Wissensrepräsentation und Anfragebearbeitung in einer logikbasierten Mediatorumgebung

Zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
von der Fakultät für Informatik
der Universität Karlsruhe (TH)

genehmigte

Dissertation

von

Peter Kullmann
aus Tübingen

Tag der mündlichen Prüfung	6. Juli 2001
Erster Gutachter	Prof. Dr. J. Calmet
Zweiter Gutachter	Prof. Dr. J. Siekmann

Zusammenfassung

Ein *Mediator* ist eine Software-Komponente, die zwischen einem Anwender und eine mehr oder weniger großen Menge von Informationsquellen unterschiedlichster Natur „vermittelt“. Die Vermittlung ist nicht nur notwendig, weil Informationen in unterschiedlichen, im Allgemeinen inkompatiblen Repräsentationen vorliegen, sondern auch weil Informationen lücken- oder fehlerhaft, unpräzise oder widersprüchlich sein können. Ein Mediator sollte mit solchen Informationen umgehen und sie in sinnvoller Weise bewerten und verschmelzen können. Diesen Vorgang nennt man *Informationsintegration* und beruht auf der Repräsentation und Verarbeitung des dazu benötigten *Integrationswissens*.

Der in dieser Arbeit gewählte Ansatz betrachtet Informationsintegration als ein Problem der logischen Programmierung. Um der Vielfalt der Ausdrucksformen von Informationen und Wissen gerecht zu werden, wurde eine parametrisierbare Logik gewählt, die es erlaubt nicht-triviale Arten von Wissen mit Hilfe von auf die Anwendung zugeschnittenen Wahrheitswerten oder Umstandsmengen zu modellieren, eine so genannte *annotierte Logik*. Für die Verarbeitung von Mediatorprogramme dieser Logik wurde ein Resolutionsverfahren mit Tabulierung entwickelt, das um Funktionen zur Einbindung externer Informationsquellen erweitert wurde.

Insbesondere die Repräsentation von unsicherem Wissen ist wichtig, wenn es darum geht, Wissen zu formalisieren, das beispielsweise durch Erfahrung oder Beobachtung gewonnen wurde. In dieser Arbeit wird untersucht, wie eine Logik, die als Teil der *possibilistischen Logik* die Formulierung von Wissen mit Hilfe von Graden von Notwendigkeit oder Evidenz erlaubt, im Rahmen der in dieser Arbeit verwendeten Logik dargestellt und verarbeitet werden kann. Es wird gezeigt, wie ein Programm der possibilistischen Logik durch eine geeignete Transformation mit dem Auswertungsverfahren der annotierten Logik verarbeitet werden kann. Dadurch existiert für die hier betrachteten Erweiterungen der possibilistischen Logik erstmals eine Operationalisierung, für die auch eine Implementierung vorliegt.

Neben den Fragen der Wissensrepräsentation erfordert die Lösung von Integrationsproblemen die Berücksichtigung einer Reihe von Systemanforderungen, die sich aus den Eigenschaften des Umfeldes der Integration verteilter, heterogener Informationsquellen ergeben können. Dabei sind in erster Linie die Bewältigung großer Datenmengen und Unwägbarkeiten geographisch verteilter Systeme zu nennen.

Ein Ziel dieser Arbeit ist daher, zu untersuchen inwieweit die verwendete Auswertungsprozedur für das Problem der Informationsintegration unter Berücksichtigung der spezifischen Anforderungen geeignet ist, und wie sie sich in diesem Sinne verbessern lässt. Außerdem werden die Eigenschaften und Vorteile eines solchen Ansatzes im Vergleich zu anderen Ansätzen bei deduktiven Datenbanken herausgearbeitet. Ansatzpunkt ist ein Mechanis-

mus, der Zwischenergebnisse der Programmauswertung in geeigneter Weise speichert und verwaltet, und auch zur Verarbeitung annotierter logischer Programme benötigt wird. Dieser Mechanismus wird gezielt für die Anforderungen von Mediatorsystemen angepasst und erweitert. In diesem Zusammenhang wird ein Tabulierungsmechanismus vorgeschlagen, der zur Verarbeitung großer Datenmengen auf den Sekundärspeicher zurück greift und damit auch die persistente Speicherung von Zwischenergebnissen gestattet. Ein solcher Mechanismus kann dann auch gezielt zur Anfrageoptimierung eingesetzt werden, wenn häufig benötigte Ableitungen gezielt gespeichert werden. Auf dieser Basis wird in dieser Arbeit eine Architektur eines logikbasierten Mediatorsystems konzipiert, das für die Bewältigung realer Integrationsaufgaben geeignet ist.

Danksagung

Zunächst möchte ich mich bei Professor Jacques Calmet und Professor Jörg Siekmann für die Betreuung und Unterstützung meiner Arbeit bedanken. Dank schulde ich auch dem Betreuer meiner Diplomarbeit, Joachim Schü, der mich vor einigen Jahren mit diesem Thema in Berührung brachte. Ebenso danke ich den früheren Diplomanden Sebastian Jekutsch und Morio Taneda, mit denen ich viel über Mediatoren diskutiert habe. Ganz besonders möchte ich Frau Prof. Sandra Sandri danken, die eine weitere neue Perspektive auf die annotierte Logik eröffnet hat. Besonderen Dank auch an Clemens Ballarin für seine Kommentare, Korrekturen und seine Kollegialität. Vielen Dank auch an meine Mutter für die abschließende Korrektur. Schließlich danke ich Frau Dietrich für ihre Hilfe im administrativen Bereich und die netten Gespräche „zwischendurch“.

Für Martina

Inhaltsverzeichnis

Wir ertrinken in Informationen, aber hungern nach Wissen.
John Naisbitt

Kapitel 1

Einführung

In diesem Kapitel wird ein Überblick über das Umfeld der Arbeit gegeben. Im ersten Abschnitt wird die grundlegende Problematik der Informationsintegration diskutiert. Im anschließenden Abschnitt wird dann die Idee der Mediatorarchitektur und die Anforderungen erläutert, die an eine solche Architektur aus dem Blickwinkel der Informationsintegration gestellt werden. Abschnitt 1.3 gibt einen Überblick über die Arbeiten und Ansätze, die in den von Mediatorsystemen tangierten Bereichen entstanden sind. Im letzten Abschnitt dieses Kapitels wird das Mediatorsystem KOMET vorgestellt, das im Rahmen dieser Arbeit weiterentwickelt wurde.

1.1 Integration heterogener Informationsquellen

Sowohl im wissenschaftlichen als auch im kommerziellen Bereich sind in den letzten Jahren unzählige Informationssysteme und Datensammlungen mit z.T. erheblichem Aufwand erstellt worden, die auf vielfältigen Datenmodellen und Zugriffsmechanismen basieren. Diese Vielfalt ist meist bedingt durch die Notwendigkeiten und Anforderungen, die bei der Erstellung eines Systems für eine konkretes, abgegrenztes Anwendungsszenario geherrscht haben. In allen Bereichen der Forschung und der Wirtschaft fallen große Datenmengen an, die mit Hilfe der Informationstechnologie gespeichert und verarbeitet werden. Die jüngere Entwicklung der Informationstechnologie erlaubt es aber zunehmend auf die weltweit verteilten Ressourcen der elektronisch gespeicherten Informationen über Netzwerke zuzugreifen. Gleichzeitig haben sich die Anforderungen an Informationssysteme dahin entwickelt, dass einerseits immer umfassendere und komplexere Fragestellungen abgedeckt werden sollen und andererseits die getätigten Investitionen in vorhandene Datenbestände und Informationssysteme geschützt werden müssen. Aus organisatorischen und ökonomischen Gesichtspunkten ist die Erstellung und

anschließende Pflege von Informationsbestände, die anderenorts bereits vorhanden sind, oft kaum sinnvoll, wenn überhaupt möglich.

Im wissenschaftlichen Bereich sind einzelne Informationssysteme in der Regel organisatorisch an die Institution gebunden, die fachlich in der Lage ist, den Datenbestand zu pflegen. Historisch gesehen sind solche Datensammlungen im Rahmen von spezifischen Fragestellungen und Forschungsprojekten entstanden, stellen aber wertvolles Wissen dar und sind in der Regel bei öffentlich finanzierten Projekten auch öffentlich zugänglich.

Im kommerziellen Bereich sind Daten zunächst dort lokalisiert, wo sie entstehen bzw. primär benötigt werden. Innerhalb eines Unternehmens gibt es meist zahlreiche Datenbestände, die primär für die Erfüllung der Aufgaben eines Teilbereichs genutzt und gepflegt werden[1]. Im Sinne eines übergreifenden Managements können aber viele Informationen aus den Teilbereichen eines Unternehmens in Kombination verwendet werden, um umfassende Analysen zu erstellen und wertvolle Hinweise für die strategische Planung zu gewinnen.

Für die Nutzer bietet die einfache Zugänglichkeit des Internets eine hervorragende Möglichkeit, auf umfassende Informationen aus allen Bereichen zuzugreifen. Dies gelingt allerdings nur dann, wenn sie in der Lage sind, die gewünschten Informationen in der unüberschaubaren Vielfalt des Internets zu finden. Aber auch wenn der Ort relevanter Informationen bekannt ist, kann das Zusammenfügen, der Vergleich und Bewertung der Informationen eine mühsame Aufgabe sein.

Aus der Sicht des Anwenders sind die einzelnen Informationsquellen Fragmente, die alle zur Beantwortung von bestimmten Anfragen beitragen können. Optimal nutzbar sind diese Fragmente für den Anwender dann, wenn sie in einen größeren Rahmen eingebettet als integriertes Informationssystem zur Verfügung stehen. Ohne Zutun des Anwenders sollte dieses System die Aufgabe übernehmen, die Informationen aus den verschiedenen Quellen in geeigneter Weise zu kombinieren.

Lösungen für diese Art von Anforderungen, die sich im kommerziellen Einsatz befinden, sind meist ad-hoc-Entwicklungen, die speziell für eine bestimmte Integrationsaufgabe hergestellt wurden. Ihnen fehlt es an der konzeptuellen Architektur und daher an der nötigen Flexibilität, um ein größeres Spektrum an Problemstellungen zu bewältigen. Sie besitzen selten abstrakte Mechanismen um allgemeine Integrationsaufgaben lösen zu können. Aussagen über die Korrektheit der verwendeten Auswertungsverfahren sind meist nicht möglich, da sie nicht formal fundiert sind.

Ein *Mediator* ist eine vermittelnde Softwarekomponente, die die transparente Integration von Informationen aus im Allgemeinen heterogenen, ent-

[1] Beispielsweise fallen Verkaufsinformationen im Vertriebsbereich eines Unternehmens an, während Daten zum Personalbestand im Personalbreich verwaltet werden. Die Möglichkeit diese Daten zur Prognose des Personalbedarfs zu verknüpfen wird normalerweise nicht von Beginn an berücksichtigt.

fernten und autonomen Informationsquellen ermöglicht. Die Problematik der Integration besteht vorrangig in der Behandlung und Auflösung von semantischen Konflikten auf der Ebene der zu integrierenden Schemata, die die Struktur der einzelnen Informationsquellen beschreiben, sowie von inhaltlichen Konflikten auf der Ebene der Daten. In den vergangenen Jahren wurde gezeigt, dass das Mediatorkonzept als grundlegende Architektur ein geeigneter Ansatz ist, um auf heterogene und verteilte Informationsquellen zuzugreifen, sie miteinander zu verknüpfen und dem Anwender transparent zugänglich zu machen. Dabei zielt die Forschung in diesem Bereich darauf ab, theoretisch untermauerte Modelle für die Integration von heterogenen Informationsquellen zu entwickeln, die in einer Umgebung mit wiederverwendbarer Funktionalität realisiert werden können.

1.2 Anforderungen an Mediatorsysteme

Die Mediatorarchitektur wurde erstmals von Gio Wiederhold [Wie92, Wie93] vorgestellt, der sie in seinen Arbeiten als eine Erweiterung der klassischen Client-Server-Architektur um eine weitere Softwareschicht beschreibt. Diese Schicht ermöglicht eine Entkoppelung von Informationsquellen und Informationskonsumenten, die es erlaubt, die Information in verschiedener Weise zu verarbeiten, zu filtern, zu verknüpfen und zu bewerten (Abb. 1.1).

Ein Mediator als Teil der Mediatorschicht vereint dabei das Wissen einer abgegrenzten Domäne in sich und stellt die Informationen der Basisinformationsquellen in einer geeigneten Form zur Verfügung. Die Aufgaben, die durch den Mediator erfüllt werden können, sind breit gefächert:

- Datenkonvertierung
- Normalisierung
- Filterung
- Aggregation, d.h. die zusammenfassende Auswertung von Mengen
- Verknüpfung, d.h. die Zuordnung von in Beziehung stehenden Informationen
- Vervollständigung
- Konsistenzprüfung
- Erkennung, Behandlung und Auflösung von Widersprüchen
- Bewertung

Aber auch Berechnungen, die zusätzliches Wissen erzeugen sind denkbar:

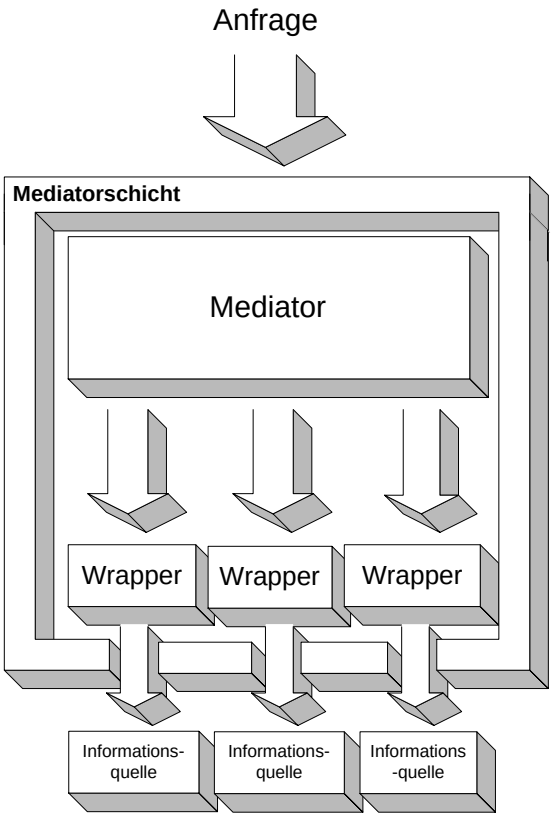

Abbildung 1.1: Mediatorarchitektur

- Interpolation und Extrapolation
- Vorhersagen mit Hilfe von Modellen
- Schätzen und Raten

Aus dieser Sicht haben Mediatorsysteme viele Gemeinsamkeiten mit den Systemen des Gebiets der Informationsgewinnung (engl.: *Information Retrieval*), aus dem entsprechende Anleihen für geeignete Verfahren genommen werden können. Jedoch wird dort die Integration heterogener Informationsquellen nicht berücksichtigt. Die Erfüllung jeder der genannten Dienste kann in einer Mediatorumgebung die Verwendung einer oder mehrerer Basisinformationsquellen bedingen oder aber den Zugriff auf einen anderen Mediator verlangen. Wir beziehen uns dabei auf einen erweiterten Begriff von *Informationsquelle*, der nicht nur Quellen gespeicherter Informationen umfasst, sondern allgemein alle Programme die zu einer gegebenen Eingabe eine Ausgabe berechnen, und sich formal als Menge von Relationen (mit gewissen Beschränkungen) beschreiben lassen. Wir werden später im Detail darauf

eingehen, wie Informationsquellen dargestellt und in eine Mediatorarchitektur eingebunden werden können. Die Informationen, die von den Informationsquellen geliefert werden, können sich in verschiedenen Gesichtspunkten stark unterscheiden (Abb. 1.2).

- Sie können entweder strukturiert, schwach strukturiert oder unstrukturiert sein.
- Sie können Unsicherheit aufweisen.
- Sie sind möglicherweise lücken- oder fehlerhaft.
- Sie sind eventuell unpräzise oder unscharf.
- Sie weisen unterschiedliche Qualität auf.
- Ihre Richtigkeit ist unterschiedlich zuverlässig.
- Sie können zeitliche oder räumliche Abhängigkeiten oder Einschränkungen aufweisen.

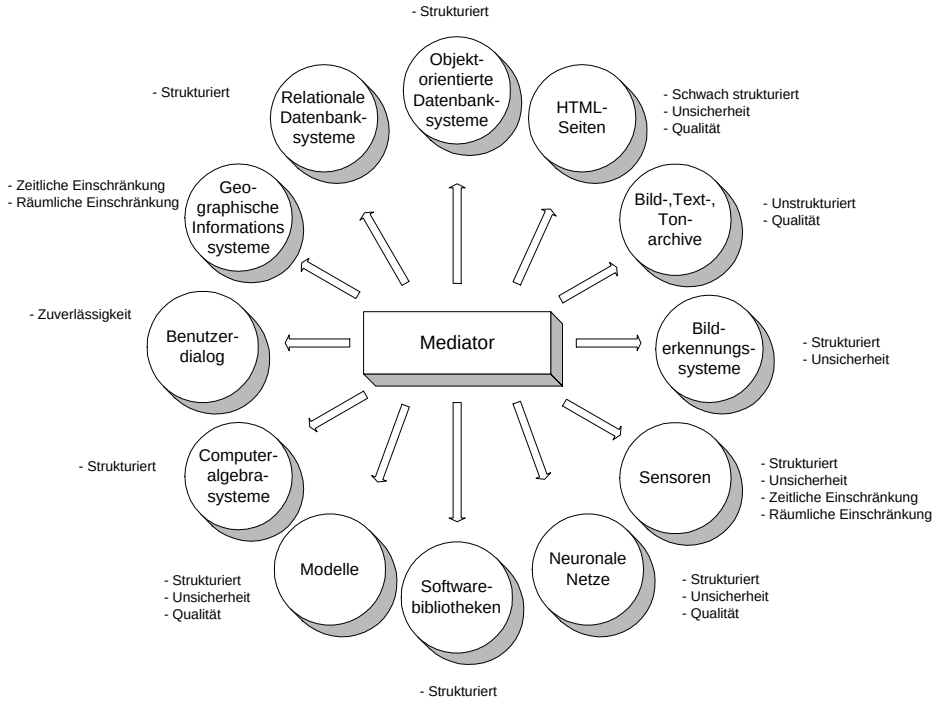

Abbildung 1.2: Beispiele unterschiedlicher Typen von Informationsquellen

Grundlage eines integrierten Systems ist meist das so genannte integrierte Schema oder Mediatorschema, das die erforderlichen Begriffe für Anfragen

an das integrierte System definiert. Dieses Schema ist weder die Schnittmenge noch die Vereinigung der Quellenschemata, sondern ein den Informationsbedürfnissen entsprechend erstelltes Schema des Diskursbereichs. Es kann sich um eine Ontologie bzw. ein semantisches Netz des Diskursbereichs handeln, kann aber auch darüber hinausgehend abgeleitete Konzepte beinhalten, die eher pragmatischer Natur sind.

Auf der Seite der Informationsquellen wird die Beschreibung der Schemata der einzelnen Quellen benötigt, die sich aus der Struktur der zugrunde liegenden Informationen oder den zur Verfügung gestellten Funktionen ergeben. Da im Allgemeinen das Schema einer Quelle nicht in einer für den Mediator geeigneten Form vorliegt, muss dieses entweder, soweit möglich, automatisch oder aber von Hand erzeugt werden. Auch kann die Konvertierung von Daten zwischen dem vom Mediator verwendeten Datenmodell und dem quelleneigenen Datenmodell notwendig sein. Außerdem muss im Allgemeinen eine Anfrage, die vom Mediator an die Quelle übergeben wird, in die Anfragesprache der Informationsquelle transformiert werden und dem quelleneigenen Anfragemechanismus übergeben werden. Die Bereitstellung der Schemainformation, die Datenkonvertierung sowie die Anfrage- und Ergebnistransformation sind in der so genannten *Wrapper*-Schicht angesiedelt. Sie abstrahiert die spezifischen Eigenschaften von Informationsquellen, so dass die Anbindung an die eigentliche Mediatorkomponente möglich wird. Typischerweise wird für jede Informationsquelle ein Exemplar eines Wrappers erzeugt, der die für die Einbindung erforderliche Schnittstelle gegenüber dem Mediator zur Verfügung stellt. Im Idealfall kann die Funktionalität von Wrapper-Komponenten für Klassen von Informationsquellen wieder verwendet werden.

Neben der Definition des Mediatorschemas und der Bereitstellung der für den Mediator geeigneten Schnittstellen zu den Informationsquellen benötigt ein Mediatorsystem eine Beschreibung, die für jede Informationsquelle die Beziehung zwischen den quelleneigenen Schemata und dem Mediatorschema definiert und Heterogenitäten der Begriffe in geeigneter Weise behandelt. Diese Beschreibung wird schließlich herangezogen um eine Anfrage an den Mediator korrekt und effizient entsprechend der gegebenen Definitionen der Informationsquellen behandeln zu können. Enthält die Beschreibung der Quellen und deren Schemata weitere Informationen über besondere Eigenschaften der Quelle und deren Inhalt, ergeben sich daraus möglicherweise Optimierungsmöglichkeiten für den Mediator bei der Koordination von Zugriffen auf Informationsquellen.

Wie bereits angedeutet, kann ein Mediator jedoch weit mehr leisten als die bloße Abbildung von Schemata. Je nach benötigter Funktionalität wird die Verwendung von Wissen in einem nicht unerheblichen Umfang notwendig. Dieses Wissen kann Fachwissen sein, das zur Bewertung benötigt wird, aber allgemeiner auch Strategiewissen. Zum Teil wird dieses Wissen domänenabhängig sein, zum Teil aber auch nicht unmittelbar von der

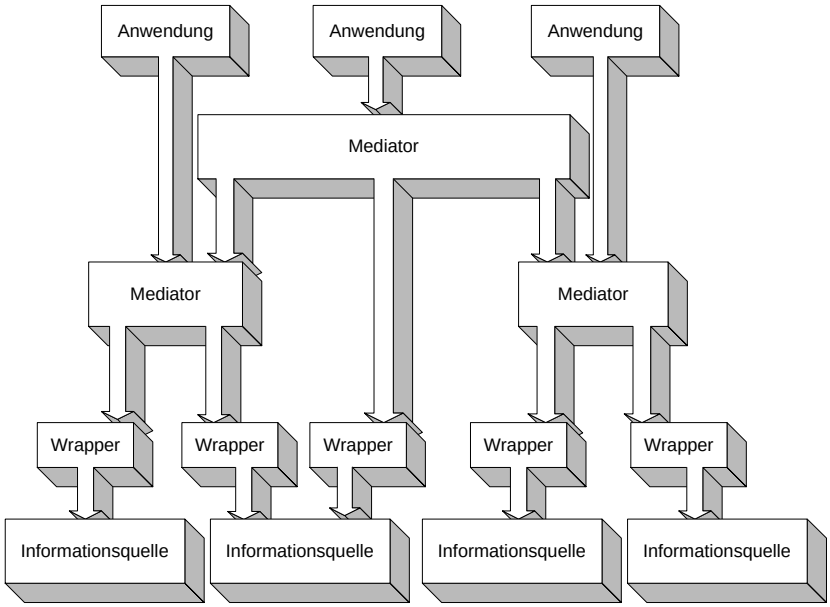

Abbildung 1.3: Komplexe Mediatorumgebung

Domäne abhängen und ist daher für andere Domänen wieder verwendbar. Solches Wissen ist insbesondere notwendig, wenn die Informationen aus einer Menge von Informationsquellen inkonsistent sein können oder sich zumindest in ihrer Qualität unterscheiden. Möglicherweise werden zur Behandlung dieser Fälle komplexe Operationen oder Informationen aus weiteren Quellen benötigt. In jedem Fall kann man von einer Mediator-Wissensbasis sprechen, auf die sich der Mediator bei der Integrationsaufgabe stützt. Aus diesem Grund scheint es angebracht, einen Mediator selbst als wissensbasiertes System anzusehen und sich auf bekannte Techniken aus diesem Bereich zu stützen. Im Licht dieser Betrachtungen zeigen sich bei den bislang existierenden Arbeiten zur Integration heterogener Informationsquellen große Unterschiede. Die überwiegende Zahl von Arbeiten beschäftigt sich mit der Schemaintegration, d.h. der Abbildung der Quellenschemata auf das Mediatorschema. Dieser Ansatz verlagert andere Aufgaben in übergeordnete Schichten einer Gesamtarchitektur. Je nach Anwendung sind die verschiedenen Teile des Integrationswissen jedoch nicht klar oder in sinnvoller Weise trennbar, was aber durch die physische Einteilung der Architektur dieser Ansätze gefordert wird.

In der Schicht zwischen Anwendung und Informationsquelle sind für den Einsatz von Mediatoren unterschiedliche Ausprägungen denkbar (Abb. 1.3). Ein Mediator kann selbst als Informationsquelle auftreten. Dies ist sinnvoll, wenn Mediatoren eingesetzt werden, die auf bestimmte Fragestellungen oder bestimmte Gruppen von Informationsquellen spezialisiert sind. Sie erfüllen

Teilaufgaben, die in mehreren Anwendungen auftreten und können somit wieder verwendet werden. Andererseits kann eine Informationsquelle von verschiedenen Mediatoren in einem unterschiedlichen Zusammenhang genutzt werden. Insgesamt kann so eine komplexes Geflecht von Mediatoren entstehen.

1.3 Stand der Forschung

Der Ansatz dieser Arbeit zur Realisierung einer Mediatorarchitektur lässt sich unter verschiedenen Aspekten betrachten. Er beruht auf der Verwendung von Techniken aus der logischen Programmierung (LP) in Verbindung mit der Verarbeitung von großen Datenmengen, wie sie im Bereich der deduktiven Datenbanken untersucht wird. Darüberhinaus unterstützt ein Mediatorsystem insbesondere die Behandlung von Heterogenitäten und die Einbindung von entfernten Informationsquellen. Dieser Themenkomplex stellt den Gegenstand der Arbeiten im Bereich der Informationsintegration dar. In den folgenden Abschnitten werden, geordnet nach Themengebiet, die wichtigsten Entwicklungen beleuchtet.

1.3.1 Mediatorsysteme

Die generelle Idee der Mediatorarchitektur [Wie92, Wie93] ist unabhängig von einem bestimmten Realisierungsmodell und lässt daher viel Spielraum für die Realisierung eines Mediatorsystems mittels unterschiedlicher Techniken und Methoden. Entsprechend vielfältig sind die Entwicklungen in der Forschung. Die Integration von Informationsquellen bietet zudem eine Vielzahl unterschiedlicher Teilprobleme. Die unterschiedlichen Sichtweisen des Integrationsproblems spiegeln sich in der I^3-Referenzarchitektur wider (Abb. 1.4), die im Rahmen des von der ARPA[2] finanzierten Programms *Intelligent Integration of Information* entstanden ist [HK95]. Mit der Erstellung dieser Architektur wurde erstmals der Versuch unternommen, den Bereich der Informationsintegration zu systematisieren. In der Referenzarchitektur werden vier Dienstfamilien sowie ihre Interaktionen beschrieben. Die Referenzarchitektur verzichtet aber auch ausdrücklich auf ein Realisierungsmodell.

Die Arbeiten im Gebiet der Mediatorsysteme lassen sich gemäß ihrer Ausrichtung meist einer der Familien zuordnen. Die grundlegenden Sichtweisen und deren Zusammenhang mit der Referenzarchitektur lassen sich wie folgt skizzieren:

- Das Mediatorsystem als Vermittler zwischen dem Anwender und einem dynamischen 'Informationsraum' von verfügbaren Informationsquellen. Der Mediator unterstützt die Navigation innerhalb der Menge

[2]Die *Advanced Research Projects Agency* ist eine staatliche Einrichtung der USA zur Förderung wissenschaftlicher Forschung.

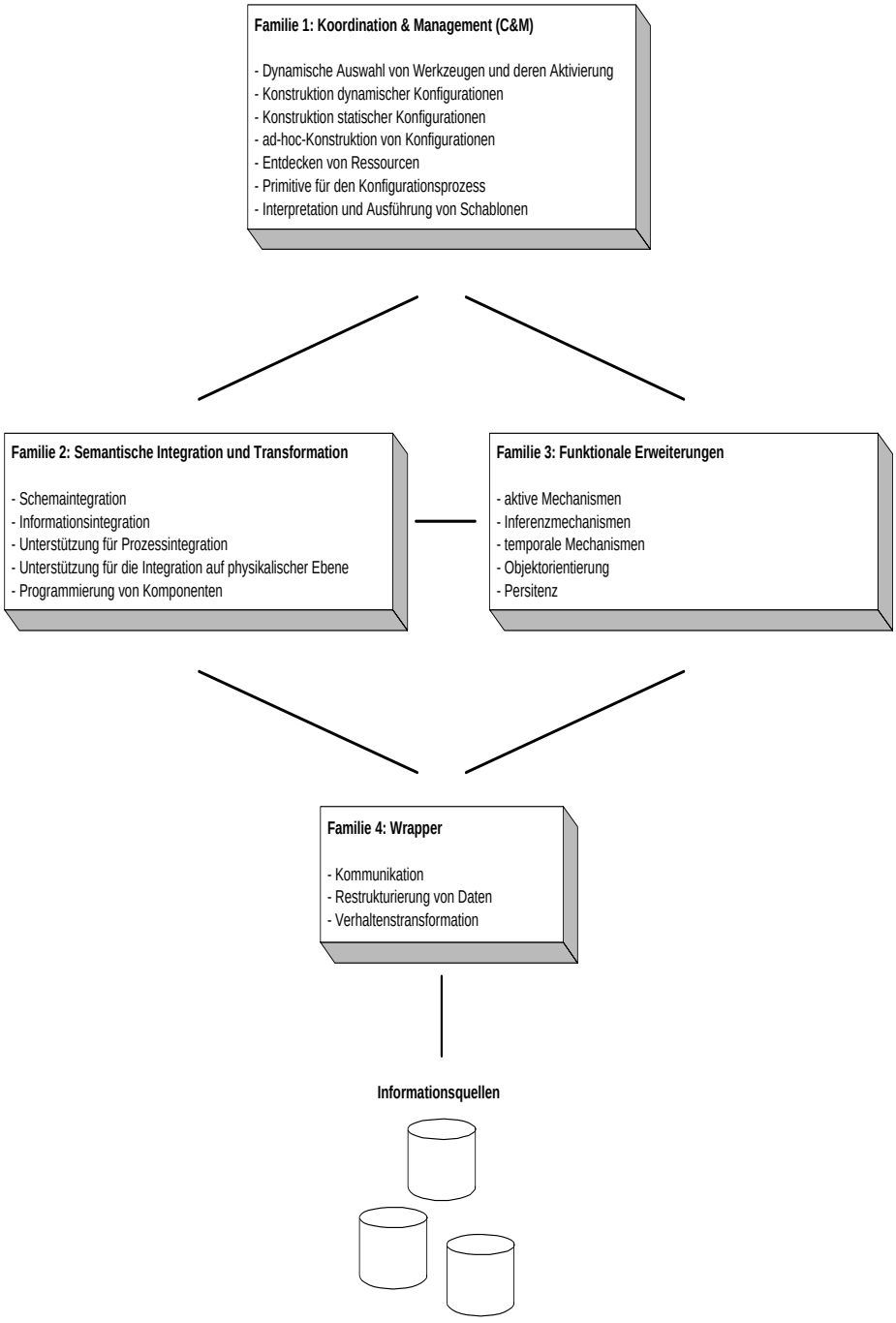

Abbildung 1.4: I^3-Referenzarchitektur

der Informationsquellen, erschließt (semi-)automatisch neue Quellen indem er sie in eine Sammlung von Quellenbeschreibungen einordnet. Systeme, die in diesem Bereich entstanden sind, basieren häufig auf Multiagentenarchitekturen. In diesem Umfeld hat sich auch der Begriff „Information Broker“ etabliert, was sich als „Informationsmakler“ übersetzen lässt. Diese Sicht des Integrationsproblems wird von Systemen und Diensten der Familie 1 der Referenzarchitektur behandelt.

- Das Mediatorsystem als Softwarekomponente, die transparent zwischen dem Anwender und einer eher statischen Menge von Informationsquellen vermittelt. Besonderes Augenmerk wird auf die Behandlung semantischer Heterogenitäten gelegt. Dabei spielt oft ontologisches Wissen eine zentrale Rolle. Es wird benutzt um eine einheitliche Sicht auf den Diskursbereich zu erzeugen. Entsprechende Systeme sind der Familie 2 zuzurechnen.

- Neben Schemaheterogenitäten, die normalerweise pro Informationsquelle einmalig behandelt werden müssen, können Informationen aus unterschiedlichen Quellen inhaltliche Konflikte enthalten, die dynamisch auf der Ebene der Objekte behandelt werden müssen. In die Familie 3 fallen auch Mechanismen, die es erlauben, erweiterte Formalismen zur Repräsentation von komplexen Informationen zu verarbeiten. Dazu zählen beispielsweise temporale oder räumliche Aspekte und Unsicherheit von Wissen.

- Die technischen Heterogenitäten der Informationsquellen werden auf der *Wrapper*-Ebene nivelliert. Dienste dieser Ebene werden der Familie 4 der Referenzarchitektur zugeordnet. In dieser Schicht der Integration werden in der Regel die Informationen in ein einheitliches Datenmodell übertragen. Jedoch ist dabei die Abgrenzung zur Schemaintegration nicht klar. Dementsprechend findet in manchen Ansätzen schon in der Wrapper-Schicht eine Abbildung auf das Mediatorschema statt. Schwerpunkt der Aktivitäten in diesem Bereich ist die zum Teil oder vollständig automatische Erzeugung von Wrapper-Beschreibungen, die den Prozess der Eingliederung von Informationsquellen in eine Mediatorumgebung erleichtern soll.

Zwischen den Dienstfamilien der Referenzarchitektur können verschiedenartige Interaktionen stattfinden. Die Dienste der Familie 3 können die Dienste jeder anderen Familie erweitern. Die Familie 1 stützt sich auf die Familien 2 und 3. Die Dienste der Wrapper-Familie kapseln die Informationsquellen gegenüber den anderen Diensten ab.

Vorläufer der Projekte im Bereich der Informationsintegration finden sich in den Bereichen *Multidatenbanken* und *föderierte Datenbanken*. Dabei steht die Integration der lokalen Schemata bezüglich eines globalen Schemas

im Vordergrund. Technische Heterogenitäten, Heterogenitäten von Datenmodellen oder Inkonsistenzen der Informationen wurden kaum beachtet.

Einige „klassische“ Projekte der Integration heterogener Informationen (z.B. Garlic [RAH+96], Infomaster [GKD97], Information Manifold [KLSS95, LRO96], Infosleuth [JBB+97], OBSERVER [MKSI96], Disco [TRV98], siehe auch [Wie96]) nehmen sich genau dieser Aspekte an. Allerdings beschränken sich diese Ansätze meist auf die Integration (relationaler) Datenbanken und der damit verbundenen eingeschränkten Sichtweise von Informationsquellen. Diesen Arbeiten liegt ein der Datenbanksichtweise typischer Ansatz zur Anfragebearbeitung zugrunde. Aus einer in einer Anfragesprache formulierten Anfrage und der Metainformation des Mediatorsystems werden ein oder mehrere Ausführungspläne generiert, die die Anfrage möglichst gut beantworten sollen. Im zweiten Schritt wird ein gewählter Ausführungsplan durchgeführt. Die Metainformation enthält dabei Beschreibungen der einzelnen Informationsquellen und die Abbildung deren Informationen auf das Mediatorschema. Bei der Beschreibung von Informationsquellen werden unterschiedliche deklarative Formalismen verwendet. Grundlage der Integration ist entweder ein auf die Anwendung zugeschnittenes Informationsschema oder, für allgemeinere Systeme, eine Ontologie, die die Begriffe der Anwendungsdomäne und deren Beziehungen zueinander enthält.

Ein gängiger Ansatz nutzt die Begriffe des Quellenschemas, um daraus die Konzepte des Mediatorschemas in Form von Regeln abzuleiten. Ein anderer Ansatz geht den anderen Weg und benutzt die Konzepte des Mediatorschemas um die Quellenkonzepte zu beschreiben. Auch eine Mischung der beiden Definitionsrichtungen ist möglich [FLM99].

Ein Grund für die Zweiteilung des Anfrageprozesses ist die Eigenschaft der zur Beschreibung der Informationsquellen verwendeten Sprachen, nicht oder nicht effizient ausführbar zu sein. Ihre hauptsächliche Funktion ist die Nutzung zur Spezifikation der Informationsquellen. Erst nach der ersten Bearbeitungsphase wird durch so genanntes *rule rewriting* aus einer Anfrage und der Quellenspezifikation ein auswertbarer Ausdruck erzeugt. Der wesentliche Nachteil dieser Trennung liegt darin, dass in der erste Phase naturgemäß noch nicht alle Metainformationen zur Verfügung stehen, die zur Feststellung der Ausführbarkeit eines Planes benötigt werden. Es ist in der Regel nicht vorhersehbar, ob auf alle Informationsquelle wie erwartet zugegriffen werden kann. Damit ist in einer verteilten Umgebung, die aus einer Vielzahl autonomer Systeme besteht, zu rechnen. Für das zweiphasige Modell bedeutet das, dass Ausführungspläne eventuell vollständig berechnet werden, die später aufgrund unvorhergesehener Umstände nicht ausgeführt werden können. Dadurch wird eine neue Planungsphase erforderlich. Optimierungsverfahren können naturgemäß nicht beide Phasen überspannen, sondern nur getrennt angewendet werden.

Einige Mediatorsysteme (TSIMMIS [CGMH+94, GMPQ+97], HERMES [SAB+95]) legen größeres Gewicht auf die Repräsentation des Integrations-

wissen mit Hilfe von logischen Formalismen. Dabei steht die Sichtweise des Integrationsproblems als ein Problem der logischen Programmierung im Vordergrund. Sie verwenden mehr oder minder ausdrucksstarke Formalismen zur deklarativen Beschreibung des Integrationsprozesses. In TSIMMIS werden mittels eines speziellen Datenmodells insbesondere semistrukturierte Informationen unterstützt. Der besondere Vorteil logischer Formalismen ist, dass sie eine formale Semantik besitzen, die gesicherte Aussagen über die Korrektheit der auf diesen Formalismen definierten Inferenzen zulassen. Die weitere Stärke logikbasierter Ansätze liegt in ihrer Allgemeinheit bezüglich der deklarativen Repräsentation von Wissen. Im Gegensatz zu den datenbankorientierten Systemen lassen sich mittels logischer Formalismen neben Regeln zur Schemaintegration ganz natürlich auch Konfliktauflösungsstrategien auf der Objektebene ausdrücken.

Andere Mediatorsysteme befassen sich mit der Organisation des sich dynamisch verändernden Informationsraums und stützen sich dabei oft auf eine Multi-Agenten-Architektur, in der die verschiedenen Aufgaben auf spezialisierte Agenten verteilt werden. Das Integrationsproblem stellt sich dann als Problem der Koordination und Kommunikation von Agenten, die unterschiedliche Aufgaben wahrnehmen, dar. Im Hinblick auf das Internet geht es dabei oft auch um die automatische Entdeckung und Eingliederung von Informationsquellen [FLM98, BBH+00].

1.3.2 Logisches Programmieren

Mit der Entwicklung des Resolutionsprinzips und dessen praktische Anwendung in der Programmiersprache PROLOG hat sich die Disziplin des logischen Programmierens zu einem wichtigen Bereich des Gebiets der wissensbasierten Systeme entwickelt. PROLOG-Systeme haben sich als adäquates Mittel zur Wissensrepräsentation in der Anwendung etabliert.

Ein Teil der Entwicklung im Gebiet des logischen Programmierens befasst sich mit der Ausdehnung bekannter Inferenzmechanismen auf allgemeinere Klassen logischer Programme, die insbesondere Negation und Rekursion beinhalten. In diesem Zusammenhang werden Semantiken nichtmonotonen Schließens betrachtet. Als Weiterentwicklung der SLD-Resolution[3] [KK71], ist die OLDT-Resolution[4] ein Vertreter der Resolution mit Tabulierung [TS86]. Tabulierende Verfahren speichern die Ergebnisse von Unterzielauswertungen, um die wiederholte Berechnung des selben Unterziels bei rekursiven Programmen zu vermeiden. So kann die Terminierung auch im Falle von positiven Programmen mit Rekursion sicher gestellt werden. Mit der SLG-Resolution[5] existiert ein tabulierendes Verfahren zur Auswertung logischer Programme mit Negation unter der *well-founded*-Semantik

[3] Linear resolution with selection function for definite clauses

[4] Ordered selection strategy with linear resolution for definite clauses with tabulation

[5] Linear resolution with selection function for general logic programs

[CW93, CSW95, CW96]. Programme, bei denen Negation in rekursiven Klauseln zugelassen ist, kann unter einer zweiwertigen Semantik möglicherweise kein eindeutiges Modell zugeordnet werden. Dieses Problem löst die *well-founded*-Semantik durch die Verwendung einer dreiwertigen Logik. Allen Klauseln, bei denen nicht eindeutig bestimmt werden kann, ob sie semantisch folgerbar sind oder nicht, wird der Wahrheitswert *unbekannt* zugeordnet[6]. Die SLG-Resolution ist beschränkt auf Programme, bei denen alle Unterzielauswertungen negierter Rumpfliterale keine Variablen enthalten und durch einen Existenztest beantwortet werden können. Als Erweiterung einer *Warren Abstract Machine* (WAM)[7] ist die SLG-Resolution im XSB-System [SSW93] implementiert. Zugunsten der besseren Einflussnahme auf die Programmabarbeitung wurde in PROLOG der rein deklarative Ansatz aufgegeben. Durch prozedurale Elemente kann auf die Durchquerung des Suchraums Einfluss genommen werden. Bei der Implementierung einer Inferenzprozedur für PROLOG müssen diese prozeduralen Elemente entsprechend berücksichtigt werden und schränken damit die Freiheit der Verarbeitung des Suchraums ein. Damit sind die Möglichkeiten zur Optimierung im Vergleich zum rein deklarativen Ansatz reduziert.

Während die Parallelisierung der Auswertung logischer Programme seit geraumer Zeit diskutiert wird [Kur90], gibt es verhältnismäßig wenig Arbeiten zur Parallelisierung von *tabulierten* Resolutionsverfahren [FHSW95, RSC99, RSC00]. Bei den verfügbaren Arbeiten handelt es sich meist um Einbettungen in PROLOG-Interpreter bzw. WAMs. Auch zum Umfeld der Tabulierung bezüglich der Verwendung unterschiedlicher Datenstrukturen und dem Zusammenhang mit der Anfrageoptimierung findet man nur wenige Arbeiten. In [RRR96, Rao97] wird eine Datenstruktur zur Tabulierung im XSB-System vorgestellt, die auf einem Präfixbaum (*Trie*) beruht. Datenstrukturen, die für tabulierende Resolutionsverfahren geeignet sind, werden im Kapitel 5 untersucht.

1.3.3 Annotierte Logik

Annotierte logische Programme wurden seit etwa 1989 auf der Grundlage mehrwertiger Logik mit Verbänden als Wertemenge untersucht [BS98, KS92, LMR93]. Ein allgemeinerer Ansatz sind *Labelled Deductive Systems*, die Algebren von Kennzeichnungen (engl: *labels*) nutzen, um logische Formeln mit Annotationen zu versehen [Gab96]. In [Hut] wird die annotierte Logik genutzt, um in Theorembeweisern strategisches Wissen über den Beweisprozess zu repräsentieren und zu transportieren. Im Zusammenhang

[6] Details zur *well-founded*-Semantik und Resolutionsverfahren können in Kapitel 2 nachgelesen werden.

