TECHNISCHE UNIVERSITÄT
ILMENAU

Fakultät für Informatik und Automatisierung
Institut für Technische Informatik und Ingenieurinformatik
Fachgebiet Künstliche Intelligenz

Bachelorarbeit

Implementierung eines Deduktionstools über HORN-Klauseln der Prädikatenlogik mit Gleichheit

Vorgelegt von:
Andreas Loth

Betreuer: apl. Prof. Dr.-Ing. habil. Rainer Knauf

Studiengang: Informatik
Matrikel-Nr.: 43358

Eingereicht: Ilmenau, den 08. Juni 2012

Kurzfassung

Diese Arbeit ist eine Machbarkeitsstudie, welche untersucht, inwieweit die Paramodulation in einem Deduktionstool realisierbar ist. Wenn sie realisierbar ist, soll ein solches Tool implementiert werden, dass die Paramodulation unterstützt. Das Ergebnis der Studie ist, dass die Paramodulation in einem Deduktionstool mit Einschränkungen realisierbar ist. In der Arbeit werden die logischen Grundlagen für die Deduktion vorgestellt. Es wird weiterhin die Paramodulationsregel definiert und gezeigt, dass sie die Gleichheitsaxiome erfüllt. Zwei eingeschränkte Varianten der Paramodulationen werden vorgestellt: die geordnete und die gerichtete Paramodulation. Für das Deduktionstool werden Algorithmen für die Komposition von Termersetzungen und für die Unifikation gezeigt. Zur einfachen Erweiterung des Tools werden Plug-ins verwendet. Als Optimierungen werden die Eliminierung von Teilzielen und die Speicherung der Hypothese bereits erreichter Ziele (für eine Erkennung von unendlichen Schleifen) eingesetzt. Weiterhin wird die Syntax der Paramodulation im Deduktionstool und die Einordnung der Paramodulation in den Kontrollfluss des Tools beschrieben. Bei der Implementierung der Paramodulationsregel sind Probleme aufgetreten. Deren Lösungen und die daraus entstandenen Einschränkungen werden beschrieben.

Danksagung

Zuerst bedanke ich mich bei meiner Familie. Ohne sie wäre ich niemals so weit gekommen.
Ich weiß, dass besonders mein erstes Semester *für euch* sehr schwer war. Auch weiß ich, dass ich immer auf euch Zählen kann!

Auch meinen Freunden danke ich. Für die Hilfe mit LaTeX, dem Korrekturlesen und dem Zuspruch, dass ich diese Arbeit doch noch irgendwie schaffen werde :-)

Natürlich gilt mein Dank auch meinem Betreuer Herr Knauf für das interessante Thema und die Hilfe bei der Anfertigung dieser Arbeit.
Die Implementierung von PHP-Prolog war zwar eine ganz schöne Herausforderung, hat aber auch Spaß gemacht.

Butterfly Effect

It has been said that something as
small as the flutter of a butterfly's
wing can ultimately cause a typhoon
halfway around the world.
- Chaos Theory
[aus: Butterfly Effect (Film, 2004)]

Es kommt oft auf die „Kleinigkeiten“ an - sie können große Auswirkungen haben.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

1.1 Themengebiet und Motivation

Methoden der Künstlichen Intelligenz werden heute sehr vielfältig eingesetzt. Der Anwendungsbereich reicht dabei von wenig kritischen Anwendungen, wie zum Beispiel Computerspiele, bis hin zu sehr kritischen Anwendungen, beispielsweise in der medizinischen Diagnose oder der Überwachung von Kernkraftwerken. [Wika, 4.1 Aufgabenklassen und bekannte Expertensysteme]

Grundlage sind dabei oft regelbasierte Systeme. Die Regeln haben im Allgemeinen die Form „Wenn A und B und C, dann X“. Sie werden von menschlichen Experten erstellt. Mit solchen Systemen werden Schlussfolgerungen abgeleitet.

Menschen treffen im Allgemeinen auf einer semantischen Ebene Schlussfolgerungen, mit den Fakten der konkreten Anwendung. Normalerweise interpretieren sie Regeln und schlussfolgern in „der Welt der Bedeutungen“. Es ist für Menschen zwar auch möglich auf rein syntaktischer Ebene zu arbeiten und zu schlussfolgern wie ein Computer, aber ein solches Vorgehen ist nicht intuitiv. Im Gegensatz dazu kann ein Computer nur auf syntaktischer Ebene arbeiten, das heißt „Rechenregeln“ auf Symbole anwenden. Computer interpretieren nicht und können keine Bedeutung erkennen.

Schlussfolgerungen können durch Vorwärtsverkettung (datengetrieben) und Rückwärtsverkettung (zielgetrieben) gezogen werden. Vorwärtsverkettung bedeutet, dass aus der vorhandenen Wissensbasis Schlussfolgerungen gezogen und diese der Wissensbasis hinzugefügt werden. Rückwärtsverkettung dagegen geht vom Ziel (der Hypothese) aus und versucht durch Anwendung der Wissensbasis zu zeigen, dass es aus der Wissensbasis geschlussfolgert beziehungsweise bewiesen werden kann. [Man08, Seite 5f]

Ein kleines Beispiel soll die unterschiedlichen Vorgehensweisen von Menschen und Computern verdeutlichen: Herr Müller ist Rentner. Weiterhin sind Herr Müller und Frau Schmidt Nachbarn. Ist der Nachbar von Frau Schmidt Rentner? In der Sprache Prolog ausgedrückt sieht das Beispiel wie folgt aus:

$rentner(herr\_mueller).$
$nachbarn(herr\_mueller, frau\_schmidt).$
$?- rentner(nachbar\_von(frau\_schmidt)).$

Ein Mensch wird, sofern er die deutsche Sprache beherrscht, sofort sagen: „Ja, der Nachbar von Frau Schmidt, nämlich Herr Müller, ist Rentner.“ Der Computer beziehungsweise der Prolog-Interpreter dagegen wird „Nein“ antworten, weil er keine Folge von Ableitungsschritten findet, welche die Hypothese bestätigt. Prolog arbeitet nach dem Prinzip der „closed world assumption“, das heißt alle nicht in der Wissensbasis notierten oder daraus ableitbaren Aussagen gelten als falsch [Man08, Seite 15]. Wenn eine Hypothese nicht ableitbar ist, antwortet es mit „Nein“. Der Mensch interpretiert aus dem Fakt, dass Herr Müller und Frau Schmidt Nachbarn sind, dass Herr Müller der Nachbar von Frau Schmidt ist. Prolog wendet ausschließlich Resolutionsregeln an und erkennt nicht den Zusammenhang zwischen dem Prädikat $nachbarn(A, B)$ und dem strukturierten Term $nachbar\_von(B)$, der einen funktionalen Zusammenhang zwischen den Individuen ausdrückt.

Eine mögliche Lösung wäre folgender Prolog-Code:

$rentner(herr\_mueller).$
$nachbarn(herr\_mueller, frau\_schmidt).$
$? - rentner(X), nachbarn(X, frau\_schmidt).$

Die Lösung hat aber zwei Nachteile. Zum einen berücksichtigt sie nicht, dass die Nachbarschaftsrelation symmetrisch ist, und zum Anderen ist sie weniger intuitiv. Eine Regel, die den Term $nachbar\_von(frau\_schmidt)$ mit dem Term $herr\_mueller$ unter bestimmten Bedingungen gleich setzt, wäre wünschenswert.

1.2 Zielstellung

Die Paramodulationsregel erlaubt es, Regeln der Form

$$(a \equiv b) \leftarrow p_1 \wedge p_2 \wedge \ldots \wedge p_n$$

zu erstellen und zu verarbeiten. Dabei sind a und b Terme, die unter den konjunktiv verknüpften Bedingungen p_1 bis p_n als semantisch gleich deklariert werden sollen. p_1 bis p_n sind nicht negierte Atomformeln der Prädikatenlogik erster Stufe. Durch die Paramodulationsregel werden die Terme a und b als einander ersetzbar erklärt und die Bedingungen p_1 bis p_n konjunktiv in die verbleibende Hypothese aufgenommen [Kna12, Seite 4f]. Interessanterweise ist noch kein Prolog-System auf dem Markt, welches auch die Paramodulation beinhaltet. Das hat sicher Gründe, die als ein „Nebenergebnis“ der Arbeit klar werden.

Es soll eine Machbarkeitsstudie durchgeführt werden, die untersucht, ob sich die Paramodulationsregel in einem Deduktionstool realisieren lässt beziehungsweise welche Einschränkungen dabei getroffen werden müssen. Dabei soll, soweit möglich, ein Deduktionstool implementiert werden, dass die Paramodulationsregel unterstützt.

Bei der Implementierung des Deduktionstools ist die Effizienz vernachlässigbar.

Das Tool soll möglichst einfach zu verstehen, zu ändern und zu erweitern sein, um damit später auch andere Versuche durchzuführen.

In Absprache mit dem Betreuer wurde als Programmiersprache PHP ausgewählt. Weiterhin sind die Syntax des Deduktionstools und der Ablauf der Inferenz ähnlich deren von Prolog.

1.3 Aufbau der Arbeit

Das folgende Kapitel 2 gibt einen kurzen Überblick über die logischen Grundlagen, die in dieser Arbeit verwendet werden. Dazu gehören die Prädikatenlogik erster Stufe, HORN-Klauseln und das Folgern. Prolog und das zu implementierende Deduktionstool basieren auf diesen Grundlagen.

Danach werden am Anfang des Kapitels 3 Möglichkeiten aufgezeigt, mit denen die Gleichheit in einem Deduktionstool realisiert werden können. Die nachfolgenden Abschnitte des Kapitels behandeln die Paramodulationsregel. Sie wird definiert und es wird beschrieben, wie sie angewendet wird. Da die Paramodulation die Gleichheit realisieren soll, wird gezeigt, dass die Paramodulation die Gleichheitsaxiome erfüllt. Außerdem werden noch zwei eingeschränkte Varianten der Paramodulation vorgestellt, die auch bei der Implementierung des Deduktionstools eingesetzt werden.

Kapitel 4 beginnt mit einem kurzen Überblick über die Arbeitsweise von Prolog, die auch als Grundlage für das implementierte Deduktionstool verwendet wurde. Die folgenden Abschnitte behandeln wichtige Ideen und Algorithmen, welche im Tool eingesetzt werden. Weiterhin wird die Syntax der Programmiersprache des Deduktionstools, die an Prolog angelehnt ist, beschrieben. Darüber hinaus wird beschrieben, wie Plug-ins den Funktionsumfang des Tools erweitern können. Zum Schluss des Kapitels werden Optimierungen behandelt, die für die Paramodulation benötigt werden.

Das Kapitel 5 handelt von der Implementierung der Paramodulationsregel im Deduktionstools. Dabei wird zuerst die Syntax der Regel im Tool beschrieben. Damit Programmierer den Ablauf der Inferenz im Deduktionstool nachvollziehen können, wird danach beschrieben, wie sich die Paramodulationsregel in den Kontrollfluss einordnet. Im Anschluss folgen Probleme, die während der Implementierung der Paramodulation aufgetreten sind, und ihre Lösungen. Da die Lösungen zu Einschränkungen geführt haben, werden die Einschränkungen am Ende des Kapitels zusammengefasst.

Das letzte Kapitel fasst die Arbeit zusammen. Außerdem wird ein Ausblick gegeben auf Aspekte, die noch genauer untersucht werden können.

1.4 Notationen

Diese Arbeit verwendet hauptsächlich die Notationen aus [Kna11, Kna12, Kna].

- *Prädikaten-, Individuen- und Funktionssymbole* beginnen mit lateinischen Kleinbuchstaben.
- *Variablennamen* beginnen mit lateinischen Großbuchstaben.
- Atomformeln der Prädikatenlogik erster Stufe sind Prädikatensymbole gefolgt von einer in runden Klammern eingeschlossenen Liste von Termen als Argumente. Terme werden im Abschnitt 2.1 definiert.
- Eine *Termersetzung* ist ein Tupel der Form $[V, t]$, wobei V eine Variable und t ein Term ist. Eine *Substitution* $\vartheta = \{[V_1, t_1], \ldots, [V_n, t_n]\}$ ist eine Menge bestehend aus einzelnen Termersetzungen. Substitutionen werden mit griechischen Kleinbuchstaben bezeichnet. Die Begriffe Termersetzungen, Termeinsetzungen und Substitution werden synonym verwendet.
- $\vartheta(t)$ ist die Anwendung der Termersetzungen ϑ auf den Term t.

$$\vartheta(t) := \begin{cases} t_i, & \text{wenn } t \text{ eine Variable ist und } \exists [t, t_i] \in \vartheta\,, \\ f(\vartheta(t_1), \ldots, \vartheta(t_n)), & \text{wenn } t = f(t_1, \ldots, t_n), \\ t, & \text{sonst.} \end{cases}$$

- Seien ϑ_1 und ϑ_2 Termersetzungen und t ein Term. Die *Komposition* beziehungsweise Hintereinanderausführung $\vartheta_1 \circ \vartheta_2$ ist definiert als:

$$(\vartheta_1 \circ \vartheta_2)(t) := \vartheta_1(\vartheta_2(t))$$

- Der Term b ist ein *Teilterm* von Term a, wenn $a = b$ oder a ein strukturierter Term und b ein Teilterm von mindestens einem Argument von a ist. b ist ein echter Teilterm von a, wenn b ein Teilterm von a ist und $a \neq b$.

