DIPLOMARBEIT

Die Jede-Welt-Annahme in der Logikprogrammierung als Paradigma zur Wissensverarbeitung für das Semantic Web

von
Tobias Matzner

eingereicht am 1. September 2006 beim
Institut für Angewandte Informatik
und Formale Beschreibungsverfahren
der Universität Karlsruhe (TH)

Referent: Prof. Dr. Rudi Studer
Betreuer: Dr. habil. Pascal Hitzler

Anschrift:
Gartenstraße 32
76133 Karlsruhe

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte fremde Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Karlsruhe, 1. September 2006 .....

Ich möchte Markus Krötzsch, Boris Motik und Umberto Straccia für ihre Anregungen und schnellen Antworten auf meine Fragen danken. Mein besonderer Dank gilt Pascal Hitzler für die wunderbare Zusammenarbeit und für seine Unterstützung auch bei Themen, die nicht direkt mit dieser Arbeit zusammenhängen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im Jahr 1976 veröffentlichte N.D. Belnap einen Artikel mit dem Titel „How a computer should think“[Bel76]. Auch wenn sich trefflich darüber streiten lässt, ob wir seitdem einem „denkenden“ Computer nähergekommen sind, haben sich seitdem zahlreiche aktive Forschungsgebiete entwickelt, die daran arbeiten. Oder etwas einschränkender gesagt, die versuchen, Wissen mit Hilfe eines Computers in einer Art und Weise zu verarbeiten, die der menschlichen Intuition mehr oder weniger gerecht wird. Eines dieser Forschungsgebiete entstand ungefähr zur gleichen Zeit wie die Arbeit von Belnap: die Logikprogrammierung. Sie geht zurück auf Kowalski und Van Emden [Kow74, vK76]. Im Gegensatz zur imperativen Programmierung, bei der mit sequenziellen Befehlen dem Rechner die Bearbeitung eines Problems vorgeschrieben wird, stellt die Logikprogrammierung eine Menge an Regeln und Fakten zur Verfügung. Bei einer Anfrage an das System muss dann durch Kombination der Regeln und deren Anwendung auf die bekannten Fakten eine Antwort gesucht werden. Wie genau dieses Suchen stattfindet, ist eines der zentralen Probleme der Logikprogrammierung. Dieses wird uns in der vorliegenden Arbeit jedoch eher am Rande interessieren. Sie beschäftigt sich mit der Semantik der Logikprogramme, also mit der Frage, wie eine gegebene Menge an Fakten und Regeln interpretiert werden sollte. Sie wird besonders spannend, wenn es darum geht, wie die Negation aufzufassen ist. Diese unterscheidet sich in der Logikprogrammierung von der Negation in der Prädikatenlogik. Auch ansonsten ist es oft nicht möglich, Parallelen zwischen diesen beiden Formalismen zu ziehen, was sich schon daran zeigt, dass die Prädikatenlogik nicht entscheidbar ist – eine Tatsache, die sie als Paradigma für Programmierung eher ungeeignet macht.

Die bekannteste Form von Negation in der Logikprogrammierung ist die sogenannte *negation as failure*. Dabei wird zur Evaluierung eines negierten Ausdrucks versucht, den nicht negierten Ausdruck abzuleiten. Gelingt dies nicht, wird der negierte Ausdruck als wahr betrachtet. Die Grundlage für dieses Vorgehen hängt mit der nächsten spannenden Frage zusammen: Was

passiert, wenn die gegebenen Informationen nicht ausreichen, um eine an das System gestellte Anfrage zu bearbeiten? Die Antwort, die der *negation as failure* zugrunde liegt, heißt Geschlossene-Welt-Annahme. Geschlossen in dem Sinn, dass davon ausgegangen wird, dass die bekannten Fakten alles darstellen, was es (Wahres) zu wissen gibt. Alles was nicht daraus abgeleitet werden kann, wird also als falsch betrachtet. Sie beinhaltet aber auch die Annahme, dass die dem System bekannten „Objekte“ oder „Dinge“ alle sind, die es gibt. Das Gegenstück dazu heißt Offene-Welt-Annahme. Sie besagt, dass über Aussagen, über die nichts bekannt ist, auch nichts gesagt werden kann und dass davon ausgegangen werden muss, dass es „Objekte“ oder „Dinge“ gibt, die nicht explizit bekannt sind. Diese Annahme mit der Logikprogrammierung zu kombinieren ist recht problematisch, was in dieser Arbeit noch thematisiert werden wird.

Es gibt jedoch diverse Ansätze, die Semantik von Logikprogrammen dahingehend zu erweitern, dass auch unvollständiges Wissen modelliert werden kann. Eine bekannte Arbeit dazu stammt von Fitting [Fit85]. Dort wird eine dreiwertige Logik verwendet, die neben den bekannten Werten *wahr* und *falsch* einen dritten einführt, der „nichts bekannt“ beschreibt. Noch spannender wird es, wenn wir auch die Frage stellen, die den anfangs erwähnten Artikel von Belnap motivierte: Was passiert, wenn das System widersprüchliche Informationen bekommt? In einer klassischen Logik kann aus einem Widerspruch alles abgeleitet werden. Belnap argumentiert, dass dies nicht das gewünschte Verhalten sein kann. Wenn ein (auch nur ansatzweise) „denkender“ Computer die Information bekommt, dass Italien 2006 Fussballweltmeister wurde und danach – etwa aus einer anderen Quelle – dass Italien 2006 nicht Fussballweltmeister wurde, wird sein Wissen über Fussball durcheinanderkommen. Er sollte daraus jedoch nicht folgern, dass der Mond aus grünem Käse besteht oder Radioaktivität für Menschen ungefährlich ist. Belnap erarbeitet eine vierwertige Logik, die neben den schon erwähnten Werten für *wahr, falsch* und *keine Information* einen zusätzlichen Wert für *widersprüchliche Information* enthält. In dieser Logik ist es möglich, einen Widerspruch zu modellieren und ihn bei einer entsprechenden Anfrage zu melden, aber dennoch sinnvoll weiterzuarbeiten. Dieser Ansatz wurde weiterentwickelt und mündete in der Konzeption sogenannter *Biverbände*, ausgehend von Ginsbergs Arbeit [Gin86]. Dabei handelt es sich um eine mathematische Struktur für vierwertige Logiken, wie die eben erwähnte, die später auf beliebig viele Werte erweitert wurde.

Darauf beruht das erste zentrale Konzept für unsere Arbeit. In [LS05] benutzen Loyer und Straccia einen Formalismus basierend auf Biverbänden, um neben Offener- und Geschlossener-Welt-Annahme eine neue Antwort auf die zweite spannende Frage zu geben: Anstatt bei unzureichenden Informationen anzunehmen, dass alles falsch oder alles unbekannt ist, erlauben sie, alles was in ihrer Logik formulierbar ist, auch als Grundannahme zu benutzen. Die

Autoren bezeichnen dies als Jede-Welt-Annahme. Diese bringt nicht nur eine größere Flexibilität für die Programmierung sondern ermöglicht auch die intuitivere Beschreibung von Anwendungen und Modellen. Man denke nur an die Aussage: „Der Angeklagte ist unschuldig.“ . Diese sollte im Zweifelsfall (also bei unvollständigen Informationen) *wahr* sein.

Die Forschung über die Semantik von Logikprogrammen beschäftigt sich jedoch nicht nur mit dem Wertebereich und der Grundannahme, die verwendet werden sollen. Wichtig ist auch die Frage, wie die Interpretation, also die Zuordnung von Aussagen im Programm zu Wahrheitswerten vonstatten geht. Auch hier spielen die oben erwähnten spannenden Fragen eine Rolle, da es beispielsweise bei unvollständigen oder widersprüchlichen Informationen durchaus mehrere Interpretationen geben kann, die sinnvoll sind, beziehungsweise es unterschiedliche Auffassungen davon gibt, was „sinnvoll“ hier bedeutet. Bereichert wurde die Forschung hier aus dem eng verwandten Bereich des *nichtmonotonen Schließens*.

Auch dies ist ein Gebiet der Logik, das sich von der Prädikatenlogik abgrenzt. Letztere ist monoton, das heißt, wenn zu einer Menge an Axiomen ein weiteres hinzugenommen wird, keine der zuvor ableitbaren Aussagen ungültig wird. Sollte das neue Axiom einem bestehenden widersprechen, werden im Gegensatz, wie bereits erwähnt, alle Aussagen ableitbar. Ein System, das beispielsweise die oben beschriebene *negation as failure* einsetzt, ist dagegen nicht monoton: Sei in einem System die Aussage A nicht ableitbar. Dann wird $not(A)$ als wahr angenommen. Kommt nun eine Information hinzu, welche die Ableitung von A ermöglicht, kann $not(A)$ nicht länger als wahr betrachtet werden.

In und zwischen diesen beiden Forschungsgebieten sind eine Vielzahl unterschiedlicher Semantiken entstanden, die sich nicht nur in ihrer Interpretation der Programme unterscheiden, sondern auch in der Art der verwendeten Formalismen oder Techniken. Eine der bekanntesten, die auch für die Semantik von imperativen Programmen eingesetzt wird, sind Fixpunktsemantiken: Interpretationen von Logikprogrammen sind allgemein gesprochen Zuordnungen von Aussagen und Wahrheitswerten. Sie werden mathematisch modelliert und dann wird ein Operator definiert, der den Einfluss des Programms auf diese Zuordnung beschreibt, eben dessen Semantik. Die Fixpunkte dieses Operators sind dann die gesuchten Ergebnisse des Programms. Weitere verwendete Techniken sind monotone und nichtmonotone semantische Operatoren, Programmtransformationen, *level-mappings*, die Lösung von Unterprogrammen, die Ermittlung von Abhängigkeiten und viele mehr (vgl. [HW05] S. 94).

Damit kommen wir zum zweiten zentralen Konzept für unsere Arbeit. In [HW05] entwerfen Hitzler und Wendt einen allgemeinen Formalismus zur Beschreibung von Semantiken für Logikprogramme. Damit sollen möglichst

viele der bekannten Semantiken dargestellt werden können und eine bessere Analyse ihrer Zusammenhänge ermöglicht werden, die bisher durch die sehr verschiedenen mathematisch-formalen Konzepte erschwert wurde. Dies ist den Autoren auch bereits für einige etablierte Semantiken gelungen. Ihr Formalismus arbeitet mit sogenannten *level-mappings*, die bestimmte Strukturen im Programm modellieren können.

Das zentrale theoretische Resultat unserer Arbeit ist die Eingliederung der mächtigen und flexiblen Semantik von Loyer und Straccia in diesen Formalismus, der dafür entsprechend erweitert wurde. Der erste Teil der vorliegenden Arbeit ist also wie folgt gegliedert: In Kapitel 2 werden die mathematischen Grundlagen vorgestellt. In Kapitel 3 behandeln wir die Semantiken von Logikprogrammen und führen insbesondere die von Loyer und Straccia konzipierte Jede-Welt-Annahme sowie den Formalismus von Hitzler und Wendt ein. In Kapitel 4 wird dann unser Resultat präsentiert.

Der zweite Teil der Arbeit befasst sich mit einem von uns realisierten Logikprogrammiersystem. Dieses basiert auf Prolog, der wohl bekanntesten Sprache für Logikprogrammierung. Ihre Semantik approximiert die sogenannte Kripke-Kleene-Semantik [Kun87]. Aufgrund unserer Analyse dieser Semantik basierend auf den theoretischen Resultaten, die im ersten Teil angewendet wurden, haben wir eine Programmtransformation entwickelt, die auch gewisse andere Semantiken auf einem System, das die Kripke-Kleene Semantik umsetzt, berechnen kann. Diese haben wir für unser System realisiert. Es verwendet eine vierwertige Logik basierend auf einem Biverband. Da diese, wie bereits beschrieben, eine gewisse Loslösung von der Geschlossenen-Welt-Annahme ermöglicht, bot sich die Verbindung zu einem weiteren Forschungsbereich für „denkende“ Computer an: den Beschreibungslogiken.

Dabei handelt es sich um entscheidbare Untermengen der Prädikatenlogik erster Ordnung. Sie sind das zentrale Werkzeug zur Wissensmodellierung im *Semantic Web* und auch Grundlage vieler entsprechender Standards, allen voran die *Web Ontology Language (OWL)*. Beschreibungslogiken können im Gegensatz zu Logikprogrammen mit der Offenen-Welt-Annahme arbeiten. Dafür sind die mit ihnen modellierbaren Konzepte gegenüber den Regeln der Logikprogramme eingeschränkt. Von der Verbindung beider Paradigmen verspricht man sich Systeme die „best of both worlds“ bieten. Das Problem hierbei ist jedoch, sehr frei gesprochen, dass der Vorteil des einen Ansatzes genau darauf beruht, auf den Vorteil des anderen zu verzichten. Dennoch gibt es vielerlei erfolgreiche Ansätze, die diese Verbindung versuchen, und die sich grob in zwei Gruppen einteilen lassen. In der ersten wird versucht, ein einheitliches formales System aufzubauen, das Aspekte beider Ansätze vereint. Die zweite beschäftigt sich mit möglichen Schnittstellen zwischen Systemen, die auf den unterschiedlichen Paradigmen basieren. In diese Gruppe ist auch die vorliegende Arbeit einzuordnen. Wir ermöglichen es, aus dem auf Prolog

basierenden System Anfragen an ein auf Beschreibungslogiken aufbauendes System (einen sogenannten *DL-Reasoner*) weiterzuleiten. Die von uns verwendete vierwertige Logik ermöglicht dabei, auf solche Anfragen gelieferte Informationen elegant zu verarbeiten. Da den Beschreibungslogiken, wie bereits erwähnt, die Offene-Welt-Annahme zugrunde liegt, ist es dabei insbesondere wichtig, die Information „nichts bekannt“ verarbeiten zu können.

Der zweite Teil unserer Arbeit gliedert sich also wie folgt: In Kapitel 5 wird die Theorie der oben erwähnten Programmtransformation erarbeitet. In Kaptitel 6 beschreiben wir die Realisierung des Prolog-Systems und theoretische sowie praktische Aspekte der Anbindung eines auf Beschreibungslogiken basierenden Systems.

Schließlich findet sich in Kapitel 7 eine Zusammenfassung unserer Resultate. Danach präsentieren und diskutieren wir einige verwandte Arbeiten, bevor wir mit einem Ausblick auf weiterführende Forschungsansätze und bei der Anfertigung dieser Arbeit aufgekommene theoretische Fragestellungen schließen.

Kapitel 2

Grundlagen

Im folgenden Kapitel werden zuerst Biverbände eingeführt, die in unserer Arbeit als Wertebereich für die Wahrheitswerte von Logikprogrammen dienen. Sie wurden von Ginsberg in [Gin86, Gin88] beschrieben und bieten eine deutlich größere Ausdruckskraft für Semantiken als der klassische Wertebereich $\{true, false\}$ oder etwa partielle Wertebereiche wie $\{true, \bot, false\}$ und andere, die beispielsweise in [Kun88, Kun87, Fit85] verwendet werden. Dabei schließen Biverbände die eben genannten und andere bekannte Wertebereiche mit ein und stellen gleichzeitig ein elegantes mathematisches Fundament für die folgenden theoretischen Überlegungen dar. Im zweiten Teil sollen die hier betrachteten Logikprogramme erläutert werden. Da unsere Arbeit auf zwei Ansätzen basiert, die unterschiedliche Auffassungen von Interpretationen für Logikprogramme verwenden, führen wir beide ein: Interpretationen als Abbildungen und als Mengen. Dieses Kapitel orientiert sich in den Definitionen und Notationen weitgehend an der Arbeit von Loyer und Straccia [LS05].

2.1 Biverbände

2.1.1 Definitionen

Definition 1. *Ein* Verband $\langle V, \leq \rangle$ *ist eine nicht-leere Menge* V *mit einer partiellen Ordnung* $\leq$*, wobei jede zweielementige Teilmenge von* V *bezüglich* $\leq$ *ein Supremum und Infimum besitzt. Ein* vollständiger Verband *liegt dann vor, wenn jede Teilmenge von* V*, insbesondere auch unendliche, ein Supremum und Infimum besitzen. Die Notation* $v < w$ *wobei* $v, w \in V$ *steht für* $v \leq w$ *und* $v \neq w$*.*

Definition 2. *Sei* $\langle V, \leq \rangle$ *ein Verband. Ein Operator* $\texttt{op}$*:* $V \times V \to V$ *heißt* monoton *wenn gilt:* $x_1 \leq y_1$ *und* $x_2 \leq y_2$ *impliziert* $x_1 \ \texttt{op} \ x_2 \leq y_1 \ \texttt{op} \ y_2$*. Eine Abbildung* f*:* $V \to V$ *heißt* monoton *wenn gilt:* $x \leq y$ *impliziert* $f(x) \leq f(y)$*.*

Definition 3. *Ein* Biverband $\langle B, \leq_t, \leq_k \rangle$ *ist eine nichtleere Menge* B *mit zwei partiellen Ordnungen* $\leq_t$ *und* $\leq_k$*, die* B *jeweils die Struktur eines vollständigen Verbandes geben.*

Wir notieren mit $\sup_t, \inf_t, \sup_k, \inf_k$ *das Supremum und Infimum bezüglich der jeweiligen Ordnungen; des Weiteren schreiben wir*

- $x \vee y$ *für* $\sup_t\{x, y\}$
- $x \wedge y$ *für* $\inf_t\{x, y\}$
- $x \oplus y$ *für* $\sup_k\{x, y\}$
- $x \otimes y$ *für* $\inf_k\{x, y\}$

wobei $x, y \in B$*. Das größte Element in* B *bezüglich* $\leq_t$ *heißt* `true`*, das kleinste* `false`*, bezüglich* $\leq_k$ *heißt das größte Element* $\top$*, das kleinste* $\bot$*.*

Im folgenden wird $\leq_t$ *die* Wahrheitsordnung *(truth-order),* $\leq_k$ *die* Wissensordnung *(knowledge-order) genannt. Für größer (kleiner) bezüglich der Wahrheitsordnung verwenden wir auch die Begriffe* t-größer (t-kleiner) *und analog* k-größer (k-kleiner) *für die Wissensordnung.*

Es sei angemerkt, dass die erwähnten größten und kleinsten Elemente in einem vollständigen Verband – und damit auch in einem Biverband – immer existieren.

Definition 4. *Ein Biverband heißt* unendlich distributiv*, wenn alle Distributivgesetze für die Operatoren* $\wedge, \vee, \otimes, \oplus$ *gelten. Er heißt* unendlich verschränkt*, wenn alle eben genannten Operatoren monoton bezüglich beider Ordnungen sind.*

Definition 5. *Die* Negation *in einem Biverband ist ein Operator* $\neg$*, der die Wahrheitsordnung umkehrt, dabei die Wissensordnung nicht verändert und für den gilt* $\neg\neg x = x, \;\; x \in B$*.*

Wir nehmen an, dass alle in dieser Arbeit betrachteten Biverbände unendlich distributiv und verschränkt sind und eine Negation besitzen.

Die Wahrheitsordnung stellt eine Generalisierung der bekannten logischen Wahrheitswerte *true* und *false* mit den Junktoren *und* ($\wedge$) und *oder* ($\vee$) dar. Die Wissensordnung soll den Informationsgehalt einer Formel modellieren, wobei der Operator $\oplus$ die Informationen zweier Formeln kombiniert, während $\otimes$ die übereinstimmende Information zweier Formeln extrahiert.

2.1.2 Beispiele für Biverbände

Dies soll nun am Beispiel des einfachsten nicht trivialen Biverbandes $\mathcal{FOUR}$ (s. Abbildung 2.1.2) erläutert werden, der auf die Arbeit von Belnap [Bel77] zurückgeht. Die Werte `true` und `false` werden dabei äquivalent zu ihrer herkömmlichen Bedeutung verwendet. Es gibt mehrere Ansätze für partielle Modelle (etwa [Kun88, Kun87, Fit85]), die den Wertebereich um einen dritten Wert erweitern, der für *undefiniert* steht und hier durch $\bot$ symbolisiert ist. Dies entspricht der Definition als Minimum der Wissensordnung: Über eine Formel, der dieser Wahrheitswert zugewiesen wird, ist nichts bekannt. Vervollständigt wird der Biverband durch $\top$, das *überdefinierte* oder *widersprüchliche* Informationen darstellt. Als Maximum der Wissensordnung betrachtet stellt dies den Fall dar, dass für eine Formel sowohl bekannt ist, dass sie wahr ist, als auch dass sie falsch ist. Es ist bei Verwendung einer Logik basierend auf diesem (und den anderen) Biverbänden also möglich, mit Parakonsistenzen zu arbeiten, d.h. widersprüchliche Informationen so zu modellieren, dass nicht automatisch alles abgeleitet wird. So können Teile der Informationen in einem Modell dennoch genutzt werden, auch wenn dieses nicht komplett konsistent ist (vgl. hierzu auch [Ari02]).

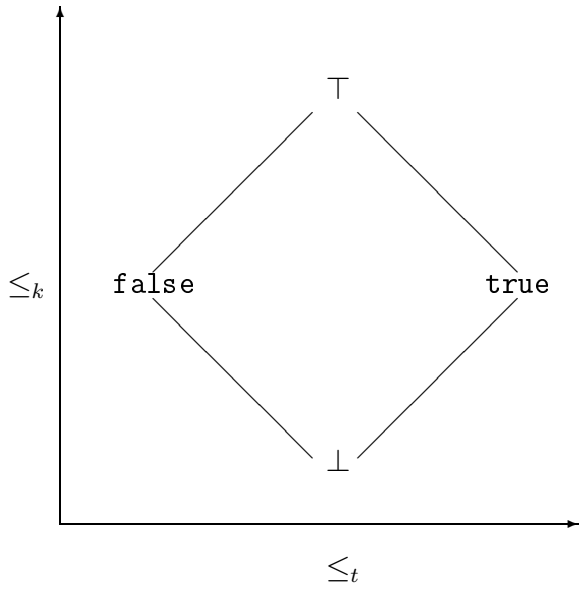

Abbildung 2.1: Der Biverband $\mathcal{FOUR}$

Ein weiterer für Logikprogramme genutzter Biverband wird von Ginsberg in [Gin88] vorgestellt. Es handelt sich hierbei um eine Struktur, die aus zwei Verbänden $\langle V_1, \leq_1 \rangle$ und $\langle V_2, \leq_2 \rangle$ gebildet wird. Der erste Verband stellt dabei das Vertrauen in die Wahrheit einer Formel dar, der zweite den Zweifel daran. Der Biverband $\langle V_1 \times V_2, \leq_t, \leq_k \rangle$ wird nun definiert durch die partiellen Ordnungen:

- $\langle x_1, x_2\rangle \leq_t \langle y_1, y_2\rangle$ wenn $x_1 \leq_1 y_1$ und $y_2 \leq_2 x_2$
- $\langle x_1, x_2\rangle \leq_k \langle y_1, y_2\rangle$ wenn $x_1 \leq_1 y_1$ und $x_2 \leq_2 y_2$

Das heißt, das Wissen nimmt zu, wenn sowohl Zweifel als auch Vertrauen steigen, die Wahrheit dagegen nimmt mit wachsendem Vertrauen aber sinkendem Zweifel zu.