[7] Eine *Warren Abstract Machine* realisiert einen definierten Befehlssatz zur unterzielbasierten Auswertung von logischen Programmen in Klauselform, der sich besonders effizient implementieren lässt

mit der Integration heterogener Informationsquellen wurde die annotierte Logik im Projekt HERMES untersucht [Sub92, LNRS93, LNS96, AS93]. Dabei orientiert sich die Sichtweise dieser Arbeiten an der für die logische Programmierung geeignete *Klauselform* logischer Formeln[8]. Der Begriff der hybriden Wissensbasen (engl.: *hybrid knowledge bases*) wurde in [LNRS93, LNS96] eingeführt und bezeichnet eine für Mediatorumgebungen geeignete Repräsentation heterogener Informationsquellen, die auf der Verwendung der annotierten Logik basiert. In der selben Arbeit wird die *well-founded*-Semantik annotierter Programme untersucht, allerdings ohne ein Auswertungsverfahren zu entwickeln. Auf der Basis der SLD-Resolution wurde von S. Leach und J. Lu [LL96] die *ca*-Resolution für annotierte Programme entwickelt, die allerdings auf positive Programme beschränkt ist. Zwar enthält die *ca*-Resolution einen Mechanismus zur Speicherung bereits berechneter Ableitungen, dieser wird jedoch nicht wie bei einer tabulierenden Resolution genutzt, so dass die Terminierung nicht sicher gestellt werden kann. Im XSB-System, welches die SLG-Resolution [CW93] implementiert, wurde die Verarbeitung annotierter Programme mittels eines Metainterpreters realisiert [DPS99].

1.3.4 Deduktive Datenbanken

Deduktive Datenbanken sind als Erweiterungen klassischer relationaler Datenbanken um Möglichkeiten des logischen Programmierens entstanden. Eine Einführung aus der Sicht der logischen Programmierung findet man in [CGH94]. Der Ansatz deduktiver Datenbanken besteht darin

- die Entwicklung logikbasierter Erweiterungen von Datenbanksprachen um mächtigere Anfragen, Schlussfolgerungen und regelbasiertes Programmieren zu erreichen und
- die effiziente und skalierbare Implementierung dieser deklarativen Sprachen für große Datenbasen zu realisieren.

Sehr verbreitet in diesem Bereich ist die Verwendung der eingeschränkten logischen Sprache Datalog, die abgesehen von Rekursion und im Falle sicherer Programme zur relationalen Algebra äquivalent ist [Ull88]. Anders als in Sprachen des logischen Programmierens, werden bei deduktiven Datenbanken Fakten (extensionale Information) und Regeln (intensionale Information) strikt getrennt. Die Menge der Regeln wird als Programm behandelt und auf möglicherweise unterschiedliche Datenbasen angewandt. Meist wird das

[8] Diese Arbeit bezieht sich durchweg auf diese, in [KS92] eingeführte Form der annotierten Logik. Die theoretischen Grundlagen dieser annotierten Logik werden im Detail in Kapitel 2 beschrieben.

Programm mit unterschiedlichen Techniken kompiliert und auf einen Operatorbaum der relationalen Algebra abgebildet [CGK+90]. Eine weit entwickelte Technik rekursive Programme auszuwerten ist die vorwärtsverkettende, *bottom-up*-Auswertung mittels so genannter *magischer Mengen* (engl.: *magic sets*), die den Raum der zu berechnenden Fakten bezüglich einer gegebenen Anfrage einschränkt [BR87]. Dabei werden in einer der Auswertung vorgeschalteten Programmtransformation zusätzliche Programmklauseln erzeugt, die Variablenbindungen so propagieren, dass bei der anschließenden vorwärtsverkettenden Auswertung irrelevante Berechnungen vermieden werden können. Die meisten Anwendungen deduktiver Datenbanken konzentrieren sich auf Verbesserungen konventioneller Datenbanksysteme in den Bereichen *data mining* (Entdeckung von Mustern und Strukturen in Datenbasen) und *knowledge discovery* (Analyse von Daten zur Entdeckung und Abstraktion von regelbasiertem Wissen) und sind auf die Verarbeitung großer Datenmengen ausgelegt [RU95].

Im Gegensatz zu Mediatorsystemen spielt die Behandlung von Heterogenitäten und die Organisation des Informationsraums bei deduktiven Datenbanken keine Rolle. Dennoch sind die hier entwickelten Techniken zur Verarbeitung großer Datenmengen für Mediatorsysteme sicherlich relevant. Die prinzipielle Unterscheidung zwischen Regeln und Fakten entspricht im Ansatz auch den Überlegungen zu logikbasierten Mediatorsystemen.

1.4 Das Mediatorsystem KOMET

Das Projekt KOMET [CJKS97], das durch diese Arbeit fortgeführt wird, verfolgt den Ansatz, eine Umgebung bereit zu stellen, die über die Grundfunktionen verfügt, die zur Programmierung von Mediatorsystemen benötigt werden. Dabei dient die KOMET-Mediatorsprache zusammen mit der Inferenzmaschine des KOMET-Systems als Basiswerkzeug zur Erstellung und zur Ausführung solcher Mediatorkomponenten. Die KOMET-Mediatorumgebung ist auf die Einbindung von Daten entfernter Informationsquellen und die adäquate, wissensbasierte Integration dieser Daten zugeschnitten und basiert auf einer flexiblen, ausdrucksstarken, deklarativen Sprache. Dabei orientiert sich die Mediatorsprache an der klassischen logischen Programmierung, die die Lösung eines breiten Problemspektrums erlaubt. Jedoch wird bewusst auf prozedurale Elemente in der Sprache verzichtet. Die Freiheit, die dadurch bei der Auswertung gewonnen wird, kann wirkungsvoll zur Optimierung eingesetzt werden. Der Anwender sollte, soweit möglich, von der Berücksichtigung technischer Aspekte der Integration, wie effizienter Zugriff und Zwischenspeicherung, befreit sein und sich gänzlich auf die Integrationsaufgabe konzentrieren können. Die Sprachmittel sollten ausreichend sein um das Wissen der Informationsquellen möglichst verlustfrei und dabei natürlich darstellen zu können. Daher basiert KOMET auf einer

Sprache, die die Darstellung unterschiedlicher Arten von Wahrheitswerten gestattet und eine Auswertungsprozedur, die sich bei einer größeren Klasse von Programmen mit Negation robust verhält. Insofern lässt sich KOMET zunächst in die Familie 3 (Funktionale Erweiterungen) der Referenzarchitektur einordnen. Die Mittel, die durch die Mediatorsprache zur Verfügung gestellt werden, eignen sich in gleicher Weise zur Realisierung von Diensten der Familie 1 und 2 (Koordination und Management, Semantische Integration und Transformation). So wäre es denkbar, ontologisches Wissen mit den Sprachmitteln von KOMET zu repräsentieren oder bestimmte Inferenzdienste als unabhängige Komponenten mit der KOMET-Sprache zu realisieren. Als notwendiger Teil eines Mediatorsystems werden im KOMET-Projekt auch Dienste der Familie 4 (Wrapper) untersucht und implementiert.

In diesem Kapitel wurden die verschiedenen Aspekte der Informationsintegration mittels Mediatoren zusammen getragen und im Lichte der Arbeiten in diesem Gebiet betrachtet. Es zeigt sich, dass ein Mediatorsystem in unterschiedlicher Weise realisiert werden kann, je nachdem wie der Schwerpunkt bei den Anforderungen an die Integrationsfähigkeiten eines Systems gesetzt wird. Das System KOMET legt das Hauptgewicht auf einen ausdrucksstarken und flexiblen logischen Formalismus, der die Repräsentation und Verarbeitung eines breiten Spektrums an Integrationsaufgaben in effizienter Weise erlaubt. Die theoretischen Grundlagen dieses Ansatzes werden im nächsten Kapitel im Detail erläutert.

Der Fortschritt geschieht heute so schnell, dass,
während jemand eine Sache für gänzlich undurchführbar erklärt,
er von einem anderen unterbrochen wird,
der sie schon realisiert hat.

Albert Einstein

Kapitel 2

Resolutionsverfahren für annotierte logische Programme

Die Mediatorumgebung KOMET, die im Rahmen dieser Arbeit weiterentwickelt wurde, kombiniert eine Reihe von theoretischen Konzepten, die im Folgenden erläutert werden. Zunächst wird die annotierte Logik vorgestellt, die in einer erweiterten Form in KOMET als Mediatorsprache eingesetzt wird. Im zweiten Abschnitt gehen wir auf Resolutionsverfahren für annotierte logische Programme ein. Anschließend wird die *well-founded*-Semantik im Zusammenhang mit der annotierten Logik diskutiert und ein Resolutionsverfahren für diese Kombination vorgestellt. Im letzten Abschnitt wird die Implementierung der vorgestellten Resolutionsprozedur im Mediatorsytem KOMET beschrieben.

2.1 Annotierte Logik

Informell kann die annotierte Logik als prädikatenlogische Sprache beschrieben werden, bei der jedes Atom mit der expliziten Repräsentation eines Wahrheitswertes oder eines Elements einer Umstandsmenge versehen wird. Dabei entstammt diese *Annotation* einer verbandsgeordneten Struktur, die als solche aber ausgetauscht werden kann. Damit erhält man die Möglichkeit unterschiedliche Arten von Wahrheitswerten bzw. Umstandsmengen problemorientiert in der logischen Sprache einzusetzen.

Ein Programm der verallgemeinerten annotierten Logik (*generalized annotated logic program*, kurz *GAP*) wird über eine verbandsgeordnete Menge $\mathcal{T}$ definiert. Hier sei $\mathcal{T}$ ein vollständiger Verband, d. h. für je zwei Elemente

aus $\mathcal{T}$ existiert eine kleinste obere Schranke und eine größte untere Schranke. Wenn im Folgenden vereinfachend von einem *Verband* gesprochen wird, ist damit ein vollständiger Verband gemeint. Für den Verband $\mathcal{T}$ sei die partielle Ordnung $\leq$ und der Operator $\sqcup$ zur Berechnung der kleinsten oberen Schranke (*least upper bound*, kurz *lub*) gegeben. Wenn $\mathcal{T}$ ein größtes Element besitzt, so sei dies $\top$. $\sqcap$ sei der Operator für die größte untere Schranke (*greatest lower bound*, kurz *glb*) und $\bot$ das kleinste Element von $\mathcal{T}$.

Zu jedem $i \geq 1$ existiere eine Familie von berechenbaren, monotonen Funktionen $\mathcal{F}_i : \mathcal{T}^i \rightarrow \mathcal{T}$. Die Funktionen der Menge $\mathcal{F} = \bigcup_{i \geq 1} \mathcal{F}_i$ werden *Annotationsfunktionen* genannt. Zu jedem $j > 1$ existiere ein Operator $\sqcup_j$, der vom Verbandsoperator $\sqcup$ abgeleitet ist und zu gegebenen Argumenten $\mu_1, \ldots, \mu_j$ die kleinste obere Schranke der Menge $\{\mu_1, \ldots, \mu_j\}$ berechnet. Statt $\sqcup_j$ wird auch abkürzend $\sqcup$ verwendet.

Neben den Annotationsfunktionen enthalte die hier betrachtete Sprache auch Symbole, die *Annotationsvariablen* repräsentieren. Die Elemente aus $\mathcal{T}$ werden auch *Annotationskonstanten* genannt. Weiterhin sind in der Sprache wie gewöhnlich logische Prädikatensymbole, Funktionssymbole, Konstanten und Variablen enthalten, die zur Unterscheidung *Objektvariablen* genannt werden.

Ein *komplexer Annotationsterm* kann wie folgt rekursiv definiert werden:

- Alle Elemente aus $\mathcal{T}$ und alle Annotationsvariablen sind Annotationsterme.

- Ist $f \in \mathcal{F}_n$ eine Annotationsfunktion, und sind $x_1, \ldots, x_n$ Annotationsterme (komplex oder nicht komplex), dann ist auch $f(x_1, \ldots x_n)$ ein komplexer Annotationsterm.

Eine *Annotation* ist ein Element aus $\mathcal{T}$, eine Annotationsvariable oder ein komplexer Annotationsterm. Eine Annotation, die eine Konstante ist, nennt man *c-Annotation*. Eine Annotation, die eine Variable ist, heißt *v-Annotation*. Entsprechend wird eine Annotation, die ein komplexer Annotationsterm ist, *t-Annotation* genannt. Die Unterscheidung zwischen den unterschiedlichen Typen von Annotationen wird benötigt, wenn ihre Verwendung in der logischen Sprache näher betrachten.

In der Literatur sind eine Reihe von typischen Verbänden im Zusammenhang mit annotierter Logik untersucht worden, so z. B. $\mathcal{TWO}, \mathcal{FOUR}, \mathcal{SIX}$ und $\mathcal{DEFAULT}$. Weitere Beispiele für Annotationsverbände sind die reellen Zahlen aus dem Intervall $[0, 1]$, die Potenzmenge der natürlichen Zahlen oder anderer Mengen, wie etwa die Menge der Zeitpunkte oder geographischer Referenzen. Sie erlauben zum Teil die explizite Darstellung und Verarbeitung inkonsistenten Wissens, was z. B. für die Integration heterogener Systeme von besonderer Bedeutung ist.

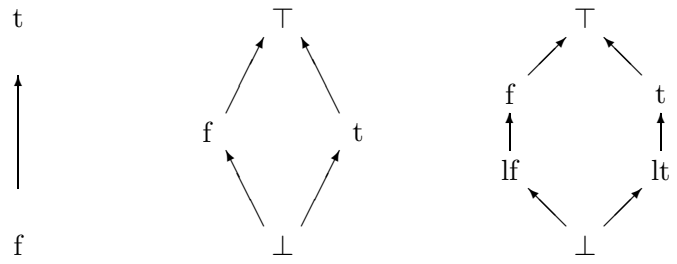

Abbildung 2.1: Verbände $\mathcal{TWO}$, $\mathcal{FOUR}$ und $\mathcal{SIX}$

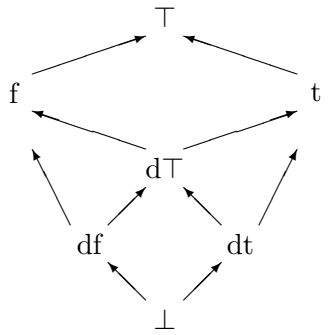

Abbildung 2.2: $\mathcal{DEFAULT}$

Im Verband $\mathcal{SIX}$ bezeichnet *lf* „wahrscheinlich falsch“ und *lt* die Bewertung „wahrscheinlich wahr“. Der Verband $\mathcal{DEFAULT}$ erlaubt die Verwendung von Vorgaben (engl.: *defaults*) als Wahrheitswerte. Gemäß der Verbandsordnung können diese von anderen Wahrheitswerten „überschrieben“ werden.

Definition 2.1 (Annotiertes Atom) *Seien A ein Atom einer gewöhnlichen prädikatenlogischen Sprache und μ eine Annotation, dann ist $A : [\mu]$ ein* annotiertes Atom. *Ist μ eine c-, v- oder t-Annotation, dann heißt A* c-, v-, bzw. t-annotiert. *Ein annotiertes Atom ohne Objektvariablen heißt* Grundatom. *Ein Grundatom ohne Annotationsvariablen heißt* striktes Grundatom.

Definition 2.2 (Annotierte Klausel)
Eine annotierte Klausel *hat allgemein die Form:*

$$p : [\mu] \quad \leftarrow \quad p_1 : [\mu_1] \wedge \cdots \wedge p_k : [\mu_k] \wedge \\ \textbf{not } (p_{k+1} : [\mu_{k+1}]) \wedge \cdots \wedge \textbf{not } (p_{k+n} : [\mu_{k+n}])$$

wobei $p : [\mu]$ ein annotiertes Atom ist und $p_i : [\mu_i]$ c- oder v-annotierte Atome sind. Das annotierte Atom $p : [\mu]$ ist der Kopf *der Klausel, $p_1 :$*

$[\mu_1] \wedge \cdots \wedge p_k : [\mu_k] \wedge \mathbf{not}\ (p_{k+1} : [\mu_{k+1}]) \wedge \cdots \wedge \mathbf{not}\ (p_{k+n} : [\mu_{k+n}])$ *wird* Rumpf *der Klausel genannt.*

Ohne Beschränkung der Allgemeinheit kann angenommen werden, dass im Kopf einer annotierten Klausel nur Annotationsvariablen existieren, die auch im Rumpf dieser Klausel auftreten. Nicht im Klauselkopf gebundene Variablen sind per Definition existenzquantifiziert. Ein mit einer solchen Annotation versehenes Rumpfliteral ist aber stets durch das minimale Element des Annotationsverband trivial erfüllbar und kann durch dieses ersetzt werden. Gemäß obiger Definition können annotierte Funktionen nur im Kopf einer Klausel existieren. Eine Klausel mit leerem Rumpf heißt *Einheitsklausel* oder *Faktum*, eine Klausel ohne Kopf *Anfrage* und eine Klausel mit Kopf und Rumpf *Regel*.

Beispiel 2.1 *Folgendes sind Beispiele für annotierte Aussagen und annotierte Klauseln unter Verwendung unterschiedlicher Annotationsverbände:*

$istVogel(tweety) : [t]$
$kannFliegen(tweety) : [f]$
$Regen(hamburg) : [0.5]$
$hatZeit(müller) : [\{Mo, Sa, So\}, 0.7]$

$kannFliegen(X) : [dt] \leftarrow istVogel(X) : [t]$
$hatZeit(X) : [T, U] \leftarrow \mathbf{not}(hatTermin(X) : [T, U])$

Eine Menge von annotierten Klauseln wird *verallgemeinertes annotiertes Programm* (*generalized annotated logic program*, kurz *GAP*) genannt.

Sind $A : [\mu_1]$ und $A : [\mu_2]$ zwei annotierte Atome, dann wird durch die Inferenzregel

$$\frac{A : [\mu_1], A : [\mu_2]}{A : [\sqcup(\mu_1, \mu_2)]}$$

eine sog. *Reduzierte (Reductant)* gebildet. Diese Inferenzregel heißt *Reduktionsregel* (engl.: *reduction rule*). Sie sammelt gewissermaßen die Annotationen eines Atoms ein und bildet daraus die logische Konsequenz in Form der kleinsten oberen Schranke der Annotationen[1].

Beispiel 2.2 *Seien die Fakten* $\mathbf{istVogel}(\mathbf{tweety}) : [f]$ *und* $\mathbf{istVogel}(\mathbf{tweety}) : [t]$ *gegeben. Die Anwendung der Reduktionsregel erlaubt die Ableitung des Faktums* $\mathbf{istVogel}(\mathbf{tweety}) : [\top]$. *Im Verband* $\mathcal{FOUR}$ *kann* $\top$ *als Inkonsistenz der Faktenbasis interpretiert werden.*

[1] Die ursprünglichen Atome werden durch die Reduzierte aufgrund der Annotationsordnung subsumiert und können im weiteren vernachlässigt werden. Die Faktenmenge wird dadurch in gewissem Sinne reduziert, woraus der Name dieser Inferenzregel abgeleitet wird.

2.1.1 Semantik annotierter Programme

Für die Definition der Semantik der annotierten Logik werden in [KS92] zwei Ansätze angegeben, die *eingeschränkte Semantik* und die *allgemeine Semantik*. Im Unterschied zur eingeschränkten Semantik erlaubt die allgemeine Semantik die Verwendung einer größeren Klasse von Annotationsfunktionen, da sie die Bildung der kleinsten oberen Schranke von *unendlichen* Mengen von Annotationen erlaubt. Da dieser Fall für die praktische Anwendung nicht relevant ist, beschreiben wir hier die eingeschränkte Semantik. Details können in [KS92] nachgelesen werden.

Definition 2.3 (r-Interpretation)
Eine eingeschränkte Herbrand-Interpretation *(auch* restricted Interpretation *oder kurz* r-Interpretation*) ist eine Abbildung von der Herbrand-Basis*[2] $B_{\mathcal{L}}$ *auf den Verband* $\mathcal{T}$*. Es gelte die Voraussetzung, dass* $\mathcal{T}$ *ein vollständiger Verband ist.*

Definition 2.4 (r-Erfüllbarkeit)
Seien I eine r-Interpretation, $\mu \in \mathcal{T}$ *eine c-Annotation und A ein Grundatom.*

1. $I \models^r A : [\mu]$ *gdw.* $\mu \leq I(A)$.
2. $I \models^r \mathbf{not}(A : [\mu])$ *gdw.* $\mu > I(A)$.
3. $I \models^r F_1 \wedge F_2$ *gdw.* $I \models^r F_1$ *und* $I \models^r F_2$.
4. $I \models^r F_1 \vee F_2$ *gdw.* $I \models^r F_1$ *oder* $I \models^r F_2$.
5. $I \models^r F_1 \leftarrow F_2$ *gdw.* $I \models^r F_1$ *oder* $I \not\models^r F_2$.
6. $I \models^r F_1 \leftrightarrow F_2$ *gdw.* $I \models^r F_1 \leftarrow F_2$ *und* $I \models^r F_2 \leftarrow F_1$.
7. $I \models^r (\forall x)F$ *gdw.* $I \models^r F\{x/t\}$ *für alle Grundterme t (x ist Objektvariable oder Annotationsvariable)*[3].
8. $I \models^r (\exists x)F$ *gdw.* $I \models^r F\{x/t\}$ *für einen Grundterm t (x ist Objektvariable oder Annotationsvariable).*
9. *Falls F keine geschlossene Formel ist:* $I \models^r F$ *gdw.* $I \models^r (\forall)F$ *((*$\forall$*)F ist der universelle Abschluß von F).*

Definition 2.5 (r-Modell)
Seien I eine r-Interpretation und F eine Formel, dann ist I ein r-Modell von F genau dann, wenn $I \models^r F$ *gilt. Sei P ein GAP, dann ist I genau dann ein r-Modell von P, wenn I ein r-Modell von allen Formeln in P ist.*

[2] Die Herbrand-Basis bezüglich einer Signatur Σ ist die Menge aller Grundatome in Σ

[3] In dieser Arbeit wird die Substitution durch „/“ dargestellt. So bedeutet x/t, daß x durch t ersetzt wird.

Die Einschränkungen, die für die Annotationsfunktionen gemacht werden müssen, damit nach einer endlichen Anzahl von Deduktionsschritten ein Fixpunkt erreicht werden kann, sind nach [LNS96] die folgenden:

- Alle Funktionen sind endlich oder antimonoton.
- Die Definitionsbereiche aller Funktionen sind endlich.

2.1.2 Mehrsortige annotierte Logik

Eine realistische Anwendung der annotierten Logik in einer heterogenen Umgebung verlangt den Umgang mit beliebigen Objekten unterschiedlicher Typen und Strukturen und macht daher die Verwendung einer mehrsortigen annotierten Logik notwendig. Sorten stellen zudem eine einfache und effiziente Methode dar, implizite Integritätsbedingungen zu formulieren. Durch die Festlegung jeder Argumentstelle eines Prädikats auf eine Sorte wird die Menge der möglichen Ausprägungen beschränkt. Einer Variablen wird fest die Sorte der Argumentstelle ihres ersten Auftretens zugeordnet. Sorteninkonsistente Variablenbindungen können schon bei einer statischen Überprüfung des Programms festgestellt werden.

Definition 2.6 (Mehrsortige annotierte Signatur)
Ist M eine beliebige Menge, so bezeichne M^* bezeichne die Potenzmenge von M, ε die leere Menge. Eine mehrsortige annotierte Signatur Σ *ist ein Tupel (S,Ω,L,Ψ,Π), für das gilt:*

1. *S ist eine endliche, nichtleere Menge, deren Elemente* Sortensymbole *oder kurz* Sorten *genannt werden.*
2. *$\Omega = (\Omega_{w,s} \mid w \in S^*, s \in S)$ ist eine Familie von paarweise disjunkten Mengen $\Omega_{w,s}$ von Funktionssymbolen der Stelligkeit $|w|$ und der Zielsorte s. Zusätzlich wird gefordert, dass Ω eine endliche Menge ist.*
3. *L ist eine endliche, nichtleere Menge, deren Elemente* Annotationsverbandssymbole *oder kurz* Annotationsverbände *genannt werden.*
4. *$\Psi = (\Psi_{w,l} \mid w \in L^*, l \in L)$ ist eine Familie von paarweise disjunkten Mengen $\Psi_{w,l}$ von Funktionssymbolen der Stelligkeit $|w|$ und des Zielverbands l. Ψ sei ebenfalls eine endliche Menge.*
5. *$\Pi = (\Pi_{w,l} \mid w \in S^*, l \in L)$ ist eine Menge von paarweise disjunkten Mengen Π_w von Prädikatensymbolen der Stelligkeit $|w|$ und des Annotationsverbands l.*

Ein Funktionssymbol $f \in \Omega_{\varepsilon,s}$ ($g \in \Psi_{\varepsilon,l}$) heißt auch *Konstantensymbol* der Sorte s (des Verbands l).

In der üblichen Weise können aus einer Variablenmenge V_Ω und Ω Funktionsterme bzw. aus einer Variablenmenge V_Ψ und Ψ Annotationsterme gebildet werden.

Beispiel 2.3 *Das Tupel (S,Ω,L,Ψ,Π) mit*

$$S = \{STRING, INT\}$$
$$\Omega = \{reverse(STRING) : STRING\}$$
$$L = \{FOUR\}$$
$$\Psi = \{not(FOUR) : FOUR\}$$
$$\Pi = \{P(STRING) : [FOUR], Q(INT) : [FOUR]\}$$

ist ein Beispiel für eine mehrsortige annotierte Signatur mit einer Sorte STRING, deren Elemente Zeichenketten sind, einer Sorte INT, deren Elemente ganze Zahlen zwischen -32767 und 32768 sind und einem Verband FOUR, dessen Elemente aus $\mathcal{FOUR}$ stammen. Die Funktion reverse *liefert zu einem Argument ein Objekt der Sorte STRING mit umgekehrter Reihenfolge der Zeichen. Die Funktion* not *bildet t auf f, f auf t und $\top$ und $\bot$ auf sich selbst ab.*

Definition 2.7 (Mehrsortige annotierte Klausel)
Sei Σ = (S,Ω,L,Ψ,Π) eine mehrsortige annotierte Signatur. Eine mehrsortige annotierte Klausel *hat dann folgende Form:*

$$l(t_1^l, \ldots, t_{w_l}^l) : [\mu] \leftarrow l_1(t_1^{l_1}, \ldots, t_{w_{l_1}}^{l_1}) : [\mu_1] \wedge \cdots \wedge l_k(t_1^{l_k}, \ldots, t_{w_{l_k}}^{l_k}) : [\mu_k]$$

Hierbei sind l_i sind Literale, deren Prädikate aus Π stammen und die (außer im Klauselkopf) auch negiert sein können. Die Prädikatenargumente $t_j^{l_i}$ sind Funktionsterme der Zielsorte, die durch die j-te Argumentstelle des Prädikatensymbols des Literals l_i gegeben ist. Die Stelligkeit eines Literals l_i ist durch w_{l_k} gegeben. μ ist ein komplexer Annotationsterm und μ_i ist eine c- oder v-Annotation aus dem Verband, der durch den Annotationsverband des Prädikatensymbols des Literals l_i gegeben ist.

Ein *mehrsortiges annotiertes Programm* ist eine Menge von mehrsortigen annotierten Klauseln, deren Klauselrümpfe auch leer sein können. In diesem Fall spricht man von *annotierten Fakten.*

In Programmen der annotierten Logik unterscheidet man zwischen zwei Arten der Negation. Die monotone, *epistemologische* oder *explizite* Negation und die nicht-monotone, *ontologische* oder *Default*-Negation. Erstere wird in Programmen der annotierten Logik durch eine Abbildung realisiert, die jedem Verbandselement sein Komplement zuordnet.

Beispiel 2.4 (Explizite Negation)
Gegeben sei folgendes Programm unter dem Verband $\mathcal{FOUR}$

$istGroß(X) : [not(V)] \leftarrow istKlein(X) : [V]$
mit den Fakten
$istKlein(paul) : [t]$ *und*
$istKlein(maria) : [f]$.

Auf die Anfrage $istGroß(X) : [V]$ *erhält man als Antwort:*
$istGroß(paul) : [f]$ *und*
$istGroß(maria) : [t]$

Man beachte, dass eine solche Abbildung nicht für alle Verbände angegeben werden kann (z. B. Zeitpunkteverband als Potenzmenge von $\mathbb{N}$). Diese Art der Negation ist monoton, das heißt Klauseln, die eine Negation enthalten, fügen beim Hinzukommen neuer Fakten allenfalls neue Fakten zur Gesamtmenge aller ableitbarer Fakten hinzu.

Im Gegensatz dazu bezieht sich die *Default*-Negation auf die Beweisbarkeit einer Aussage. Ein negiertes Atom A ist dann erfüllt, wenn A nicht beweisbar ist. Diese Art der Negation ist nicht monoton. Bei neu hinzukommenden Fakten kann eine Klausel mit *Default*-Negation bewirken, dass Aussagen, die vorher getroffen wurden, nun nicht mehr haltbar sind, das heißt die Gesamtmenge der ableitbaren Fakten kann abnehmen.

Beispiel 2.5 (Default-Negation)
Sei folgendes Programm gegeben:

$istGroß(X) : [V] \leftarrow \mathbf{not}(istKlein(X) : [V])$
Für den Verband $\mathcal{FOUR}$ *und mit den Fakten*
$istKlein(paul) : [t]$ *und*
$istKlein(maria) : [f]$

erhält man auf die Anfrage $istGroß(X) : [V]$ *als Antwort:*
Für alle X *außer 'paul' und 'maria' ist* $V > \bot$
Für $X = paul$ *ist* $V \in \{f, \top\}$.
Für $X = maria$ *ist* $V \in \{t, \top\}$.

Um diese Lösung folgern zu können, muss zunächst die Annahme gemacht werden, dass alle nicht explizit als wahr gegebenen Fakten als falsch zu betrachten sind[4]. Diese Annahme wird *Closed World Assumption* genannt

[4]Diese Zuordnung von Wahrheitswerten zu annotierten Aussagen ist streng zu trennen von den Wahrheitswerten, die durch die Annotationen gegeben sind. *Wahr* wird hier verwendet im Sinne von *beweisbar* oder *ableitbar*. Entsprechend meint man mit *falsch nicht beweisbar* oder *nicht ableitbar*. Man muss sich diesen Unterschied insbesondere vor Augen halten, wenn man im Zusammenhang mit der „well-founded"-Semantik mit einer dreiwertigen Logik arbeitet. Dort kann die Ableitung eines Atoms *unbekannt* sein, wenn es weder bewiesen noch widerlegt werden kann.

und geht auf R. Reiter [Rei78] zurück[5]. Mit Clarks *Negation as Failure*-Regel [Cla78] wird **not**(A) aus einem Programm P dann abgeleitet, wenn A keine semantische Folgerung aus P ist. Für obiges Programm muss also festgestellt werden, ob *istKlein(X)* abgeleitet werden kann. Ist X nicht instantiiert, muss bestimmt werden, für welche X das Atom beweisbar ist. Nur wenn X instantiiert wurde, kann dies explizit aus der Existenz oder Nicht-Existenz von Antworten für das Unterziel *istKlein(X)* festgestellt werden. Im anderen Fall muss die Antwort konstruiert werden. Wegen der Komplexität der konstruktiven Negation beschränken sich gängige Implementierungen auf sog. *sichere Programme*. In sicheren Programmen gibt es keine negativen Literale, die zum Zeitpunkt der Auswertung Variablen enthalten, welche instantiiert werden müssen. Es muss lediglich bestimmt werden, ob das grundinstantiierte Atom beweisbar ist; entsprechend scheitert die Klausel oder wird weiter bearbeitet. Durch negative Literale werden so keine Variablenbindungen in der Klausel erzeugt.

Annotierte Logik kann in natürlicher Weise zur Integration heterogener Wissensquellen erweitert werden [Sub92, AS93, LNRS93]. Um Klauseln formulieren zu können, die zur Konfliktauflösung dienen, benützt man einen Annotationsverband, der die Herkunft einer annotierten Aussage ausdrückt. Solche Klauseln werden *Mediatorklauseln* genannt. Ein *Mediator* ist ein Programm, dessen Menge von Mediatorklauseln die Konfliktauflösung zwischen mehreren Wissensquellen betreibt.

2.1.3 Resolution annotierter Programme

Kifer und Subrahmanian beschreiben in [KS92] eine Resolution für annotierte Programme, bei der Klauseln um spezielle Randbedingungen (engl.: *Constraints*) erweitert werden. Diese Randbedingungen drücken die Beziehungen zwischen den Annotationen aus und entstehen bei einem Resolutionsschritt. So ist die Resolvente einer Anfrage $\leftarrow A : [t]$ und einer Programmklausel $A : [V] \leftarrow B : [V]$ gegeben durch folgende Klausel mit Randbedingung:

$$\leftarrow B : [V] \wedge t \leq V$$

Definition 2.8 (Anfrage mit Annotationsrandbedingung)
Sei Ξ eine Konjunktion von Ausdrücken der Form $t_1 = t_2$ oder $t_1 \leq t_2$, wobei t_1 und t_2 Annotationsterme sind. Dann ist

$$\leftarrow \Xi \parallel q_1 : [\alpha_1] \wedge \cdots \wedge q_n : [\alpha_n]$$

eine annotierte Anfrage mit Annotationsrandbedingung. *Ist t_1 eine c- oder v- Annotation, dann wird die Randbedingung* normal *genannt.*

[5] Für die annotierte Logik bedeutet das, dass ein Atom, für das explizit nichts anderes angegeben wurde, die Annotation $\perp$, das kleinste Element des Annotationsverbands, gilt.

Die im Folgenden vorgestellte Resolutionsregel beschränkt sich auf die Resolution einer annotierten Anfrage mit einer Programmklausel und ist für rückwärtsverkettende, lineare Resolutionsprozeduren ausreichend.[6]

Definition 2.9 (Resolutionsregel für annotierte Programme)
Seien p eine annotierte Klausel und q eine annotierte Anfrage mit Annotationsrandbedingung. Sei θ der allgemeinste Unifikator, der die Literale q_i und p_0 unifiziert. Dann ist

$$R^a : \frac{\begin{array}{rcl} p_0 : \delta_p & \leftarrow & p_1 : \beta_1 \wedge \cdots \wedge p_n : \beta_n \\ & \leftarrow & \Xi \parallel q_1 : \alpha_1 \wedge \cdots \wedge q_n : \alpha_n \end{array}}{\begin{array}{rcl} \leftarrow [\, \Xi, \alpha_i \leq \delta_p \parallel & & q_1 : \alpha_1 \wedge \cdots \wedge q_{i-1} : \alpha_{i-1} \wedge \\ & & p_1 : \beta_1 \wedge \cdots \wedge p_n : \beta_n \wedge \\ & & q_{i+1} : \alpha_{i+1} \wedge \cdots \wedge q_m : \alpha_m \,] \, \theta \end{array}}$$

die Resolutionsregel für annotierte Programme.

Das Anwendung von R^a liefert die *Resolvente* einer Anfrage und einer Programmklausel eines Programms der annotierten Logik. Desweiteren ist die Reduktionsregel wie folgt definiert.

Definition 2.10 (Reduktionsregel der Annotierten Logik)
Seien p und r annotierte Klauseln. Sei θ der allgemeinste Unifikator, der die Kopfliterale p_0 und q_0 unifiziert. Dann ist

$$D^a : \frac{\begin{array}{c} p_0 : \delta_p \leftarrow p_1 : \beta_1 \wedge \cdots \wedge p_n : \beta_n; \\ r_0 : \delta_r \leftarrow r_1 : \gamma_1 \wedge \cdots \wedge r_m : \gamma_m \end{array}}{\begin{array}{rcl} [\, p_0 : \sqcup(\delta_p, \delta_r) & \leftarrow & p_1 : \beta_1 \wedge \cdots \wedge p_n : \beta_n \wedge \\ & & r_1 : \gamma_1 \wedge \cdots \wedge r_m : \gamma_m \,] \, \theta \end{array}}$$

die Reduktionsregel der annotierten Logik.

Eine *Deduktion* einer annotierten Anfrage Q_0 mit Randbedingung und einer Menge von annotierten Klauseln P ist eine Sequenz:

$$Q_0, \langle R_0, \theta_0 \rangle, \ldots, Q_n, \langle R_n, \theta_n \rangle, Q_{n+1}.$$

Dabei ist

- R_i eine Reduzierte zweier Klauseln aus P
- Q_{i+1} eine Resolvente von Q_i und R_i mit dem Unifikator θ_i

[6] Eine lineare Resolution beginnt mit einer Anfrage und resolviert nur jeweils die Resolvente des vorhergehenden Resolutionsschritts mit einer Programmklausel oder dem Ergebnis eines vorhergehenden Resolutionsschritts.

Eine Deduktion ist eine *Widerlegung*, genau dann wenn die Resolvente Q_{i+1} keine Atome enthält und die Konjunktion der Randbedingungen bezüglich des Annotationsverbands $\mathcal{T}$ und der Annotationsfunktionen in $\mathcal{F}$ lösbar ist. Nach [KS92] ist die annotierte Resolution vollständig:

Theorem 2.1 *Sei $\mathcal{T}$ ein Verband, P ein GAP und Q eine normale Anfrage. Es gelte $P \models (\exists)Q$. Dann existiert eine Widerlegung von Q aus P.*

Die annotierte Resolution setzt voraus, dass für alle Klauseln, deren Köpfe unifizierbar sind, Reduzierte gebildet werden. Obwohl diese Resolution im strengen Sinne nicht linear ist, wird sie in [KS92] *SLD-artig* genannt. Dies ist wohl berechtigt, wenn alle Reduzierten zu Beginn durch eine Programmtransformation erzeugt werden und die Resolution dann mit dem transformierten Programm arbeitet. Offensichtlich ist dieses Vorgehen im allgemeinen nicht effizient, da Klauseln gebildet werden, die für die Anfrage unerheblich sind. Für theoretische Betrachtungen jedoch ist diese einfache annotierte Resolution nützlich, wie wir später sehen werden.

Wie Leach und Lu [LMR93, LL96] beschreiben, kann die Bildung der Reduzierten auch dynamisch erfolgen. Dies wurde in der *ca*-Resolution realisiert.

Bezeichne $\Uparrow \mu$ das *up-set* einer Annotation μ und sei wie folgt definiert: $\Uparrow \mu = \{\nu | \nu \geq \mu\}$. Das *up-set* einer Annotation enthält also alle Annotationen die im Sinne der Ordnung des Annotationsverbands größer oder gleich sind.

Sei $A_2 : [\mu_2]$ das Kopfliteral einer Programmklausel und $A_1 : [\mu_1]$ ein Literal der Anfrage, wobei A und B Varianten sind, d. h. sind sind bis auf die Umbenennung von Variablen identisch. Bezeichne außerdem M' das Komplement einer Menge M. In der *ca-Resolution* kann $A_1 : [\mu_1]$ in der Anfrage eliminiert werden, wenn gilt:

$$\Uparrow \mu_1 \cap (\Uparrow \mu_2)' = \emptyset \tag{2.1}$$

Ist die Schnittmenge des *up-sets* von μ_1 und des Komplements des *up-sets* von μ_2 nicht leer, beschreibt genau diese Schnittmenge diejenigen Annotationen, die zumindest erfüllt sein müssen, um A_2 durch Resolution eliminieren zu können. Die *ca*-Resolution sammelt die gefundenen Annotationen eines Atoms und testet nach jedem Resolutionsschritt die Bedingung (2.1) in der allgemeineren Form $\mu_1 \leq \sqcup(\nu_1, \ldots, \nu_n)$, wobei ν_i die bisher gefundenen Annotationen sind. Die Randbedingungen der *ca*-Resolution sind keine Randbedingungen im Sinne des *CLP*-Schemas (*Constraint Logic Programming*, [JL87]). Die Unerfüllbarkeit verursacht keine Rückverfolgung (engl.: *backtracking*). Andererseits können sich die Randbedingungen in jedem Resolutionsschritt ändern, bis sie schließlich erfüllbar sind.

2.1.4 Die *well-founded*-Semantik

Die üblichen Semantiken der logischen Programmierung sind zweiwertig, d. h. eine Klausel ist entweder erfüllbar oder nicht erfüllbar. Allerdings gibt es Programme, denen unter Verwendung einer zweiwertigen Semantik kein eindeutiges Modell zugeordnet werden kann (Beispiel 2.6). Sie sind daher in der Praxis nur für eingeschränkte Klassen von Programmen einsetzbar. Durch die Verwendung einer dreiwertigen Logik auf der ontologischen Ebene kann die *well-founded*-Semantik hingegen allen Programmen mit Negation eine eindeutiges kleinstes Modell zuordnen. Daher ist sie als Semantik für allgemeine Programme geeignet.

Dieser Abschnitt präsentiert die *well-founded*-Semantik für annotierte Logik nach [KS92]. Die Darstellung beschränkt sich auf eine einsortige annotierte Logik. Sie lässt sich jedoch unmittelbar auf die mehrsortige Logik des letzten Abschnitts übertragen.

Die grundlegenden Begriffe der Interpretation und Modelle annotierter Programme wurden bereits im vorletzten Abschnitt eingeführt.

Beispiel 2.6
Das folgende annotiertes Programm über den Verband $\mathcal{FOUR}$ hat unter einer zweiwertigen Semantik kein eindeutiges Modell:

$$p : [t] \leftarrow \mathbf{not}(p : [t])$$

Die *well-founded*-Semantik [Fit91] behandelt negative und positive Information symmetrisch und vermeidet Unerfüllbarkeit durch eine dreiwertige Logik. Sie ist in der Lage, jedem logischen Programm ein Modell zuzuordnen. Im Beispiel 2.6 wird die Anfrage $p : [t]$ zu *unbekannt* ausgewertet.

Für jedes annotierte Programm P können wir einen Operator R_P angegeben, der eine Abbildung von r-Interpretationen auf r-Interpretationen ist.

Definition 2.11 (Fixpunktoperator R_P)
Seien I eine r-Interpretation und A Element der Herbrandbasis $B_{\mathcal{L}}$ einer Sprache $\mathcal{L}$. Dann ist

$$\begin{aligned} R_P(I)(A) \quad = \sqcup\{ \quad & f(\mu_1, \ldots, \mu_n) \mid \\ & A : [f(\mu_1, \ldots, \mu_n) \leftarrow B_1 : [\mu_1] \wedge \cdots \wedge B_n : [\mu_n] \wedge \\ & \mathbf{not}(B_{n+1} : [\mu_{n+1}]) \wedge \cdots \wedge \mathbf{not}(B_{n+m} : [\mu_{n+m}])\} \end{aligned}$$

eine Grundinstanz einer Klausel in P, so dass gilt $I \models^r (B_1 : [\mu_1] \wedge \cdots \wedge B_n : [\mu_n])$, und für alle $1 \leq j \leq m$ gilt $I \not\models^r B_{n+j} : [\mu_{n+j}]\}$.

Kifer und Subrahmanian [KS92] haben gezeigt, dass der Fixpunktoperator R_P ein eindeutiges kleinstes Modell besitzt, wenn P keine Klauseln mit negierten Literalen enthält.