KAPITEL 2

Logische Grundlagen

Wie bereits in Abschnitt 1.2 erwähnt, soll die Sprache Prolog als Grundlage für das zu entwickelnde Deduktionstool dienen. Die Grundlage für Prolog ist die Prädikatenlogik. Genauer gesagt nutzt Prolog nicht die komplette Prädikatenlogik, sondern HORN-Klauseln der Prädikatenlogik. [Rot92, Seite 14]

2.1 Kurzer Überblick über die Prädikatenlogik erster Stufe (PK1)

Es sei U die Objektmenge, die Menge aller Individuensymbole. Eine n-stellige Funktion ist eine Abbildung:

$$f\colon U^n \to U$$

Ein n-stelliges Prädikat ist eine Abbildung:

$$p\colon U^n \to \{wahr, falsch\}$$

[Kna11, Seite 3]

Weiterhin gibt es im PK1 Variablen. Ihr Wertebereich ist nicht beschränkt. Variablen können durch Quantoren gebunden werden. Der Allquantor $\forall$ hat die Bedeutung, dass eine Aussage nur wahr ist, wenn sie für alle möglichen Werte der allquantifizierten Variable wahr ist. Der Existenzquantor $\exists$ dagegen hat die Bedeutung, dass eine Aussage wahr ist, wenn sie für mindestens einen Wert der existenzquantifizierten Variable wahr ist. [Rot92, Seite 15]

Außerdem wird noch der Begriff Term benötigt. Alle Variablen und alle Individuensymbole sind Terme. Weiterhin sind nur noch n-stellige Funktionssymbole, gefolgt von einer in Klammern eingeschlossenen Liste von n Termen, Terme. [Kna11, Seite 4]

Der Zusatz „erste Stufe“ bedeutet, dass nicht über Prädikate, also Mengen oder Eigenschaften, quantifiziert werden kann, sondern nur über Individuensymbole. [Man08, Seite 7, Fußnote]

2.2 HORN-Klauseln

Klauseln haben die Form

$$\forall x_1 \ldots \forall x_n (a_1 \vee \ldots \vee a_m \vee \neg b_1 \vee \ldots \vee \neg b_o)$$

wobei a_1 bis a_m und b_1 bis b_o für quantorenfreie Atomformeln stehen. [Rot92, Seite 16]

Literale sind negierte oder unnegierte (positive) Atomformeln. HORN-Klauseln sind Klauseln mit höchstens einem positiven Literal. Seien a, b_1 bis b_o quantorenfreie Atomformeln. Dann ist

$$\forall x_1 \ldots \forall x_n (a \vee \neg b_1 \vee \ldots \vee \neg b_o)$$

eine HORN-Klausel. In Klauselschreibweise sieht die Klausel wie folgt aus:

$$a \leftarrow b_i \wedge \ldots \wedge b_n$$

Alle Variablen werden implizit als allquantifiziert mit Wirkungsbereich über die gesamte Klausel angenommen. [Rot92, Seite 17] [Kna11, Seite 6]

Jede prädikatenlogische Formel lässt sich mittels der folgenden Schrittfolge in Klausel-Form (eine Menge von Klauseln, die konjunktiv verknüpft sind) umwandeln:

1. Pränex-Form: Alle Quantoren stehen am Anfang der Formel.
2. Skolem-Standardform: Existenzquantoren werden eliminiert.
3. Klausel-Form: Der quantorenfreie Teil am Ende der Formel wird in die konjunktive Normalform überführt.

[Rot92, Seite 15f]

[Kna, Folie 20 - 22] schlägt vor, vor Schritt 1 die Formel in die verneinungstechnische Normalform zu überführen. Der Schritt kann aber, nach Meinung des Autors, implizit in den Schritten 1 und 3 erfolgen. Nach [Kna, Folie 20] ist Schritt 2 keine äquivalente Umformung, aber kontradiktorizitätserhaltend.

Wenn in jeder Klausel maximal ein positives Literal vorkommt, ist die ursprüngliche Formel kontradiktorizitätserhaltend in eine Menge von HORN-Klauseln umgeformt worden. Wenn in einer Klausel mehr als ein positives Literal vorkommt, kann die Formel nicht in eine Menge von HORN-Klauseln umgewandelt werden. [Kna, Folie 22]

2.3 Folgern

Eine Aussage A folgt aus einer (möglicherweise unendlichen) Menge von Aussagen B, in Zeichen $B \models A$, wenn für jedes Modell von B auch A wahr ist. Sei $B :=$

$\{B_1, \ldots, B_n\}$. Dann gilt:

$$B \models A$$
$$\Leftrightarrow ((B_1 \wedge \ldots \wedge B_n) \rightarrow A) \text{ ist eine Tautologie}$$
$$\Leftrightarrow (B_1 \wedge \ldots \wedge B_n \wedge \neg A) \text{ ist eine Kontradiktion}$$

[Kna11, Seite 5]

Sei B eine endliche Menge von Aussagen und A eine Aussage. $\vdash$ steht für die Ableitung nach einem Inferenzverfahren. Das Verfahren ist vollständig, wenn alle Folgerungen auch ableitbar sind. Weiterhin ist das Verfahren korrekt, wenn alle ableitbaren Aussagen auch Folgerungen sind. [Kna11, Seite 17] [Kna, Folie 13]

KAPITEL 3

Die Paramodulationsregel

Die Paramodulation ist nur eine Möglichkeit, die semantische Gleichheit syntaktisch verschiedener Terme zu realisieren. Deshalb wird zunächst im nächsten Abschnitt 3.1 kurz auf die verschiedenen Möglichkeiten eingegangen.

3.1 Möglichkeiten der Realisierung der Gleichheit

Nach [RW68, Seite 2f] gibt es drei Möglichkeiten, die Gleichheit zu realisieren:

1. Die Einführung einer Menge von Axiomen in PK1 für die Gleichheit. Dabei müssen die Axiome jede vorkommende Funktion und jedes vorkommende Prädikat berücksichtigen.
2. Die Einführung einer kleineren Menge von Axiomen für die Gleichheit in der Prädikatenlogik zweiter Stufe.
3. Die Einführung einer Inferenzmethode für die Gleichheit.

Die Möglichkeit 1 hat nach [RW68, Seite 6f] drei Nachteile. Der erste Nachteil sind die vielen Inferenzschritte, die nötig sind, um eine Gleichheit anzuwenden. Bei Möglichkeit 3 wird dagegen nur ein einziger Inferenzschritt benötigt. Ein weiterer Nachteil der Möglichkeit 1 sind die vielen generierten Zwischenergebnisse, die den Suchraum „verschmutzen“ [RW68, Seite 7, übersetzt ins Deutsche]. Der dritte und letzte genannte Nachteil ist, dass die Gleichheitsaxiome in das Programm (die Wissensbasis) aufgenommen werden müssen. Dieser Nachteil wird von Robinson und Wos aber als der geringste der drei Nachteile bezeichnet. Sie schreiben, dass die Axiome automatisch generiert werden können: „The clerical chore of writing them all down could be eliminated merely by incorporating into the theorem-prover a program to generate them. Alternatively they may be specified by means of a schema (we shall call this variation approach 1b), or in approach 2 by means of a few second-order axioms.“ [RW68, Seite 7]

Die Möglichkeit 2 hat nach [RW68, Seite 7] die ersten beiden Nachteile mit Möglichkeit 1 gemein: viele Inferenzschritte und ein Suchraum mit vielen wenig nützlichen Zwischenschritten. Einen weiteren Nachteil sieht der Autor dieser Arbeit im Verlassen des PK1, weil es nach [Kna, Folie 13] für die Prädikatenlogik höherer Stufe

als eins kein vollständiges Inferenzverfahren gibt.

In dieser Arbeit wird Möglichkeit 3 verwendet. Dabei wird neben der Resolution die Paramodulation als Inferenzmethode verwendet.

3.2 Syntax, Semantik und Anwendung

Die Paramodulationsregel erweitert die Resolution über HORN-Klauseln des PK1 um die Verarbeitung von Regeln der Form:

$$(a \equiv b) \leftarrow p_1 \wedge p_2 \wedge \ldots \wedge p_n$$

Dabei stehen a und b für Terme und p_1 bis p_n für Atomformeln des PK1. Darin vorkommende Variablen gelten bezüglich der gesamten Klausel als allquantifiziert. Die Semantik der Regel ist, dass die Terme a und b unter den Bedingungen p_1 bis p_n semantisch gleich sind. [Kna12, Seite 4f] [RW68, Seite 8]

Die Paramodulationsregel wird wie folgt angewendet: Seien

- die Hypothese:
$$H \equiv h_1 \wedge h_2 \wedge \ldots \wedge h_n$$
- eine von der Paramodulationsregel zu verarbeitende Regel:
$$(a \equiv b) \leftarrow p_1 \wedge p_2 \wedge \ldots \wedge p_m$$
- t ein Term in der Hypothese, ohne Beschränkung der Allgemeinheit sei t ein Term in h_i
- ϑ_1 und ϑ_2 Termersetzungen, so dass $\vartheta_1(a) = \vartheta_2(t)$
- h_i' die Atomformel, die durch das Ersetzen eines Vorkommens von t durch $\vartheta_1(b)$ in h_i entsteht.

Wird die Paramodulationsregel nun erfolgreich auf die Hypothese, genauer gesagt auf die Atomformel (Teilziel) h_i, angewendet, das heißt die Termersetzungen ϑ_1 und ϑ_2 werden gefunden, so lautet die neue Hypothese:

$$H' \equiv \vartheta_2(h_1) \wedge \ldots \wedge \vartheta_2(h_{i-1}) \wedge \vartheta_2(h_i') \wedge \vartheta_1(p_1) \wedge \ldots \wedge \vartheta_1(p_m) \wedge \vartheta_2(h_{i+1}) \wedge \ldots \wedge \vartheta_2(h_n)$$

[Kna12, Seite 5]

Da die Gleichheit symmetrisch ist, gilt obiges analog für den Fall, dass es Termersetzungen ϑ_1 und ϑ_2 gibt, so dass $\vartheta_1(b) = \vartheta_2(t)$ und h_i' das Prädikat ist, dass durch Ersetzen eines Vorkommens von t durch $\vartheta_1(a)$ in h_i entsteht.

Die Beschreibung „die Suche nach einem Term, der sich mit a unifizieren lässt, die Berechnung des allgemeinsten Unifikators ϑ und das Ersetzen des Terms durch $\vartheta(b)$“ ist sehr lang und umständlich. Sie wird deshalb in dieser Arbeit abgekürzt mit „a wird gesucht und durch b ersetzt“.

Eine die Gleichheit definierende Regel ohne Bedingungen ist ein Fakt und kann verkürzt aufgeschrieben werden:

$$(a \equiv b)$$

In jedem Paramodulationsschritt wird immer nur ein einziges Vorkommen ersetzt. Mehrere Ersetzungen sind durch mehrfache Anwendung der Regel möglich. [Geo, Paramodulation]

Um die Paramodulation bei der Resolution zu verwenden, wird der Satz von Robinson so erweitert, dass nicht nur die Resolution, sondern auch die Paramodulation als Inferenzschritt erlaubt sind. Auch mit dem erweiterten Satz ist eine Menge von Klauseln genau dann kontradiktorisch, wenn die leere Klausel $false \leftarrow true$ abgeleitet werden kann. [Kna12, Seite 5]

Nach [Sá10] ist die Paramodulation mit Resolution vollständig und korrekt.

3.3 Paramodulation und die Gleichheitsaxiome

Die Gleichheitsaxiome (Reflexivität, Symmetrie, Transitivität, Funktions- und Prädikatsubstitutivität; siehe [San05, Seite 1] und [Bar77, Seite 29]) können mit der Paramodulationsregel erreicht werden.

- Reflexivität:

  $$(a \equiv a)$$

  Die Reflexivität wird erreicht, in dem die Paramodulationsregel nicht angewendet wird. Damit bleiben alle Terme unverändert. Bei variablenbehafteten Termen wird die Reflexivität gemeinsamer Instanzen durch den Unifikationsalgorithmus umgesetzt.

- Symmetrie:

  $$(a \equiv b) \Rightarrow (b \equiv a)$$

  Die Symmetrie wird durch die Implementierung der Paramodulationsregel erreicht. Wie bereits oben ausgeführt, kann sowohl ein Vorkommen von a durch b ersetzt werden, als auch ein Vorkommen von b durch a.

- Transitivität:

  $$(a \equiv b \wedge b \equiv c) \Rightarrow (a \equiv c)$$

  Die Transitivität wird durch die mehrfache Anwendung der Paramodulationsregeln erreicht. Seien folgende Regeln explizit gegeben:

  $$(a \equiv b)$$

  $$(b \equiv c)$$

  Die Regel

  $$(a \equiv c)$$

muss dann nicht explizit aufgeschrieben werden, weil sie durch die Hintereinanderanwendung der obigen Regeln der Paramodulation implizit folgt. Wenn a gegeben ist, kann es durch b ersetzt werden, welches wiederum durch c ersetzt werden kann. Analog gilt dies für die Ersetzung von c durch a.