Der letzte Biverband, der hier vorgestellt werden soll, wurde ebenfalls in Ginsbergs Arbeit [Gin88] angesprochen und von Fitting in [Fit91b] ausgearbeitet. Hier wird nicht versucht, exakte Werte anzugeben, sondern Intervalle, in denen sich die Wahrheitswerte befinden und sich somit unscharfen Logiken anzunähern. Zugrunde liegt der Verband $\langle [0,1], \leq\rangle$ bestehend aus dem Einheitsintervall und der üblichen Ordnung auf den rationalen Zahlen. Die Menge $[0,1] \times [0,1]$ wird als die Menge der geschlossenen Intervalle in $[0,1]$ aufgefasst und der Biverband $\langle [0,1] \times [0,1], \leq_t, \leq_k\rangle$ mit folgenden Ordnungen definiert:

- $\langle x_1, x_2\rangle \leq_t \langle y_1, y_2\rangle$ wenn $x_1 \leq_1 y_1$ und $x_2 \leq_2 y_2$
- $\langle x_1, x_2\rangle \leq_k \langle y_1, y_2\rangle$ wenn $x_1 \leq_1 y_1$ und $y_2 \leq_2 x_2$

Die Wahrheit einer Formel nimmt also mit größeren Werten in den Intervallen zu, während das Wissen mit kleineren Intervallen zunimmt, oder anders gesagt, die Informationen genauer werden.

Eine detailliertere und umfassendere Besprechung der Konstruktion von Biverbänden als Wertebereich für Logikprogramme findet sich neben [LS05] in [Fit02]. Für die Anwendung von Biverbänden in anderen Bereichen der Logik außerhalb der Logikprogrammierung sei auf die Arbeit von Arieli und Avron [AA98] verwiesen. Dort wird auch gezeigt, dass der Biverband $\mathcal{FOUR}$ eine ähnliche Rolle spielt, wie die zweiwertige Algebra für boolsche Algebren, nämlich dass prinzipiell alle Logiken, die auf Biverbände als Wertebereich aufsetzen, durch vierwertige Logiken charakterisiert werden können.

2.1.3 Das Theorem von Knaster-Tarski

Das Theorem von Knaster-Tarski, publiziert in [Tar55], stellt eines der wichtigsten Werkzeuge zur Arbeit mit Operatoren auf Verbänden dar.

Theorem 1. *Sei V ein vollständiger Verband und $f : V \to V$ eine* monotone *Funktion. Dann bilden die Fixpunkte von f ebenfalls einen vollständigen Verband.*

Die wichtigen Konsequenzen für die hier betrachteten Funktionen sind:

1. Jede monotone Funktion hat einen größten und kleinsten Fixpunkt.
2. Der kleinste Fixpunkt ergibt sich durch (transfinite) Iteration, beginnend beim Minimum des Verbandes.

2.2 Logikprogramme

2.2.1 Definition der formalen Sprache

Ein Logikprogramm basiert auf einer Menge an Symbolen $\mathcal{P}$ für Prädikate, $\mathcal{V}$ für Variablen, $\mathcal{C}$ für Konstanten und $\mathcal{F}$ für Funktionen.

Definition 6. *1. Ein* Term *wird induktiv wie folgt definiert:*

- *Die Elemente von $\mathcal{V}$ und $\mathcal{C}$ sind Terme.*
- *Der Ausdruck $f(t_1, \ldots, t_n)$ ist ein Term, wenn $f \in \mathcal{F}$ und alle $t_i,\ i = 1 \ldots n$ Terme sind.*

Ein Grundterm *ist ein Term, der nur aus Elementen von $\mathcal{F}$ und $\mathcal{C}$ besteht.*

2. Ein Atom *ist ein Ausdruck der Form $p(t_1, \ldots, t_m)$ wobei $p \in \mathcal{P}$ und $t_i,\ i = 1 \ldots m$ Terme sind. Zusätzlich erlauben wir die Elemente eines Biverbandes als Atome.*

3. Ein Literal *hat die Form A oder $\neg A$, wobei A ein Atom ist.*

4. Eine Formel *wird induktiv wie folgt definiert:*

- *Literale sind Formeln.*
- *Ausdrücke der Form φ_1 `op` φ_2 sind Formeln, wenn φ_1, φ_2 Formeln sind und `op` einer der Operatoren $\wedge, \vee, \otimes, \oplus$ eines Biverbandes ist.*
- *Die Ausdrücke $\forall\varphi$ und $\exists\varphi$ sind Formeln wenn φ eine Formel ist.*

5. Eine Regel *hat die Form $p(x_1, \ldots, x_k) \leftarrow \varphi(x_1, \ldots, x_k)$, wobei $p \in \mathcal{P}$, $x_1, \ldots, x_k \in \mathcal{V}$ und φ eine Formel ist. Das Atom $p(x_1, \ldots, x_k)$ heißt* Kopf*, die Formel $\varphi(x_1, \ldots, x_k)$* Körper *der Regel. Wir setzen voraus, dass die freien Variablen von φ in $\{x_1 \ldots x_k\}$ und universell quantifiziert sind.*

6. Ein Logikprogramm *P ist eine endliche Menge von Regeln.*

Man beachte, dass nicht alle Formen von Atomen als Kopf einer Regel zugelassen sind. Wir erlauben hier nur Prädikate ohne Funktionssymbole. Dies stellt jedoch keine Einschränkung dar, wie folgendes Beispiel demonstriert. Das Programm:

$$\begin{aligned} p(s(x)) &\leftarrow p(x) \\ p(0) &\leftarrow \texttt{true} \end{aligned}$$

kann wie folgt umformuliert werden:

$$\begin{aligned} p(y) &\leftarrow \exists x(eq(y, s(x)) \wedge p(x)) \\ p(y) &\leftarrow eq(y, 0) \end{aligned}$$

Das Prädikat *eq* modelliert dabei Gleichheit (vgl. hierzu [LS05] S.356).

Definition 7. *Gegeben sei ein Logikprogramm P. Das* Herbrand-Universum *des Programms P ist die Menge aller Grundterme, die aus den Funktions- und Konstantensymbolen in P bestehen. Die* Herbrand-Basis $\mathcal{B}_P$ *des Programms besteht aus den Grundatomen, das sind alle Atome, die sich aus den Prädikatensymbolen des Programms bilden lassen, indem jeder Variablen ein Grundterm zugewiesen wird. Mit ground(P) bezeichnen wir die* Grundinstanzenmenge *von P, also die Menge aller Regeln, die sich bilden lassen, indem man jedes Prädikatensymbol durch ein daraus gebildetes Grundatom ersetzt.*

Wenn ein Programm mindestens ein Funktionssymbol enthält, so wird das Herbrand-Universum und damit auch die Herbrand-Basis sowie die Grundinstanzenmenge unendlich. Auf die folgenden theoretischen Überlegungen hat dies keinen Einfluss. Für die Implementierung konkreter Anwendungen sind aber gewisse Einschränkungen zu treffen, um die Terminierung der Berechnungen zu garantieren. Häufig geschieht dies durch die Beschränkung auf Programme ohne Funktionssymbole.

2.2.2 Interpretationen und Modelle als Abbildungen

Definition 8. *Gegeben sei ein Biverband $\langle B, \leq_t, \leq_k \rangle$ und ein Logikprogramm P. Eine* Interpretation *des Logikprogramms ist eine Abbildung $I : \mathcal{B}_P \to B$. Für Formeln wird die Interpretation wie folgt induktiv fortgesetzt:*

- $I(b) = b$ *für* $b \in B$
- $I(\varphi_1 \ \mathtt{op} \ \varphi_2) = I(\varphi_1) \ \mathtt{op} \ I(\varphi_2)$ *für Formeln* φ_1, φ_2 *und* $\mathtt{op} \in \{\wedge, \vee, \oplus, \otimes\}$.
- $I(\neg\varphi) = \neg I(\varphi)$ *für eine Formel* φ.
- $I(\exists x\varphi(x) = \bigvee\{I(\varphi(t)) | t \text{ ist Grundterm}\}$ *und*
  $I(\forall x\varphi(x) = \bigwedge\{I(\varphi(t)) | t \text{ ist Grundterm}\}$ *für eine Formel* φ *und Variable* x.

Definition 9. *Die Menge aller Interpretationen $I(B)$ bildet ebenfalls einen Biverband $\langle I(B), \leq_t, \leq_k \rangle$. Dessen Wahrheits- und die Wissensordnung sind eine punktweise Fortsetzung der Ordnungen aus dem Biverband B:*

- $I_1 \leq_t I_2$ *gdw.* $I_1(A) \leq_t I_2(A)$ *für alle Grundatome* A.

- $I_1 \leq_k I_2$ *gdw.* $I_1(A) \leq_k I_2(A)$ *für alle Grundatome* A.

Für zwei Interpretationen werden die Operatoren wie folgt definiert:

$$(I_1 \ \mathtt{op} \ I_2)(\varphi) = I_1(\varphi) \ \mathtt{op} \ I_2(\varphi) \text{ wobei } \mathtt{op} \in \{\wedge, \vee, \oplus, \otimes\}$$

Mit I_f *bezeichnen wir die Interpretation, die allen Atomen* `false` *zuweist,* I_t *bildet entsprechend alle Atome nach* `true` *ab.* I_f *ist bezüglich der Wahrheitsordnung das kleinste,* I_t *das größte Element in* $I(B)$. *Analog bezeichnen wir mit* $I_\bot$ *und* $I_\top$ *das kleinste und größte Element bezüglich der Wissensordnung. Dabei handelt es sich um die Interpretationen, die jedes Atom nach* $\bot$ *beziehungsweise* $\top$ *abbilden.*

Um die Arbeit mit Interpretationen für Logikprogramme zu vereinfachen, führen wir eine Transformation der betrachteten Programme ein, die wir mit P^* bezeichnen. Jede Semantik, die wir für ein Programm P^* definieren, lässt sich jedoch auch für das zugehörige Programm P formulieren [Fit02].

Definition 10. *Gegeben sei ein Logikprogramm* P *und eine Interpretation* H. *Mit* P^* *bezeichnen wir das Programm, das folgende Regeln enthält:*

- $A \leftarrow \varphi_1 \vee \varphi_2 \vee \ldots$ *wenn* $A \leftarrow \varphi_1, A \leftarrow \varphi_2, \ldots$ *alle Elemente von* $ground(P)$ *mit Kopf* A *sind.*

- $A \leftarrow H(A)$ *wenn* A *nicht als Kopf einer Regel in* $ground(P)$ *vorkommt.*

Hier handelt es sich um eine Erweiterung der Konstruktion, wie sie z.B. in [Cla78] oder [Fit02] beschrieben wird. Dort wird statt $A \leftarrow H(A)$ die Regel $A \leftarrow$ `false` eingefügt, was der Geschlossenen-Welt-Annahme entspricht. Durch die Einführung einer beliebigen Interpretation H wird die Möglichkeit gegeben, flexiblere Modelle mit anderen Grundannahmen zu schaffen, wie sie in den folgenden Kapiteln verwendet werden.

Definition 11. *Eine Interpretation* I *ist* Modell *eines Logikprogramms* P *genau dann, wenn für alle* $A \leftarrow \varphi \in P^*$ *gilt* $I(A) = I(\varphi)$. *Wir schreiben hierfür* $I \models P$.

Obige Definition folgt der Idee der Clark-Completion aus [Cla78]. Der bekannte Modellbegriff entspräche eher der Forderung, dass lediglich $I(A) \leq_t I(\varphi)$ gelten muss. Die Idee hinter der hier verwendeten Definition eines Modells ist eine implizite Minimierung auf der Wahrheitsordnung. Man kann Definition 11 auch wie folgt schreiben:
$I \models P$ genau dann, wenn $I = \min_t\{J | I(\varphi) \leq_t J(A), \text{für alle } A \leftarrow \varphi \in P^*\}$
wobei mit $\min_t$ das Minimum bezüglich der Wahrheitsordnung bezeichnet wird [LS04].

2.2.3 Interpretationen und Modelle als Mengen

Eine zentrale Quelle unserer Arbeit, [LS05] von Loyer und Straccia, verwendet die im letzten Abschnitt beschriebene Auffassung von Interpretationen als Abbildungen. Für unsere Arbeit werden Interpretationen ebenfalls als Abbildungen aufgefasst, wenn nichts anderes festgelegt wird. Daneben ist es auch üblich, Interpretationen als Mengen von Literalen zu betrachten. Diese Form wird in dem zweiten für unsere Arbeit wichtigen Artikel [HW05] von Hitzler und Wendt verwendet und soll im folgenden Abschnitt präsentiert werden.

Definition 12. *Die Menge der Literale wird in zwei Mengen unterteilt, die positiven Literale, die der Menge $\mathcal{B}_P$ entsprechen und die negativen Literale, die aus der Menge der negierten Grundatome besteht und mit $\neg.\mathcal{B}_P$ bezeichnet wird.(Die Notation $\neg.X$ wird in dieser Arbeit analog dazu als Negation aller Atome einer Menge $X \subseteq \mathcal{B}_P$ verwendet.) Eine* Interpretation in Mengenschreibweise *ist dann eine Teilmenge $I \subseteq \mathcal{B}_P \cup \neg.\mathcal{B}_P$. Sie heißt* total, *wenn jedes Atom $A \in \mathcal{B}_P$ in positiver oder negativer Form in I enthalten ist, andernfalls heißt sie* partiell. *Sie heißt* konsistent, *wenn kein Atom in positiver und negativer Form darin enthalten ist.*

Für klassische zweiwertige Logiken wird gefordert, dass eine Interpretation eine konsistente Menge ist.

Wir zeigen den Zusammenhang zwischen den beiden Auffassungen von Interpretationen anhand von $\mathcal{FOUR}$. Bei größeren Biverbänden gestaltet sich dies schwierig, weshalb sich die Auffassung als Abbildung hier als deutlich praktischer erweist. Sei nun I_b eine Interpretation nach Definition 8 auf dem Biverband $\mathcal{FOUR}$ und I_m eine Interpretation in Mengenschreibweise. Nun entspricht $I_b(A) = \texttt{true}$ dem Fall $A \in I_m$ und $I_b(A) = \texttt{false}$ dem Fall $\neg A \in I_m$. Wir wollen jedoch eine vierwertige Logik, die auch Inkonsistenzen darstellen kann: $I_b(A) = \top$ entspricht also $A \in I_b$ und $\neg A \in I_b$. Schließlich soll auch die Tatsache, dass nichts bekannt ist, dargestellt werden, was dem Fall entspricht, dass ein Atom weder positiv noch negativ in I_b vorkommt. Es handelt sich dann um eine *partielle Interpretation* wozu eine Vielzahl von Arbeiten existiert. Für eine Überblick verweisen wir auf [Fit02]. Hier gilt die Entsprechung zwischen $I_b(A) = \bot$ und dem Fall, dass sowohl $A \notin I_b$ als auch $\neg A \notin I_b$. Aus diesen Korrespondenzen ergibt sich, dass die Mengeninklusion $\subseteq$ der Wissensordnung in $\mathcal{FOUR}$ entspricht.

Kapitel 3

Semantiken für Logikprogramme

Im Lauf der Zeit wurden für Logikprogramme und das eng verwandte Forschungsgebiet des nichtmonotonen Schließens eine Vielzahl verschiedener Semantiken entwickelt. Wie bereits einleitend erwähnt, ist die Arbeit von Hitzler und Wendt [HW05] dadurch motiviert, einen einheitlichen Zugang zu diesen mit verschiedensten Techniken arbeitenden Semantiken zu bieten. Wir werden im Folgenden nur diejenigen vorstellen, die eine wichtige Rolle für unsere Arbeit spielen, namentlich die *Kripke-Kleene-Semantik* und die *Wohlfundierte Semantik*. Danach wird anhand dieser der auf sogenannten *level-mappings* aufbauende Formalismus aus [HW05] vorgestellt. Schließlich werden im dritten Teil die *Jede-Welt-Annahme* und die darauf basierenden *H-fundierten Modelle* aus [LS05] vorgestellt, die eine Generalisierung der Wohlfundierten Semantik darstellen.

3.1 Die Kripke-Kleene Semantik

Diese Semantik wurde von Fitting in [Fit85] für eine dreiwertige Logik basierend auf $\{\texttt{true}, \bot, \texttt{false}\}$ entwickelt und in [Fit90, Fit91a] auf $\mathcal{FOUR}$ erweitert. Intuitiv gesprochen beschreibt sie das kleinste Modell eines Programms auf der Wissensordnung. Formal gesehen arbeitet sie mit einem sogenannten *immediate-consequence operator*, der auf die Arbeiten von Apt, van Emden und Kowalski [AvE82, vK76] zurückgeht. Die Idee dabei ist, dass die Regeln des Programms schrittweise auf ihre Auswirkungen bezüglich der Wahrheitswerte der Atome untersucht werden.

Definition 13. *Gegeben sei ein Logikprogramm P. Wir definieren den Operator Φ_P: Für eine Interpretation I und für jedes Grundatom A gilt:*

$$\Phi_P(I)(A) = I(\varphi) \text{ genau dann, wenn } A \leftarrow \varphi \in P^*.$$

Der hier verwendete Operator ist monoton in der Wissensordnung [Fit85] und hat folglich nach dem Theorem von Knaster-Tarski (s. Abschnitt 2.1.3) einen kleinsten Fixpunkt. Das ermöglicht folgende Definition:

Definition 14. *Das* Kripke-Kleene Modell *des Programms P ist der kleinste Fixpunkt von Φ_P bezüglich der Wissensordnung. P^* wird dabei mit $H = I_f$ gebildet.*

Aus der Definition des Operators Φ_P geht zudem folgendes Lemma hervor:

Lemma 1. *Die Fixpunkte von Φ_P sind genau die Modelle des Programms P nach Definition 11.*

Beweis. M ist Fixpunkt gdw. $M(A) = \Phi_P(M)(A) = M(\varphi)$ für alle Regeln $A \leftarrow \varphi \in P^*$. □

3.2 Die Wohlfundierte Semantik

Diese Semantik wurde von van Gelder, Ross und Schlipf in [vRS91] vorgestellt und basiert auf der *stable model semantics* von Gelfond und Lifschitz [GL88]. Ihre Motivation war, dass nicht für alle Programme ein sinnvolles Modell gefunden werden kann, wenn man keine partiellen Interpretationen als Modelle zulässt. Es sollte nun eine Semantik gefunden werden, die auch noch bei partiellen Modellen möglichst viele Informationen bieten kann und insbesondere aussagekräftiger als die Kripke-Kleene Semantik sein sollte. Dies wurde erreicht durch die getrennte Verarbeitung von positiven und negativen Literalen. Obwohl auch für diese Semantik eine Charakterisierung mittels monotonen Operatoren auf dem Biverband der Interpretationen als Abbildungen existiert (s. [Fit93]), zeigen wir hier die ursprüngliche Version, die Interpretationen in Mengenschreibweise (s. Kapitel 2.2.3) verwendet. So zeigt sich deutlicher die Struktur, deren Generalisierung zu den im Folgenden besprochenen H-fundierten Modellen geführt hat.

Definition 15. *Betrachten wir Interpretationen als konsistente Teilmengen von $\mathcal{B}_P \cup \neg.\mathcal{B}_P$, wobei partielle Interpretationen zugelassen sind. Des Weiteren seien die Rümpfe der Regeln in Programmen ausschließlich Konjunktionen von Literalen, die Regeln in P^* haben folglich alle die Form $A \leftarrow \varphi$ wobei $\varphi = \varphi_1 \vee \cdots \vee \varphi_n$ und $\varphi_i = L_{i_1} \wedge \cdots \wedge L_{i_m}$ mit $L_{i_k} \in \mathcal{B}_P \cup \neg.\mathcal{B}_P$. Wir nennen diese Rahmenbedingungen in Analogie zu [LS05]* klassische logische Umgebung.

Das zentrale Konzept der Wohlfundierten Semantik ist die sogenannte *Unfundierte Menge*, die alle Atome enthält, von denen problemlos angenommen werden kann, sie seien nicht wahr.

Definition 16. *Gegeben sei eine klassische logische Umgebung mit einem Programm P, sowie eine Interpretation I. Als* Unfundierte Menge *$U_P(I)$ für P bezüglich I bezeichnen wir eine Teilmenge $X \subseteq \mathcal{B}_\mathcal{P}$, wenn für jedes Atom A mit $A \in X$ mit $A \leftarrow \varphi \in P$ eine der folgenden Aussagen gilt:*

1. *Ein positives oder negatives Literal in φ ist falsch bezüglich I.*

2. *Ein (nicht negiertes) Atom in φ ist in X enthalten.*

Die beiden Punkte modellieren den Fall, dass entweder ein Literal als falsch interpretiert wird oder ein Atom nicht wahr werden kann, weil dazu ein bereits als falsch bekanntes Literal wahr werden müsste.

Für die positiven Literale wird ein *immediate-consequence operator,* ähnlich dem für die Kripke-Kleene Semantik vorgestellten, verwendet. In den hier zugrundeliegenden engeren Rahmenbedingungen ließe sich letzterer auch wie folgt definieren:

$$\Phi_P(I) = T_P(I) \cup \neg.F_P(I)$$

wobei $T_P(I)$ die Menge aller Atome $A \in \mathcal{B}_P$ ist, für die eine Regel $A \leftarrow \varphi$ in P existiert und φ bezüglich I wahr ist. $F_P(I)$ ist analog dazu die Menge der Atome für die φ bezüglich I falsch ist. Nun wird nur T_P benutzt, die positive Information wird also wie in der Kripke-Kleene Semantik behandelt, während die negative durch die Unfundierte Menge modelliert wird.

Definition 17. *Die* Wohlfundierte Semantik *eines Programms P ist der kleinste Fixpunkt bezüglich der Mengeninklusion des Operators*

$$W_P(I) = T_P(I) \cup \neg.U_P(I).$$

Dieser Operator ist monoton bezüglich der Mengeninklusion [vRS91] welche als Ordnung der Menge $\mathcal{B}_P \cup \neg.\mathcal{B}_P$ die Struktur eines vollständigen Verbandes gibt. Folglich ist nach dem Theorem von Knaster und Tarski die Existenz des kleinsten Fixpunktes garantiert.

