPathway-Exact` und `toLocation` enthält. Solche semantische Rollen sind aber für die Aktion `FillingProcess` nicht benötigt. Eventuell erkennt der Agent im Satz „Fill the cup“ noch, dass ein `objectTranslating` die Tasse sein könnte. Auf der Suche nach den korrekten Entitäten für den Bewegungspfad und Zielort gehen dann spätestens die möglichen Entitäten aus. Der Agent markiert diese dann als fehlend, was wiederum zu *falsch positiv* markierten Aussagen führt. Eine allgemeine Lösung für dieses Problem ist schwierig, da es direkt auf Einträgen in der Wissensbasis von Cyc basiert. Grundsätzlich lassen sich die betroffenen Wissenseinträge anpassen und richtige, sinnvolle Rollen könnten für das Prädikat gesetzt werden. Dies ist aber immer nur dann möglich, wenn bereits aufgefallen ist, das ein solches problematisches Prädikat vorliegt.

Ebenso lässt sich die schlechte Präzision auf die Menge der Cyc-Rollen zurückführen. Für viele Prädikate sind Rollen gespeichert, welche *eventuell* sinnvoll sein könnten. Für jede Bewegung ist `motionPathway-Exact`, `fromLocation` und `toLocation` in der Liste der semantischen Rollen. Auch ein Versuch diese Rollen zu reduzieren funktioniert nicht, da

die drei Rollen keine Verallgemeinerungen voneinander sind, oder in ihren Kommentaren ausreichend übereinstimmen. Dies wäre aber auch nicht sinnvoll, da diese Rollen an sich verschiedene Konzepte beschreiben. Das Problem ist jedoch, dass nicht jede Aussage zwangsläufig die Information `fromLocation` und `motionPathway-Exact` benötigt. Bei der Aktion, die in dem Satz „robot go to the dishwasher“ beschrieben wird, reicht es dem Roboter zu wissen, dass er sich zu der Spülmaschine bewegen soll. Diese Überspezifikation der benötigten Informationen führt wieder dazu, dass Rollen, welche eigentlich nicht benötigt würden, als fehlend markiert werden. Um Probleme aus dieser Kategorie zu lösen könnte zum Beispiel für ganze Prädikats-Sammlungen, wie etwa `Movement-TranslationEvent` (Oberklasse von `GoingSomewhere`), eine Indexliste erstellt werden, die bestimmte Rollen aus diesen verbieten.

7.4. Evaluation der automatischen Vervollständigung

Die zwei wichtigen Bausteine für die Vervollständigung sind das Finden der richtigen Konzepte aus dem Kontext und der Umgebungsontologie und die Bewertung der Eignung für jeden dieser Kandidaten. Diese beiden Teilschritte werden nun im folgenden für sich betrachtet evaluiert.

7.4.1. Evaluation des Konstantenfinders

Der Konstantenfinder (Klasse `OntologyConnector`) überträgt natürlichsprachliche Konzepte auf Cyc-Terme. Die Korrektheit dieser Übertragung ist für das Vervollständigen von Rollen wichtig. Ist beispielsweise das einzig mögliche Konzept für eine Rolle nicht korrekt übertragen worden, so kann sie nicht mehr vervollständigt werden. Ebenso ist dies problematisch, wenn aus einer - in Sprache - unpassenden Rolle durch die Übertragung eine passende Cyc-Konstante gefunden wird.

Um den Ansatz zu evaluieren, wurde eine Liste von Konzepten aus der Umgebungsontologie extrahiert, und einige beim Testen produzierte Beispiele aus dem Kontext gesammelt.

Von den 64 betrachteten Einträgen konnten neun Stück nicht übertragen werden. In ca. 86% der Fälle wurde also eine Konstante gefunden. Von den verbliebenen 35 einzigartigen Suchbegriffen wurden 77,14% sinnvoll auf ein Konzept der Ontologie übertragen.

Für eine relativ simple Disambiguierung erscheint dieses Ergebnis befriedigend. Die auftretenden Fehler lassen sich auf den `CycConstantFinder` zurückführen, der eine Suche in der Programmierschnittstelle von Cyc verwendet um Konstanten aus Sprache zu generieren. Vermutlich ist bei der Wahl eines Terms über den `CycConstantFinder` oftmals eine falsche Konstante selektiert worden. An dieser Stelle wird der erste, namentlich am besten passende, Treffer als Konzept übernommen. Komplexere Disambiguierungsverfahren, zum Beispiel über den Satzkontext könnten durchgeführt werden um die Präzision zu erhöhen.