- Funktionssubstitutivität:

$$(a_1 \equiv b_1 \wedge \ldots \wedge a_n \equiv b_n) \Rightarrow (f(a_1, \ldots, a_n) \equiv f(b_1, \ldots, b_n))$$

(f ist eine Funktion.)
Die Paramodulation ersetzt einen beliebigen Term. Insbesondere wird nicht nur die Funktion als ganzes versucht zu ersetzen, sondern auch jedes Argument einzeln.

- Prädikatsubstitutivität:

$$(a_1 \equiv b_1 \wedge \ldots \wedge a_n \equiv b_n) \Rightarrow (p(a_1, \ldots, a_n) \leftrightarrow p(b_1, \ldots, b_n))$$

(p ist ein Prädikat)
Die Paramodulation ersetzt Terme innerhalb von Prädikaten.

3.4 Einschränkungen der Paramodulation

Die Einschränkungen basieren auf der Idee, bestimmte Paramodulationen zu verbieten. Damit wird der Suchraum eingeschränkt.

3.4.1 Geordnete Paramodulation

Bei der geordneten Paramodulation wird eine Reduktionsordnung $\succ$ eingeführt [San05, Seite 6] [NR99, Seite 4f]. Nach [NR99, Seite 12] ist die Ordnung $\succ$ strikt und partiell. Weiterhin ist eine Ordnung $\succ$ auf Terme abgeschlossen unter Termersetzungen, das heißt $a \succ b \Rightarrow \vartheta(a) \succ \vartheta(b)$.

Außerdem erfüllt sie die Teilterm-Eigenschaft: $a[b]_p \succ b$ gilt für alle Terme a und b und $p \neq \varepsilon$. p ist eine Sequenz von Ganzzahlen, die eine Position im Term angibt. ε ist die leere Sequenz. $a[b]_p$ ist der Term, der entsteht, wenn der Teilterm von a an Position p durch den Term b ersetzt wird. Weil $p \neq \varepsilon$, wird nicht der gesamte Term a durch b ersetzt. Insbesondere ist b ein Teilterm von $a[b]_p$. Mit anderen Worten bedeutet die Teilterm-Eigenschaft, dass ein Term a, von dem b ein echter Teilterm ist, größer ist bezüglich $\succ$ als b.

Es soll a durch b ersetzt werden. Der zu ersetzende Term im aktuellen Ziel sei t. ϑ sei der allgemeinste Unifikator von t und a. In [San05, Seite 6] wird die Ersetzung nur ausgeführt, wenn $\vartheta(b) \not\succeq \vartheta(a)$. Die Termersetzung wird also nur dann ausgeführt, wenn der neue Term nicht größer ist bezüglich $\succ$ als der vorhandene und die beiden Terme nicht syntaktisch gleich sind.

In [NR99, Seite 4f] wird eine Termersetzung dagegen nur dann ausgeführt, wenn der neue Term kleiner ist bezüglich $\succ$ als der vorhandene. Dadurch terminiert die Paramodulation nach endlich vielen Schritten, wenn ein kleinster Term erreicht wird. Der kleinste Term wird auch als Normalform bezeichnet.

Der Aufbau einer totalen Ordnungsrelation scheint sehr aufwendig, weil während der Abarbeitung des Prolog-Programms potenziell unendlich viele Terme auftreten können. Dagegen kann es bei einer partiellen Ordnungsrelation Terme geben, die nicht miteinander in Relation stehen.

Die Paramodulation wird nur auf das aktuelle Ziel angewendet, nicht auf die Programmklauseln. Um die Unifikation bei der Resolution nicht zu verhindern, sollten deswegen beim Einsatz der geordneten Paramodulation die Terme in den Köpfen der Programmklauseln möglichst in Normalform sein.

3.4.2 Gerichtete Paramodulation

In [FHS89, Seite 329] wird die gerichtete Paramodulation beschrieben. Die normale, in Abschnitt 3.2 beschriebene Paramodulation, lässt Termersetzungen von links-nach-rechts und von rechts-nach-links zu. Bei der gerichteten Paramodulation ist ausschließlich eine Richtung der Termersetzung erlaubt. Die gerichtete Paramodulation ist korrekt, aber nicht mehr vollständig.

3.5 Erwartete Probleme bei der Implementierung

Die Gleichheit ist, wie in Abschnitt 3.2 bereits erwähnt, symmetrisch. Dadurch kann immer wieder die selbe Regel der Paramodulation angewendet werden, was der Interpreter auch tun würde. Das Programm würde nie terminieren. Weiterhin würde, wenn die Regel Bedingungen enthält, die Hypothese expandieren, bis kein weiterer Speicher mehr angefordert werden kann und der Interpreter deswegen abstürzt.

Ein weiteres Problem besteht, wenn in einer Regel der Art

$$(a \equiv b)$$

(mit oder ohne Bedingungen ist für dieses Problem irrelevant) a ein echter Teilterm von b ist beziehungsweise b ein echter Teilterm von a. Sei a ein echter Teilterm von b. Ein Vorkommen von a wird durch die Paramodulation durch den größeren Term b ersetzt. Im nächsten Schritt wird a in b gefunden und wieder durch b ersetzt. Dadurch expandiert der Term. Ein Beispiel zur Veranschaulichung:

$(a \equiv f(a))$.
$? - p(a)$.

Der Interpreter würde nun diese unendliche Ableitungsfolge berechnen:

$? - p(a).$
$? - p(f(a)).$
$? - p(f(f(a))).$
$? - p(f(f(f(a)))).$
$\ldots$

Wie die Probleme in dieser Arbeit gelöst wurden, wird in Kapitel 5 beschrieben.

KAPITEL 4

Das Deduktionstool

Nach Aufgabenstellung soll ein Deduktionstool implementiert werden. Dabei wurde eine Prolog-ähnliche Syntax gewählt. Der Leser wird sich sicher fragen, wieso kein Open Source Prolog verwendet und erweitert wurde. Im Vorfeld der Arbeit wurde kein einfaches, leicht zu änderndes und gut dokumentiertes Prolog gefunden. Weiterhin wären umfangreiche Anpassungen notwendig gewesen. Der Parser muss die Regeln der Paramodulation akzeptieren und sie müssen in einer vom Interpreter nutzbaren Form gespeichert werden.

Eventuell vorhandene Optimierungen erschweren die Speicherung zusätzlich. Die erste Idee des Autors für die Speicherung der Wissensbasis war, die Regeln nach dem Namen und der Arität der Atomformel im Kopf zu sortieren. Die Regeln der Paramodulation können aber nicht nur auf eine bestimmte Atomformel angewendet werden, sondern potenziell auf die Terme jeder Atomformel. Die Regeln der Paramodulation passen deshalb nicht in das Sortierungsschema.

SWI-Prolog benutzt beispielsweise Klausel-Indizierung. Die Indizes werden zur effizienteren Suche nach anwendbaren Regeln verwendet. Die Auswahl der Regeln erfolgt dann nach einem bestimmten Prozess [SWI, 2.17 Just-in-time clause indexing]. Die Regeln der Paramodulation müssten dann so in die Indizierung und den Auswahlprozess eingepasst werden, dass sie trotzdem für jede Atomformel angewendet werden.

Außerdem müssen für eine Optimierung die nach jedem Resolutions- oder Paramodulationsschritt entstandenen neuen Ziele (Hypothesen und konstruierte Lösungen) gespeichert werden. Nach jedem Inferenzschritt muss das neue aktuelle Ziel dann mit den bisherigen Zielen verglichen werden. Die Optimierung ist für die Paramodulation im zu implementierenden Deduktionstool notwendig, um Berechnungskreise zu erkennen, und wurde mit dem Betreuer abgesprochen. Sie wird in Abschnitt 4.6.2 genauer erklärt.

Schließlich muss bei der Inferenz auch beachtet werden, dass die Regeln für die Paramodulation anders verarbeitet werden als „normale“ Regeln und auch Daten auf dem Backtrack-Stack speichern.

4.1 Arbeitsweise von Prolog

Bevor es in den nächsten Abschnitten dieses Kapitels um das zu implementierende Deduktionstool geht, wird in diesem Abschnitt kurz auf die Arbeitsweise von Prolog eingegangen. Sie ist die Grundlage für das Deduktionstool.

In [Col93, Kapitel 3, Seite 12 - 17] wird der Vorläufer von Prolog beschrieben. Am Anfang von [Col93, Kapitel 4, Seite 17 - 23] werden dann die wesentlichen Unterschiede in der Arbeitsweise vom Vorläufer und dem finalen Prolog beschrieben. Dieser Abschnitt bezieht sich nur auf [Col93, Kapitel 3, Seite 12 - 17], wenn die Arbeitsweise in das finale Prolog übernommen wurde.

Prolog verwendet die Resolution nach Robinson. Die Klauseln werden in zwei Mengen eingeteilt: dem Programm und die Hypothesen (Fragen). Die Atomformeln der Klauseln im Klauselkörper werden von rechts nach links geordnet. Die Resolution wird nur zwischen der ersten Atomformel des Resolventen und dem Kopf einer Programmklausel ausgeführt. Die Auswahl der Klauseln wird in der Reihenfolge vorgenommen, in der sie der Programmierer aufgeschrieben hat. Die Resolvente ist die aktuelle Liste von Teilzielen. [Col93, Seite 13]

Der Nichtdeterminismus wird durch Backtrack behandelt. Durch den Backtrack wird die Vollständigkeit verloren, weil es unendliche Ableitungspfade geben kann. Dadurch kann ein eventuell existierender endlicher Ableitungspfad, der zur leeren Hypothese führt, nicht gefunden werden. Die Entwickler von Prolog sehen den Programmierer dafür Verantwortlich, sicherzustellen, dass das Programm terminiert [Col93, Seite 13f]. Zur manuellen Einschränkung des Backtracks gibt es im finalen Prolog einen Operator: den „search space cut operator “!”“ [Col93, Seite 17].

Backtrack und das Ordnen der Klauseln sind in Prolog die Basistechniken für den Umgang mit Nichtdeterminismus: „Backtracking and ordering the set of clauses defining a predicate were the basic elements retained as a technique for managing non-determinism.“ [Col93, Seite 18] Beide Techniken und die Resolution werden auch im Deduktionstool, welches im Rahmen dieser Arbeit implementiert wird, eingesetzt.

4.2 Variablenfremdheit

Zwei Variablen, die den selben Namen besitzen, müssen nicht zwangsläufig auch die selben Variablen sein. In Prolog sind gleich benannte Variablen in unterschiedlichen Klauseln unterschiedliche Variablen, weil sie für jede Klausel separat allquantifiziert sind. Prolog nimmt eine automatische Variablenumbenennung vor, damit die anzuwendende Programmklausel und das aktuelle Ziel variablenfremd sind [hmi, Seite 96]. Die Resolution wird immer nur auf eine Programmklausel und das aktuelle Ziel angewendet. Deshalb ist es ausreichend, wenn jede Programmklausel und das aktuelle Ziel variablenfremd sind. Die Programmklauseln untereinander müssen nicht variablenfremd sein.

Um unterschiedliche Variablen mit dem selben Namen unterschieden zu können, können sie zusätzlich zu ihrem Namen mit einer Nummer versehen werden. Wenn das Deduktionstool nun dafür sorgt, dass die Variablen in den Programmklauseln immer andere Nummern haben, als die Variablen im aktuellen Ziel, ist die Variablenfremdheit immer gewährleistet. Das zu implementierende Deduktionstool wird mit dieser Idee die Variablenfremdheit zwischen dem aktuellen Ziel und den Programmklauseln gewährleisten. Dabei wird in jedem Inferenzschritt ein Zähler inkrementiert. Alle neu zum Ziel hinzukommenden Variablen erhalten die neue Nummer.

4.3 Unifikation und Verkettung von Termersetzungen

Für die Resolution wird nach [Rob65, Seite 35] die Unifikation verwendet. Der Unifikationsalgorithmus, der in dieser Arbeit genutzt wird, ist aus [Baa01, Seite 447f] entnommen und wurde vom Autor angepasst. Für die Unifikation wird die Komposition von Termersetzungen benötigt. Der dafür notwendige Algorithmus wurde aus [Baa01, Seite 445] entnommen und auch angepasst.