3.3 Beispiel für Kripke-Kleene und Wohlfundierte Semantik

Wir wollen Anhand eines kleinen Beispiels den Unterschied zwischen Kripke-Kleene und wohlfundierter Semantik illustrieren. Betrachten wir dazu das folgende aussagenlogische Programm:

$$\begin{aligned} s &\leftarrow q \\ q &\leftarrow \neg p \\ p &\leftarrow p \end{aligned}$$

Die Kripke-Kleene Semantik liefert als einziges Ergebnis $I_\perp$, also den Ausgangspunkt der Fixpunktiteration, da es keine Regel gibt, die diese Interpretation in ihrem Rumpf beeinflusst und die Kripke-Kleene Semantik keine anderen Informationen verarbeitet. Die größte Unfundierte Menge für obiges Programm ist jedoch $\{p\}$. Deshalb erreicht die Iteration für die Wohlfundierte Semantik das Modell $\{\neg p, q, s\}$ in Mengenschreibweise.

3.4 Semantiken basierend auf *level-mappings*

Wir bleiben für diesen Abschnitt in einer klassischen logischen Umgebung und bei der Mengenschreibweise von Interpretationen. In [HW05] präsentieren Hitzler und Wendt einen Formalismus mit dem Ziel, die unterschiedlichen Semantiken, die für Logikprogramme existieren, einheitlich darzustellen, um deren Zusammenhänge besser untersuchen zu können. Ihr Ansatz basiert auf sogenannten *level-mappings*. Diese wurden bereits auf eine Reihe teilweise verwandter Probleme angewandt (s. [HW05] S. 85 für einen Überblick).

Definition 18. *Gegeben sei ein Logikprogramm P und eine Interpretation I. Ein* I-partielles level mapping *für P ist eine Abbildung $l : \mathcal{B}_P \to \alpha$ mit Definitionsmenge $dom(l) = \{A | A \in I \text{ oder } \neg A \in I\}$, wobei α eine abzählbare Ordinalzahl ist.*

Es wird also eine Ordnung auf den Grundatomen definiert, die abbilden soll, welche Abhängigkeiten zwischen den Atomen für die Anwendung einer Semantik eine Rolle spielen. Wir illustrieren nun die Beschreibung von Semantiken mittels *level-mappings* anhand der beiden im vorigen Abschnitt eingeführten Semantiken. Für weitere Beispiele und eine ausführliche Diskussion sei auf [HW05, Hit05] verwiesen.

Definition 19. *Sei P ein Logikprogramm, M ein Modell von P und l ein M-partielles level mapping für P. Wir sagen P erfüllt (F) bezüglich I und l, wenn für jedes Atom $A \in dom(l)$ eine der folgenden Aussagen gilt:*

(Fi) *$A \in I$ und es existiert eine Regel $A \leftarrow L_1 \wedge \cdots \wedge L_n$ in $ground(P)$, wobei für alle $i \in \{1, \ldots, n\}$ gilt: $L_i \in I$ und $l(A) > l(L_i)$.*

(Fii) *$\neg A \in I$ und für jede Regel $A \leftarrow L_1 \wedge \cdots \wedge L_n$ in $ground(P)$ existiert ein $i \in \{1, \ldots, n\}$ so dass $\neg L_i \in I$ und $l(A) > l(L_i)$.*

Das Kripke-Kleene Modell[1] von P ist nun das größte Modell M für das ein M-partielles level mapping l für P existiert, so dass P (F) bezüglich I und l erfüllt.

Betrachten wir nun die Charakterisierung der Wohlfundierten Semantik.

Definition 20. *Sei P ein Logikprogramm, M ein Modell von P und l ein M-partielles level mapping für P. Wir sagen P erfüllt (WF) bezüglich I und l, wenn für jedes Atom $A \in dom(l)$ eine der folgenden Aussagen gilt:*

(WFi) *$A \in I$ und es existiert eine Regel $A \leftarrow L_1 \wedge \cdots \wedge L_n$ in $ground(P)$, wobei für alle i gilt: $L_i \in I$ und $l(A) > l(L_i)$.*

(WFii) *$\neg A \in I$ und für jede Regel $A \leftarrow A_1 \wedge \cdots \wedge A_n \wedge \neg B_1 \wedge \cdots \wedge \neg B_m$ in $ground(P)$ ist mindestens eine der folgenden Bedingungen wahr:*

[1] In [HW05] wird diese als *Fitting model* bezeichnet.

1. *Es existiert ein* $i \in \{1, \ldots, n\}$ *so dass* $\neg A_i \in I$ *und* $l(A) \geq l(A_i)$.
2. *Es existiert ein* $j \in \{1, \ldots, m\}$ *so dass* $B_j \in I$ *und* $l(A) > l(B_j)$.

Das Wohlfundierte Modell von P ist nun das größte aller Modelle M, für das ein M-partielles *level-mapping* l für P existiert, so dass P die Bedingung (WF) bezüglich M und l erfüllt.

Wie die Charakterisierung durch monotone Operatoren erwarten lässt, zeigt sich auch hier, dass die positiven Literale in beiden Semantiken gleich behandelt werden: (Fi) und (WFi) sind identisch. Die genaueren Erkenntnisse über die negativen Literale durch die Verwendung der Unfundierten Menge, zeigen sich hier formal in der detaillierteren Bedingung (WFii), die den „Ursprung“ der negativen Information als Bedingung für jeweils andere Anforderungen stellt.

3.5 Die Jede-Welt-Annahme

Betrachten wir noch einmal die Charakterisierung der Wohlfundierten Semantik aus Definition 17: $W_P(I) = T_P(I) \cup \neg.U_P(I)$. Im Licht der in Abschnitt 2.2.3 gezeigten Korrespondenz – insbesondere der Tatsache, dass die Menge $\mathcal{B}_P \cup \neg.\mathcal{B}_P$ mit der Mengeninklusion $\subseteq$ einen Verband bildet, wobei $\subseteq$ der Wissensordnung aus $\mathcal{FOUR}$ entspricht – ist die Vereinigung der Mengen die Kombination der Informationen, die wir allgemein mit $\oplus$ notieren. Es ergibt sich also folgende „Übersetzung“ der Definition: $W_P(I) = T_P(I) \oplus \neg.U_P(I)$. Diese zeigt, dass die Unfundierte Menge die negative Information beiträgt, oder anders gesagt: es handelt sich hier um die Kripke-Kleene Semantik, die mit den Informationen der Unfundierten Menge ergänzt wird. Das Konzept der Unfundierten Menge beruht nun auf der in der Logikprogrammierung gebräuchlichen Geschlossenen-Welt-Annahme: Was nicht als wahr bekannt ist, wird als falsch angenommen. Genauer gesagt, weist die Wohlfundierte Semantik aufgrund der Unfundierten Menge jedem Atom, das nicht wahr werden kann, den Wert falsch zu. Die grundlegende Idee von Loyer und Straccia in [LS05] ist nun, hier flexiblere Annahmen basierend auf einem beliebigen Biverband machen zu können.

3.5.1 Hypothese und Support

Diese *Grundannahme* wird in Form einer Interpretation H ausgedrückt, die auch *Hypothese* genannt wird. Sie weist jedem Atom in $\mathcal{B}_P$ einen beliebigen Wahrheitswert aus einem Biverband zu. Die bekannte Geschlossene-Welt-Annahme entspräche dann der Hypothese: $H(A) = \texttt{false}$ für alle $A \in \mathcal{B}_P$ oder kürzer $H = I_f$. Wir werden im Folgenden betrachten, wie die semantischen Operatoren abgeändert werden können, um diese Grundannahme zu

berücksichtigen. Das betrifft aber nur Atome, die als Kopf einer Regel vorkommen und folglich durch die semantische Interpretation dieser Regel einen Wert zugewiesen bekommen. Um auch die Atome, die nicht im Kopf einer Regel vorkommen, in die Grundannahme einschließen zu können, arbeiten alle Semantiken, die im Folgenden betrachtet werden, auf P^* wie in Definition 10 festgelegt. Jetzt zeigt sich der Sinn der bei dessen Konstruktion verwendeten Interpretation H. Sie legt die Grundannahme für genau die Atome fest, die nicht in Köpfen vorkommen. Folglich wurde in Arbeiten die P^* verwenden, aber auf der Geschlossenen-Welt-Annahme basieren (siehe etwa [Fit02, Cla78]), hier die Regel $A \leftarrow \texttt{false}$ statt $A \leftarrow H(A)$ eingefügt.

Um eine beliebige Grundannahme in die Semantik einzubringen, wird der sogenannte *Support* verwendet. Es handelt sich hier um die Generalisierung der Idee der Unfundierten Menge. So wie letztere die Atome bestimmt, die problemlos nach der Geschlossenen-Welt-Annahme mit falsch bewertet werden können, modelliert der Support die Atome, die nach der Grundannahme bewertet werden können oder anders gesagt, die Menge an Information, die aus der Grundannahme in die semantische Interpretation eingebracht werden kann.

Nehmen wir also an, wir haben bereits eine Interpretation I für ein Programm P. Diese wollen wir nun mit Informationen aus einer Hypothese H ergänzen. Im Normalfall wird es nun nicht möglich sein, unter Berücksichtigung von I das ganze Wissen aus H hinzuzufügen. Wir wollen also eine Interpretation $J \leq_k H$ bestimmen, so dass $J(A)$ nicht mehr Wissen enthält als eine Semantik basierend auf dem gemeinsamen Wissen $I \oplus J$ dem Atom A aufgrund der Regeln in P zuweisen könnte. Formal ausgedrückt heißt das, dass für jede Regel $A \leftarrow \varphi \in P^*$ gelten sollte: $J(A) \leq_k (I \oplus J)(\varphi)$. Erinnern wir uns an die Definition der Kripke-Kleene Semantik, so können wir den Operator Φ benutzen, der die Interpretation eines Kopfes mit der Interpretation des entsprechenden Rumpfes ersetzt und erhalten folgende Schreibweise: $J(A) \leq_k \Phi_P(I \oplus J)(A)$ sowie folgende Definition:

Definition 21. *Gegeben sei ein Logikprogramm P, eine Interpretation I und eine Hypothese H. Eine Interpretation J heißt* sicher *bezüglich P, I und H, wenn folgende Bedingungen gelten:*

1. $J \leq_k H$,
2. $J \leq_k \Phi_P(I \oplus J)$.

Da die Hypothese H jede beliebige Interpretation sein kann, spricht man in Anlehnung an die Offene- und Geschlossene-Welt-Annahme von der *Jede-Welt-Annahme*.

Dass es sich bei sicheren Interpretationen um eine Generalisierung der Unfundierten Menge handelt, zeigt folgende Aussage, die in [LS05] bewiesen wird:

Theorem 2. *In einer klassischen logischen Umgebung und unter der Hypothese $H = I_f$ sind sichere Interpretationen äquivalent zu Unfundierten Mengen.*

Da möglichst viel Information aus der Hypothese benutzt werden soll, interessiert insbesondere die k-größte sichere Interpretation:

Definition 22. *Gegeben sei ein Logikprogramm P, eine Interpretation I und eine Hypothese H. Der* Support *von P bezüglich I und H ist die auf der Wissensordnung größte sichere Interpretation bezüglich P, I und H. Wir schreiben dafür $s_P^H(I)$.*

Wir zeigen nun, dass die Existenz dieser größten sicheren Interpretation immer garantiert ist.

Beweis. (nach [LS05] S.364) Betrachten wir die Menge X aller sicheren Interpretationen bezüglich P, I und H (Notation wie in Definition 22). Wie in Abschnitt 9 beschrieben, bilden die Interpretationen ebenfalls einen Biverband. Deshalb gilt $sup_k(X) = \bigoplus_{J \in X} J$. Wir bezeichnen diese Interpretation mit $\bar{J} = sup_k(X)$. Nachdem alle Elemente von X sichere Interpretationen sind, gilt $J \leq_k \Phi_P(I \oplus J)$ für alle $J \in X$. Per Definition gilt $J \leq_k \bar{J}$. Zudem wissen wir, dass Φ und der Operator $\oplus$ monoton bezüglich der Wissensordnung sind. Daraus folgt $J \leq_k \Phi_P(I \oplus \bar{J})$ für alle $J \in X$. Also ist $\Phi_P(I \oplus \bar{J})$ eine obere Schranke von X bezüglich der Wissensordnung. H ist ebenfalls eine obere Schranke, da für alle $J \in X$ aufgrund der Definition von sicheren Interpretationen gilt $J \leq_k H$. Da $\bar{J}$ die kleinste obere Schranke von X ist, gilt folglich $\bar{J} \leq_k \Phi_P(I \oplus \bar{J})$ sowie $\bar{J} \leq_k H$. $\bar{J}$ ist also sicher. □

Der Support lässt sich ebenfalls als Fixpunkt eines auf der Wissensordnung monotonen Operators bestimmen. Für die vorliegende Arbeit spielt diese Charakterisierung jedoch keine Rolle. Es sei deshalb auf [LS05] verwiesen. Dort wird auch folgendes wichtige Theorem bewiesen:

Theorem 3. *(entspricht Theorem 9 aus [LS05]) Der Support-Operator $s_P^H(I)$ ist bezüglich der Wissensordnung monoton in den Argumenten I und H.*

3.5.2 H-fundierte Modelle

Die Modelle nach der nun zu definierenden Semantik sollen nun analog zur Wohlfundierten Semantik mit dem Wissen aus sicheren Interpretationen ergänzt werden. Insbesondere interessant sind Modelle, die alles Wissen enthalten, das aus der gegebenen Hypothese angewendet werden kann, die also den Support bezüglich des betreffenden Modells einschließen.

Definition 23. *Sei P ein Logikprogramm und H eine Hypothese. Eine Interpretation M heißt* H-fundiertes Modell *von P genau dann, wenn $M \models P$ und $s_P^H(M) \leq_k M$.*

Diese Modelle lassen sich ebenfalls mittels eines monotonen Operators auf dem Biverband der Interpretationen charakterisieren. Er beruht auf folgendem Theorem:

Theorem 4. *(entspricht einem Teilergebnis aus [LS05], Theorem 12) Sei P ein Logikprogramm, M eine Interpretation und H eine Hypothese. M ist ein H-fundiertes Modell von P genau dann wenn $M = \Phi_P(M) \oplus s_P^H(M)$.*

Beweis. (nach [LS05] S.377): Nehmen wir an, M sei ein H-fundiertes Modell von P. Dann gilt nach der Definition 11 eines Modells, $M(A) = M(\varphi)$ für alle Regeln $A \leftarrow \varphi \in P^*$. Nach Lemma 1 folgt daraus $M(A) = \Phi_P(M)(A)$ für alle $A \leftarrow \varphi \in P^*$ oder kürzer $M = \Phi_P(M)$. Nach der Definition der H-fundierten Modelle gilt $s_P^H(M) \leq_k M$ und damit $M = \Phi_P(M) \oplus s_P^H(M)$.

Nehmen wir nun an, es gilt $M = \Phi_P(M) \oplus s_P^H(M)$. Daraus folgt sofort $s_P^H(M) \leq_k M$. Es bleibt also zu zeigen, dass $M \models P$. Da der Support eine sichere Interpretation ist folgt aus Definition 21 $s_P^H(M) \leq_k \Phi_P(M \oplus s_P^H(M))$. Wir wissen bereits, dass $s_P^H(M) \leq_k M$, damit ist $s_P^H(M) \oplus M = M$ und folglich $\Phi_P(M \oplus s_P^H(M)) = \Phi_P(M)$. Deshalb gilt $s_P^H(M) \leq \Phi_P(M)$ und damit $M = \Phi_P(M)$ und (wiederum mittels Lemma 1) $M \models P$. □

Definition 24. *Gegeben sei ein Logikprogramm P, eine Interpretation I und eine Hypothese H. Wir definieren den Operator $\tilde{\Pi}_P^H(I) = \Phi_P(I) \oplus s_P^H(I)$.*[2]

Es folgt nun direkt aus Theorem 4, dass die H-fundierten Modelle die Fixpunkte des Operators $\tilde{\Pi}_H^P$ sind. Da dieser ausschließlich aus bezüglich der Wissensordnung monotonen Operatoren zusammengesetzt ist, ist er selbst monoton, was auch die Existenz eines k-kleinsten H-fundierten Modells nach dem Theorem von Knaster-Tarski garantiert.

3.5.3 Offene- und Geschlossene-Welt-Annahme

Wie bereits erwähnt liegt der Logikprogrammierung normalerweise die Geschlossene-Welt-Annahme zugrunde. Diese besagt, dass als falsch gilt, was nicht abgeleitet werden kann. So versucht Prolog beispielsweise beim Aufruf von `not(A)` das Atom `A` abzuleiten und wenn dies misslingt wird `not(A)` wahr.

[2] Die Notation mit Tilde $\sim$ auf $\tilde{\Pi}_P^H$ stammt aus [LS05], wo diese benutzt wird, um von dem ebenfalls dort eingeführten Operator $\Pi_P^H = \Phi_p(I \oplus s_P^H(I))$, der dieselben Fixpunkte besitzt, zu unterscheiden. Obwohl dieser für unsere Arbeit keine Rolle spielt, wollen wir mit dieser Notation konsistent bleiben.

Dieses Prinzip wird als *negation as failure* bezeichntet. Dem gegenüber steht die Offene-Welt-Annahme, die keine Aussagen macht, über Ausdrücke deren Wahrheitswert nicht ableitbar ist. Es gibt einige Bestrebungen, diese beiden Konzepte zusammenzubringen (s. die Diskussion in Kapitel 7.2). Auch die Jede-Welt-Annahme spielt hier eine Rolle. In [LS05] und auch in dieser Arbeit wird gesagt, dass für ein Atom A die Offene-Welt-Annahme gilt, wenn es unter der Hypothese $H(A) = \perp$ evaluiert wird. Die Geschlossene-Welt-Annahme wird, wie bereits gesehen, durch $H(A) = \texttt{false}$ dargestellt. Damit lässt sich etwa das Verhalten von $\neg A$ steuern. Gilt $H(A) = \perp$, so entspricht $\neg A$ der expliziten Negation, während für $H(A) = \texttt{false}$ das Ergebnis der *negation as failure* entpricht. Es ist jedoch zu beachten, dass sich hier die Formulierung Offene-Welt-Annahme darauf bezieht, dass für das jeweilige Atom bzw. die jeweilige Grundinstanz nichts voraussetzend angenommen wird. Sie bedeutet nicht offen im Sinne einer offenen Domäne oder der auch als offen bekannten Prädikatenlogik erster Stufe. Für beide Fälle ist bekannt, dass die Gültigkeit von Aussagen nicht entscheidbar ist. Ein mögliche Annäherung an diese Problematik findet sich in den Arbeiten von Heymanns, Van Nieuwnborgh und Vermeir [HVV05, HNV04] auf die ebenfalls in Kaptitel 7.2 nochmals eingegangen wird.

Die Begriffe Offene- und Geschlossene-Weltannahme beinhalten also unterschiedliche Aspekte und werden in der Literatur diesbezüglich nicht exakt gleich verwendet. Zwei Gesichtspunkte, die zur Vermeidung von Missverständnissen betrachtet werden können, sind folgende:

- Was sagt die Annahme über die Voraussetzungen bezüglich des Wahrheitswerts eines Atoms, d.h. was gilt, wenn dieser nicht oder nur ungenau ermittelt werden kann?
- Was sagt die Annahme über den Gültigkeitsbereich einer Aussage, also über die Domäne des Programmes oder der Wissensbasis?

Kapitel 4

H-fundierte Modelle beschrieben durch *level-mappings*

Die zentrale theoretische Fragestellung unserer Arbeit ist es, analog zu den in Kapitel 3.4 gezeigten Beschreibungen von Semantiken durch *level-mappings* auch die H-fundierten Modelle mittels *level-mappings* zu charakterisieren. Dieses Resultat soll im folgenden Kapitel vorgestellt werden.

4.1 Vorbereitende Überlegungen und Definitionen

Eine der Ideen von Hitzler und Wendt in [HW05], die im dort verwendeten Beweisschema zum Vorschein kommt, ist mittels des einem Atom zugewiesenen Levels nachzuvollziehen, bei welchem Schritt dessen Wahrheitswert während der Iteration eines Operators definiert wird. Da hier nun auf einem Biverband als Wertebereich gearbeitet wird, ist es möglich, dass während der Iteration von $\tilde{\Pi}_P^H$ einem Atom mehrmals ein anderer Wahrheitswert zugewiesen wird. Da $\tilde{\Pi}_P^H$ monoton bezüglich der Wahrheitsordnung ist, bilden diese Werte eine aufsteigende Kette. Um dies modellieren zu können, erweitern wir das *level-mapping* auf zwei Argumente und sind somit in der Lage anzugeben, wann welches Atom welchen Wert annimmt.

Definition 25. *Sei P ein Programm, $\mathcal{B}_P$ die entsprechende Herbrand-Basis, B ein Biverband und I eine Interpretation. Wir definieren ein* partielles zweiwertiges level-mapping *l auf I als $l : \mathcal{B}_P \times B \to \omega_1$, wobei ω_1 die kleinste überabzählbare Ordinalzahl ist und alle der folgenden Bedingungen gelten:*

1. *Für alle $(A, v) \in \mathrm{dom}(l)$ gilt $v \leq_k I(A)$.*
2. *Für alle $A \in \mathcal{B}_P$ ist die Abbildung $l(A, \cdot)$ injektiv.*
3. *Für alle (A, v), $(A, w) \in dom(l)$ gilt:*

   (a) *v und w sind vergleichbar bezüglich der Wissensordnung.*
   (b) *$v <_k w$ impliziert $l(A, v) < l(A, w)$.*

4. *Für alle $A \in \mathcal{B}_P$ gilt $l(A, \bot) = -1$.*

Um die Notation übersichtlicher zu gestalten, verwenden wir dabei -1 für die kleinste Ordinalzahl.

Um nun die auf einem bestimmten Level zugewiesenen Werte untersuchen zu können, betrachten wir Interpretationen, die bei einem gewissen Level „abgeschnitten“ sind; d.h. einem Atom wird nicht der endgültige Wert zugewiesen, sondern der, der ihm auf dem betrachteten Level zugeordnet ist.

Definition 26. *Gegeben sei eine Interpretation I, eine Ordinalzahl α und ein partielles zweiwertiges level-mapping auf I. Wir definieren die Interpretation I^α wie folgt: für alle $A \in \mathcal{B}_P$, $I^\alpha(A) = \max_k\{v | l(A, v) < \alpha\}$.*

Das verwendete Maximum auf der Wissensordnung existiert dabei grundsätzlich aufgrund von Definition 25, Bedingungen 2 und 3. In anderen Worten weist I^α einem Atom A den Wert v zu, so dass $l(A, v)$ maximal unterhalb von α wird. Damit sind wir nun in der Lage, die Charakterisierung des H-fundierten Modells zu beschreiben.