Seien P ein annotiertes Programm (möglicherweise mit negierten Literalen) und I eine Interpretation. P kann wie folgt in ein negationsfreies Programm $G(P, I)$ transformiert werden:

1. Ist $A : [\mu] \leftarrow A_1 : [\mu_1] \wedge \cdots \wedge A_n : [\mu_n] \wedge \mathbf{not}(A_{n+1} : [\mu_{n+1}]) \wedge \cdots \wedge \mathbf{not}(A_{n+m} : [\mu_{n+m}])$ eine Grundinstanz von P, und für alle $1 \leq j \leq m$ gilt $\mu_{n+j} \not\leq I(A_{n+j})$, dann ist $A : [\mu] \leftarrow A_1 : [\mu_1] \wedge \cdots \wedge A_n : [\mu_n]$ in $G(P, I)$.

2. Außer den Klauseln aus 1. ist nichts in $G(P, I)$.

Da $G(P, I)$ ein negationsfreies Programm ist, hat der Operator $R_{G(P,I)}$ einen kleinsten Fixpunkt, der mit $\mathrm{lfp}(R_{G(P,I)})$ bezeichnet wird. Diesen ordnen wir als Operator $\mathcal{G}_P(I)$ einem annotierten Programm P zu.

Definition 2.12 (r-stabiles Modell)
Sei P ein annotiertes logische Programm. Die r-Interpretation I wird genau dann r-stabiles Modell von P *genannt, wenn $I = \mathcal{G}_P(I)$ gilt.*

Da der Operator $\mathcal{G}_P$ anti-monoton ist [Sub92], ist die Funktion $\mathcal{G}_P^2$, die $\mathcal{G}_P$ zweimal anwendet, monoton und hat einen Fixpunkt. Die Tatsache, dass $\mathcal{G}_P^2$ einen Fixpunkt hat, bedeutet allerdings nicht unbedingt, dass auch $\mathcal{G}_P$ einen Fixpunkt besitzt.

Beispiel 2.7 *Sei P ein Programm mit einer Klausel $p : [0.7] \leftarrow \mathbf{not}(p : [0.5])$ unter dem Verband der reellen Zahlen von 0 bis 1. $\mathcal{G}_P$ hat keinen Fixpunkt und besitzt deshalb kein r-stabiles Modell. Dagegen ist die Interpretation, die p 0.7 zuweist, ein Fixpunkt von $\mathcal{G}_P^2$, da $\mathcal{G}_P(I)(p) = 0$ und $\mathcal{G}_P^2(I)(p) = 0.7$. Die Interpretation, die p 0 zuweist, ist auch ein Fixpunkt von $\mathcal{G}_P^2$.*

Definition 2.13 (*Well-founded*-Semantik)
Die well-founded-*Semantik eines annotierten Programms P wird durch folgende Folgerungsrelation definiert:*

- $P \overset{wfs}{\models^r} A : [\mu]$ *gdw.* $\mu \leq lfp(\mathcal{G}_P^2)(A)$.
- $P \overset{wfs}{\models^r} \mathbf{not}(A : [\mu])$ *gdw.* $\mu \not\leq gfp(\mathcal{G}_P^2)(A)$.

Definition 2.14 (*Well-founded*-Modell)
Sei P ein annotiertes Programm über den Verband $\mathcal{T}$ und ein A ein Atom. Das well-founded-*Modell von A ist definiert als*

$$WM_P(A) = (T(A), U(A), F(A))$$

mit $T(A), U(A) und F(A)$ ist eine Partition des Verbands $\mathcal{T}$, so dass

1. $T(A) = \{\mu \in \mathcal{T} | \mu \leq lfp(\mathcal{G}_P^2)(A)\}$
2. $F(A) = \{\mu \in \mathcal{T} | \mu \not\leq gfp(\mathcal{G}_P^2)(A))\}$
3. $F(A) = \mathcal{T} - (T(A) \cap F(A))$

Beispiel 2.8 *Für Beispiel 2.7 ist demnach* $p : 0$ *ableitbar und* $p : [V]$ *mit* $V > 0.7$ *nicht ableitbar. Die Fakten* $p : [V]$ *mit* $0 < V \leq 0.7$ *sind weder ableitbar noch nicht ableitbar. Die Ableitbarkeit ist also im Sinne der* well-founded-*Semantik* unbekannt.

Die *well-founded*-Semantik ist allgemein als robuste und natürliche Semantik für alle logischen Programme anerkannt. Sie fällt zusammen mit dem kleinsten dreiwertigen stabilen Modell. Ihre Robustheit macht sie gut geeignet für die Anwendung in allgemeinen wissensbasierten Systemen, insbesondere für heterogene Systeme, in denen von vornherein nur schwer Aussagen über die Konsistenz des Gesamtsystems gemacht werden können. Die Nützlichkeit der *well-founded*-Semantik ist allerdings abhängig von ihrer effizienten Implementierung.

2.2 SLG-Resolution

Die SLG-Resolution[7] wurde von Chen und Warren [CW96, CW93, CSW95] vorgestellt. Mit ihr ist es möglich, das *well-founded*-Modell eines sicheren, normalen Logikprogramms[8] zu berechnen. Insbesondere hat sie folgende Eigenschaften:

- Die SLG-Resolution ist eine partielle Deduktionsprozedur[9], die aus mehreren Transformationen besteht.
- Die SLG-Resolution ist korrekt und vollständig für sichere Programme bezüglich aller dreiwertigen stabilen Modelle (einschließlich des partiellen *well-founded*-Modells).
- Sowohl die Berechnungsregel[10] als auch die Kontrollstrategie[11] für die Auswahl der auszuführenden Transformationen ist beliebig.

[7] *Linear resolution with selection function for general logic programs.*

[8] Normale Logikprogramme enthalten nur normale Programmklauseln. Eine Programmklausel ist normal, wenn sie die Form *Kopf* $\leftarrow$ *Rumpf* hat und im Kopf genau ein positives Literal enthält. Insbesondere sind annotierte Klauseln, wie sie hier definiert wurden, normal.

[9] Eine partielle Deduktionsprozedur ist eine Deduktionsprozedur, die bezüglich einer Anfrage definiert ist. Sie berechnet nicht das komplette Modell eines Programms, sondern nur ein partielles Modell, das zur Beantwortung der Anfrage ausreicht.

[10] Eine Berechnungsregel wählt aus den Literalen einer Anfrage genau eines aus. Dieses Literal wird im nächsten Resolutionsschritt zur Resolution herangezogen.

[11] Eine Kontrollstrategie kontrolliert, welche Klauseln wann zur Resolution mit der Anfrage herangezogen werden. Bei der SLG-Resolution bestimmt die Kontrollstrategie, wann welche Transformation angewendet wird.

- Sie vermeidet sowohl positive[12] als auch negative [13] Schleifen und terminiert immer für Programme, die die *bounded term size*-Eigenschaft [CW96] erfüllen (Abschnitt 2.3).

- Für die Negation unter der *well-founded*-Semantik hat die SLG-Resolution polynomiale Zeit- und Speicherkomplexität für funktionsfreie Programme.

Als konzeptueller Rahmen besteht die SLG-Resolution aus einer Reihe von Transformationen, durch die eine Anfrage zu einer Menge von Antwortklauseln reduziert wird. Sie spezifiziert nicht, in welcher Reihenfolge diese Transformationen durchgeführt werden. Dadurch ist sie unabhängig von der konkreten prozeduralen Implementierung. Zwei wichtige Transformationen sind COMPLETION und DELAYING. COMPLETION behandelt Unterziele, die vollständig ausgewertet wurden, so dass das negative Gegenstück des Unterziels resolviert werden kann. DELAYING verzögert die Auswertung negativer Literale, so dass die Berechnung trotz negativer Schleifen fortgeführt werden kann. Kopfliterale von Klauseln, die verzögerte Literale enthalten, gelten im Sinne der *well-founded*-Semantik als unbekannt. Ebenso werden Literale verzögert, die von einer unbekannten Lösung abhängig sind. Antwortklauseln, die als *unbekannt* gelten, können unter Umständen vereinfacht werden, wenn das betreffende Unterziel vollständig ausgewertet wurde.

Suchwald

SLG-Resolution kann als Suche in einem Suchbaum oder Suchwald betrachtet werden [CW93, CSW95]. Sei P ein Programm und A ein Unterziel. Wir konstruieren eine *Suchwald* für A bezüglich P. Der Suchwald besteht aus einer Menge von Bäumen, die jeweils ein Unterziel repräsentieren. Jeder Knoten eines Baumes im Wald wird mit einer Klausel markiert. Zu Beginn hat der Suchwald genau einen Baum, nämlich den Baum für Unterziel A. Dessen Wurzel ist mit der Klausel $A \leftarrow A$ markiert. Für jede Resolvente des Atoms A im Rumpf der Wurzelklausel mit einer Programmklausel aus P hat die Wurzel einen Nachfolger. Wenn ein Knoten im Baum von A mit einem Faktum[14] B markiert ist, dann ist B eine Antwort für A. Zwei Antworten, die Varianten sind, werden als identisch betrachtet. Sei v ein Knoten im Baum für Unterziel A, sei G die Klausel, mit der v markiert ist und B das gewählte Atom in G. Wenn der Suchwald für Unterziel B noch keinen Baum enthält, dann wird ein Baum für B hinzugefügt, dessen Wurzel mit $B \leftarrow B$ markiert wird. Für jede Antwort B' für B hat v einen Nachfolger, der mit

[12] z.B. $p \leftarrow p$
[13] z.B. $p \leftarrow \neg p$
[14] Ein Faktum ist eine Klausel mit leerem Rumpf.

der Resolvente von G mit B' bezüglich des gewählten Atoms B markiert ist. Wenn das gewählte Literal in G ein negatives grundinstantiiertes Literal der Form $\neg B$ ist, dann wird für B ein Baum erzeugt, falls der Suchwald noch keinen Baum für B enthält. Falls für B eine Antwort gefunden wird, wird jeder Knoten mit selektiertem $\neg B$ als gescheitert markiert. Wurde B vollständig ausgewertet und hat keine Antworten, dann ist $\neg B$ beweisbar, und jeder Knoten mit selektiertem $\neg B$ hat einen einzigen Nachfolger, der mit G' markiert wird, wobei G' durch Löschen von $\neg B$ entsteht. Dieser Prozess wird solange fortgeführt, bis kein neuer Knoten oder kein neuer Baum erzeugt werden kann. Dieses Vorgehen beruht allerdings auf der Annahme, dass das Unterziel eines negativen Literal vollständig ausgewertet ist, bevor das Ergebnis verarbeitet werden kann. Das trifft jedoch nur für stratifizierte Programme[15] zu. Im Allgemeinen können sehr wohl negative Schleifen auftreten. Damit auch solche Fälle bearbeitet werden können, gibt es bei der SLG-Resolution die Möglichkeit, Literale zurück zu stellen, was die Bearbeitung der restlichen Literale im Klauselrumpf erlaubt. Das Zurückstellen von Literalen wird durch die Transformation DELAYING erreicht. Im Resolutionsprozess werden nun diejenigen Literale zurückgestellt, deren Unterziele aufgrund von Schleifen nicht vollständig ausgewertet werden können. Dabei können Klauseln entstehen, die im Rumpf nur zurück gestellte Literale enthalten. Die Ableitbarkeit oder Beweisbarkeit des Kopfliterals einer solchen Klausel wird als *unbekannt* definiert. Es kann jedoch im weiteren Verlauf der Resolution vorkommen, dass eine solche Antwort vereinfacht werden kann, wenn das zu einem zurück gestellten Literal gehörige Unterziel doch noch vollständig ausgewertet werden konnte.

Beispiel 2.9 *Gegeben sei folgendes Programm und die Anfrage* $\leftarrow w(a)$*:*

$w(X) \leftarrow m(X,Y), \neg w(Y), p(Y)$
$m(a,b)$
$m(b,c)$
$m(c,b)$
$p(b)$

Abbildung 2.3 zeigt den Zustand des Suchbaums, nachdem alle negativen Literale, die nicht ausgewertet werden konnten, verzögert wurden. Da das Unterziel $p(b)$ positiv beantwortet werden kann, werden zwei Antwortklauseln $w(a) \leftarrow \neg w(b) \mid$ und $w(c) \leftarrow \neg w(b) \mid$ in den Unterzielen $w(a)$ und $w(c)$ gebildet. Das Unterziel $p(c)$ hingegen scheitert, wodurch die Berechnung des Unterziels $w(b)$ ohne Antworten abgeschlossen werden kann. Dies erlaubt, die Antwortklauseln der Unterziele $w(a)$ und $w(c)$ zu vereinfachen, was zur positiven Beantwortung dieser beiden Unterziele führt.

[15] In einem stratifizierten Programm gibt es keine Prädikatensymbole, die gleichzeitig gegenseitig und negativ von einander abhängen.

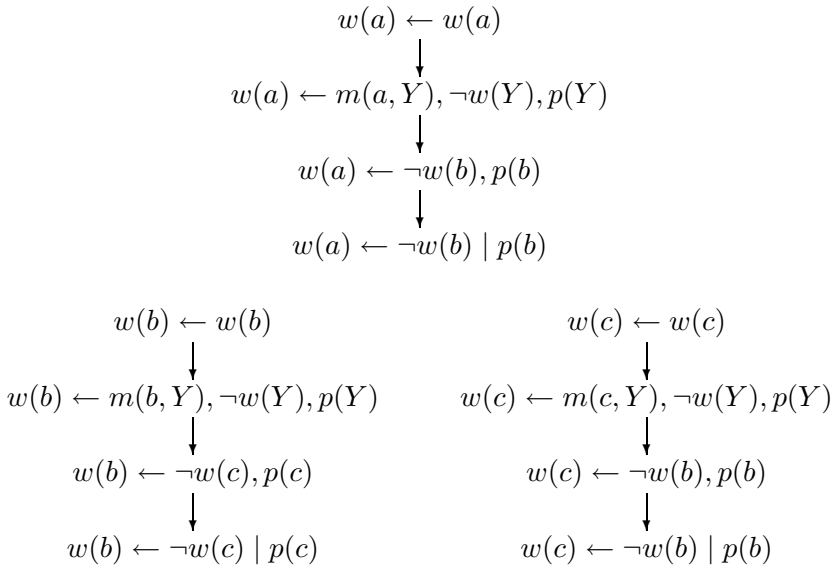

Abbildung 2.3: Suchwald für die Anfrage $w(a)$ im Beispiel 2.9

Im Allgemeinen ist das *well-founded*-Modell dreiwertig. Antwortklauseln können verzögerte Literale enthalten, die nicht durch Vereinfachungstransformationen eliminiert werden können. Diese Antworten sind im *well-founded*-Modell weder beweisbar noch widerlegbar.

2.2.1 Transformationen der SLG-Resolution

Definition 2.15 (X-Klausel)
Eine X-Klausel *ist eine Klausel der Form:*

$$A \leftarrow D \mid B$$

wobei A ein Atom, D eine Konjunktion von Literalen (wobei negative Literale grundinstantiiert sind) und B eine Konjunktion von Literalen ist. Die Literale in D werden zurückgestellte *oder* verzögerte Literale *genannt. Ist B leer, wird die X-Klausel auch* Antwortklausel *genannt.*

Eine Programmklausel entspricht einer X-Klausel, bei der D leer ist. Aus dem Blickpunkt der deklarativen Semantik wird eine X-Klausel als gewöhnliche Klausel betrachtet, deren Rumpf die Konjunktion von D und B bildet. Das heißt, | ist eine Markierung, die ausschließlich zu Kontrollzwecken dient.

Sei eine X-Klausel $A \leftarrow D \mid B$ gegeben, bei der B nicht leer ist, dann wählt eine *Berechnungsregel* R genau ein Literal aus B, das sog. *gewählte* Literal.

Definition 2.16 (SLG-Resolution)
Sei G eine X-Klausel $A \leftarrow D \mid L_1 \wedge \cdots \wedge L_n$ mit $n > 0$, und L_i sei das gewählte Literal. Sei C eine X-Klausel ohne verzögerte Literale. C', eine Klausel der Form $A' \leftarrow L'_1 \wedge \cdots \wedge L'_m$, sei eine Variante von C, so dass G und C' keine gemeinsamen Variablen besitzen. Die Klausel

$$(A \leftarrow D \mid L_1 \wedge \cdots \wedge L_{i-1} \wedge L'_1 \wedge \cdots \wedge L'_m \wedge L_{i+1} \wedge \cdots \wedge L_n)\theta$$

ist die SLG-Resolvente *von G mit C, wobei θ der allgemeinste Unifikator von L_i und A' ist.*

Die SLG-Resolution wird entweder zur Resolution mit Programmklauseln oder mit solchen Antwortklauseln benützt, die keine verzögerten Literale besitzen.

Bei Antwortklauseln mit verzögerten Literalen werden die relevanten Variablenbindungen im Kopf mittels *SLG-Faktorisierung* propagiert. Die verzögerten Literale im Rumpf werden jedoch nicht propagiert.

Definition 2.17 (SLG-Faktorisierung)
Seien G eine X-Klausel der Form $A \leftarrow D \mid L_1 \wedge \cdots \wedge L_n$ mit $n > 0$ und L_i das gewählte Atom. Sei C eine Antwortklausel. C', eine Klausel der Form $A' \leftarrow D' \mid$, sei eine Variante von C, so dass C und C' keine gemeinsamen Variablen besitzen. Ist D' nicht leer und sind L_i und A' unifizierbar mit einem allgemeinsten Unifikator θ, dann ist der SLG-Faktor *von G und C definiert durch*

$$(A \leftarrow D \wedge L_i \mid L_1 \wedge \cdots \wedge L_{i-1} \wedge L_{i+1} \wedge \cdots \wedge L_n)\theta$$

Jeder Knoten eines Baumes des SLG-Suchwalds, der keine Wurzel ist, hat einen *Status*, der entweder *new, answer, active, floundered* oder *disposed* sein kann. Der anfängliche Status eines neu erzeugten Knotens ist *new*. Die Verarbeitung eines solchen Knotens mittels Transformationen kann dessen Status beeinflussen. Er wird zu

- *answer*, wenn die Klausel, die den Knoten markiert, eine Antwortklausel ist;
- *floundered*, wenn das gewählte Literal negativ und nicht grundinstantiiert ist;
- *active*, wenn der Status des gewählten Literals nicht *floundered* ist und das zugehörige Unterziel nicht vollständig ausgewertet ist;
- *disposed*, wenn alle möglichen Nachfolger gebildet wurden[16];

[16]Das bedeutet, dass die Klausel für die Resolution nicht mehr nützlich ist.

Wenn ein Knoten den Status *floundered* erhält, wird die Bearbeitung abgebrochen, da kein sicheres Programm vorliegt bzw. das Programm bezüglich der gestellten Anfrage nicht sicher ist.

Sei eine Anfrage in Form eines Atoms A gegeben. Dann besteht der Suchwald zu Beginn der Resolution lediglich aus einem einzigen Baum für A. Dieser Baum hat eine Wurzel, die mit $A \leftarrow A$ markiert ist und einen Nachfolger für jede SLG-Resolvente von $A \leftarrow A$ mit einer Programmklausel hat.

Jede der Transformationen verändert den Suchwald. Transformationen (i) - (iii) verarbeiten die X-Klauseln eines Knotens mit Status *new*, der entweder *answer*, *floundered* oder *active* gesetzt wird. Transformation (iii) löst die Erzeugung eines neuen Unterziels aus, falls es noch nicht im Wald vorhanden ist. Sei G die X-Klausel eines neuen Knotens v.

(i) NEW ANSWER. Ist G eine Antwortklausel, wird der Status von v zu *answer* geändert.

Ist G keine Antwortklausel, dann sei L das gewählte Literal in G.

(ii) FLOUNDERING. Ist L ein negatives Literal, welches nicht grundinstantiiert ist, wird der Status von v zu *floundered*.

(iii) NEW ACTIVE. Ist L ein Atom B oder ein grundinstantiiertes negatives Literal $\neg B$, wird der Status von v zu *active*. Falls für B kein Baum im Suchwald existiert, wird ein Baum erzeugt, dessen Wurzel mit $B \leftarrow B$ markiert ist. Für jede SLG-Resolvente von $B \leftarrow B$ mit einer Programmklausel hat die Wurzel einen Nachfolger.

Sei G die Klausel eines Knotens v mit Zustand *active* und L das gewählte Literal in G.

(iv) POSITIVE RETURN. Sei L ein Atom B und C eine Antwortklausel der Form $H \leftarrow D \mid$ im Baum für B, wobei H noch nicht mit G resolviert wurde. Dann wird ein Nachfolger für v generiert, der mit der SLG-Resolvente von G mit C markiert wird, falls D leer ist. Ist D nicht leer, wird v mit dem SLG-Faktor von G und C markiert.

Transformation (v) löst ein grundinstantiiertes negatives Unterziel mit der *Negation-as-Failure*-Regel, wenn feststeht, ob das entsprechende positive Unterziel abgeleitet werden kann oder nicht. Gibt es jedoch nur Antworten mit zurückgestellten Literalen, ist die Ableitbarkeit des Unterziels im Sinne der *well-founded*-Semantik *unbekannt*.

(v) NEGATIVE RETURN. Sei das gewählte Literal L der X-Klausel G eines Knotens mit Zustand *active* ein grundinstantiiertes negatives Literal $\neg B$, dann

- wird der Status von v zu *disposed*, wenn B eine Antwort ohne verzögerte Literale hat (NEGATION FAILURE-R);
- wird für v eine Nachfolger erzeugt, der mit G' markiert wird, wenn B vollständig ausgewertet ist und keine Antworten hat. G' entsteht aus G durch Löschen von L. Der Status von v wird zu *disposed* (NEGATION SUCCESS-R).

(vi) DELAYING. Trifft keiner der beiden Fälle in (v) zu,d. h. es gibt nur Antworten mit verzögerten Literalen, wird für v ein Nachfolger erzeugt, der mit G' markiert wird. G' entsteht aus G, indem L zur Menge der verzögerten Literale hinzugefügt wird. Der Status von v wird zu *disposed*.

Ob eine Menge von Unterzielen vollständig ausgewertet wurde, kann anhand der Bäume für die Unterziele im Suchwald festgestellt werden. Mengen werden deshalb betrachtet, da es durch Schleifen vorkommen kann, dass mehrere Unterziele zyklisch voneinander abhängen. Für diese Unterziele ist es nicht möglich, einzeln die Vollständigkeit der Auswertung festzustellen. Alle an einem Zyklus beteiligten Unterziele können nur gleichzeitig in den Zustand *vollständig* gelangen. Folgende Definition stellt sicher, dass dabei keine neuen Antworten für die beteiligten Unterziele enstehen können, wodurch das Ergebnis jedes beteiligten Unterziels beeinflusst werden könnte.

Definition 2.18 (Vollständige Unterzielauswertung)
Seien P ein Programm und Q ein Anfrageatom. Gegeben sei der Suchwald zu einem beliebigen Zeitpunkt der Berechnung von Q bezüglich P und eine Menge $\mathcal{A}$ von Unterzielen. $\mathcal{A}$ ist vollständig ausgewertet*, wenn für jedes Unterziel $A \in \mathcal{A}$ ein Baum im Suchwald existiert, dessen Wurzel mit $A \leftarrow A$ markiert ist und der folgende Bedingungen erfüllt:*

- *Die Wurzel hat für jede SLG-Resolvente G' des Rumpfatoms A von $A \leftarrow A$ mit einer Pogrammklausel einen Nachfolger, der mit G' markiert ist.*
- *Bei jedem Knoten v, der nicht die Wurzel ist und mit einer X-Klausel G markiert ist, dessen gewähltes Literal ein Atom B ist, ist B entweder als* vollständig *markiert oder $B \in \mathcal{A}$ und es existiert für jede Antwortklausel C der Form $B' \leftarrow D \mid$ ein Nachfolger, der mit der SLG-Resolvente von G mit C markiert ist, falls D leer ist, und mit dem SLG-Faktor von G mit C markiert ist, falls D nicht leer ist.*
- *Bei jedem Knoten v, der nicht die Wurzel ist und mit einer X-Klausel G mit gewähltem Literal $\neg B$ markiert ist, ist B grundinstantiiert. Außerdem gilt: Entweder hat B eine Antwort und v ist ein gescheitertes Blatt, oder B ist* vollständig ausgewertet *und hat keine Antworten. In*

diesem Fall hat v genau einen Nachfolger, der mit G' markiert ist. G' entsteht aus G durch Löschen von $\neg B$. Oder aber B hat eine Antwort mit verzögerten Literalen, dann hat v genau einen Nachfolger, der mit G' markiert wird, wobei G' aus G entsteht, indem $\neg B$ zur Menge der verzögerten Literale hinzugefügt wird.

Erst wenn für ein Unterziel A feststeht, dass es vollständig ausgewertet wurde und keine Antworten vorhanden sind, können negative Literale $\neg A$ erfolgreich resolviert werden.

(vii) COMPLETION. Sei $\mathcal{A}$ eine nicht leere Menge von Unterzielen, die vollständig ausgewertet ist. Für jedes Unterziel $A \in \mathcal{A}$ wird der Status jedes Knotens, der kein Antwortknoten ist, zu *disposed*, und A wird als *vollständig* markiert.

(viii) SIMPLIFICATION. Sei A ein Unterziel und G eine Antwortklausel der Form $H \leftarrow D \mid$ mit einem verzögerten Literal L in der Konjunktion D, wobei L entweder $\neg A$ ist oder positiv ist und von A subsumiert wird. Falls L erfolgreich ist[17], hat G einen Nachfolger G', der durch Löschen von L entsteht. Scheitert hingegen L, hat G keinen Nachfolger und wird zu *disposed*.

(ix) ANSWER COMPLETION. Sei A ein Unterziel, das vollständig ausgewertet wurde. Sei G eine Antwortklausel für A, die ein positives verzögertes Literal A' enthält, wobei A' von A subsumiert wird. Existiert im Baum für A keine Antwortklausel außer G, die A' subsumiert, wird G mit *disposed* markiert.

Die Vereinfachung verzögerter positiver Literale erzeugt keine Variablenbindungen. Diese werden nämlich bereits durch SLG-Faktorisierung propagiert, wenn durch **POSITIVE RETURN** das positive Literal verzögert wird.

Theorem 2.2 (Korrektheit und Vollständigkeit) *SLG-Resolution ist korrekt und vollständig für sichere, normale Programme bezüglich der* well-founded-*Semantik.*

Die Beweise werden hier ausgelassen und sind ausführlich in [CW96] beschrieben.

2.2.2 SLG-Resolution für annotierte Logik

SLG-Resolution kann mit einigen Modifikationen auf Programme der annotierten Logik übertragen werden. Der wesentliche Schritt ist, die SLG-

[17] D. h. L ist negativ und A hat keine Antworten, oder L ist positiv und A hat Antworten.

Resolution um die Reduktionsregel der annotierten Logik zu ergänzen. Außerdem müssen SLG-Resolution und SLG-Faktorisierung für annotierte Atome angepasst werden. Diese Ergänzungen erhalten jedoch alle Eigenschaften der SLG-Resolution. Dieses Kapitel beschränkt sich wieder auf die gewöhnliche annotierte Logik. Wie im letzten Abschnitt des vorigen Kapitels lassen sich alle Definitionen direkt auf die mehrsortige annotierte Logik übertragen.

Annotationen können in der Resolution nicht wie gewöhnliche Argumentstellen in Atomen behandelt werden. Das liegt an der Semantik der annotierten Logik: Ein annotiertes Atom $A : [\mu]$ subsumiert alle Atome $A' : [\mu']$, für die A' spezieller als A und $\mu' \leq \mu$ ist. Das heißt, die Annotation eines annotierten Atoms $A : [\mu]$ repräsentiert die Menge $\mathcal{M} = \{\mu' \mid \mu' \leq \mu\}$. Diese Menge wird als *down-set* von μ bezeichnet. Die SLD-artige Beweisprozedur für annotierte logische Programme, die in [Sub92] entwickelt wurde, setzt voraus, dass für alle Klauseln mit unifizierbaren Köpfen Reduzierte vorliegen. Die Beschränkung der Annotation, die dann bei der Resolution entsteht (siehe Abschnitt 2.1.3), wird als Randbedingung in die Resolvente aufgenommen. Da diese Resolution im Beweis Klauseln verwendet, die nicht im Originalprogramm vorhanden sind, handelt es sich nicht um eine lineare Resolution im eigentlichen Sinne. Die Notwendigkeit zur Bildung von Reduzierten aus Programmklauseln kann vermieden werden, indem Reduzierte dynamisch und nur für die Antworten von Unterzielen, also Einheitsklauseln, gebildet werden. Dies bietet sich hier in besonderem Maße an, da die SLG-Resolution sowieso unterzielorientiert arbeitet. Die Verarbeitung von Annotationsrandbedingungen wird überflüssig, wenn man sicherstellt, dass die Randbedingung, die in der Resolvente enstehen, vernachlässigt werden können. Dies ist der Fall, wenn in der Resolution nur Randbedingungen entstehen können, die positiv beantwortet werden, und man verlangt, dass Annotationsvariablen im Rumpf nur einmal auftauchen. Dann nämlich kann es für jede Annotationsvariable im Rumpf höchstens eine Randbedingung geben, welche trivial lösbar ist. Angenommen alle Programmklauseln eines Programm genügen der zweiten Bedingung, dann können gültige Resolutionen durch folgende Definitionen erreicht werden:

Definition 2.19 (A-Unifizierbarkeit)
Seien μ und ν Annotationen. μ heißt A-unifizierbar *mit ν, wenn gilt: μ ist eine v-Annotation, oder μ und ν sind c-Annotationen und es gilt: $\nu \geq \mu$*

Beachte: A-Unifizierbarkeit ist keine symmetrische Relation. Das liegt daran, dass die Subsumptionsrelation des Annotationsverbands nicht kommutativ ist.

Beispiel 2.10 *Gegeben sei ein annotiertes Programm mit den Klauseln unter dem Verband $\mathcal{FOUR}$:*

$A : [V] \leftarrow B : [V] \quad (*)$

$A : [t] \leftarrow$
$B : [f] \leftarrow$

Die Annotation der Anfrage $\leftarrow A : [\top]$ ist nicht A-unifizierbar mit dem Kopf von $(*)$. Würde man die Anfrage mit $(*)$ zu $A : [c] \leftarrow B : [c]$ resolvieren[18], würde die Anfrage negativ beantwortet werden. Das liegt daran, dass die Klausel $(*)$ als Vertreter einer Menge von Klauseln anzusehen ist, nämlich der Menge aller Klauseln $A : [V] \leftarrow B : [V]$ mit $V \in \mathcal{FOUR}$. Damit alle Resolventen, die logische Konsequenzen sind, gebildet werden können, müsste man $(*)$ durch diese Menge ersetzen. Dann ist die Annotation der Anfrage mit allen Annotationen der Klauselköpfe A-unifizierbar, und die Anfrage kann positiv beantwortet werden. Da nur bei endlichen Verbänden diese Klauselmenge explizit bestimmt werden kann, muss ein anderer Weg gewählt werden, um A-Unifizierbarkeit zu erreichen. Bei der SLG-Resolution wird ein Atom im Klauselrumpf als Unterziel betrachtet. Es wird getrennt ausgewertet, und alle anfallenden Antworten für das Unterziel werden mit der Klausel, in der das Atom auftrat, resolviert. Damit alle möglichen Resolventen gebildet werden, wertet man bei annotierten Programmen das Unterziel immer mit v-Annotation aus, anstatt ein Unterziel mit c- oder t-Annotation direkt mit Programmklauseln zu resolvieren. Alle Antworten werden gesammelt und mittels der Reduktionsregel verarbeitet. Es werden schließlich nur die Antworten weitergegeben, die die eigentliche Annotation des Unterziels subsumieren. Dadurch wird obige Anfrage zu $A : [V]$, ergänzt um die Randbedingung $V \geq \top$.

Definition 2.20 (Annotiertes Unterziel mit Randbedingung)
Ein annotiertes Unterziel mit Randbedingung *ist ein Ausdruck*

$$A : V \parallel \xi$$

wobei $A : V$ ein v-annotiertes Atom (d.h. V ist die v-Annotation) und ξ eine Annotations-Randbedingung ist.

Intuitiv repräsentiert die v-Annotation einen Platzhalter für die Annotationen aller Instanzen von A. Die Annotationsrandbedingung ξ stellt sicher, dass nur diejenigen mit einer Substitution assoziierten Annotationen als Antwort zugelassen werden, die ξ erfüllen.

In obigem Beispiel ist das Atom $A : [V]$ einerseits A-unifizierbar mit $(*)$, was zur Antwort $A : [f]$ führt, andererseits mit dem Faktum $A : [t]$, was die Antwort $A : [t]$ liefert. Zusammen mit der ersten Antwort unter Verwendung der Reduktionsregel ergibt sich das Ergebnis $A : [c]$. Für diese Antwort wird die Annotations-Randbedingung zu *wahr* ausgewertet, was schließlich zur positiven Beantwortung der Anfrage führt.

[18] Dies wird jedoch durch obige Definition ausgeschlossen.

Definition 2.21 (Antwort und Partielle Antwort) *Sei $A : V \parallel \xi$ ein annotiertes Unterziel mit Randbedingung. Dann ist jedes annotierte Atom $B : \mu$, das mittels eines Unifikators φ mit A unifiziert werden kann, eine* partielle Antwort *zum Unterziel A. Erfüllt μ die Annotations-Randbedingung ξ, dann ist $B : \mu$ eine* Antwort.

Ein Unterziel sammelt demnach Antworten, die auch partielle Antworten sein können. Durch Anwendung der Reduktionsregel können im Laufe des Deduktionsprozesses aus partiellen Antworten solche Antworten entstehen, die die Randbedingung des Unterziels erfüllen.

Definition 2.22 (A-Unifikator)
Seien $\leftarrow A : [\mu_A]$ eine Anfrage und $B : [\mu_B]$ der Kopf einer annotierten Klausel und μ_A und μ_B A-unifizierbar. Der A-Unifikator *ist eine Substitution, die wie folgt gebildet wird:*

- *Ist μ_A eine v-Annotation, dann ist der A-Unifikator $\{\mu_A/\mu_B\}$.*
- *Anderenfalls sind μ_A und μ_B c-Annotationen mit $\mu_B \geq \mu_A$, und der A-Unifikator ist leer.*

Definition 2.23 (Annotierte X-Klausel)
Eine annotierte X-Klausel *ist eine Klausel der Form*

$$A \leftarrow D \mid B,$$

wobei A eine annotiertes Atom ist und D und B Konjunktionen von c- oder v-annotierten Literalen sind. Die Literale in D werden verzögerte Literale *genannt. Ist B leer, wird die annotierte X-Klausel auch* annotierte Antwortklausel *genannt.*

Aus deklarativem Blickpunkt ist eine annotierte X-Klausel eine gewöhnliche annotierte Klausel, deren Rumpf aus der Konjunktion von D und B besteht. $\mid$ ist eine syntaktische Markierung zur Trennung der verzögerten Literale von den regulären Literalen.

Definition 2.24 (Annotierte SLG-Resolution)
Sei G eine annotierte X-Klausel der Form $A \leftarrow D \mid L_1 : [\mu_1] \wedge \cdots \wedge L_n : [\mu_n]$ mit $n > 0$, und $L_i : [\mu_i]$ sei das gewählte Literal. Sei C eine annotierte X-Klausel ohne verzögerte Literale und C' von der Form $H' : [\mu'] \leftarrow L'_1 : [\mu'_1] \wedge \cdots \wedge L'_m : [\mu'_m]$, sei eine Variante von C, so dass C und C' keine gemeinsame Variablen besitzen. G ist resolvierbar mit C, falls L_i und H' unifizierbar und μ_i und μ' A-unifizierbar sind. Die Klausel

$$\begin{array}{lll} (A & \leftarrow D \mid & L_1 : [\mu_1] \wedge \cdots \wedge L_{i-1} : [\mu_{i-1}] \wedge \\ & & L'_1 : [\mu'_1] \wedge \cdots \wedge L'_m : [\mu'_m] \wedge \\ & & L_{i+1} : [\mu_{i+1}] \wedge \cdots \wedge L_n : [\mu_n])\theta\lambda \end{array}$$

ist die SLG-Resolvente *von G mit C, wobei θ der allgemeinste Unifikator von L_i und H' ist und λ der A-Unifikator von μ' und μ_i ist.*

Definition 2.25 (Annotierte SLG-Faktorisierung)
Sei G eine annotierte X-Klausel der Form $A \leftarrow D \mid L_1 : [\mu_1] \wedge \cdots \wedge L_n : [\mu_n]$ mit $n > 0$, und $L_i : [\mu_i]$ sei das gewählte Atom. Sei C eine annotierte Antwortklausel und C' von der Form $H' : [\mu'] \leftarrow D' \mid$ sei eine Variante von C, so dass C und C' keine gemeinsame Variablen besitzen. Ist D' nicht leer, und sind L_i und H' unifizierbar mit einem allgemeinsten Unifikator θ, und sind μ_i und μ' A-unifizierbar mit einem A-Unifikator λ, dann ist der annotierte SLG-Faktor *von G und C gegeben durch*

$$(A \leftarrow D, L_i : [\mu_i] \mid \begin{array}{l} L_1 : [\mu_1] \wedge \cdots \wedge L_{i-1} : [\mu_{i-1}] \wedge \\ L_{i+1} : [\mu_{i+1}] \wedge \cdots \wedge L_n : [\mu_n])\theta\lambda. \end{array}$$

Die Transformationen der SLG-Resolution können weitgehend übernommen werden, dabei werden sie lediglich um die Verarbeitung von Annotationen ergänzt. Bei den Änderungen der folgenden Transformationen geht es einerseits darum, ein neues Unterziel statt mit c-Annotation mit v-Annotation auszuwerten, andererseits die Antworten des Unterziels nur dann zu propagieren, wenn ihre Annotation die Randbedingung des Unterziels übersteigt.

(iii) NEW ACTIVE Ist L ein Atom $B : [\mu]$ oder ein grundinstantiiertes negatives Literal $\neg B : [\mu]$, dann wird der Status von v zu *active*. Falls für $B : [\mu]$ kein Baum im Suchwald existiert, wird ein Baum erzeugt, dessen Wurzel mit $B : [\nu] \leftarrow B : [\nu]$ markiert ist. Die Annotation ν ist eine v-Annotation. Für jede SLG-Resolvente von $B : [\nu] \leftarrow B : [\nu]$ mit einer Programmklausel hat die Wurzel einen Nachfolger.

Sei G die Klausel eines Knotens v mit Zustand *active* und L das gewählte Literal in G.

(iv) POSITIVE RETURN. Sei L ein Atom $B : [\mu]$ und C eine Antwortklausel der Form $H : [\nu] \leftarrow D \mid$ im Baum für $B : [\mu]$ mit $\nu \geq \mu$. Wurde $H : \nu$ noch nicht mit G resolviert, dann wird ein Nachfolger für v generiert, der mit der SLG-Resolvente von G und C markiert wird, falls D leer ist. Ist D nicht leer, wird v mit dem SLG-Faktor von G und C markiert.

(v) NEGATIVE RETURN. Sei das gewählte Literal L der X-Klausel G eines Knotens mit Zustand *active* ein grundinstantiiertes negatives Literal $\neg B : [\mu]$, dann

- wird der Status von v zu *disposed*, wenn das Unterziel $B : [\mu]$ eine Antwort ohne verzögerte Literale hat und die Annotation dieser Antwort größer oder gleich μ ist (NEGATION FAILURE-R);

- wird für v ein Nachfolger erzeugt, der mit G' markiert wird, wenn $B : [\mu]$ vollständig ausgewertet ist und keine Antworten hat, deren Annotation größer oder gleich μ ist. G' entsteht aus G durch Löschen von L. Der Status von v wird zu *disposed* (NEGATION SUCCESS-R).

Für jeden Baum eines Unterziels $B : [\mu]$ im Suchwald gilt grundsätzlich, dass die Wurzel mit der Klausel $B : [\nu] \leftarrow B : [\nu]$ markiert wird. Dies stellt sicher, dass alle Resolventen, die unter Vernachlässigung der Annotation gebildet würden, auch hier gebildet werden. Die Korrektheit wird dadurch gewahrt, dass nur die Antworten resolviert werden, die die Annotation des Unterziels subsumieren.

Eine weitere Transformation wird zur Bildung der Reduzierten benötigt:

(x) GAP-REDUCTANTS. Seien $A : [\mu_1]$ und $A' : [\mu_2]$ zwei annotierte Antwortklauseln, wobei A' und A mit einem allgemeinsten Unifikator θ unifizierbar sind. Dann wird eine Antwortklausel $A\theta : [\sqcup(\mu_1, \mu_2)]$ gebildet. Diese Klausel markiert einen Knoten, der als direkter Nachfolger der Wurzel erzeugt wird. Werden A oder A' durch die neue Antwortklausel subsumiert, werden A oder A' gelöscht.

Theorem 2.3 (Korrektheit und Vollständigkeit)
Annotierte SLG-Resolution ist korrekt und vollständig für sichere Programme bezüglich der well-founded-*Semantik.*

Korrektheit und Vollständigkeit der annotierten SLG-Resolution ergeben sich aus der Korrektheit und Vollständigkeit der SLG-Resolution. Die Verallgemeinerung annotierter Unterziele stellt einerseits sicher, dass alle Resolventen gebildet werden, die zur Beantwortung des Unterziels beitragen können, andererseits werden nur diejenigen Antworten propagiert, die die Annotation des Unterziels erfüllen. Dies entspricht einem Resolutionsschritt in der *ca*-Resolution [LL96], bei dem ein Literal in der Resolvente eliminiert wird. Die annotierte SLG-Resolution gewährleistet durch die Eliminierung eines Atoms bei jeder Anwendung, dass die Eigenschaften der SLG-Resolution bezüglich der Wahl der Berechnungsregel erhalten bleiben.

2.2.3 Einbindung von Informationsquellen

Um Fakten aus externen Informationsquellen in die Mediatorumgebung zu integrieren, ist ein Mechanismus und ein sprachliches Mittel notwendig um dieses externe Wissen im Rahmen eines Mediatorprogramms anzusprechen. Im Mediatorsystem KOMET werden Informationsquellen als Domänen über Randbedingungen (engl.: *constraint domain*) modelliert. Ihr Wissen wird mit Hilfe von Randbedingungen (engl.: *constraints*) in ein Programm eingebunden[19]. Dabei haben diese Randbedingungen nicht nur eine beschränkende

Funktion sondern erzeugen vorrangig Variablenbindungen. In Analogie zur klassischen Logik mit Randbedingungen (*Constraint Logic Programming (CLP)*, [JL87]) kann eine Informationsquelle als Löser von partiellen Randbedingungen betrachtet werden. Es ist die Aufgabe des Mediators eine komplexe Randbedingungsklausel in partielle Randbedingungen zu zerteilen, die von einzelnen Informationsquellen beantwortet werden können. Verfügt eine Informationsquelle nicht über die Fähigkeit komplexe Randbedingungen (z.B. Konjunktionen von Randbedingungsrelationen) zu verarbeiten, wird dies, soweit möglich, vom Mediator übernommen. Genauso muss der Mediator die Ergebnisse in angemessener Weise im Inferenzprozess verarbeiten. Zur Steuerung des Aufrufs externer Wissensquellen wird Metawissen über mögliche Bindungsmuster für die Relationen der Domänen eingesetzt.