Der Unifikationsalgorithmus aus [Baa01, Seite 447f] (auch zu finden im Anhang A) ist nicht fehlerfrei. Seien zum Beispiel

$$s = f(nil, Ls1)$$

$$t = f(Ls0, g(Sym, Ls0))$$

Im ersten Schritt erkennt der Algorithmus korrekt, dass die beiden Terme strukturierte Terme mit dem gleichen Funktionsnamen und der gleichen Arität sind. Danach werden die Terme nil und $Ls0$ verarbeitet. $Ls0$ ist eine Variable, weshalb die aktuellen Termersetzungen auf sie angewandt werden. Die Menge der Termersetzungen ϑ ist noch die leere Menge. Die Unifikation der Terme ergibt die Termersetzungen

$$\vartheta = \{[Ls0, nil]\}$$

Im nächsten Schritt werden die Terme $Ls1$ und $g(Sym, Ls0)$ verarbeitet. $Ls1$ ist eine Variable. Die Anwendung der aktuellen Termersetzungen ergibt $\vartheta(Ls1) = Ls1$. $g(Sym, Ls0)$ ist keine Variable, weshalb auch keine Termersetzungen auf den Term angewendet werden. Die Unifikation ergibt, zusammen mit der vorher errechneten Termersetzung, die Termersetzungen

$$\vartheta = \{[Ls0, nil], [Ls1, g(Sym, Ls0)]\}$$

Das Einsetzen der Termersetzungen in die beiden Terme führt dann zu:

$$\vartheta(f(nil, Ls1)) = f(nil, g(Sym, Ls0))$$

$$\neq f(nil, g(Sym, nil)) = \vartheta(f(Ls0, g(Sym, Ls0)))$$

Der korrekte Unifikator wäre jedoch

$$\vartheta = \{[Ls0, nil], [Ls1, g(Sym, nil)]\}$$

und führt zu

$$\vartheta(s) = \vartheta(t) = f(nil, g(Sym, nil))$$

Der Fehler liegt in den ersten beiden IF-Anweisungen. Die Termersetzungen dürfen nicht nur auf die beiden Terme angewendet werden, wenn sie Variablen sind. Sie müssen auch dann angewendet werden, wenn Variablen in ihnen vorkommen. Das wird auch in [Baa01, Seite 447] so beschrieben: „Otherwise, write down "$x \mapsto t$" as part of the solution, replace x everywhere by t (including in the solution)[...]" Die Termersetzungen können auch problemlos immer auf die Terme angewendet werden, weil sie Terme ohne Variablen nicht ändern.

Neben der Fehlerkorrektur wurde der Algorithmus so erweitert, dass er auch anonyme Variablen unterstützt. Außerdem wurde die in [Baa01, Seite 448] vorgeschlagene Optimierung eingesetzt, die beiden Terme direkt zu tauschen statt dies durch einen weiteren Selbstaufruf mit vertauschten Parametern zu realisieren.

Der in dieser Arbeit verwendete Algorithmus ist:

```
global ϑ: Termersetzungen (initialisiert mit ∅)

bool Unifikation(s: Term, t: Term)
    variablen
        unif: bool
        i: integer
    begin
        s := ϑ(s)
        t := ϑ(t)
        wenn ((s keine Variable) und (t ist Variable))
            dann
                tausche s und t
        wenn ((s = t)
              oder (s ist anonyme Variable)
              oder (t ist anonyme Variable))
            dann
                rückgabe true
            sonst wenn ((s keine Variable) und (t keine Variable))
                unif := false
                wenn ((s ist f(s_0, ..., s_{n-1}))
                      und (t ist f(t_0, ..., t_{n-1})))
                    dann
                        unif := true
                        i := 0
                        solange (unif und (i < n))
                            unif := Unifikation(s_i, t_i)
                            i := i + 1
                rückgabe unif
```

```
            sonst wenn (s ist Teilterm von t)
                rückgabe false
            sonst
                ϑ := {[s, t]} ∘ ϑ
                rückgabe true
    ende
```

Für den obigen Algorithmus fehlt noch ein Algorithmus für die Verkettung von zwei Termersetzungen. In [Baa01, Seite 445] wird ein solcher Algorithmus vorgeschlagen. Der Algorithmus ist für die Verkettung von beliebigen Termersetzungen geeignet. Der Unifikationsalgorithmus benötigt aber nur die Verkettung von Termersetzungen mit nur einer Termersetzung mit beliebigen Termersetzungen. Deshalb kann der Algorithmus aus [Baa01, Seite 445] für diese Arbeit vereinfacht werden.

Der in dieser Arbeit verwendete Algorithmus ist:

```
Termersetzungen Verkettung(ϑ: Termersetzungen,
                           [V, t]: Termersetzung)
    begin
        für alle [V_i, t_i] ∈ ϑ
            ersetzte [V_i, t_i] durch [V_i, {[V, t]}(t_i)]
        lösche aus ϑ alle [V_i, t_i] mit V_i = t_i
        wenn (nicht ∃ t_i: [V, t_i] ∈ ϑ)
            ϑ := ϑ ∪ [V, t]
        rückgabe ϑ
    ende
```

Sobald eine Variable ersetzt wird, werden alle weiteren Vorkommen dieser Variable auch ersetzt. Weiterhin wird durch Zeile 5 im Algorithmus für die Verkettung von Termersetzungen sicher gestellt, dass die ersetzte Variable nicht durch eine andere Termersetzung wieder auftaucht. Für Termersetzungen ϑ, die mit den obigen beiden Algorithmen erstellt wurden, gilt dann: ϑ ersetzt eine Variable nicht duch einen Term, der eine Variable enthält, die auch von ϑ ersetzt wird.

Im zu implementierenden Deduktionstool kommen nur durch die obigen beiden Algorithmen erstellten Termersetzungen vor.

4.4 Syntax des Deduktionstools

Wie bereits erwähnt ist die Syntax ähnlich deren von Prolog. Die Syntax lässt sich wie folgt mit Backus-Naur-Form und regulären Ausdrücken darstellen:

```
<programm>      ::= ε | <klausel> . <programm>
<klausel>       ::= <regel> | <hypothese>
<hypothese>     ::= ?– <atomformelliste>
```

```
<regel>              ::= <atomformel_rh>
                       | <atomformel_rh> := <atomformelliste>
<atomformelliste>    ::= <atomformel_rb>
                       | <atomformel_rb> , <atomformelliste>
<atomformel_rh>      ::= <atomformel> | <rim>
<atomformel_rb>      ::= <atomformel> | <sep> | !
<rim>                ::= <atomformel>
<sep>                ::= <atomformel>
<atomformel>         ::= <name> ( <termliste_e> )
<termliste_e>        ::= ε | <termliste>
<termliste>          ::= <term> | <term> , <termliste>
<term>               ::= <atom> | <func> | <number> | <variable>
<atom>               ::= <name>
<func>               ::= <name> ( <termliste> )
<number>             ::= <zahl>
<variable>           ::= <var>
<name>               ::= <name1> | <name2>
<name1>              ::= [a-zäöüß][a-zäöüßA-ZÄÖÜ_]*
<name2>              ::= "([^\\"]|(\\.))*"
<zahl>               ::= -?[0-9]+(\.[0-9]+)?
<var>                ::= [A-ZÄÖÜ_][a-zäöüßA-ZÄÖÜ_]*
```

Als Zeichensatz für den Quelltext wird UTF-8 verwendet. Die Grundmenge für die obigen regulären Ausdrücke ist deshalb die Menge aller Unicode-Zeichen. Die Syntax und Semantik der regulären Ausdrücke ist aus [Wikb] entnommen.

Der Punkt . soll dabei für ein beliebiges (Unicode-)Zeichen inklusive dem Zeilenumbruch stehen.

In $< name2 >$ ist der Backslash \ das Escape-Zeichen, um auch Anführungszeichen " in einem Namen benutzen zu können.

Als Kommentarzeichen sind % für einzeilige und $/ * \ldots * /$ für mehrzeilige Kommentare erlaubt.

$< sep >$ steht für Self Evaluating Predicate und $< rim >$ für Rule with own Inference Method. Die beiden Konstrukte werden für die Erweiterung des Deduktionstools mittels Plug-ins verwendet und im folgenden Abschnitt 4.5 näher behandelt.

4.5 Plug-ins

Durch Plug-ins kann der Funktionsumfang erweitert werden. Die Erweiterung kann mittels Prolog-Code, Self Evaluating Predicates, Rules with own Inference Method, Prä- und Postprozessoren erfolgen.

4.5.1 Self Evaluating Predicate (SEP)

SEPs sind Prädikate, die nicht durch die Anwendung der Resolution ausgewertet werden, sondern eine eigene Methode dafür mitbringen. Innerhalb dieser Methode kann jeglicher Code ausgeführt werden. Nebeneffekte, wie zum Beispiel Ausgaben auf dem Bildschirm, können durch SEPs erreicht werden. Durch den Zugriff auf die Regelbasis sind weitreichende Nebeneffekte möglich.

4.5.2 Rule with own Inference Method (RIM)

Bei Regeln wird versucht die Atomformel im Klauselkopf der Regel mit der ersten Atomformel der Hypothese zu unifizieren. Bei Erfolg werden die notwendigen Termersetzungen durchgeführt und die Atomformel der Hypothese durch den Körper der Regel ersetzt. Dabei werden die Unifikation und die Resolution eingesetzt.

RIMs sind Regeln, die nicht die Standard-Inferenzmethode, im Fall des Deduktionstools dieser Arbeit die Resolution, einsetzen, sondern ein (möglicherweise gänzlich) anderes Verfahren. Die Regel bestimmt dabei selber, ob sie auf ein bestimmtes Teilziel angewendet werden kann und wie. Insbesondere gilt das auch für die notwendigen Ersetzungen und neuen Teilziele.

Die Paramodulation wurde im zu implementierenden Deduktionstool mit einer RIM realisiert.

Die RIMs werden gleichberechtigt zu den „normalen“ Regeln in der Reihenfolge angewendet, in der sie aufgeschrieben wurden. Es gibt keine Priorisierung von „normalen“ Regeln oder RIMs.

4.5.3 Weitere Möglichkeiten von Plugins

Plug-ins können eigene Prä- und Postprozessoren mitbringen. Präprozessoren führen eine Vorverarbeitung des Prolog-Codes aus, bevor der eigentliche Parser ihn erhält. Postprozessoren führen eine Nachbearbeitung der Ausgaben des Prolog-Interpreters aus, bevor der Benutzer sie sieht.

Weiterhin können Plug-ins eigenen Prolog-Code mitbringen, der automatisch in jedes Prolog-Programm eingebunden wird. Die Regeln werden dabei aus einer Prolog-Quelldatei eingelesen und vor den Regeln des auszuführenden Prolog-Programms in die Regelbasis eingefügt.

4.6 Optimierungen

Die folgenden Optimierungen sind notwendig, damit Prolog-Programme, die die Paramodulation nutzen, terminieren können.

4.6.1 Eliminierung von Teilzielen

Im Ziel (der Hypothese) werden das Idempotenz- und das Kommutativitätsgesetz der Konjunktion angewendet. Wird ein Teilziel der Hypothese hinzugefügt, so wird erst überprüft, ob ein gleiches bereits vorhanden ist. Wenn kein gleiches Teilziel bereits im Ziel vorhanden ist, wird das neue Teilziel hinzugefügt, andernfalls nicht.

Die Kommutativität der Konjunktion wird benötigt, um die beiden gleichen Teilziele so in der Hypothese zu verschieben, dass sie direkt nebeneinander stehen. Durch die Idempotenz der Konjunktion kann dann eines der beiden gleichen Teilziele weggelassen werden.

Die Subsumption ist für diese Optimierung auch denkbar und sogar besser geeignet, weil sie allgemeiner ist. Allerdings ist sie für den Rahmen dieser Arbeit zu aufwendig zu implementieren.

Zur Verdeutlichung der Optimierung soll ein Beispiel dienen, bei der einer Hypothese Schritt für Schritt neue Teilziele hinzugefügt werden:

1. Das aktuelle Ziel ist:

$$?- nachbarn(herr\_mueller, frau\_schmidt).$$

2. Das Teilziel $nachbarn(X, Y)$ soll hinzugefügt werden. Es wird hinzugefügt, weil noch kein Gleiches vorhanden ist. Das neue Ziel lautet:

$$?- nachbarn(X, Y), nachbarn(herr\_mueller, frau\_schmidt).$$

3. Danach soll das Teilziel $nachbarn(herr\_mueller, frau\_schmidt)$ hinzugefügt werden. Es wird nicht hinzugefügt, weil ein gleiches Teilziel bereits vorhanden ist. Das Ziel bleibt unverändert:

$$?- nachbarn(X, Y), nachbarn(herr\_mueller, frau\_schmidt).$$

4. Dann soll das Teilziel $nachbarn(A, B)$ hinzugefügt werden. Es wird hinzugefügt, weil noch kein Gleiches vorhanden ist. Bei der Überprüfung auf Gleichheit der Teilziele beziehungsweise deren Terme reichen unterschiedliche Variablennamen aus, um als ungleich erkannt zu werden. Es findet keine Variablenumbenennung bei der Überprüfung statt. Das neue Ziel lautet:

$$\begin{aligned} ?- \quad & nachbarn(A, B), nachbarn(X, Y), \\ & nachbarn(herr\_mueller, frau\_schmidt). \end{aligned}$$

5. Als nächstes soll das Teilziel $nachbarn(X, Y)$ hinzugefügt werden. Es wird nicht hinzugefügt, weil ein gleiches Teilziel bereits vorhanden ist. Das Ziel bleibt unverändert:

   $?-\quad nachbarn(A, B), nachbarn(X, Y),$
   $nachbarn(herr\_mueller, frau\_schmidt).$

Auf den ersten Blick wirkt die Ausnutzung der Idempotenz und der Kommutativität einleuchtend. Doch bei genauerer Betrachtung fällt auf, dass es nicht nur „normale“ Prädikate gibt, sondern auch Prädikate mit Nebeneffekten, wie ! und *write*. Prädikate mit Nebeneffekten sind nicht kommutativ, weil ihre Ausführung nur unter bestimmten Bedingungen erwünscht sein kann, die vor dem entsprechenden Prädikat ausgewertet werden müssen.