4.2 Die Bedingung (AW)

Definition 27. *Gegeben sei ein Programm P, eine Interpretation I, eine Hypothese H und ein partielles zweiwertiges level-mapping l auf I. Wir sagen I erfüllt (AW) bezüglich l, H und P, wenn für alle Ordinalzahlen α und für alle Atome A in P^* genau eine der folgenden Bedingungen erfüllt ist:*

(AWi) *$I^{\alpha+1}(A) = I^\alpha(\varphi)$.*

(AWii) *(AWi) gilt nicht und alle der folgenden Bedingungen sind erfüllt:*

1. *$I^{\alpha+1}(A) \leq_k H(A)$.*
2. *Es existiert eine Interpretation $\hat{I}^\alpha$ die die folgenden Bedingungen erfüllt:*
   (a) *$\hat{I}^\alpha(A) = v$ wenn $l(A, v) = \alpha$ und $v \leq_k H(A)$,*
   (b) *$\hat{I}^\alpha \leq_k H$,*
   (c) *$\hat{I}^\alpha \leq_k \Phi_P(I^\alpha \oplus \hat{I}^\alpha)$.*

(AWiii) *Weder (AWi) noch (AWii) gelten und alle der folgenden Bedingungen sind erfüllt:*

1. $I^{\alpha+1}(A) >_k H(A)$,
2. *Es existiert eine bezüglich* I^α *und* H *sichere Interpretation* S *so dass* $I^{\alpha+1}(A) = I^\alpha(\varphi) \oplus S(A)$.

Theorem 5. *Sei* P *ein Programm und* H *eine Hypothese. Das k-kleinste H-fundierte Modell* M *ist die k-größte Interpretation* I*, für die ein partielles zweiwertiges level-mapping* l *auf* I *existiert, so dass* I *(AW) bezüglich* l*,* H *und* P *erfüllt.*

Beweis. Wir zeigen:
1) Es existiert ein *level-mapping* l_M, so dass M (AW) bezüglich l_M und P erfüllt.
2) Gegeben eine Interpretation I und ein *level-mapping* l_I auf I, so dass I (AW) bezüglich l_I und P erfüllt, gilt $I \leq_k M$.

Wir definieren $l_M(A, v) := \alpha$ wobei α die kleinste Ordinalzahl ist, so dass $(\tilde{\Pi}_P^H \uparrow \alpha + 1) = v$. Anders gesagt, wenn ein Atom einen Wahrheitswert zum ersten Mal unter der Interpretation $(\tilde{\Pi}_P^H \uparrow \alpha + 1)$ annimmt, wird ihm der Level α zugewiesen. Folglich bildet $(\tilde{\Pi}_P^H \uparrow \alpha)$ nur auf Werte mit Level unter α ab, d.h. $l(A, v) < \alpha$ wenn $(\tilde{\Pi}_P^H \uparrow \alpha)(A) = v$. Deshalb gilt wegen der Monotonie von $\tilde{\Pi}_P^H$:

$$M^\alpha = (\tilde{\Pi}_P^H \uparrow \alpha). \tag{4.1}$$

zu 1)
Wir zeigen, dass für jede Ordinalzahl α, (AW) von M^α bezüglich l_M und den gegebenen H und P erfüllt wird. (Wir benutzen die Schreibweise aus Definition 25.)

Sei α eine beliebige Ordinalzahl und A ein Atom. Betrachten wir den Iterationsschritt $(\tilde{\Pi}_P^H \uparrow \alpha + 1)(A) = (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A)$. Es gibt nun drei möglich Fälle:
(Fall 1): $(\tilde{\Pi}_P^H \uparrow \alpha + 1)(A) = (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi)$.
(Fall 2): (Fall 1) gilt nicht und $(\tilde{\Pi}_P^H \uparrow \alpha + 1)(A) \leq_k H(A)$.
(Fall 3): keiner der obigen Fälle gilt.

(Fall 1): Mit (4.1) gilt $M^{\alpha+1}(A) = M^\alpha(\varphi)$ und folglich ist (AWi) erfüllt.

(Fall 2): Wir definieren die Interpretation $\hat{M}^\alpha$ und zeigen, dass diese die Bedingungen für die Interpretation erfüllt, deren Existenz von (AWii) 2 gefordert wird:

$$\text{Sei } \hat{M}^\alpha(A) = \begin{cases} (\tilde{\Pi}_P^H \uparrow \alpha + 1)(A) & \text{wenn } l(A, (\tilde{\Pi}_P^H \uparrow \alpha + 1)(A)) = \alpha \\ s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A) & \text{sonst} \end{cases}$$

Der erste Fall dieser Definition entspricht den Bedingungen aus Definition 27 (AWii) 2a). Der zweite Teil der Bedingung (AWii) 2a, der erfordert, dass

$\hat{M}^\alpha(A) \leq_k H(A)$, ist bereits durch die Bedingungen für (Fall 2) erfüllt. Diese garantieren auch, dass (AWii)2b für den ersten Fall der Definition von $\hat{M}^\alpha$ gilt. Für den zweiten Fall wird (AWii) 2b von $\hat{M}^\alpha$ aufgrund der Definition des Supports erfüllt.

Wir zeigen nun, dass auch (AWii)2c von $\hat{M}^\alpha$ erfüllt wird. Betrachten wir alle Atome A mit

$$\hat{M}^\alpha(A) = s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A).$$

Für diese gilt aufgrund der Definition des Supports:

$$\hat{M}^\alpha(A) = s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A) \leq_k (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi).$$

Für alle Atome B gilt $(\tilde{\Pi}_P^H \uparrow \alpha{+}1)(B) \geq_k s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(B)$. Damit wissen wir, dass $\hat{M}^\alpha(B) \geq_k s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(B)$ für alle Atome B und mit der Monotonie der in Formeln erlaubten Operatoren folgt daraus $\hat{M}^\alpha(\varphi) \geq_k s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi)$. Also gilt

$$\hat{M}^\alpha(A) \leq_k (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus \hat{M}^\alpha(\varphi).$$

und mit (4.1) folgt daraus

$$\hat{M}^\alpha(A) \leq_k M^\alpha(\varphi) \oplus \hat{M}^\alpha(\varphi).$$

Betrachten wir nun alle Atome A mit

$$\hat{M}^\alpha(A) = (\tilde{\Pi}_P^H \uparrow \alpha + 1)(A) = (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A).$$

Aus der Definition des Supports wissen wir

$$s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A) \leq_k (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi).$$

Trivialerweise gilt $(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \leq_k (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi)$. Mit der Monotonie des Operators $\oplus$ folgt daraus

$$(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A) \leq_k (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi).$$

Da $\oplus$ idempotent ist, gilt

$$\hat{M}^\alpha(A) \leq_k (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi).$$

Daraus folgt mit dem selben Argument wie oben

$$\hat{M}^\alpha(A) \leq_k M^\alpha(\varphi) \oplus \hat{M}^\alpha(\varphi).$$

Folglich ist (AWii) erfüllt.

(Fall 3): Wenn (Fall 1) nicht gilt, kann (AWi) nicht erfüllt sein, da diese mittels (4.1) äquivalent sind. Wenn (Fall 2) nicht erfüllt ist, das heißt $(\tilde{\Pi}_P^H \uparrow \alpha + 1)(A) >_k H(A)$, ist auch (AWii) 1 nicht erfüllt. Also sind weder

(AWi) noch (AWii) erfüllt. Es gilt also $(\tilde{\Pi}_P^H \uparrow \alpha+1)(A) >_k H(A)$. Mit (4.1) folgt $M^{\alpha+1} >_k H(A)$ und (AWiii) 1 ist erfüllt. Wir wissen, dass

$$(\tilde{\Pi}_P^H \uparrow \alpha+1)(A) = (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A)$$

und dass der Support in dieser Gleichung sicher bezüglich M^α ist. Folglich sei nun $S = s_P^H$ (wobei wir das Symbol S wie in (AWiii) benutzen), dann ist mit (4.1 (AWiii) 2 erfüllt.

zu 2)
Mittels transfiniter Induktion zeigen wir, dass $I^\alpha \leq_k M^\alpha$ für alle α.
Basisfall: I^0 bildet auf alle Werte mit Level -1 ab, weist also jedem Atom $\perp$ zu. Also gilt

$$I^0 = I_\perp = (\tilde{\Pi}_P^H \uparrow 0) = M^0$$

wobei wir (4.1) für die letzte Gleichheit benutzen.

Induktionsschritt für Stufenordinalzahlen:
Wir nehmen als Induktionshypothese an: $I^\alpha \leq_k M^\alpha$. Nun betrachten wir wiederum drei Fälle:
(Fall 1): (AWi) gilt. Dann ist $I^{\alpha+1}(A) = I^\alpha(\varphi)$. Mit der Induktionshypothese folgt $I^{\alpha+1}(A) \leq_k M^\alpha(\varphi)$ und mit (4.1)

$$\begin{array}{rcl} I^{\alpha+1}(A) & \leq_k & M^\alpha(\varphi) = (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \leq_k \\ & \leq_k & (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A) = \\ & = & (\tilde{\Pi}_P^H \uparrow \alpha+1)(A) = M^{\alpha+1}(A). \end{array}$$

(Fall 2): Für alle Werte $v = I^{\alpha+1}(A)$, die sich von $I^\alpha(A)$ unterscheiden, gilt $l(A,v) = \alpha$. Daraus folgt, dass $I^{\alpha+1}(A) = v = \hat{I}^\alpha(A)$ aufgrund von (AWii) 1 und (AWii) 2a. Wegen (AWii) 2b und 2c wissen wir, dass die Interpretation $\hat{I}^\alpha$ sicher bezüglich I^α ist. Also folgt aus der Tatsache, dass der Support die k-größte sichere Interpretation ist, dass

$$\hat{I}^\alpha(A) = I^{\alpha+1}(A) \leq_k s_P^H(I^\alpha)(A).$$

Mit der Induktionshypothese und der Monotonie des Support-Operators gilt dann

$$I^{\alpha+1}(A) \leq_K s_P^H(M^\alpha)(A).$$

Nun ist per Definition

$$(\tilde{\Pi}_P^H \uparrow \alpha+1)(A) = (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A).$$

Mit (4.1) folgt daraus

$$M^{\alpha+1}(A) \geq_k s_P^H(M^\alpha)(A)$$

und folglich

$$I^{\alpha+1}(A) \leq_k M^{\alpha+1}.$$

(Fall 3): (AWiii) gilt. Also ist

$$I^{\alpha+1}(A) = I^{\alpha}(\varphi) \oplus S(A).$$

Mit der Induktionshypothese folgt daraus

$$I^{\alpha+1}(A) \leq_k M^{\alpha}(\varphi) \oplus S(A).$$

Wir wissen, dass S sicher bezüglich I^α ist. Per Definition ist die k-größte derartige Interpretation $s_P^H(I^\alpha)$. Aus der Induktionshypothese und der Monotonie des Support-Operators wissen wir auch, dass $s_P^H(I^\alpha) \leq_k s_P^H(M^\alpha)$. Damit gilt

$$\begin{array}{rcl} I^{\alpha+1}(A) & \leq_k & M^{\alpha}(\varphi) \oplus S(A) \leq_k M^{\alpha}(\varphi) \oplus s_P^H(I^\alpha)(A) \leq_k \\ & \leq_k & M^{\alpha}(\varphi) \oplus s_P^H(M^\alpha)(A) = M^{\alpha+1}(A) \end{array}$$

und es folgt $I^{\alpha+1}(A) \leq_k M^{\alpha+1}(A)$.

Induktionsschritt für Limesordinalzahlen:

Sei α eine Limesordinalzahl und sei $I^\beta \leq_k M^\beta$ für alle $\beta < \alpha$ (Induktionshypothese). Nun folgt aus $I^\alpha(A) = v$ dass ein $\gamma < \alpha$ existiert, mit $l(A, v) = \gamma$. Also ist $I^{\gamma+1}(A) = v$. Mit der Induktionshypothese folgt

$$I^{\gamma+1}(A) \leq_k M^{\gamma+1}(A) = (\tilde{\Pi}_P^H \uparrow \gamma + 1)(A).$$

Da $(\tilde{\Pi}_P^H \uparrow \alpha) = \sup_k\{(\tilde{\Pi}_P^H \uparrow \beta) | \beta < \alpha\}$ und $\gamma + 1 < \alpha$, gilt

$$(\tilde{\Pi}_P^H \uparrow \gamma + 1)(A) \leq_k (\tilde{\Pi}_P^H \uparrow \alpha)(A)$$

oder mit (4.1)

$$M^{\gamma+1} \leq_k M^{\alpha}.$$

Also folgt $I^\alpha \leq_k M^\alpha$.

□

Obwohl in der Definition von (AW) das *level-mapping* nicht mehr direkt auftaucht, handelt es sich hier um eine Charakterisierung durch ein *level-mapping*. Dieses wird implizit durch die Interpretationen I^α verwendet. Man könnte im Licht von Definition 26 statt $I^\alpha(A) = v$ im Beweis auch $l(A, v) < \alpha$ schreiben. Dabei geht aber die Forderung nach Maximalität des Levels aus

Definition 26 verloren. Zudem ist der Support über Eigenschaften der ganzen Interpretation definiert und nicht über Eigenschaften einzelner Regeln (wie etwa die Unfundierte Menge), deshalb haben wir uns für die Betrachtungsweise mit den „abgeschnittenen" Interpretationen I^α entschieden. Wir werden im Abschnitt 4.4 noch näher auf die Beziehung zwischen (AW) und der Beschreibung der Wohlfundierten Semantik mittels *level-mappings* (WF), die in Kapitel 3.4 erläutert wurde, eingehen. Zuvor wollen wir jedoch zwei Korollare aus dem Beweis von Theorem 5 präsentieren.

4.3 Korollare

Korollar 1. *Wenn während der Iteration von* $\tilde{\Pi}_P^H$ *die Werte von* $(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi)$ *und* $s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A)$ *unvergleichbar bezüglich der Wissensordnung werden, dann gilt* $(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A) \geq_k H(A)$.

Beweis. Wir zeigen zuerst durch Widerspruch, dass das Gegenteil nicht eintreten kann: Nehmen wir an, dass $(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi)$ und $s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A)$ bezüglich der Wissensordnung unvergleichbar sind, aber

$$(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A) <_k H(A)$$

gilt. Wir wissen, dass (AW) von M erfüllt wird.

Nehmen wir nun für den Iterationsschritt $\alpha+1$ an, dass (AWi) erfüllt ist. Dann gilt

$$(\tilde{\Pi}_P^H \uparrow \alpha+1)(A) = (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A) = (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi),$$

was im Widerspruch zu der vorausgesetzen Unvergleichbarkeit steht.

Nehmen wir also an, dass (AWii) erfüllt ist. Dann ist

$$v = (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A) = \hat{M}^\alpha(A)$$

im Bild der bezüglich M^α sicheren Interpretation $\tilde{M}^\alpha$. Wir wissen per Definition, dass $s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)$ die k-größte solche Interpretation ist, also gilt

$$v = (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A) \leq_k s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A).$$

Daraus folgt $v = s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A)$, was der angenommenen Unvergleichbarkeit widerspricht.

Schließlich ist es nicht möglich, dass (AWiii) erfüllt ist, da dies erfordert $(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A) >_k H(A)$, was in direktem Widerspruch zur Annahme steht. Also kann (AW) nicht erfüllt werden.

Es bleibt zu zeigen, dass (AW) erfüllbar ist, bei Unvergleichbarkeit und $(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A) \geq_k H(A)$. Dies ist der Fall, da dann (AWiii) erfüllt wird, weil der Support per Definition sicher ist. □

Der Beweis dieses Korollars macht die Intuition hinter der Konstruktion von (AW) sichtbar. Insbesondere bei der Behandlung von (AWii) wird deutlich, dass diese Bedingung genau den Fall modelliert, in dem der Support-Operator in $\tilde{\Pi}_P^H$ den Wert der Interpretation liefert. Wird dagegen der Wert via *immediate consequence operator* übernommen – also der Wert des Rumpfes mittels $(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi)$ – so kommt (AWi) zum tragen. Für (AWiii) bleibt der Fall, dass die beiden Teile tatsächlich unvergleichbar bezüglich der Wissensordnung sind. Da die Iteration dabei, wie eben gezeigt, zwingenderweise über die Hypothese hinauswachsen muss, ergibt sich folgendes Resultat:

Korollar 2. *Für jedes Atom $A \in \mathcal{B}_P$ kann es während der Iteration von $\tilde{\Pi}_P^H$ nur einmal vorkommen, dass $(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi)$ und $s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A)$ unvergleichbar bezüglich der Wissensordnung werden.*

Beweis. Korollar 1 zeigt, dass im Falle der Unvergleichbarkeit der Werte von $(\tilde{\Pi}_P^H \uparrow \alpha)(\varphi)$ und $s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A)$ gilt

$$(\tilde{\Pi}_P^H \uparrow \alpha + 1)(A) = (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A) \geq_k H(A).$$

Da wir wissen, dass $s_P^H(\tilde{\Pi}_P^H \uparrow \alpha)(A) \leq_k H(A)$, folgt daraus

$$(\tilde{\Pi}_P^H \uparrow \alpha + 1)(A) = (\tilde{\Pi}_P^H \uparrow \alpha)(\varphi) \oplus H(A).$$

Da der Support-Operator monoton ist, können wir ihn deshalb in allen folgenden Iterationsschritten durch $H(A)$ ersetzen. Deren Resultate hängen dann nur noch vom Wert von $(\tilde{\Pi}_P^H \uparrow \beta)(\varphi)$ ab, wobei $\beta > \alpha + 1$. Die Monotonie des Operators $\tilde{\Pi}_P^H$ verhindert jedoch, dass dieser Wert jemals unvergleichbar zu $H(A)$ werden kann. □

4.4 (WF) als Sonderfall von (AW)

Die H-fundierten Modelle wurden von Loyer und Straccia aus der Analyse und Weiterentwicklung der Wohlfundierten Modelle konzipiert (s. dazu [LS03, LS05]). In dem Artikel [LS05] wird bewiesen, dass in einer klassischen logischen Umgebung und unter der Hypothese $H = I_f$ der Support die Unfundierte Menge darstellt und damit die H-fundierten Modelle die Wohlfundierten Modelle sind. Dabei basiert der Beweis auf den Eigenschaften der jeweiligen Fixpunktoperatoren. Wir wollen nun diese Beziehung in der Darstellung mittels *level-mappings* zeigen, genauer, dass wenn in einer klassischen logischen Umgebung und unter der Hypothese $H = I_f$ (AW) gilt, dann gilt auch (WF) aus Abschnitt 3.4.

Dabei ist zu beachten, dass die Bedingung (WF) bezüglich eines einwertigen, (AW) aber bezüglich eines zweiwertigen *level-mappings* definiert ist. In einer klassisch logischen Umgebung kann der einem Atom zugewiesene

Wahrheitswert sich jedoch nur einmal ändern: von $\bot$ auf `true` oder `false`. Der Wert $\top$ ist aufgrund der Forderung nach konsistenten Interpretationen in Definition 15 nicht möglich. Damit wird das *level mapping* de facto einstufig und der Wahrheitswert im Argument ist damit nicht mehr nötig. Wir benutzen also eine Projektion auf das erste Argument eines zweiwertigen *level-mappings* l:

Definition 28. *Gegeben sei ein zweiwertiges partielles level-mapping l auf einer Interpretation I. Wir definieren das einwertige I-partielle level-mapping $l^1 : \mathcal{B}_P \cup \neg.\mathcal{B}_P \to \omega_1$ wie folgt:*

$$l^1(L) = \begin{cases} l(L, \texttt{true}) & \textit{wenn } I(L) = \texttt{true} \\ l(L, \texttt{false}) & \textit{wenn } I(L) = \texttt{false} \end{cases}$$

Ausserdem ist (WF) für Interpretationen in Mengenschreibweise definiert, (AW) benutzt Interpretationen als Abbildungen. Wir werden im folgenden Beweis einen Teil der in Abschnitt 2.2.3 beschriebene Analogie benutzen:

Sei I_b eine Interpration als Abbildung aufgefasst, I_m als Menge. Dann gilt:

- $I_b(A) = \texttt{true}$ entspricht $A \in I_m$.
- $I_b(A) = \texttt{false}$ entspricht $\neg A \in I_m$.
- $I_b(A) = \bot$ entspricht $A \notin I_m$ und $\neg A \notin I_m$.

Für weitere Erläuterungen dazu siehe [Fit91a, Fit02].

Theorem 6. *Wir setzen eine klassische logische Umgebung voraus. Erfüllt eine Interpretation I die Bedingung (AW) bezüglich eines partiellen zweiwertigen level-mappings l, der Hypothese $H = I_f$ und eines Programmes P, dann erfüllt P die Bedingung (WF) bezüglich I und l^1.*

Beweis. Angenommen I erfüllt (AW) bezüglich l, $H = I_f$ und P.

Nehmen wir des Weiteren an, dass $l(A, I(A)) = \alpha$ und $I(A) = \texttt{true}$. Daraus folgt $I^{\alpha+1}(A) = \texttt{true}$ nach Definition von I^α. Dann muss (AWi) gelten, da (AWii) erfordert, dass $I^{\alpha+1} \leq_k H(A) = \texttt{false}$ und (AWiii) erfordert $I^{\alpha+1} > H(A) = \texttt{false}$, was beides nicht erfüllt ist. Aus (AWi) folgt, dass $I^\alpha(\varphi) = \texttt{true}$. Also gilt für alle Literale $L_i, i = 1 \ldots n$ in φ dass $I^\alpha(L_i) = \texttt{true}$. Nach der Definition von I^α folgt daraus $l(L_i, \texttt{true}) < \alpha$ und folglich $l^1(L_i) < \alpha$. Des Weiteren ergibt sich aus der Monotonieeigenschaften von l (Definition 25. 3) und der Tatsache, dass unter den gegebenen Bedingungen `true` keinen Nachfolger auf der Wissensordnung hat, dass aus $I^\alpha(L_i) = \texttt{true}$ auch $I(L_i) = \texttt{true}$ folgt. Dies entspricht $L_i \in I$ und damit gilt (WFi).

Nehmen wir nun an $I(A) = \texttt{false}$. Es gelte weiterhin $l(A, I(A) = \alpha$.