Definition 2.26 (Randbedingungsdomäne)
Sei eine Menge $\mathcal{D}$ von Konstanten, eine Menge $\mathcal{F}$ von Funktionen, die über $\mathcal{D}$ definiert sind und eine Menge $\mathcal{R}$ von Relationen über $\mathcal{D}$ gegeben, dann ist eine Randbedingungsdomäne $\Sigma_\mathcal{D}$ definiert als das Tupel (F,R).

Ein *Randbedingungsterm* kann in der üblichen Weise aus den Elementen in $\mathcal{D}$ und $\mathcal{F}$ gebildet werden. Eine *atomare Randbedingung* ist eine Relation $r(a_1, \ldots, a_n)$ mit $r \in \mathcal{R}$ und a_i ist ein Randbedingungsterm.

Definition 2.27 (Randbedingung)
Eine Randbedingung Ξ über $\Sigma_\mathcal{D}$ ist eine logische Formel erster Stufe, die in $\Sigma_\mathcal{D}$ entweder zu wahr oder falsch auswertet.

Eine Randbedingungsdomäne $\Sigma_\mathcal{D}$ erfüllt eine Randbedingung Ξ (auch $\Sigma_\mathcal{D} \rhd \Xi$), wenn

1. Ξ eine atomare Randbedingung $r(a_1, \ldots, a_n)$ ist und $(a_1, \ldots, a_n) \in r$
2. Ξ ist $\neg A$, und $\Sigma_\mathcal{D} \not\rhd A$
3. Ξ ist $(A \wedge B)$, und $\Sigma_\mathcal{D} \rhd A$ und $\Sigma_\mathcal{D} \rhd B$
4. Ξ ist $(A \vee B)$, und $\Sigma_\mathcal{D} \rhd A$ oder $\Sigma_\mathcal{D} \rhd B$

Definition 2.28 (Annotierte Klausel mit Randbedingungen)
Eine annotierte Klausel mit Randbedingungen *C ist ein Ausdruck der Form*

$$P_0 : \mu_0 \leftarrow \Xi \parallel P_1 : \mu_1 \wedge \ldots \wedge P_n : \mu_n$$

wobei $\parallel$ ein syntaktisches Symbol zur Trennung der normalen GAP-Literale von der Randbedingung Ξ der Klausel ist.

[19] Die Randbedingungen, von denen hier die Rede ist, sind zu unterscheiden von den Annotationsrandbedingungen, die in Abschnitt 2.1.3 behandelt worden sind. Die beiden Typen von Randbedingungen haben unterschiedliche Funktion in annotierten logischen Programmen.

Eine detaillierte Ausführung über die formalen Aspekte der Einbettung von Randbedingungen in annotierte logische Programme kann [Sch95] entnommen werden. Eine Randbedingungsdomäne wird charakterisiert durch die Menge von Funktionen und Relationen, die sie zur Verfügung stellt. Diese Relationen und Funktionen repräsentieren das Wissen der Informationsquelle. Sie sind gewöhnlich im Exportschema einer Wissensquelle definiert, das somit die Schnittstelle zum Zugriff auf die Quelle darstellt. Im Gegensatz zu Interpretern für Logikprogrammierung mit Randbedingungen ($CLP(X)$) über eine einzelne, feste Domäne X, können in KOMET Randbedingungen aus Relationen und Funktionen mehrerer Domänen bestehen. Jede Domäne wird durch einen eindeutigen Bezeichner identifiziert. Funktionen und Relationen einer Domäne werden durch dessen Bezeichner qualifiziert.

Im Allgemeinen können die durch Informationsquellen realisierten Relationen nicht wie vollwertige Prädikate im logischen Programm ausgewertet werden. Das liegt an Einschränkungen denen das Informationsangebot der Quellen, aus welchen Gründen auch immer, unterworfen ist. Um solche Einschränkungen zu modellieren und in der Programmauswertung zu berücksichtigen werden für die Relationen der Informationsquellen Bindungsmuster definiert. Diese Bindungsmuster beschreiben, welche Auswertungen die hinter einem Relationensymbol liegende Informationsquelle zulässt. Ein Bindungsmuster oder *Modus* besteht aus einem Tupel von Argumentmodi mit der Stelligkeit der Relation. Ein Argumentmodus ist entweder + oder ?. + besagt, dass bei der Auswertung die entsprechende Argumentstelle an eine Konstante gebunden sein muss, während ? ausdrückt, dass diese Argumentstelle auch eine Variable enthalten kann. Für eine Relation können mehrere Modi definiert sein. Bindungsmuster kontrollieren den Datenfluss bei der Auswertung der Randbedingungen. Dabei werden die Aufrufe einzelner Informationsquellen so geordnet, dass alle Bindungsmuster erfüllt werden. Daher stellt die Angabe von Bindungsmustern eine Möglichkeit dar, auf deklarative Weise auf die Reihenfolge der Auswertung und damit auf prozedurale Aspekte der Inferenzprozedur Einfluss zu nehmen.

Beispiel 2.11 *Eine Klausel, die spezifiziert welche Aktion ein Roboter ausführen soll, kann folgende Form haben:*

$$
\begin{aligned}
\mathit{cool\_spray}(\mathit{Object}, X, Y) : [1, T] \quad \leftarrow \quad & \mathit{SP::in\_range}((X_1, Y_1), (X, Y), 2), \\
& \mathit{SP::at}(\mathit{Object}, X_1, Y_1), \\
& \mathit{SN::temp}(\mathit{Object}, \mathit{Temp}), \\
& \mathit{NUM::ge}(\mathit{Temp}, 100) \;||\; \\
& \mathit{robot\_at}(X, Y) : [0.5, T] \\
\mathit{robot\_at}(X, Y) : [V, T] \quad \leftarrow \quad & \mathit{TR::at}(X, Y, V)
\end{aligned}
$$

Im Beispiel wird ein Annotationsverband verwendet, der die Evidenz einer Aussage zu einem gewissen Zeitpunkt repräsentiert. Die erste Pro-

grammklausel hat die Aussage: „Wenn sich der Roboter an Position (X,Y) befindet und die Sicherheit, dass diese Angabe korrekt ist wenigstens 50% beträgt und es existiert ein Objekt O im Abstand von 2 Einheiten, das eine Temperatur von über 100 Grad aufweist, dann besprühe Objekt O mit Kühlmittel.“ Obige Klausel enthält im Randbedingungsteil Relationen aus den Informationsquellen SP, SN, NUM und TR, die respektive eine geographische Datenbank, Temperatursensoren, eine numerische Domäne und ein Verfolgungssystem darstellen.

Die Bindungsmuster der im obigen Beispiel verwendeten Relationen sind SP::in_range$(+,+,+)$, SP::at$(+,?,?)$, SN::temp$(+,?)$, NUM::ge$(+,+)$ und TR::at$(+,?,?)$

2.3 SLANG-Resolution

Die SLANG-Resolution[20] ist die Ergänzung der annotierten SLG-Resolution mit der Verarbeitung von allgemeinen Randbedingungen zur Anbindung externer Informationen. Die SLANG-Resolution wird ausführlicher in [Sch95] und [KLSS] diskutiert, wo auch Vollständigkeits- und Korrektheitsuntersuchungen zu finden sind. Hier wollen wir zusammenfassend noch einmal charakterisieren, für welche Klasse von Programmen die Vollständigkeit und Terminierung der SLANG-Resolution gegeben ist. Die Einschränkungen, die zu machen sind, folgen einerseits aus den Anforderungen der annotierten Resolution, andererseits hängen sie mit der Verwendung von Negation und Funktionen in der logischen Sprache zusammen. Als dritte Komponente müssen die Besonderheiten von Randbedingungen beachtet werden.

- GAPs sind auf der Basis normaler logischer Programme definiert. Die Regelschreibweise unterstreicht den definierenden Charakter solcher normaler Programmklauseln. Solche logische Programme liegen formal gesehen in Klausel-Normalform, d.h. als Konjunktion von Klauseln, wobei jede Klausel wenigstens ein positives Literal enthält. Genau dieses Literal ist der Klauselkopf (s. Abschnitt 2.1).

- Wie in Abschnitt 2.1.1 ausgeführt wurde, muss die Art der Funktionen, die in Annotationen verwendet werden, eingeschränkt werden. Uneingeschränkt erlaubt sind Funktionen mit endlichem Definitions- oder Bildbereich. Andere zugelassene Funktionen sind monotone Funktionen mit diskretem Definitionsbereich. Andere Funktionen können im Einzelfall verwendet werden, wenn durch sie keine unendlichen Mengen von Annotationen erzeugt werden.

- Die SLG-Resolution für annotierte logische Programme bedingt, dass Annotationsterme nur in der Annotation des Klauselkopfes vorkom-

[20] SLG resolution for annotated logic programming with negation

men. Außerdem darf jede Annotationsvariable nur einmal im Klauselrumpf auftreten (s. Abschnitt 2.2.2).

- Aus Berechenbarkeits- und Effizienzgründen beschränkt sich die SLANG-Resolution auf die Verwendung der *Negation as Failure*-Regel zur Behandlung negierter Literale. Dies setzt sichere Programme voraus (s. Abschnitt 2.1.2).

- Wenn man zulässt, dass Randbedingungen, die zur Einbindung externer Informationen formuliert werden, Variablenbindungen erzeugen können, muss man die zugrunde liegenden Funktionen und Relationen dahin gehend einschränken, dass die Anzahl der erzeugten Variablenbindungen in jedem Fall endlich bleibt. Im anderen Fall würden die Terminierungseigenschaften der SLG-Resolution untergraben.

- Im Zusammenhang mit Randbedingungen muss außerdem sicher gestellt sein, dass diese auch ausgewertet werden können. Insbesondere müssen die Einschränkungen in Form von Bindungsmustern erfüllt werden können. Diese Eigenschaft nennt man *Modussicherheit*.

- Für Funktionen, die in Argumentstellen von Literalen vorkommen, müssen ebenfalls Einschränkungen gemacht werden. Für funktionsfreie Programme hat die SLG-Resolution polynomiale Zeit- und Speicherkomplexität. Bei Programmen mit Funktionen, die die *bounded-term-size*-Eigenschaft erfüllen, kann immerhin die Terminierung garantiert werden. Die *bounded-term-size*-Eigenschaft verlangt, dass in einer Deduktion lediglich Terme begrenzter Größe entstehen können. Eine formale Definition kann man in [CW93] nachlesen.

Aus der Sicht der Resolutionsprozedur sind Vollständigkeit und Terminierung für die genannten Einschränkungen gegeben, allerdings sind sie aus dem Blickwinkel der Integration externer Informationen differenzierter zu betrachten. In einer typischen Mediatorumgebung werden Informationsquellen als *autonom* betrachtet. Das hat die Konsequenz, dass die Korrektheit und Vollständigkeit eines integrierten Systems von den entsprechenden Eigenschaften aller Informationsquellen abhängen. Liefert eine der Informationsquellen inkorrekte oder nicht alle möglichen Antworten, kann das auch Auswirkungen auf das Gesamtergebnis haben. Diese Problematik greift an jeder Stelle der Verbindung zwischen Mediatorsystem und Informationsquelle, so kann genauso eine fehlerhafte Kommunikationsverbindung zu diesbezüglichen Problemen führen. Auf jeden Fall hat man in einer Mediatorumgebung mit Einschränkungen der Korrektheit und Vollständigkeit in diesem Sinne zu rechnen.

2.4 KOMET

Das Mediatorsystem KOMET wurde am Institut für Algorithmen und Kognitive Systeme zunächst im Rahmen einer Diplomarbeit entworfen und entwickelt [CJKS97]. Es verwirklicht das erläuterte Konzept einer Inferenzmaschine, die auf der Grundlage der SLANG-Resolution Programme der mehrsortigen verallgemeinerten annotierten Logik mit Randbedingungen verarbeitet. Dabei beschränkt sich die Implementierung auf die Klasse derjeniger Programme, die den im vorigen Abschnitt beschrieben Einschränkungen genügen. Zusätzlich sind keine freien Funktionssymbole erlaubt. Stattdessen können *auswertbare* Funktionen in Form von Randbedingungen in das System eingebunden werden. Als Randbedingungsformeln sind nur Konjunktionen von Termen zugelassen und es wird gefordert, dass Randbedingungen von Einheitsklauseln vollständig lösbar sind.

Beispiel 2.12 *KOMET-Programm*

```
#domains
DB1 = ODBC(MyDb.CFG)
DB2 = ODBC(HisDb.CFG)

#predicates
Address1(STRING,STRING,STRING):[REAL01]
Address2(STRING,STRING,STRING):[FOUR]
Names_DB1(STRING,STRING):[FOUR]

#clauses
Address1(X,Y,Z):[1.0] <- DB1::ADDRESS(X,Y,Z)
Address1(X,Y,Z):[0.5] <- DB2::ADDRESS(X,Y,Z)

Names_DB1(X,Y):[t] <- DB1::ADDRESS(X,Y,Z)
Address2(X,Y,Z):[t] <- DB1::ADDRESS(X,Y,Z)
Address2(X,Y,Z):[t] <- DB2::ADDRESS(X,Y,Z) &
                       not Names_DB1(X,Y):[t]
```

Die KOMET-Sprache ist mehrsortig und modular, d.h. für jedes Prädikat ist die freie Wahl der Argumenttypen sowie eines Annotationstyps aus einer Bibliothek von Basistypen erlaubt. Weitere Argument- und Annotationstypen können mit Hilfe von speziellen Sprachkonstrukten innerhalb eines Programms erzeugt werden. Eine definierte Programmierschnittstelle erlaubt auch die Implementierung und Einbindung neuer Typen mit Hilfe der Programmiersprache C++, in der auch das KOMET-System selbst implementiert wurde.

Das Beispiel 2.12 zeigt ein einfaches KOMET-Programm, mit dem Adressdaten aus zwei Datenbanken verarbeitet werden. Zu Beginn des Programms werden zwei Informationsquellen definiert, die mittels der parametrisierbaren ODBC-Domäne und Konfigurationsdateien erzeugt werden. Die Konfigurationsdateien enthalten die technischen Informationen für den Zugriff

auf die unterliegenden Datenbanken. Die Regeln zeigen zwei Möglichkeiten der Integration. Das Mediatorprädikat *Address1* versieht jeden Eintrag mit einem *Konfidenzwert*. Dabei wird durch die Zuweisung niedriger Werte an Einträge aus `DB2` ausgedrückt, dass diese Datenquelle weniger zuverlässig ist. Für das Prädikat *Address2* kommt eine Präferenzstrategie zur Anwendung. Einträge aus `DB2` sollen nur dann als Antwort für `Address2` übernommen werden, wenn in der Tabelle `DB1` kein entsprechender Eintrag vorhanden ist.

Unterschiedliche Informationsquellen haben unterschiedliche spezialisierte Fähigkeiten zur Auswertung komplexer Anfragen. Für eine Anfrage, die im Mediator zunächst als relationaler Ausdruck vorliegt[21], gibt es prinzipiell immer die Möglichkeit der Auswertung im Mediator. Dazu wird im Allgemeinen die Bildung von Kreuzprodukten und Selektionen benötigt. Die Ausführung dieser Operationen im Mediator bedingt, dass alle Daten der Basisrelationen komplett zum Mediator übertragen werden. Aus Effizienzgründen wird daher in einer Mediatorumgebung angestrebt, die individuellen Fähigkeiten der Informationsquellen so weit wie möglich auszunutzen. Dabei sollte dies allerdings ohne spezielle Spracherweiterungen transparent ermöglicht werden. Teilaufgaben, die in der Mediatorsprache formuliert sind, und von der Informationsquelle als Ganzes verarbeitet werden können, sollten als solche an die Informationsquelle übergeben werden. Als typisches Beispiel sind Datenbanksysteme zu nennen, die selbst über eine komplexe relationale Anfragesprache verfügen und in der Lage sind, komplexe relationale Anfragen über Tabellen auszuwerten.

Zur Anbindung externer Informationsquellen wurde im Rahmen einer Diplomarbeit [Jek98] eine Umgebung zur Entwicklung von *Wrapper*-Komponenten entwickelt, die auf der Grundlage von Techniken aus dem Compilerbau die Übersetzung und Ausführung von konjunktiven Anfragen der Mediatorsprache unterstützt. Dabei wird ein System von Randbedingungen mit Aufrufen externer Quellen, das vom Mediator an die Übersetzerkomponente gereicht wird, einer Reihe von Transformationen unterzogen, die mit semantischen Aktionen verknüpft sind und so zum Beispiel einen Ausdruck der Zielsprache erzeugen oder direkt eine Reihe von Operationen ausführen. Das Ergebnis der Ausführung wird an den Mediator zurückgegeben. Dabei fällt dem Übersetzer auch die Aufgabe zu, die Ergebnisse in das Datenmodell des Mediators zu konvertieren.

In [Sch95] wird der Zusammenhang zwischen der annotierten Logik und der Verwaltung von Sichten[22] dargestellt. Mediatorprädikate, die durch Mediatorklauseln definiert werden, können als Sichten auf die Kombination der unterliegenden Informationen betrachtet werden. In einer dynamischen Umgebung stellt sich dabei das Problem, wie sich Änderungen innerhalb

[21] Zur Erinnerung: In KOMET werden Informationsquellen als Menge von Relationen mit Moduseinschränkungen aufgefasst

[22] engl.: *view maintenance*; Eine *Sicht* ist eine Relation, die auf der Basis anderer Relationen definiert ist.

der Informationsquellen auf das Mediatorsystem auswirken. Eine effiziente Verarbeitung von Änderungen erfordert die Propagierung von geänderten Daten, die einerseits *vollständig* ist, d.h. alle Auswirkungen berücksichtigt, andererseits *minimal* ist. Entsprechende Algorithmen werden in [Sch95] für annotierte logische Programme unter der *well-founded*-Semantik dargestellt.

Die Verarbeitung komplexer Annotationen wurde in einer Diplomarbeit untersucht [Tan99, CKT00]. Durch die Kombination verbandsgeordneter Mengen können neue Verbände konstruiert werden, die als Annotationen einsetzbar sind. Die Konstruktion kombinierter Annotationsverbände dient zur Repräsentation von Informationen, die eine gleichzeitige Abhängigkeit von mehreren Zustandsvariablen zeigen. Solche Zustände sind die Bedingungen unter denen ein Atom gültig ist, und können beispielsweise Zeit- oder Ortsangaben sein. Zusätzlich können diese Umstände mit Bewertungen kombiniert sein. Aus Umstandsmengen können leicht Annotationsverbände als Teilmengen der dazugehörigen Potenzmenge zusammen mit der Teilmengenordnung gebildet werden. Für zahlreiche Anwendungen interessant sind Annotationen, die aus Ungleichungen über mehrere Verbände konstruiert werden und die Form $\bigvee_j \bigwedge_k (X_{l_{jk}}) \leq a_{jk}$ haben, wenn a_{ij} Annotationen aus der Vereinigung der einzelner Verbände $\bigcup_i L_i$ sind und $l_{jk} = i$ gdw. $a_{jk} \in L_i$. Eine Struktur, die dieser Formelmenge entspricht ist das freie distributive Produkt von distributiven Verbänden. Zur Konstruktion von solchen Annotationen in KOMET wurde ein entsprechender parametrisierbarer Verbandstyp implementiert.

Beispiel 2.13 *Komplexe Annotationstypen*

```
#lattices
DOW = POWERSET(Mo,Di,Mi,Do,Fr,Sa,So)
STAFF = POWERSET(Mueller,Maier,Schneider,Schmidt)
SDOWC = FREEDPR(STAFF,DOW,REAL01)
LDOWC = LEXPR(REAL01,DOW)

#predicates
Team(STRING):[STAFF]
Schedule(ULONG):[SDOWC]
BestMeeting(STRING,ULONG):[LDOWC]

#clauses
Team('T1'):[Mueller,Maier]
Schedule(23):[(Maier*Mo,Sa*0.8 + Maier*Mi,Fr*0.6)]
Schedule(23):[(Mueller*Mo*0.2 + Mueller*Fr*0.8)]
BestMeeting(X,Y):[LDOWC::LDOWC(C,D)] <-
  Team(X):[Z] & Schedule(Y):[SDOWC::CTERM(Z,D,C)]
```

Das Beispiel 2.13 illustriert, wie zusammengesetzte Annotationstypen in einer Anwendung zur Terminplanung eingesetzt werden könnten. `FREEDPR` und `LEXPR` sind parametrisierbare Verbandstypen, mit denen Produkte anderer Verbände gebildet werden. Die durch solche Konstruktionen erreich-

bare Flexibilität erlaubt die differenzierte, auf die Anwendungsbedürfnisse zugeschnittene Modellierung des Domänenwissens.

In [CK99b] wird der Einsatz von KOMET als Metasuchmaschine im Internet diskutiert und dabei einige Möglichkeiten des KOMET-Systems demonstriert. Weitere Anwendungsmöglichkeiten von Mediatoren im Zusammenhang mit wissenschaftlichen Datenbanken und im Zusammenhang mit Computeralgebra werden in in [KK] bzw. in [CBK00] beschrieben.

Die vorliegenden Arbeiten zeigen, dass der gewählte Ansatz durch die Verwendung der annotierten Logik Möglichkeiten der Repräsentation von Integrationswissen bietet, die ihn von anderen Arbeiten im Bereich der Wissensintegration abhebt.

In diesem Kapitel wurde die annotierte Logik vorgestellt und Erweiterungen im Hinblick auf die Verwendung zur Informationsintegration beschrieben. Es wurden Resolutionsverfahren für annotierte logische Programme (GAPs) diskutiert und gezeigt, wie die SLG-Resolution, ein effizientes Verfahren zur Berechnung der *well-founded*-Semantik, für GAPs angepasst werden kann. Abschließend wurde die Implementierung des beschriebenen Resolutionsverfahrens im Mediatorsystem KOMET vorgestellt.

Überall geht ein frühes Ahnen dem späteren Wissen voraus.
Alexander von Humboldt

Kapitel 3

Repräsentation unsicheren Wissens

In diesem Kapitel wird die Behandlung von unsicherem Wissen mit der annotierten Logik untersucht. Im ersten Abschnitt werden zunächst Quellen unsicheren Wissens in einer Mediatorumgebung diskutiert. Anschließend wird die possibilistische Logik [DLP94] eingeführt und gezeigt, wie sie in die annotierte Logik eingebettet werden kann. Im dritten Abschnitt wird eine Erweiterung der possibilistischen Logik um Elemente der unscharfen Logik besprochen. Im letzten Abschnitt wird die Umsetzung dieser Erweiterungen im Mediatorsystem KOMET vorgestellt.

3.1 Informationsintegration und Unsicherheit

Ein Merkmal der spezifischen Anwendungsgebiete von KOMET ist die Unsicherheit von Wissen, die adäquat repräsentiert werden muss, um gegebenes Wissen möglichst genau zu erhalten. Insbesondere im Bereich der Integration heterogener Informationen können Unsicherheiten das Ergebnis des Integrationsprozesses beeinflussen. Solche Unsicherheiten können an verschiedenen Stellen des Prozesses einfließen:

- Eine Information ist per se unsicher.
- Eine Informationsquelle insgesamt wird als unsicher eingestuft.
- Durch die Abbildung auf das gemeinsame Datenmodell im Wrapper werden Unsicherheiten eingeführt. (z.B. Präzisionsverlust durch Datenkonvertierung)
- Das Wissen über Abbildung des Quellenschemas auf das Mediatorschema ist unsicher.

- Das Wissen zur Verarbeitung der integrierten Informationen ist unsicher.

Der allgemeine Ansatz der annotierten Logik bietet einen Rahmen, um unsicheres Wissen in einer mehrwertigen Logik darzustellen, sofern sich deren Trägermenge in Form eines Verbandes beschreiben lässt. Insbesondere bietet sie die Möglichkeit, bekannte Logiken zu repräsentieren und zu verarbeiten.

Aus der Literatur sind verschiedene Logiken zur Darstellung von unsicherem Wissen bekannt, die eingehend untersucht wurden [GHR94]. In [KS92] werden verschiedene Ansätze zur Repräsentation mehrwertiger Logiken in Bezug auf die Darstellung und Verarbeitung mittels GAPs untersucht. Dabei handelt es sich um van Emdens quantitative Deduktion, biverband-wertige Logiken (engl.: *bilattice-valued logics*), sowie lineare und intervallbasierte Temporallogiken. Für diese Logiken wird gezeigt, inwiefern sie als GAP ausgedrückt und mittels einer entsprechender Auswertungsprozedur behandelt werden können.

Eine Logik zur Repräsentation von unsicherem Wissen ist die *possibilistische Logik* (PL). Diese dazugehörige Theorie ist im Detail in [DLP94] beschrieben. In den folgenden Abschnitten wird zunächst eine Einführung in die possibilistische Logik gegeben, sowie eine Erweiterung um Elemente der *unscharfen Logik* vorgestellt. Danach wird die Darstellung possibilistischer Formeln in Klauselform und deren Verarbeitung mittels geeigneter Inferenzregeln beschrieben. Im Kern dieses Kapitels wird gezeigt, wie possibilistische Klauseln in Programmen der annotierten Logik dargestellt und mit entsprechenden Resolutionsverfahren verarbeitet werden können.

3.2 Possibilistische Logik

Die *possibilistische Logik* ist eine Logik der Unsicherheit, zugeschnitten auf das Schließen bei unvollständiger Evidenz und teilweise inkonsistentem Wissen. Sie ist entstanden aus der Möglichkeitstheorie (engl.: *possibility theory*, [DLP94]), in der Wissen durch *Möglichkeitsverteilungen* beschrieben wird. Beispielsweise wird der Wert einer Variablen x über einer Grundmenge U als eine Abbildung π_x von U nach $[0, 1]$ angegeben. Folgende Konventionen werden der Interpretation von π_x zugrunde gelegt:

- $\pi_x(u) = 0$ bedeutet, dass $x = u$ unmöglich ist;
- $\pi_x(u) = 1$ bedeutet, dass $x = u$ vollständig möglich ist;
- $\pi_x(u) \geq \pi_x(u')$ bedeutet, dass $x = u$ gegenüber $x = u'$ bevorzugt wird.

Eine Möglichkeitsverteilung stellt eine 'elastische Einschränkung' dar für die Werte, die eine Variable x annehmen kann. Somit kann eine Aussage wie

„John ist groß“ als eine Element unvollständiger Evidenz betrachtet werden. Diese Sichtweise unterscheidet sich grundsätzlich von der mehrwertigen Interpretation der unscharfen Logik, wo einer Aussage wie „John ist groß“ ein Wahrheitswert zugeordnet wird, die sich aus der charakteristischen Funktion der durch die linguistische Variable „groß“ bezeichnete unscharfe Menge berechnet.

Ein Möglichkeitsmaß ist eine Mengenfunktion Π die jeder Untermenge $A \subseteq U$ einen Wert $\Pi(A) \in [0, 1]$ zuweist. Ein Möglichkeitsmaß kann aus einer Möglichkeitsverteilung π erzeugt werden, wenn gilt $\forall u \in U, \pi_x(u) = \Pi(\{u\})$.

Zu einer Möglichkeitsverteilung dual verhält sich die *Notwendigkeitsverteilung*. Im Gegensatz zur Möglichkeitsverteilung beschreibt sie die Notwendigkeit oder Evidenz, die für eine Aussage besteht. Genau wie bei Möglichkeitsverteilungen lässt sich aus einer Notwendigkeitsverteilung das zugehörige Maß N erzeugen. Der Zusammenhang zwischen Π und N ist wie folgt gegeben:

$$N(A) = 1 - \Pi(\overline{A}) = \inf_{u \notin A} 1 - \pi_x(u)$$

Hierbei bezeichne $\overline{A}$ das Komplement der Menge A bezüglich der Grundmenge U. Wie wir gesehen haben, lässt sich unsicheres Wissen in Form von möglichkeits- und notwendigkeitsqualifizierten Aussagen ausdrücken. Die possibilistische Logik behandelt syntaktische Objekte, die aus Aussagen resultierende Ungleichungen ausdrücken. Sie haben die Form $N(\phi) \geq \alpha$ für notwendigkeitsqualifizierte Aussagen und $\Pi(\phi) \geq \alpha$ für möglichkeitsqualifizierte Aussagen, wenn ϕ eine geschlossene logische Formel erster Stufe ist. Diese Objekte, die *possibilistische Formeln* genannt werden, sind die Basiselemente der possibilistischen Logik. Eine *unsichere Wissensbasis* ist dann eine Menge von notwendigkeits- und möglichkeitsqualifizierten Aussagen, die logisch als Konjunktion possibilistischer Formeln repräsentiert wird.

Auf der syntaktischen Ebene behandelt sie Formeln der Aussagenlogik oder der Prädikatenlogik erster Stufe, denen Zahlen zwischen 0 und 1, oder allgemeiner, Elemente einer totalgeordneten Menge angehängt werden. Diese Gewichte sind untere Schranken des Grades der Notwendigkeit oder des Grades der Möglichkeit der jeweiligen Formel. Der Grad der Notwendigkeit (oder Sicherheit) einer Formel drückt aus, in welchem Maße die verfügbare Evidenz die Wahrheit der Formel unterstützt. Der Grad der Möglichkeit beschreibt, in welchem Maße die Wahrheit der Formel der verfügbaren Evidenz nicht widerspricht.

Obwohl die allgemeine possibilistische Logik die Modellierung von Aussagen mit Hilfe von Notwendigkeiten als auch mit Hilfe von Möglichkeiten zulässt, ist die Einschränkung auf Notwendigkeitswerte ausreichend um eine Präferenzordnung über Formeln zu modellieren. Diese Einschränkung sorgt zudem für eine Vereinfachung formaler und algorithmischer Aspek-

te. Im Folgenden werden wir uns auf die Verwendung von Notwendigkeiten beschränken.

Zunächst wollen wir mit einigen grundlegenden Definition die Syntax und Semantik der possibilistischen Logik festlegen, wobei diese im Wesentlichen hier einer klassischen, aber mehrsortigen Logik entspricht.

Definition 3.1 (Mehrsortige Signatur)
Eine possibilistische Signatur Σ *ist ein Tupel* (S, X, C, P)*, für das gilt:*

- S *ist eine endliche, nichtleere Menge von* Sortensymbolen *oder kurz* Sorten*. Ein* Typ *ist ein Tupel von Sorten.*
- X *ist eine endliche Menge von* Variablensymbolen *oder kurz* Variablen*.*
- C *ist eine endliche Menge von* Konstantensymbolen *oder kurz* Konstanten*.*
- P *ist eine endliche, nichtleere Menge von* Prädikatensymbolen *oder kurz* Prädikaten*. Jedes Prädikat besitzt einen Typ aus* 2^S*.*

Definition 3.2 (Mehrsortige Sprache)
Eine mehrsortige Sprache $\mathcal{L}$ *besteht aus einer Signatur* Σ *und den logischen Konnektiven* $\vee$ *und* $\neg$*. Atome der Sprache* $\mathcal{L}_{PL}$ *werden aus Elementen aus P,X und C gebildet. Die Terme, die aus diesen Atomen und den logischen Konnektiven gebildet werden können, sind* mehrsortige Formeln*.*

Jede Formel der (notwendigkeitswertigen) possibilistischen Logik wird durch ein Paar (p, α) repräsentiert, wobei p eine mehrsortige Formel ist und $\alpha \in (0, 1]$ eine untere Schranke der Sicherheit von p in Form eines Notwendigkeitsmaßes (*necessity measure*). Eine Formel (p, α) wird also interpretiert als $N(p) \geq \alpha$, wobei N ein Notwendigkeitsmaß ist, das die Menge der logischen Formeln auf ein totalgeordnetes beschränktes Intervall abbildet. Ein Notwendigkeitsmaß ist charakterisiert durch folgende Axiome:

1. $N(\top) = 1$,
2. $N(\bot) = 0$,
3. $N(p \wedge q) = \min(N(p), N(q))$,
4. $N(p) = N(q)$, wenn p und q klassisch äquivalent sind,

wobei $\top$ und $\bot$ Tautologie bzw. Widerspruch bezeichnen.

Definition 3.3 (Interpretation)
Eine Interpretation $w = (U, i, m)$ *bildet*

1. *jede Sorte* σ *auf eine nichtleere Domäne* U_σ *ab,*

2. *ein Prädikat p des Typs $(\sigma_1, \ldots, \sigma_n)$ auf eine Relation $i(p) \subseteq U_{\sigma_1} \times \cdots \times U_{\sigma_n}$,*

3. *eine Konstante c der Sorte σ auf einen Wert $m(c) \in U_\sigma$.*

Um ein Resolutionverfahren für die possibilistische Logik angeben zu können, wird zunächst die possibilistische Klauselform definiert. Darauf aufbauend wird die Resolution für possibilistische Klauseln erweitert.

3.2.1 Klauseldarstellung und Inferenz

Die Klauselform mehrsortigen Logik ist analog zur Klauselform der klassischen Prädikatenlogik definiert.

Definition 3.4 (Possibilistische Klausel)
Sei (p, α) eine eine Formel der possibilistischen mehrsortigen Logik, d.h p ist eine mehrsortige Formel und α ist ein Wert aus (0..1]*, dann ist (c, α) eine* possibilistische Klausel*, wenn c die aus p transformierte Klausel ist.*

Definition 3.5 (Possibilistische Klauselform)
Eine possibilistische Klauselform *ist eine universell quantifizierte Konjunktion von possibilistischen Klauseln.*

Definition 3.6 (Possibilistische Hornklausel)
Sei p eine possibilistische Klausel der Form $p = (p_0 \vee \neg p_1 \vee \cdots \vee \neg p_n, v)$, in deren logischen Teil höchstens ein positives Literal existiert. Jedes p_i sei ein positives Literal mit einer beliebigen Anzahl von possibilistischen logischen Termen als Argumenten. Dann ist p eine possibilistische Hornklausel.

In Regelschreibweise wird eine possibilistische Klausel folgendermaßen dargestellt:

$$(p_0 \leftarrow p_1 \wedge \cdots \wedge p_n, v)$$

Typische Beispiele für die possibilistische Logik kommen aus dem Bereich der Diagnose. Das folgende Programm berechnet in welchem Maße Anhaltspunkte für die Annahme existieren, dass eine bestimmte Komponente eines PKWs defekt ist bei gegebenen Beobachtungen. Prädikatnamen und Konstanten haben folgende Bedeutungen:

Ff(c)	Fehlfunktion der Komponente c
Def(c)	Defekt der Komponente c
Ger(s)	Geräusch der Art s
Anz(s)	Anzeige des Typs s
Lichtm.	Lichtmaschine
Keilr.	Keilriemen
Batt.	Batteriewarnleuchte

Beispiel 3.1 (Possibilistisches Programm) *Ein possibilistisches Programm zur Diagnose von schadhaften PKW-Komponenten sei wie folgt definiert.*

$$\begin{aligned} (Ff(Lichtm.) &\leftarrow Anz(Batt.), 0.7) \\ (Def(Keilr.) &\leftarrow Ff(Lichtm.) \wedge Ger(Quietschen), 0.8) \\ (Ger(Quietschen), 1) & \\ (Anz(Batt.), 1) & \end{aligned}$$

Die beiden Einheitsklauseln in Beispiel 3.1 sind die Beobachtungen, die in einem exemplarischen Fall gemacht wurden. Über die beiden Regeln kann eine Diagnose mittels geeigneter Inferenzprozeduren abgeleitet werden. So kann das Resolutionsprinzip auf die possibilistische Logik mit Notwendigkeitswerten übertragen werden, um den Grad der Inkonsistenz solcher Programme zu berechnen. Die folgende Resolutionsregel entspricht der Definition in [DLP94]

$$R^{\pi} : \frac{(\neg p \vee l_{p_1} \vee \cdots \vee l_{p_n}, v_p); (q_0 \vee l_{q_1} \vee \cdots \vee l_{q_m}, v_q)}{[(l_{p_1} \vee \cdots \vee l_{p_n} \vee l_{q_1} \vee \cdots \vee l_{q_m}, \min(v_q, v_p))]\theta}$$

wobei θ der *allgemeinste Unifikator* ist, der die beiden Atome p_0 und q_0 unifiziert und $l_{p_i}, 0 \leq i \leq n$ und $l_{q_j}, 0 \leq j \leq m$ negative oder positive Literale sind.

Die Korrektheit und Vollständigkeit der possibilistischen Resolution wurde in [DP89] gezeigt.

Wir verwenden für possibilistische Klauseln ein Version der Resolutionsregel, die für die lineare Resolution definiter Programme eingeschränkt ist. Dies ist ausreichend, wenn wir uns auf der Seite der annotierten Logik ebenfalls auf eine lineare Resolution beschränken.

Definition 3.7 (Possibilistische Resolutionsregel)
Sei q eine Anfrage, und p eine possibilistische Klausel. Sei θ der allgemeinste Unifikator, der die Kopfliterale p_0 der Klausel p und das Rumpfliteral q_i der Klausel q unifiziert. Dann ist

$$R^{\pi}_{*} : \frac{(\leftarrow q_1 \wedge \cdots \wedge q_m, v_q); (p_0 \leftarrow p_1 \wedge \cdots \wedge p_n, v_p)}{[(\leftarrow q_1 \wedge q_{i-1} \wedge p_1 \wedge \cdots \wedge p_n \wedge q_{i+1} \wedge \cdots \wedge q_m, \min(v_q, v_p))]\theta}$$

die possibilistische Resolutionsregel.

Die Fusionsregel für PL lässt sich leicht aus der Interpretation possibilistischer Formeln herleiten.

Seien p eine logische Formel und α und β Werte eines Notwendigkeitsmaßes. Dann ist die *Fusionsregel* der PL folgendermaßen definiert:

$$F^{\pi} : \frac{(p, \alpha), (p, \beta)}{(p, \max(\alpha, \beta))}$$

Daraus lässt sich für unsere Zwecke eine für Einheitsklauseln ausgelegte Form ableiten:

Definition 3.8 (Possibilistische Fusionsregel)
Seien p und q Einheitsklauseln und θ der allgemeinste Unifikator der p und q unifiziert. Seien α und β Werte eines Notwendigkeitsmaßes. Dann ist die possibilistische Fusionsregel *gegeben durch:*

$$F_*^\pi : \frac{(p, \alpha), (q, \beta)}{[(p, \max(\alpha, \beta))]\theta}$$

3.2.2 Transformation

In diesem Abschnitt werden wir untersuchen, wie sich possibilistische Klauseln im Kontext der annotierten Logik darstellen lassen. Dazu beschreiben wir wie eine Klausel s der possibilistischen Logik in eine Klausel der annotierten Logik s^a transformiert werden kann.

Definition 3.9 (Transformierte possibilistische Klausel)
Sei p eine possibilistische Klausel der Form $(p_0 \leftarrow p_1 \wedge \cdots \wedge p_n, v_p)$ und $x_i, \ldots, x_n$ eine Menge paarweiser disjunkter Annotationsvariablen, dann ist die annotierte Klausel

$$p_0 : \min(v_p, x_1, \ldots, x_n) \leftarrow p_1 : x_1 \wedge \cdots \wedge p_n : x_n$$

die transformierte possibilistische Klausel *von p. Sie wird durch p^a bezeichnet.*

Die possibilistische Wertung geht also in die Annotation des Klauselkopfs ein. Insbesondere erzeugt die Transformation einer positiven Einheitsklausel (p_f, v_f) das annotierte Faktum $p_f : v_f$.

Beispiel 3.2 (Transformiertes possibilistisches Programm)
Das possibilistische Programm aus Beispiel 3.1 hat folgende transformierte Form.

$$\begin{array}{rcll}
\mathit{Ff}(\mathit{Lichtm.}) : \min(0.7, V) & \leftarrow & \mathit{Anz}(\mathit{Batt.}) : V & \langle R_1 \rangle \\
\mathit{Def}(\mathit{Keilr.}) : \min(0.8, V_1, V_2) & \leftarrow & \mathit{Ff}(\mathit{Lichtm.}) : V_1 \wedge & \\
 & & \mathit{Ger}(\mathit{Quietschen}) : V_2 & \langle R_2 \rangle \\
\mathit{Ger}(\mathit{Quietschen}) : 1 & & & \langle F_1 \rangle \\
\mathit{Anz}(\mathit{Batt.}) : 1 & & & \langle F_2 \rangle
\end{array}$$

Eine Anfrage der possibilistischen Logik wird in eine annotierte Anfrage mit Annotationsrandbedingung (siehe Abschnitt 2.8) übersetzt. Die Anfrage

$$q : (\leftarrow q_1 \wedge \cdots \wedge q_n, v_q)$$

wird dabei zu

$$q^a :\leftarrow y_0 \preceq \min(v_q, y_1, \ldots, y_n) \parallel q_1 : y_1 \wedge \cdots \wedge q_n : y_n$$

wobei $\alpha_i, 0 \leq i \leq n$, Annotationsvariablen sind. Wenn die anfängliche Anfrage eine Einheitsklausel der Form $(\leftarrow q_1, 1)$ ist, dann erzeugt die Transformation die annotierte Anfrage $\leftarrow y_0 \preceq y_1 \parallel q_1 : y_1$.

Wir definieren die Verbandsstruktur $\mathcal{T}_{\mathcal{PL}}$ als Verband der reellen Zahlen aus dem Intervall $[0, 1]$ mit der üblichen Totalordnung. Die kleinste obere Schranke (engl.: *least upper bound, lub*, $\sqcup$) zweier Elemente des Verbands ist gegeben durch die Funktion max, die das größere der beiden Elemente zurück gibt. Zu bemerken ist, dass für Verbände mit Totalordnung und bei Verwendung der Funktion max zur Berechnung der kleinsten oberen Schranke, die Reduktionsregel der annotierten Logik keine Rolle spielt. Ihre Anwendung erzeugt keine höheren Annotationen im Sinne der zum Verband gehörigen Ordnung und fällt mit der Fusionsregel der PL zusammen.

Sei K eine Menge possibilistischer Klauseln und K^a die Menge der transformierten Klauseln. Sei q eine possibilistische Anfrage und q^a die transformierte Anfrage. Sei $(q^a)^i$ die i-te Anfrage die im i-ten Resolutionsschritt erzeugt wurde und $(q^a)^0 = q^a$. Sei p^a eine Klausel aus K^a deren Kopf mit einem Literal aus q_i^a unifizierbar ist. Dann erzeugt eine Anwendung der annotierten Resolutionsregel auf $(q^a)^i$ die annotierte Anfrage

$$(q^a)^{i+1} = R^a((q^a)^i, p^a).$$

Das folgendes Theorem konstatiert die Äquivalenz von $(q^a)^{i+1}$ und $(q^{i+1})^a$.

