Die Stable Model Semantik für Logikprogramme

Seminararbeit

Seminar Logikprogrammierung
Sommersemester 2004

Lutz Böhne
Matrikelnummer: 227424

Lehrstuhl für Informatik II,
RWTH Aachen, Ahornstraße 55,
52074 Aachen, Germany
Betreuer: Dipl.-Inform. Benedikt Bollig

Literatur

[1] K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Kaufmann, Los Altos, CA, 1988.

[2] Weidong Chen and David Scott Warren. Computation of stable models and its integration with logical query processing. *Knowledge and Data Engineering*, 8(5):742–757, 1996.

[3] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

[4] T. C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Kaufmann, Los Altos, CA, 1988.

[5] Allen van Gelder, Kenneth Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

Inhaltsverzeichnis

1 Einleitung

Logikprogramme mit Negation sind ein wichtiges Hilfsmittel in der künstlichen Intelligenz, da sie es ermöglichen, Wissen zu repräsentieren und aus diesem Wissen (automatisiert) Schlüsse zu ziehen. Doch leider ist die Semantik eines Logikprogramms mit Negation nicht immer eindeutig, so dass großer Aufwand getrieben wird, Semantiken zu finden, die auf eine möglichst große Klasse von Logikprogrammen anwendbar ist. Gelfond und Lifschitz haben 1988 mit der Beschreibung der Stable Model Semantik in [3] einen wichtige Ansatz vorgestellt, der neben Konformität mit anderen bis dato vorgestellten Semantiken darüberhinaus auch auf Logikprogramme anwendbar ist, denen diese Semantiken keine Bedeutung zuordnen.

In dieser Arbeit soll nun nach einer kurzen Bestimmung der verwendeten Terminologie ein Einblick in andere deklarative Semantiken für Logikprogramme gegeben werden, um dann die Stable Model Semantik vorzustellen, sie mit diesen anderen Ansätzen zu vergleichen und abschließend ein Verfahren vorzustellen, wie man den Aufwand der Berechnung von stabilen Modellen verkleinern kann.

2 Logikprogramme mit Negation

2.1 Terminologie

Im Folgenden sollen zunächst einige Begriffe definiert werden, die im weiteren Verlauf dieser Arbeit verwendet werden.

Definition (Signatur) Ein Paar $\Sigma = (F, P)$

- einer Menge $F = \bigcup_{n\in\mathbb{N}_0} F^{(n)}$ von *Funktionssymbolen* und
- einer Menge $P = \bigcup_{n\in\mathbb{N}} P^{(n)}$ eine Menge von *Prädikatssymbolen*

heißt *Signatur*. Ein nullstelliges Funktionssymbol $c \in F^{(0)}$ nennt man auch Konstantensymbol. Im Folgenden wird die Signatur, unter der ein Programm betrachtet wird, in der Regel durch die in einem Programm vorkommenden Prädikats- und Funktionssymbole bestimmt.

Definition (Σ-Terme) Die Menge der $T_\Sigma(X)$ der Σ-Terme über einer abzählbaren Variablenmenge X ist induktiv definiert durch:

- $X \subseteq T_\Sigma(X)$
- $f \in F^{(n)}$, $t_1, t_2, \ldots, t_n \in T_\Sigma(X) \curvearrowright f(t_1, t_2, \ldots, t_n) \in T_\Sigma(X)$

$T_\Sigma(\emptyset)$, also die Menge der variablenfreien Terme, heißt auch Menge der *Grundterme* über Σ.

Definition (Atome, Literale) Ein *Atom* A hat die Form $p(t_1, t_2, \ldots, t_n)$ für

- ein n-stelliges Prädikatssymbol p und
- Termen $t_1, t_2, \ldots, t_n$.

Ein Atom A nennt man auch *positives Literal*, die Negation $\neg A$ eines Atoms nennt man *negatives Literal*. Für ein Literal L bezeichne $\overline{L}$ das *Komplement* von L, also $\overline{\neg A} = A$ und $\overline{A} = \neg A$ für ein Atom A.

Definition (Regel) Eine *Regel* hat die Form

$$A \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_n$$

mit $n \geq 0$, einem Atom A und Literalen L_i. A heißt auch *Kopf* oder *Konsequenz*, $L_1 \wedge L_2 \wedge \ldots \wedge L_n$ heißt *Körper* oder *Prämisse* der Regel. Eine Regel, deren Körper keine negativen Literale enthält, nennt man *definite Regel*. Eine Regel mit $n = 0$ nennt man auch *Fakt*.

Definition (Logikprogramm) Eine Menge Π von Regeln heißt *Logikprogramm*. Sind alle Regeln in einem Logikprogramm Π definit, so nennt man Π auch ein *definites Logikprogramm* oder *Hornklauselprogramm*. Variablenfreie Atome (-Literale, Regeln, Programme) nennt man auch *Grund*atome (-Literale, Regeln, Programme).

Definition (Herbrand-Universum) Das *Herbrand-Universum* $\mathfrak{U}_\Pi$ eines Logikprogramms Π ist die Menge aller Grundterme, die sich aus in Π vorkommenden Konstanten- und Funktionssymbolen konstruieren lässt.

Dabei ist zu beachten, dass das Herbrand-Universum eines Logikprogramms Π leer ist, falls Π keinerlei Konstantensymbole enthält, und unendlich, falls Π Funktionssymbole der Stelligkeit ≥ 1 enthält. Den ersten Fall schließt man oft einfach dadurch aus, dass man dem Herbrand-Universum zur Not eine beliebige Konstante hinzufügt, der zweite Fall führt mitunter zu Schwierigkeiten bei Berechnungen auf Logikprogrammen, falls man diese auf der möglicherweise unendlichen Herbrand-Instanz eines Programms durchführt.

Definition (Herbrand-Basis) Die *Herbrand-Basis* $\mathfrak{B}_\Pi$ eines Programms Π ist die Menge aller Grundatome $p(t_1, t_2, \ldots, t_n)$ mit $t_i \in \mathfrak{U}_\Pi, 1 \leq i \leq n$ für alle Prädikatssymbole p, die in Π vorkommen.

Definition (Herbrand-Instanz) Die *Herbrand-Instanz* $\mathfrak{I}_\Pi$ eines Programms Π ist die (möglicherweise unendlich große) Menge aller Grundregeln, die man durch Substitution von Variablen in Regeln von Π durch Terme aus dem Herbrand-Universum von Π erhält.

Definition (Interpretation) Eine Menge I von Literalen heißt *widerspruchsfrei*, falls für jedes Literal gilt:

$$L \in I \curvearrowright \overline{L} \notin I$$

Eine *partielle Interpretation* I ist eine widerspruchsfreie Menge von Grundliteralen, die also die Bedingung

$$L \in I \curvearrowright (L = A \vee L = \neg A) \wedge A \in \mathfrak{B}_\Pi \wedge \overline{L} \notin I \tag{1}$$

erfüllt. Eine *totale Interpretation* I ist eine partielle Interpretation, die jedes Atom aus der Herbrand-Basis von Π oder dessen Negation enthält, die zusätzlich zu (1) also die Bedingung

$$\{A \in \mathfrak{B}_\Pi | A \in I\} \cup \{A \in \mathfrak{B}_\Pi | \neg A \in I\} = \mathfrak{B}_\Pi$$

erfüllt.