1. Es gelte (AWi). Folglich ergibt sich $I^{\alpha+1}(A) = \texttt{false} = I^{\alpha}(\varphi)$. Also wissen wir für mindestens ein Literal L_i in φ dass $I^{\alpha}(L_i) = \texttt{false}$. Sei nun $\varphi = B_1 \wedge \cdots \wedge B_k \wedge \neg C_1 \wedge \cdots \wedge \neg C_l$. Wenn $L_i = B_j, 1 \leq j \leq k$, dann ist $I^{\alpha}(B_j) = \texttt{false}$ und $l(B_j, \texttt{false}) < \alpha$. Also gilt $l^1(B_j) < \alpha$. Ausserdem folgt $I(B_j) = \texttt{false}$ aus $I^{\alpha}(B_j) = \texttt{false}$. Dies gilt analog zum vorigen Fall aufgrund der Monotonieeigenschaften von l und der Tatsache, dass auch $\texttt{false}$ keinen Nachfolger auf der Wissensordnung hat. Damit gilt auch $\neg B_j \in I$. Also ist (WFii 1) erfüllt.

Wenn $L_i = \neg C_j, 1 \leq j \leq l$ dann gilt $I^{\alpha}(C_j) = \texttt{true}$ und $l(C_j, \texttt{true}) < \alpha$. Also ist $l^1(C_j) < \alpha$. Ausserdem gilt wieder $I(C_j) = \texttt{true}$ weil $I^{\alpha}(C_j) = \texttt{true}$ und damit auch $C_j \in I$. Also ist (WFii 2) erfüllt.

2. Es gelte (AWii). Dann erfordert (AWii) 2a die Existenz einer Interpretation $\hat{I^{\alpha}}$ wobei $\hat{I^{\alpha}}(A) = \texttt{false}$ immer wenn $l(A, \texttt{false}) = \alpha$. Dann impliziert (AWii) 2c, dass auch $\Phi_P(I^{\alpha} \oplus \hat{I^{\alpha}})(A) = I^{\alpha}(\varphi) \oplus \hat{I^{\alpha}}(\varphi) = \texttt{false}$ da $\top$ kein gültiger Wahrheitswert in der vorausgesetzten klassichen Logischen Umgebung ist. Es ist weiterhin nicht möglich, dass $I^{\alpha}(\varphi) = \texttt{false}$, weil dann (AWi) gelten würde und die Unterbedingungen von (AW) sich per Definition gegenseitig ausschließen. Also gilt $\hat{I^{\alpha}}(\varphi) = \texttt{false}$. Daraus folgt $\hat{I^{\alpha}}(L_i) = \texttt{false}$ für ein Literal L_i in φ wobei wiederum $\varphi = B_1 \wedge \cdots \wedge B_k \wedge \neg C_1 \wedge \cdots \wedge \neg C_l$. Da (AWii) 2b erfordert, dass $\hat{I^{\alpha}} \leq_k H = \texttt{false}$, kann $\hat{I^{\alpha}}$ keinem Literal den Wert $\texttt{true}$ zuweisen. Also ist $L_i = B_j, 1 \leq j \leq k$ und $\hat{I^{\alpha}}(B_j) = \texttt{false}$. Wir wissen, dass sich der Wert von B_j auf dem aktuellen level α geändert haben muss, da wir bereits wissen, dass $I^{\alpha}(\varphi) \neq \texttt{false}$. Also gilt $I^{\alpha+1}(B_j) = \texttt{false}$ und $l(B_j, \texttt{false}) = \alpha$. Folglich ist $l^1(B_j) = \alpha$. Zudem folgt aus $I^{\alpha+1}(B_j) = \texttt{false}$ dass $I(B_j) = \texttt{false}$. Damit gilt (WFii 1). □

Es zeigt sich, dass (AWi) die Fälle einschließt, für die in Bedingung (WFii 1) der Level des Kopfes echt kleiner ist als der des betreffenden Literals, während (AWii) die Fälle modelliert die die Gleichheit in (WFii 1) nötig machen. Dies ist ein weiteres Beispiel für die enge Verwandschaft von Unfundierter Menge und Support, da (AWii) den Teil der Interpretation erfasst, der vom Support im Operator $\tilde{\Pi}_P^H$ beigetragen wird, genauso wie (WFii) den Teil der Interpretation erfasst, der auf der Unfundierten Menge beruht.

Kapitel 5

Die KKS-Transformation

In diesem Kapitel soll gezeigt werden, wie Semantiken, die in gewissem Sinne „offener“ sind, als die Kripke-Kleene Semantik durch diese modelliert werden können. Dies geschieht mit Hilfe einer Programmtransformation, die durch die Beschreibung der Kripke-Kleene-Semantik als Spezialfall der H-fundierten Semantik motiviert ist.

5.1 Kripke-Kleene-Modelle als Spezialfall von H-fundierten Modellen

Betrachten wir die Definition 27 von (AW) unter der Annahme $H = I_\perp$. (AWi) bleibt dabei unverändert. (AWii) greift aufgrund der Bedingung 1: $I^{\alpha+1}(A) \leq_k H(A)$ nur noch für $I^{\alpha+1} = \perp$. In diesem Fall erfüllt $I_\perp$ die Bedingungen für $\hat{I}^\alpha$ aus (AWii) 2. Da $I^{\alpha+1} = \perp$ und $I_\perp$ jedem Atom $\perp$ zuweist, ist die Gleichheit aus (AWii) 2a gegeben. Die Bedingungen (AWii) 2b und (AWii) 2c gelten, weil $I_\perp$ die kleinste Interpretation bezüglich der Wissensordnung ist. Deshalb ist auch für (AWiii) die Bedingung (AWiii) 1 immer erfüllt. (AWiii) 2 verlangt eine sichere Interpretation. Die einzige sichere Interpretation unter $H = I_\perp$ ist jedoch $I_\perp$ selbst. Dies folgt direkt aus der Definition von sicheren Abbildungen, die verlangt, dass diese k-kleiner als H sind. Deshalb wird gefordert $I^{\alpha+1}(A) = I^\alpha(\varphi) \oplus \perp = I^\alpha(\varphi)$. (AWiii) ist also identisch mit (AWi). Die H-fundierten Modelle für $H = I_\perp$ lassen sich also komplett durch (AWi) charakterisieren.

Dies entspricht dem Resultat, das sich aus der Beschreibung mittels des Fixpunktoperators $\tilde{\Pi}_P^H$ ableiten lässt. Da wie eben erwähnt, die einzige sichere Interpretation unter $H = I_\perp$ genau $I_\perp$ ist, ist sie auch die größte sichere Interpretation, also der Support, egal bezüglich welcher Interpretation $I_\perp$. Damit wird $\tilde{\Pi}_P^{I_\perp}(I) = \Phi_P(I) \oplus s_P^{I_\perp}(I) = \Phi_P(I) \oplus I_\perp = \Phi_P(I)$.

Es zeigt sich also eine auffallende Ähnlichkeit zur Kripke-Kleene-Semantik. Es handelt sich jedoch nicht um die Kripke-Kleene-Semantik. Diese ist unter

der Geschlossenen-Welt-Annahme definiert: In Definition 14 ist die Hypothese $H = I_f$ festgelegt. Folglich werden bei der Bildung von P^* Regeln der Form $A \leftarrow$ `false` eingefügt, wenn A nicht als Kopf in P vorkommt. Im vorliegenden Fall dagegen werden Regeln der Form $A \leftarrow \bot$ eingefügt. Es wird also deutlich, dass die Geschlossene-Welt-Annahme hier nur für die bei der Bildung von P^* eingefügten Standardregeln eine Rolle spielt. Um die Kripke-Kleene-Semantik in unserem Rahmen zu beschreiben, bietet sich also folgende Grundannahme an:

Definition 29. *Gegeben sei ein Logikprogramm P. Wir definieren die Hypothese H_{KK} wie folgt: $H_{KK}(A) =$* `false` *für alle Atome A, die Kopf einer Regel in P sind, $H_{KK}(A) = \bot$ sonst.*

Dabei ist die Abbildung nach `false` für die Atome, die nicht als Kopf vorkommen, nötig, um die geschlossene Weltannahme zu modellieren. Für alle anderen Atome muss die Hypothese jedoch nach $\bot$ abbilden, um wie oben beschrieben den Einfluss des Supports auf den Operator $\tilde{\Pi}_P^H$ „auszuschalten".

In [LS05] wird anhand der Eigenschaften des Operators $\tilde{\Pi}_P^H$ bewiesen, dass unter der Hypothese $H = H_{KK}$ die H-fundierten Modelle genau die Kripke-Kleene-Modelle sind. Wir zeigen dies, indem wir im nächsten Abschnitt beweisen, dass unter dieser Hypothese und einer klassischen logischen Umgebung (AW) zurückführbar ist auf (F) aus Abschnitt 3.4 (s. a. die Erläuterungen in [HW05]).

5.2 (F) als Sonderfall von (AW)

Die Bedingung (F) ist bezüglich eines einwertigen, (AW) aber bezüglich eines zweiwertigen level-mappings definiert. In einer klassisch logischen Umgebung kann der einem Atom zugewiesene Wahrheitswert sich jedoch nur einmal ändern: von $\bot$ auf `true` oder `false` . Der Wert $\top$ ist aufgrund der Forderung nach konsistenten Interpretationen in Definition 15 nicht möglich. Wir benutzen also auch hier die Projektion $\cdot^1$ nach Definition 28.

Theorem 7. *Gegeben sei eine klassische logische Umgebung nach Definition 15. Erfüllt eine Interpretation I die Bedingung (AW) bezüglich eines zweiwertigen level-mappings l, der Hypothese $H = H_{KK}$ und eines Programms P, dann erfüllt P auch (F) bezüglich I und l^1.*

Beweis. Aus dem letzten Abschnitt ist bekannt: erfüllt eine Interpretation I unter $H = H_{KK}$ die Bedingung (AW), so erfüllt sie (AWi) für jede Ordinalzahl. Das gilt insbesondere auch für $\alpha = l(A, I(A))$. Das heißt $I(A) = I^{\alpha+1}(A) = I^{\alpha}(\varphi)$. Da die Rümpfe in P nur aus Disjunktionen von Literalen bestehen (s. Definition15), hat φ die Form $\varphi_1 \vee \cdots \vee \varphi_n$, wobei jedes

$\varphi_i,\ \ 1 \leq i \leq n$ nur aus Disjunktionen von Literalen besteht und dem Rumpf einer Regel in P entspricht.

Nehmen wir nun an $I(A) = \texttt{true}$, also $I^\alpha(\varphi) = \texttt{true}$. Folglich existiert ein $\varphi_i,\ \ 1 \leq i \leq n$ mit $I^\alpha(\varphi_i) = \texttt{true}$. Es gilt dann $I^\alpha(L) = \texttt{true}$ für alle $L \in \varphi_i$. Nach Definition 26 von I^α folgt daraus $l(L, \texttt{true}) < \alpha$ und damit $l^1(L) < \alpha$. Aus der Montonieeigenschaft von l (Definition 25. 3) und der Tatsache, dass unter den gegebenen Bedingungen `true` keinen Nachfolger auf der Wissensordnung hat, ergibt sich aus $I^\alpha(L) = \texttt{true}$ auch $I(L) = \texttt{true}$ oder in Mengenschreibweise $L \in I$. Also existiert eine Regel in P mit Rumpf φ_i. Für diese gilt folglich $L \in I$ und $l^1(A) = \alpha > l^1(L)$ für alle L in φ_i. Damit ist (Fi) erfüllt.

Nehmen wir an $I(A) = \texttt{false}$, also $I^\alpha(\varphi) = \texttt{false}$. Dann gilt auch $I^\alpha(\varphi_i) = \texttt{false}$ für alle φ_i, wobei $i \leq i \leq n$. Also existiert für jedes φ_i ein $L \in \varphi_i$ mit $I^\alpha(L) = \texttt{false}$. Mit Definition 26 folgt daraus $l(L, \texttt{false}) < \alpha$ und folglich $l^1(L) < \alpha$. Dieselbe Argumentation wie für den vorherigen Fall ergibt, dass aus $I^\alpha(L) = \texttt{false}$ hier auch $I(L) = \texttt{false}$ folgt. Da die Formeln $\varphi_i,\ \ 1 \leq i \leq n$ genau den Rümpfen der Regeln in P entsprechen, gilt also für jede Regel in P dass ein Literal L im Rumpf existiert mit $I(L) = \texttt{false}$ oder $\neg L \in I$ und $l^1(A) = \alpha > l^1(L)$. Damit ist (Fii) erfüllt. □

5.3 Die KKS-Transformation

Wir haben in Abschnitt 5.1 gesehen, dass unter der Hypothese H_{KK} die H-fundierten Modelle den Kripke-Kleene Modellen entsprechen. Interessant dabei ist die Tatsache, dass die iterative Berechnung des Modells dabei genau der Kripke-Kleene-Semantik entspricht, da $\tilde{\Pi}_P^H$ durch das „Ausschalten“ des Supports dem Operator Φ_P entspricht. Dies ist nun nicht nur für H_{KK} der Fall, sondern für alle Hypothesen, die den Atomen, die Kopf einer Regel in P sind, den Wert $\bot$ zuweisen und damit der Support diesen ebenfalls $\bot$ zuweisen muss. Die Modelle unterscheiden sich nur noch aufgrund der in P^* eingefügten Regeln für Atome, die nicht als Kopf in P auftauchen. Dies eröffnet die Möglichkeit, mit der Kripke-Kleene Semantik auch „offenere“ Semantiken als diese zu berechnen, d.h. Modelle unter Hypothesen, die auch einigen Atomen den Wert $\bot$ zuweisen, die von H_{KK} auf den Wert `false` abgebildet werden. Da sich dieser Unterschied nur in den zusätzlich eingefügten Regeln manifestiert, können diese Regeln zuvor mit einem anderen Rumpf als `false` hinzugefügt werden. Damit tritt das betreffende Atom bereits als Kopf einer Regel auf und wird bei der Bildung von P^* deshalb nicht mehr mit einer zusätzlichen Standardregel belegt.

5.3.1 Die Hypothesenfamilie KKS

Definieren wir nun diese „offeneren“ Semantiken formal:

Definition 30. *Gegeben sei ein Programm P. In der Hypothesenmenge* KKS *befinden sich alle Hypothesen H für die gilt: $H(A) = \bot$, wenn A Kopf einer Regel in P ist, ansonsten ist $H(A) = \bot$ oder $H(A) = \texttt{false}$.*

Im Biverband der Interpretationen auf $\mathcal{FOUR}$ sind die Hypothesen in KKS genau diejenigen, die bezüglich der Wissensordnung kleiner als H_{KK} sind. Insbesondere sind auch H_{KK} sowie $I_\bot$ in KKS.

In [LS05] wird gezeigt, dass unter der Hypothese H_{KK} der Support genau dieser entspricht, also $s_P^{H_{KK}}(I) = H_{KK}(I)$ für jede Interpretation I. Wir zeigen, dass das auch für die anderen Hypothesen in KKS gilt.

Lemma 2. *Gegeben sei ein Programm P, eine Hypothese $H_{KKS} \in$ KKS und eine Interpretation I. Es gilt: $s_P^{H_{KKS}}(I) = H_{KKS}$.*

Beweis. Betrachten wir ein Atom A, das Kopf einer Regel in P ist. Dann erfordert Punkt 1 der Definition 21 sicherer Interpretationen, dass

$$s_P^{H_{KKS}}(A) = \bot = H_{KKS}(A).$$

Kommt A nicht als Kopf einer Regel in P vor, gibt es zwei Möglichkeiten: Wenn $H_{KKS}(A) = \bot$, dann folgt aus Definition 21.1, dass $s_P^{H_{KKS}}(A) = \bot$. Ist $H_{KKS}(A) = \texttt{false}$, dann befindet sich die Regel $A \leftarrow \texttt{false}$ in P^*. Entsprechend wird Definition 21.2 zu folgendem Ausdruck:

$$s_P^{H_{KKS}}(A) \leq_k \Phi_P(I \oplus s_P^{H_{KKS}})(A) = I(\texttt{false}) \oplus s_P^{H_{KKS}}(\texttt{false}) = \texttt{false}.$$

Da der Support die k-größte sichere Interpretation ist, gilt: $s_P^{H^{KKS}}(A) = \texttt{false}$ und folglich $s_P^{H^{KKS}}(A) = H^{KKS}(A)$. □

5.3.2 Die KKS-Programmtransformation

Wir definieren nun eine Programmtransformation, die es ermöglicht, die H-fundierten Modelle unter Hypothesen aus KKS mittels der Kripke-Kleene Semantik zu berechnen.

Definition 31. *Gegeben sei ein Logikprogramm P und eine Hypothese H. Wir definieren die* Programmtransformation $\mathcal{C}^H$ *wie folgt: Eine Regel $A \leftarrow \varphi$ in $\mathcal{C}^H(P)$ hat die Form $A \leftarrow \varphi_1 \vee \cdots \vee \varphi_n$, wenn $A \leftarrow \varphi_1, \ldots, A \leftarrow \varphi_n$ Regeln in P sind, und hat die Form $A \leftarrow H(A)$, wenn A nicht als Kopf einer Regel in P vorkommt.*

Die Transformation führt die selbe Berechnung durch, die bei der Bildung von P^* vonstatten geht, jedoch mit einer frei wählbaren Hypothese H.

Theorem 8. *Das H-fundierte Modell unter $H = H_{KKS}$ eines Logikprogrammes P ist identisch zum H-fundierten Modell von $\mathcal{C}^{H_{KKS}}(P)$ unter $H = H_{KK}$, wobei $H_{KKS} \in$ KKS.*

Beweis. Sei $\mathcal{C}^{H_{KKS}}(P) = P'$. Wir zeigen nun, dass die Operatoren $\tilde{\Pi}_{P'}^{H_{KK}}$ und $\tilde{\Pi}_{P}^{H_{KKS}}$ für jeden Iterationsschritt dasselbe Resultat haben.

$\tilde{\Pi}_{P'}^{H_{KK}}$ arbeitet auf P'^*, was identisch zu P' ist, da in P' jedes Atom Kopf genau einer Regel ist und folglich bereits alle Bedingungen für P'^* erfüllt sind. Die Bildung von P^* unter der Hypothese H_{KKS} ergibt ebenfalls dasselbe Programm P': Für Atome, die Köpfe mehrerer Regeln sind, wird dieselbe Konjunktion der Rümpfe eingefügt, wie in der Definition von P^* beschrieben. Da die Transformation bezüglich H_{KKS} durchgeführt wird und somit für Atome, die nicht als Kopf vorkommen, der Wert ebendieser Atome bezüglich H_{KKS} als Rumpf der eingefügten Regeln verwendet wird, sind auch diese identisch zu den in P^* eingefügten. Die beiden Operatoren arbeiten also auf demselben Programm.

Betrachten wir nun den Iterationsschritt α, wobei α beliebig gewählt wurde:

$$(\tilde{\Pi}_{P'}^{H^{KK}} \uparrow \alpha + 1)(A) = (\tilde{\Pi}_{P'}^{H^{KK}} \uparrow \alpha)(\varphi) \oplus s_{P'}^{H^{KK}}(\tilde{\Pi}_{P'}^{H^{KK}} \uparrow \alpha)(A).$$

Es gilt $H^{KK}(A) = \bot$ für alle Atome A in P', da alle Atome in P' Kopf einer Regel sind. Aus Lemma 2 folgt, dass der Support identisch zu H^{KK} ist und wir erhalten

$$(\tilde{\Pi}_{P'}^{H^{KK}} \uparrow \alpha + 1)(A) = (\tilde{\Pi}_{P'}^{H^{KK}} \uparrow \alpha)(\varphi). \tag{5.1}$$

Betrachten wir nun den Operator

$$(\tilde{\Pi}_{P}^{H^{KKS}} \uparrow \alpha + 1)(A) = (\tilde{\Pi}_{P}^{H^{KKS}} \uparrow \alpha)(\varphi) \oplus s_{P}^{H^{KKS}}(\tilde{\Pi}_{P}^{H^{KKS}} \uparrow \alpha)(A).$$

Wiederum folgt aus Lemma 2, dass $H^{KKS}(A) = s_{P}^{H^{KKS}}(\tilde{\Pi}_{P}^{H^{KKS}} \uparrow \alpha)(A)$. Für die Atome A, die Kopf einer Regel in P sind, gilt $H^{KKS}(A) = \bot$. Entsprechend ist

$$(\tilde{\Pi}_{P}^{H^{KKS}} \uparrow \alpha + 1)(A) = (\tilde{\Pi}_{P}^{H^{KKS}} \uparrow \alpha)(\varphi). \tag{5.2}$$

Da die Regeln in P^* und P'^* dieselben sind, ist das Resultat der beiden Operatoren in (5.1) und (5.2) identisch.

Für die Atome A, die nicht als Kopf einer Regel in P vorkommen, gilt, dass entweder $H^{KKS}(A) = \bot$ oder $H^{KKS}(A) = \texttt{false}$. Das folgende Argument gilt analog für beide Fälle, wir betrachten den letzteren: Dann muss gelten $H^{KKS}(A) = \texttt{false}$ und es gibt eine Regel $A \leftarrow \texttt{false}$ in P^* und damit auch in P'. Also ist

$$(\tilde{\Pi}_{P}^{H^{KKS}} \uparrow \alpha)(\varphi) = s_{P}^{H^{KKS}}(\tilde{\Pi}_{P}^{H^{KKS}} \uparrow \alpha)(A) = \texttt{false}$$

genauso wie

$$(\tilde{\Pi}_{P'}^{H^{KK}} \uparrow \alpha)(\varphi) = \texttt{false}.$$

Folglich ergeben beide Operatoren das gleiche Resultat. □

5.3.3 Beispiel

Im folgenden Beispiel soll anhand eines kleinen Programms die Relevanz der Hypothesen illustriert werden. Dabei wird sich zeigen, dass in diesem Fall keine der bekannten Grundannahmen (Offene- und Geschlossene-Welt-Annahme) geeignet ist, sondern eine Hypothese aus KKS.

Wir formulieren in Anlehnung an [LS05] S.370 den vereinfachten Entscheidungsprozess eines Richters als Logikprogramm, basierend auf $\mathcal{FOUR}$:

$$\begin{aligned} is\_suspect(X) &\leftarrow has\_motive(X) \vee has\_witness(X) \\ is\_cleared(X) &\leftarrow \neg contradict\_alibi(X) \wedge has\_alibi(X) \\ charge(X) &\leftarrow is\_suspect(X) \oplus \neg is\_cleared(X) \end{aligned}$$

Der Richter benutzt zum einen Informationen darüber, ob der Angeklagte verdächtig ist ($is\_suspect$), zum anderen über seine Unschuld ($is\_cleared$). Für einen Verdacht spricht die Existenz eines Motivs (has_motive) oder eines Zeugen, der die Tat beobachtet hat ($has\_witness$). Die Unschuld wird durch ein Alibi bezeugt (has_alibi), das aber nur anerkannt wird, wenn es nicht den Aussagen des Angeklagten widerspricht($contradict\_alibi$). Dann kombiniert der Richter die so gesammelten Informationen, um zu entscheiden ob der Angeklagte verurteilt wird ($charge$).