An anderen Stellen erlaubt die Cyc-Ontologie keine Ambiguität, da die Begriffe eindeutige Namen tragen. Die einzige andere Möglichkeit, die noch zu falschen Konzepten führt ist, wenn der einzige Eintrag in Cyc eine andere Bedeutung besitzt als die gewünschte. Dann ist es nur möglich die Disambiguierung zu verbessern, indem diese neuen gewünschten Bedeutungen manuell zu Cyc hinzugefügt werden.

7.4.2. Evaluation der Rollenvervollständigung

Zur Evaluation des zweiten Ausführungsteils des Agenten wird betrachtet, wie gut die erstellten Kandidaten für die unvollständigen Rollen sind. Auch hierbei muss wieder das Ergebnisformat berücksichtigt werden, um eine sinnvolle Bewertung zu erstellen.

Der Agent verwendet für das Finden der Kandidaten die Liste der unvollständigen Rollen aus dem letzten Teilschritt. Sind diese bereits fehlerhaft, so würde der Vergleich mit einer Musterlösung deutlich schlechter ausfallen, als bei einer Verarbeitung mit der Musterlösung. Damit der Ansatz unabhängig von der vorherigen Verarbeitung erfolgen kann, werden dem Vervollständiger die fehlenden Rollen aus der Musterlösung vorgegeben. So können alle Ursachen für Fehler direkt beim Agenten, bzw. dem konkreten verwendeten Ansatz, angenommen werden und nicht mehr bei vorangegangenen Schritten.

Als Bewertungsmetrik wurden für jede vervollständigte Rolle geprüft, ob die Vervollständigung sinnvoll ist oder nicht. Die Evaluation wurde dabei auf verschiedene Rollen aufgeteilt, da die Prüfung auf Annahmen zu diesen basiert und auch pro Rolle unterschiedlich ausfällt. Somit soll bestimmt werden, ob es Rollen gibt, die besonders gut vervollständigt werden können, und Rollen identifiziert werden, für die eine Vervollständigung schwer fehlt.

Unter der 79 unvollständigen Rollen befand sich 59 mal die Rolle „doneBy“. Diese Rolle wurde als korrekt gelöst angesehen, wenn das Konzept `Person` gewählt wurde. In der Ontologie hätte auch ein neues Konzept für Armar angelegt werden können, welches dann von der Klasse `Person` geerbt hätte. Diese Oberklasse Klasse `Person` wurde gewählt, da Armar als humanoider Roboter ähnliche Eigenschaften wie ein menschlicher Handelnder haben soll. Da aber in ein erstelltes Ontologiekonzept keine weiteren Eigenschaften neben dem Namen eingetragen wären, wurde für Testzwecke weiterhin die Konstante `Person` verwendet. Für jede fehlende `doneBy`-Rolle wurde das richtige Konzept gewählt.

Für die verbleibenden Rollen sind die Anzahl der richtigen und falschen Vervollständigungen in der folgenden Tabelle aufgeführt.

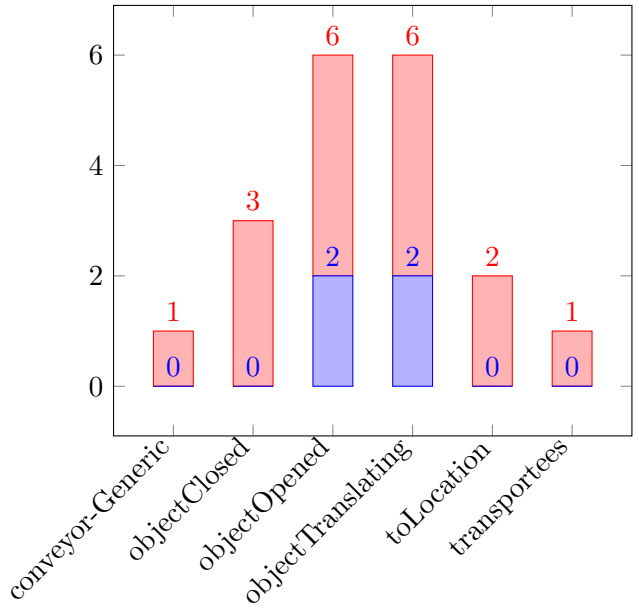

Abbildung 7.2.: Anzahl der sinnvollen Ergebnisse (blau) und der nicht sinnvollen Ergebnisse (rot) je Rolle