Für eine Interpretation I heißt ein Grundliteral L

- *wahr* in I (man sagt auch: L gilt in I), falls $L \in I$, es heißt
- *falsch* in I (man sagt auch: L gilt nicht in I), falls $\overline{L} \in I$.

Eine Regel wird durch eine Interpretation I

- *erfüllt*, falls der Kopf der Regel wahr oder eines der Teilziele in I falsch ist.
- *falsifiziert*, falls der Kopf der Regel wahr und alle Teilziele in I falsch sind.
- *schwach falsifiziert*, falls der Kopf und keines der Teilziele der Regel in I falsch sind.

Für eine Interpretation I enthalte die Menge $Pos(I)$ alle positiven Literale aus I und die Menge $Neg(I)$ alle negativen Literale aus I.

Definition (Modell eines Logikprogramms) Ein *totales Modell* eines Logikprogramms Π ist eine totale Interpretation, die jede Regel aus der Herbrand-Instanz von Π erfüllt. Ein *partielles Modell* eines Logikprogramms Π ist eine partielle Interpretation, die zu einem totalen Modell von Π ergänzt werden kann.

Gebräuchlich ist es auch, totale Interpretationen und Modelle durch die Menge der in ihnen enthaltenen positiven Literale (also Grundatome) zu repräsentieren und die negativen Literale einfach wegzulassen und alle in einer solchen Menge enthaltenen Atome als wahr, alle anderen als falsch zu interpretieren.

Definition (minimales Modell) Die Ordnungsrelation $\preceq$ sei für zwei totale Interpretationen I und J definiert durch

$$I \preceq J \equiv Pos(I) \subseteq Pos(J) \wedge Neg(I) \supseteq Neg(J)$$

Ein *minimales Modell* eines Programms Π ist ein totales Modell I von Π für das kein weiteres totales Modell J von Π existiert mit $J \preceq I$ und $J \neq I$.

2.2 Deklarative Semantiken

In dieser Arbeit sollen deklarative Semantiken für Logikprogramme mit Negation betrachtet werden. Eine deklarative Semantik beschreibt mathematisch-formal die Bedeutung eines Logikprogramms, häufig in Form eines minimalen Herbrand-Modells. Dieses Modell nennt man auch *beabsichtigtes* oder *kanonisches Modell* eines Logikprogramms.

Ziel einer solchen Semantik (oder auch eines solches Modelles) ist es normalerweise, einen Ausführungsmechanismus für ein Logikprogramm zu verifizieren, also ein Maß für die Korrektheit eines solchen Mechanismus zu bestimmen.

Für definite Logikprogramme (also Programme, in deren Regeln keine Negation auftritt) und eine Anfrage existieren mit der Fixpunkt-Semantik und der Semantik des kleinsten Herbrand-Modells zwei äquivalente deklarative Semantiken, für die mittels der Prozeduralen Semantik (also SLD-Resolution) ein korrekter Ausführungsmechanismus zur Verfügung steht

Für Logikprogramme mit Negation ist es hingegen nicht so einfach, kanonische Modelle zu finden, da sie mehrere minimale Herbrand-Modelle besitzen können und nicht immer klar ist, welches dieser Modelle *das* kanonisches Modell eines solchen Programms sein soll. Zudem gibt es verschiedene Möglichkeiten der Behandlung von Negation in Logikprogrammen. Klassische Negation erlaubt es, auch im Kopf einer Regel negative Literale zu verwenden und so explizit auch der Wahrheitswert eines negativen Literals aus einem Programm herleitbar ist.

Weitaus üblicher in der Logikprogrammierung, gerade wenn man an den Einsatz in deduktiven Datenbanken denkt, ist es, *Negation-by-Failure* zu verwenden. Dabei wird ein Literal $\neg A$ als wahr bewertet, wenn sich A nicht aus dem Programm

herleiten lässt. Das bietet den Vorteil, dass es genügt, positives Wissen in einem Logikprogramm aufzuschreiben.

Möchte man beispielsweise ein Logikprogramm verwenden, um Informationen über die Studierenden an einer Hochschule zu sammeln, ist die Anzahl der „positiven“ Informationen die man in das Programm aufnehmen muss, um einen Sachverhalt unter Verwendung von Negation-by-Failure vollständig zu beschreiben, wesentlich kleiner, als die der „negativen“ Informationen, die man bei der Verwendung klassischer Negation zusätzlich aufnehmen müsste. So reicht es beispielsweise aus, alle an einer Hochschule eingeschriebenen Studierenden als Fakten der Form $eingeschrieben(X)$ zu erfassen. Möchte man dann wissen, ob eine Person nicht an der Hochschule studiert, genügt es festzustellen, dass $eingeschrieben(X)$ nicht herleitbar ist. Bei der Verwendung von klassischer Negation müsste man für jede nicht eingeschriebene Person einen Fakt $\neg eingeschrieben(X)$ in das entsprechende Programm aufnehmen, was nicht praktikabel erscheint.

Allerdings bringt die Verwendung von Negation-By-Failure den Nachteil mit sich, dass bei der Auswertung eines negativen Literals sämtliche Versuche, das Komplement dieses Literals aus dem Programm herzuleiten, fehlschlagen müssen, was den Aufwand natürlich erheblich vergrößert.

Im Folgenden sollen einige Ansätze kanonische Modelle für Logikprogramme mit Negation (by failure) auszuwählen kurz vorgestellt werden, um später den Bezug zu der Stable Model Semantik herstellen zu können.

2.2.1 Iterative Fixpunktsemantik für geschichtete Logikprogramme

Apt, Blair und Walker haben in [1] die Klasse sogenannter geschichteter Logikprogramme beschrieben und eine Fixpunkt-Semantik für diese Klasse von Logikprogrammen vorgestellt, die für diese Programme ein kanonisches Modell auswählt.

Die Idee dahinter ist es, ein Programm in mehrere untereinander geordnete „Schichten“ zu partitionieren, so dass in einer Schicht nur solche Literale negiert im Körper einer Regel auftauchen, die ausschließlich in Regeln der darunterliegenden Schichten im Kopf auftreten um so rekursive Abhängigkeiten mit Negation zu vermeiden.

Terminologie Ein Relationssymbol p *bezieht sich auf* ein Relationssymbol r, wenn es eine Regel in Π mit p im Kopf und r im Körper gibt.

Die *Definition* eines Relationssymbols r ist die Menge aller Regeln aus Π mit r auf der linken Seite.

Ein Relationssymbol r erscheint *positiv* in einem positiven Literal und *negativ* in einem negativen Literal.