Theorem 3.1

$$\forall i \geq 0, R^a((q^a)^i, p^a) = (R^\pi(q^i, p))^a$$

Beweis 3.1 *Sei* $(q^a)^i :\leftarrow x_0 \preceq \min(v_q, x_1, \ldots, x_n) \parallel q_1 : x_1 \wedge \cdots \wedge q_n : x_n$ *eine annotierte Anfrage, die durch i Anwendungen der annotierten Resolutionsregel entstanden ist. Sei* $p^a : p_0 : \min(v_p, y_1, \ldots, y_m) \leftarrow p_1 : y_1 \wedge \cdots \wedge p_m : y_m$ *eine transformierte possibilistische Programmklausel deren Kopfliteral mit dem Literal* q_j *mittels des allgemeinsten Unifikators* θ *unifizierbar ist. Dann ergibt die Anwendung der annotierten Resolutionsregel die Anfrage*

$$\begin{aligned}&(q^a)^{i+1} : [\leftarrow x_0 \preceq \min(v_q, x_1, \ldots, x_n), x_j \preceq min(v_p, y_1, \ldots, y_m) \parallel \\ &q_1 : x_1 \wedge \cdots \wedge q_{j-1} : x_{j-1} \wedge p_1 : y_1 \wedge \cdots \wedge \\ &p_m : y_m \wedge q_{j+1} : x_{j+1} \wedge \cdots \wedge q_n : x_n]\theta.\end{aligned}$$

Durch Einsetzen ergibt sich

$$\begin{aligned}&(q^a)^{i+1} : [\leftarrow x_0 \preceq \min(v_q, x_1, \ldots, x_{j-1}, v_p, y_1, \ldots, y_m, x_{j+1}, x_n) \parallel \\ &q_1 : x_1 \wedge \cdots \wedge q_{j-1} : x_{j-1} \wedge p_1 : y_1 \wedge \cdots \wedge \\ &p_m : y_m \wedge q_{j+1} : x_{j+1} \wedge \cdots \wedge q_n : x_n]\theta.\end{aligned}$$

Die possibilistische Resolution der Anfrage $q^i :\leftarrow q_1 \wedge \cdots \wedge q_n, v_q$ *und der possibilistischen Programmklausel* p *über das gleiche Literal* p_j *mit Unifikator* θ *ergibt* $q^{i+1} : [\leftarrow q_1 \wedge \cdots \wedge q_{j-1} \wedge p_1 \wedge \cdots \wedge p_m \wedge q_{j+1} \wedge \cdots \wedge q_n, min(v_q, v_p)]\theta$. *Die Transformierte* $(q^{i+1})^a$ *ist dann*

$$[\leftarrow x_0 \preceq \min(\min(v_q, v_p), x_1, \ldots, x_{j-1}, y_1, \ldots, y_m, x_{j+q}, \ldots, x_n) \parallel \\ q_1 : x_1 \wedge \cdots \wedge q_{j-1} : x_{j-1} \wedge p_1 : y_1 \wedge \cdots \wedge \\ p_m : y_m \wedge q_{j+1} : x_{j+1} \wedge \cdots \wedge q_n : x_n]\theta$$

Daraus ergibt sich die Behauptung.

Wir sehen also, dass ein Resolutionsschritt der annotierte Resolution mit entsprechendem Verband und ein Schritt der possibilistischen Resolution analog verlaufen. Eine Widerlegung mit Hilfe der possibilistischen Resolution kann also entsprechend im Rahmen der annotierten Logik durchgeführt werden. Die optimale Bewertung wird bei der possibilistischen Resolution dadurch ermittelt, dass alle möglichen Widerlegungspfade erzeugt und daraus der maximale Inkonsistenzwert ermittelt wird. Resolutionsverfahren für die annotierte Logik gehen im Prinzip ebenso vor. Eine Unterzielauswertung erzeugt alle möglichen Antworten für ein Anfrageliteral, die dann mit der Anfrage resolviert werden. Durch die Subsumptionseigenschaft der Annotationen ist die maximale Annotation für eine Antwort relevant.

Theorem 3.2 *Sei* $\mathcal{F}$ *eine possibilistische Wissensbasis und* φ *eine Anfrage. Dann berechnet eine vollständige Resolutionsprozedur der annotierten Logik über die transformierte Wissensbasis* $\mathcal{F}'^a$ *in Klauselform mit der transformierten Anfrage* φ'^a *den maximalen Grad der Inkonsistenz von* $\mathcal{F} \cup \{(\neg\, \varphi \;\; 1)\}$ *bzw. den maximalen Grad der Notwendigkeit von* φ.

Beweis 3.2 *Nach [DLP94] ist der maximale Grad der Inkonsistenz von* $C' = \mathcal{F} \cup \{(\neg\, \varphi \;\; 1)\}$ *die optimale Widerlegung durch Resolution von* C'. *Aus Theorem 2.1 und Theorem 3.1 folgt, dass die annotierte Resolution die optimale Widerlegung von* $\mathcal{F}' \cup (\neg\, \varphi \;\; 1)$ *berechnet. Dies ist genau der maximale Inkonsistenzgrad* $Val(\varphi, \mathcal{F}) = \sup\{\alpha \in (0, 1], \mathcal{F} \models (\varphi \;\; \alpha)\}$. □

Dabei lassen sich eine Reihe von Heuristiken für eine Einschränkung des Suchraums anwenden. So werden Klauseln, deren Köpfe von bereits berechneten Antworten subsumiert werden, nicht weiter betrachtet. Genauso werden Antworten, die nach einer allgemeineren Antwort gefunden werden, nicht weiter resolviert.

Beispiel 3.3 *Zum Programm aus Beispiel 3.2 sei folgende annotierte Anfrage gegeben*

$$\leftarrow \quad V_0 \preceq V_1 \parallel Def(x) : V_1 \quad \langle Q_0 \rangle$$

Die Anwendung der Resolutionsregel liefert die folgende Widerlegung

$$
\begin{array}{lll}
\langle Q_0, R_2\rangle : & \leftarrow & V_0 \preceq V_1, V_1 \preceq \min(0.8, W_1, W_2) \parallel \\
& & \mathit{Ff}(\mathit{Lichtm.}) : W_1 \wedge \mathit{Ger}(\mathit{Quietschen}) : W_2 \quad \langle Q_1\rangle \\
\langle Q_1, R_1\rangle : & \leftarrow & V_0 \preceq V_1, V_1 \preceq \min(0.8, W_1, W_2), \\
& & W_1 \preceq \min(0.7, U) \parallel \\
& & \mathit{Anz}(\mathit{Batt.}) : U \wedge \mathit{Ger}(\mathit{Quietschen}) : W_2 \quad \langle Q_2\rangle \\
\langle Q_2, F_2\rangle : & \leftarrow & V_0 \preceq V_1, V_1 \preceq \min(0.8, W_1, W_2), \\
& & W_1 \preceq \min(0.7, U), U \preceq 1 \parallel \\
& & \mathit{Ger}(\mathit{Quietschen}) : W_2 \quad \langle Q_3\rangle \\
\langle Q_3, F_1\rangle : & \leftarrow & V_0 \preceq V_1, V_1 \preceq \min(0.8, W_1, W_2), \\
& & W_1 \preceq \min(0.7, U), U \preceq 1, W_2 \preceq 1 \parallel \\
& & \bot \quad \langle Q_4\rangle
\end{array}
$$

Der Semantik der annotierten Logik folgend, suchen wir eine Lösung der Annotationsrandbedingung, so dass V_0 maximal ist. Dies erlaubt uns die Randbedingung entsprechend zu vereinfachen zu

$$V_0 = \min(0.8, 0.7, 1, 1) = 0.7$$

Die Notwendigkeit der Aussage Def(x) bei der Substitution $[x/Keilr.]$ beträgt also .7.

3.3 Possibilistische Logik mit variablen Gewichten und unscharfen Konstanten

In [DPS96, DPS98] wird die Erweiterung der possibilistischen Logik um variable Gewichte und unscharfe Konstanten vorgeschlagen. In [AGS99] wird eine formale Semantik und eine Deduktionsprozedur für diese erweiterte possibilistische Logik angegeben. Sie erlauben unscharfe Prädikate in den Rahmen der possibilistischen Logik einzubetten. Diese erweiterte Logik bezeichnen wir im Folgenden mit PLFC. In [AGS99] wird zwar die Korrektheit der Resolutionsregel gezeigt, allerdings ist die Vollständigkeit der vorgeschlagenen Prozedur nicht gegeben. Dennoch bietet PLFC eine besondere Ausdrucksstärke, so dass die Verwendung für bestimmte Anwendungen sinnvoll sein kann. Bislang wurde kein System von Inferenzregeln angegeben, für das die Vollständigkeit nachgewiesen werden konnte. Insofern überträgt sich die Unvollständigkeit auf die annotierte Resolution.

3.3.1 Variable Gewichte

Variable Gewichte werden benutzt, um Aussagen wie „Je mehr x zu A gehört, um so sicherer ist $p(x)$“ modellieren zu können, wobei A eine unscharfe Menge bezeichnet. Damit lassen sich Aussagen modellieren, die im

normalen Sprachgebrauch oft verwendet werden. Typisch sind derartige Aussagen bei der Formulierung von Daumenregeln, wie z.B. „Je teurer ein Auto ist, um so mehr kann man davon ausgehen, dass es schnell fährt“. Anders formuliert ergibt sich die Form „Je mehr ein Auto X zur Menge der teuren Autos gehört, um so größer ist die Evidenz dafür, dass es auch schnell fahren kann“. Eine solche Aussage wird formalisiert als „für alle x gilt $p(x)$ mit einer Notwendigkeit von wenigstens $\mu_A(x)$“ und durch folgende Formel repräsentiert:

$$(p(x), \mu_A(x))$$

Dabei ist μ_A die charakteristische Funktion der unscharfen Menge A, die jedem Element der Grundmenge über die A definiert ist, einen Wert aus dem Intervall [0,1] zuweist. Formal erweitern wir zunächst die Sprache PL um unscharfe Mengen und unscharfe Konstanten.

Definition 3.10 (α-Schnitt) *Sei A eine unscharfe Menge mit der charakteristischen Funktion μ_A. Der α-Schnitt $[A]_\alpha$ ist wie folgt definiert:*

$$[A]_\alpha = \{a \in A | \mu_A(a) \geq \alpha\}$$

Definition 3.11 (PLFC-Signatur)
Eine PLFC-Signatur *erweitert eine mehrsortige Signatur um eine Menge von* erweiterten Sorten *$f\sigma$ und ein Menge FC von unscharfen Konstanten; desweiteren um eine Menge FC_{cuts} von ungenauen Konstanten die den α-Schnitten der unscharfen Konstanten in FC entsprechen, d. h. wenn $A \in FC$, dann ist $[A]_\alpha \in FC_{cuts}$, für $0 \leq \alpha \leq 1$.*

Definition 3.12 (PLFC-Interpretation)
Neben den üblichen Abbildungen, bildet die PLFC-Interpretation $w = (U, i, m)$

1. *jede erweiterte Sorte $f\sigma$ auf die Menge $F(U_\sigma)$ von unscharfen Mengen U_σ ab. Eine unscharfe Menge $A \in U_\sigma$ ist dabei charakterisiert durch die Elementfunktion $\mu_{m(A)}$,*

2. *eine Objektkonstante c der Sorte σ auf einen Wert $m(c) \in U_\sigma$, eine unscharfe Konstante A der Sorte $f\sigma$ auf eine (normalisierte) unscharfe Menge aus $m(A) \in F(U_\sigma)$.*

Für die Definition von possibilistischen Modellen als Möglichkeitsverteilung über Interpretationen ist es notwendig, dass wir uns auf diejenigen Interpretationen beschränken, denen eine gemeinsame Interpretation der unscharfen Konstanten zugrunde liegt. Nur dann ist es sinnvoll, die Sicherheit possibilistischer Klauseln mit unscharfen Konstanten anzugeben.

Definition 3.13 (Kontext)
Seien U eine Domäne und m eine Interpretation von Objektkonstanten über U (oder über $[0,1]^U$ im Falle von unscharfen Konstanten). Außerdem sei gegeben, dass m unterschiedliche Objektkonstante auf unterschiedliche Elemente in U abbildet. Der Kontext*, der durch U und m bestimmt wird, ist definiert als die Menge*

$$\Omega_{U,m} = \{w \in \Omega | w = (U, i, m)\}.$$

Definition 3.14 (Possibilistische Erfüllbarkeit)
Seien $\Omega_{U,m}$ ein Kontext und π eine Möglichkeitsverteilung für die gilt $\pi : \Omega_{U,m} \to [0,1]$. Dann ist die Erfüllbarkeitsrelation $\models^{U,m}_{PLFC}$ wie folgt definiert:

$$\pi \models^{U,m}_{PLFC} (\varphi, \alpha) \iff N([\varphi]|\pi) \geq \alpha$$

mit dem erweiterten Notwendigkeitsmaß

$$N([\varphi]|\pi) = \inf_{w \in \Omega_{U,m}} \max(1 - \pi(w), q(\varphi)).$$

Im Allgemeinen ist eine possibilistische Formel mit variablen Gewichten wie folgt definiert:

Definition 3.15 (Possibilistische Klausel mit variablen Gewichten)
Sei $\varphi(\overline{x})$ eine possibilistische Klausel mit freien Variablen $\overline{x}$ und f eine Funktion mit der Stelligkeit $|\overline{y}|$, wobei $\overline{y}$ eine Untermenge von $\overline{x}$ ist. Die Funktion f ist in einem gegebenen Kontext berechenbar sobald alle Variablen in $\overline{y}$ instantiiert sind und sie deren Werte auf $[0,1]$ abbildet. Eine possibilistische Klausel mit variablen Gewichten *hat dann die Form*

$$(\varphi(\overline{x}), f(\overline{y}))$$

Beispiel 3.4 *Gegeben sei die folgende PLFC-Klausel*

$$(p(A, x) \vee q(y), \min(\alpha, B(x), C(y))),$$

wobei A eine Objektkonstante und B und C unscharfe Konstanten sind. Der Kontext (U, m) bildet die unscharfen Konstanten auf die Elementfunktionen $\mu_{m(B)}$ und $\mu_{m(C)}$ ab.

Die möglichen Funktionen f, die die possibilistische Bewertung einer PLFC-Klausel repräsentieren, sind entweder Konstanten, Elementfunktionen, max-min-Kombinationen von Konstanten und Elementfunktionen oder Notwendigkeitsmaße über solche Kombinationen. Damit sind die Funktionen, die als possibilistische Bewertungen entstehen können, direkt als Annotationsfunktionen darstellbar.

Definition 3.16 (Transformierte PL-Klausel mit var. Gewichten) *Sei p eine possibilistische Klausel mit variablen Gewichten der Form $(p_0 \leftarrow p_1 \wedge \cdots \wedge p_n, f(\overline{y}))$, $\overline{y}$ eine Untermenge von $\overline{x}$, der freien Objektvariablen in p und $v_i, \ldots, v_n$ eine Menge paarweiser disjunkter Annotationsvariablen, dann ist die annotierte Klausel $p^a : p_0 : \min(f(\overline{y}), x_1, \ldots, x_n) \leftarrow p_1 : x_1 \wedge \cdots \wedge p_n : x_n$ die* transformierte possibilistische Klausel mit variablen Gewichten *von p.*

Bei der possibilistische Logik mit variablen Gewichten bleibt die Vollständigkeit der Resolution erhalten.

Theorem 3.3 *Theoreme 3.1 und 3.2 gelten für transformierte possibilistische Programme mit variablen Gewichten.*

Der Beweis verläuft analog zum Beweis der o.g. Theoreme. Der Unterschied besteht in der Art der funktionalen Terme, die als Annotationen in den Klauselköpfen auftauchen. Zu beachten ist lediglich, dass Objektvariablen im Klauselrumpf durch Resolution verschwinden können und dadurch möglicherweise ungebundene Variablen im Annotationsterm des Klauselkopfes stehen bleiben. Dieses Problem kann so auch in der possibilistischen Logik mit variablen Gewichten entstehen. Eine possibilistische Klausel der Form

$$(\Delta(\overline{y}), v(x, \overline{y}))$$

wird dann in

$$(\Delta(\overline{y}), \max_{x \in X} v(x, \overline{y}))$$

transformiert, gegeben Δ als Disjunktion von Literalen über die Objektvariablen in $\overline{y}$, x eine Variable der Domäne X, die durch Resolution eliminiert wurde, und v ein Funktionsterm. Entsprechendes gilt auch für die annotierte Logik: Ein Annotationsterm der Form $\forall x : f(x)$ wird durch $\max_x f(x)$ ersetzt. Dieses Transformation wird trivialerweise durch die Reduktionsregel gerechtfertigt.

3.3.2 Unscharfe Konstanten

Unscharfe Konstanten können benutzt werden, um typische unscharfe Aussagen zu modellieren, wie z.B. „Die Außentemperatur ist hoch“. Diese Aussage wird in PLFC repräsentiert durch *out_temp(high)*, wobei *out_temp* ein klassisches Prädikat und *high* eine verallgemeinerte Konstante sind. Bezeichnet *high* eine scharfe Menge, dann wird *out_temp(high)* insbesondere als „$\exists x \in$ *high* so dass *out_temp(x)* wahr ist“ interpretiert. Unscharfe Konstanten können also als Beschränkung eines Existenzquantors betrachtet werden. Repräsentiert die Konstante eine unscharfe Menge, so kann man von einer elastischen Beschränkung des Existenzquantors sprechen.

Folgende Beispiele illustrieren die Ausdrucksmöglichkeiten der PLFC:

- $(p(x) \leftarrow, min(\mu_A(x), \alpha))$, „Mit Sicherheit α gilt $\forall x \in A, p(x)$“
- $(p(B) \leftarrow, \alpha)$„Mit Sicherheit α gilt $\exists x \in B, p(x)$“
- $(r(C) \leftarrow p(A) \wedge q(y), \mu_B(y))$, „Es gilt $\exists x \in A, \forall y \in B, \exists z \in C, \neg p(x) \vee \neg q(y) \vee r(z)$“

Unscharfe Konstanten unterscheiden sich grundsätzlich von herkömmlichen Objektkonstanten. Sie konstatieren die *Existenz* eines Wertes der (unscharfen) Menge die durch die Konstante bezeichnet wird. An jeder Stelle, an der herkömmliche Objektkonstanten gestattet sind, können in PLFC auch unscharfe Konstanten auftreten, sofern die assoziierte unscharfe Menge innerhalb der entsprechenden Domäne liegt. Insbesondere können Variablen mit unscharfen Konstanten unifiziert werden. Werden variable Gewichte und unscharfe Konstanten kombiniert, kann durch Resolution die Konstellation auftreten, dass eine Elementfunktion einer unscharfen Menge bezüglich einer unscharfen Konstante ausgewertet werden muss. In der *generalisierten Resolutionsregel* aus [AGS99] wird dieser Umstand berücksichtigt:

Seien A, B, C, D, E unscharfe Konstanten, x, y, z, r Variablen, b eine Objektkonstante und α und β konstante possibilistische Bewertungen. φ und ψ seien Klauseln.

$$R^{PLFC}: \frac{(\neg p(x,b,y) \vee \psi(x,y), \min(A(x), B(y), \beta)) \quad (p(C,z,r) \vee \varphi(z,r), \min(D(z), E(r), \alpha))}{(\psi(C,r) \vee \varphi(b,r), \delta(r))}$$

mit

$$\delta(r) = min(\beta, \alpha, B(r), E(r), D(b), N(A \mid [C]_{min(D(b),E(r),\alpha)}))$$

und

$$N(A \mid B) = \inf_x \max(1 - \mu_B(x), \mu_A(x))$$

Die allgemeine Form der Resolutionsregel in Regelschreibweise lässt sich unmittelbar daraus ableiten.

Definition 3.17 (Generalisierte Resolutionsregel der PLFC) *Sei v_q ein Term der Form* $\min(\alpha, F_{q_1}(x_1), \ldots, F_{q_n}(x_n))$ *und v_p ein Term der Form* $\min(\alpha, F_{p_1}(y_1), \ldots, F_{p_m}(y_m))$ *wobei F_{r_i} die charakteristische Funktion der entsprechenden unscharfen Konstanten darstellt. Sei θ der allgemeinste Unifikator der die Literale q_i und p_0 unifiziert. Die Resolutionsregel hat dann die Form*

$$R_*^{PLFC}: \frac{(\leftarrow q_1 \wedge \cdots \wedge q_n, v_q) \quad (p_0 \leftarrow p_1 \wedge \cdots \wedge p_m, v_p)}{(\leftarrow q_1 \wedge \cdots \wedge q_{i-1} \wedge p_1 \wedge \cdots \wedge p_m \wedge q_{i+1} \wedge \cdots \wedge q_n, v_s)\theta}$$

mit

$$v_s = \min(\alpha, \beta, F_{s_1}(x_1), \ldots, F_{s_i}(x_i), F_{s_{i+1}}([x_{i+1}]_{v_q}), \ldots, F_{s_l}([x_l]_{v_q}))$$

wenn $\forall x \in \{x_1, \ldots, x_i\}, \exists s = \{x/v\} \in \theta : v$ *ist eine Variable oder eine gewöhnliche Konstante und* $\forall x \in \{x_{i+1}, \ldots, x_l\}, \exists s = \{x/w\} \in \theta : w$ *ist eine unscharfe Konstante.*

Ein Ausdruck $F_1(F_2)$ *wird dann ausgewertet zu* $N(F_1 \mid F_2)$.

Zur Vervollständigung benötigen wir eine ebenfalls generalisierte Fusionsregel, die sich aber direkt aus der possibilistischen Fusionsregel aus Definition 3.8 ableiten lässt.

Definition 3.18 (Generalisierte Fusionsregel der PLFC)
Seien p und q Einheitsklauseln der PLFC und v_p *und* v_q *deren possibilistische Bewertungen. Sei* θ *der allgemeinste Unifikator, der p und q unifiziert.*

$$F_*^{PLFC} : \frac{(p, v_p), (q, v_q)}{[(p, \max(v_p, v_q))]\theta}$$

ist die generalisierte Fusionsregel der PLFC.

Im Unterschied zur Fusionsregel aus Definition 3.8 können nun die possibilistischen Bewertungen selbst funktionale Terme sein, die sich aus Konstanten, Variablen, variablen Gewichten und aus Minimum- und Maximumfunktionen zusammensetzen können.

Die funktionalen Ausdrücke, die bei der annotierten Logik im Klauselkopf verwendet werden, sind in der Art ihres Aufbaus nicht eingeschränkt. Daher lässt sich die generalisierte possibilistische Resolutionsregel durch Einsetzen des entsprechenden Funktionsterms aus der Definition 3.17 bei der Transformation eines possibilistischen Programms realisieren. Das ändert nichts Grundsätzliches an den Aussagen der Theoreme 3.1 und 3.2.

Beispiel 3.5 *Sei* $\mathcal{F}$ *eine possibilistische Wissensbasis, bestehend aus den possibilistischen Klauseln* $\{R1, R2, R3, F1\}$*, wobei die Prädikate und Mengen wie folgt interpretiert werden:*

$P(x, y)$:	*„x geht Baden am Strand y“*
$T(x, y)$:	*„x hat einen Bräunung der Intensität y“*
$M(x)$:	*„x geht in ein Solarium“*
F :	*Menge der Städte in Frankreich*
S :	*Menge der Städte in Spanien*
C :	*Menge der Städte in Katalonien*
normal :	*Unscharfe Konstante über den Bräunegrad*
stark :	*Unscharfe Konstante über den Bräunegrad*

Die Klauseln R1,R2,R3,F1 seien gegeben durch:

$$\begin{array}{lrcl} R1: & (T(x,\text{normal}) & \leftarrow & P(x,y), \min(\mu_F(y),\alpha) \\ R2: & (T(x,\text{stark}) & \leftarrow & P(x,y), \min(\mu_S(y),\beta) \\ R3: & (T(x,\text{normal}) & \leftarrow & M(x),\gamma) \\ F1: & (P(\mathit{Teresa},C),1) & \leftarrow & \end{array}$$

Die Anfrage „Wie stark ist Teresas Bräunung?“ wird mit folgender Anfrageklausel formalisiert:

$$Q0: \quad (\quad \leftarrow \quad T(\text{Teresa},z),1)$$

Die Transformation des possibilistischen Programms aus Beispiel 3.5 erzeugt folgendes Programm der annotierten Logik:

$$\begin{array}{lrcl} R1^a: & T(x,\mathit{normal}) : \min(\mu_{\text{F}}(y),\alpha,\Delta) & \leftarrow & P(x,y):\Delta \\ R2^a: & T(x,\mathit{stark}) : \min(\mu_{\text{S}}(y),\beta,\Delta) & \leftarrow & P(x,y):\Delta \\ R3^a: & T(x,\mathit{normal}) : \min(\gamma,\Delta) & \leftarrow & M(x):\Delta \\ F1^a: & P(\text{Teresa},\text{C}):1 & \leftarrow & \end{array}$$

sowie die annotierte Anfrage:

$$Q0^a: \quad \leftarrow y_0 \preceq y_1 \parallel T(\text{Teresa},z) : y_1$$

Bei der Resolution ergeben sich (in vereinfachter Schreibweise) folgende Widerlegungen:

$$\begin{array}{lrcll} \langle Q0^a, R1^a\rangle: & \leftarrow & P(\text{Teresa},z):\Delta & & \\ & & y_1 \preceq \min(\mu_F(y),\alpha,\Delta) & & \langle Q1^a\rangle \\ \langle Q1^a, F1^a\rangle: & \leftarrow & y_1 \preceq \min(N(F \mid [C]_\alpha),\alpha), z=\text{normal} & & \langle Q2^a\rangle \\ \langle Q0^a, R2^a\rangle & \leftarrow & P(\text{Teresa},z):\Delta & & \\ & & y_1 \preceq \min(\mu_S(y),\beta,\Delta) & & \langle Q2^a\rangle \\ \langle Q2^a, F1^a\rangle & \leftarrow & y_1 \preceq \min(N(S \mid [C]_\beta),\beta), z=\text{stark} & & \langle Q4^a\rangle \end{array}$$

Sowohl bei $Q2^a$ als auch bei $Q4^a$ ergibt sich ein maximaler wert von 0 für y_1.

3.3.3 Vollständigkeit der PLFC

Die Auswertung des Beispiels 3.5 im vorigen Abschnitt zeigt, dass die PLFC-Resolution alleine in der vorgestellten Form nicht vollständig ist. Das optimale Ergebnis, das die Semantik der PLFC erwarten lässt, wäre

$$z = \text{normal} \vee \text{stark}$$

mit der Notwendigkeit

$$N(Q0) = \min(\mu_{S \cup F}(C), \alpha, \beta) = \min(\alpha, \beta)$$

Deshalb wird für die PLFC eine weitere Inferenzregel benötigt, die die Verschmelzung von unscharfen Konstanten und variablen Gewichten erlaubt. In [AGS99] wird eine *generalisierte Verschmelzungsregel* (engl.: *generalized merging rule*) der PLFC wie folgt vorgeschlagen: Sei $(\varphi(\overline{x}), f_1(\overline{x}))$ eine Variante von $(\varphi(\overline{y}), f_2(\overline{y}))$, d.h. es existiere eine Substitution θ, die nur Abbildungen von Variablen in $\overline{x}$ auf Variablen in $\overline{y}$ beinhaltet. Seien $f_1(\overline{x})$ und $f_2(\overline{y})$ gültige Bewertungsfunktionen über die Variablen in $\overline{x}$ bzw. $\overline{y}$, dann ist die *generalisierte Verschmelzungsregel* wie folgt definiert:

$$\frac{(\varphi(\overline{x}), f_1(\overline{x})), (\varphi(\overline{y}), f_2(\overline{y}))}{(\varphi(\overline{y}), \max(f_1(\overline{x}))\theta, f_2(\overline{y})))}$$

Die Anwendung dieser Regel würde allerdings für die Auswertung des Beispiels 3.5 keine höhere Bewertung bringen, da die Verschmelzung von unscharfen Konstanten nicht berücksichtigt wird. Auch in einem weiteren Punkt ist obige Regel unzulänglich. Die Anwendung ist auf Varianten von Klauseln ausgerichtet und leitet sich von der Fusionsregel ab. Wie das folgende Beispiel zeigt, ist die Verschmelzung von variablen Gewichten auch bei Klauseln möglich, die keine Varianten sind. Lediglich das Atom, in dem die Variable gebunden wird, die in den zu verschmelzenden Funktionstermen auftritt, muss übereinstimmen.

Beispiel 3.6 *Gegeben sei das Programm aus Beispiel 3.5. Statt der Regeln $R1$ und $R2$ seien die folgenden Regeln $R1'$ und $R2'$ sowie $F2$ und $F3$ enthalten.*

$$\begin{array}{lrcl}
R1': & (T(x, \text{normal}) & \leftarrow & P(x,y) \wedge L_1(y), \min(\mu_F(y), \alpha) \\
R2': & (T(x, \text{stark}) & \leftarrow & P(x,y) \wedge L_2(y), \min(\mu_S(y), \beta) \\
F2: & (L_1(C), 1) & \leftarrow & \\
F3: & (L_2(C), 1) & \leftarrow &
\end{array}$$

Sieht man von der Verwendung der unscharfen Konstanten in den Klauselköpfen von $R1'$ und $R2'$ ab, sind die Klauseln aufgrund unterschiedlicher Rumpfliterale immer noch nicht Varianten im Sinne der generalisierten Verschmelzungsregel.

Zur Lösung der genannten Probleme schlagen wir eine *erweiterte Verschmelzungsregel* für PLFC vor. Das folgende Lemma zeigt zunächst den Zusammenhang zwischen der Konjunktion auf der logischen Ebene und der Vereinigung unscharfer Konstanten.

Lemma 3.1 *Seien $\varphi = (a(x_0, \ldots, x_n), v)$ und $\psi = (a(y_0, \ldots, y_n), w)$ zwei possibilistische Einheitsklauseln, wobei deren Argumente x_i bzw. y_i entweder Variablen, gewöhnliche Konstanten oder unscharfe Konstanten sind. Sei θ der allgemeinste Unifikator der diejenigen Argumente unifiziert, die entweder Variablen oder aber gewöhnliche Konstanten sind. Existiert ein solches θ, dann gilt*

$$\varphi, \psi \models a(z_0, \ldots, z_n), \min(v, w)$$

mit

$$z_i = \begin{cases} [x_i]\theta & \text{wenn } x_i \text{ eine Variable ist,} \\ x_i & \text{wenn } x_i \text{ eine gewöhnliche Konstante ist,} \\ x_i \cup y_i & \text{wenn } x_i \text{ und } y_i \text{ unscharfe Konstanten sind.} \end{cases}$$

Beweis 3.3 *Sei π eine Möglichkeitsverteilung über Formeln und φ und ψ zwei possibilistische Formeln, die abgesehen von den unscharfen Konstanten F und G, die an der gleichen Stelle in φ und ψ auftreten, unifizierbar sind. Der PLFC-Term $\varphi(F)$ ist äquivalent zu $\exists x \in F : \varphi(x)$. Wenn $\pi \models (\exists x \in F : \varphi(x), v)$ und $\pi \models (\exists y \in G : \psi(y), w)$ dann gilt sicher auch $\pi \models (\exists x \in F, \exists y \in G : \varphi(x) \wedge \psi(y), \min(v, w))$. Sind φ und ψ unifizierbar mittels des allgemeinsten Unifikators θ, dann gilt $\pi \models (\exists x \in F \cup G : [\varphi(x)]\theta, \min(v, w))$, was äquivalent zu $[(\varphi(F \cup G), \min(v, w))]\theta$ ist.* □

Die erweiterte Verschmelzungsregel basiert auf dieser Eigenschaft.

Definition 3.19 (Erweiterte Verschmelzungsregel der PLFC) *Seien p und q possibilistische Klauseln der Form*

$$p : (p_0(\Lambda_0, F_p) \leftarrow p_i(\Lambda_i, x) \wedge \varphi(\overline{x}), \min(F_{p_i}(x), f_p(\overline{x}))$$

bzw.

$$q : (q_0(\Gamma_0, F_q) \leftarrow q_j(\Gamma_j, y) \wedge \psi(\overline{y}), \min(F_{q_i}(y), f_q(\overline{y}))$$

mit Λ_i (Γ_j) die Menge der Argumente des Literals p_i (q_j) ohne die Menge der unscharfen Konstanten in F_p (F_q) bzw. der Variablen x (y) aus $\overline{x}$ ($\overline{y}$). Sei $\varphi(\overline{x})$ ($\psi(\overline{y})$) eine Konjunktion von Literalen über die Variablen in $\overline{x}$ ($\overline{y}$). Existiert eine Substitution θ, die p_0 und q_0 abgesehen von den unscharfen Konstanten in F_p und F_q sowie p_i und q_i unifiziert, dann ist die erweiterte Verschmelzungsregel *so definiert:*

$$M_{PLFC} : \frac{p, q}{[(p_0(\Lambda_0, \{F_{p_j} \cup F_{q_j}\}) \leftarrow p_i(\Lambda_i, x) \wedge \varphi(\overline{x}) \wedge \psi(\overline{y}), \delta)]\theta}$$

mit

$$\delta = \min(\mu_{F_{p_i} \cup F_{q_i}}(x), f_p(\overline{x}), f_q(\overline{y}))$$

Theorem 3.4 (Korrektheit der erweiterten Verschmelzungsregel) *Sei π eine Möglichkeitsverteilung über Formeln. Wenn $\pi \models \varphi$ und $\pi \models \psi$ im Sinne der Definition 3.19, dann gilt auch $\pi \models M_{PLFC}(\varphi, \psi)$.*

Beweis 3.4 *Die Behauptung kann mit Hilfe von Lemma 3.1 und der possibilistischen Fusionsregel durch einfaches Umformen gezeigt werden.*

Die erweiterte Verschmelzungsregel eröffnet die Möglichkeit, direkt aus den Programmklauseln die benötigten Klauseln zu erzeugen. Für das erweiterte Beispiel 3.6 würde die Anwendung der erweiterten Verschmelzungsregel folgende Klausel hervorbringen.

$$R4: \quad (T(x, normal \vee stark) \quad \leftarrow \quad P(x,y), \min(\mu_{F \cup S}(y), \alpha, \beta))$$

Die transformierte Form von R4 ist dann

$$R4^a: \quad T(x, normal \vee stark) : \min(\mu_{F \cup S}(y), \alpha, \beta, \Delta) \quad \leftarrow \quad P(x,y) : \Delta$$

Die lineare Resolution des Programms $\{R1^a, R2^a, R3^a, R4^a, F1^a\}$ mit der initialen Anfrage $Q0^a$ ergibt neben den bereits gezeigten Widerlegungen folgenden Widerlegungspfad, der uns letztlich das gewünschte Ergebnis liefert.

$$\begin{array}{lll} \langle Q0^a, R3^a \rangle : & \leftarrow & P(\text{Teresa}, z) : \Delta \\ & & y_1 \preceq \min(\mu_{F \cup S}(y), \alpha, \beta, \Delta) \quad \langle Q5^a \rangle \\ \\ \langle Q5^a, F1^a \rangle : & \leftarrow & y_1 \preceq \min(\mu_{F \cup S}(C), \alpha, \beta, 1), \\ & & z = \text{normal} \vee \text{stark} \end{array}$$

3.4 Realisierung in KOMET

Zur Formulierung von Programmen der PLFC wird die Repräsentation von Annotationstermen benötigt, die auch Objektvariablen enthalten können. Prinzipiell können solche Terme in KOMET als Randbedingungen formuliert werden. Allerdings verlangt eine Einschränkung der Implementierung von KOMET, dass die Randbedingungen von Einheitsklauseln vollständig lösbar sind. In der gegenwärtigen Implementierung wurde auf die Verarbeitung von Einheitsklauseln mit Randbedingungen zugunsten einer größeren Effizienz verzichtet, sie ist jedoch nicht grundsätzlich ausgeschlossen. Eine andere Möglichkeit wäre die Verwendung eines Verbandstyps, der selbst in

der Lage ist, Annotationsterme darzustellen und symbolisch zu verarbeiten. In diesem Abschnitt zeigen wir, inwieweit PLFC-Programme unter Verwendung von Randbedingungen in KOMET ausgedrückt werden können. Die generalisierte Resolutionsregel der PLFC aus Definition 3.17 verlangt bei der Resolution von unscharfen Konstanten die Durchführung von Schnitten unscharfer Mengen. Dabei muss möglicherweise ebenfalls ein komplexer Term berücksichtigt werden, der die Schwelle des Schnitts in Abhängigkeit von den bislang nicht instantiierten Variablen beschreibt. Eine Vereinfachung lässt sich zunächst durch eine Einschränkung erzielen. Beschränken wir uns bei unscharfen Konstanten auf diejenigen, deren charakteristische Funktion für jedes Element der Grundmenge entweder 0 oder 1 ist, dann gilt für alle α mit $0 < \alpha \leq 1$ die Eigenschaft

$$N(A \mid [B]_\alpha) = N(A \mid B)$$

da $[B]_\alpha = B$ ist. Solche unscharfen Konstanten nennen wir *ungenaue Konstanten* (engl.: *imprecise constants*). Wenn die Notwendigkeit, Schnitte aufgrund komplexer Terme zu bilden entfällt, ergibt sich für die Resolutionsregel der PLFC in Definition 3.17 für die resultierende Bewertung v_s folgende vereinfachte Form:

$$v_s = \min(\alpha, \beta, F_{s_1}(x_1), \ldots, F_{s_l}(x_l)),$$

wobei die Elemente aus $\{x_1, \ldots, x_l\}$ Variablen, gewöhnliche Konstanten oder ungenaue Konstanten sind. Die Berechnung von $N(A \mid B)$ vereinfacht sich entsprechend zu

$$N(A \mid B) = \begin{cases} 1 & \text{falls } B \subseteq A \\ 0 & \text{sonst} \end{cases}$$

Für die Implementierung in KOMET genügt es dann, an Stelle einer erweiterten Sorte, deren Konstanten entweder gewöhnlich oder aber unscharf sein können, eine Sorte von Mengen über Elemente der Grundsorte einzuführen. Gewöhnliche Konstanten werden dann als einelementige Mengen dargestellt. Die charakteristische Funktion μ_A für eine gegebene ungenaue Konstante A wird durch den Teilmengentest realisiert. Für viele Anwendungen genügt die Repräsentation endlicher Mengen, die durch Aufzählung angegeben werden und mit der parametrisierbaren KOMET-Sorte FCSET realisiert wurden. Der Annotationsverband ist der Verband der reellen Zahlen Zwischen 0 und 1, mit ihrer natürlichen Ordnung.

Beispiel 3.7
Das Programm aus Beispiel 3.5 wird durch folgendes KOMET-Programm repräsentiert.

```
#sorts
LOC = FCSET(STRING,
    s={Blanes,Barcelona},
    f={Colliours,Marseille},
    c={Blanes,Colliours})
DEG = FCSET(ULONG,
    wenig={1},
    normal={2},
    stark={3})

#predicates
T(STRING,DEG):[PL]
P(STRING,LOC):[PL]
M(STRING):[PL]

#clauses
T(X,normal):[PL::MIN(LOC::N(f,Y),V) <- P(X,Y):[V]
T(X,stark):[PL::MIN(LOC::N(s,Y),V) <- P(X,Y):[V]
T(X,DEG::OR(normal,stark)):[PL::MIN(LOC::N(LOC::OR(f,s),V) <-
    P(X,Y):[V]
T(X,normal):[V] <- M(X):[V]
P('Teresa',c):[1]
```

Für die Anfrage `T('Teresa',X):[V]` wird die Antwort

```
X = normal v stark, V = 1
```

berechnet.

Die Sorte FCINT repräsentiert unscharfe Konstanten, die über Intervalle reeller Zahlen definiert sind. Folgendes Programm illustriert die Verwendung von FCINT. In diesem Beispiel werden zunächst mehrere ungenaue Konstanten definiert, die linguistische Ausdrücke für verschiedene Temperaturbereiche repräsentieren. Die Regeln des Beispiels ermitteln für einen gegebenen Temperaturbereich die geeignete Freizeitbeschäftigung.

Beispiel 3.8 (FCINT)
Das folgende Programm macht Vorschläge zur Freizeitgestaltung

```
#sorts
TEMP = FCINT(sehr_heiss=[32,45],heiss=[26,33],warm=[17,26],
             kuehl=[11,18],kalt=[-2,12],bitter_kalt[-15,0])

#predicates
Temp(TEMP):[PL]
Freizeit(STRING):[PL]

#clauses
Freizeit('Wintersport'):[PL::MIN(TEMP::N(bitter_kalt,X),V)] <-
```

```
    Temp(X):[V]
Freizeit('Wintersport'):[PL::MIN(TEMP::N(kalt,X),V)] <-
    Temp(X):[V]
Freizeit('Wintersport'):[PL::MIN(TEMP::N(
                        TEMP::OR(bitter_kalt,kalt),X),V)] <-
    Temp(X):[V]
Freizeit('Jogging'):[PL::MIN(TEMP::N(kuehl,X),V] <-
    Temp(X):[V]
Freizeit('Spazieren'):[PL::MIN(TEMP::N(warm,X),V)] <-
    Temp(X):[V]
Freizeit('Sonnenbad'):[PL::MIN(TEMP::N(heiss,X),V)] <-
    Temp(X):[V]
Freizeit('Schwimmen'):[PL::MIN(TEMP::N(sehr_heiss,X),V)] <-
    Temp(X):[V]

Temp([-2,4]):[1]
```

Man beachte, dass in obigem Beispiel die 3. Regel, die durch Anwendung der erweiterten Verschmelzungsregel entstanden ist, die 1. und 2. Regel subsumiert. Insofern könnten diese Regeln weggelassen werden ohne die Semantik des Programms zu ändern. Das kommt daher, dass den beiden Regeln implizit die gleiche Notwendigkeit (hier 1) zugeordnet ist, so dass sich dadurch keine unterschiedlichen Bewertungen ergeben können.

Die Realisierung allgemeiner unscharfer Mengen wird dadurch kompliziert, dass im Allgemeinen die charakteristischen Funktionen unscharfer Mengen beliebig sein können[1]. Insofern müssten beliebige mathematische Funktionen darstellbar sein. Für die Mehrzahl der Anwendungen ist es ausreichend, charakteristische Funktionen auf bestimmte Typen von Funktionen einzuschränken. Wir wollen im letzten Abschnitt dieses Kapitel unscharfe Mengen betrachten, deren charakteristische Funktionen durch so genannte *Trapezfunktionen* beschrieben werden.

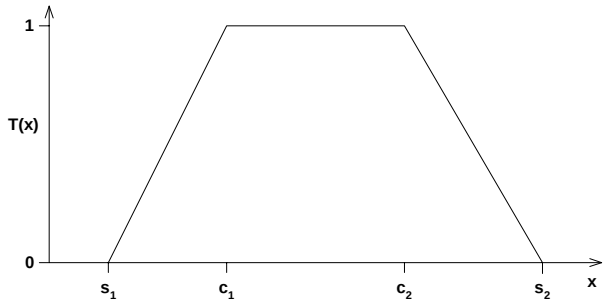

Abbildung 3.1: Trapezfunktion

[1]Wir gehen von Zugehörigkeitsfunktionen aus, deren Bildbereich das Intervall $[0, 1]$ ist

Eine Trapezfunktion wird stückweise durch 5 einfache lineare Funktionen beschrieben und hat im Allgemeinen die Form eines Trapezes. Im so genannten *Kern*, der zwischen den Werten c_1 und $c_2$liegt, berechnet sich eine Trapezfunktion zu 1, bis zum Startwert s_1 und ab dem Endwert s_2 zu 0. Eine Trapezfunktion lässt sich so durch die vier Parameter s_1, c_1, c_2, s_2 beschreiben. Die Funktionsgleichung der Trapezfunktion lautet dann:

$$T(s_1, c_1, c_2, s_2, x) = \begin{cases} 0 & x \leq s_1, x \geq s_2 \\ 1 & c_1 \leq x \leq c_2 \\ \frac{x-s_1}{c_1-s_1} & s_1 < x < c_1 \\ \frac{x-c_2}{s_2-c_2} & c_2 < x < s_2 \end{cases}$$

Die Darstellung mittels 4 Parameter ist allerdings noch nicht allgemein genug um Verknüpfungen solcher unscharfer Mengen darzustellen. Durch Operation wie die Bildung von Minimum und Maximum kann eine komplexere Funktion entstehen, die jedoch immer aus Stücken linearer Funktionen besteht. Daher wird für die KOMET-Sorte FCTRF als Repräsentation ein Vektor von Paaren gewählt, der die Schnittpunkte zwischen jeweils zwei aneinander grenzenden linearen Teilfunktionen enthält. Die zur Kombination zweier unscharfer Mengen dieser Art benötigten Funktionen sind durch klassische geometrische Algorithmen mit dem *Scan line*-Verfahren[2] zu realisieren, die linearen Aufwand in Abhängigkeit der Anzahl der Stützpunkte aufweisen.