Nehmen wir nun weiter an, der Richter hat lediglich folgende gesicherte Informationen über einen Angeklagten namens „Ted“:

$$has\_witness(Ted) \leftarrow \texttt{false}$$

Aufgrund dieser wenigen Informationen dürfte Ted nicht schuldig gesprochen werden.

Legen wir nun die Geschlossene-Welt-Annahme zugrunde, so wird sowohl $has\_motive(Ted)$ als auch $has\_witness(Ted)$ der Wert falsch zugewiesen und damit auch $is\_suspect(Ted)$. Da jedoch auch has_alibi durch die Standardannahme falsch wird, ist $is\_cleared(Ted)$ ebenfalls falsch. Für die Evaluation von $charge(Ted)$ sind also widersprüchliche Informationen vorhanden, das Programm evaluiert $charge(Ted)$ zu $\top$. Der Richter kann also keine Aussage machen.

Wird nun stattdessen die Offene-Welt-Annahme verwendet, also alle Atome haben den Standardwert $\bot$, so bekommt $is\_suspect(Ted)$ den Wert $\bot$, da über $has\_motive(Ted)$ keine weiteren Informationen vorliegen und die Standardannahme greift. Weil diese auch auf $contradict\_alibi(Ted)$ und $has\_alibi(Ted)$ angewendet wird, bekommt $is\_cleared(Ted)$ ebenfalls den Wert $\bot$. Somit wird auch $charge(Ted)$ der Wert $\bot$ zugewiesen, es kann ebenfalls keine Aussage getroffen werden.

Betrachten wir nun folgende Hypothese H_m aus KKS:

$$\begin{aligned} H_m(has\_witness) &= \texttt{false} \\ H_m(has\_motive) &= \texttt{false} \\ H_m(has\_alibi) &= \bot \\ H_m(contradict\_alibi) &= \bot \end{aligned}$$

Damit wird wie unter der geschlossenen Weltannahme $is\_suspect(Ted)$ falsch. Die unter der Geschlossenen-Welt-Annahme widersprüchliche Information aus $is\_cleared(Ted)$ wird jetzt jedoch nicht mehr abgeleitet. Dieses Atom wird wie unter der Offenen-Welt-Annahme mit $\bot$ belegt, da hier die Informationen nicht ausreichen, um eine Aussage zu treffen. Also wird folgerichtig $charge(Ted)$ falsch.

Es wird also klar, dass die erste Programmzeile nach der in der Logikprogrammierung üblichen Geschlossenen-Welt-Annahme konzipiert ist. Der zweiten Zeile dagegen liegt ein anderer Gedanke zugrunde: Damit $is\_cleared$ falsch wird, genügt bereits der Wert `false` für has_alibi. Deshalb hat dieser Wert in etwa die Bedeutung, dass nachgewiesen werden konnte, dass niemand dem Angeklagten ein Alibi geben kann. Es muss dann aber auch modelliert werden können, dass darüber nichts bekannt ist, was dem Wert $\bot$ entspricht. Da dies der Standardwert sein sollte, basiert dieser Ansatz also auf der Offenen-Welt-Annahme.

H-fundierte Modelle ermöglichen, trotz dieser verschiedenen Ansätze in einem Programm mittels der entsprechenden Hypothese die richtigen Ergebnisse zu erzielen. Für Programme wie dem vorliegenden kann nun durch die KKS-Transformation die Lösung sogar mit einem System, das nur die Kripke-Kleene Semantik berechnet, erreicht werden.

Kapitel 6

Implementierung für Prolog und Anbindung an DL-Reasoner

In diesem Kapitel stellen wir eine Sammlung von Tools und Bibliotheken vor. Diese ermöglichen, Programme in einer Syntax ähnlich zu Prolog zu schreiben, die aber auf der vierwertigen Logik $\mathcal{FOUR}$ basiert. Die Programme können mittels der KKS-Transformation unter verschiedenen Hypothesen aus KKS verarbeitet werden. Ein Compiler übersetzt diese Programme dann in gängiges Prolog, wobei auf eine von uns realisierte Implementierung von $\mathcal{FOUR}$ für Prolog zurückgegriffen wird. Des Weiteren präsentieren wir eine Bibliothek, welche die mit $\mathcal{FOUR}$ gewonnene größere semantische Expressivität nutzt, um Zugriffe auf einen DL-Reasoner aus Prolog zu ermöglichen – also auf Reasoning basierend auf der Offenen-Welt-Annahme zurückzugreifen.

6.1 Implementierung der KKS-Transformation

Die KKS-Transformation haben wir in JAVA realisiert. Eingabeformat ist dabei eine Syntax ähnlich zu Prolog. Es wird jedoch „-“ (Bindestrich) anstelle von „:-“ (Doppelpunkt Bindestrich) als Operator verwendet, um den Unterschied zu visualisieren. Die Negation im Biverband wird mit „$\sim$“ (Tilde) notiert. Für die Operatoren $\oplus, \otimes, \wedge, \vee$ kommen die Bezeichnungen sup, inf, and, or zum Einsatz. Für and kann analog zu Prolog auch ein Komma geschrieben werden. Die Werte $\top, \bot,$ `true`, `false` werden mit top, bottom, true, false notiert. Fakten muss mit einer Regel der Form Atom - true. explizit der Wahrheitswert zugewiesen werden. Die aus Prolog bekannte Form Atom. funktioniert nicht. Abbildung 6.1 stellt die komplette Grammatik der Eingabesprache dar. Wir schreiben dabei den Infix-Operator „|“ für Alternativen, und die Postfix-Operatoren „$*$“ für keine oder beliebig viele Vorkommnisse, „+“ für ein oder beliebig viele Vorkommnisse und „?“ für kein oder genau ein Vorkommnis. Terminale stehen in Anführungszeichen.

```
RULE ::= ATOM "-" [FORMULA | TV]
FORMULA ::= LITERAL [JUNCTOR LITERAL]*
LITERAL ::= [ATOM | "~"ATOM]
ATOM ::= NAME "(" [TV | TERM ][","  [TV | TERM ]]* ")"
JUNCTOR ::= [","|"and"|"or"|"inf"|"sup"]
TV ::=["true"|"false"|"top"|"bottom"]
TERM ::= NAME ["(" NAME ")"]?
NAME ::= ["A"-"Z","a"-"z","_","0"-"9"]+
```

Abbildung 6.1: Grammatik der Eingabesprache für Programme

Die Standardhypothese wird in Form einer zweiten Eingabedatei bestehend aus Fakten der Form a(true). spezifiziert. Die formale Definition findet sich in Abbildung 6.2.

```
HYPOTHESE ::= NAME "(" TV ")" "."
TV ::=["true"|"false"|"top"|"bottom"]
NAME ::= ["A"-"Z","a"-"z","_","0"-"9"]+
```

Abbildung 6.2: Grammatik der Eingabesprache für Hypothesen

Im Gegensatz zur theoretischen Erläuterung im letzten Kapitel arbeitet das Programm nicht mit Grundinstanzen. Es muss also nur einmal pro Atom bzw. Prädikat eine Hypothese angegeben werden, nicht für jede daraus gebildete Grundinstanz; dies wäre auch nicht praktikabel da die Grundinstanzenmenge potentiell unendlich ist. Als weiterer Unterschied zur obigen formalen Beschreibung der KKS-Transformation werden die Regeln mit gleichem Kopf nicht zu einer Regel zusammengefasst. In der formalen Analyse und Beschreibung haben wir diese Formulierung gewählt, um mit der Verwendung von P^* nach Definition 10 im Theorieteil kompatibel zu sein. Da es sich bei dieser Zusammenfassung jedoch nur um eine vereinfachende Notation handelt (siehe dazu etwa [Fit02]), Programme mit mehreren Regeln aber viel übersichtlicher und auch gängige Praxis in der Programmierung mit Prolog sind, wurde für die Implementierung darauf verzichtet.

Ausgegeben wird also ein Programm, das um die entsprechenden Standardregeln erweitert ist und ebenfalls der Syntax in Abbildung 6.1 folgt.

6.2 $\mathcal{FOUR}$ in Prolog

Um in Prolog mit $\mathcal{FOUR}$ arbeiten zu können, haben wir uns für folgenden Ansatz entschieden: Jedes Prädikat ist mindestens einstellig. Im letzten bzw. einzigen Argument findet sich der Wahrheitswert, dargestellt durch

true, false, top, bottom. Ein Atom $a(T_1, \ldots, T_n)$ wird also im Programm zu a(T_1, ..., T_n,TV). Es ist in Prolog genau dann ableitbar, wenn es den Wahrheitswert TV hat. Unsere Bibliothek verarbeitet derartige Atome und stellt dabei Prädikate für die bekannten Operatoren auf dem Biverband $(\wedge, \vee, \oplus, \otimes, \neg)$ zur Verfügung. Quantoren sind nicht verfügbar, stattdessen gilt die für Prolog übliche Annahme, dass sämtliche Variablen allquantifiziert sind. Weiterführende Erläuterungen hierzu finden sich etwa in [Apt97] Kapitel 3.

Noch ein Hinweis: Für Prolog hat das Wort „Atom“ die Bedeutung „nicht numerische Konstante“. Wenn im folgenden von „Atomen“ die Rede sein wird, sind jedoch immer Atome nach Definition 6 gemeint. Der Begriff wird also wie im bisherigen Text weiterverwendet.

6.3 Compiler für KKS-Transformierte Programme

Nach Theorem 8 lassen sich die Modelle der KKS-Transformierten Programme mit der Kripke-Kleene Semantik berechnen. Wir wollen nun Prolog, das diese Semantik approximiert (siehe dazu etwa [Apt97] Kapitel 4) benutzen, um Programme, wie sie in 6.1 beschrieben wurden, zu bearbeiten. Dazu stellen wir einen Compiler zu Verfügung, der Programme in der Syntax nach Abbildung 6.1 in Prolog übersetzt und dabei auf die in Abschnitt 6.2 beschriebene Bibliothek zurückgreift, um die Logik $\mathcal{FOUR}$ nutzen zu können.

Der Benutzer muss nicht mit den Prädikaten für $\mathcal{FOUR}$ vertraut sein, um dieses System zu verwenden. Er schreibt Programme in der Prolog-ähnlichen Syntax aus Abbildung 6.1. Nach einer eventuellen KKS-Transformation zur Verwendung der gewünschten Hypothese aus KKS, gibt der Compiler dann ein Programm aus, das in jedem gängigen Prolog-System laufen sollte. Entwickelt wurde unser System für SWI-Prolog[1]. Zur Formulierung von Anfragen muss sich der Benutzer lediglich mit dem Konzept vertraut machen, dass die vierwertigen Wahrheitswerte grundsätzlich den letzten Parameter eines Prädikates bilden. Nach diesen kann wie nach jeder anderen Variablen in Prolog gefragt werden.

6.4 Beispiel

Wir wollen den Ablauf anhand eines sehr kleinen Beispiels demonstrieren. Dazu dient folgendes einzeilige Programm:

```
a(X) - b(Y) sup c.
```

[1] `http://www.swi-prolog.org`

Als Hypothese spezifizieren wir:

```
a(false).
b(false).
c(bottom).
```

Daraus wird nach der Transformation:

```
a(X) - b(Y) sup c(Z).
b(_TV0) - false.
c - bottom.
```

Für a wird, obwohl spezifiziert, kein Wert eingefügt, da dieses Prädikat als Kopf vorkommt. Nach dem Compilieren erhalten wir:

```
a(X,TVHead) :- fw_sup(TV1,TV2,TVHead),
               b(Y,TV1),
               c(TV2).
b(_TV0,false).
c(bottom).
```

Die Prädikate wurden also wie beschrieben jeweils um ein Argument ergänzt, das den Wahrheitswert aufnimmt. Die „expliziten“ Fakten b(_TV0) – false. und c – bottom. werden damit zu Fakten für Prolog. Auch das bisher nullstellig Prädikat c wird einstellig. Für die erste Programmzeile lässt sich beobachten, wie der Operator sup durch das Prädikat fw_sup modelliert wird, das aus der Bibliothek für den Biverband $\mathcal{FOUR}$ stammt. Dabei ist fw_sup(X,Y,Z) genau dann ableitbar, wenn $X \oplus Y = Z$. Die Verknüpfung mit den Prädikaten b und c erfolgt durch entsprechende Benennung der automatisch eingefügten Variablen für die Wahrheitswerte.

Das Programm kann nun in Prolog geladen werden. Wird nichts weiter spezifiziert, so liefert die Anfrage

```
?- a(t,X).
```

das Ergebnis

```
Y = false.
```

Es wird also auf die Standardwerte zurückgegriffen und es wird `false` $\oplus \bot =$ `false` ausgegeben. Setzt der Benutzer nun andere Werte oder ist das Prädikat Teil eines größeren Programms, in dem sich die Wahrheitswerte von b und c ändern, liefert die Anfrage entsprechend andere Ergebnisse.

6.5 Einbindung eines DL-Reasoners

In diesem Abschnitt soll die von uns realisierte Einbindung eines DL-Reasoners in das oben beschriebene Prolog-System vorgestellt werden. Bevor wir im Abschnitt 6.5.2 unseren Ansatz erläutern, soll im nächsten Teil zuerst eine kurze Einführung in Beschreibungslogiken gegeben werden.

6.5.1 Beschreibungslogiken und DL-Reasoner

Beschreibungslogiken (engl.: description logics, Abgekürzt DL) sind allgemein gesprochen ein entscheidbares Fragment der Prädikatenlogik erster Stufe. Um die Komplexität der Berechnungen zu reduzieren, ist es jedoch üblich, gegenüber der Prädikatenlogik zusätzliche Einschränkungen zu machen. Beschreibungslogiken stellen eines der zentralen formalen Instrumentarien für das Semantic Web [BLFD99, BLHL01] und ontologiebasierte Systeme dar. So basiert die Web Ontology Language (OWL) [MvH] auf Beschreibungslogiken. DL-Reasoner sind Systeme, die Wissensbasen in einer Beschreibungslogik verwalten und mit diesen Informationen ebenfalls in Beschreibungslogik gestellte Anfragen bearbeiten können. Da sie mit Teilmengen der Prädikatenlogik erster Stufe arbeiten, liegt diesen Systemen inhärent die Offene-Welt-Annahme zugrunde. Bekannte DL-Reasoner sind beispielsweise RacerPro[2], Pellet[3], FaCT++[4] und KAON2[5]. Letzterer kommt bei unserem System zum Einsatz. Da der Zugriff jedoch über das systemunabhängige DIG-Interface [Bec03] realisiert wurde, ist prinzipiell auch die Zusammenarbeit mit anderen Reasonern möglich. In der Praxis muss jedoch beachtet werden, dass die Spezifikation des DIG-Interfaces Spielraum für Interpretationen lässt, der nicht von allen Systemen gleich ausgefüllt wird.

Unter Beschreibungslogiken wird eine ganze Familie von Logiken verstanden, die sich in ihrer Expressivität und damit auch in der Zeitkomplexität der auf ihr basierten Anfragen unterscheiden. Das von uns benutzte System KAON2 benutzt beispielsweise (neben anderen) OWL-DL, eine syntaktische Variante der Logik $\mathcal{SHOIN}(\mathcal{D})$ [HST99, HPS03]. Da wir aber nur einige bestimmte Anfragen an DL-Systeme ermöglichen und das verwendete DIG-Interface zusätzlich Einschränkungen gegenüber der vollen Expressivität der dem Reasoner zugrunde liegenden Logik aufweist, wollen wir in der folgenden formalen Einführung nur die für diese Arbeit relevanten Teile erläutern. Diese entsprechen in etwa der Logik $\mathcal{ALC}$ aus [SSS91]. Ausführliche Informationen finden sich etwa in [BBH+90] oder [BCM+03].

Definition 32. *Eine Beschreibungslogik besteht aus einer Menge $\mathcal{C}$ an* Konzepten *und $\mathcal{R}$ an* Rollen. *Die Menge $\mathcal{C}$ besteht aus den sogenannten* atomaren Konzepten *und solchen, die nach der folgenden induktiven Definition gebildet werden: Seien C, D Konzepte und R eine Rolle. Dann sind auch $(C \sqcap D)$, $(C \sqcup D)$, $(\neg C)$, $(\forall R.C)$ und $(\exists R.C)$ Konzepte. Ist R eine Rolle, so ist auch R^- eine Rolle.*

[2] `http://www.racer-systems.com`
[3] `http://www.mindswap.org/2003/pellet/index.shtml`
[4] `http://owl.man.ac.uk/factplusplus`
[5] `http://kaon2.semanticweb.org/`

Definition 33. *Eine* Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *einer Beschreibungslogik besteht aus einer Menge* $\Delta^{\mathcal{I}}$*, der sogenannten* Domäne*, und der Abbildung* $\cdot^{\mathcal{I}}$*. Diese bildet atomare Konzepte auf Teilmengen von* $\Delta^{\mathcal{I}}$ *und Rollen auf Teilmengen von* $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ *ab. Des Weiteren gilt:*

- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
- $(\forall R.C)^{\mathcal{I}} = \{x | \text{für alle } y \text{ gilt: } \langle x, y\rangle \in R^{\mathcal{I}} \text{ impliziert } y \in C^{\mathcal{I}}\}$
- $(\exists R.C)^{\mathcal{I}} = \{x | \text{es existiert ein } y \text{ mit } \langle x, y\rangle \in R^{\mathcal{I}} \text{ und } y \in C^{\mathcal{I}}\}$

sowie

- $(R^{-})^{\mathcal{I}} = \{\langle x, y\rangle | \langle y, x\rangle \in R^{\mathcal{I}}\}$

Definition 34. *Ein Konzept* C *heißt* erfüllbar*, wenn gilt* $C^{\mathcal{I}} \neq \emptyset$*. Elemente von* $\Delta^{\mathcal{I}}$ *(auch* Individuen *genannt), die ein Konzept* C *erfüllen, heißen* Instanzen *von* C*. Ein Konzept* C subsumiert *ein Konzept* D*, wenn gilt* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$*. Wir schreiben dafür* $C \sqsubseteq D$*. Zwei Konzepte* C, D *heißen* äquivalent*, wenn gilt* $C \sqsubseteq D$ *und* $D \sqsubseteq C$*. Sie heißen* disjunkt*, wenn gilt* $C \sqsubseteq \neg D$*.*

6.5.2 Anfragen an DL-Reasoner aus Prolog

Um Anfragen an DL-Reasoner aus dem in Abschnitt 6.2 beschriebenen Prolog-System zu ermöglichen, verwenden wir einen ähnlichen Ansatz wie ihn Eiter et. al. in [ELST04a] beschreiben. Unser System verwendet jedoch Anfragen, die durch die Semantik basierend auf $\mathcal{FOUR}$ motiviert sind.

Wir definieren besonders ausgezeichnete Atome, die einer Anfrage an einen DL-Reasoner entsprechen. Diese werden also nicht wie gewöhnlich anhand der Semantik des Logikprogramms ausgewertet, sondern sie werden basierend auf Anfragen an einen Reasoner mit Werten belegt.

Definition 35. *Ein* DL-Atom *hat die Form* $dl_q(\mathbf{p}_q)$*, wobei* q *eine DL-Anfrage ist und* $\mathbf{p}_q$ *der zu Anfrage* q *gehörige Parametervektor.*

Als Anfragen bezüglich Konzepten stehen $subsumes, unsatisfiable$ und $disjoint$ zu Verfügung. Mit $instance$ wird angefragt, ob ein Individuum Instanz eines Konzeptes ist. Schließlich kann mit has_role nach der Verbindung zweier Individuen durch eine Rolle gefragt werden.

Gewöhnlich haben Anfragen an DL-Reasoner zwei möglich Antworten: Der angefrage Sachverhalt ist beweisbar oder nicht. Unserem System liegt aber eine expressivere Logik zugrunde, die insbesondere den Wert $\perp$ verarbeiten kann. Deshalb sollten die Anfragen auch entsprechend umfassendere

Informationen liefern. Dies wird ermöglicht, indem zur Auswertung jedes DL-Atoms zwei Anfragen an den DL-Reasoner gestellt werden, eine die dem DL-Atom entspricht und eine, die dessen „Gegenteil“ darstellt.

Wir wollen dies zuerst anhand des Atoms $dl_{subsumes}(C, D)$ erläutern. Die erste Anfrage, die gestellt wird, ist $C \sqsubseteq D$. Bei einer positiven Antwort wissen wir, dass dies beweisbar ist; das DL-Atom erhält folglich den Wert `true`. Bei einer negativen Antwort sind nun zwei Fälle möglich. Im ersten kann $C \sqsubseteq D$ gelten, dies ist aber nicht notwendig. Es kann auch sein, dass nicht alle Elemente von C in D liegen. In diesem Fall wissen wir also nichts oder zu wenig über das Verhältnis von C und D. Entsprechend wird deshalb das DL-Atom mit $\bot$ belegt. Es bleibt der Fall, dass es aufgrund der Informationen in der Wissensbasis unmöglich ist, dass $C \sqsubseteq D$ gilt. Dann bekommt das DL-Atom den Wert `false`. Sei KB die verwendete Wissensbasis. Dann lässt sich die Evaluierung von $dl_{subsumes}(C, D)$ wie folgt zusammenfassen:

Ergebnis der Anfrage: $C \sqsubseteq D$	Ergebnis der Anfrage: Ist $KB \cup \{C \sqsubseteq D\}$ erfüllbar?	Wert von $dl_{subsumes}(C, D)$
yes	no	`true`
no	yes	`false`
no	no	$\bot$

Die Anfrage nach Erfüllbarkeit eines Konzeptes lässt sich durch Reduktion auf $dl_{subsumes}$ lösen: Ein Konzept C ist genau dann erfüllbar, wenn $C \sqsubseteq \bot$ nicht gilt (s. etwa [BCM+03]). Versucht man nun, darauf basierend die Erfüllbarkeit von C nach obiger Tabelle zu modellieren, zeigt sich, dass hier die Antwort *yes* auf beide Anfragen dem Wert $\bot$ entspricht. Dies widerspricht der Intuition unserer Semantik, in der der Wert $\bot$ den Normalfall, also „nichts bekannt“ bedeutet. Dies beruht darauf, dass die Anfrage nach Erfüllbarkeit nach der Existenz eines Modells fragt, während die Auswertung nach obiger Tabelle eine All-Aussage anfragt: Ist in allen Modellen C unter D? Wir suchen also eine Existenzaussage mittels All-Aussagen. Dabei ist eine Umkehrung der Antworten nötig, die sich auch in obiger Formulierung zeigt: Ein Konzept C ist genau dann erfüllbar, wenn $C \sqsubseteq \bot$ *nicht* gilt. Um mit der Semantik des Wertes $\bot$ konsistent zu bleiben, führen wir deshalb das Atom $dl_{unsatisfiable}(C)$ ein, das ebenfalls nach einer All-Aussage fragt: Ist in allen Modellen C unerfüllbar? Die Auswertung dieses Atoms lässt sich dann auf den vorigen Fall reduzieren:

Ergebnis der Anfrage: $C \sqsubseteq \bot$	Ergebnis der Anfrage: Ist $KB \cup \{C \sqsubseteq \bot\}$ erfüllbar?	Wert von $dl_{unsatisfiable}(C, D)$
yes	no	`true`
no	yes	`false`
no	no	$\bot$

Die Auswertung nach dieser Tabelle geschieht wiederum nach derselben Intuition wie für $dl_{subsumes}(C, D)$. Ist C unerfüllbar, gilt also $C \sqsubseteq \bot$, ist die Antwort positiv. Bei einer negativen Antwort gibt es wieder zwei Fälle: Es kann sein, dass es zwar nicht notwendig ist, dass C unerfüllbar ist, aber dies aufgrund der restlichen Informationen der Wissensbasis möglich wäre. Dann erhält $dl_{unsatisfiable}(C)$ den Wert $\bot$. Oder aber die Unerfüllbarkeit ist nicht nur nicht beweisbar, sie bildet sogar einen Widerspruch zu den Informationen der Wissensbasis. Dann erhält das DL-Atom den Wert `false`.