Durchschnittlich wurde also, abseits von `doneBy`, mit einer 25 prozentigen Wahrscheinlichkeit ein richtiges Konzept für die verbleibenden Rollen gefunden. An dieser Zahl lässt sich ablesen, dass für viele der Prädikate keine sinnvollen Handlungsträger gefunden werden. Die Ausnahme dazu ist jedoch die Rolle `doneBy`, welche in jedem Fall sinnvoll vervollständigt wurde. Das erfolgt aufgrund der Annahme in Cyc, dass ein Handlungsträger der Rolle

`doneBy` eine Instanz der Sammlung `Individual` sein muss. Alle Objekte aus dem Kontext und der Umgebungsontologie sind jedoch unbelebte Gegenstände. Bei der groben Prüfung der Kandidaten fallen all diese weg, da sie keine Unterklassen von `Individual` sein können. Damit bleibt für die Rolle `doneBy` nur noch eine Entität übrig, die dann richtigerweise gewählt wird.

Bei den anderen Rollen war ein Beweis der Tauglichkeit schwieriger. Viele Prädikate definieren den sehr allgemeinen Argumenttypen `SomethingExisting`. Damit kann durch die grobe Prüfung praktisch kein Konzept ausgeschlossen werden.

Im zweiten Schritt der Prüfung wurden genauere Annahmen aus der Ontologie geprüft. Diese wurden auf den verbleibende Kandidaten angewendet, welche in diesem Fall weiterhin fast alle waren. Dabei kann es aber leicht vorkommen, dass zum Beispiel durch das Vorkommen von `Microwave` und `Refrigerator` im Kontext beide Konzepte die selbe Bewertung erhalten. Dies liegt daran, dass die Annahmen zur Überprüfung der Eignung auf Eigenschaften wie „openable“ bezieht. Für jedes der Konzepte gilt dies, wodurch sie auch beide vom Agenten als gleich geeignet bewertet werden.

Eine Verbesserung dieses Fehlertyps lässt sich eventuell erzielen, indem explizit der Kontext der Situation zuerst für die Vervollständigung verwendet wird und erst danach die Umgebungsontologie. In den Sätzen aus dem Korpus wird meistens nur eine der beiden öffenbaren Entitäten - Kühlschrank und Mikrowelle - genannt. Die andere wird über die Umgebungsontologie zur Aussage hinzugefügt. Korrekt wäre in vielen Fällen jedoch die Entität, welche sich im Kontext befindet.

Ein weiterer Grund, der vermutlich zu der geringen Genauigkeit des Agenten beiträgt ist die Übertragung der Konzepte aus dem Kontext und der Umgebungsontologie auf Konzepte aus Cyc. In einigen Aussagen wird zum Beispiel statt der richtigen Entität `Refriderator` das Konzept `Container` ausgegeben. Dieses wurde fälschlicherweise als Hypernym (Oberbergriff) von `Cup` aus dem Modell des Kontextes extrahiert. Daraus folgt dasselbe Problem, denn eine in Cyc öffenbarer `Container` erhält dann vom Agenten die selbe Bewertung wie das in diesem Fall richtige Wort `Refrigerator`. Außerdem erkennt man hieran Probleme in der Implementierung des Agenten. Diese selektiert offenbar nicht nur Entitäten aus dem Kontextmodell, sondern auch andere Konzepte, die dann hier zu Fehlern führen.

Insgesamt ist die Überprüfung der Kandidaten nur sehr grob. Die Implementierung des Agenten verwendet nur eine kleine Menge des Wissens aus Cyc. Eine Steigerung der Genauigkeit ist durch den Zuzug von weiterem Wissen, und anderen Prüfmetriken vermutlich noch möglich.

8. Zusammenfassung und Ausblick

Diese Arbeit beschäftigte sich mit der Entwicklung eines Ansatzes zur Erkennung von unvollständigen Aussagen in gesprochener Sprache. Ebenfalls wurde betrachtet, ob und wie solche entdeckten Probleme automatisch gelöst werden können. Dazu wurden vorerst Arbeiten vorgestellt, die im Bereich der Sprachverarbeitung und der Informationsextraktion verschiedene ähnliche Probleme aufgreifen.