Definition (geschichtetes Programm) Ein Programm Π heißt *geschichtet*, wenn es eine Partition

$$\Pi = \Pi_1 \dot\cup \Pi_2 \dot\cup \ \dots \ \dot\cup \Pi_n$$

gibt, die die folgenden beiden Bedingungen für $i = 1, 2, \dots, n$ erfüllt:

1. Kommt ein Relationssymbol in einer Regel in Π_i positiv vor, dann ist seine Definition enthalten in $\bigcup_{j \leq i} \Pi_j$.

2. Kommt ein Relationssymbol in einer Regel in Π_i negativ vor, dann ist seine Definition enthalten in $\bigcup_{j < i} \Pi_j$

Π_1 kann auch leer sein.

In einem geschichteten Logikprogramm kann Rekursion innerhalb einer Schicht nur durch positive Literale stattfinden. Ist ein negatives Literal $L = \neg A$ Teilziel

einer Regel, so muss die Definition des Relationssymbols von A in einer der darunterliegenden Schichten erfolgt sein.

Beispiel Sei Π_1 das folgende Programm:

$p(x) \leftarrow \neg q.$
$r.$
$q \leftarrow q \wedge \neg r.$

Für dieses Programm ist $\Pi_1 = \{r\} \dot\cup \{q \leftarrow q \wedge \neg r\} \dot\cup \{p(x) \leftarrow \neg q\}$ eine gültige Zerlegung in Schichten. Für das Programm Π_2:

$p \leftarrow q.$
$q \leftarrow \neg p.$

hingegen lässt sich keine gültige Zerlegung finden, da die zweite Regel erfordert, dass die Definition von p in einer tieferen Schicht als die Definition von q (Bedingung 2) enthalten ist. Da p sich aber auf q bezieht, muss die Definition von g in einer tieferen oder derselben Schicht wie die Definition von p enthalten sein (Bedingung 1). Das ist offensichtlich nicht möglich, Π_2 ist also kein geschichtetes Logikprogramm.

Apt, Blair und Walker definieren für ein geschichtetes Programm Π einen Operator T dessen kleinster Fixpunkt ein minimales Herbrand-Modell von Π ist. Dabei verwenden sie die vereinfachte Repräsentation eines totalen Modells durch die Menge derjenigen Grundatome aus der Herbrandbasis von Π, die in dem Modell wahr sind.

Definition (Fixpunktoperator T) Sei $\Pi = \Pi_1 \dot\cup \Pi_2 \dot\cup \ldots \dot\cup \Pi_n$ eine Zerlegung von Π in Schichten. Dann ist der Operator T_i für $i = 1, 2, \ldots, n$ definiert wie folgt:

Für jedes Herbrand-Modell M und jedes Grundatom A ist $A \in T_i(M)$ genau dann, wenn $A \in M$ oder für eine Klausel $A' \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_m$ in Π_i und eine Grundsubstitution θ sind $L_1\theta, L_2\theta, \ldots, L_m\theta$ in M wahr und $A = A'\theta$.

Die Mengen $M_0, M_1, \ldots, M_n$ von Grundatomen sind definiert durch:

$$M_0 = \emptyset, M_i = \bigcup_{j \geq 0} T_i^j(M_{i-1}), i = 1, 2, \ldots, n$$

Dabei sei T_i^j die j-fache Anwendung von T_i. Man kann den hier verfolgten Ansatz leicht erkennen: Zunächst bestimmt man das Fixpunkt-Modell der untersten Schicht und dann anhand dieser Informationen ein Fixpunkt-Modell der nächsthöheren Schicht usw., bis schließlich das Modell der obersten Schicht mit dem Modell des gesamten Programms übereinstimmt. Man sieht auch, dass Negation bei dieser Fixpunktbestimmung nicht zu unendlichen Berechnungen führen kann, da mit der leeren Menge begonnen wird und danach sämtliche in einer Schicht negiert vorkommenden Literale bereits vollständig durch das Modell aller darunterliegenden Schichten bestimmt ist.

Satz M_n ist Modell von Π und unabhängig von der gewählten Zerlegung $\Pi = \Pi_1 \dot\cup \Pi_2 \dot\cup \ldots \dot\cup \Pi_n$.

Man sieht also, dass es für geschichtete Logikprogramme genau ein minimales Herbrand-Modell gibt. Die iterative Fixpunkt-Semantik für geschichtete Logikprogramme wählt dieses Modell als kanonisches Modell für ein solches Programm aus.

2.2.2 Perfect Model Semantik

Przymusinksi beschreibt in [4] eine Semantik für *disjunktive Logikprogramme*, das sind Mengen von Regeln der Form

$$A_1 \wedge A_2 \wedge \ldots \wedge A_m \wedge \neg B_1 \wedge \neg B_2 \wedge \ldots \wedge \neg B_n \rightarrow C_1 \vee C_2 \vee \ldots \vee C_p$$

mit $m, n \geq 0$, $p \geq 1$ und A_i, B_j, C_k Atomen. Wie man sofort sieht enthält die Klasse der disjunktiven Logikprogramme auch die Logikprogramme, um die es bisher in dieser Arbeit ging ($p = 1$ für alle Regeln in einem Programm), so dass die hier vorgestellten Ergebnisse ohne weiteres auf diese Logikprogramme übertragbar sind. Für alle Regeln eines sogenanntes positives disjunktives Logikprogramms gilt: $n = 0$.

Beispiel Das disjunktive Logikprogramm Π:

$informatik(X) \wedge \neg praxis(X) \rightarrow theorie(X).$
$informatik(Turing).$
$mathematik(Euler).$

hat zwei minimale Modelle,

$M_1 = \{informatik(Turing), mathematik(Euler), theorie(Turing), \neg praxis(Turing)\}$
und
$M_2 = \{informatik(Turing), mathematik(Euler), \neg theorie(Turing), praxis(Turing)\}.$

Hier stellt sich die Frage, welches dieser beiden Modelle denn nun die Bedeutung des Programms repräsentiert. Betrachtet man die erste Regel, so liest man sie vermutlich intuitiv als „Wenn jemand InformatikerIn ist und nicht mit der Praxis beschäftigt, dann beschäftigt er oder sie sich mit der Theorie.“ Da wir über Turing wissen, dass er Informatiker ist und das Programm keinen Anhaltspunkt dafür liefert, dass er sich mit der Praxis beschäftigt, scheint es naheliegend $praxis(Turing)$ als falsch anzunehmen und davon auszugehen, dass die Bedeutung des Programms durch das Modell M_1 repräsentiert wird.

Sei nun Π' das Logikprogramm, dass man aus Π dadurch erhält, dass man die erste Regel durch

$$informatiker(X) \rightarrow praxis(X) \vee theorie(X)$$

ersetzt. Π' scheint nun eine von Π verschiedene Bedeutung zu haben, denn es scheint die Prädikatssymbole $praxis(X)$ und $theorie(X)$ gleichwertig zu behandeln, so dass die Modelle M_1 und M_2 gleich plausibel erscheinen.