Falls zwischen zwei Teilzielen keine Prädikate mit Nebeneffekten stehen, können die Kommutativität und die Idempotenz beziehungsweise auch die (in dieser Arbeit nicht verwendete) Subsumption immer ausgenutzt werden.

Das folgende einfache Beispiel zeigt einen Fall, in welchem das Prolog-Programm durch die Optimierung fehlerhaft arbeitet:

$input(2, 2, 1).$
$check(A, B, C) :- add(A, B, C), write(A, \text{“}=\text{”}, B, \text{“}+\text{”}, C).$
$?- input(X, Y, Z), check(X, Y, Z), add(2, 2, 1).$

$add(A, B, C)$ ist hierbei ein eingebautes Prädikat, welches das Tripel $[A, B, C]$ genau dann auf den Wahrheitswert *wahr* abbildet, wenn $A = B + C$. Das $input(2, 2, 1)$ soll dabei eine Nutzereingabe simulieren. Der Programmierer wird erwarten, dass die Ausgabe (mit eingesetzten Zahlen statt der Variablen)

$$A = B + C$$

nur erscheint, wenn die Gleichung auch wahr ist. Das wird auch durch das *add* vor dem *write* sicher gestellt. Der Ablauf sieht folgendermaßen aus:

1. Das aktuelle Ziel ist:

   $$?- input(X, Y, Z), check(X, Y, Z), add(2, 2, 1).$$

2. Das Teilziel $input(X, Y, Z)$ wird ausgewertet. Dabei gibt es die Termersetzungen $\{[X, 2], [Y, 2], [Z, 1]\}$. Das neue Ziel ist:

   $$?- check(2, 2, 1), add(2, 2, 1).$$

3. Das Teilziel $check(2, 2, 1)$ wird ausgewertet. Dabei gibt es die Termersetzungen $\{[A, 2], [B, 2], [C, 1]\}$. Die Teilziele $add(2, 2, 1)$ und $write(2, \text{“}=\text{”}, 2, \text{“}+\text{”}, 1)$ sollen der Hypothese hinzugefügt werden. Da $add(2, 2, 1)$ aber bereits im Ziel vorhanden ist, wird es nicht hinzugefügt. Das neue Ziel ist:

   $$?- write(2, \text{“}=\text{”}, 2, \text{“}+\text{”}, 1), add(2, 2, 1).$$

4. Das Teilziel $write(2, " = ", 2, " + ", 1)$ wird ausgewertet, wobei $2 = 2 + 1$ ausgegeben wird. Die Aussage ist falsch. Das neue Teilziel ist:

$$? - add(2, 2, 1).$$

5. Zuletzt wird das Teilziel $add(2, 2, 1)$ ausgewertet. Da $2 \neq 2 + 1$, ergibt die Auswertung false und es wird, mangels weiterer Ableitungsalternativen, $No.$ ausgegeben.

Das Ausnutzen der Kommutativität und der Idempotenz muss so angepasst werden, dass die Bedingungen für Prädikate mit Nebeneffekten immer vor deren Aufruf stehen.

4.6.2 Speichern der Historie bereits erreichter Ziele

Das Speichern bereits erreichter Ziele wird benötigt, um sich wiederholende Zyklen im Ableitungspfad zu erkennen. Wird ein bereits erreichtes Ziel, dazu gehört auch das ursprüngliche Ziel im Prolog-Programm, nochmals erreicht, so gibt es zwei Möglichkeiten: das aktuelle Ziel wurde bereits auf einem anderen Ableitungspfad erreicht oder es gibt eine Schleife im Ableitungspfad. Es ist in beiden Fällen nicht sinnvoll vom aktuellen Ziel ausgehend weiter abzuleiten. Dann würde entweder ein bereits vorher abgeleitetes Ziel nochmals abgeleitet werden, was keinen Erkenntnisgewinn darstellt, oder die Ableitung in einer Schleife stecken bleiben, die zu einem nicht terminierenden Programm führt. Derartige Schleifen führen im Allgemeinen auch zum Absturz des Programms aufgrund mangelndem Speicherplatzes, weil alternative Ableitungswege auf dem Backtrack-Stack abgelegt werden, welcher dann irgendwann überläuft.

Nach jedem Resolutionsschritt muss das Deduktionstool prüfen, ob das neue Ziel bereits vorher abgeleitet wurde. Wenn dies der Fall ist, führt es einen Backtrack aus. Ist das neue Ziel noch nicht vorher aufgetreten, wird es in die Liste der bereits gesehenen Ziele aufgenommen und ausgehend von dem Ziel weiter abgeleitet.

Ein Beispiel, bei der die Vollständigkeit der Resolution nach Robinson wegen der Implementierung mittels Tiefensuche mit Backtrack nicht mehr gegeben ist, ist die Kommutativität:

$nachbarn(A, B) : -nachbarn(B, A).$
$nachbarn(herr\_mueller, frau\_schmidt).$
$? - nachbarn(frau\_schmidt, herr\_mueller).$

Normalerweise wird immer wieder die erste Regel angewendet, so dass abwechselnd immer wieder die beiden Ziele

$$? - nachbarn(frau\_schmidt, herr\_mueller).$$

und

$$? - nachbarn(herr\_mueller, frau\_schmidt).$$

abgeleitet werden.

Mit der Speicherung der Historie bereits erreichter Ziele wird die Schleife beendet, sobald die Hypothese

$$? - nachbarn(frau\_schmidt, herr\_mueller).$$

erneut abgeleitet wurde. Es wird dann ein Backtrack ausgeführt und die zweite Klausel kommt zur Anwendung.

Durch Vertauschen der ersten und zweiten Klausel würde Prolog auch ohne die Optimierung die richtige Lösung finden, aber unendlich oft immer wieder die selbe.

Die Speicherung kann auf zwei Arten erfolgen:

1. Alle bereits erreichten Ziele werden gespeichert. Das ist etwas einfacher zu implementieren und bietet mehr Optimierungsmöglichkeiten bei der Ableitung. Dafür wird aber mehr Speicher benötigt.
2. Nur die im aktuellen Ableitungspfad bereits erreichten Ziele werden gespeichert. Bei einem Backtrack werden auch die erreichten Ziele gelöscht, die nun nicht mehr auf dem aktuellen Ableitungspfad liegen. Es wird weniger Speicherplatz benötigt. Dafür ist der Implementierungsaufwand höher und es werden nicht alle Optimierungsmöglichkeiten ausgeschöpft.

Grundsätzlich gibt es bei der Speicherung der Historie zwei Nachteile. Der offensichtliche Nachteil ist der Speicherverbrauch. Weiterhin kann, je nach Implementierungsweise, die Ausführung eines Inferenzschrittes immer langsamer werden, weil am Ende jedes Schrittes das errechnete neue Ziel mit den bisherigen verglichen wird. Da die Effizenz des zu implementierenden Deduktionstools nach Aufgabenstellung aber im Hintergrund steht, ist dieses negative Verhalten für die Arbeit nicht relevant. Wichtig für die Arbeit ist vor allem das Stoppen von Ableitungsschleifen.

In [Sti88, Seite 360] wird, allerdings für eine andere Aufgabenstellung, vorgeschlagen, die bereits erreichten Ziele nicht zu speichern, sondern immer wieder neu zu berechnen. Die Methode kann auch hier als Alternative zur sehr speicherintensiven Speicherung der Ziele eingesetzt werden. Offensichtlich ist der große Vorteil, dass erheblich weniger Speicher benötigt wird. Ebenfalls offensichtlich ist der Nachteil, dass die Laufzeit erhöht wird. Es sei n die Anzahl der für das erreichen der Lösung berechneten Ziele. Dann müssen für die Lösung n Ziele berechnet werden. Weiterhin muss das erste Ziel mit 0 vorhergehenden verglichen werden, das zweite mit 1, das dritte mit 2 bis zum n-ten Ziel, das mit $n-1$ vorhergehenden Zielen verglichen werden muss. Die vorhergehenden Ziele müssen immer wieder berechnet werden. Daraus ergibt sich eine Laufzeit von

$$0 + 1 + \ldots + (n-1) + n = \frac{n(n-1)}{2} = O(n^2)$$

Ein weiterer Nachteil kommt zum Vorschein, wenn interaktive Prolog-Programme ausgeführt werden. Dann müssen die Benutzereingaben zwischengespeichert und an den entsprechenden Stellen automatisch eingefügt werden.

Bei interaktiven Programmen gibt es ein weiteres Problem mit der Schleifenerkennung. Der Benutzer kann mehrfach die gleiche Eingabe tätigen. Dabei kann das gleiche Ziel mehrfach auftreten, so dass ein Backtrack ausgelöst wird. Der Benutzer kann aber beispielsweise wollen, dass das Programm bestimmte Schritte mehrfach ausführt und danach erst andere Berechnungen vornimmt. Benutzerintentionen, die sich einer logikbasierten Interpretation entziehen, wie zum Beispiel gewünschte Interaktionen, sind für diese Arbeit nicht von Interesse. Daher werden interaktive Prolog-Programme nicht betrachtet.

Für das Deduktionstool, das im Zusammenhang mit dieser Arbeit implementiert wird, hat der Autor in Absprache mit dem Betreuer die Variante der Speicherung der bereits abgeleiteten Ziele nach Art 1 gewählt. Damit wird eine gute Zeitkomplexität auf Kosten einer schlechten Speicherkomplexität erreicht. Weiterhin ist sie am einfachsten zu implementieren. Zwei Hypothesen werden außerdem auf exakte Gleichheit getestet. Das heißt, dass bereits eine Variablenumbenennung ausreicht, um zwei Hypothesen als verschieden zu erkennen. Ein Vergleich unter Berücksichtigung von Variablenumbenennung wäre für die Implementierung im Rahmen dieser Arbeit zu aufwendig.

KAPITEL 5

Implementierung der Paramodulation

5.1 Syntax

Die Syntax der Paramodulationsregel im Deduktionstool weicht von der im Kapitel 3 vorgestellten ab, weil es nach Abschnitt 4.4 keine Infixnotation unterstützt. Außerdem werden die in Prolog üblichen Notationen für die Inhibition ($:-$ statt $\leftarrow$) und Konjunktion ($,$ statt $\wedge$) verwendet. Weiterhin wird jede Klausel mit einem Punkt . abgeschlossen und die Allquantoren vor den Klauseln weggelassen.

Die Regeln (5.1) beziehungsweise die Fakten (5.2) haben die Form:

$$\text{"}=\text{"}(a, b) :- p_1, p_2, \ldots, p_n. \tag{5.1}$$

$$\text{"}=\text{"}(a, b). \tag{5.2}$$

Dabei stehen a und b für Terme und p_1 bis p_n für Atomformeln. Bei Regeln muss für n gelten: $n \geq 1$.

Ist a ein echter Teilterm von b oder umgekehrt b ein echter Teilterm von a, so wird nur der größere Term durch den kleineren ersetzt. Die Paramodulation wird nicht ausgeführt, wenn a und b syntaktisch gleich sind. Andernfalls kann die Paramodulation in beide Richtungen ausgeführt werden. Die Paramodulation entspricht also einer geordneten Paramodulation, in der Termersetzungen erlaubt sind, wenn der neue Term nicht größer ist bezüglich $\succ$ als der vorhandene Term und die Terme nicht syntaktisch gleich sind. $\succ$ ist dabei wie folgt definiert:

$$a \succ b \Leftrightarrow b \text{ ist ein echter Teilterm von } a$$

In dieser Arbeit wird die Paramodulation in beide Richtungen vereinfachend als ungeordnete Paramodulation bezeichnet. Mit der geordneten Paramodulation ist die Paramodulation gemeint, bei der die beiden Terme bezüglich $\succ$ in Relation stehen und damit die Paramodulation nur in eine Richtung erfolgt.

Weiterhin ist die gerichtete Paramodulation möglich. Sie wird durch ein drittes Argument im Kopf der Regel gekennzeichnet:

$$\text{"}=\text{"}(a, b, d) :- p_1, p_2, \ldots, p_n.$$

$$" = "(a, b, d).$$

d ist ein Atom mit dem Namen „d“ (directed). Bei der gerichteten Paramodulation wird nur b durch a ersetzt, aber nicht umgekehrt. Es findet, im Gegensatz zur oben dargestellten ungerichteten Paramodulation, keine Überprüfung statt, ob a ein Teilterm von b ist oder umgekehrt. Sollte b ein Teilterm von a sein, so wird der Term zum Programmlauf durch wiederholte Anwendung der Regel expandieren.