Daraufhin wurde eine breite Problemanalyse durchgeführt und darauf basierend ein Ansatz entwickelt, der in natürlichsprachlichen Aussagen Unvollständigkeiten erkennt und diese automatisch wieder vervollständigt. Dazu wurde ein wissensbasierter Ansatz verwendet, der auf Annahmen und Regeln aus diversen Ontologien zurückgreift. Zuerst werden hierzu Beschriftungen von einem Rollen- und Aktionserkenner generiert, die dann auf Rollen in der Cyc-Ontologie übertragen werden. Über Listen, welche in Cyc Aktionen Rollen zuweisen, kann dann eine Angabe zur Vollständigkeit der Aussage gemacht werden. Mithilfe dieser Informationen wurden dann Kandidaten aus dem Kontext der Situation und einer Umgebungsontologie zur Vervollständigung der Aussagen überprüft. Dazu wurden Wissensannahmen über die Rollen auf die Kandidaten angewendet, um nicht sinnvolle Kandidaten verwerfen zu können und passende Kandidaten zu selektieren. Der implementierte Agent erzielte dabei beim Erkennen von unvollständigen Aussagen eine Präzision von 51% und beim Vervollständigen von Rollen eine Genauigkeit von 100%, falls das Subjekt des Satzes fehlte und 25% für alle anderen fehlenden Rollen.

Trotz dieser relativ geringen Präzision, bzw. Genauigkeit folgen aus diesem Ansatz noch weitere interessante Ideen, die verfolgt werden könnten um die Ergebnisse zu verbessern. Der Agent produziert neben den eigentlichen Ergebnissen für viele der in dem Satz vorkommenden Wörter Cyc-Konzepte. Dabei wäre es spannend zu überprüfen, ob durch die Betrachtung dieser noch andere Aussagen über die Sätze gemacht werden können, als nur zur Vollständigkeit. Eine Betrachtung der gesamten Aussage, und einer Darstellung des gesamten Satzes in CycL würde bedeuten, dass mittels geeigneter Prädikate und Inferenz der gesamte Satzinhalt mit Wissen aus der Ontologie verglichen werden kann. Dadurch könnten Sätze inhaltlich betrachtet werden und eventuelle Widersprüche und Probleme auch zwischen den Rollen könnten somit gefunden werden.

Viele falsche Vervollständigungen wurden nur produziert, weil nicht ausreichend detailliert geprüft wurde, ob Einträge sinnvoll sein könnten oder nicht. Anderen detaillierter Prüfmetriken könnten auf Basis der Cyc Ontologie entwickelt werden. Ebenso interessant wäre es noch zu verfolgen, welcher weiteren Möglichkeiten sich mit dem von Cycorp geplanten Nl-to-CycL Modul ergeben.

Literaturverzeichnis

[AR18] Ali, Ahmed ; Renals, Steve: Word error rate estimation for speech recognition: e-WER. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* Bd. 2, 2018, S. 20–24 (zitiert auf Seite 22).

[CBL77] Collins, Allan ; Brown, John S. ; Larkin, Kathy M.: Inference in text understanding. In: *BBN report; no. 3684* (1977) (zitiert auf Seite 27).

[CBW+09] Curtis, Jon ; Baxter, David ; Wagner, Peter ; Cabral, John ; Schneider, Dave ; Witbrock, Michael J.: Methods of Rule Acquisition in the TextLearner System. In: *AAAI Spring Symposium: Learning by Reading and Learning to Read*, 2009, S. 22–28 (zitiert auf Seite 16).

[CCB06] Curtis, Jon ; Cabral, John ; Baxter, David: On the Application of the Cyc Ontology to Word Sense Disambiguation. In: *FLAIRS Conference*, 2006, S. 652–657 (zitiert auf den Seiten 13 und 26).

[Ear70] Earley, Jay: An efficient context-free parsing algorithm. In: *Communications of the ACM* 13 (1970), Nr. 2, S. 94–102 (zitiert auf Seite 15).

[GBC07] Guzzoni, Didier ; Baur, Charles ; Cheyer, Adam: Modeling Human-Agent Interaction with Active Ontologies. In: *AAAI Spring Symposium: Interaction Challenges for Intelligent Assistants*, 2007, S. 52–59 (zitiert auf Seite 23).