Dieses kleine Beispiel legt den Gedanken nahe, Prädikatssymbolen unterschiedliche Prioritäten hinsichtlich der Bestimmung eines minimalen Modells zuzuordnen, abhängig davon ob sie (negiert oder auch nicht) im Körper eine Regel stehen oder im Kopf:

- Negative Voraussetzungen $\neg A$ sollten eine *höhere* Priorität haben als Konsequenzen, also sollten sie eher minimiert (das Atom A also als falsch angenommen werden) werden, als der Kopf einer Regel.
- Positive Voraussetzungen sollten *dieselbe oder eine höhere* Priorität haben als Konsequenzen,
- Alle Atome in einer Konsequenz eine bestimmten Klausel sollten *dieselbe* Priorität haben.

Daraus ergibt sich die folgende Definition für zwei Prioritätenrelationen zwischen den Atomen eines Logikprogramms.

Definition (Prioritäten-Relationen $<$ und $\leq$) Die *Prioritäten-Relationen* $<$ *und* $\leq$ zwischen Atomen aus der Herbrandbasis eines Programms Π seien definiert wie folgt:

- $C < B$, falls B eine negative Voraussetzung und C eine Konsequenz einer Grundinstanz einer Klausel aus Π ist.
- $C \leq A$, falls A eine positive Voraussetzung und C eine Konsequenz einer Grundinstanz einer Klausel aus Π ist.
- $C \leq C'$, falls C und C' beide Konsequenzen derselben Grundinstanz einer Klausel aus Π sind.
- Falls $A \leq B$ und $B \leq C$, dann gilt: $A \leq C$.
- Falls $A \leq B$ und $B < C$ (bzw. $D < A$), dann gilt: $A < C$ (bzw. $D < B$).
- Falls $A < B$, dann gilt $A \leq B$.
- Die Herbrand-Basis von Π ist abgeschlossen unter $<$ und $\leq$.

Die Idee nun auf Grundlage dieser Relationen ein „perfektes Modell“ zu bestimmen sieht so aus, dass man bei der Berechnung eines Modells Atome höherer Priorität zuerst minimiert (also im Modell als falsch annimmt), auch wenn das bedeutet, dass dafür möglicherweise sogar mehrere Atome niedrigerer Priorität im Modell als wahr angenommen werden müssen. Aus dieser Idee ergibt sich nun unmittelbar die Definiton eines perfekten Modells:

Definition (Perfektes Modell) Für zwei verschiedene Modelle M und N eines Programms Π sagen wir N ist M *vorzuziehen* (Schreibweise: $N << M$), falls für jedes Grundatom $A \in N \backslash M$ ein weiteres Grundatom $B \in M \backslash N$ existiert mit $A < B$. Ein Modell M von Π heißt *perfekt*, falls Π kein Modell hat, das M vorzuziehen wäre.

Satz Falls $N \subset M$, dann ist $N << M$. Insbesondere bedeutet das: ein perfektes Modell ist minimal.

Satz Falls Π ein positives disjunktives Logikprogramm ist, gilt: $M << N$ genau dann, wenn $M \subset N$ und insbesondere: Ein Modell ist perfekt genau dann, wenn es minimal ist. Die Perfect Model Semantik ist also mit der Standard-Semantik für definite Logikprogramme verträglich.

Satz Um zu zeigen, dass ein Modell M eines disjunktiven Logikprogramms Π perfekt ist, genügt es zu zeigen, dasss es kein minimales Modell von Π gibt, dass M vorzuziehen wäre.

Beispiel Das Programm

$p(0)$.
$p(x) \leftarrow (p(s(x))$.

hat unendliche viele minimale Herbrand-Modelle und für ein Modell M existiert immer ein Modell N, so dass gilt: $M << N$ und $N << M$. Dieses Programm hat also kein perfektes Modell.

Betrachtet man das Eingangsbeispiel

$informatik(X) \wedge \neg praxis(X) \rightarrow theorie(X).$
$informatik(Turing).$
$mathematik(Euler).$

so stellt man fest, dass das „intuitiv richtige“ Modell

$M_1 = \{informatik(Turing), mathematik(Euler), theorie(Turing), \neg praxis(Turing)\}$

das perfekte Modell dieses Programms ist.

2.2.3 Wohlfundierte Semantik

Van Gelder, Ross und Schlipf beschreiben in [5] eine weitere Semantik für Logikprogramme mit Negation, die ebenfalls ein kanonisches Modell für ein solches Programm auswählt. Dieses Modell kann allerdings durchaus nur ein partielles Modell sein, d.h., dass nicht allen Atomen ein Wahrheitswert zugewiesen wird, sondern dass man auch unvollständige Informationen in Modellen zulässt.

Ein wichtiges Hilfskonstrukt für die Definition der wohlfundierten Modelle sind die sogenannten nicht-fundierten Mengen, das sind Mengen von Literalen, deren Wahrheitswert aus einem Logikprogramm nicht folgerbar ist.

Definition (nicht-fundierte Mengen) Sei Π ein Logikprogramm und I eine partielle Interpretation. $A \subseteq \mathfrak{H}_\Pi$ heißt *nicht-fundierte Menge* (von Π) in Bezug auf I, falls für jedes Atom $p \in A$ die folgende Bedingung erfüllt ist:
Jede instanziierte Regel R von Π deren Kopf p ist, hat (mindestens) eine der beiden folgenden Eigenschaften:

1. Ein (positives oder negatives) Teilziel Q im Körper wird von I falsifiziert,
2. Ein positives Teilziel im Körper ist in A enthalten.

Ein Literal, das eine dieser der beiden Eigenschaften erfüllt, nennt man auch einen *Zeugen für die Unbrauchbarkeit* der Regel R (in Bezug auf I).

Was also ist die Bedeutung dieser nicht-fundierten Mengen? Angenommen I repräsentiert unseren (möglicherweise unvollständigen) Kenntnisstand über die Atome eines Logikprogramms Π. Regeln, die die erste Bedingung erfüllen erlauben es nicht, neue Erkenntnisse über das Atom im Regelkopf zu erlangen, da mindestens eine Voraussetzung bereits als falsch bekannt ist. Regeln, die die zweite Bedingung (die sogenannte *unfoundedness condition*) erfüllen, sind ebenfalls nicht brauchbar, da sie voraussetzen, dass der Wahrheitswert eines Atoms aus A bekannt ist. Es gibt aber kein Atom in A, das alleine mit Regeln aus Π als erstes bewiesen werden kann (mit Wissen I).