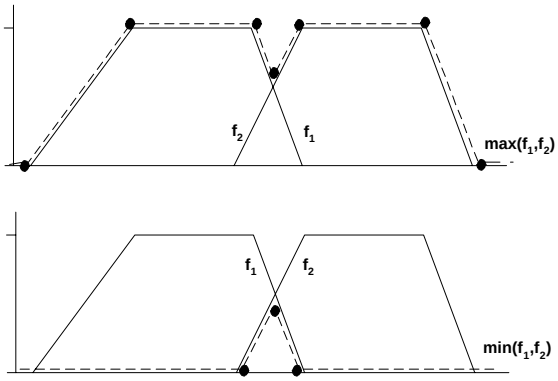

Abbildung 3.2: min und max-Funktion bei Trapezfunktionen mit resultierenden Schnittpunkten

Dies ist mit linearem Aufwand in Abhängigkeit von der Anzahl der Punkte, aus denen die unscharfen Mengen der Argumente bestehen, durchführbar.

[2]Beim *Scan line*-Verfahren stellt man sich eine vertikale Linie vor, die den Definitionsbereich des Problems horizontal überstreicht. Entsprechend der dabei entstehenden Reihenfolge werden die angetroffenen Raumpunkte verarbeitet.

Um die Resolutionsregel der Definition 3.17 realisieren zu können wird eine Funktion benötigt, die für unscharfe Mengen α-Schnitte berechnet, d.h. diejenige scharfe Menge, deren Elemente eine Zugehörigkeit zu der Argumentmenge von größer oder gleich α haben. Die Regeln des PLFC-Programms werden dann entsprechend der Definition 3.17 transformiert.

Beispiel 3.9 (FCTRF)
Das folgende Programm ist eine Verfeinerung des vorangegangenen Programms, in dem unscharfe Konstanten durch Trapezfunktionen repräsentiert werden.

```
#sorts
TEMP = FCTRF(sehr_heiss=<31,34,45,50>,heiss=<25,27,32,24>,
             warm=<15,17,25,26>,kuehl=<8,11,15,18>,
             kalt=<-2,0,7,12>,bitter_kalt<-20,-15,-5,0>)

#predicates
Temp(TEMP):[PL]
Freizeit(STRING):[PL]

#clauses
Freizeit('Ski'):[PL::MIN(TEMP::N(bitter_kalt,TEMP::CUT(X,V)),V)] <-
    Temp(X):[V]
Freizeit('Ski'):[PL::MIN(TEMP::N(kalt,X),V)] <-
    Temp(X):[V]
Freizeit('Ski'):[PL::MIN(TEMP::N(
    TEMP::OR(bitter_kalt,kalt),TEMP::CUT(X,V)),V)] <- Temp(X):[V]
Freizeit('Jogging'):[PL::MIN(TEMP::N(kuehl,TEMP::CUT(X,V)),V] <-
    Temp(X):[V]
Freizeit('Spazieren'):[PL::MIN(TEMP::N(warm,TEMP::CUT(X,V)),V)] <-
    Temp(X):[V]
Freizeit('Sonnenbad'):[PL::MIN(TEMP::N(heiss,TEMP::CUT(X,V)),V)] <-
    Temp(X):[V]
Freizeit('Schwimmen'):[PL::MIN(TEMP::N(sehr_heiss,
    TEMP::CUT(X,V)),V)] <- Temp(X):[V]

Temp(<-4,-2,4,6>):[0.9]
```

Eine Anfrage des Prädikats `Freizeit` ergibt die Antwort *Ski* mit einer Notwendigkeit von 0.9.

KOMET ist prinzipiell offen für die Einbindung weiterer Arten von Zugehörigkeitsfunktionen und damit Repräsentationen unscharfer Mengen. Je nach Anwendung ist es auch möglich, die unterschiedlichen Typen von unscharfen Mengen in einem Programm gemischt zu verwenden. Da jedoch jeder Typ einer unscharfen Menge oder Konstante als eigene Sorte in KOMET realisiert werden muss und unterschiedliche Sorten strikt unterschieden werden, ist die direkte Austauschbarkeit nicht immer möglich. In diesem Fall es dann aber möglich, entsprechende Konvertierungsfunktionen zu implementieren.

In diesem Kapitel wurde die possibilistische Logik als Mittel zur Repräsentation unsicheren Wissens vorgestellt. Es wurde untersucht wie possibilistische Programme mit der annotierten Logik dargestellt und mit Hilfe deren Inferenzmechanismen verarbeitet werden können. Für eine interessante Erweiterung der possibilistischen Logik um Elemente der unscharfen Logik konnte ebenfalls gezeigt werden, dass sie in die annotierte Logik eingebettet werden kann. Durch die Einführung der *erweiterten Verschmelzungsregel* für die PLFC konnte ein Beitrag zur Vervollständigung der possibilistischen Resolution mit variablen Gewichten und unscharfen Konstanten geleistet werden. Schließlich wurde demonstriert in welcher Form die Elemente der possibilistischen Logik in KOMET realisiert wurden. Die Ergebnisse dieses Kapitels wurden in [KS99] und [KS01] veröffentlicht.

Es ist gewiss kein geringes Wagnis,
von zweifelhaftem Erfolg und äußerst gefährlich,
eine neue Ordnung einzuführen.

Nicollò Machiavelli

Kapitel 4

Eine Architektur für logikbasierte Mediatorsysteme

Mit der verallgemeinerten annotierten Logik besteht die Möglichkeit, komplexes Integrationswissen deklarativ zu formulieren und damit semantische Heterogenitäten zu behandeln. Das vorige Kapitel zeigt, wie in dieser Logik unsicheres Wissen dargestellt und verarbeitet werden kann. Mit der SLANG-Resolution steht prinzipiell ein Verfahren zur Auswertung entsprechender Mediatorprogramme zur Verfügung, die zur Familie der linearen Resolutionen gehört. Eine *top-down*-Auswertung logischer Programme besitzt einige Eigenschaften, die im Umfeld heterogener, entfernter Informationsquellen besonders wünschenswert sind. Andererseits sind solche Resolutionsverfahren per se nicht optimal für alle Anforderungen einer solchen Umgebung geeignet.

Zunächst werden typische Eigenschaften einer Umgebung mit verteilten, heterogenenen, autonomen Informationsquellen skizziert. Anschließend folgt eine Diskussion der Vor- und Nachteile der Verwendung einer linearen Resolutionsprozedur in diesem Umfeld. Dann wird eine Architektur vorgestellt, die auf auf der Basis der SLG-Resolution als Mediatorarchitektur geeignet ist.

4.1 Anforderungen an Integrationssysteme

Die Informationsquellen, auf die in einem Mediatorsystem zugegriffen wird, sind im Allgemeinen

- entfernt. Als typisches Beispiel bietet das WWW eine unüberschau-

bare Menge an Daten und Informationen, die in ein Mediatorsystem eingebunden werden können[1]. Zugriffe auf Dienste und Daten über das Internet sind in hohem Maße dadurch gekennzeichnet, dass in aller Regel kaum Aussagen über die Erreichbarkeit der Zieladresse bzw. Geschwindigkeit des Zugriffs gemacht werden können. Die Qualität der Netzwerkverbindung kann erheblich mit dem Zeitpunkt des Zugriffs variieren. Die Benutzung des Internets verursacht unter Umständen zeit- oder volumenbezogene Kosten.

- autonom. Die Verfügbarkeit einer Informationsquelle ist nicht sichergestellt. Daher kann eine Anfrage unvorhergesehen fehlschlagen. Abhängig von der Auslastung des Dienstes können weitere Latenzzeiten entstehen. Auch können sich Struktur oder Zugriffsmechanismus einer Informationsquelle unvorhergesehen ändern, so dass Zugriffe fehlschlagen.

- datenintensiv. Viele Integrationsaufgaben verlangen die Verarbeitung großer Datenmengen. Prinzipiell ist der Transport großer Datenmenge über Netzwerke kein Problem. Vielmehr stellt der zur Verfügung stehende Hauptspeicher für das Mediatorsystem eine Beschränkung dar. Prinzipiell jedoch sollte die Verarbeitung beliebiger Datenmengen möglich sein, um auch anspruchsvolle Auswertungen durchführen zu können, wie sie in den Bereichen *data mining* und *online analytical processing* (OLAP) üblich sind.

Diesen Eigenschaften muss in einem Mediatorsystem, das den Anspruch erhebt, allgemeine Integrationsaufgaben bewältigen zu können, Rechnung getragen werden.

4.2 Datenorientierte Sichtweise

Der Integrationsprozess lässt sich am besten als ein Prozess der Verarbeitung von *Mengen von Daten* beschreiben, bei dem ein Mediatorprogramm deklarativ festlegt, wie die aus den Informationsquellen stammenden Basisdaten verarbeitet werden. Die datenorientierte Sichtweise der logischen Programmierung ist durch die Trennung eines logischen Programms in *Regeln* einerseits und *Fakten* andererseits gekennzeichnet und ist der Ausgangspunkt der Forschung im Bereich der deduktiven Datenbanken. Der Regelteil kann auf unterschiedliche Faktenbasen angewandt werden. In einem Mediatorsystem gilt diese Trennung um so mehr, da die Basisfakten in der Regel aus entfernten Informationsquellen bezogen werden und somit auch physisch von

[1] Eine besondere Kategorie eines Mediatorsystems stellen so genannte *value-added web services* dar, die sich ausschließlich auf das Informationsangebot im WWW stützen und durch die Integration solcher Informationsquellen eine qualitative oder quantitative Verbesserung gegenüber den einzelnen Quellen erbringen.

den Regeln getrennt abgelegt und verwaltet werden. Typischerweise stehen in einem solchen System relativ wenigen Regeln eine große Zahl von Fakten gegenüber.

Auf linearer Resolution beruhende Auswertungsprozeduren werten logische Programme typischerweise von oben nach unten oder rückwärts verkettend, d.h. von der Anfrage her zu den Basisdaten hin, aus. Solche Verfahren werden daher als *top-down*-Verfahren oder *backward chaining*-Verfahren bezeichnet. Die Auswertung wird in der Regel tupelweise und durch Tiefensuche durchgeführt. Dadurch stehen in aller Regel erste Anfrageergebnisse zur Verfügung ohne dass der potenzielle Suchraum komplett bearbeitet werden muss. Das Prinzip, möglichst schnell ein erstes Ergebnis zu erzeugen, hat seinen Ursprung in der Verwendung von logischen Sprachen zur Automatisierung von Beweisen. Das Interesse bei klassischen Theorembeweisern liegt darin, überhaupt einen Beweis für eine Anfrage zu finden. Auch die Sprache PROLOG folgt primär diesem Prinzip. Bei der Auswertung in einem logikbasierten Mediatorsystem wird im Gegensatz dazu typischerweise die Berechnung aller Antworten verlangt. Daher ist wohl die Verwendung einer rückwärtsverkettenden Auswertungsprozedur nicht zwingend. Deduktive Datenbanken verwenden aus Effizienzgründen meist vorwärtsverkettende Auswertungsprozeduren.

Trotzdem haben die Eigenschaften von *top-down*-Verfahren mit Tiefensuche in Anbetracht der Besonderheiten eines Mediatorsystems Vorteile, die wünschenswerte Fähigkeiten ermöglichen:

- Ein Teil der Ergebnisse kann bereits angezeigt oder weiterverarbeitet werden, obwohl der Auswertungsprozess noch nicht abgeschlossen ist. Dadurch kann Parallelität auf der Prozessebene erreicht werden und der Mediator im Zusammenspiel mit anderen Teilen einer Anwendung als Verarbeitungskette(engl.: *pipeline*) organisiert werden.

- Die Verarbeitung kann abgebrochen werden. Die Korrektheit der bislang errechneten Ergebnisse ist gegeben. In verschiedenen Situation ist diese Möglichkeit erforderlich:

  - Für die Beantwortung der Anfrage sind wenige Antworten ausreichend. Dem Anwender kommt es nicht darauf an, alle oder die besten Ergebnisse zu bekommen, sondern einige wenige Antworten sind ausreichend.
  - In einem System sind redundante Informationsquellen beteiligt, die unterschiedliches Antwortzeitverhalten zeigen. Das Antwortverhalten kann sich in einer autonomen Umgebung wie dem Internet unvorhergesehen ändern. In diesem Fall sollte es möglich sein, die Auswertung abzubrechen, ohne auf bereits berechnete Ergebnisse verzichten zu müssen.

– An das Mediatorsystem werden Echtzeitanforderungen gestellt, d. h. es sind Zeitschranken vorgegeben. In diesem Fall soll das System die Ergebnisse liefern, die es innerhalb einer gewissen Zeitspanne berechnen kann.

Bei reinen linearen Resolutionsverfahrens stellen zum einen die Verarbeitung rekursiver Programme zum anderen die wiederholte Berechnung von Unterzielen eine Schwierigkeit dar. Diese lässt sich jedoch durch die Zwischenspeicherung von Unterzielauswertungen (*Tabulierung*) beheben. Die Verwendung von Tabulierung wird im nächsten Abschnitt diskutiert.

4.3 Tabulierung bei linearer Resolution

Die Tabulierung in *top-down*-Beweisprozeduren wurde ursprünglich vorgeschlagen um die Terminierung rekursiver Programme zu gewährleisten und dadurch ihre Berechnung erst möglich zu machen [TS86]. Die klassische SLD-Resolution terminiert nicht zwingend bei der Auswertung rekursiver Programme. Die grundsätzliche Idee bei der Tabulierung ist, dass die Neuberechnung von Antwortsubstitutionen vermieden werden kann, wenn ein Unterziel bereits berechnet wurde oder die Berechnung im Gange ist. Das neue Unterziel kann auf die Ergebnisse des existierenden Unterziels zurückgreifen. Der Nutzen von Tabellen ist aber nicht auf rekursive Programme beschränkt. Vielmehr gibt es eine ganze Reihe von Aspekten, die die Bedeutung der Tabulierung unterstreichen. Letztendlich resultiert daraus eine tabulierungszentrierte Sicht des Inferenzsystems, die zu einer entsprechenden Architektur führt.

4.3.1 Aspekte der Tabulierung

Es stellt sich heraus, dass ein Tabulierungsmechanismus der zentrale Ansatzpunkt für eine Reihe von Mechanismen ist, die die genannten Anforderungen an ein Mediatorsystem erfüllen sollen. Allerdings ist die Tabulierung, wie sie in [TS86] beschrieben ist spezialisiert für das Problem der Terminierung. Die Anforderungen weiterer Nutzungsmöglichkeiten dieses Mechanismus verlangen die Konzeption einer geeigneten Datenstruktur und Einbettung in das Gesamtsystem. Der Tabulierungsmechanismus wird dabei zu einer der zentralen Komponenten des Mediatorsystems.

Nicht-monotones Schließen

In Abschnitt 2.2 wurde die SLG-Resolution beschrieben. Sie organisiert die Speicherung der Ergebnisse von Unterzielauswertungen, um die Auswertung von logischen Programmen mit Negation unter der *well-founded*-Semantik

zu ermöglichen. Die SLG-Resolution erfordert, dass zumindest für diejenigen Unterziele Tabellen existieren, die eine Zusammenhangskomponente im Abhängigkeitsgraphen bilden, d.h. die entweder positiv oder negativ voneinander abhängen. Die Tabellen existieren wenigstens solange, bis alle an einer Zusammenhangskomponente beteiligten Unterziele vollständig ausgewertet sind. Der Tabulierungsmechanismus wird dazu verwendet, die Berechnung rekursiver Unterziele, die negativ voneinander abhängen, zum Teil aufzuschieben, bis endgültige Aussagen über eine Untermenge von Unterzielen gemacht werden können.

Verarbeitung von Annotationen und Reduktionsregel

Die SLANG-Resolution, die im Abschnitt 2.2.2 beschrieben wurde, verlangt die Speicherung derjenigen Antwortsubstitutionen, die eine partielle Antwort im Sinne des Annotationsverbands darstellen. Es bietet sich an, durch sofortige Anwendung der Reduktionsregel die Anzahl der annotierten Antwortsubstitutionen minimal zu halten, so dass immer nur diejenige Substitution mit der maximalen Annotation existiert. Eine Tabelle ist zumindest für diejenige Unterziele notwendig, deren Randbedingungen Annotationen fordern, die größer als das kleinste Element des Verbands sind. Dies tritt dann auf, wenn ein Rumpfliteral mit einer Annotationskonstante versehen ist. Auf die Verarbeitung von annotierten Antwortsubstitutionen werden wir genauer in Kapitel 5 eingehen.

Vermeidung redundanter Berechnungen

Unterzieltabellen erfüllen ganz natürlich die Aufgabe, redundante Berechnungen zu vermeiden, indem Berechnungsergebnisse gespeichert werden. Bei rekursiven Programmen wirkt sich das so aus, dass Endlosschleifen effektiv vermieden werden können und so die Terminierung gewährleistet ist. Nichtrekursive Programme profitieren dann von der Tabulierung, wenn der Aufwand ein bereits berechnetes Ergebnis in einer Unterzieltabelle zu finden, den Aufwand der Neuberechnung nicht übersteigt. Voraussetzung für diese Art der Verwendung von Unterzieltabellen ist deren Bereitstellung als globaler Speicher für die Inferenzprozedur, auf den jederzeit zugegriffen werden kann. Während der Programmauswertung wird dann bei jedem Unterzielaufruf geprüft, ob der selbe Aufruf bereits aufgetreten ist. In diesem Falle werden die bereits berechneten Antworten verwendet. Jedoch ist zu bedenken, dass, abhängig von der Problemstellung, der Speicherbedarf für Zwischenergebnisse extrem hoch sein kann. Daher ist eine Speicherverwaltung für Unterzieltabellen sinnvoll, die die Wechselwirkung zwischen Zeitgewinn und Speicherbelegung steuert.

Die Optimierung durch Tabulierung kann durch *subsumptive Tabulierung* weiter verbessert werden. Hierbei werden Antworten von Unterzielen wieder-

verwendet, die allgemeiner sind als das zu berechnende Unterziel. Allerdings muss die Antwortmenge dann auf diejenige Untermenge eingeschränkt werden, die das aktuelle Unterziel erfüllt. Dazu ist die Verwendung einer geeigneten Datenstruktur notwendig, die diese Operation effizient ausführen kann. Dies wird im Detail in Kapitel 5 diskutiert.

Lokale Zwischenspeicherung externer Daten

Die Vermeidung wiederholter Berechnungen hat im Umfeld eines Mediatorsystems, das auf entfernte Daten zugreift, besondere Bedeutung. Die Kosten von Netzwerkverbindungen und -übertragungen sind im Verhältnis zur den Kosten der lokalen Verarbeitung dominierend. Desweiteren können Zugriffe auf entfernte Informationen frequenz-, volumen- oder zeitabhängig tatsächliche Kosten verursachen. Die zunehmende Verbillingung lokaler sekundärer Speichermedien unterstreicht die Bedeutung von Mechanismen, die externe Daten lokal speichern. Im Rahmen der Inferenzprozedur mit Tabulierung kann dies transparent und anwendungsgetrieben geschehen. Die Verwendung des Tabulierungsmechanismus erfordert in diesem Zusammenhang einen Mechanismus zur Verwaltung von solchen Unterzieltabellen, die auch sitzungsübergreifend persistent gemacht werden können.

Bearbeitung datenintensiver Probleme

Abgesehen vom Zugriff auf entfernt gespeicherte Informationen kann es für bestimmte Problemstellungen notwendig sein, große Datenmengen zu verarbeiten. In typischen Logiksystemen ist die Programmauswertung vollständig hauptspeicherorientiert, d.h. dass alle Datenstrukturen und Zwischenergebnisse auch im primären Speicher abgelegt werden. Offensichtlich setzt daher der verfügbare Hauptspeicher eine obere Schranke für die Größe der verarbeitbaren Probleme. Zwar existieren in allen gängigen Betriebssystemen Mechanismen, die den Inhalt des primären Speichers transparent auf Sekundärspeichermedien auslagern, doch kann dieses Verhalten nicht gezielt beeinflusst werden und ist deshalb kein probates Mittel um Skalierbarkeit zu erreichen. Wesentlich effektiver ist es, die kritischen Datenstrukturen zu identifizieren und gezielt für die Auslagerung auf Sekundärspeichermedien zu konzipieren. Im Falle des vorliegenden Typs von System sind das die Unterzieltabellen. Hierzu ist es notwendig, die Anforderungen an die Datenstruktur zu identifizieren, die einerseits durch die Eigenschaften der zu speichernden Objekte, andererseits durch die Anforderungen der Inferenzprozedur gegeben sind.

Parallelisierung und asynchroner Quellenaufruf

Die Menge der Unterzieltabellen kann als für die Auswertungsprozedur globaler Speicher betrachtet werden, in den Berechnungsergebnisse abgelegt

werden. Ein Teil des Zustands der Programmauswertung ist durch diejenigen Unterzieltabellen gekennzeichnet, deren zugehöriges Unterziel noch nicht vollständig ausgewertet wurde. Der andere Teil des Zustandes steht in engem Zusammenhang mit den zugehörigen Unterzielen. Klauseln, die durch Resolution entstanden sind und deren Rumpf nicht leer sind, zählen zu diesem Teil des Zustandes. Diese Klauseln werden durch neue Einträge in die Unterzieltabellen weiter resolviert oder können behandelt werden, wenn das Unterziel vollständig ausgewertet ist. Die SLG-Resolution ist als Menge von Transformationen definiert, die auf dem Zustand der Auswertung operieren. Dabei ist die Reihenfolge der Transformationen nicht vorgegeben. Transformationen werden dann ausgeführt, wenn die erforderlichen Vorbedingungen erfüllt sind. Diese Sichtweise der Resolution eignet sich besonders für die parallele Verarbeitung. Dabei können einerseits unterschiedliche Transformationen parallel ablaufen, aber auch die parallele Ausführung der gleichen Transformation auf verschiedene Teile des Zustandes ist denkbar. Für den Tabulierungsmechanismus bedeutet dies die Bereitstellung der notwendigen Synchronisation für den nebenläufigen Zugriff.

Durch die parallelen Ausführung der Transformationen, die die SLG-Resolution definiert, erreicht man *OR*-Parallelität, d.h die Zweige eines Suchbaums, die durch Resolution eines Atoms mit Programmklauseln entstehen, werden unabhängig voneinander verarbeitet. Durch die Modifikation der Transformation *NEW ACTIVE* kann zudem *AND*-Parallelität erreicht werden, d. h. die Rumpfliterale einer Klausel werden parallel bearbeitet. Für die vollständige *AND*-Parallelität müssen eventuelle Abhängigkeiten zwischen Literalen im Klauselrumpf aufgelöst werden. Ansonsten können Rumpfliterale, die nicht durch Variablenbindungen voneinander abhängen, getrennt und damit nebenläufig verarbeitet werden.

Auch wenn nur ein Prozessor zur Ausführung eines Mediatorprogramms zur Verfügung steht, kann die Parallelisierung der Auswertungsprozedur Vorteile bringen. Durch die quasi-parallele Ausführung der Transformation *NEW ACTIVE* erreicht man implizit die Asynchronität bei Aufrufen von externen Informationsquellen, falls solche durch die Auswertung von Randbedingungen im Klauselrumpf ausgelöst werden. Dadurch kann die implizite Parallelität, die dadurch gegeben ist, dass ein Mediatorsystems die Möglichkeiten externer Prozessoren einbezieht, ausgenutzt werden.

Optimierung

Subsumptive Tabulierung liefert Freiheitsgrade bei der Wahl des auszuwertenden Unterziels. Wenn im Laufe der Programmauswertung die Erstellung eines neuen Unterziels angestoßen wird, muss sichergestellt sein, dass das geforderte Unterziel gemäß der gewählten Semantik berechnet wird. Bei der subsumptiven Tabulierung können zur Beantwortung eines Unterzielaufufes auch die Unterzieltabellen verwendet werden, deren Unterziel allgemei-

ner ist als das geforderte. Die Menge der Antwortsubstitutionen wird dann auf die Instantiierung des geforderten Unterziels eingeschränkt. Ohne die Korrektheit der Auswertungsprozedur zu verletzen, ist es daher möglich, einen Unterzielaufruf abzuschwächen, indem Konstanten durch Variablen ersetzt werden. In klassischen logischen Systemen ist dies unerwünscht, da dadurch der Suchraum vergrößert wird. Der Zusammenhang zwischen Größe des Suchraums und des zur Bearbeitung notwendigen Aufwandes ist aber in einem Mediatorsystem nicht zwingend gegeben. Falls eine Unterzielauswertung den Zugriff auf externe Informationsquellen bedingt, wird der Aufwand möglicherweise durch den externen Zugriff dominiert. Der Aufwand für einen externen Zugriff korreliert zumeist nicht linear mit der Größe der Antwortmenge, bzw. des Suchraums sondern enthält konstante Anteile, die bei jedem Zugriff entstehen. Daher ist der Aufwand für den einmaligen Zugriff mit einer allgemeineren Anfrage oftmals weniger aufwendig als eine größere Menge spezieller Anfragen. Dieses Verhalten kann jedoch nicht verallgemeinert werden, sondern hängt von den spezifischen Merkmalen der Informationsquellen ab. Um die Eigenschaften verschiedener Quellen angemessen zu berücksichtigen eignen sich kostenbasierte Optimierungsverfahren. Allerdings wird in einer Mediatorumgebung dies dadurch erschwert, dass im Allgemeinen keine Informationen über die spezifischen Kosten bestimmter Anfragen oder Anfragemuster zur Verfügung stehen. Hier kann es notwendig sein Messungen durchzuführen und auf gespeicherte Erfahrungswerte zurückzugreifen.

4.4 Eine Architektur für logikbasierte Mediatorsysteme

In diesem Abschnitt beschreiben wir die Architektur eines auf SLANG-Resolution beruhenden Mediatorsystems. Dabei erweitern wir die Funktion der Tabellen in zwei Punkten:

1. Die Lebensdauer von Tabellen wird von den Anforderungen der Resolutionsprozedur entkoppelt. Dadurch können Tabellen über einzelne Anfragen hinweg zur Speicherung von Zwischenergebnissen genutzt werden. Die Lebensdauer wird statt dessen durch einen Verdrängungsmechanismus geregelt.

2. Tabellen können Sekundärspeicher nutzen. Dies ist einerseits notwendig zur Bewältigung großer Datenmengen, andererseits um Tabelleninhalte persistent zu speichern.

Wir führen zunächst eine abstrakte Sichtweise der Komponenten ein und beschreiben das Zusammenspiel der Komponenten durch deren Schnittstellen. Teile unserer Architektur, nämlich die Resolutionskomponente und Teile der Tabulierungskomponente haben Ähnlichkeit mit dem in [RRR96, Rao97]

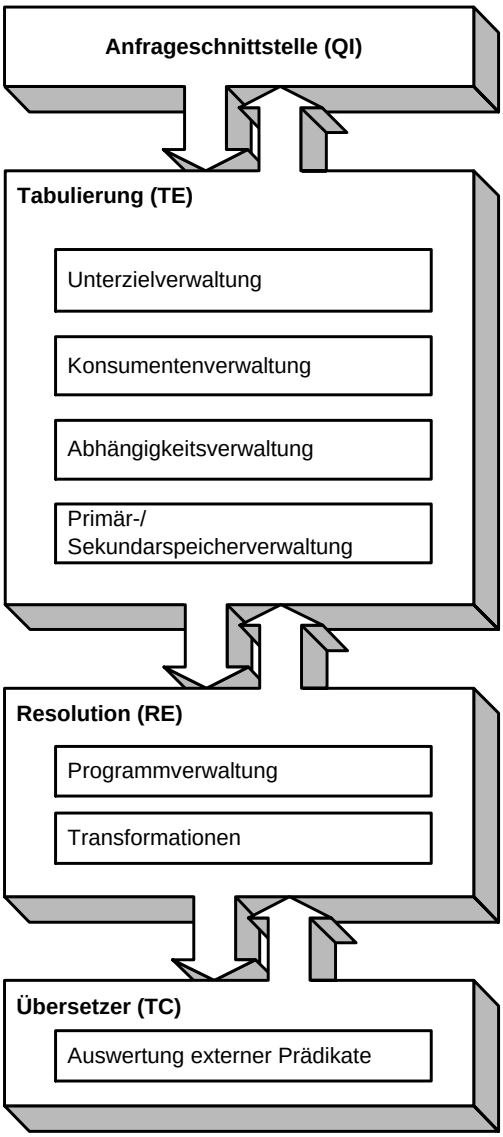

Abbildung 4.1: Architektur eines Mediators mit Tabulierung

beschriebenen Modell der tabulierten Resolution. Neben den spezifischen Erweiterungen unseres Ansatzes bezüglich der Tabulierung, berücksichtigen wir zudem die Parallelisierung der Funktionalität der Teilkomponenten und die Anbindung externer Informationsquellen.

Grundlage der Architektur ist die strikte Trennung der Funktionen der Resolution von den Funktionen der Tabulierung. In diesem Zusammenhang betrachten wir Unterziele als *Produzenten* und eine zu resolvierenden Klausel mit gewähltem Literal als *Konsument* von Antwortsubstitutionen. Wird in einer Klausel ein Literal gewählt, gilt es für die nächsten Resolutionsschritte einen entsprechenden Produzenten zu finden. Dieser Produzent kann entweder schon aufgrund vorhergehender Berechnungen existieren oder er muss neu erzeugt werden. Dies impliziert in aller Regel, dass Resolution mit Programmklauseln angestoßen wird. Im Folgenden legen wir die Verwendung von Substitutionsfaktorisierung für die Speicherung von Antworten für Unterziele zugrunde. Dabei werden als Objekte der Tabellenspeicherung statt Instanzen des Unterzielliterals Substitutionen für die Variablen des Unterzielliterals herangezogen. Die konstanten Anteile der Antworten werden in diesem Sinne heraus faktorisiert. Dies dient vorrangig zur Erhöhung der Speichereffizienz, hat aber sonst keine Auswirkungen auf die Tabulierung, da durch Anwendung der Substitutionen jederzeit die entsprechenden Unterzielantworten erzeugt werden können [DRR$^+$95].

Die Architektur besteht auf der obersten Ebene aus vier Teilsystemen (s. Abb. 4.1). Die Anfrageschnittstelle $\mathcal{QI}$, die Tabulierungskomponente $\mathcal{TE}$, die Resolutionskomponente $\mathcal{RE}$ und die Wrapper-Komponenten, die unter $\mathcal{TC}$ zusammengefasst sind.

Die Komponenten haben im Einzelnen folgende Aufgaben:

Anfrageschnittstelle Die Anfrageschnittstelle ist der Punkt, über den Anwendungen auf das Mediatorsystem zugreifen. Über diese Schnittstelle werden Anfragen entgegen genommen und können Ergebnisse übernommen werden. Da Prädikate in der Logik eine reine Mengen-Semantik haben, können andere Anforderungen, wie z.B. Sortierung nach bestimmten Kriterien von der Resolutions- bzw. Tabulierungskomponente nicht erfüllt werden. Die Anfrageschnittstelle kann diese Lücke schließen und Anfrageergebnisse nachbearbeiten.

Tabulierungskomponente Die Tabulierungskomponente übernimmt im weitesten Sinne die Speicherung von Antwortsubstitutionen. Sie verwaltet sowohl die Gesamtmenge der Unterzieltabellen als auch deren individuelle Organisation im Hauptspeicher und auf Sekundärspeichermedien. Dazu müssen die Abhängigkeiten zwischen Unterzielen, die sich aus der Resolution ergeben, berücksichtigt werden. Zudem werden in dieser Komponente Abhängigkeiten zwischen nicht vollständig ausgewerteten Tabellen und den Konsumenten der Antwortsubstitu-

tionen verwaltet. Insgesamt repräsentiert die Tabulierungskomponente den Datenhaltungsaspekt des Systems.

Resolutionskomponente Die Resolutionskomponente enthält die für die Resolution benötigen Funktionen. Sie verwaltet außerdem die Regelbasis. Die Resolutionskomponente führt die Auswahl von Literalen, Programm- sowie Antwortresolution durch.

Wrapper-Komponenten In den Wrapper-Komponenten ist die Funktionalität konzentriert, die den Aufruf externer Informationsquellen regelt. Teile dieser Funktionen können abstrahiert sein und für ganze Klassen von Informationsquellen wiederverwendet werden. In dieser Schicht werden insbesondere Verbindungen zu entfernten Informationsquellen verwaltet. Physisch kann diese Schicht auf mehrere Prozessoren verteilt sein. Das ist dann der Fall, wenn Teile einzelner Wrapper aus Effizienzgesichtspunkten direkt bei der Informationsquelle lokalisiert sind.

Der Prozess einer Anfragebearbeitung wird durch die Anfrageschnittstelle initiiert. Sie veranlasst die Tabulierungskomponente ein entsprechendes Unterziel zu erzeugen und, wenn nötig, die Resolutionskomponente aufzurufen.

Die Tabulierungskomponente und die Resolutionskomponente sind gegenseitig voneinander abhängig. Entsprechend der Auswertungsstrategie rufen sie sich gegenseitig auf. Die Entkoppelung der beiden Module, die durch die Verwendung von Warteschlangen erreicht wird, ermöglicht den Grad der Parallelität unabhängig von der Anzahl der Funktionsaufrufe zu variieren. In einer nicht nebenläufigen Umgebung ist die Auswertungsstrategie durch die Reihenfolge bestimmt, in der die Module bzw. deren Funktionen aufgerufen und die Warteschlangen abgearbeitet werden.

Die Wrapper-Komponenten können als Subsysteme der Resolutionskomponente verstanden werden. Die Wrapper werden ausschließlich durch die Resolutionskomponente aufgerufen.

Im den folgenden Abschnitten werden die Komponenten der Mediatorarchitektur im Einzelnen beschrieben. Alle hier beschriebenen Funktionen sind in einer Pseudokodierung im Anhang zu finden.

4.4.1 Anfrageschnittstelle

Die Anfrageschnittstelle ist eine Schicht von Funktionen, die zwischen der Anwendung und der Tabulierungskomponente vermittelt. Sie dient letztlich als Homogenisierungsschicht zwischen der Umgebung der Anwendung und dem Mediatorsystem. Im einfachsten Fall ist diese Schicht sehr dünn und ist eine Untermenge der Schnittstelle der Tabulierungskomponente. Für viele

Anwendungen werden allerdings einige zusätzliche Funktionen benötigt, die in dieser Schicht realisiert werden. Dazu zählen z.B.

- Datenkonvertierung zwischen dem Datenmodell der Anwendungsumgebung und dem Datenmodell des Mediators,
- Sortierung von Anfrageergebnissen,
- Gruppierung von Anfrageergebnissen,
- Zugriffsmechanismen für Anfrageergebnisse.
- Benachrichtigungsmechanismus zur Verkettung von Verarbeitungsprozessen.

Die Funktion, die die Anfrageschnittstelle zumindest bietet, ist:

Query(q) **:** Diese Funktion nimmt eine Anfrage q entgegen und liefert das Anfrageergebnis. Das Ergebnis ist entweder eine Menge von Substitutionen über die Variablen in q oder aber, für variablenfreie Anfragen, *yes*, *no* oder *unknown*, je nachdem ob das entsprechende Unterziel im Sinne der *well-founded*-Semantik bezüglich des gegebenen Programms beweisbar, widerlegbar oder unbekannt ist. Falls q keine atomare Anfrage ist, wird eine Anfrageklausel erzeugt, die als temporäre Klausel zum Programm hinzugefügt wird. Dies ist notwendig, da die Tabulierungskomponente nicht in der Lage ist, Klauseln, die keine Einheitsklauseln sind, zu beantworten.

4.4.2 Tabulierung

Die Tabulierungskomponente besteht aus den Subkomponenten *Unterzielverwaltung*, *Konsumentenverwaltung*, *Abhängigkeitsverwaltung* und *Speicherverwaltung*.

Unterzielverwaltung Diese Komponente ist für die Verwaltung von Unterzielen und deren Daten zuständig. Sie regelt den Lebenszyklus von Unterzieltabellen. Bei einer Anfrage, für die kein Unterziel existiert, wird ein neues Unterziel erzeugt. Ansonsten wird die Anfrage mit den Antworten eines existierenden Unterziels beantwortet. Unterzieltabellen können bestehen bleiben, auch wenn die Anfrage, die diese Tabellen erzeugt hat, abgeschlossen ist. Sofern die Informationen, die zur Erzeugung der Tabelle verwendet wurden, noch Gültigkeit haben, können die Ergebnisse zur Beantwortung weiterer Anfragen genutzt werden. Die Lebensdauer von Unterzieltabellen hängt einerseits von den gegebenen Speicherplatzbeschränkungen ab und andererseits davon, wie groß ihr Nutzen für die Beantwortung weiterer Anfragen ist. Im Allgemeinen kann nicht vorausgesagt werden, welche Anfragen zu erwarten sind.

Allerdings lassen sich aus den vergangenen Anfragen Schlüsse über die Nützlichkeit von Antworttabellen ziehen. Zu diesem Zweck wird für die existierenden Unterzieltabelle die Häufigkeit und Zeitpunkte des Zugriffs protokolliert. Zusammen mit den Kosten, die zur Erzeugung der Tabelle aufgebracht werden mussten, lassen sich verschiedene Verdrängungsstrategien realisieren. Bekannte Verdrängungsstrategien sind *LRU* (*least recently used*), *LFU* (*least frequently used*) und Kombinationen davon.

Konsumentenverwaltung Während der Berechnung eines Unterziels kann es bei rekursiven Programmen zum Unterzielaufruf des selben Literals oder eines spezielleren Literals kommen. Die subsumptive Tabulierung nutzt zur Beantwortung dieser Unterziele die bereits berechneten Ergebnisse des existierenden Unterziels, um einerseits die Terminierung in solchen Fällen zu gewährleisten und andererseits um mehrfache Berechnungen zu vermeiden. Da die Berechnung des Unterziels noch nicht abgeschlossen ist, müssen alle Konsumenten von Antworten vermerkt werden. Wenn weitere Ergebnisse für ein Unterziel entstehen, werden diejenigen Konsumenten bestimmt, die mit der Antwort unifizierbar sind und mittels Antwortresolution weiter verarbeitet. Sobald die Berechnung einer Tabelle abgeschlossen ist, kann die Speicherung der Konsumenten für dieses Unterziel aufgegeben werden.

Abhängigkeitsverwaltung Eine Schwierigkeit der SLG-Resolution besteht darin, festzustellen, wann die Berechnung eines Unterziels abgeschlossen ist. Dies ist um so mehr der Fall, wenn die Programmauswertung parallelisiert wird. Allgemein ist die Unterzielauswertung dann beendet, wenn alle Antworten, die durch Programmresolution entstehen können, berechnet worden sind. Da diese Antworten selbst durch Unterzielaufrufe erzeugt werden, ist es notwendig zu verfolgen, wie die Unterziele untereinander in Abhängigkeit stehen. Durch rekursive Programme entsteht eine Abhängigkeitsstruktur, die sich als Graph repräsentieren lässt. Bei der SLG-Resolution ist eine Menge von Unterzielen dann vollständig ausgewertet, wenn jedes der Unterziele vollständig ausgewertet ist. Da die vollständige Auswertung eines Unterziels abhängt von der Auswertung der abhängigen Unterziele, lassen sich zirkular abhängige Mengen von Unterzielen nicht ohne weiteres in den Zustand der vollständigen Auswertung überführen. In Kapitel 2 wurde ausgeführt, wie bei der SLG-Resolution zirkulare Abhängigkeiten behandelt werden. Dafür ist die Berechnung von Zusammenhangskomponenten im Abhängigkeitsgraphen notwendig. Ein geeigneter Algorithmus zur Bestimmung der Vollständigkeit von Unterzielen ist in [FHSW95] beschrieben.

Speicherverwaltung Für die Speicherung der Unterzieltabellen im Pri-

märspeicher ist die Speicherverwaltung trivial. Falls aber entweder der Primärspeicher nicht für alle Unterziele ausreicht oder aber Unterzielauswertung nicht-flüchtig gespeichert werden sollen, wird die Organisation der Unterzieltabellen auf Sekundärspeichermedien notwendig. Das Problem der Nutzung von Sekundärspeicher ist der im Vergleich zum primären Speicher um Größenordnungen größere Zeitaufwand für jeden Zugriff. Die Datenmenge, die pro Zugriff übertragen wird, hat hingegen wenig Einfluss auf den Aufwand. Daher ist es wichtig, die Daten einerseits seitenweise zu organisieren, und andererseits diese so anzuordnen, dass möglichst wenige Zugriffe erforderlich sind, um bestimmte Informationen in der Datenstruktur zu finden. Geeignete Datenstrukturen für die Speicherung von Antwortsubstitutionen in diesem Zusammenhang werden im Detail in Kapitel 5 untersucht. Die Speicherverwaltung hält im Primärspeicher Platz für eine begrenzte Menge von Datenseiten vor, der je nach Bedarf aus dem Sekundärspeicher befüllt wird. Eine Verdrängungsstrategie sorgt dafür, dass häufig benutzte Datenseiten länger im Primärspeicher verbleiben.

Der Zustand der Tabulierungskomponente $\mathcal{TE}$ ist gegeben durch das Tupel $< Q_s, Q_c, S >$. Q_s ist eine Warteschlange mit Anfragen, Q_c eine Warteschlange von Unterziele und S eine Menge von tabulierten Unterzielen. Jedes Unterziel s dieser Menge wird definiert durch das Tupel

$$\langle l, A, C, Q \rangle$$

Die Komponente l bezeichnet ein Unterzielliteral, A repräsentiert eine Menge von Antworten, C repräsentiert eine Menge von wartenden Klauseln mit selektiertem Literal und einem assoziiertem Unterziel für das Kopfliteral, Q steht für eine Schlange mit wartenden Antworten, die in die Unterzieltabelle eingefügt werden sollen. Die Antwortmenge A enthält alle Fakten, die von l subsumiert werden und bis zum Zeitpunkt des gegenwärtigen Systemzustandes gefunden wurden. Wenn bei der Tabulierung Substitutionsfaktorisierung zur Anwendung kommt, ist A eine Menge von Antwortsubstitutionen über die Menge der Variablen in l. Substitutionsfaktorisierung kann in diesem Gerüst transparent eingesetzt werden. Wir können sie hier ohne Beeinträchtigung der Allgemeinheit vernachlässigen.

$\mathcal{TE}$ unterstützt folgenden Operationen:

NewAnswer(s, A) **:** Diese Operation fügt eine Antwort A in die angegebene Unterzieltabelle s ein, sofern sie nicht subsumiert wurde. Sie bestimmt alle interessierten Konsumenten in C und ruft $\mathcal{RE}$ :: *ResolveAnswer* mit jedem Konsumenten auf.