Zur Auswertung des Atoms $dl_{disjoint}(C, D)$ benutzen wir folgenden Sachverhalt: Die Konzepte C und D sind genau dann disjunkt, wenn $C \sqsubseteq \neg D$ (s. etwa [BCM+03]). Da es sich hier wieder um eine All-Aussage handelt, lässt sich diese Anfrage problemlos auf den ersten Fall zurückführen.

Betrachten wir nun die DL-Atome, die sich auf die bekannten Individuen beziehen. Das Atom $dl_{instance}(I, C)$ bekommt den Wert `true` zugewiesen, wenn I eine Instanz von C ist und `false`, wenn I eine Instanz von $\neg C$ ist. Der Wert $\bot$ bedeutet schließlich, dass keines von beidem der Fall ist.

Das Atom $dl_{has\_role}(I_1, R, I_2)$ bekommt den Wert `true`, wenn die Individuen I_1 und I_2 durch die Rolle R verbunden sind, also wenn gilt $\langle I_1, I_2 \rangle \in R$. Für den Wert `false` müssen wir uns auf die bekannten Individuen beschränken, da es in OWL nicht möglich ist, nach negierten Rollen zu fragen (vgl. hierzu [ELST04b] S.7). Damit ist die Semantik von `false` aber immer noch nicht klar: Es wird ausgesagt, dass zwei Individuen nicht über die Rolle R verknüpft sind. Hier könnte man nun allen Paaren von Individuen für die gilt $\langle I_1, I_2 \rangle \notin R$ den Wert `false` zuweisen. Wir haben uns für einen anderen Ansatz entschieden: Fragt man, ob zwei Individuen über eine Rolle verknüpft sind, ist davon auszugehen, dass dies der Fall sein könnte, also dass $\langle I_2, X \rangle \in R$ bzw. $\langle X, I_1 \rangle \in R$ für ein Individuum X. Wir weisen nun solchen Paaren den Wert `false` zu, wo dies der Fall ist, aber $I_1 \neq X$ bzw. $X \neq I_2$. Für alle anderen Paare von Individuen weisen wir dem Atom den Wert $\bot$ zu.

Zur Implementierung definieren wir besonders ausgezeichnete Atome in der Eingabesprache für den Compiler. Dieser übersetzt sie dann in Aufrufe einer Bibliothek, die wir mittels des *Extended DIG Description Logic Interface for Prolog* [HV03] realisiert haben. Diese greift über das DIG Interface [Bec03] auf KAON2 zu. Da es sich bei DIG um einen verbreiteten Standard handelt, können auch andere DL-Reasoner mit dem System verwendet werden. Es kann jedoch nötig sein, die Aufrufe in unserer Bibliothek etwas anzupassen, da die Reasoner die in DIG spezifizierten Anfragen teilweise unterschiedlich interpretieren.

Auch wenn die Auswertungen der DL-Atome über jeweils zwei Anfragen an den Reasoner realisiert sind, bietet unsere Bibliothek für Prolog jeweils

ein Prädikat, das genau einem DL-Atom entspricht.

Das DIG-Interface in seiner aktuellen Version 1.1 ist im Vergleich zu OWL eingeschränkt. Deshalb wird die zweite Anfrage zur Evaluierung der DL-Atome $dl_{subsumes}(C, D)$, $dl_{unsatisfiable}(C)$ und $dl_{disjoint}(C, D)$, die nach der Erfüllbarkeit der ganzen Wissensbasis fragt, in der aktuellen Implementierung durch andere Anfragen approximiert. In der Version 2.0 sollte die Anfrage dann direkt möglich sein (s. [Bec]).

6.6 Formale Semantik für DL-Atome

Wir wollen nun die DL-Atome formal in die gegebene Semantik eingliedern.

Definition 36. *Wir erweitern die Menge der Atome nach Definition 6.2 um die DL-Atome. Ansonsten gilt Definition 6 unverändert.*

Man beachte, dass Definition 6 als Kopf von Regeln nur Prädikate der Form $p(x_1, \ldots, x_n)$ zulässt. Das schließt insbesondere DL-Atome als Kopf aus.

Definition 37. *Gegeben sei ein Programm P. Mit $\mathcal{B}_{DL}$ bezeichnen wir die Menge aller Atome, die sich bilden lassen, indem man die Variablen der in P enthaltenen DL-Atome mit Elementen aus dem Herbrand-Universum von P belegt. Die Herbrand-Basis von $\mathcal{B}_P$ wird weiterhin wie in Definition 7 gebildet, unter Ausnahme der DL-Atome. Die* Grundinstanzenmenge *$ground(P)$ entsteht, wenn man jedes Atom das kein DL-Atom ist durch die daraus gebildeten Atome in $\mathcal{B}_P$ ersetzt, jedes DL-Atom durch die daraus gebildeten Atome in $\mathcal{B}_{DL}$.*

Definition 38. *Die Funktion*

$$query_{KB} : \mathcal{B}_{DL} \to \{\texttt{true}, \bot, \texttt{false}\}$$

bildet ein DL-Atom auf das Ergebnis der Anfrage wie in Abschnitt 6.5.2 beschrieben, ab. Dabei wird die Wissensbasis KB zugrundegelegt. Wir setzen für unsere gesamte Arbeit voraus, dass die Wissensbasis statisch ist, dass also dieselbe Anfrage immer dasselbe Ergebnis liefert.

Dann können wir die Definition 8 von Interpretationen wie folgt erweitern:

Definition 39. *Gegeben sei ein Biverband $\langle B, \leq_t, \leq_k \rangle$ ein Logikprogramm P und eine Wissensbasis KB. Eine* Interpretation *des Logikprogramms ist eine Abbildung*

$$I : \mathcal{B}_P \times \mathcal{B}_{DL} \to B$$

für die gilt:

$$I(A) = query_{KB}(A) \;\; wenn \;\; A \in \mathcal{B}_{DL}.$$

Die Fortsetzung für Formeln nach Definition 8 gilt unverändert.

Der Bildbereich von $query_{KB}$ besteht nur aus kleinsten bzw. größten Elementen eines Biverbandes. Da diese nach Definition 3 immer existieren, verträgt sich Definition 39 mit jedem beliebigen Biverband.

Bei der Bildung von P^* müssen die DL-Atome beim Einfügen der Standardregeln ausgenommen werden. Definition 10 wird also wie folgt abgeändert:

Definition 40. *Gegeben sei ein Logikprogramm P mit DL-Atomen und eine Interpretation H. Mit P^* bezeichnen wir das Programm, das folgende Regeln enthält:*

- $A \leftarrow \varphi_1 \vee \varphi_2 \vee \ldots$ *wenn* $A \leftarrow \varphi_1, A \leftarrow \varphi_2, \ldots$ *alle Elemente von* $ground(P)$ *mit Kopf* A *sind.*

- $A \leftarrow H(A)$ *wenn* $A \in \mathcal{B}_P$ *und* A *nicht als Kopf in* $ground(P)$ *vorkommt.*

Aufgrund der Annahme, dass die Wissensbasis unveränderlich ist, besteht prinzipiell kein Unterschied zum bekannten Basisfall in Definition 8, der nur für Grundinstanzen definiert ist. Alle Überlegungen bleiben also auch nach Eingliederung der DL-Atome gültig.

Die Semantik basierend auf H-fundierten Modellen und insbesondere solchen die Hypothesen aus KKS verwenden, ermöglicht einen eleganten Umgang mit Regeln und Informationen, denen die Offene-Welt-Annahme zugrunde liegt. Deshalb können die Ergebnisse von Anfragen an DL-Reasoner ohne größere Modifikationen problemlos integriert werden. Zudem erlaubt die Freiheit der Jede-Welt-Annahme die beliebige Kombination von beiden Prinzipien im Logikprogramm.

Unter der Annahme einer statischen Wissensbasis ergäbe sich eine deutlich einfachere Semantik, wenn man vor der Evaluierung des Logikprogramms alle DL-Atome durch den Wahrheitswert, den die entsprechende Anfrage liefert, ersetzt. Dies ist möglich, da nach Definition 6 die Elemente des verwendeten Biverbandes als Atome zugelassen sind. Dadurch würde das Programm frei von DL-Atomen und könnte wie bisher behandelt werden. Da es in Zukunft jedoch auch möglich sein soll, Informationen aus dem Logikprogramm in die Wissensbasis zu übertragen, haben wir uns für den oben beschriebenen etwas komplexeren Ansatz entschieden, der diese Erweiterung problemlos zulässt. Schon im jetzigen Rahmenwerk kann die Forderung nach einer statischen Datenbank fallengelassen werden, wenn garantiert ist, dass die Ergebnisse der Anfragen von $query_{KB}$ monoton bezüglich der Wissensordnung sind.

6.7 Beispiele für DL-Anfragen aus einem Logikprogramm

Wir wollen anhand eines etwas ausführlicheren Beispiels die Funktionsweise unseres Systems präsentieren. Dabei greifen wir ein Beispiel aus dem Artikel [ELST04a] von Eiter et. al. auf, um zu zeigen, wie unser System die dort beschriebene Problematik löst. Als Beispiel zur Erläuterung dieses Abschnitts erfüllt es seine Funktion jedoch auch ohne Kenntnis des oben genannten Artikels.

Nehmen wir an, wir suchen einen Experten für ein wissenschaftliches Thema. Dafür steht uns eine Wissensbasis mit Daten über die Publikationen verschiedener Autoren zur Verfügung. Diese enthält eine Relation *author*, die Autoren mit ihren Werken verknüpft. Außerdem gibt es die Relation *area*, die einem Werk ein bestimmtes Forschungsgebiet zuweist. Wir suchen nun Autoren, die mindestens zwei Publikationen in einem Forschungsgebiet veröffentlicht haben. Als Regel mit DL-Atomen lässt sich das so formulieren:

$$\begin{aligned} expert(X, A) \leftarrow\ & dl_{has\_role}(X, author, Y), \\ & dl_{has\_role}(X, author, Z), \\ & dl_{has\_role}(Y, area, A), \\ & dl_{has\_role}(Z, area, A), \\ & neq(Y, Z). \end{aligned}$$

Das Prädikat $neq(Y, Z)$ modelliert dabei, dass $Y \neq Z$.

Unter Verwendung der in diesem Kapitel vorgestellten Tools und Bibliotheken, lässt sich dies wie folgt in Prolog umsetzen:

```
1  expert(X,A,TVHead) :- fw_and(TV1,TV2,TV3),
2                        fw_related(X,author,Y,TV1),
3                        fw_related(X,author,Z,TV2),
4                        fw_and(TV3,TV4,TV5),
5                        fw_related(A,area,Y,TV4),
6                        fw_and(TV5,TV6,TVHead),
7                        fw_related(A,area,Z,TV6),
8                        Y\==Z.
```

Die Zeilen 2,3,5 und 7 setzen die Anfragen an einen DL-Reasoner um. Wie bereits erwähnt, werden diese intern teilweise durch mehrere Aufrufe des DL-Reasoners realisiert. Unsere Bibliothek ermöglicht jedoch, wie im Beispiel zu sehen ist, eine eins zu eins Korrespondenz zwischen den DL-Atomen in der Regel oben und den im Programm verwendeten Prädikaten.

Die Zeilen 1, 4 und 6 setzen die logische Verknüpfung der Prädikate basierend auf $\mathcal{FOUR}$ um. Im Gegensatz zu den standardisierten DIG-Anfragen kann unsere Umsetzung der DL-Anfragen, wie in Abschnitt 6.5.2 beschrieben,

auch bottom als Ergebnis liefern. So lassen sich durch ein einzelnes DL-Atom mehr Informationen aus der Wissensbasis gewinnen. Die Verwendung von $\mathcal{FOUR}$ in Prolog ermöglicht, diese auch weiterverarbeiten zu können.

Zeile 7 modelliert die Ungleichheit der *areas* mit dem dafür gebräuchlichen Ausdruck aus Prolog. Dies demonstriert, dass unsere Implementierung von $\mathcal{FOUR}$ und DL-Anfragen mit bekannten Prozeduren aus Prolog zusammen funktioniert.

Als zweites Beispiel wollen wir nochmals auf das Programm aus Abschnitt 5.3.3 zurückgreifen:

$$\begin{aligned} is\_suspect(X) &\leftarrow has\_motive(X) \vee has\_witness(X) \\ is\_cleared(X) &\leftarrow \neg contradict\_alibi(X) \wedge has\_alibi(X) \\ charge(X) &\leftarrow is\_suspect(X) \oplus \neg is\_cleared(X) \end{aligned}$$

Wir haben in Abschnitt 5.3.3 gesehen, dass die zweite Zeile dieses Programms nach der Offenen-Welt-Annahme konzipiert ist, die auch Beschreibungslogiken zugrunde liegt. Folglich lassen sich die hier verwendeten Atome durch DL-Anfragen realisieren.

Nehmen wir an, wie haben eine Wissensbasis mit diversen Informationen über Aussagen, Aufenthaltsorte, Bekanntschaften etc. von Personen. Diese werden genutzt, um ein Konzept *alibi* zu definieren, das alle Personen enthält, die ein Alibi haben. Nun kann $has\_alibi(X)$ im Programm ersetzt werden durch $dl_{instance}(X, alibi)$. Betrachten wir dessen Auswertung, so zeigt sich, dass erwartungsgemäß der Wert `true` ausgegeben wird, wenn X in *alibi* ist. Der Wert `false` wird nur ausgegeben, wenn X in $\neg alibi$ ist, wenn also wirklich sichergestellt ist, dass X kein Alibi hat. Ist über X nichts bekannt, wird $\bot$ ausgegeben. Dies entspricht genau den Werten, die auch von $has\_alibi(X)$ verlangt wurden, damit das Programm erwartungsgemäß arbeitet. Es zeigt sich also, dass die Anfrage $dl_{instance}(X, alibi)$ die Offene-Welt-Annahme des DL-Reasoners konsistent ins Logikprogramm transportiert, welches diese durch die entsprechende Grundannahme auch weiterverarbeiten kann, wie in Abschnitt 5.3.3 gezeigt wurde.

Kapitel 7

Zusammenfassung, verwandte Arbeiten und Ausblick

7.1 Zusammenfassung

Die H-fundierten Modelle von Loyer und Straccia erweitern die bekannten auf Fixpunkten monotoner Operatoren basierenden Semantiken für Logikprogramme um eine interessante und flexible Alternative, die sich nicht zuletzt dadurch auszeichnet, viele etablierte Semantiken abbilden zu können und damit neue Erkenntnisse über deren Zusammenhang zu gewinnen. Als erstes zentrales Resultat unserer Arbeit ist es gelungen, diese Semantik in den von Hitzler und Wendt konzipierten allgemeinen Formalismus basierend auf *level-mappings* einzubetten. Damit wurde deren Anspruch untermauert, eine Beschreibungsmethode zu entwickeln, die möglichst viele Semantiken modellieren kann und Schlüsse auf deren Verhältnis zulässt. Wir haben dazu die bestehende Definition von *level-mappings* konsistent erweitert, um auch mit Interpretationen auf beliebigen Biverbänden zurechtzukommen, die einem Atom während der Iteration eines Fixpunktoperators beliebig viele unterschiedliche Werte zuweisen können, bevor der endgültige erreicht ist. Es ist uns gelungen, die Zusammenhänge, die Loyer und Straccia durch ihre Analyse von Fixpunktoperatoren erkannt haben, in den Beschreibungen durch *level-mappings* nachzuvollziehen. Damit wurde ein Berührungspunkt zwischen den Forschungsrichtungen basierend auf Biverbänden zum einen, *level-mappings* zum anderen geschaffen: Unsere Charakterisierung verwendet *level-mappings*, integriert dabei aber die Operatoren und die Wissensordnung eines Biverbandes als zentrale Elemente. Basierend auf den Erkenntnissen über die H-fundierten Modelle und deren Zusammenhang mit den Kripke-Kleene-Modellen haben wir die Hypothesenfamilie KKS konzipiert, die es erlaubt, die Geschlossene- und Offene-Welt-Annahme in einem Logikprogramm zu kombinieren, zu dessen Auswertung aber lediglich ein System benötigt wird, das die Kripke-Kleene-Semantik berechnet.

Wir haben die Logik $\mathcal{FOUR}$ für Prolog realisiert. Die erweiterte semantische Aussagekraft der H-fundierten Modelle unter KKS-Hypothesen wurde genutzt, um Anfragen an *description logic* Reasoner elegant an dieses Prologsystem anzubinden. Somit gelang es, eine bekannte Umgebung für Logikprogrammierung mit den Möglichkeiten der Beschreibungslogik zu verbinden.

7.2 Verwandte Arbeiten

Unsere Arbeit berührt zwei Felder, in denen aktuelle Forschung betrieben wird: zum einen die Kombination von Offener- und Geschlossener-Welt-Annahme, zum anderen die Kombination von Beschreibungslogik und Logikprogrammierung. Beide Arbeitsgebiete sind verwandt.

Im ersten Bereich zeigt sich ein enger Zusammenhang zwischen der jeweiligen Weltannahme und der verwendeten Form der Negation, entweder als explizite Negation oder als *negation as failure*. Dieser taucht mehr oder weniger konkret in den meisten Arbeiten zum Thema auf. Loyer und Straccia zeigen in [LS05], dass sich durch die passenden Hypothesen für die entsprechenden Atome die Negation im Biverband sowohl wie die explizite Negation als auch wie die *negation as failure* verhalten kann. Eine andere Herangehensweise an diese Problematik sind so genannte *extended logic programs* [AP92, GL91]. Diese benutzen beide Formen der Negation getrennt mit zwei verschiedenen Operatoren. Für diese Programme beschreibt Arieli in [Ari02], motiviert durch Überlegungen aus dem *paraconsistent reasoning*, eine Semantik, die ebenfalls auf der Wissensordnung eines Biverbandes basiert und eine Generalisierung der Wohlfundierten Semantik darstellt. Die Modelle werden ähnlich wie die der *stable model semantics* [GL88] berechnet, indem durch eine Reduktion erst alle Vorkommnisse der *negation as failure* entfernt werden, und dann das Modell berechnet wird. Der Wertebereich ist ein Biverband, dabei wird der Modellbegriff jedoch anders als in unserer Arbeit über Primfilter in diesem Verband definiert. Dies geht zurück auf Betrachtungen von Belnap [Bel76], die besagen, dass ein Atom in einem Modell „mindestens wahr“ sein sollte: In $\mathcal{FOUR}$ ist der entsprechende Primfilter $\{\top, \texttt{true}\}$. Außerdem verwendet Arieli priorisierte Regeln. Nach der Intuition von Loyer und Straccia sollte dieser zusätzliche Formalismus nicht nötig sein, wenn man als Wertebereich etwa einen Biverband benutzt, der Werte für „Vertrauen“ oder Konfidenz durch Intervalle modelliert, statt exakte Wahrheitswerte zu verwenden ([LS05] S.374). Dort (S.373) skizzieren die Autoren auch eine Transformation von *extended logic programs* in ihren Formalismus. Arieli stellt also eine Semantik vor, die durch Forschungen im Bereich des *paraconsistent reasoning* motiviert ist, aber teilweise dennoch sehr ähnliche Ideen verwendet, wie die hier untersuchte Semantik von Loyer und Straccia. Es wäre interessant zu untersuchen, ob eine Modellierung der Arbeit von Arieli

im Framework mit *level-mappings* hier Aufschlüsse bringen könnte.

Damásio et. al. verwenden in [DAAW06] ebenfalls zwei Arten von Negation. Die Zuweisung von offener und geschlossener Weltannahme beruht auf Axiomen aus [AP92]. Für ein offenes Atom A wird $\bar{A} \leftarrow not(A)$ oder $A \leftarrow not(\bar{A})$ eingefügt, für ein geschlossenes Atom alle beide. Dabei bezeichnet $\bar{A}$ die explizite Negation, während $not(A)$ für *negation as failure steht.* Damásio et. al. bauen darauf jedoch einen gegenüber den bisher vorgestellten Arbeiten deutlich pragmatischeren Ansatz auf, der dem Benutzer einer Anwendung des *semantic web* ermöglicht, verschiedene Wissensbasen modular zu benutzen, und die darin enthaltenen Prädikate durch diverse Schlüsselwörter mit unterschiedlichen Gültigkeitsbereichen zu deklarieren. Da die Zuweisung von unterschiedlichen Hypothesen für Atome nach der Arbeit von Loyer und Straccia einen ähnlichen Effekt hat, wäre zu untersuchen, ob sich deren Semantik nicht auch oder sogar besser als Grundlage für ein solches System anbietet.