Beispiel Sei Π das folgende Programm:

$p(a) \leftarrow p(c) \wedge \neg p(b).$
$p(b) \leftarrow \neg p(a).$
$p(e) \leftarrow \neg p(d).$
$p(c).$
$p(d) \leftarrow q(a) \wedge \neg q(b).$
$p(d) \leftarrow q(b) \wedge \neg q(c).$
$q(a) \leftarrow p(d).$
$q(b) \leftarrow q(a).$

In Bezug auf die leere Interpretation bildet $M_1 = \{p(d), q(a), q(b), q(c)\}$ eine nicht-fundierte Menge. Man sieht sofort, dass $q(c)$ zu dieser Menge gehören muss, da es keine Regel in Π gibt, deren Kopf $q(c)$ ist, kann für $q(c)$ kein Wahrheitswert bestimmt werden. Die Menge $\{p(d), q(a), q(b)\}$ ist nicht-fundiert wegen Bedingung 2: Es gibt kein Atom in dieser Menge, das zuerst bewiesen werden könnte, ohne eines der anderen Atome aus dieser Menge vorher zu beweisen. Im Gegensatz dazu ist $M_2 = \{p(a), p(b)\}$ allerdings keine keine nicht-fundierte Menge, da diese beiden Atome ausschließlich über Negation voneinander abhängen. Hier spielt die Tatsache eine Rolle, dass, falls ein Atom in M_1 als falsch deklariert, sich trotzdem keines der anderen Atome aus M_1 als wahr ableiten lässt, wohingegen die Annahme eines der Atome aus M_2 als falsch direkt die Beweisbarkeit des anderen Atoms mit sich bringt.

Definition (größte nicht-fundierte Menge) Die *größte nicht-fundierte Menge* eines Programms Π in Bezug auf I, bezeichnet mit $U_\Pi(I)$ ist die Vereinigung aller nicht-fundierten Mengen in Bezug auf I.

Seien nun die Transformationen T_Π, U_Π, W_Π definiert für ein Logikprogramm Π und eine Interpretation I wie folgt:

- $p \in T_\Pi(I)$ genau dann, wenn es eine Grundinstanz einer Regel $R \in \Pi$ gibt mit Kopf p und jedes Literal auf der rechten Regelseite gilt in I.
- $U_\Pi(I)$ entspricht der größten nicht-fundierten Menge (s.o.).
- $W_\Pi(I) = T_\Pi(I) \cup \{\neg A | A \in U_\Pi(I)\}$.

Auf der Grundlage dieser Transformationen seien nun die Interpretationen I^∞ und I_α wie folgt definiert:

- $I_0 := \emptyset$
- $I_\alpha := \bigcup_{\beta<\alpha} I_\beta$
- $I_{\gamma+1} := W_\Pi(I_\gamma)$
- $I^\infty := \bigcup_{\alpha\in\mathbb{N}} I_\alpha$

I^∞ ist der kleinste Fixpunkt der Transformation W_Π und repräsentiert das wohlfundierte Modell eines Programms Π.

Definition (wohlfundiertes Modell) Der kleinste Fixpunkt I^∞ der Transformation W_Π für ein Logikprogramm Π ist ein Modell von Π. Falls I^∞ eine totale Interpretation ist (also für jedes Atom der Herbrandbasis von Π einen Wahrheitswert festlegt), so nennt man I^∞ *wohlfundiertes Modell* von Π, anderenfalls nennt man I^∞ *wohlfundiertes partielles Modell*.

Definition (wohlfundierte Semantik) Die *wohlfundierte Semantik* eines Programms Π ist die Bedeutung, die durch das wohlfundierte (partielle) Modell I^∞ von Π repräsentiert wird. Jedes Atom $A \in I^\infty$ ist wahr, jedes Atom A mit $\neg A \in I^\infty$ ist falsch und jedes Atom, das nicht (negiert) in I^∞ vorkommt ist unbekannt.

Das wohlfundierte Modell eines definiten Logikprogramms und einer Anfrage stimmt mit seinem einzigen minimalen Herbrand-Modell überein. Das wohlfundierte Modell eines geschichteten Logikprogramms stimmt mit seinem iterativen Fixpunkt-Modell überein.

3 Stable Model Semantik

Hier soll nun der Ansatz der Stable Model Semantik beschrieben werden, den Gelfond und Lifschitz 1988 in [3] erstmals vorgestellt haben. Die Stable Model Semantik ist eine weitere deklarative Semantik, die versucht, einem Logikprogramm ein kanonisches Modell als Bedeutung zuzuordnen.

3.1 Definition der Stable Model Semantik

Definition (Gelfond-Lifschitz Transformatierte) Sei Π ein Logikprogramm und I eine Interpretation. Die *Gelfond-Lifschitz-Transformierte* von Π bezüglich I (Bezeichnung: Π_I) ist das Programm, das man aus der (möglicherweise unendlich großen) Herbrand-Instanz $\mathfrak{I}_\Pi$ von Π erhält, wenn man

- jede Klausel mit einem negativen Literal $\neg B$ im Körper für ein $B \in I$ und
- alle negativen Literale $\neg B \in I$ aus den Körpern der verbleibenden Klauseln

entfernt. Falls I eine totale Interpretation ist, ist Π_I nun offensichtlich frei von negativen Literalen uns hat somit genau ein minimales Herbrand-Modell, bezeichnet mit $\mathcal{M}(\Pi_I)$. Wenn dieses Modell mit I übereinstimmt, dann wird I auch *stabile Menge* von Π genannt.

Definition (Fixpunkt-Operator S_Π) Die stabilen Mengen von Π sind die Fixpunkte des *Operators* S_Π, definiert wie folgt: Für eine totale Interpretation M ist $S_\Pi(M)$ das kleinste Herbrand-Modell von Π_M, also:

$$S_\Pi(M) = \mathcal{M}(\Pi_M)$$

Beispiel Seien in diesem Beispiel alle Interpretationen total und mit der Menge der in ihnen vorkommenden positiven Literale identifiziert. Das Logikprogramm Π:

$p(1,2).$
$q(x) \leftarrow p(x,y) \wedge \neg q(y).$

hat die Grundinstatz $\mathfrak{I}_\Pi$:

$p(1,2).$
$q(1) \leftarrow p(1,1) \wedge \neg q(1).$
$q(1) \leftarrow p(1,2) \wedge \neg q(2).$
$q(2) \leftarrow p(2,1) \wedge \neg q(1).$
$q(2) \leftarrow p(2,2) \wedge \neg q(2).$

Für die Interpretation $I = \{q(2)\}$ ergibt sich Π_I zu:

$p(1,2).$
$q(1) \leftarrow p(1,1).$
$q(2) \leftarrow p(2,1).$

Dieses Programm hat das minimale Herbrand-Modell $\mathcal{M}(\Pi_I) = \{p(1,2)\}$, das nicht mit I übereinstimmt. I ist also keine stabile Menge von Π, was aber eigentlich auch vorauszusehen war, denn I ist kein Modell von Π. Wählt man hingegen $I' = \{p(1,2), q(1)\}$, so ergibt sich $\Pi_{I'}$ zu:

$p(1,2)$.
$q(1) \leftarrow p(1,2)$.
$q(2) \leftarrow p(2,2)$.

mit dem kleinsten Herbrand-Modell $\mathcal{M}(\Pi_{I'}) = \{p(1,2), q(1)\}$, das mit I' übereinstimmt. I' ist also eine stabile Menge von Π.

Satz Jede stabile Menge von Π ist ein minimales Herbrand-Modell von Π. Aus diesem Grund nennt man eine stabile Menge auch *stabiles Modell*.