5.2 Einordnung der Paramodulationsregel in die Tiefensuche mit Backtrack

Die Regeln der Paramodulation werden gleichberechtigt zu anderen Regeln angewendet. Es erfolgt keine Priorisierung. Allein die Reihenfolge, in der die Regeln im Quelltext aufgeschrieben wurden, entscheidet über die Reihenfolge der Anwendung.

Bei der Paramodulation in beide Richtungen mit einem Klauselkopf $" = "(a, b)$ wird zuerst nach a gesucht und durch b ersetzt. Erst wenn die Suche nach a abgeschlossen ist, wird angefangen nach b zu suchen und durch a zu ersetzen. Bei der Paramodulation in nur eine Richtung (geordnete oder gerichtete Paramodulation) wird nur nach Ersetzungen in die entsprechende Richtung gesucht.

Die Paramodulation wird nur auf das erste Teilziel des aktuellen Ziels angewendet. Die Argumente werden dabei in der Reihenfolge ihres Auftretens von links nach rechts durchsucht. Das Durchsuchen erfolgt mittels Tiefensuche, also zuerst in die Tiefe. Bei Funktionen wird zuerst versucht die gesamte Funktion zu ersetzten, danach die einzelnen Argumente, wieder von links nach rechts.

Das folgende Beispiel verdeutlicht die Reihenfolge der Anwendung einer Regel der Paramodulation:

$" = "(*(X, Y), *(Y, X)).$
$? - p(*(a, *(b, c))).$

Die Ableitungsreihenfolge ist wie folgt:

1. Die Regel vertauscht die Argumente der äußeren Funktion $*$. Das neue Ziel ist:
$$? - p(*(*(b, c), a)).$$
2. Mit dem neuen Ziel aus Schritt 1 wird weitergerechnet, bis durch einen Backtrack die Anwendung der obigen Regel der Paramodulation auf das obige Ziel fortgesetzt wird.
3. Jetzt werden die Argumente der inneren Funktion $*$ vertauscht. Das neue Ziel ist:
$$? - p(*(a, *(c, b))).$$

4. Mit dem neuen Ziel aus Schritt 3 wird weitergerechnet, bis durch einen Backtrack die Anwendung der obigen Regel der Paramodulation auf das obige Ziel fortgesetzt wird.

5. Da keine weiteren Ersetzungen ausgeführt werden können, wird die Anwendung der Regel abgebrochen und ein Backtrack ausgeführt.

5.3 Aufgetretene Probleme und deren Lösungen

Die ungeordnete und ungerichtete Paramodulationsregel ersetzt einen Term durch einen anderen, was in beide Richtungen erfolgen kann. Das heißt insbesondere, dass nach Anwendung der Paramodulation genau die selbe Regel aus dem Prolog-Code sofort wieder auf den gerade ersetzten Term anwendbar ist. Im Regelkörper vorhandene Bedingungen verhindern das nicht, weil das geänderte Teilziel immer an den Anfang des Ziels geschrieben wird, dahinter erst die Bedingungen. Weiterhin wird bei jedem Inferenzschritt immer das erste Teilziel betrachtet. Das Anwenden der gleichen Regel wird nicht verhindert. Es kann sogar notwendig sein, dass das geschieht, um einen weiteren Term im aktuellen Ziel zu ersetzen.

Um die Berechnungsschleifen, die mit der Paramodulation entstehen, verhindern zu können, werden die in Abschnitt 4.6 besprochenen Optimierungen, Eliminierung von Teilzielen und Speichern der Historie bereits erreichter Ziele, benötigt.

Bei der gerichteten Paramodulation ist der Programmierer in der Verantwortung, keine Regeln zu benutzen, die ein Expandieren der Hypothese ermöglichen. Dazu darf in den im Abschnitt 5.1 gezeigten Regeln der gerichteten Paramodulation der Term b kein echter Teilterm des Terms a sein.

Die nächsten beiden Abschnitte beschäftigen sich mit Problemen, die bereits bei einer einzigen Regel der Paramodulation auftreten.

5.3.1 Vermeidung der Expansion der Hypothese

Die Idempotenz und die Kommutativität werden angewendet, um das Expandieren der Hypothese zu verhindern. Angenommen die Regel lautet

$$\text{"}=\text{"}(c, d) :- b_1, \ldots, b_n.$$

wobei b_1 bis b_n Atomformeln sind, und sie wird auf den gerade ersetzten Teilterm wieder angewendet. c und d erfüllen die folgenden Bedingungen:

- c und d enthalten genau die selben Variablen.
- c ist kein Teilterm von d und d kein Teilterm von c.

Weiterhin erfüllen die Atomformeln b_1 bis b_n folgende Bedingungen:

- In b_1 bis b_n kommen keine anderen Variablen vor als in c und d.

- Keine der Atomformeln b_1 bis b_n ist ein Aufruf eines Prädikats mit Nebeneffekten.

Dann sind die neu hinzuzufügenden Teilziele identisch mit den bereits einen Schritt vorher der Hypothese hinzugefügten. Deshalb werden sie nicht wieder der Hypothese hinzugefügt.

Beweis. Gegeben sei folgende Regel und Hypothese, die variablenfremd sind:

$" = "(c, d) : -b_1, \ldots, b_n.$
$? - p_1(a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_m), p_2, \ldots, p_o.$

Ohne Beschränkung der Allgemeinheit wird zuerst c gesucht und durch d ersetzt. Weiterhin wird der gesamte Term a_i ersetzt. Analog gilt das auch für den Fall, dass ein Teilterm von a_i ersetzt wird. Die Paramodulation läuft nun in folgenden Schritten ab:

1. Es sei a_i mit c unifizierbar und die Paramodulation wird auf den Term a_i angewendet, so dass c gesucht und durch d ersetzt wird. Weiterhin ist ϑ_1 die Menge der notwendigen Termersetzungen bei der Unifikation von a_i und c. Sei

   $$\vartheta^* := \{[V, t] \in \vartheta_1 | V \text{ ist eine Variable, die im Term } c \text{ vorkommt}\}$$

   Es gilt $\vartheta_1(d) = \vartheta^*(d)$ und $\vartheta_1(b_j) = \vartheta^*(b_j)$, weil d und b_j nur Variablen enthalten, die c auch enthält. Seien

   $$d' := \vartheta^*(d)$$

   $$a'_j := \vartheta_1(a_j)$$
   $$b'_j := \vartheta^*(b_j)$$
   $$p'_j := \vartheta_1(p_j)$$

   Das neue Ziel ist:

   $$? - p_1(a'_1, \ldots, a'_{i-1}, d', a'_{i+1}, \ldots, a'_m), b'_1, \ldots, b'_n, p'_2, \ldots, p'_o$$

2. Die Regel wird wieder auf das i-te Argument von p_1, jetzt d', angewendet. Da c kein Teilterm von d ist und d kein Teilterm von c, muss nun d durch c ersetzt werden.
   Zu beachten ist, dass das neue Ziel und die Regel wieder variablenfremd sind. Die Terme d und d' werden miteinander unifiziert, was die Termersetzungen $\vartheta_2 = \vartheta^* \circ \vartheta''_2$ ergibt. Dabei ist ϑ''_2 die Menge der Termersetzungen, die die Variablen von d aus Schritt 1 für die entsprechenden Variablen von d aus diesem Schritt ersetzt. ϑ''_2 hat dabei nur Auswirkungen auf c, d und die b_j. Alle Terme der Hypothese bleiben bei der Anwendung von ϑ''_2 unverändert. Für die Bedingungen b_j der Regel gilt, dass $\vartheta''_2(b_j)$ gleich den b_j aus Schritt 1 ist.

Da $\vartheta^* \subseteq \vartheta_1$ und eine Termersetzung ϑ, die mit den Algorithmen aus Abschnitt 4.3 erstellt wurde, eine Variable nicht durch einen Term ersetzt, der eine Variable enthält, die auch von ϑ ersetzt wird, gilt für einen beliebigen Term x: $\vartheta^*(\vartheta_1(x)) = \vartheta_1(x)$ und $\vartheta^*(\vartheta^*(x)) = \vartheta^*(x)$. Sei

$$c'' := \vartheta^*(\vartheta_2''(c))$$

$$a'_j = \vartheta^*(\vartheta_2''(a'_j))$$
$$b'_j = \vartheta^*(\vartheta_2''(b'_j)) = \vartheta^*(\vartheta_2''(b_j))$$
$$p'_j = \vartheta^*(\vartheta_2''(p'_j))$$

Ohne Ausnutzung der Idempotenz und der Kommutativität würde das neue Ziel

$$? - p_1(a'_1, \ldots, a'_{i-1}, c'', a'_{i+1}, \ldots, a'_m), b'_1, \ldots, b'_n, b'_1, \ldots, b'_n, p'_2, \ldots, p'_o.$$

lauten. Aber da die Atomformeln b'_1 bis b'_n keine Prädikate mit Nebeneffekten aufrufen, kann die Optimierung angewendet werden. Die neue Hypothese ist:

$$? - p_1(a'_1, \ldots, a'_{i-1}, c'', a'_{i+1}, \ldots, a'_m), b'_1, \ldots, b'_n, p'_2, \ldots, p'_o.$$

Zu beachten ist, dass die Bedingungen der Regel nur einmal in der Hypothese stehen.

3. Das erneute Anwenden der Regel mit Ersetzen des i-ten Arguments von p_1, jetzt c'', ist Analog zur vorherigen Ersetzung von d' durch c. Auch dabei expandiert die Hypothese nicht. □

Falls in der Regel " = "$(c, d) : -b_1, \ldots, b_n$. c ein echter Teilterm von d oder d ein echter Teilterm von c ist, wird die Paramodulation nicht mehr in beide, sondern nur noch in eine Richtung (geordnet) ausgeführt. Die Regel kann nur endlich oft angewendet werden (näheres dazu im folgenden Abschnitt 5.3.2). Dadurch kann die Hypothese auch ohne die obigen weiteren an c, d und b_1 bis b_n gestellten Bedingungen nicht unendlich expandieren.

Bei der gerichteten Paramodulation ist der Programmierer dafür verantwortlich, dass es keine Regel der Formen

$$" = "(a, b, d).$$

$$" = "(a, b, d) : -b_1, \ldots, b_n.$$

gibt, in denen b ein Teilterm von a ist. Wenn b kein Teilterm von a ist, gelten die obigen Erläuterungen für die geordnete Paramodulation auch für die gerichtete.

5.3.2 Vermeidung von unendlichen Schleifen

Nachdem das Expandieren der Hypothese bei einer Regel der Paramodulation verhindert wurde, bleibt noch das Problem der Schleifen zu lösen. Wie bereits erwähnt, kann eine Regel der Paramodulation wegen der Symmetrie der Gleichheit auf einen gerade ersetzten Term sofort wieder angewendet werden. Dadurch würde das Prolog-Programm bereits bei einer einzigen Regel der Paramodulation in eine Schleife geraten und nicht mehr terminieren. Weil die Hypothese nicht unendlich expandiert, ist es ausreichend in diesem Abschnitt nur Regeln der Paramodulation zu betrachten, die Fakten sind.

Eine Regel der Paramodulation wird nur endlich oft auf einen bestimmten Teilterm angewendet. Dabei gibt es eine Ausnahme: die gerichtete Paramodulation, wenn ein Term b durch einen Term a ersetzt werden soll und b ein Teilterm von a ist.

Beweis. Gegeben sei die Regel

$$" = "(c, d)$$

c und d erfüllen die folgenden Bedingungen:

- c und d enthalten genau die selben Variablen.
- c ist kein Teilterm von d und d kein Teilterm von c.

Weiterhin ist ein Ziel (eine Hypothese) gegeben. Das Prolog-Programm sieht dann wie folgt aus:

$" = "(c, d).$
$? - p_1(a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_m), p_2, \ldots, p_o.$

Ohne Beschränkung der Allgemeinheit wird zuerst c gesucht und durch d ersetzt. Weiterhin wird der gesamte Term a_i ersetzt. Analog gilt der Beweis aber auch, wenn ein Teilterm von a_i ersetzt wird.