[GFAH17] Gao, Wei ; Farahani, Mohammad R. ; Aslam, Adnan ; Hosamani, Sunilkumar: Distance learning techniques for ontology similarity measuring and ontology mapping. In: *Cluster Computing* 20 (2017), Nr. 2, S. 959–968 (zitiert auf Seite 27).

[Gün15] Günes, Zeynep: *Aufbau eines Sprachkorpus zur Programmierung autonomer Roboter mittels natürlicher Sprache*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Bachelor's Thesis, Mai 2015. `https://code.ipd.kit.edu/weigelt/parse/wikis/Theses/guenes_ba` (zitiert auf Seite 53).

[Hea92] Hearst, Marti A.: Automatic acquisition of hyponyms from large text corpora. In: *Proceedings of the 14th conference on Computational linguistics-Volume 2*, Association for Computational Linguistics, 1992, S. 539–545 (zitiert auf Seite 17).

[Hey16] Hey, Tobias: *Kontext- und Korreferenzanalyse für gesprochene Sprache*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Diplomarbeit, 2016. `https://code.ipd.kit.edu/weigelt/parse/wikis/Theses/hey_ma` (zitiert auf Seite 6).

[JM14] Jurafsky, Dan ; Martin, James H.: *Speech and language processing*. Bd. 3. Pearson London, 2014 (zitiert auf den Seiten xi, 3, 4, 22, 24 und 30).

[KB09] Korner, Sven J. ; Brumm, Torben: Resi-a natural language specification improver. In: *Semantic Computing, 2009. ICSC'09. IEEE International Conference on*, IEEE, 2009, S. 1–8 (zitiert auf Seite 11).

[KL10] Körner, Sven J. ; Landhäusser, Mathias: Semantic Enriching of Natural Language Texts with Automatic Thematic Role Annotation. In: Hopfe, C. J. (Hrsg.): *Natural Language Processing and Information Systems* Bd. 6177. Berlin, Heidelberg : Springer, Juni 2010 (Lecture Notes in Computer Science). – ISBN 978–3–642–13880–5 978–3–642–13881–2, 92–99. – bibtex: Korner2010a (zitiert auf Seite 11).

[Koc15] Kocybik, Markus: *Projektion von gesprochener Sprache auf eine Handlungsrepräsentation*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Bachelor's Thesis, Juli 2015. `https://code.ipd.kit.edu/weigelt/parse/wikis/Theses/kocybik_ba` (zitiert auf Seite 4).

[Kör14] Körner, Sven J.: *RECAA-Werkzeugunterstützung in der Anforderungserhebung*. KIT Scientific Publishing, 2014 (zitiert auf Seite 22).

[Len95] Lenat, Douglas B.: CYC: A Large-scale Investment in Knowledge Infrastructure. In: *Commun. ACM* 38 (1995), November, Nr. 11, 33–38. `http://dx.doi.org/10.1145/219717.219745`. – DOI 10.1145/219717.219745. – ISSN 0001–0782 (zitiert auf Seite 7).

[Les86] Lesk, Michael: Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In: *Proceedings of the 5th annual international conference on Systems documentation*, ACM, 1986, S. 24–26 (zitiert auf Seite 38).

[MCK04] Mihalcea, Rada ; Chklovski, Timothy ; Kilgarriff, Adam: 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*, 2004 (zitiert auf Seite 26).

[MCN14] Moro, Andrea ; Cecconi, Francesco ; Navigli, Roberto: Multilingual Word Sense Disambiguation and Entity Linking for Everybody. In: *International Semantic Web Conference (Posters & Demos)*, 2014, S. 25–28 (zitiert auf Seite 36).

[MRN14] Moro, Andrea ; Raganato, Alessandro ; Navigli, Roberto: Entity linking meets word sense disambiguation: a unified approach. In: *Transactions of the Association for Computational Linguistics* 2 (2014), S. 231–244 (zitiert auf Seite 26).

[Mue00] Mueller, Erik T.: A calendar with common sense. In: *Proceedings of the 5th international conference on Intelligent user interfaces*, ACM, 2000, S. 198–201 (zitiert auf den Seiten 12 und 23).

[MWK+05] Matuszek, Cynthia ; Witbrock, Michael ; Kahlert, Robert C. ; Cabral, John ; Schneider, David ; Shah, Purvesh ; Lenat, Doug: Searching for common sense: populating Cyc™ from the web. In: *AAAI*, 2005, S. 1430–1435 (zitiert auf den Seiten 17 und 43).