Beweis Sei M eine stabile Menge von Π und somit minimales Herbrand-Modell von Π_M.

Teil 1: M ist Modell von Π Sei $R = A \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_n$ eine Regel aus Π. Gilt $L_i = \neg B$ mit $B \in M$ für ein Literal L_i im Körper von R, dann ist die Implikation

$$A \leftarrow L_1 \wedge \ldots \wedge \underbrace{L_i}_{falsch} \wedge \ldots \wedge L_n$$

trivialerweise erfüllt und R gilt somit in M. Gibt es kein solches Literal in R, dann betrachten wir die Regel R', die man aus R erhält, indem man alle in M gültigen negativen Literale $\neg B \in M$ aus dem Körper von R entfernt. R' gilt in M (da M minimales Herbrand-Modell von Π_M ist und $R' \in \Pi_M$) und R unterscheidet sich von R' nur durch zusätzliche in M gültige Literale im Körper. Somit gilt auch R in M.

Teil 2: M ist *minimales* Modell von Π Sei $M' \preceq M$ ein totales Modell von Π. Sei weiter R' eine Regel von Π_M, die aus einer Regel $R \in \Pi$ dadurch entstanden ist, dass alle negativen Literale $\neg B \in M$ aus ihrem Körper gelöscht wurden. R ist wahr in M', da M' Modell von Π ist, jedes negative Literal $\neg B$ im Körper von R ist wahr in M', da $\neg B \in M$ und $M' \preceq M$ (insbesondere: $Neg(M) \subseteq Neg(M')$ und R' per Resolution aus R und diesen Literalen ableitbar ist. Also ist M' ein totales Modell von Π_M. Da laut der ursprünglichen Annahme $M' \preceq M$ und M minimales Modell von Π_M ist, gilt: $M' = M$.

□

Definition (Stable Model Semantik) Die *Stable Model Semantik* ist für ein Logikprogramm Π definiert genau dann, wenn Π genau ein stabiles Modell hat. Die Semantik wählt dann dieses Modell als kanonisches Modell von Π.

Es gibt zwei Arten von Programmen, auf die die Stable Model Semantik nicht anwendbar ist, und zwar die Klasse derjenigen Programme, die kein stabiles Modell besitzen und die Klasse derjenigen Programme, die mehrere stabile Modelle besitzen.

Beispiel Das Programm Π_1:

$p \leftarrow \neg p$.

besitzt offensichtlich kein stabiles Modell. Das Programm Π_2:

$p \leftarrow \neg q$.
$q \leftarrow \neg p$.

hingegen besitzt zwei stabile Modelle, $\{p, \neg q\}$ und $\{\neg p, q\}$. Es scheint schwierig für dieses letzte Programm ein einziges kanonisches Modell festzulegen, bzw. zu begründen, welches der beiden minimalen Herbrand-Modelle man als einziges kanonisches Modell auswählt. Dies zeigt, dass es offensichtlich Logikprogramme gibt, deren Semantik schwer zu erfassen, zumindest aber nicht eindeutig ist.

3.2 Vergleich mit anderen deklarativen Ansätzen

Nun stellt sich die Frage, ob die Stable Model Semantik auch mit anderen Ansätzen, kanonische Modelle für Logikprogramm mit Negation auszuwählen, verträglich ist, also insbesondere, ob andere Ansätze ebenfalls das stabile Modell eines Programms als kanonisches Modell wählen. Gelfond und Lifschitz stellen nach der Definition ihrer Semantik in [3] die wichtigsten Ergebnisse in dieser Hinsicht vor.

Eine wichtige Erkenntnis ist es, dass, falls ein Logikprogramm ein wohl-fundiertes Modell besitzt, dieses mit seinem einzigen stabilen Modell übereinstimmt. Das ist insofern von Bedeutung, als dass im allgemeinen Fall die Berechnung (oder der Fehlschlag dieser) eines stabilen Modells eines Logikprogramms NP-vollständig ist, die Berechnung des wohl-fundierten Modells eines Logikprogramms aber mit polynomiellem Aufwand möglich ist. So kann man zunächst die Existenz eines wohlfundierten Modells überprüfen um anschließend nur nach einem Fehlschlag mit einem aufwändigerem Algorithmus nach stabilen Modellen zu suchen, wie es u.a. auch Chen und Warren in [2] vorschlagen.

Für die Klasse der *lokal geschichteten* Logikprogramme, einer Oberklasse der geschichteten Logikprogramme stimmt die wohlfundierte Semantik mit der Perfect Model Semantik überein, so dass gilt:

Falls ein Logikprogramm lokal geschichtet ist, besitzt es genau ein stabiles Modell, dass mit seinem perfekten Modell übereinstimmt.

Die Einschränkung der Perfect Model Semantik auf geschichtete Logikprogramme ist wiederum äquivalent zur iterativen Fixpunktsemantik für diese Programmklasse, so dass gilt:

Ist ein Logikprogramm geschichtet, so stimmt sein iteratives Fixpunktmodell mit seinem einzigen stabilen Modell überein.

Man sieht also, das für die Klassen der (lokal) geschichteten Logikprogramme Einigkeit darüber zu bestehen scheint, welche Bedeutung den Programmen in dieser Klasse zuzuordnen ist. Für Programme außerhalb dieser Klassen ist das nicht ganz so einfach, so hat beispielsweise das Programm

$p \leftarrow \neg p.$

ein perfektes Modell $\{p\}$ aber kein stabiles Modell, umgekehrt besitzt das Programm

$p(1, 2),$
$q(x) \leftarrow p(x, y), \neg q(y).$

zwar genau ein stabiles Modell $\{p(1, 2), q(1)\}$, aber kein perfektes Modell.

Es scheint allerdings Einigkeit darüber zu herrschen, dass es für beliebige Logikprogramme im Allgemeinen keine geeignete Methode gibt ein kanonisches Modell auszuwählen.

3.3 Stable Model Semantik und autoepistemische Logik

Autoepistemische Logik einer der bedeutendsten Ansätze, unser natürliches Verständnis von Wissen und darauf basierendem Schlussfolgern zu beschreiben. Sie ist eine Erweiterung der Aussagenlogik, die „Wissenszustände“ eines Agenten mit Hilfe des Modaloperators L („der Agent glaubt“) beschreibt.

In diesem Abschnitt sei zwecks einer einfacheren Darstellung der Sachverhalte ein Modell eines Logikprogramms durch die Menge der Grundatome repräsentiert, die in dem Modell wahr sind.

Definition (Autoepistemische Formel) Sei AL die Menge der aussagenlogischen Formeln, die in der autoepistemischen Logik auch *objektive Formeln* genannt werden. Dann ist die Menge AEL der Autoepistemischen Formeln Induktiv definiert durch:

- $\phi \in AL \curvearrowright \phi \in AEL$
- $\phi \in AEL \curvearrowright L\phi \in AEL$
- $\phi, \psi \in AEL \curvearrowright \neg\phi, \phi \wedge \psi, \phi \vee \psi, \phi \rightarrow \psi \in AEL$

Definition (Autoepistemisches Atom) In einer autoepistemischen Formel ϕ bezeichnet man diejenigen aussagenlogischen Atome (Variablen) und diejenigen Teilformeln $L\psi$ als *autoepistemische Atome*, die an der „Oberfläche“ von ϕ auftreten, also nicht innerhalb einer Teilformel der Form $L\vartheta$.