1. Es sei a_i mit c unifizierbar und die Paramodulation wird auf den Term a_i angewendet, so dass c gesucht und durch d ersetzt wird. Weiterhin ist ϑ_1 die Menge der notwendigen Termersetzungen für die Unifikation von a_i und c. Sei

   $$\vartheta^* := \{[V, t] \in \vartheta_1 | V \text{ ist eine Variable, die im Term } c \text{ vorkommt}\}$$

   Es gilt $\vartheta_1(d) = \vartheta^*(d)$, weil d nur Variablen enthält, die c auch enthält. Seien

   $$d' := \vartheta^*(d)$$

   $$a'_j := \vartheta_1(a_j)$$
   $$p'_j := \vartheta_1(p_j)$$

   Das neue Ziel ist:

   $$? - p_1(a'_1, \ldots, a'_{i-1}, d', a'_{i+1}, \ldots, a'_m), p'_2, \ldots, p'_o.$$

2. Die Regel wird abermals auf das i-te Argument von p_1, jetzt d', angewendet. Da c kein Teilterm von d ist und d kein Teilterm von c, muss nun d durch c ersetzt werden.
   Zu beachten ist, dass die neue Hypothese und die Regel wieder variablenfremd sind.
   Die Terme d und d' werden miteinander unifiziert, was die Termersetzungen $\vartheta_2 = \vartheta^* \circ \vartheta_2''$ ergibt. Dabei ist ϑ_2'' die Menge der Termersetzungen, welche die Variablen von d aus Schritt 1 für die entsprechenden Variablen von d aus diesem Schritt einsetzt. ϑ_2'' hat dabei nur Auswirkungen auf c und d. Alle Terme der Hypothese bleiben bei Anwendung von ϑ_2'' unverändert. Da $\vartheta^* \subseteq \vartheta_1$ und eine Termersetzung ϑ, die mit den Algorithmen aus Abschnitt 4.3 erstellt wurde, eine Variable nicht durch einen Term ersetzt, der eine Variable enthält, die auch von ϑ ersetzt wird, gelten für einen beliebigen Term x: $\vartheta^*(\vartheta_1(x)) = \vartheta_1(x)$ und $\vartheta^*(\vartheta^*(x)) = \vartheta^*(x)$. Sei

$$c'' := \vartheta^*(\vartheta_2''(c))$$

$$a_j' = \vartheta^*(\vartheta_2''(a_j'))$$
$$p_j' = \vartheta^*(\vartheta_2''(p_j'))$$

   Das neue Ziel ist:

$$? - p_1(a_1', \ldots, a_{i-1}', c'', a_{i+1}', \ldots, a_m'), p_2', \ldots, p_o'$$

3. Die Regel wird wieder auf das i-te Argument von p_1, jetzt c'', angewendet. c wird gesucht und durch d ersetzt. Die Terme c und c'' werden miteinander unifiziert, was die Termersetzungen $\vartheta_3 = \vartheta^* \circ \vartheta_3'''$ ergibt. Dabei ist ϑ_3''' die Menge der Termersetzungen, welche die Variablen von c'' aus Schritt 2 für die entsprechenden Variablen von c aus diesem Schritt einsetzt. Da c und d genau die gleichen Variablen enthalten, ist $\vartheta_3''' = \vartheta_2''$ und damit auch $\vartheta_3 = \vartheta_2$. Insbesondere ist $\vartheta_3'''(d) = \vartheta_2''(d)$ gleich dem d aus Schritt 1. Sei

$$d''' := \vartheta^*(\vartheta_3'''(d)) = d'$$

$$a_j' = \vartheta^*(\vartheta_3'''(a_j'))$$
$$p_j' = \vartheta^*(\vartheta_3'''(p_j'))$$

   Das neue Ziel ist:

$$? - p_1(a_1', \ldots, a_{i-1}', d', a_{i+1}', \ldots, a_m'), p_2', \ldots, p_o'$$

   Durch die Speicherung der Historie der bereits erreichten Ziele wird die Schleife erkannt, das gerade errechnete Ziel nicht akzeptiert und ein Backtrack ausgelöst.

Falls in der Regel " = "(c, d). c ein echter Teilterm von d oder d ein echter Teilterm von c ist, wird die Paramodulation nicht mehr in beide, sondern nur noch in eine Richtung (geordnet) ausgeführt. Es wird nur noch der größere Term durch den kleineren ersetzt. Da Terme nur endlich groß sein können und es nur endlich viele

Terme im Kopf des Ziels gibt, muss diese Ersetzung zwangsläufig enden, weil der größere Term nicht mehr gefunden wird.

Für die gerichtete Paramodulation mit einer Regel der Form

$$" = "(a, b, d).$$

gelten die obigen Erläuterungen für die geordnete Paramodulation, wenn b kein echter Teilterm von a ist. □

5.3.3 Paramodulation in Variablen

Paramodulation in Variablen bedeutet, dass der Term des Ziels, der durch die Paramodulation ersetzt wird, eine Variable ist [Geo, Paramodulation]. Bei der Paramodulation in Variablen kann es zu einer Expansion eines Terms des Ziels kommen, wenn die erste Atomformel des Ziels und die Regel der Paramodulation jeweils Variablen besitzen. Die Variable im Ziel wird dabei durch einen anderen, möglicherweise größeren, Term ersetzt, der wieder eine Variable enthält. Diese Variable kann abermals ersetzt werden. Ein kleines Beispiel veranschaulicht dies:

$" = "(f(A), g(A)).$
$? - p(B).$

Durch die Paramodulation wird B mit $f(A)$ unifiziert und schließlich durch $g(B)$ ersetzt:

$$? - p(g(B)).$$

Auf die Variable B kann nun wieder die Paramodulation angewendet werden. Das neue Ziel ist:

$$? - p(g(g(B))).$$

Das Prolog-Programm wird nie terminieren. Neben dem Nachteil von nicht terminierenden Programmen kann die Paramodulation in Variablen auch Resolutionsmöglichkeiten einschränken, indem eine Variable durch einen spezielleren Term ersetzt wird. Weiterhin wird Rechenzeit und Speicher benötigt um die Paramodulation durchzuführen und die neu entstandenen Ziele zu speichern.

Nach [Geo, Paramodulation] ist die Paramodulation in Variablen nicht notwendig. Das Ziel der Paramodulation ist, dass ein Term durch den Austausch mit einem semantisch gleichen, aber sytaktisch verschiedenen, Term mit einem Term eines Klauselkopfes unifizierbar wird. Eine Variable ist aber bereits mit allen anderen variablenfremden Termen unifizierbar. Das aktuelle Ziel und die Regeln sind immer variablenfremd. Die Paramodulation in Variablen kann deshalb ohne Folgen verboten werden.

5.3.4 Termination eines Prolog-Programms mit Paramodulation

Ein Ziel enthält immer nur endlich viele Terme, insbesondere enthält auch das erste Teilziel des Ziels nur endlich viele Terme. Eine Regel der Paramodulation kann wegen der endlichen Zahl an Termen und den in den Abschnitten 5.3.2 und 5.3.3 vorgesehenen Maßnahmen nur endlich oft auf das erste Teilziel angewendet werden. Die Anzahl der dem Ziel hinzugefügten Bedingungen ist nach Abschnitt 5.3.1 auch endlich.

Es fehlt noch die Betrachtung der Anwendbarkeit einer Regel der Paramodulation auf ihre Bedingungen und das Zusammenspiel von mehreren Regeln innerhalb eines Prolog-Programms. Letzteres wird in dieser Arbeit nicht betrachtet, weil bereits in Prolog ohne Paramodulation mehrere Regeln leicht unendliche Schleifen bilden können.

Wenn eine Regel der Paramodulation eine Bedingung mit einem Term enthält, auf den die Regel selber anwendbar ist, kann dies zu einem nicht terminierenden Programm führen. Es folgen zwei Beispiele:

Beispiel 1: Das Programm terminiert.

$" = "(f(A), g(A)) :- p(f(A)).$
$p(A).$
$? - p(g(a)).$

Beispiel 2: Das Programm terminiert nicht.

$" = "(f(A), g(A)) :- p(f(h(A))).$
$p(A).$
$? - p(g(a)).$

Im ersten Beispiel bleibt der Term in der Bedingung in jedem Schritt gleich. Dadurch werden immer wieder die gleichen Ziele errechnet, was durch die Historie der bereits erreichten Ziele erkannt wird. Im zweiten Beispiel lautet die Bedingung, die dem Ziel hinzugefügt wird, zuerst $p(f(h(a)))$. Wenn diese Bedingung die erste Atomformel des Ziels und die Regel der Paramodulation darauf angewendet wird, wird die Bedingung $p(f(h(h(a))))$ dem Ziel hinzugefügt. Die Bedingung expandiert. Insbesondere ändert sie sich. Dadurch werden neue Ziele errechnet, die noch nie zuvor erreicht wurden und die Schleifenerkennung durch die Historie der bereits erreichten Ziele erkennt diesen Fall nicht.

Der Programmierer des Prolog-Programms muss dafür Sorge tragen, dass solche Schleifen nicht auftreten. Doch auch Prolog ohne Paramodulation hat dieses Problem. Beispielsweise kann eine Regel, die die Transitivität beschreibt, immer wieder auf ihre eigenen Bedingungen angewendet werden. Eine solche Regel ist zum

Beispiel:

$$liegt\_auf(A, B) : -liegt\_auf(A, C), liegt\_auf(C, B).$$

Die Entscheidung, dem Programmierer die Verantwortung für das Terminieren der Programme zu übertragen, wurde auch von den Prolog-Entwicklern getroffen: „Although, certainly, the use of backtracking led to a loss of completeness in deductions comprising infinite branches, we felt that, given the simplicity of the deduction strategy (the execution of literals from left to right and the choice of clauses in the order they were written), it was up to the programmer to make sure that the execution of his program terminated.“ [Col93, Seite 14]

Die Deduktionsstrategie von Prolog wurde im Deduktionstool übernommen. Auch die Reihenfolge der Termersetzungen, die die Paramodulation vornimmt, ist, wie in Abschnitt 5.2 beschrieben, fest vorgegeben und einfach.

5.4 Überblick über die Einschränkungen

Die folgenden Abschnitte geben einen Überblick über die in dieser Arbeit getroffenen Einschränkungen. Dabei besitzen die drei Arten der Paramodulation, die in dieser Arbeit verwendet werden (ungeordnete, geordnete und gerichtete), jeweils eigene Einschränkungen.

Allen drei Arten ist aber gemein, dass die Paramodulation in Variablen nicht erlaubt ist. Sie kann zu nicht terminierenden Programmen führen und ist nicht nötig, weil für eine Variable jeder andere variablenfremde Term eingesetzt werden kann.

5.4.1 Einschränkungen bei der ungeordneten Paramodulation

Es ist nicht möglich eine Regel der Paramodulation mit ” = ”(a, b) im Klauselkopf zu schreiben, die die Terme a und b miteinander in beide Richtungen ersetzt und bei der b ein echter Teilterm von a ist oder umgekehrt. Eine solche Regel würde zu einem nicht terminierenden Programm führen. Weiterhin sind Regeln nicht erlaubt, bei denen a und b syntaktisch gleich sind. Solche Regeln lassen sich aber durch zwei Regeln der gerichteten Paramodulation nachbilden.

Weiterhin sind im Körper einer Regel mit ” = ”(a, b) im Klauselkopf keine Prädikate mit Nebeneffekten erlaubt. Prädikate mit Nebeneffekten verhindern die Eliminierung von Teilzielen, welche das Expandieren des Ziels verhindert, wodurch das Programm wieder nicht terminiert. Diese Einschränkung kann umgangen werden, indem eine neue Regel (in diesem Fall nicht der Paramodulation) eingeführt wird, deren Kopf die Stelle des Prädikats mit Nebeneffekten im Regelkörper einnimmt und in ihrem Körper das Prädikat mit Nebeneffekten enthält. Ein kleines Beispiel zur Veranschaulichung: Die Regel

$$" = "(a, b) :- write(a, " = ", b).$$

enthält das Prädikat *write*, das einen Nebeneffekt hat. Die beiden Regeln

$$" = "(a, b) :- p(a, b).$$
$$p(a, b) :- write(a, " = ", b).$$

erzielen den gewünschten Effekt und verhindern dabei nicht die Eliminierung von Teilzielen.

Außerdem dürfen in einer Regel der Paramodulation in beide Richtungen keine neuen Variablen eingeführt werden. Das heißt, dass bei einer Regel der Form

$$" = "(a, b) :- p_1, \ldots, p_n.$$

in den Termen a und b genau die selben Variablen vorkommen müssen. In den Bedingungen p_1 bis p_n dürfen darüber hinaus keine Variablen vorkommen, die nicht in a und b vorkommen. Den neuen Variablen werden nach Abschnitt 4.2 neue Nummern zugewiesen. Da die Speicherung bereits erreichter Ziele (Abschnitt 4.6.2) ohne Berücksichtigung von Variablenumbennungen arbeitet, wird jedes Ziel mit neuen Variablen als neu eingestuft. Die Schleifenerkennung funktioniert nicht mehr. Dadurch wird das Prolog-Programm nicht terminieren. Im Bedingungsteil kann die Einschränkung umgangen werden. Dazu wird, wie bei der obigen Einschränkung, eine neue Regel eingeführt.

5.4.2 Einschränkungen bei der geordneten Paramodulation

Die geordnete Paramodulation wird immer automatisch angewendet, wenn bei einer Regel der Form

$$" = "(a, b) :- p_1, \ldots, p_n.$$

a ein Teilterm von b ist oder umgekehrt. Dieses Verhalten kann nicht ausgeschaltet werden. Insbesondere ist es nicht möglich, mit der geordneten Paramodulation einen Term c mit einem Term zu ersetzen, von dem c ein echter Teilterm ist.

Weiterhin ist es bei der geordneten Paramodulation nicht möglich, einen Term durch einen syntaktisch gleichen zu ersetzen.

Die Paramodulation wird nur auf das Ziel angewendet, nicht auf die Programmklauseln. Deshalb sollten die Terme in den Köpfen der Programmklauseln in der Normalform stehen, die die geordnete Paramodulation erreichen soll. Die Unifikation wird verhindert, wenn der entsprechende Term im Kopf der Programmklausel nicht auf dem Weg zur Normalform liegt, die die Paramodulation für den Term des Ziels anstrebt.