[Ou16] Ou, Yue: *Erkennung von Aktionen in gesprochener Sprache*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Bachelor's Thesis, 2016. `https://code.ipd.kit.edu/weigelt/parse/wikis/Theses/ou_ba` (zitiert auf den Seiten 5, 24 und 34).

[Pal09] Palmer, Martha: Semlink: Linking propbank, verbnet and framenet. In: *Proceedings of the generative lexicon conference*, GenLex-09, Pisa, Italy, 2009, S. 9–15 (zitiert auf den Seiten 27, 34 und 47).

[RCCN17] Raganato, Alessandro ; Camacho-Collados, Jose ; Navigli, Roberto: 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* Bd. 1, 2017, S. 99–110 (zitiert auf Seite 26).

[SSM+06] Shah, Purvesh ; Schneider, David ; Matuszek, Cynthia ; Kahlert, Robert C. ; Aldag, Bjørn ; Baxter, David ; Cabral, John ; Witbrock, Michael J. ; Curtis, Jon: Automated Population of Cyc: Extracting Information about Named-entities from the Web. In: *FLAIRS Conference*, 2006, S. 153–158 (zitiert auf Seite 17).

[Wei] Weigelt, Sebastian: Programming ARchitecture for Spoken Explanations(PARSE) (zitiert auf Seite 4).

Anhang

A. Penn Treebank Tagset

Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP	Possessive wh-pronoun
WRB	Wh-adverb

B. VerbNet Glosses

Thematic Role	Role Description
Actor	Participant that is the instigator of an event
Agent	Actor in an event who initiates and carries out the event intentionally or consciously, and who exists independently of the event.
Asset	Value that is a concrete object.
Attribute	Undergoer that is a property of an entity or entities, as opposed to the entity itself.
Beneficiary	Undergoer in a state or an event that is (potentially) advantaged or disadvantaged by the event or state

Cause	Actor in an event (that may be animate or inanimate) that initiates the event, but that does not act with any intentionality or consciousness; it exists independently of the vent.
Co-Agent	Agent who is acting in coordination or reciprocally with another agent while participating in the same event (specific to events with symmetrical participants).
Co-Patient	Patient that participates in an event with another patient, both participate equally in the event (specific to events with symmetrical participants).
Co-Theme	Theme that participates in an event or state with another Theme; both participate equally (thereby distinguishing this role from Pivot; specific to events with symmetrical participants).
Destination	Goal that is a concrete, physical location.
Duration	Length or extent of time.
Experiencer	Patient that is aware of the event undergone (specific to events of perception).
Extent	Value indicating the amount of measurable change to a participant over the course of the event.
Final_Time	Time that indicates when an event ends or a state becomes false
Frequency	Number of occurrences of an event within a given time span.
Goal	Place that is the end point of an action and exists independently of the event
Initial_Location	Source that indicates the concrete, physical location where an event begins or a state becomes true.
Initial_Time	Time that indicates when an event begins or a state becomes true
Instrument	Undergoer in an event that is manipulated by an agent, and with which an intentional act is performed; it exists independently of the event
Location	Place that is concrete
Material	Patient that exists at the starting point of action (inheritance from Source), which is transformed through the event into a new entity; concrete or abstract.
Participant	Entity involved in a state or event.
Patient	Undergoer in an event that experiences a change of state, location or condition, that is causally involved or directly affected by other participants, and exists ndependently of the event.
Pivot	Theme that participates in an event with another theme unequally. Pivot is much more central to the event (thereby distinguishing it from CoTheme).
Place	Participant that represents the state in which an entity exists.
Product	Result that is a concrete object
Recipient	Destination that is animate.
Result	Goal that comes into existence through the event.
Source	Place that is the starting point of action; exists independently of the event
Stimulus	Cause in an event that elicits an emotional or psychological response (specific to events of perception)
Time	Participant that indicates an instant or an interval of time during which a state exists or an event took place.

Theme	Undergoer that is central to an event or state that does not have control over the way the event occurs, is not structurally changed by the event, and/or is characterized as eing in a certain position or condition throughout the state.
Trajectory	
Topic	Theme characterized by information content transferred to another participant (specific to events of communication).
Undergoer	Participant in a state or event that is not an instigator of the event or state
Value	Place along a formal scale