Definition (Autoepistemische Interpretation) Eine *autoepistemische Interpretation* ist eine Belegung von autoepistemischen Atomen in einer Formel ψ mit Wahrheitswerten und bestimmt so über die übliche Bedeutung der aussagenlogischen Junktoren einen Wahrheitswert für ψ. Dabei spielen die Wahrheitswerte von Atomen innerhalb einer Teilformel der Form $L\phi$ keine Rolle, da $L\phi$ selbst als Atom gilt und direkt einen Wahrheitswert zugewiesen bekommt.

Definition (Autoepistemische Theorie) Eine Menge $T \subseteq AEL$ wird *autoepistemische Theorie* genannt.

Ein zentrales Thema in der autoepistemischen Logik sind sogenannte stabile Expansionen einer Theorie T die für gewöhnlich als „eine Menge von Dingen an die ein rational denkender Agent auf der Grundlage von T glauben könnte“ beschrieben werden.

Definiton (stabile Expansion) Formal sind stabile Expansionen wie folgt definiert: Sei A eine Theorie. Dann heißt eine Menge E *stabile Expansion* von A, falls

$$E = cons(A \cup \{LF | F \in E\} \cup \{\neg LF : F \notin E\})$$

Dabei sei $cons(X)$ die Menge aller aussagenlogischen Konsequenzen einer Formelmenge X. Falls alle Formeln einer Theorie A objektiv sind, dann hat A genau eine stabile Expansion E und eine objektive Formel ist in E enthalten genau dann, wenn sie eine aussagenlogische Folgerung aus A ist.

Die intuitive Bedeutung von stabilen Modellen eines Logikprogramms ist nun ganz ähnlich der intuitiven Bedeutung dieser stabilen Expansionen. Falls M die Menge der Grundatome ist, die ein Agent als wahr erachtet, dann kann er jede Regel mit einem Teilziel $\neg B$ mit $B \in M$ löschen, da er aus ihr aufgrund einer bereits als falsch bekannten Prämisse keinen Schluss mehr ziehen kann. Andererseits ist

jedes Teilziel $\neg B$ mit $B \notin M$ vom Standpunkt des Agenten aus trivial, so dass er Π vereinfachen und durch Π_M ersetzen kann. Falls M genau die Menge der Atome ist, die logisch aus seinen vereinfachten Voraussetzungen Π_M folgen, dann ist der Agent rational.

Für jedes variablenfreie Logikprogramm Π sei $I(\Pi)$ die Menge der autoepistemischen Formeln, die man aus Π enthält, indem man jedes negierte Atom $\neg A$ durch $\neg LA$ ersetzt. Mit At sei die Menge der Atome aus Π bezeichnet.

Satz Falls ein Logikprogramm Π genau ein stabiles Modell M hat, dann hat $I(\Pi)$ genau eine stabile Expansion E und $M = E \cap At$.

Um diesen Satz zu beweisen, wird als Hilfsmittel folgendes Lemma benötigt:

Lemma E ist eine stabile Expansion von $I(\Pi)$ genau dann, wenn E stabile Expansion von $\Pi_{E \cap At}$ ist.

Beweis (des Lemmas) Es genügt zu zeigen, dass

$$I(\Pi) \cup \{LF | F \in E\} \cup \{\neg LF | F \notin E\}$$

äquivalent zu

$$\Pi_{E \cap At} \cup \{LF | F \in E\} \cup \{\neg LF | F \notin E\}$$

ist: Die Menge $\{LF | F \in E\} \cup \{\neg LF | F \notin E\}$ enthält LF für jedes $F \in E \cap At$ und $\neg LF$ für jedes Atom $F \notin E \cap At$. Dann ist $I(\Pi)$ äquivalent zu $\Pi_{E \cap At}$.

□

Beweis (des Satzes) Sei M das einzige stabile Modell von Π. Da Π_M eine Menge von objektiven Formeln ist, hat Π_M genau eine stabile Expansion E und $E \cap At = cons(\Pi_M) \cap At = S_\Pi(M) = M$ Also ist E eine stabile Expansion von $\Pi_{E \cap At}$. Aus dem Lemma folgt, dass E eine stabile Expansion von $I(\Pi)$ ist, bleibt also noch zu zeigen, dass $I(\Pi)$ keine weiteren stabilen Expansionen hat:

Sei E' eine stabile Expansion von $I(\Pi)$. Dann folgt aus dem Lemma, dass E' eine stabile Expansion von $\Pi_{E' \cap At}$ ist. Da letztere eine Menge von objektiven Formeln ist, gehört eine objektive Formel zu E' genau dann, wenn sie eine aussagenlogische Konsequenz aus $\Pi_{E' \cap At}$ ist. Also gilt

$$E' \cap At = cons(\Pi_{E' \cap At}) \cap At = S_\Pi(E' \cap At)$$

so dass $E' \cap At$ ein stabiles Modell von Π ist. Da M das einzige stabile Modell von Π ist, folgt: $E' \cap At = M$, daher ist $\Pi_{E' \cap At} = \Pi_M$ und E' ist eine stabile Expansion von Π_M. Also gilt: $E = E'$.

□

3.4 Berechnung von stabilen Modellen

Man kann herausfinden, ob und wie viele stabile Modelle Π besitzt, indem man jede totale Interpretation I für Π aufzählt und dann überprüft, ob I mit dem minimalen Herbrand-Modell von Π_I übereinstimmt. Leider wächst die Anzahl der möglichen totalen Interpretationen exponentiell mit der Größe der Herbrand-Basis, so dass diese Lösung zunächst einmal ungeeignet scheint. Allein die Entscheidung, ob Π überhaupt ein stabiles Modell hat, ist NP-vollständig. Allerdings gibt es einige

Ansätze, die Anzahl der nötigen Rechenschritte so weit zu reduzieren, dass trotz des exponentiellen Aufwands für einige praktisch relevante Logikprogramme eine noch akzeptable Berechnungsdauer erreicht wird.

Hier soll nun der Ansatz, den Chen und Warren in [2] verfolgen, vorgestellt werden. Dabei werden sich diese Ausführungen auf endliche variablenfreie Logikprogramme beschränken.

3.4.1 Verkleinerung des Suchraums

Beobachtung 1 Die Gelfond-Lifschitz-Transformation eines Programms Π in Bezug auf eine Interpretation I operiert ausschließlich auf Regeln von Π, in denen negative Literale auftreten. Um einen geeigneten Kandidaten für ein stabiles Modell eines Programms Π zu finden, genügt es also Interpretationen zu betrachten, die nur den in Π negiert vorkommenden Grundatomen Wahrheitswerte zuweisen.