5.4.3 Einschränkungen bei der gerichteten Paramodulation

Bei der gerichteten Paramodulation sind Regeln

$$" = "(a, b, d) : -p_1, \ldots, p_n.$$

möglich, bei denen b ein echter Teilterm von a ist. Dadurch würde der entsprechende Term des Ziels durch die wiederholte Anwendung der Regel expandieren. Das Prolog-Programm würde nie Terminieren. Solche Regeln sollten deshalb vom Programmierer vermieden werden.

Regeln, bei denen ein Term durch einen syntaktisch gleichen ersetzt werden soll, sind auch bei der gerichteten Paramodulation verboten. Sie sind sinnlos und würden bei einem Prädikat mit Nebeneffekten in der Bedingung zu einem nicht terminierenden Programm führen.

Ähnlich wie bei der geordneten Paramodulation müssen auch bei der gerichteten Paramodulation die Terme in den Köpfen der Programmklauseln auf dem Weg liegen, den die Paramodulation bei der Ersetzung der Terme des Ziels nimmt.

KAPITEL 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

Die logischen Grundlagen für das im Rahmen dieser Arbeit entwickelten Deduktionstools sind die Prädikatenlogik erster Stufe, HORN-Klauseln und das Folgern. In der grundlegenden Arbeitsweise ist es an Prolog angelehnt. Dabei werden die Unifikation, die Resolution, das Ordnen der Klauseln und der Atomformeln innerhalb des Klauselkörpers und Backtrack eingesetzt.

Plug-ins ermöglichen die Erweiterung des Tools ohne das Tool selber ändern zu müssen. Dabei haben sie mehrere Möglichkeiten die Erweiterungen in das Deduktionstool einzubauen.

Durch Optimierungen wird erreicht, dass das Deduktionstool trotz der Symmetrie der Gleichheit terminieren kann.

Es gibt mehrere Möglichkeiten, die Gleichheit zu implementieren. Die Paramodulation ist eine davon. Neben der (ungeordneten) Paramodulation wurden auch zwei Einschränkungen der Paramodulation vorgestellt: die geordnete und die gerichtete Paramodulation.

Das Deduktionstool verwendet alle drei Varianten der Paramodulation, um möglichst wenige Einschränkungen treffen zu müssen. Dabei wird automatisch zwischen der ungeordneten und der geordneten Paramodulation ausgewählt. Die gerichtete Paramodulation kann explizit vom Programmierer gewählt werden. Dennoch sind weiterhin Einschränkungen vorhanden.

Das Beispiel aus der Einleitung, Abschnitt 1.1, kann durch die Paramodulation leicht gelöst werden:

$rentner(herr\_mueller).$
$nachbarn(herr\_mueller, frau\_schmidt).$
$nachbarn(A, B) :- nachbarn(B, A).$
$" = "(A, nachbar\_von(B), d) :- nachbarn(A, B).$
$? - rentner(nachbar\_von(frau\_schmidt)).$

Die Regel

$$nachbarn(A, B) :- nachbarn(B, A).$$

löst dabei die Symmetrie der Nachbarschaftsrelation. Durch die eingeführten Optimierungen terminiert das Programm trotz dieser Regel. Weiterhin stellt die Regel der (gerichteten) Paramodulation

$$\text{"}=\text{"}(A, nachbar\_von(B), d) : -nachbarn(A, B).$$

den Zusammenhang zwischen dem Prädikat $nachbarn(A, B)$ und dem strukturierten Term $nachbar\_von(B)$ her.

Das Ergebnis dieser Machbarkeitsstudie ist, dass die Paramodulation in einem Deduktionstool realisierbar ist, aber unter Einschränkungen. Die Einschränkungen, die bei dem in dieser Arbeit verwendeten Ansatz notwendig sind, sind im Abschnitt 5.4 zusammengefasst. Allerdings wurden in dieser Arbeit aus Gründen der Zeit und des Aufwands keine interaktiven Programme betrachtet.

6.2 Ausblick

Bei der ungeordneten Paramodulation müssen bei einer Regel

$$\text{"}=\text{"}(a, b) : -b_1, \ldots, b_n$$

die Terme a und b die selben Variablen enthalten. Außerdem dürfen die Bedingungen b_1 bis b_n keine Variablen enthalten, die nicht in a und b vorkommen. Interessant wäre eine Vorverarbeitung der Regel, damit sie möglichst variablenarm ist. Für das Beispiel der Nachbarn würde das bedeuten, dass aus den Regeln

---

$\text{"}=\text{"}(A, sekretaerin(B)) : -sekretaerin\_von(A, B).$
$sekretaerin\_von(frau\_schreiber, herr\_winter).$
$sekretaerin\_von(frau\_schreiber, frau\_haas).$

---

durch die Anwendung des Prädikats $sekretaerin\_von$ die Fakten

---

$\text{"}=\text{"}(frau\_schreiber, sekretaerin\_von(herr\_winter)).$
$\text{"}=\text{"}(frau\_schreiber, sekretaerin\_von(frau\_haas)).$

---

generiert werden.

Vermutlich fällt die Einschränkung, dass bei der ungeordneten Paramodulation keine neuen Variablen auftauchen dürfen, weg, wenn die Idempotenz und der Vergleich der Hypothesen bei der Schleifenerkennung mit gebundener Umbenennung der Variablen arbeiten und nicht auf exakte Gleichheit testen.

Bei der geordneten Parmodulation sollten die Terme in den Köpfen der Programmklauseln in Normalform sein, um eine Unifikation nicht zu verhindern. Eine Möglichkeit, die Programmklauseln so vorzuverarbeiten, dass ihre Terme in der Normalform sind, wäre wünschenswert.

Interaktive Prolog-Programme wurden in dieser Arbeit nicht betrachtet. Sie können Probleme bei der Schleifenerkennung verursachen. Die Entwicklung einer Schleifenerkennung, die auch bei interaktiven Programmen funktioniert, wäre interessant. Weiterhin könnte untersucht werden, ob die Paramodulation nur mit geordneter Paramodulation realisiert werden kann. Dann wäre eine Schleifenerkennung für die Paramodulation nicht mehr notwendig.

Aus Zeitmangel wurde das implementierte Deduktionstool nicht ausführlich getestet. Der Test sollte nachgeholt werden, wenn das Tool im Produktivbetrieb verwendet werden soll.

Anhang

A Kapitel 4: Unifikationsalgorithmus aus [Baa01, Seite 447f]

Die Autoren von [Baa01] verwenden andere Notationen. Termersetzungen werden in Postfixnotation angegeben:

$$s\sigma$$

bedeutet das gleiche wie in der Notation dieser Arbeit $\sigma(s)$, wobei σ Termersetzungen sind und s ein Term ist. Weiterhin wird auch die Komposition von Termersetzungen in Postfixnotation angegeben:

$$\sigma\vartheta$$

bedeutet in der Notation dieser Arbeit $\vartheta \circ \sigma$ für Termersetzungen σ und ϑ. Id ist die Identität. Für Termersetzungen σ, welche die Identität sind, gilt: $\sigma = \emptyset$.

Der Unifikationsalgorithmus aus [Baa01, Seite 447f] ist:

```
global σ : substitution;   { Initialized to Id }

Unify( s : term; t : term )
    begin
        if s is a variable then    { Instantiate variables }
            s := sσ;
        if t is a variable then
            t := tσ;
        if s is a variable and s = t then
            { Do nothing }
        else if s = f(s_1, ..., s_n)
                and t = g(t_1, ..., t_m) for n, m ≥ 0 then begin
            if f = g then
                for i := 1 to n do
                    Unify( s_i, t_i );
            else Exit with failure     { Symbol clash }
        end
        else if s is not a variable then
            Unify( t, s );
        else if s occurs in t then
            Exit with failure;     { Occurs check }
```

```
        else σ := σ{s ↦ t};
    end;
```

„(In an actual implementation, the case “Unify(t, s)” could be moved up before the first “else if” and simply swap s and t if the former is not a variable.)“ [Baa01, Seite 448]

Literaturverzeichnis

[Baa01] Baader, F. und W. S.: Chapter 8 Unification Theory. In: *HANDBOOK OF AUTOMATED REASONING*, Elsevier Science Publishers B.V., 2001

[Bar77] Barwise, Jon: An Introduction to First-Order Logic. Version: 1977. http://dx.doi.org/10.1016/S0049-237X(08)71097-8. In: Barwise, Jon (Hrsg.): *HANDBOOK OF MATHEMATICAL LOGIC* Bd. 90. Elsevier, 1977. – DOI 10.1016/S0049–237X(08)71097–8. – ISSN 0049–237X, 5 - 46

[Col93] Colmerauer, Alain: The Birth of Prolog. In: *III, CACM Vol.33, No7*, 1993, 37–52

[FHS89] Furbach, Ulrich; Hölldobler, Steffen; Schreiber, Joachim: Horn equational theories and paramodulation. In: *Journal of Automated Reasoning* 5 (1989), 309-337. http://dx.doi.org/10.1007/BF00248322. – ISSN 0168–7433. – 10.1007/BF00248322

[Geo] *Paramodulation.* http://www.cs.miami.edu/~geoff/Courses/TPTPSYS/FirstOrder/Paramodulation.shtml, Abruf: 14.02.2012

[hmi] *Kapitel 4 Prolog.* http://www-hm.ma.tum.de/archiv/in2/ss04/skript/hmin2_logik4.pdf, Abruf: 05.05.2012

[Kna] Knauf, Rainer: *Inferenzmethoden.* http://www.tu-ilmenau.de/fileadmin/media/ki/lehrveranst/InfMeth/InfMeth-9.pdf, Abruf: 01.11.2011

[Kna11] Knauf, Rainer: *Skript zur Lehrveranstaltung Künstliche Intelligenz.* Version: 2010/2011. http://www.tu-ilmenau.de/fileadmin/media/ki/lehrveranst/KI/KIABL.pdf, Abruf: 14.12.2010

[Kna12] Knauf, Rainer: *Arbeitsblätter zur Lehrveranstaltung Inferenzmethoden.* Version: 2011/2012. http://www.tu-ilmenau.de/fileadmin/media/ki/lehrveranst/InfMeth/ABLIM.pdf, Abruf: 01.11.2011

[Man08] Manhart, Klaus: *Wissensbasierte Modellierung: Eine kurze Einführung in Prolog.* Version: 2008. http://www.klaus-manhart.de/mediapool/28/284587/data/prolog.pdf, Abruf: 24.04.2012

[NR99] Nieuwenhuis, Robert; Rubio, Albert: Chapter 1 Paramodulation-based theorem proving. In: *HANDBOOK OF AUTOMATED REASONING*, Elsevier Science Publishers B.V., 1999

[Rob65] Robinson, J. A.: A Machine-Oriented Logic Based on the Resolution Principle. In: *J. ACM* 12 (1965), Januar, Nr. 1, 23–41. http://dx.doi.org/10.1145/321250.321253. – DOI 10.1145/321250.321253. – ISSN 0004–5411

[Rot92] Rottmann, Torsten: *Effiziente Prolog – Übersetzung*, University of Oldenburg, Diploma Thesis, 1992. http://zeizen.com/Thesis_files/diplom.pdf, Abruf: 21.04.2012

[RW68] Robinson, George; Wos, Lawrence: PARAMODULATION AND THEOREM PROVING IN FIRST ORDER THEORIES WITH EQUALITY. (1968). http://www-public.slac.stanford.edu/sciDoc/docMeta.aspx?slacPubNumber=SLAC-PUB-0482. – Presented at 4th Annual Machine Intelligence Workshop, Edinburgh, Scotland, Aug 1968

[Sá10] Sági, G.: A SHORT PROOF FOR THE COMPLETENESS OF PARAMODULACION. In: *Bulletin of the Section of Logic* 39 (2010), Nr. 3/4, 147–152. http://www.filozof.uni.lodz.pl/bulletin/pdf/39_34_4.pdf

[San05] Sandow, Saskia: *Einführung in Schließen über Gleichheit*. Version: 2005. http://www.cs.uni-potsdam.de/ti/lehre/05-Inferenzmethoden/slides/SIM2005_sandow.pdf, Abruf: 24.04.2012

[Sti88] Stickel, Mark E.: A prolog technology theorem prover: Implementation by an extended prolog compiler. In: *Journal of Automated Reasoning* 4 (1988), 353-380. http://dx.doi.org/10.1007/BF00297245. – ISSN 0168–7433. – 10.1007/BF00297245

[SWI] *SWI-Prolog manual*. http://www.swi-prolog.org/pldoc/refman/, Abruf: 17.05.2012

[Wika] *Expertensystem – Wikipedia*. http://de.wikipedia.org/wiki/Expertensystem, Abruf: 19.01.2012

[Wikb] *Regulärer Ausdruck – Wikipedia*. http://de.wikipedia.org/wiki/Regul%C3%A4rer_Ausdruck, Abruf: 02.06.2012

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Ilmenau, den 08. Juni 2012

Andreas Loth