FindOrCreateSubgoal(l, Cl) **:** Diese Operation kombiniert eine Reihe von Schritten. Zunächst wird bestimmt, ob ein geeigneter Unterzielaufruf

bereits in S existiert. Ist dies der Fall und ist l positiv, werden alle subsumierten Antworten mittels $\mathcal{RE} :: ResolveAnswer$ zur konsumierenden Klausel Cl geschickt. Ist l negativ und die Unterzieltabelle enthält bereits Antworten, scheitert die Klausel. Ist die Unterzielauswertung vollständig durchgeführt und enthält die Tabelle keine Antworten, dann kann das negative Literal l mittels $\mathcal{RE} :: ResolveNegative$ resolviert werden. Enthält die Tabelle nur Antworten mit verzögerten Literalen wird SLG-Faktorisierung durchgeführt. Wenn die Unterzielauswertung noch nicht beendet ist, wird Cl, sowohl bei positivem als auch negativem l außerdem als Konsument registriert. Wenn der Aufruf oder ein allgemeinerer jedoch noch nicht vorhanden ist, wird eine neues Unterziel erzeugt und $\mathcal{RE} :: NewSubgoal$ aufgerufen.

CompleteSubgoal(s) **:** Diese Operation schließt die Auswertung eines Unterziels ab. Alle Konsumenten, die an ein negatives Literal gekoppelt sind und für die bislang keine Antwort gefunden wurde, können als erfüllt betrachtet und resolviert werden. Dies geschieht mittels $\mathcal{RE} :: ResolveNegative$. Bei Konsumenten mit einem positiven Literal, für das keine Antworten existieren, gilt die zugehörige Klausel als gescheitert.

Lebensdauer von Tabellen

Als Besonderheit der hier vorgeschlagenen Architektur ist die Lebensdauer von Tabellen nicht unmittelbar abhängig von den Erfordernissen der Resolution. Tabellen können sowohl vor dem Zeitpunkt berechnet werden, an dem sie unmittelbar benötigt werden, als auch über die durch die Resolution gegebene Zeitdauer hinaus bestehen bleiben. Allerdings muss ein Unterziel mindestens für die Dauer der Berechnung bis zur vollständigen Auswertung im Sinne der SLG-Resolution bestehen bleiben. Ein zusätzlicher Mechanismus, der auf einer Verdrängungsstrategie beruht, sorgt für die Freigabe von nicht mehr benötigten Unterzieltabellen.

4.4.3 Resolution

Die Resolutionskomponente besteht aus den zwei Subkomponenten *Programmverwaltung* und *Transformationen*.

Programmverwaltung Die Programmverwaltung gestattet den Zugriff auf die Programmklauseln des Mediatorprogramms. Für Unterzielauswertungen werden alle Klauseln benötigt, deren Kopf sich mit dem Unterzielliteral unifizieren lässt. Dazu sind die Klauseln so organisiert, dass der Zugriff über das Kopfliteral möglichst effizient realisiert ist. Für diesen Zweck eignen sich im Prinzip die gleichen Datenstrukturen, wie zur Speicherung von Antworten in Antworttabellen. Wenn nötig, kann

zur Speicherung des Programms ebenfalls Sekundärspeicher genutzt werden.

Transformationen Diese Komponente realisiert die Operationen, die denjenigen Transformationen der SLG-Resolution entsprechen, die Resolutionsschritte durchführen. Dazu gehören Antwortresolution sowie Programmresolution.

Der Zustand der Resolutionskomponente $\mathcal{RE}$ ist durch das Tupel

$$\langle P, Q_c, Q_s, Q_r \rangle$$

gekennzeichnet.

Die Komponente P ist ein logisches Programm, Q_c, Q_s und Q_r repräsentieren Warteschlangen, die Objekte zur weiteren Verarbeitung enthalten.

Die Komponente Q_c repräsentiert eine Schlange mit wartenden Paaren $\langle s, Cl \rangle$. Ein Paar besteht aus einer Klausel Cl und einer assoziierten Unterzieltabelle s für den Klauselkopf. Die Komponente Q_s ist eine Schlange mit neuen Unterzielen, die für die Programmresolution bereit stehen. Schliesslich steht Q_r für eine Schlange mit Tripeln $\langle s, Cl, \theta \rangle$. Ein solches Tripel besteht aus einer Klausel Cl mit einem selektiertem Literal, einer assoziierten Unterzieltabelle s für den Klauselkopf und einer Substitution θ für die anstehende Antwortresolution.

$\mathcal{RE}$ bietet folgende Operationen:

$NewClause(s, Cl)$ **:** Diese Operation erwartet eine Klausel Cl und ihre assoziierte Unterzieltabelle s als Eingabe. Ist der Klauselrumpf leer, wird $\mathcal{TE} :: NewAnswer$ aufgerufen, im anderen Fall wird ein Literal gewählt und $\mathcal{TE} :: FindOrCreateSubgoal$ wird aufgerufen.

$NewSubgoal(s)$ **:** Diese Operation bestimmt die passenden Programmklauseln für die Resolution mit einem Unterzielliteral s. Für jede gefundene Programmklausel wird die entsprechende Substitution durchgeführt und $\mathcal{RE} :: NewClause$ aufgerufen. Anschließend wird das Unterziel mit $\mathcal{TE} :: CompleteSubgoal$ abgeschlossen.

$ResolveAnswer(s, Cl, Cl_s, \theta)$ **:** Diese Operation führt Antwortresolution der Klausel Cl mit dem selektierten Literal Cl_s aus unter Verwendung der Substitution θ, sofern das selektierte Literal positiv ist. Dann wird $\mathcal{RE} :: NewClause$ mit s und der resolvierten Klausel aufgerufen. Ist das selektierte Literal jedoch negativ, scheitert die Klausel, was bedeutet, dass die Klausel Cl nicht weiter verarbeitet wird. Der dazugehörige Konsumenteneintrag kann also aus der Unterzieltabelle entfernt werden.

$ResolveNegative(s, Cl, Cl_s)$ **:** $ResolveNegative$ löscht das negative Literal Cl_s aus der Klausel Cl und ruft damit $\mathcal{RE} :: NewClause$ auf.

4.4.4 Wrapper-Komponenten

Die Wrapper-Komponenten dienen zur Einbindung von Informationsquellen in die Mediatorumgebung. Dort sind die von den Informationsquellen bereit gestellten Informationen und Funktionen über Randbedingungen in Programmklauseln eingebettet. Die Lösung solcher Randbedingungen liefert, entsprechend den Unterzielaufrufen, Variablenbindungen, die wie bei der Antwortresolution verarbeitet werden. Die Lösung von Randbedingungen ist in unserem System nur bei denjenigen Teilen möglich, für die das geforderte Bindungsmuster erfüllt ist. Für den Zeitpunkt der Auswertung von Randbedingungen gibt es unterschiedliche Möglichkeiten. Eine Möglichkeit ist die Auswertung, sobald das geforderte Bindungsmuster erfüllt ist, eine andere die Auswertung, nachdem alle Rumpfliterale resolviert wurden.

Wrapper-Komponenten bieten die folgende Operation:

Evaluate(C) : Die Funktion *Evaluate* wertet eine Konjunktion von Randbedingungsliteralen aus und liefert entweder Variablenbindungen oder *true* bzw. *false* zurück, je nachdem ob die Bedingung erfüllt werden kann oder nicht.

4.4.5 Prozessmodell

Die Architektur gestattet die Realisierung einer nebenläufigen Ausführung mit der Verwendung von gemeinsam genutztem Speicher. Die Anzahl der gleichzeitig ablaufenden Prozesse oder Fäden (engl.: *threads*) ist variabel und kann entsprechend den tatsächlich vorhandenen Prozessoren gewählt werden. Um asynchrone Zugriffe auf Informationsquellen zu erreichen, wird die gleiche Zahl von Prozessen oder Fäden gewählt, wie die maximalen Anzahl der gleichzeitigen Verbindungen. Im besten Fall kann dann zur gleichen Zeit in jedem Faden, unabhängig von den anderen, ein Zugriff auf eine entfernte Informationsquelle stattfinden. Eine andere Strategie ist, die Anzahl der Prozesse bzw. Fäden in Abhängigkeit des Füllgrades der Warteschlangen zu steuern. So können Prozessoren lastabhängig in die Verarbeitung einbezogen werden.

Die größte Effizienz kann erzielt werden, wenn alle Prozesse bzw. Fäden den Speicher gemeinsam nutzen. Dann ist es ausreichend, Referenzen auf Daten zu übergeben, an Stelle der Daten selbst.

In Abbildung 4.4.5 bezeichnet Sg die Menge der Unterziele, Sg_a die Menge der aktiven Unterziele, das heißt derjenigen Unterziele, die im Sinne der SLG-Resolution nicht vollständig sind. Q_{sg} sind Warteschlangen, die jeweils mit einem aktiven Unterziel assoziiert sind. C_{sg} ist die Menge der Datenstrukturen zur Verwaltung der Konsumenten von Unterzielergebnissen. R_1, R_2 und R_3 bzw. T_1, T_2 und T_3 stellen Fäden oder Prozesse dar, die jeweils eine der Operationen einer Komponente ausführen. Pfeile illustrieren, auf welche anderen Elemente dabei zugegriffen wird. R_1 führt die

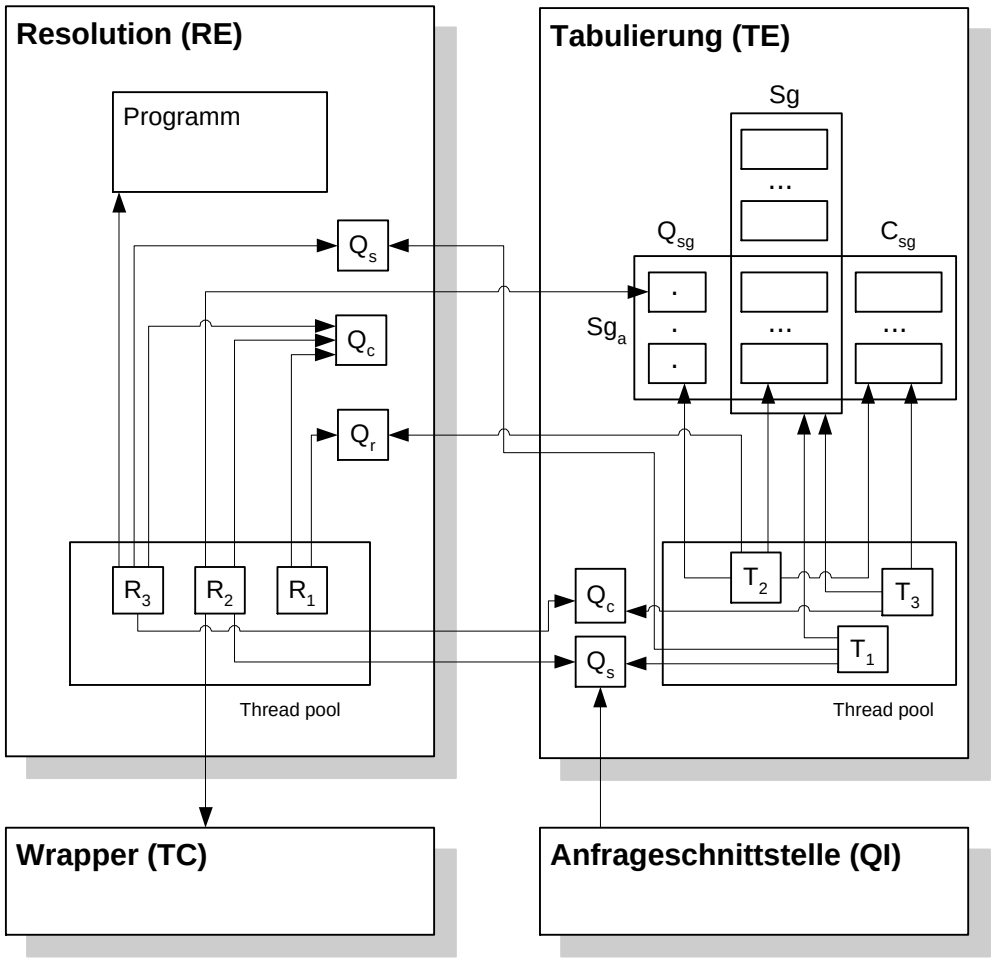

Abbildung 4.2: Prozessmodell

Operation *ResolveAnswer*, R_2 die Operation *NewClause* und R_3 die Operation *NewSubgoal* aus. T_1 führt die Operation *FindOrCreateSubgoal*, T_2 die Operation *NewAnswer* und T_3 die Operation *CompleteSubgoal* aus. Q_l, Q_c und Q_s der Resolutionskomponente repräsentieren die in Abschnitt 4.4.3 beschriebenen Warteschlangen. Q_c, Q_s und Q_{sg} entsprechen den Warteschlangen, die in Abschnitt 4.4.2 erläutert wurden.

4.4.6 SLG-Resolution

Der Aufruf und die Ausführung der Operationen der Komponenten sind durch Warteschlangen entkoppelt. Die Terminierungseigenschaft der SLG-Resolution kann auf die vorgeschlagene Architektur übertragen werden, wenn

sicherstellt wird, dass die Operationen mit Elementen der Warteschlangen solange aufgerufen werden, solange noch Einträge in einer der Schlangen vorhanden sind.

4.4.7 Einsatz in einer Mediatorumgebung

Die Architektur beschreibt den Aufbau eines Moduls zur Verarbeitung von Mediatorprogrammen. In einer Mediatorumgebung wird ein solches Modul vorrangig als zentrale Komponente zur Informationsintegration eingesetzt werden. Es ist aber auch denkbar, ein solches Modul im Kleinen in der *Wrapper*-Schicht einzusetzen, um Informationsquellen mit zusätzlicher Funktionalität zu versehen. Das ist insbesondere bei kooperativen Informationsquellen ein geeigneter Ansatz zur Effizienzsteigerung, wenn das Modul unmittelbar bei der Quelle angesiedelt wird. Abbildung 4.3 zeigt verschiedene Arten der Verteilung in einer Mediatorumgebung.

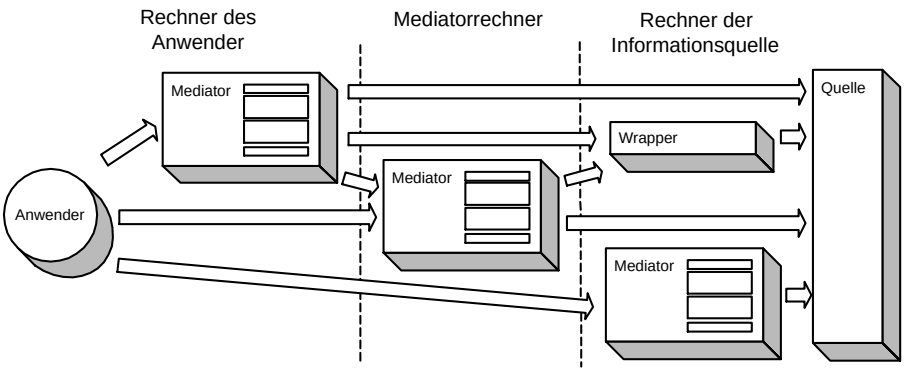

Abbildung 4.3: Verteilungsszenarien

In diesem Kapitel wurde eine Architektur für einen logikbasierten Mediator vorgeschlagen, dessen integraler Bestandteil neben einer Inferenzkomponente eine Komponente zur Verwaltung von Unterzielen mit gespeicherten Auswertungsergebnissen ist. Die Tabulierungskomponente ist verantwortlich für die Auslagerung der Daten auf den Sekundärspeicher, wenn dies erforderlich ist. Das Prozessmodell der Architektur beruht auf den Transformationen der SLANG-Resolution, die im Anhang A zusätzlich als Pseudokode ausgeführt sind.

Lasst die Erinnerung uns nicht belasten mit dem Verdrusse, der vorüber ist.

William Shakespeare

Kapitel 5

Eine Datenstruktur zur subsumptiven Tabulierung

Im vorigen Kapitel wurde diskutiert, welche Anforderungen an den Tabulierungsmechanismus als zentraler Bestandteil eines Mediatorsystems zu stellen sind und wie sich die Tabulierung in die Architektur eines Mediatorsystems einbetten lässt. In diesem Kapitel wird untersucht, wie Unterzieltabellen unter den Voraussetzungen des in dieser Arbeit diskutierten Ansatzes realisiert werden können. Zunächst wird das Problem der Speicherung von Unterzielen und existierende Lösungsansätze erläutert. Im zweiten Abschnitt wird die subsumptive Tabulierung bezüglich der annotierten Logik betrachtet. In den folgende Abschnitten wird ein geeignete Datenstruktur auf der Basis einer Multiattribut-Indizierung vorgeschlagen. Die Ergebnisse dieses Kapitels wurden zum Teil in [CK99a] veröffentlicht.

5.1 Datenstrukturen für die Tabulierung

Die Objekte, die in Tabellen abgelegt werden, sind Antwortsubstitutionen über der Menge der Variablen eines Unterziels. Eine solche Antwortsubstitution lässt sich als Vektor darstellen, dessen Dimension durch die Anzahl der Variablen des Unterziels bestimmt ist. Da wir uns, wie in Abschnitt 2.3 erläutert, auf funktionsfreie Programme beschränken, haben Antwortsubstitutionen die Form, dass Variablen entweder an Konstanten oder an Variablen gebunden werden. Die Speicherung von Antwortsubstitution lässt sich also als ein Problem der Speicherung mehrdimensionaler Daten auffassen. Die Datenstruktur, die im Rahmen eines Mediatorsystems dafür genutzt werden soll, muss also folgende Kriterien erfüllen:

- Unabhängigkeit von der Dimensionalität der Daten

- Speicherung partiell geordneter Wertemengen
- Speicherung unvollständiger Daten, d.h. im Sinne der Logik teilinstantiierte Antwortsubstitutionen
- Effizienter Zugriff auf (Unter-)Mengen, die durch eine partielle Substitution definiert werden.
- Effiziente Speicherung auf Sekundärspeicher durch seitenororientierte Speicherorganisation.
- Zur Erhöhung der Parallelisierung sind nebenläufige Zugriffe wünschenswert.

Für die spezielle Anwendung können auch Einschränkungen angegeben werden:

- Eine Einfügeoperation ist ausreichend. Antworttabellen werden nur komplett gelöscht.
- Strukturell: Eine Variable der Antwortsubstitution ist entweder instantiiert oder nicht, d.h. Unterräume erstrecken sich immer über die gesamte Dimension, in der sie variabel sind.
- Es ist nicht erforderlich, ähnliche Werte in Nachbarschaft zu speichern. Daher können Raumtransformationen den angewendet werden. Insbesondere können beispielsweise Abbildungen verwendet werden, die partiell geordnete Wertemengen auf eine totalgeordnete Menge von Werten abbilden, wenn dies aufgrund der gewählten Datenstruktur erforderlich ist.

Gebräuchliche Ansätze zur Verwaltung von Unterzielen basieren auf *variantenbasierter Tabulierung*. Dabei wird ein existierendes Unterziel als Produzent herangezogen, wenn das gewählte Literal eine Variante des Unterzielliterals darstellt. Zwei Literale sind dann Varianten, wenn sie nur durch die Umbenennung von Variablen identisch gemacht werden können. Variantenbasierte Tabulierung weist einige Eigenschaften auf, die ihren Einsatz bei der Verwaltung von Unterzieltabellen einfach und effizient machen.

- Die Suche nach einem geeigneten Unterziel (Produzent) verlangt keine Unifikation. Um festzustellen, ob zwei Literale Varianten sind, genügt bei normalisierter Variablenbenennung ein Test auf Gleichheit.
- Alle Antwortsubstitutionen, die in die Unterzieltabelle eingefügt werden, sind gleichzeitig Antworten, die mit allen Konsumenten resolviert werden können.

- Eine spezielle Ordnung oder Indizierung der Antwortsubstitutionen ist nicht notwendig, da niemals eine Untermenge der Antworten benötigt wird, sondern immer die gesamte Menge der Antwortsubstitutionen.

Verfahren zur variantenbasierten Tabulierung stützen sich meist auf stukturteilende Datenstrukturen (engl.: *structure sharing*) zur Repräsentation von Unterzieltabellen. Die gängigsten Datenstrukturen in diesem Bereich sind Präfixbäume (*Tries*). Ein Trie ist ein Baum, dessen Knoten Teile der zu speichernden Schlüssel der Länge nach enthalten. Die Präfixe eines Schlüssels entscheiden, wie der Baum von oben nach unten durchlaufen wird. Da Tries die Repräsentationen von gemeinsamen Atomen nur einmal ablegen, sind sie unter gewissen Umständen sehr speichereffizient. Sie zeigen auch sehr effizientes Verhalten bei Einfüge- und Suchoperationen.

Der wesentliche Nachteil von variantenbasierter Tabulierung ist die Tatsache, dass das Potenzial von Tabellen als Zwischenspeicher nicht voll ausgeschöpft wird. Die Verwendung von variantenbasierter Tabulierung neigt dazu, große Mengen von Unterzielen zu erzeugen und dabei viele redundante Berechnungen auszuführen. Das ist darauf zurückzuführen, dass die Subsumptionseigenschaft von Unterzielen nicht genutzt wird. Wenn ein Unterziel A ein Unterziel B subsumiert, so sind die Antwortsubstitutionen für B eine Untermenge der Substitutionen für A. Wurde das Unterziel A bereits berechnet, liegen alle Antworten für B bereits vor und müssen nicht erneut berechnet werden. Die Ausnutzung dieses Umstandes kann die Performanz von unterzielbasierten Resolutionsprozeduren erheblich verbessern. Dies wurde in [RRR96, Rao97] gezeigt, wo Tries für die Verwendung für subsumptionsbasierte Tabulierung erweitert wurden.

Eine Datenstruktur, die zur Tabulierung bei datenintensiven logikbasierten Anwendungen geeignet ist, sollte nach unserer Ansicht folgende Eigenschaften besitzen:

Unterstützung der Besonderheiten bei subsumptionsbasierter Tabulierung

Die Datenstruktur muss effiziente Einfüge- und Suchoperationen ermöglichen. Subsumptionsbasierte Tabulierung ist auf die Möglichkeit angewiesen, Teilmengen des Tabelleninhalts zu ermitteln. Die Teilmenge wird durch ein Suchliteral spezifiziert, welches spezieller ist als das Unterziel, aber dennoch Variablen enthalten kann. Als Ergebnis sind all jene Antworten erwünscht, die spezieller sind als das angegebene Suchliteral. Da nicht vorhergesehen werden kann, welche Bindungsinformation bei solchen Anfragen gegeben ist, müssen die Antwortsubstitutionen bezüglich aller Argumente indiziert werden. Nur dadurch kann gewährleistet werden, dass die Suchoperation weniger aufwendig ist als eine lineare Suche in der Menge der Antworten. Es wird ein Mechanismus benötigt, der es erlaubt, Literale als Konsumen-

ten für Antwortsubstitutionen einzutragen. Das ist dann notwendig, wenn die Berechnung eines Unterziels noch nicht abgeschlossen ist. In diesem Fall müssen die Antworten, die erst später im Resolutionsprozess entstehen, an die interessierten Konsumenten weitergereicht werden. Dies entspricht dann einem *bottom-up*-Verfahren.

Reduktion von subsumierten Antworten

Einerseits müssen neue Antworten, die bereits von in der Struktur vorhandenen Einträgen subsumiert werden, ignoriert werden. Sie können nicht mehr zur Lösung der Anfrage beitragen, da sich keine weitere Information liefern als bereits errechnete Antworten. Andererseits sollten Antworten, die von einer neu einzufügenden Substitution subsumiert werden, aus der Datenstruktur entfernt werden. Das spart einerseits Platz, andererseits werden so überflüssige Berechnungen im weiteren Verlauf des Resolutionsprozesses vermieden. Diese Konzepte werden später formal als *Vollständigkeit* und *Minimalität* einer Tabellenstruktur eingeführt und formuliert werden.

Unterstützung von lokalem Sekundärspeicher

Große Datenmengen, wie sie in modernen Informationssystemen häufig auftreten, sollten problemlos vom Mediator verarbeitet werden können. Der Hauptspeicher des Rechners sollte nicht das mögliche Datenvolumen begrenzen. Gefragt ist eine effiziente Verwaltung der Daten dergestalt, dass die Anzahl der Zugriffe auf den Sekundärspeicher minimiert wird.

Unterstützung von Persistenz

Um das Potential der Tabulierung voll auszuschöpfen, kann es sinnvoll sein, Tabellen über einen längeren Zeitraum lokal zu speichern. In einer konkreten Anwendung könnten Tabelle in regelmäßigen Zeitintervallen dann neu berechnet werden, wenn die produktive Umgebung dadurch nicht beeinträchtigt wird. Für diese Anforderung kann die Struktur der Speicherung auf dem Sekundärspeicher prinzipiell eine andere sein, als die der Speicherung im Hauptspeicher. Es bietet sich jedoch an, eine Datenstruktur zu wählen, die diese Eigenschaft mit der vorigen verbindet.

5.2 Subsumptive Tabulierung für annotierte Logik

Die Verwendung der annotierten Logik stellt weitere Anforderungen an den Tabulierungsmechanismus. Für jede Antwortsubstitution muss die zugehörige Annotation vermerkt werden. Dies ist unabhängig davon notwendig, ob ein Unterziel mit konstanter Annotationsrandbedingung angegeben wurde

oder nicht. Antwortsubstitutionen, deren Annotation unterhalb des geforderten Wertes liegt, können dennoch als partielle Antwort zu einer vollständigen Antwort beitragen. Durch Anwendung der Reduktionsregel der annotierten Logik werden aus entsprechenden partiellen Antworten solche erzeugt, die die gegebene Randbedingung des Unterziels erfüllt. Die Zusammenhänge wurden bereits im Abschnitt 2.2.2 erläutert. In diesem Kapitel untersuchen wir, welche Eigenschaften Tabellen für die subsumptive Tabulierung mit annotierter Logik bezüglich der gespeicherten Inhalte haben und stellen geeignete Algorithmen zur Verwaltung der Tabellen vor.

Definition 5.1 (Kompatibilität)
Sei L das Unterzielliteral einer Tabelle T. Die Substitutionen θ und φ sind kompatibel*, wenn gilt: $[L]\theta$ und $[L]\varphi$ sind unifizierbar durch den allgemeinsten Unifikator ψ. $\theta\psi$ (ebenso $\varphi\psi$) wird der* allgemeinste gemeinsame Nenner *(engl.:* most general common denominator*) genannt und durch $mgcd(\theta, \varphi)$ bezeichnet.*

Definition 5.2 (Substitutionsordnung)
Seien θ und φ Substitutionen. θ heißt spezieller *als φ wenn gilt: $\exists\psi : \varphi\psi = \theta$. Wir schreiben auch $\theta \leq \varphi$. Gilt $\psi \neq \emptyset$, dann ist θ* echt spezieller *als φ und man schreibt $\theta < \varphi$.*

Definition 5.3 (Vollständigkeit von Tabellen)
Eine Tabelle T bestehe aus einer Menge von Paaren (θ_i, μ_i), wobei θ_i eine Substitution über die Variablen des Unterzielliterals und μ_i eine Annotation des Verbandes ist, über den das Unterzielprädikat definiert ist. Wenn für alle $(\theta_i, \mu_i) \in T$ und $(\theta_j, \mu_j) \in T$, für die θ_i und θ_j kompatibel sind, gilt:

$$\sqcup(\mu_i, \mu_j) > \mu_i, \sqcup(\mu_i, \mu_j) > \mu_j \Rightarrow$$
$$\exists(\psi, \gamma) \in T \text{ mit } \psi \geq mgcd(\theta_i, \theta_j) \text{ und } \gamma \geq \sqcup(\mu_i, \mu_j)$$

dann ist T vollständig*.*

Diese Definition besagt, dass eine Tabelle dann vollständig ist, wenn sie alle Antworten enthält, die sich durch die Reduktionsregel bilden lassen. Somit sind auch die höchsten erreichbaren Annotationen in der Tabelle explizit enthalten.

Definition 5.4 (Minimalität von Tabellen)
Eine Tabelle T ist minimal*, wenn für alle $(\theta_i, \mu_i) \in T$ und $(\theta_j, \mu_j) \in T$ gilt:*

$$\theta_i \leq \theta_j \Rightarrow \mu_i > \mu_j$$

Bei der Minimalität von Tabellen wird sicher gestellt, dass alle Antworten, die spezieller sind als andere Antworten in der Tabelle auch eine höhere Annotation im Sinne der Ordnung des Annotationsverbands aufweisen. Andernfalls würden sie von den allgemeineren Substitutionen vollständig subsumiert.

Es ist sinnvoll für Tabellen bei annotierten logischen Programmen beide Eigenschaften zu fordern. Die Vollständigkeit stellt sicher, dass die Reduktionsregel vollständig angewendet wurde und die Tabelle keine Antworten enthält, die in Kombination höhere Annotationen ergeben würden. D.h. außer den Antworten in der Tabelle, die die Annotationsrandbedingung erfüllen, gibt es keine Antworten für das Unterziel. Die Minimalität vermeidet die Speicherung überflüssiger Substitutionen. Dies ist nicht nur wichtig, um den Speicherbedarf minimal zu halten, dadurch wird auch sicher gestellt, dass keine überflüssigen Substitutionen weiterverarbeitet werden und in Folge Aufwand verursachen.

Um Vollständigkeit und Minimalität in einer konkreten Datenstruktur sicherzustellen, bieten sich zwei Ansätze an:

1. Nach jeder Operation, die die Datenstruktur verändert (Einfügen oder Löschen) wird Vollständigkeit und Minimalität wieder hergestellt. Dies hat den Vorteil, dass die bekannten Operationen für bestimmte Datenstrukturen ohne Änderung verwendet werden können. Es ist eine zusätzliche Operation erforderlich, die unabhängig von der durchgeführten Operation die Eigenschaften wieder herstellt.

2. Die Sicherstellung der Eigenschaften wird in die Operationen eingearbeitet. Im Vergleich zu 1) verursacht dieser Ansatz einen geringeren Aufwand, da der Aufwand zur Bestimmung der betroffenen Antwortmenge nur einmal anfällt.

In der vorgestellten Architektur wird für Tabellen lediglich eine Operation zum Einfügen von neuen Antworten verlangt. Es bietet sich daher an, die Sicherstellung der Eigenschaften in dieser Operation zu realisieren. Im Folgenden stellen wir den grundlegenden Algorithmus zum Einfügen von Antwortsubstitutionen in eine subsumptive Tabelle für annotierte Logik vor. In diesem wird die Menge nur der Antwortsubstitutionen in einer Tabelle durchlaufen, die mit der neu einzufügenden Substitution kompatibel sind. Entsprechend der Subsumptionsordnung werden dann subsumierte Element aus der Tabelle entfernt und gegebenenfalls *allgemeinste gemeinsame Nenner* (MGCDs) gebildet. Wird der neue Eintrag von einer existierenden Substitution subsumiert, bricht der Algorithmus ab. Im anderen Fall wird die neue Substitution und eventuell entstandene MGCDs in die Tabelle eingefügt.

Algorithmus 5.1 (Einfügen)

T	:	*Unterzieltabelle.*
(φ, ν)	:	*Annotierte Antwortsubstitution, die in T eingefügt werden soll.*
U	:	*Menge der annotierten Antwortsubstitutionen (θ_i, μ_i) mit θ_i und φ sind kompatibel*
M	:	*temporäre Menge annotierter Antwortsubstitutionen.*

begin
  forall $(\theta_i, \mu_i) \in U$ **do**
    if $\theta_i \leq \varphi$ **then**
      if $\mu_i \leq \nu$ **then**
        $T := T \backslash \{(\theta_i, \mu_i)\}$.
      else
        $\mu_i := \sqcup(\mu_i, \nu)$
      end if.
    else if $\varphi < \theta_i$ **then**
      if $\nu \leq \mu_i$ **then**
        return.
      else
        $\nu := \sqcup(\mu_i, \nu)$
      end if.
    else if $\sqcup(\mu_i, \nu) > \mu_i \wedge \sqcup(\mu_i, \nu) > \nu$ **then**
        $M := M \cup \{(mgcd(\theta_i, \varphi), \sqcup(\mu_i, \nu))\}$
    end if.
  end forall.
  $T := T \cup \{(\varphi, \nu)\}$
  $T := T \cup M$
end.

Theorem 5.1 *Algorithmus 5.1 ist korrekt, d.h. er erhält die Vollständigkeit und Minimalität einer Datenstruktur im Sinne der Definitionen 5.3 und 5.4.*

Beweis 5.1 *Durch das Einfügen einer annotierten Antwortsubstitution (φ, ν) können laut Definitionen 5.3 und 5.4 nur diejenigen Einträge in einer Tabelle betroffen sein, die kompatibel zu φ sind*[1]*.*

Terminierung: *Die Schleife läuft über die Anzahl der Elemente in U oder bricht vorher ab. Da U stets endlich ist, terminiert der Algorithmus.*

Vollständigkeit: *Angenommen, nach Durchführung existieren zwei kompatible Einträge (θ_1, μ_1) und (θ_2, μ_2) in T mit $\sqcup(\mu_1, \mu_2) > \mu_1$ und*

[1] Für Definition 5.4 reicht es aus, wenn ein Eintrag spezieller ist. Man sieht leicht, dass, wenn eine Substitution φ spezieller ist als eine Substitution θ, φ und θ auch kompatibel sind.

$\sqcup(\mu_1, \mu_2) > \mu_2$ und $(mgcd(\theta_1, \theta_2), \sqcup(\mu_1, \mu_2))$ ist nicht in T. Da die Tabelle vor Ausführung vollständig war, muss entweder θ_1 oder θ_2 ein neu eingefügter Eintrag sein. Angenommen (θ_2, μ_2) sei der neue Eintrag.

1. $\theta_1 \leq \theta_2$
   (a) *$\mu_1 \leq \mu_2$. Dann ist $(mgcd(\theta_1, \theta_2), \sqcup(\mu_1, \mu_2))$ gleich (θ_1, μ_2), welches spezieller ist als (θ_2, μ_2). Dieses jedoch wird in T eingefügt. Widerspruch.*
   (b) *$\mu_1 \not\leq \mu_2$. Dann ist $(mgcd(\theta_1, \theta_2), \sqcup(\mu_1, \mu_2))$ gleich $(\theta_1, \sqcup(\mu_1, \mu_2))$, durch welches aber (θ_1, μ_1) ersetzt wird. Widerspruch.*
2. $\theta_2 < \theta_1$
   (a) *$\mu_2 \leq \mu_1$. Dann ist $(mgcd(\theta_1, \theta_2), \sqcup(\mu_1, \mu_2))$ gleich (θ_2, μ_1), welches spezieller ist als (θ_1, μ_1). Dieses ist bereits in T. Widerspruch.*
   (b) *$\mu_2 \not\leq \mu_1$. Dann ist $(mgcd(\theta_1, \theta_2), \sqcup(\mu_1, \mu_2))$ gleich $(\theta_2, \sqcup(\mu_1, \mu_2))$, durch welches aber (θ_2, μ_2) ersetzt und welches in T eingefügt wird. Widerspruch.*
3. *Weder 1. noch 2. trifft zu.*
   (a) *$\sqcup(\mu_1, \mu_2) = \mu_1$. Dann ist $(mgcd(\theta_1, \theta_2), \sqcup(\mu_1, \mu_2))$ gleich $(mgcd(\theta_1, \theta_2), \mu_1)$, welches spezieller ist als (θ_1, μ_1). Dieses ist jedoch in T. Widerspruch.*
   (b) *$\sqcup(\mu_1, \mu_2) = \mu_2$. Dann ist $(mgcd(\theta_1, \theta_2), \sqcup(\mu_1, \mu_2))$ gleich $(mgcd(\theta_1, \theta_2), \mu_2)$, welches spezieller ist als (θ_2, μ_2). Dieses wird jedoch in T eingefügt. Widerspruch.*
   (c) *Es gilt weder 3a noch 3b, dann wird $(mgcd(\theta_1, \theta_2), \sqcup(\mu_1, \mu_2))$ in M und schließlich in T eingefügt. Widerspruch.*

Aus den Widersprüchen ergibt sich die Vollständigkeit von T. □

Minimalität: *Angenommen, es existieren zwei Einträge (θ_1, μ_1) und (θ_2, μ_2) in T mit $\theta_1 \leq \theta_2$ und $\mu_1 \leq \mu_2$. Da T zuvor minimal war, muss entweder θ_1 oder θ_2 ein neu eingefügter Eintrag sein. Ein Eintrag, der über M eingefügt wurde kommt nicht in Betracht, da dort nur Einträge gemacht werden, wenn die Substitution spezieller ist als die Argumente und die resultierende Annotation die ursprünglichen Annotationen übersteigt. Sei (θ_2, μ_2) der neue Eintrag. Unter den gegebenen Bedingungen wird jedoch (θ_1, μ_1) aus T entfernt. Aus dem Widerspruch ergibt sich die Minimalität von T.* □

Der Aufwand zum Einfügen einer neuer Antwort ist linear bezüglich der Anzahl der kompatiblen Einträge in der Tabelle. Die Bestimmung der kompatiblen Einträge ist im schlechtesten Fall linear in Abhängigkeit von

der Gesamtzahl der Einträge in der Tabelle, so dass sich im schlechtesten Fall ein linearer Gesamtaufwand in Abhängigkeit von der Größe der Tabelle ergibt.

Enthält die Tabelle keine oder wenige kompatible Substitutionen, lässt sich ein weitaus geringerer Aufwand erreichen. Für das konkrete Problem der Speicherung von Antwortsubstitutionen ist die Ausnutzung derjenigen Variablenbelegungen entscheidend, die Substitutionen von vornherein inkompatibel machen und dazu führen, dass eine Substitution beim Einfügen nicht berücksichtigt zu werden braucht. In dieser Weise können Substitutionen mittels ihrer Belegungen durch Konstanten diskriminiert werden. Daher ist es sinnvoll, die konstanten Anteile zur Indizierung von Antwortsubstitutionen zu verwenden. Mit der Suchoperation kann dann einfach die Menge der potenziell kompatiblen Substitutionen ermittelt werden.

Wir bereits beschrieben, werden *Tries* als Datenstruktur zur variantenbasierten Tabulierung verwendet. Die Verwendung von *Tries* für subsumptionsbasierte Tabulierung hat einen entscheidenden Nachteil. Ein *Trie* stellt eigentlich eine eindimensionale Indexstruktur dar. Als Schlüssel wird die Verkettung der Attribute der Einträge verwendet. Es ist der *Trie*-Datenstruktur inhärent, dass die Indizierung der Einträge auf Präfixe der Schlüssel stattfindet. Daher sind nur diejenigen Suchoperationen effizient, bei denen ein möglichst langer Präfix definiert ist. Ist bereits das erste Argument des Suchschlüssels variabel und unterscheiden sich alle Einträge im ersten Argument, degeneriert die Suche zur linearen Suche über die gesamte Menge der Einträge. Die Suche könnte aber schneller sein, wenn ein anderes, konstantes Argument zur Suche herangezogen würde.

5.3 Multiattribut-Indexstrukturen

Multiattribut-Indexstrukturen wurden vorrangig im Forschungsbereich *Informationssysteme* untersucht. Von entscheidender Bedeutung bei Informationssystemen ist der Umgang mit großen Informationsmengen unter Verwendung des Sekundärspeichers. Da die Zugriffe auf den Sekundärspeicher der bestimmende zeitliche Faktor in diesem Umfeld darstellt, ist das Ziel beim Entwurf einer geeigneten Datenstruktur, die Anzahl der Zugriffe zu minimieren. Dazu werden die Daten blockweise organisiert und pro Zugriff ganze Blöcke gelesen oder gespeichert. Ein Datenblock wird auch *Seite* genannt. Die besondere Problematik bei Datenstrukturen mit Seitenorganisation besteht darin, den verfügbaren Speicherraum optimal auszunutzen und gleichzeitig die maximale Anzahl der benötigten Zugriffe zum Wiederfinden eines Datums möglichst gering zu halten.

Eine geeignete Indexstruktur zur datenintensiven subsumptiven Tabulierung sollte zumindest folgende Voraussetzungen erfüllen:

- Sie sollte eine gute mittlere Speichernutzung in Index- und Datenseiten

aufweisen.

- Ein hoher Fächerungsgrad sichert eine geringe Höhe des Indexbaumes bezüglich der Anzahl der Indexseiten. Das führt zu einer geringen Anzahl von Plattenzugriffen.
- Wenn die Struktur wächst, sollte die inkrementelle Reorganisation effizient durchführbar sein.
- Suchen mit partiellen Suchmustern müssen möglich sein.

Methoden zum Datenzugriff über Ordnungen, die für beliebige Mengen von Schlüsselwerten anwendbar sind, basieren auf der Baumsuche. Die klassische Indexstruktur für eindimensionale Schlüsselmengen ist der B-Baum [BM72]. B-Bäume sind annähernd balancierte, seitenorientierte Binärbäume.

Der naive Ansatz, Multiattribut-Indizierung zu erreichen, besteht darin, für jede Dimension einen eigenen konventionellen Index einzusetzen. Nachteile dieses Ansatzes sind hauptsächlich die schlechte Speichereffizienz und die hohen Kosten für Einfüge- und Löschoperationen. Diese Operationen müssen nämlich in jedem Index einzeln durchgeführt werden. Die Kosten steigen daher proportional zur Anzahl der Dimensionen. Grid File [NHS84], K-D-B-Bäume [Rob84] und R-Bäume [Gut84] mit Varianten sind bekannte Datenstrukturen für mehrdimensionale Indizierung. In jüngerer Zeit wurden TV-Bäume, X-Bäume [BKK96], hB-Bäume [LS90, ELS95] und UB-Bäume [Bay97] diskutiert. Diese Ansätze unterscheiden sich hauptsächlich in der Art und Weise, wie der Suchraum in Unterräume zerteilt wird, und welche Auswirkungen diese Unterteilung mit sich bringt. Außer bei Grid Files wird der geeignete Unterraum für ein Objekt durch die Suche in einer gemeinsamen Baumstruktur ermittelt, in der jeder Knoten den Unterraum entlang einer Dimension teilt. Ein Grid File ist ein Hyperwürfel, der die Dimension des Suchraums hat. Entsprechend des Füllgrads ist ein Grid File in den Dimensionen unterteilt in Unterräume, in die ein Eintrag eingeordnet wird.

Die meisten der obigen Indexstrukturen weisen die gewünschten Eigenschaften nur zum Teil auf. Für die Zwecke der subsumptiven Tabulierung scheinen hB-Bäume die geeignete Datenstruktur zu sein, da sie alle Eigenschaften bei beliebigen Daten garantiert. Eine detaillierte Analyse von Methoden zur Multiattribut-Indizierung kann man in [Lom92] nachlesen.

5.4 hB-Bäume

hB-Bäume, wie sie von Lomet und Salzberg [ELS95] vorgestellt wurden, weisen neben unseren Anforderungen weitere wünschenswerte Eigenschaften auf.

- hB-Bäume können einfach für die komplette Speicherung und Verwaltung von kleinen Datenmengen im Primärspeicher angepasst werden.