Einen Aspekt der Offenen-Welt-Annahme, der näher an deren Auffassung in Bezug auf die Prädikatenlogik ist (s.a. Abschnitt 3.5.3), betrachten Heymans, Van Nieuwnborgh und Vermeir in [HVV05, HNV04]. Ihre Analyse zeigt, dass die übliche Vorgehensweise, Semantiken auf Grundinstanzen zu berechnen, nicht immer die Idee hinter einem Programm erfasst. Demonstriert werden soll dies an folgendem Beispiel[1] aus [HNV04], das modelliert, dass Manager große Autos fahren:

$$\begin{aligned} bigCar(X) &\leftarrow Manager(X) \\ Manager(X) \vee \neg Manager(X) &\leftarrow \\ \neg Manager(felix) &\leftarrow \end{aligned}$$

Da nun $felix$ die einzige Konstante im Programm ist, betrachtet eine Semantik, die mit Grundinstanzen arbeitet, eben nur diese. Die von den Autoren analysierte *answer set semantic* etwa hat als Modell $\{\neg Manager(felix)\}$. Es wäre denkbar, dass andere Semantiken auch noch Aussagen über den Wahrheitswert von $bigCar(felix)$ machen. Jedoch ist aus keinem Modell ersichtlich, dass es durchaus Manager geben kann, die große Autos fahren. Um dies zu ermöglichen, wird das *grounding* über einer erweiterten und potentiell unendlichen Domäne durchgeführt. Regeln wie die zweite Zeile im Beispiel dienen dann dazu, die entsprechenden Literale ins Modell zu integrieren. Dieses Vorgehen wird als *open answer set programming* bezeichnet. Dabei erscheint das Problem, dass die Erfüllbarkeit mit offener Domäne nicht entscheidbar ist. Die Autoren beschränken sich deshalb auf so genannte *local conceptual logic programs*, die auf *answer set programming* reduziert werden

[1] Ein Programm in dieser Form ist in unserer Arbeit nicht erlaubt, da weder Konjunktionen noch Negationen im Kopf einer Regel vorkommen dürfen. Die angesprochene Problematik gilt jedoch auch für andere Formen von Logikprogrammen.

können, und damit entscheidbar sind. In [HVV05] wird stattdessen ein Fragment der *fixed point logic* verwendet, das ebenfalls entscheidbar ist. In einem weiteren Schritt gelang es den Autoren in [HVFV06] mit einer Logik, die *open answer set programming* entscheidbar realisiert, eine Beschreibungslogik zu simulieren, also ein Fragment der Prädikatenlogik erster Stufe. Es ist in dieser bisher allerdings nicht möglich, den in Beschreibungslogiken üblichen Operator zur Einschränkung der Anzahl ($\leq n$) der Individuen, die eine Rolle erfüllen, auszudrücken.

Unser System zur Kombination von Beschreibungslogik und Logikprogrammierung, der durch Atome in Logikprogrammen realisiert ist, verwendet einen ähnlichen Ansatz wie die Arbeit [ELST04a] von Eiter et. al. Dort werden ebenfalls ausgezeichnete Atome verwendet, die eine Anfrage an ein DL-System realisieren. Diese arbeiten mit der zweiwertigen *answer set semantics*, die entsprechend erweitert wurde, um die DL-Anfragen zu integrieren. Dabei ist jedoch im Gegensatz zu unserem System der Informationsfluss in beide Richtungen möglich. Dies gelingt, da in der verwendeten Semantik die Monotonie der meisten DL-Anfragen auch nach Änderungen an der Wissensbasis gewährt werden kann. In [EIST06] erweitern die Autoren dieses Konzept auf die mächtigen HEX-Programme, die sowohl Konjunktionen in Köpfen als auch Atome höherer Ordnung zulassen und insbesondere nicht nur DL-Anfragen, sondern jede entscheidbare externe Berechnung über spezielle Atome einbinden können. Um die Entscheidbarkeit zu garantieren, werden gewisse Einschränkungen für erlaubte Regeln gemacht. Das Programm wird mit einem sogenannten *Splitting-Algorithmus* in mehrere Teile unterteilt, die dann unter der *answer set semantics* verarbeitet werden können. Nachteil dieses Ansatzes ist, dass die *answer set semantics* keine Inkonsistenzen modellieren kann: Tritt eine Inkonsistenz auf, existiert kein *answer set*.

Umberto Straccia präsentiert in [Str06] einen Resolutionsalgorithmus zur Beantwortung von Anfragen nach dem Wahrheitswert einzelner Atome eines Programms unter der H-fundierten Semantik. Dieser arbeitet mit einer Sprache ohne Funktionssymbole. In den Rümpfen von Regeln sind neben den Operatoren aus einem Biverband beliebige berechenbare Funktionen zur Kombination der Wahrheitswerte der Atome erlaubt, was die Programme gegenüber [LS05] nochmals erweitert. Es wird, um die Terminierung der Berechnung in endlich vielen Schritten zu garantieren, von endlichen Biverbänden als Wertebereich ausgegangen. Der Algorithmus transformiert das Logikprogramm in ein Gleichungssystem. Dazu wird jedem Grundatom eine Variable zugewiesen und jedem Rumpf eine Funktion in den Variablen, die den darin vorkommenden Atomen entsprechen. Eine Anfrage nach dem Wahrheitswert eines Atoms wird dann mit einem *top-down* Verfahren gelöst. Es wird also nicht wie in der formalen Definition der Semantik mittels eines Fixpunktoperators das ganze Modell berechnet – obwohl dies im *worst case* immer noch

notwenig sein kann. Stattdessen werden ausgehend vom angefragten Atom nur Regeln ausgewertet, die für dessen Wahrheitswert von Bedeutung sind. Dieser Algorithmus bietet alle Möglichkeiten der sehr flexiblen Jede-Welt-Annahme und der H-fundierten Semantik. Im Gegensatz dazu hat unser für Prolog realisiertes System die bekannten Probleme für Prologprogramme, wie etwa unendliche Rekursionen bei ungeeigneter Programmierung. Unsere Motivation jedoch war die Möglichkeit, ein bekanntes und verbreitetes System elegant erweitern zu können. Es wäre zu untersuchen, wie sich der Algorithmus von Straccia mit Anfragen an DL-Systeme verbinden lässt, da wir die H-fundierte Semantik aufgrund ihrer Flexibilität für sehr geeignet halten, um mit Informationen aus DL-Systemen umzugehen.

7.3 Ausblick

Unsere Arbeit bietet Ansatzpunkte für weitere Forschung, darunter einige konkrete, aber auch theoretische Fragen.

Wir haben ein System realisiert, das aus einer Umgebung für Logikprogrammierung Anfragen an einen DL-Reasoner stellen kann. Es wäre wünschenswert, wenn auch die Informationsübertragung in die andere Richtung, also aus dem Logikprogramm in die Wissensdatenbank möglich wäre. Um dies in der H-fundierten Semantik erfassen zu können, müsste garantiert sein, dass die Ergebnisse der DL-Anfragen bei Änderungen der Wissensdatenbank monoton in der Wissensordnung sind. Unter diesen Umständen kann die von uns formulierte Erweiterung der H-fundierten Semantik auch solche Informationen mit einbeziehen.

Ein weiterer Ansatzpunkt wäre die Erweiterung unseres Systems von $\mathcal{FOUR}$ auf einen aussagekräftigeren Biverband, etwa den Intervallverband $\langle [0,1] \times [0,1], \leq_t, \leq_k \rangle$. Damit gäbe es dann – neben vielen anderen – die Möglichkeit unterschiedliche Wissensquellen verschieden zu priorisieren.

Für die Modellierung der H-fundierten Semantik mit *level-mappings* haben wir zweistellige *level-mappings* eingeführt und damit das bestehende Konzept erweitert. In den Beweisen bei Hitzler und Wendt [HW05] und auch in unserem wird anhand des Levels die Iteration des Fixpunktoperators nachempfunden: Es wird eine Korrespondenz zwischen dem Iterationsschritt, in dem einem Atom ein Wert zugewiesen wird und dessen Level hergestellt. Da sich nun bei der Iteration auf einem Biverband der Wert eines Atoms mehrmals und theoretisch beliebig oft (die Iteration ist im Allgemeinen transfinit) ändern kann, verwenden wir das zweistellige Mapping, um dies nachvollziehen zu können. Die Definition der H-fundierten Modelle benutzt jedoch lediglich Eigenschaften des Modells, namentlich die, dass es seinen eigenen Support einschließt. Wie dieses Modell berechnet wurde, spielt dabei kei-

ne Rolle. Folglich sollte es möglich sein, auch die H-fundierten Modelle mit einem einwertigen *level-mapping* zu beschreiben. Diesem könnte eine Korrespondenz zwischen dem Level eines Atoms und dem Iterationsschritt, bei dem sich dessen Wahrheitswert zum letzten Mal ändert, zu Grunde liegen.

Schließlich wollen wir noch auf eine Fragestellung hinweisen, auf die wir bei der Beschäftigung mit den für diese Arbeit zentralen Artikeln von Loyer und Straccia [LS05, LS04, LS03] zum einen und von Hitzler und Wendt [HW05, Hit05] zum anderen aufmerksam geworden sind. Beide kommen unter anderem zu dem Ergebnis, dass die Wohlfundierte Semantik eine „ergänzte“ Form der Kripke-Kleene Semantik ist. Loyer und Straccia formulieren die Wohlfundierte Semantik als Fixpunkt des Operators $W_P(I) = T_P(I) \oplus \neg.U_P(I)$. Es ist auch möglich diesen wie folgt zu schreiben: $W_P(I) = \Phi_P(I) \oplus \neg.U_P(I)$. Es handelt sich also um die Kripke-Kleene Semantik, ergänzt mit den Informationen aus der Unfundierten Menge.

Hitzler und Wendt kommen dagegen zu dem Ergebnis, dass die wohlfundierte Semantik eine *stratifizierte* Version der Kripke-Kleene Semantik darstellt. Das Konzept der stratifizierten Programme geht zurück auf Arbeiten von Apt et. al. [ABW88] und Przymusinski [Prz88]. Ein klassisches Logikprogramm heißt *lokal stratifiziert*, wenn es ein einstelliges *level mapping* $l : \mathcal{B}_P \to \alpha$ für eine Ordinalzahl α gibt, so dass für jede Regel der Form $A \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ in $ground(P)$ gilt, dass $l(A) \geq l(A_i)$ und $l(A) > l(B_j)$. Diese Bedingung wurde ursprünglich erdacht, um Rekursion durch negierte Atome zu verhindern (vgl. [HW05]). Der Vergleich mit den Definitionen in Abschnitt 3.4 zeigt, dass die Definition der Wohlfundierten Semantik sich in der Tat durch die oben beschriebene Struktur von der Fitting-Semantik unterscheidet.

Zwei unterschiedliche Konzepte wie die Unfundierte Menge und stratifizierte Programme können also anscheinend ähnliche Informationen modellieren. Die Untersuchung dieses Zusammenhangs könnte einen Ansatz für weitere Erkenntnisse über Semantiken von Logikprogrammen bieten.

Literaturverzeichnis

[AA98] Arieli, Ofer und Arnon Avron: *The Value of the Four Values.* Artificial Intelligence, 102(1):97–141, 1998.

[ABW88] Apt, Krzysztof R., Howard A. Blair und Adrian Walker: *Towards a Theory of Declarative Knowledge.* In: Minker, Jack (Herausgeber): *Foundations of Deductive Databases and Logic Programming*, Seiten 89–142. Morgan Kaufmann, Los Altos, 1988.

[AP92] Alferes, José Júlio und Luís Moniz Pereira: *On Logic Program Semantics with Two Kinds of Negation.* In: Apt, Krzysztof (Herausgeber): *Proceedings of the Joint International Conference and Symposium on Logic Programming (JICSLP-92)*, Seiten 574–588. MIT Press, Cambridge, 1992.

[Apt97] Apt, Krzysztof R.: *From Logic Programming to Prolog.* Prentice-Hall International Series in Computer Science. Prentice Hall, Hemel Hempstead, 1997.

[Ari02] Arieli, Ofer: *Paraconsistent declarative semantics for extended logic programs.* Annals of Mathematics and Artificial Intelligence, 36(4):381–417, 2002.

[AvE82] Apt, Krizysztof R. und Maarten H. van Emden: *Contributions to the Theory of Logic Programming.* Journal of the ACM, 29(3):841–862, 1982.

[BBH+90] Baader, Franz, Hans-Jürgen Bürckert, Jochen Heinsohn, Bernhard Hollunder, Jürgen Müller, Bernhard Nebel, Werner Nutt und Hans-Jürgen Profitlich: *Terminological Knowledge Representation: A Proposal for a Terminological Logic.* Technischer Bericht TM-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz Kaiserslautern, 1990.

[BCM+03] Baader, Franz, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi und Peter Patel-Schneider (Heraus-

geber): *The Description Logic Handbook: Theory, Implementation and Applications.* Cambridge University Press, Cambridge, England, 2003.

[Bec] Bechhofer, Sean: *DIG 2.0: The DIG Description Logic Interface Document Index.* Online im WWW, Version vom 18.5.2006, URL: `http://homepages.cs.manchester.ac.uk/~seanb/dig/index.html` (abgerufen am 21.8.2006). DIG Working Group Note.

[Bec03] Bechhofer, Sean: *The DIG description logic interface: DIG/1.1.* Technischer Bericht, University of Manchester, 2003.

[Bel76] Belnap, Jr., Nuel D.: *How a Computer Should Think.* In: Ryle, Gilbert (Herausgeber): *Contemporary Aspects of Philosophy*, Seiten 30–56. Oriel Press, Stocksfield, 1976.

[Bel77] Belnap, Nuel D.: *A useful four-valued logic.* In: Epstein, G. und J. M. Dunn (Herausgeber): *Modern Uses of Multiple-Valued Logic*, Seiten 5–37. Reidel, Dordrecht, Netherlands, 1977.

[BLFD99] Berners-Lee, Tim, Mark Fischetti und Michael Dertouzos: *Weaving the Web.* HarperCollins, San Francisco, California, 1999.

[BLHL01] Berners-Lee, Tim, James Hendler und Ora Lassila: *The semantic web.* Scientific American, 284(5):34–43, 2001.

[Cla78] Clark, Keith L.: *Negation as Failure.* In: Gallaire, H. und J. Minker (Herausgeber): *Logics and Data Bases*, Seiten 293–322. Plenum Press, New York, 1978.

[DAAW06] Damásio, Carlos Viegas, Anastasia Analyti, Grigoris Antiniou und Gerd Wagner: *Supporting Open and Closed World Reasoning on the Web.* In: *Principles and Practice of Semantic Web Reasoning (PPSWR06), Proceedings.* Springer, Berlin/Heidelberg, 2006.

[EIST06] Eiter, Thomas, Giovambattista Ianni, Roman Schindlauer und Hans Tompits: *Effective Integration of Declarative Rules with External Evaluations for Semantic-Web Reasoning.* In: Sure, York und John Domingue (Herausgeber): *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006, Proceedings*, Band 4011 der Reihe *Lecture Notes in Computer Science*, Seiten 273–287. Springer, Berlin/Heidelberg, 2006.

[ELST04a] Eiter, Thomas, Thomas Lukasiewicz, Roman Schindlauer und Hans Tompits: *Combining Answer Set Programming with Description Logics for the Semantic Web.* In: Dubois, Didier, Christopher A. Welty und Mary-Anne Williams (Herausgeber): *KR2004: Principles of Knowledge Representation and Reasoning*, Seiten 141–151. AAAI Press, Menlo Park, California, 2004.

[ELST04b] Eiter, Thomas, Thomas Lukasiewicz, Roman Schindlauer und Hans Tompits: *Well-Founded Semantics for Description Logic Programs in the Semantic Web.* In: Antoniou, Grigoris und Harold Boley (Herausgeber): *Rules and Rule Markup Languages for the Semantic Web: Third International Workshop, RuleML 2004, Hiroshima, Japan, November 8, 2004. Proceedings*, Band 3323 der Reihe *Lecture Notes in Computer Science*, Seiten 81–97. Springer, Berlin/Heidelberg, 2004.

[Fit85] Fitting, Melvin: *A Kripke-Kleene Semantics for Logic Programs.* Journal of Logic Programming, 2(4):295–312, 1985.

[Fit90] Fitting, Melvin C.: *Bilattices in Logic Programming.* In: *20th International Symposium on Multiple-Valued Logic, Charlotte*, Seiten 238–247. IEEE CS Press, Los Alamitos, 1990.

[Fit91a] Fitting, Melvin: *Bilattices and the semantics of logic programming.* Journal of Logic Programming, 11(2):91–116, 1991.

[Fit91b] Fitting, Melving: *Kleene's Logic, Generalized.* Journal of Logic and Computation, 1(6):797–810, 1991.

[Fit93] Fitting, Melvin: *The Family of Stable Models.* Journal of Logic Programming, 17(2/3&4):197–225, 1993.

[Fit02] Fitting, Melvin: *Fixpoint semantics for logic programming A survey.* Theoretical Computer Science, 278(1-2):25–51, 2002.

[Gin86] Ginsberg, Matthew L.: *Multi-Valued Logics.* In: *Proceedings of AAAI-86, 5th National Conference on Artificial Intelligence*, Seiten 243–247. Morgan Kaufmann Publishers, San Francisco, California, 1986.

[Gin88] Ginsberg, Matthew L.: *Multivalued Logics: A Uniform Approach to Inference in Artificial Intelligence.* Computational Intelligence, 4(3):265–316, 1988.

[GL88] Gelfond, Michael und Vladimir Lifschitz: *The Stable Model Semantics for Logic Programming.* In: Kowalski, Robert A. und Kenneth Bowen (Herausgeber): *Proceedings of the Fifth International Conference on Logic Programming*, Seiten 1070–1080. The MIT Press, Cambridge, Massachusetts, 1988.

[GL91] Gelfond, Michael und Vladimir Lifschitz: *Classical Negation in Logic Programs and Disjunctive Databases.* New Generation Computing, 9(3/4):365–386, 1991.

[Hit05] Hitzler, Pascal: *Towards a Systematic Account of Different Semantics for Logic Programs.* Journal of Logic and Computation, 15(3):391–404, 2005.

[HNV04] Heymans, Stijn, Davy Van Nieuwenborgh und Dirk Vermeir: *Semantic Web Reasoning with Conceptual Logic Programs.* In: Antoniou, Grigoris und Harold Boley (Herausgeber): *Rules and Rule Markup Languages for the Semantic Web: Third International Workshop, RuleML 2004, Proceedings*, Band 3323 der Reihe *Lecture Notes in Computer Science*, Seiten 113–127. Springer, Berlin/Heidelberg, 2004.

[HPS03] Horrocks, Ian und Peter F. Patel-Schneider: *Reducing OWL Entailment to Description Logic Satisfiability.* In: Fensel, Dieter, Katia P. Sycara und John Mylopoulos (Herausgeber): *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Proceedings*, Band 2870 der Reihe *Lecture Notes in Computer Science*, Seiten 17–29. Springer, Berlin/Heidelberg, 2003.

[HST99] Horrocks, Ian, Ulrike Sattler und Stephan Tobies: *Practical Reasoning for Expressive Description Logics.* In: Ganzinger, H., D. McAllester und A. Voronkov (Herausgeber): *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, Band 1705 der Reihe *Lecture Notes in Artificial Intelligence*, Seiten 161–180. Springer, Berlin/Heidelberg, 1999.

[HV03] Huang, Zhisheng und Cees Visser: *An Extended DIG Description Logic Interface for Prolog.* Technischer Bericht SEKT TR D3.4.1.2, 2003.

[HVFV06] Heymans, Stijn, Davy Van Nieuwenborgh, Dieter Fensel und Dirk Vermeir: *Reasoning with the Description Logic $\mathcal{DLRO}^{-\{\leq\}}$ using Bound Guarded Programs.* In: *Reasoning on the Web Workshop at WWW2006*, Edinburgh, 2006.

[HVV05] Heymans, Stijn, Davy Van Nieuwenborgh und Dirk Vermeir: *Guarded Open Answer Set Programming.* In: *8th International Conference on Logic Programming and Non Monotoicn Reasoning*, Band 3662 der Reihe *Lecture Notes in Artificial Intelligence*, Seiten 92–104. Springer, Berlin/Heidelberg, 2005.

[HW05] Hitzler, Pascal und Matthias Wendt: *A uniform approach to logic programming semantics.* Theory and Practice of Logic Programming, 5(1-2):93–121, 2005.

[Kow74] Kowalski, Robert: *Predicate logic as a programming language.* In: *Proceedings of the IFIP-74 Congress*, Seiten 569–574. North-Holland, Amsterdam, 1974.

[Kun87] Kunen, Kenneth: *Negation in Logic Programming.* Journal of Logic Programming, 4(4):289–308, 1987.

[Kun88] Kunen, Kenneth: *Some Remarks on the Completed Database.* In: Kowalski, Robert A. und Kenneth A. Bowen (Herausgeber): *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, Seiten 978–992. The MIT Press, Cambridge, Massachusetts, 1988.

[LS03] Loyer, Yann und Umberto Straccia: *The Well-Founded Semantics of Logic Programs over Bilattices: an Alternative Characterisation.* Technischer Bericht, ISTI, 2003.

[LS04] Loyer, Yann und Umberto Straccia: *Epistemic Foundation of the Well-Founded Semantics over Bilattices.* In: Fiala, Jirí, Václav Koubek und Jan Kratochvíl (Herausgeber): *Mathematical Foundations of Computer Science 2004, 29th International Symposium, MFCS 2004, Prague, Czech Republic, August 22-27, 2004, Proceedings*, Band 3153 der Reihe *Lecture Notes in Computer Science*, Seiten 513–524. Springer, Berlin/Heidelberg, 2004.

[LS05] Loyer, Yann und Umberto Straccia: *Any-world assumptions in logic programming.* Theoretical Computer Science, 342(2-3):351–381, 2005.

[MvH] McGuinness, Deborah L. und Frank van Harmelen: *OWL Web Ontology Language Overwiev.* Online im WWW, Version vom 10.2.2004, URL: `http://www.w3.org/TR/2004/REC-owl-features-20040210/` (abgerufen am 21.8.2006). W3C Recommendation.

[Prz88] PRZYMUSINSKI, TEODOR: *On the declarative semantics of deductive databases and logic programs.* In: MINKER, JACK (Herausgeber): *Foundations of Deductive Databases*, Kapitel 5, Seiten 193–216. Morgan Kaufmann, Los Altos, CA, 1988.

[SSS91] SCHMIDT-SCHAUSS;, MANFRED und GERT SMOLKA: *Attributive Concept Descriptions with Complements.* Artificial Intelligence, 48(1):1–26, 1991.

[Str06] STRACCIA, UMBERTO: *Query Answering under the Any-World Assumption for Normal Logic Programs.* In: DOHERTY, PATRICK, JOHN MYLOPOULOS und CHRISTOPHER A. WELTY (Herausgeber): *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, Seiten 329–339. AAAI Press, Menlo Park, California, 2006.

[Tar55] TARSKI, ALFRED: *A Lattice-Theoretic Fixpoint Theorem and Its Applications.* Pacific Journal of Mathematics, 5(2):285–309, 1955.

[vK76] VAN EMDEN, MAARTEN H. und ROBERT A. KOWALSKI: *The semantics of predicate logic as a programming language.* Journal of the ACM, 23(4):733–742, 1976.

[vRS91] VAN GELDER, ALLEN, KENNETH ROSS und JOHN S. SCHLIPF: *The Well-Founded Semantics for General Logic Programs.* Journal of the ACM, 38(3):620–650, 1991.