- hB-Bäume wurden für nebenläufigen Zugriff erweitert [ELS95]. Diese Eigenschaft macht hB-Bäume für eine parallele Resolutionsprozedur besonders geeignet. Der Vorteil der Parallelität kommt umso stärker zum tragen, je feiner die Granularität der atomaren Operationen der Datenstruktur ist. In [ELS95] wird definiert, welche Teiloperationen minimal beim gleichzeitigen Zugriff geschützt werden müssen, um die Korrektheit der Operation zu gewährleisten.

hB-Bäume sind von K-D-B-Bäumen abgeleitet. Sie bestehen aus Seiten, die jeweils einen k-d-Baum enthalten[2]. Ein k-d-Baum besteht im Wesentlichen aus zwei Sorten von Knoten. *Datenknoten* enthalten Datenobjekte und repräsentieren einen rechteckigen Unterraum des gesamten Indexraums. Indexknoten enthalten einen Indexterm und Referenzen zum linken und rechten Unterbaum. Eine Seite des hB-Baumes ist die Einheit zur Speicherung auf dem Sekundärmedium. Im Gegensatz zu K-D-B-Bäumen ist das Ziel der hB-Baumstruktur, abwärts kaskadierende Seitenteilungen zu verhindern und dadurch sowohl Restrukturierungskosten als auch ungünstige Speichernutzung zu vermeiden. Wenn eine Seite voll ist, wird die Aufteilung aufgrund der existierenden Grenzen innerhalb des Unterraumes der Seite durchgeführt. Daraus folgt die Entfernung eines kleineren „ziegelsteinförmigen“ Unterraumes aus einem größeren „ziegelsteinförmigen“ Unterraum. Das Resultat ist ein „lückenhafter Ziegelstein“ (engl.: *holey brick*), worin der Name dieser Struktur begründet liegt.

Im Rest des Kapitels werden wir die Anpassungen und Erweiterungen erläutern, die zur Verwendung von hB-Bäumen als Tabellenstruktur für subsumptive Tabulierung notwendig sind. Baumstrukturen, wie hB-Bäume, die den Indexraum sukzessive unterteilen, benötigen eine Totalordnung der für die Indizierung in Frage kommenden Domänen. Jedoch ist es möglich, Werte einer teilgeordneten oder ungeordneten Domäne auf eine totalgeordnete Domäne abzubilden[3]. Insofern stellt dies keine echte Einschränkung dar.

5.5 hBT-Tabellen

Wir stützen uns auf die hB-Baum Datenstruktur, wie sie in [LS90, ELS95] beschrieben ist. Die in Kapitel 4 vorgeschlagene Architektur erfordert die Bereitstellung folgender Mechanismen:

- Einfügen von Antwortsubstitutionen mit nicht gebundenen Variablen,
- Bestimmung von subsumierten Substitutionen (Suche mit teilweiser Übereinstimmung),

[2] Um Verwechslungen mit den Knoten des k-d-Baumes zu vermeiden, verwenden wir den Begriff *Seite* für die Knoten eines hB-Baums.

[3] z.B. durch *Hashing*

- Korrekte Seitenteilung unter Berücksichtigung dieser Änderungen.

Einfügen

Substitutionen mit ungebundenen Variablen werden wie bei der Standardversion der Einfügeoperation behandelt. Das heißt, der richtige Blattknoten wird durch Abstieg im hB-Baum bestimmt. In Indexknoten wird das jeweilige Argument mit dem Indexterm des Knotens verglichen und der Vorgang wird dem Vergleichsergebnis entsprechend im linken oder rechten Unterbaum fortgesetzt. Substitutionen mit ungebundenen Variablen repräsentieren Gebiete im Indexraum; sie werden in alle Datenknoten eingefügt, deren Unterraum sich mit dem Raum der Substitution überschneidet. Das bedeutet, dass, wenn die Argumentposition, die durch den Indexterm eines Indexknotens bestimmt wird, in der Substitution nicht gebunden ist, beide Unterbäume verfolgt werden. Dies stellt sicher, dass bei einer Subsumptionsanfrage immer alle relevanten Substitutionen gefunden werden. Auf der anderen Seite ist es möglich, dass Substitutionen in einem Datenknoten nicht mit der Anfragesubstitution kompatibel sind, obwohl die Suchanfrage zu dieser Seite geleitet wurde. Das liegt daran, dass eine Datenseite all die Substitutionen enthält, die entweder in dem durch die Datenseite definierten Unterraum liegen oder diesen überlappen. Es ist deshalb erforderlich, dass die Einträge in Datenknoten als Kandidaten für eine dort hingeführte Anfrage betrachtet werden. Das impliziert, dass für alle Einträge in der Datenseite ein Kompatibilitätstest durchgeführt werden muss[4]. Substitutionen, die keine Konstanten enthalten, werden gesondert behandelt (z.B. $[X/Y]$)). Diese Substitutionen müssten wegen ihrer Allgemeinheit in allen Datenseiten eingefügt werden. Um Redundanz zu vermeiden und die Korrektheit der Seitenteilung zu gewährleisten, werden diese Substitutionen an der Wurzel des Baumes abgelegt. Im Folgenden wird der Algorithmus im Pseudocode angegeben.

Algorithmus 5.2

InsertAnswer(*Knoten*,(φ, ν)**)**
begin
  if *Node ist ein Indexknoten* **then**
    SDim ist die Teilungsdimension des Indexterms
    if φ *ist nicht grundinstantiiert*
      an der Argumentposition SDim **then**
      InsertAnswer(linkes Kind,(φ, ν)*)*
      InsertAnswer(rechtes Kind,(φ, ν)*)*

[4]Man beachte, dass bei der hier verwendeten Behandlung Abhängigkeiten zwischen Variablen vernachlässigt werden. Eine Substitution $[X/1]$ wird an denselben Stellen eingefügt wie die Substitution $[X/1, Y/X]$.

else if *der Argumentwert von* φ *an der Argumentposition SDim ist kleiner als der Vergleichswert des Indexterms* **then**
InsertAnswer(linkes Kind,(φ, ν)*)*
else
InsertAnswer(rechtes Kind,(φ, ν)*)*
end if.
else *Knoten ist Datenknoten*
S ist die im Knoten gespeicherte Antwortmenge Wende Algorithmus 5.1 auf die kompatiblen Einträge von S an.
if $|S|$ *übersteigt die zulässige Knotengröße* **then**
SplitNode
end if.
end if.
end.

Bestimmung spezieller oder kompatibler Substitutionen

Die Suche nach spezielleren (kompatiblen) Substitutionen entspricht zunächst einer Anfrage mit teilweiser Übereinstimmung (engl.: *partial match query*). Eine solche Suche definiert einen Unterraum innerhalb des Suchraums. Als Ergebnis werden alle Substitutionen zurückgegeben, die im Unterraum enthalten sind (den Unterraum überlappen). Die Kandidaten-Datenknoten werden in der selben Weise, wie bei der Einfüge-Operation bestimmt. Der Algorithmus zur Bestimmung der spezielleren Substitutionen wird im Anschluss ausgeführt. Der Algorithmus für kompatible Substitutionen ist analog aufgebaut.

Algorithmus 5.3 GetSubsumed(*Knoten*,(φ, ν),*Ergebnis***)**
begin
if *Knoten ist ein Indexknoten* **then**
SDim ist die Teilungsdimension des Indexterms
if *Anfrage ist nicht grundinstantiiert an der Argumentposition SDim* **then**
GetSubsumed(linkes Kind,(φ, ν)*,Ergebnis)*
GetSubsumed(rechtes Kind,(φ, ν)*,Ergebnis)*
else if *der Argumentwert von Anfrage an der Argumentposition SDim ist kleiner als der Wert des Indexterms* **then**
GetSubsumed(linkes Kind,(φ, ν)*,Ergebnis)*
else
GetSubsumed(rechtes Kind,(φ, ν)*,Ergebnis)*
end if.
else if *Knoten ist Datenknoten* **then**
forall *Substitutionen* (θ_i, μ_i) *in der Antwortsammlung*

```
        if φ ≤ θ_i ∧ ν ≤ μ_i then
          füge (θ_i, μ_i) zu Ergebnis hinzu
        end if.
      end forall.
    end if.
end.
```

Seitenteilung

Da unsere Datenstruktur nur die Struktur der Datenknoten des k-d-Baums innerhalb einer Seite des hB-Baums abändert, ist es ausreichend, den Überlauf von Datenknoten hier zu betrachten. Ansonsten bleibt der Algorithmus zur Teilung von Seiten in einem hB-Baum bestehen. Die allgemeine Idee besteht darin, bei einem Knotenüberlauf den Inhalt des Knotens gleichmäßig auf zwei neue Knoten zu verteilen. Dazu muss das geeignete Kriterium zur Teilung gefunden werden.

In [LS90] wurde gezeigt, dass in einem Knoten mit Punktdaten eine Teilung immer mindestens im Verhältnis von 2 : 1 möglich ist. Im Falle von Antwortsubstitutionen sind das diejenigen, die keine Variablen enthalten. Man beachte, dass das Teilungskriterium möglicherweise mehrere Dimensionen des Indexraums einbezieht. Substitutionen werden dann auf die entsprechenden Seiten verteilt.

Bei räumlichen Daten, d. h. Antwortensubstitutionen mit variablen Argumenten, wie z.B. $A(1, X, Y)$, liegt eine andere Situation vor. Wenn eine Substitution oder in der Variable der Teilungsdimension nicht gebunden ist, d.h. beide Unterräume überschneidet, wird sie dupliziert und in beide Seiten eingefügt. Diese Technik wird auch bei *R+-Bäumen* angewendet [SRF87]. Ein Problem tritt dann auf, wenn jeder Eintrag den gesamten Unterraum eines Knotens überdeckt. Dann ist es nicht möglich ein geeignetes Teilungskriterium zu finden und der Knoten wird nicht geteilt. Unter diesen Umständen können größere Knoten entstehen, die allerdings höchstens $\frac{(dim-2)(dim-1)}{2}+1$ Einträge besitzen. Dies ergibt sich aus der maximalen Anzahl von möglichen Substitutionen mit ungebundenen Variablen bei Tabellen, die die Minimalitätseigenschaft erfüllen. Eine ähnliche Technik wird auch in [BKK96] im Zusammenhang mit X-Bäumen vorgeschlagen.

Algorithmus 5.4 SplitNode(*DatenKnoten***)**
begin
Finde Teilungskriterium welches die Substitutionen im Verhältnis 2 : 1 *teilt*
NeuerKnoten ist ein neuer Indexknoten mit linken und rechtem Nachfolger und einem Indexterm
forall *Substitutionen* (θ_i, μ_i) *in der Antwortsammlung*
Füge (θ_i, μ_i) *in NeuerKnoten ein*

end forall.
Tausche NeuerKnoten mit DatenKnoten aus
end.

5.6 Konsumentenregistrierung

Konsumenten werden in der vorgestellten Architektur durch Substitutionen repräsentiert. Da es für ein Unterziel größere Mengen von Konsumenten geben kann, stellt sich das Problem der Registrierung von Konsumenten ähnlich der Speicherung von Antwortsubstitutionen dar. Insofern bietet es sich an, die registrierten Konsumenten in einer ähnlichen Datenstruktur zu verwalten. Allerdings gibt es für diese Datenstruktur nicht die Forderung der Vollständigkeit oder Minimalität. Im Gegensatz zu Antwortsubstitutionen, bei denen die Reduktionsregel Anwendung findet, dürfen die Konsumenteneinträge nicht manipuliert werden. Außerdem wird eine Funktion zum Löschen von Konsumenten benötigt. Eine geeignete Datenstruktur zur Verwaltung der Konsumenten kann leicht aus der beschriebenen Datenstruktur abgeleitet werden.

5.7 Ergebnisse und Implementierung

Ein Vergleich mit anderen Implementierung gestaltet sich schwierig, da es keine vergleichbaren Datenstruktur für Tabulierung mit Sekundärspeicher gibt, die zudem die annotierte Logik unterstützen. Die folgenden Vergleiche sind auf die Ausführung verschiedener Tabulierungsvarianten im Hauptspeicher beschränkt. Tabelle 5.7 zeigt den Vergleich zwischen einer variantenbasierten Version von hbT-Tabellen und der subsumptionsbasierten Version. Als Test wurde die Berechnung der transitiven Hülle eines vollständigen binären Baums mit Faktenbasen unterschiedlicher Größe verwendet.

Beispiel 5.1 (Verwandtschaftsproblem)

```
#predicates
parent(ULONG,UONG):[FOUR]
ancestor(ULONG,ULONG):[FOUR]

#clauses
ancestor(X,Y):[t] <- parent(X,Y):[t]
ancestor(X,Y):[t] <- parent(X,Y):[t] & ancestor(Y,Z):[t]
parent(1,2):[t]
parent(1,3):[t]
parent(2,4):[t]
parent(2,5):[t]
```

⋮

```
parent(2046,4092):[t]
parent(2046,4093):[t]
parent(2047,4094):[t]
parent(2047,4095):[t]
```

Die Zeiten sind die in Sekunden angegebenen Laufzeiten der Anfrage ancestor(X,Y):[t] auf einem Pentium 233 MMX.

Größe von *parent*	Größe von *ancestor*	KOMET 1.2 variantenb.	KOMET 1.2 subsumptionsb.	Relative Verbesserung
512	3586	4.29	0.82	5.23
1024	8194	15.81	2.5	6.32
2048	18434	59.19	8.92	6.63
4096	40962	235.89	33.7	7.0
8192	90114	963.93	128.84	7.4

Tabelle 5.1: Performanzvergleich

In unseren Experimenten zeigt die Verwendung von hBT-Tabellen einen deutlichen Zeitgewinn gegenüber der variantenbasierten Tabulierungsmethode. hBT-Tabellen sind stabil unter verschiedenen Bedingungen, wie unterschiedliche Verteilungen und Prädikatstelligkeit. Detaillierte Untersuchungen zum Verhalten von hB-Bäumen bei unterschiedlicher Dimension des Suchraums sind in [LS90] zu finden. Die variantenbasierte Version der hBT-Tabellen weist eine geringe Performanz auf, da gegenwärtig Unterziele selbst im KOMET-System nicht indiziert werden. Dadurch kommt es bei der Abfrage nach einer bestehenden Unterzieltabelle zu einer linearen Suche in der Menge der tabulierten Unterziele. Durch die Verwendung von hB-Bäumen zur Verwaltung von Unterzielen und Programmklauseln könnten weitere Verbesserungen erzielt werden.

In diesem Kapitel wurden Datenstrukturen auf ihre Eignung zur subsumptiven Tabulierung hin untersucht. Es zeigt sich, dass durch Verwendung der annotierten Logik ganz spezifische Anforderungen an die Operationen einer Datenstruktur bestehen. Daher wurde ein Algorithmus zur Tabulierung von annotierten Antwortsubstitutionen entwickelt und dessen Einbindung in eine Indexstruktur zur Speicherung mehrdimensionaler Daten erläutert. Diese Datenstruktur wurde in das Mediatorsystem KOMET integriert.

Die Eitelkeit der Logik ist ja imstande,
eines Menschen Hirn gänzlich zu verwirren.

Edgar Allen Poe

Kapitel 6

Diskussion

Die Integration heterogener, verteilter Informationen erlangt im Zuge der wachsenden Bedeutung des Internets und der Verfügbarkeit von Daten in elektronischer Form eine immer größere Bedeutung. Um die verschiedenen Aspekte dieser Aufgabe zu lösen, werden flexible, effiziente und wiederverwendbare Systeme benötigt. Trotz aller Bestrebungen, Standards für den Austausch von Daten und Informationen zu etablieren, kann man keinesfalls davon ausgehen, dass das Problem der Informationsintegration auf die Erhaltung von 'Altsystemen' beschränkt ist. Es sind eine Reihe von Gründen auszumachen, weshalb mit Heterogenitäten auf der physischen als auch auf der semantischen Ebene bei der Repräsentation von Informationen und Wissen grundsätzlich zu rechnen ist:

- Es gibt keine *optimale* Repräsentation für Daten, Informationen oder Wissen. Bestenfalls stehen sich gleichwertige Repräsentationen für ein und das selbe Datum gegenüber bzw. je nach spezifischen Anforderungen ist eine gewisse Repräsentation vorzuziehen.
- Die Wahl der Repräsentation ist meist anwendungsgetrieben. Für Informationsquellen die nicht von vorne herein zur Anbindung an ein Mediatorsystem konzipiert wurden, können ganz andere Anforderungen gestellt worden sein, als sie für die Informationsintegration notwendig wären.
- Es bestehen unterschiedliche Auffassungen über die Anwendungsdomäne und der damit verbundenen Begriffe. Allein deshalb ist die Erstellung eines kanonisches Modells des Diskursbereichs ausgeschlossen.
- Eine einheitliche Datenrepräsentation ist aus ökonomischer Sicht nicht unbedingt im Interesse derjenigen Protagonisten des Marktes, die Einfluss auf diese Entwicklung hätten.

- In einer vernetzten Welt bietet sich die Möglichkeit, auf ein und die selbe Informationsquelle aus verschiedenen Kontexten zuzugreifen, die allerdings unterschiedliche Anforderungen stellen können.
- Es ist nicht abzusehen, welche Arten von Informationen in Zukunft zu repräsentieren sind. Es dürfte kaum möglich sein, eine Repräsentation zu finden, die mächtig genug wäre, alle denkbaren Ausprägungen von Wissen adäquat darzustellen. Eine Standardisierung würde die weitere Entwicklung allenfalls hemmen.

Die Bemühungen zur Schaffung eines Standards zum Datenaustausch der letzten Zeit reduzieren sich letztlich auf die Definition syntaktischer Vorgaben.

6.1 Zusammenfassung

Wie man sieht, werden Werkzeuge zur Informationsintegration angesichts wachsender Informations- und Wissensressourcen sicherlich weiterhin benötigt werden. In der vorliegenden Arbeit wird der Versuch unternommen, eine Architektur für ein universelles Basiswerkzeug zur Erstellung komplexer Mediatorsysteme zu entwickeln. Durch Verwendung der annotierten Logik zur Repräsentation von Integrationswissen kann die die Logik an die Anforderungen der Anwendung flexibel angepasst werden. Als Auswertungsprozedur wurde die SLG-Resolution zur Berechnung der *well-founded*-Semantik für annotierte Logik angepasst und für die Einbindung externer Informationen erweitert. Fortschritte wurden dabei in den folgenden Bereichen erzielt:

- Es wurde gezeigt, inwiefern sich die possibilistische Logik mit variablen Gewichten und unscharfen Konstanten (PLFC) aus [DPS96] auf die annotierte Logik abbilden lässt. Dies ist von Interesse, da einerseits die annotierte Logik selbst keine anwendungsspezifische Semantik besitzt, die in unmittelbarem Bezug zu typischen Formen des logischen Schließens steht. Vielmehr kann sie als Rahmen für solche Logiken dienen. Andererseits existiert für PLFC bislang keine Auswertungsprozedur, die implementiert worden wäre. Aus den Erfahrungen mit der Verarbeitung annotierter logischer Programme lassen sich so Erkenntnisse für die PLFC gewinnen. Andererseits wird damit die Flexibilität der annotierten Logik unter Beweis gestellt.
- Es wurde die *erweiterte Verschmelzungsregel* vorgeschlagen, womit ein Beitrag zur Vervollständigung der Resolution von Programmen der PLFC geleistet werden konnte. Es wurde gezeigt, wie diese Regel bei der Abbildung auf die annotierte Logik zu berücksichtigen ist.
- Es wurde eine Resolutionsprozedur im Hinblick auf die Verwendung in einem Mediatorsystem im Allgemeinen und auf die Verarbeitung

unbegrenzt großer Datenmengen durch Anpassungen eines Tabulierungsmechanismus unter Einbeziehung des Sekundärspeichers im Besonderen untersucht.

- Es wurde eine Datenstruktur vorgeschlagen, die für die subsumptive Tabulierung geeignet ist. Dabei wird insbesondere gezeigt, wie die Anforderungen der annotierten Logik mit den Operationen dieser Datenstruktur verbunden werden können.

- Es wurden eine Vielzahl von Aspekten einer Mediatorumgebung in einer einheitlichen Architektur zusammengefasst, und die Bedeutung eines Tabulierungsmechanismus im Bezug auf die daraus resultierenden Anforderungen aufgezeigt. Insbesondere erlaubt die Architektur die Parallelisierung der Anfragebearbeitung und erzielt damit aufgrund des asynchronen Zugriffs auf externe Informationsquellen potentiell ein verbessertes Antwortverhalten.

- Die Ergebnisse wurden in der Mediatorshell KOMET implementiert. Mit beispielhaften Anwendungen konnte die grundsätzliche Tragfähigkeit des Ansatzes demonstriert werden.

Der vorgeschlagene Ansatz zielt in die Richtung, den Anforderungen konkreter Anwendungen an die Mediatortechnologie in komplexen Umgebungen gerecht zu werden und damit Entwicklungen aus den Bereichen der Logik und des maschinellen Schließens für reale Anwendungen umzusetzen und nutzbar zu machen.

Neben den in dieser Arbeit behandelten Problemen sind jedoch eine Vielzahl anderer Aspekte in einer Mediatorumgebung zu berücksichtigen. Diese reichen von Datensicherheit und Zugriffsschutz über Skalierbarkeit bis zu Berücksichtigung von Änderungen von Informationsquellen. Die annotierte Logik könnte als Ansatzpunkte für entsprechende Lösungen dienen, da Annotationen genutzt werden können um Zusatzinformationen zu jedem Faktum zu transportieren. Wenn diese Informationen in einer Verbandsstruktur repräsentiert werden können, sind sie grundsätzlich auch der Verwendung in der annotierten Logik zugänglich.

6.2 Ausblick

Zum Schluss wollen wir einen groben Überblick über einige konkrete Fragestellungen für den logikbasierten Ansatz zur Informationsintegration, wie er hier vertreten wird, geben. Diese sind entweder insgesamt oder speziell im Projekt KOMET bisher unzureichend behandelt worden.

Ausdrucksmittel des logischen Formalismus

Aggregation Aggregation, d. h. die Zusammenfassung von Mengen mittels Berechnungsfunktionen, ist im Bereich der Verarbeitung großer Datenmengen ein wichtiges Mittel zur Reduktion der Datenmenge und Erzeugung von Informationen in 'kondensierter' Form. Die Einbettung von Aggregatfunktionen in die annotierte Logik bezüglich der *well-founded*-Semantik muss formal untersucht werden, um die Erhaltung der Eigenschaften des logischen Formalismus sicher zu stellen.

Typsysteme Bislang beruhen unsere Betrachtungen auf einer mehrsortigen Logik. Diese hat den Vorzug, dass alle Typprüfungen statisch, also einmalig, erfolgen können. In KOMET zeigt sich, dass auch ein solches einfaches Typsystem für eine Reihe von Anwendungen flexibel genug ist. Trotzdem könnte es sinnvoll sein, ein semantisch reicheres Typsystem zu verwenden, wenn man den zusätzlichen Berechnungsaufwand in Kauf nehmen kann.

Randbedingungen Die Anbindung externer Informationsquellen ist bislang auf einfache Randbedingungen (Konjunktionen von Relationen) beschränkt. Dies kann bestimmten Optimierungen im Wege stehen, wenn eine Informationsquelle auch in der Lage ist, komplexere Formen von Randbedingungen zu verarbeiten.

Auswertung annotierter Mediatorprogramme

Optimierung Antworttabellen können gezielt eingesetzt werden, um, unter Einsatz von zusätzlichem Speicherraum, die wiederholte Berechnung eines Teils des Suchraums zu vermeiden. Ein möglicher Ansatz diesen Vorgang zu steuern ist die kostenbasierte Bewertung des Nutzens der Tabulierung bestimmter Unterziele.

Verwalten von Änderungen Die Korrektheit eines Integrationssystems wird durch die Einbindung autonomer Informationsquellen aufgeweicht. Das Problem verschärft sich, wenn Zwischenergebnisse gespeichert werden. Einerseits sind die Grundlagen für einen angepassten Korrektheitsbegriff zu legen, andererseits könnte die Korrektheit für bestimmte Typen von Informationssystemen durch die Entwicklung geeigneter Mechanismen sicher gestellt werden.

Koordination und Verwaltung von Informationsquellen

Beschreibung von Informationsquellen Die semantisch reiche Beschreibung von Informationsquellen ist für verschiedene Aspekte der Integration wünschenswert. Metawissen kann beispielsweise zur Anfrageoptimierung genutzt werden. Es ist vorstellbar, dass ein Teil des Me-

diatorprogramms aus Metainformation automatisch generiert werden kann.

Dienste Informationssysteme komplexer Anwendungsdomänen lassen sich am besten auf der Grundlage einer expliziten Repräsentation der entsprechenden Domänenkonzepte realisieren. Um diese Konzeptualisierung nutzen zu können, müssen in einem Integrationssystem geeignete Dienste realisiert werden. Z.B. könnte eine ontologische Wissensbasis direkt als KOMET-Programm implementiert und in Mediatorprogramme als Informationsquelle eingebunden werden.

Wissensakquisition

Die Beispielanwendungen, die mit KOMET realisiert wurden, zeigen, dass die KOMET-Mediatorsprache aufgrund ihrer Ausdrucksstärke eine sehr kompakte Darstellung von deklarativem Integrationswissen zulässt. Allerdings wird die Erzeugung und vor allem die Wartung von Mediatorprogrammen dadurch eher erschwert. Bekanntermaßen stellt die Akquisition des benötigten Wissens ein Engpass bei der Erstellung wissensbasierter Systeme dar. In der Praxis ist den Trägern des benötigten Wissens dessen Formalisierung in der reinen Form oft kaum zuzumuten.

Automatische Erzeugung von Mediatorprogrammen Liegt über eine Informationsquelle ausreichendes Wissen in geeigneter Form vor, können einfache Integrationsprobleme durch die automatische Erzeugung von Mediatorprogramme gelöst werden. Im Allgemeinen allerdings dürfte dies aufgrund der Komplexität von Integrationsaufgaben nicht möglich sein.

Werkzeuge zur Akquisition Vielversprechender ist der Einsatz von geeigneten Werkzeugen, die die Formalisierung des Integrationswissens unterstützen. So wurde beispielsweise im Projekt KOMET die Darstellung von Mediatorprogrammen als Petrinetze untersucht. Wahrscheinlich sind solche graphischen Werkzeuge eine wesentliche Voraussetzung für den breiten Einsatz komplexer wissensbasierter Systeme.

Anhang A

Resolutions- und Tabulierungsoperationen

Wie in Kapitel 4 beschrieben, wird die Resolution mit Tabulierung in der vorgeschlagenen Architektur mit Hilfe einiger Funktionen realisiert. Drei davon werden durch die Resolutionskomponente $\mathcal{RE}$ realisiert und führen Resolutionsschritte durch. Drei weitere gehören zur Tabulierungskomponente $\mathcal{TE}$ und führen Datenspeicherung und -abfragen durch.

NewClause(s,Cl)
begin
  if Rumpf von Cl leer ist **then**
    Call $\mathcal{TE}$ :: $NewAnswer(s, Cl_h)$
  else
    Wähle ein Literal Cl_s in Cl
    Rufe $\mathcal{TE}$ :: $FindOrCreateSubgoal(Cl_s, Cl)$ auf
  end if.
end.

NewSubgoal(s)
begin
  forall Programmklauseln Cl die mit s unifizierbar sind
    Bestimme Substitution θ die Cl_h mit s unifiziert
    Wende θ auf Cl an
    Rufe $\mathcal{RE}$ :: $NewClause(s, Cl)$ auf
  end forall.
  Markiere s als vollständig
end.

ResolveAnswer(s,Cl,Cl_s,θ)

begin
  Resolviere das gewählte Literal Cl_s in Cl unter Verwendung von θ
  Rufe $\mathcal{RE} :: NewClause(s, Cl)$ auf
end.

NewAnswer(s,A)
begin
  if A wird nicht durch s subsumiert **then**
    Füge A in s ein und sende A an die entsprechenden Konsumenten
  end if.
end.

FindOrCreateSubgoal(l,Cl)
begin
  if l wird von einer Unterzieltabelle s subsumiert **then**
    Bestimme Menge U der unifizierbaren Antworten in s
    forall Antworten A in U
      Bestimme Substitution θ die A mit l unifiziert
      Rufe $\mathcal{RE} :: ResolveAnswer(s, Cl, l, \theta)$ auf
    end forall.
  else
    Erzeuge Unterziel s mit l als Unterzielliteral
  if s ist nicht vollständig ausgewertet **then**
    Registriere Cl als Konsument für s
  end if.
end.

CompleteSubgoal(s)
begin
  forall Konsumenten, mit negativem Literal Cl_s in der Klausel Cl
    Rufe $calTE :: ResolveNegative(s, Cl_s, Cl)$ auf
  end forall.
  Führe die Transformation *Simplification* durch
end.

Literaturverzeichnis

[AGS99] T. Alsinet, L. Godo und S. Sandri. On the Semantics and Automated Deduction for PLFC, a Logic of Possibilistic Uncertainty and Fuzziness. In *15th Conference on Uncertainty in Artificial Intelligence (UAI), Schweden, Stockholm*, Juli 1999.

[AS93] S. Adali und V. S. Subrahmanian. Amalgamating Knowledge Bases ii: Algorithms, Data Structures and Query Processing. *Univ. of Maryland Tech. Report CS-TR-3124*, August 1993.

[Bay97] R. Bayer. The Universal B-Tree for Multidimensional Indexing. In *Proceedings of the WWAC Conference*, pp. B–3–2–1–B–3–2–12, 1997.

[BBH+00] A. Bouguettaya, B. Benatallah, L. Hendra, M. Ouzzani und J. Beard. Supporting Dynamic Interactions among Web-Based Information Sources. *IEEE Tranactions on Knowledge and Data Engineering*, 12(5):779–801, 2000.

[BKK96] S. Berchthold, D. A. Keim und H.-P. Kriegel. The X-Tree: An Index Structure for High-dimensional Data. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pp. 28–39, 1996.

[BM72] R. Bayer und E. M. McCreight. Organization and Maintenance of Large Ordered Indices. *Acta Inf.*, 1(3):173–189, 1972.

[BR87] C. Beeri und R. Ramakrishnan. On the Power of Magic. In *Proceedings of the 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), San Diego, California, USA*, pp. 269–283, März 1987.

[BS98] H. A. Blair und V. S. Subrahmanian. Paraconsistent Logic Programming. *Theoretical Computer Science*, 68(2):135–833, 1998.

[CBK00] J. Calmet, C. Ballarin und P. Kullmann. Integration of Deduction and Computation. In *International Symposium on Applications of Computer Algebra (ISACA)*, Oktober 2000.

[CGH94] A. B. Cremers, U. Griefhahn und R. Hinze. *Deduktive Datenbanken*. Künstliche Intelligenz. Vieweg, 1994.

[CGK+90] D. Chimenti, R. Gamboa, R. Krishnamurty, S. Naqvi, S. Tsur und C. Zaniolo. The LDL System Prototype. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):76–90, 1990.

[CGMH+94] S.S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman und J. Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *Proc. IPSJ, Tokyo, Japan*, 1194.

[CJKS97] J. Calmet, S. Jekutsch, P. Kullmann und J. Schü. KOMET – A System for the Integration of Heterogeneous Information Sources. In *10th International Symposium on Methodologies for Intelligent Systems (ISMIS)*, 1997.

[CK99a] J. Calmet und P. Kullmann. A Data Structure for Subsumption-Based Tabling in Top-Down Resolution Engines for Data-Intensive Logic Applications. In *Zbigniew W. Ras, Andrzej Skowron (Eds.): Foundations of Intelligent Systems, 11th International Symposium, ISMIS '99, Warsaw, Poland, June 8–1, 1999, Proceedings. Lecture Notes in Computer Science, Vol. 1609*, pp. 475–483. Springer, 1999.

[CK99b] J. Calmet und P. Kullmann. Meta Web Search with KOMET. In *Workshop for Intelligent Information Integration (IJCAI), Stockholm, Sweden*, pp. 68–73, Juli 1999.

[CKT00] J. Calmet, P. Kullmann und M. Taneda. Composite Distributive Lattices as Annotation Domains for Mediators. In *5th International Conference on Artificial Intelligence and Symbolic Computation (AISC)*, Juli 2000.

[Cla78] K. L. Clark. Negation as Failure. In Gallaire und Minker [GM78], pp. 293–322.

[CSW95] W. Chen, T. Swift und D.S. Warren. Efficient Top-Down Computation of Queries under the Well-Founded Semantics. *Journal of Logic Programming*, 24(3):161–199, 1995.

[CW93] W. Chen und D. S. Warren. Query Evaluation Under the Well-founded Semantics. In *12th ACM Symposium on Principles of Database Systems*. ACM Press, 1993.

[CW96] W. Chen und D. S. Warren. Tabled Evaluation with Delaying for General Logic Programs. *Journal of the ACM*, 43(1):20–74, 1996.

[DLP94] D. Dubois, J. Lang und H. Prade. Possibilistic Logic. In Gabay et al. [GHR94].

[DP89] D. Dubois und H. Prade. Fuzzy sets, Probability and Measurement. *European Journal of Operational Research*, 40:135–154, 1989.

[DPS96] D. Dubois, H. Prade und S. Sandri. Possibilistic Logic Augmented with Fuzzy Constants and Fuzzy Quantifiers. In *Proc. IPMU'96*, pp. 1009–1014, 1996.

[DPS98] D. Dubois, H. Prade und S. Sandri. Possibilistic Logic with Fuzzy Constants and Fuzzy Quantifiers. In Arcelli und Martins, Hrsg., *Logic Programming and Soft Computing*, pp. 69–90. Wiley, 1998.

[DPS99] C. V. Damasio, L. M. Pereira und T. Swift. Coherent Well-Founded Annotated Logic Programs. In *International Workshop on Logic Programming and Non-Monotonic Reasoning, El Paso, Texas, USA*, pp. 206–220, Dezember 1999.

[DRR+95] S. Dawson, C. R. Ramakrishnan, I. V. Ramakrishnan, K. F. Sagonas, S. Skiena, T. Swift und D. S. Warren. Unification Factoring for Efficient Execution of Logic Programs. In *22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California*, pp. 247–258, Januar 1995.

[ELS95] G. Evangelidis, D. Lomet und B. Salzberg. A Multi-attribute Index Supporting Concurrency, Recovery and Node Consolidation. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 1995.

[FHSW95] J. Freira, R. Hu, T. Swift und D. S. Warren. Pallelizing Tabled Evaluations. In *7th International PLILP Symposium*, pp. 115–132. Springer, März 1995.

[Fit91] M. C. Fitting. Well-Founded Semantics, Generalized. In *Proc. Intl. Logic Programming Symposium*, pp. 71–83. MIT Press, 1991.

[FLM98] D. Florescu, A. Y. Levy und A. O. Mendelzon. Database Techniques for the World-Wide Web: A Survey. *SIGMOD Record*, 27(3):59–74, 1998.

[FLM99] M. Friedman, A. Levy und T. Millstein. Navigational Plans for Data Integration. In *Workshop for Intelligent Information Integration (IJCAI), Stockholm, Sweden*, Juli 1999.

[Gab96] D. M. Gabay. *Labelled Deductive Systems*, Vol. 1. Oxford University Press, 1996.

[GHR94] D. M. Gabay, C. J. Hogger und J. A. Robinson, Hrsg. *Handbook of Logic in Artificial Intelligence and Logic Programming*, Vol. 3. Oxford University Press, 1994.

[GKD97] M. R. Genesereth, A. M. Keller und O. M. Duschka. Infomaster: An Information Integration System. In Joan Peckham, Hrsg., *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pp. 539–542. ACM Press, 1997.

[GM78] H. Gallaire und J. Minker, Hrsg. *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, 1977. Advances in Data Base Theory.* Plenum Press, New York, 1978.

[GMPQ+97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J.D. Ullman, V. Vassalos und J. Widom. The TSIMMIS Approach to Mediation; Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[Gut84] Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In Beatrice Yormark, Hrsg., *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, USA June 18-21, 1984*, pp. 47–57. ACM Press, 1984.

[HK95] R. Hull und R. King. Reference Architecture for the Intelligent Integration of Information. In *Technischer Bericht, I3 Projekt*, 1995.

[Hut] D. Hutter. Annotated Reasoning. *Annals of Mathematics and Artificial Intelligence (AMAI). Special issue on Strategies in Automated Deduction. In Vorbereitung.*

[JBB+97] R. J. Bayardo Jr., B. Bohrer, R. S. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. H. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan,

A. Unruh und D. Woelk. The Infosleuth Project. In Joan Peckham, Hrsg., *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pp. 543–545. ACM Press, 1997.

[Jek98] S. Jekutsch. Entwurf und Implementierung eines Generischen Anfrageübersetzers zur Integration Heterogener Informationsquellen. Diplomarbeit, Institut für Algorithmen und Kognitive Systeme, Universität Karlsruhe, Juni 1998.

[JL87] J. Jaffar und J. L. Lassez. Constraint Logic Programming. In *Proc. ACM Principles of Programming Languages*, pp. 111–119, 1987.

[KK] P. Kullmann und J. Kullmann. PaleoWeb: Integrierte Nutzung verteilter heterogener paläontologischer Datenbanken. *Mathematische Geologie. In Vorbereitung.*

[KK71] R. A. Kowalski und D. Kuehner. Lineare Resolution with Selection Function. *Artificial Intelligence*, 2:227–260, 1971.

[KLSS] P. Kullmann, J. J. Lu, J. Schü und T. Swift. SLG-Resolution for Annotated Logic Programming with Negation. *In Vorbereitung.*

[KLSS95] T. Kirk, A. Y. Levy, Y. Sagiv und D. Srivastava. The Information Manifold. In *Proceedings of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments, Stanford, California*, März 1995.

[KS92] M. Kifer und V. S. Subrahmanian. Theory of Generalized Annotated Logic Programming and its Applications. *Journal of Logic Programming*, 12(1):335–367, 1992.

[KS99] P. Kullmann und S. Sandri. Possibilistic Logic as an Annotated Logic. In *8th International Conference on Fuzzy Systems (IEEE-FUZZ), Seoul, Korea*, August 1999.

[KS01] P. Kullmann und S. Sandri. Implementation of Possibilistic Logig in an Annotated Logic Theorem Prover. In *Zur Veröffentlichung angenommen in Joint IFSA/NAFIPS Conference on Fuzziness and Soft Computing in the new Millenium, Vancouver, Canada*, Juli 2001.

[Kur90] F. Kurfeß. *Potentiality of Parallelism in Logic*, pp. 3–25. Springer, 1990.

[LL96] S. M. Leach und J. J. Lu. Query Processing in Annotated Logic Programming: Theory and Implementation. *Journal of Intelligent Informormation Systems*, 6(1):33–58, 1996.

[LMR93] J. J. Lu, N. V. Murray und E. Rosenthal. Signed Formulas and Annotated Logics. *Proc. 23rd International Symposium on Multiple-Valued Logics*, pp. 48–53, 1993.

[LNRS93] J. J. Lu, A. Nerode, J. Remmel und V. S. Subrahmanian. Towards a Theory of Hybrid Knowledge Bases. *Univ. of Maryland Tech. Report CS-TR-3037*, 1993.

[LNS96] J. J. Lu, A. Nerode und V. S. Subrahmanian. Hybrid knowledge bases. *IEEE Transactions on Data and Knowledge Engineering*, 8(5):773–785, 1996.

[Lom92] D. Lomet. A Review of Recent Work on Multi-attribute Access Methods. *SIGMOD Record*, 20(3), September 1992.

[LRO96] A. Levy, A. Rajaraman und J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB), Mumbai (Bombay), India*, pp. 251–262, September 1996.

[LS90] D. Lomet und B. Salzberg. The hB-Tree: A Multiattribute Indexing Method with Good Guaranteed Performance. *ACM Transactions on Database Systems*, 15(4), Dezember 1990.

[MKSI96] E. Mena, V. Kashyap, A. P. Sheth und A. Illarramendi. OBSERVER: An Approach for Query Processing in Global Iformation Systems based on Interoperation of Pre-existing Ontologies. In *Proc. of CoopIS, Brussels, Belgium*, 1996.

[NHS84] J. Nievergelt, H. Hinterberger und K. J. Sevcik. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Transactions on Database Systems*, 9(1), 1984.

[RAH+96] M. T. Roth, M. Arya, L. M. Haas, M. J. Carey, W. F. Cody, R. Fagin, P. M. Schwarz, J. Thomas II und E. L. Wimmers. The Garlic Project. In H. V. Jagadish und I. S. Mumick, Hrsg., *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, p. 557. ACM Press, 1996.

[Rao97] P. Rao. *Efficient Data Structures for Tabled Resolution*. Dissertation, State University of New York at Stony Brook, November 1997.

[Rei78] R. Reiter. On Closed-World Databases. In Gallaire und Minker [GM78], pp. 55–76.

[Rob84] J. T. Robinson. The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes. In *Proceedings ACM SIGMOD Conference on Management of Data, Boston, USA*, pp. 10–18, 1984.

[RRR96] P. Rao, C. R. Ramakrishnan und I. V. Ramakrishnan. A Thread in Time Saves Tabling Time. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, 1996.

[RSC99] R. Rocha, F. Silva und V. S. Costa. Or-parallelism within Tabling. In *Int. Workshop on Practical Aspects of Deductive Languages (PADL'99), San Antonio, Texas, USA, LNCS 1551*, pp. 137–151. Springer, Januar 1999.

[RSC00] R. Rocha, F. Silva und V. S. Costa. A Tabling Engine for the Yap Prolog System. In *Joint Conference on Declarative Programming (AGP2000), La Habana, Cuba*, Dezember 2000.

[RU95] R. Ramakrishnan und J. D. Ullman. A Survey of Research in Deductive Database Systems. *Journal of Logic Programming*, 23(2):125–149, 1995.

[SAB+95] V.S. Subrahmanian, S. Adali, A. Brink, J.J. Lu, R. Emery, A. Rajput, T.J. Rogers, R. Ross und C. Ward. HERMES: A Heterogeneous Reasoning and Mediator System. Technischer Bericht, 1995.

[Sch95] J. Schü. *Updates and Query-Processing in a Mediator Aechitecture*. Dissertation, Fakultät für Informatik, Universität Karlsruhe, 1995.

[SRF87] T. Sellis, N. Roussopoulus und C. Faloutsus. The R+-Tree: A Dynamic Index for Multi-dimensional Objects. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 1987.

[SSW93] K. F. Sagonas, T. Swift und D. S. Warren. The XSB Programming System. In *Workshop on Programming with Logic Databases (ILPS), Vancouver, B.C., Canada*, p. 164, Oktober 1993.

[Sub92] V. S. Subrahmanian. Amalgamating Knowledge Bases. *Univ. of Maryland Tech. Report CS-TR-2949*, August 1992.

[Tan99] M. Taneda. Semantik Mehrfacher Annotationen und Implementation in der Mediatorshell KOMET. Diplomarbeit, Fakultät für Informatik, Universität Karlsruhe, Juni 1999.

[TRV98] A. Tomasic, L. Raschid und P. Valduriez. Scaling Access to Heterogeneous Data Sources with DISCO. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):808–823, 1998.

[TS86] H. Tamaki und T. Sato. OLD Resolution with Tabulation. In *International Conference on Logic Programming*, pp. 84–98, 1986.

[Ull88] J. D. Ullman. *Principles of Data and Knowledge-Base Systems*, Vol. 3. Computer Science Press, New York, 1988.

[Wie92] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, März 1992.

[Wie93] G. Wiederhold. Intelligent Integration of Information. In *ACM SIGMOD Conference*, pp. 434–437, Mai 1993.

[Wie96] G. Wiederhold. *Intelligent Integration of Information.* Special Issue of the Journal of Intelligent Information Systems. Kluwer Academic Publishers, 1996.