Beispiel Die Gelfond-Lifschitz-Transformierte von Π:

$p(1,2).$
$q(1) \leftarrow p(1,1) \wedge \neg q(1).$
$q(1) \leftarrow p(1,2) \wedge \neg q(2).$
$q(2) \leftarrow p(2,1) \wedge \neg q(1).$
$q(2) \leftarrow p(2,2) \wedge \neg q(2).$

und damit auch deren minimales Herbrand-Modell ist durch die Wahrheitswerte von $q(1)$ und $q(2)$ eindeutig bestimmt. Bei der Suche nach stabilen Modellen von Π reicht es also, Vermutungen ausschließlich über die Wahrheitswerte von $q(1)$ und $q(2)$ anzustellen.

Definition (Die Menge $\mathcal{N}(\Pi)$) $\mathcal{N}(\Pi) := \{A | \neg A \in R \text{ für ein } R \in \Pi\}$ ist die Menge aller Grundatome, die negiert in Π vorkommen.

Lemma 1 Sei Π ein variablenfreies Logikprogramm und I eine totale Interpretation. Dann ist I ein stabiles Modell von Π genau dann, wenn für eine Interpretation J mit $Pos(J) \cup Neg(J) = \mathcal{N}(\Pi)$, $J \subseteq I$ und I mit dem minimalen Herbrand-Modell von Π_J übereinstimmt.

Beweis Sei I eine totale Interpretation und J die Einschränkung von I auf $\mathcal{N}(\Pi)$, also $Pos(J) = Pos(I) \cap \mathcal{N}(\Pi)$ und $Neg(J) = Neg(I) \cap \mathcal{N}(\Pi)$. Aus der Definition der Gelfond-Lifschitz-Transformierten von Π bezüglich einer Interpretation folgt nun unmittelbar $\Pi_I = \Pi_J$, da I und J allen in Π vorkommenden negativen Literalen denselben Wahrheitswert zuweisen und die Transformation ausschließlich von den Wahrheitswerten dieser Literale abhängt. Dann folgt aus der Definition der stabilen Modelle, dass I stabiles Modell von Π ist genau dann, wenn I mit dem minimalen Herbrand-Modell von Π_J übereinstimmt.

□

Beobachtung 2 Hat man in einer Interpretation einen Wahrheitswert für ein in Π negiert vorkommendes Atom festgelegt, so kann (und sollte) man Π basierend auf dieser Annahme weiter vereinfachen und so den Suchraum für Wertzuweisungen an Grundatome in $\mathcal{N}(\Pi)$ bei der Suche nach stabilen Modellen weiter verkleinern.

Sei also Π ein variablenfreies Programm und $\neg A$ ein Literal in einer Regel von Π. Dann gibt es zwei Möglichkeiten, entweder $\neg A$ ist wahr, oder A ist wahr. Ausgehend

von diesen beiden Möglichkeiten lassen sich aus Π zwei vereinfachte Programme Π_A (A ist wahr) und $\Pi_{\neg A}$ ($\neg A$ ist wahr) ableiten. Ziel dieser Vereinfachung ist es, aus den stabilen Modellen von Π_A und $\Pi_{\neg A}$ auf stabile Modelle von Π zu schließen. Dabei muss man allerdings darauf achten, das Programm so zu vereinfachen, dass keine Modelle generiert werden können, die zwar durch die Annahme gestützt werden, aber nicht stabil sind.

Definition (Programm Π_L) Sei Π ein variablenfreies Programm und L ein Grundliteral. Dann ist Π_L das Programm, das man aus Π erhält, wenn man

- jede Regel $R \in \Pi$ mit $\overline{L} \in R$ und
- jedes Vorkommen von L in Π, falls L ein negatives Literal ist,

löscht.

Lemma 2 Sei Π ein variablenfreies Programm, A ein Grundatom und I eine totale Interpretation. Dann ist I ein stabiles Modell von Π genau dann, wenn entweder A in I gilt und I stabiles Modell von Π_A ist, oder $\neg A$ in I gilt und I stabiles Modell von $\Pi_{\neg A}$ ist.

Beweis Wenn A in I gilt dann ist $\Pi_I = (\Pi_A)_I$, da die Transformation von Π zu Π_A sich für $A \in I$ analog zur Gelfond-Lifschitz-Transformation von Π bezüglich I verhält.Also ist I genau dann stabiles Modell für Π, wenn I stabiles Modell für Π_A ist.

Im anderen Fall gilt $\neg A$ in I. Vergleicht man die Vereinfachung von Π zu $\Pi_{\neg A}$ mit der Gelfond-Lifschitz-Transformation, so stellt man fest, dass auch bei der Vereinfachung zu $\Pi_{\neg A}$ alle Regeln mit positivem Literal A im Körper gelöscht werden. Also stimmt Π_I mit $(\Pi_{\neg A})_I$ überein, mit Ausnahme einiger zusätzlichen Regeln in Π_I mit einem positiven Literal A im Körper. Nach Definition ist I stabiles Modell von Π genau dann, wenn I mit dem kleinsten Herbrand-Modell $\mathcal{M}(\Pi_I)$ übereinstimmt. Da $\neg A$ in I gilt, muss A in $\mathcal{M}(\Pi_I)$ falsch sein. Somit gilt $\mathcal{M}(\Pi_I) = \mathcal{M}((\Pi_{\neg A})_I)$ und I ist stabiles Modell von Π genau dann, wenn I stabiles Modell von $\Pi_{\neg A}$ ist.

□

Beobachtung 3 Die Vereinfachung von Π zu Π_L für ein Grundliteral L kann die Wahrheitswerte weiterer Grundatome festlegen. Diese sollten natürlich so weit wie möglich propagiert werden, um das Programm, für das stabile Modelle gesucht werden, weiter zu vereinfachen. Dadurch muss man weniger oft Wahrheitswerte für Grundatome auswählen, die möglicherweise zu inkonsistenten Ergebnissen führen und dadurch Backtracking erfordern könnten.

Beispiel Für das Programm Π betrachten wir die Vereinfachung $\Pi_{\neg u}$:

Programm Π:	Programm $\Pi_{\neg u}$:
$p \leftarrow \neg q \wedge \neg r.$	$p \leftarrow \neg q \wedge \neg r.$
$p \leftarrow \neg u.$	$p.$
$q \leftarrow \neg s.$	$q \leftarrow \neg s.$
$q \leftarrow \neg u.$	$q.$
$s \leftarrow \neg q.$	$s \leftarrow \neg q.$
$r \leftarrow \neg t.$	$r \leftarrow \neg t.$
$t \leftarrow \neg r.$	$t \leftarrow \neg r.$
$u \leftarrow \neg v.$	$u \leftarrow \neg v.$
$v \leftarrow \neg u.$	$v.$

Nun kann man anhand dieses vereinfachten Programms die Wahrheitswerte weiterer Atome bestimmen, so dass man die partielle Interpretation $\{p, q, v, \neg s, \neg u\}$ und das viel einfachere Programm Π':

$r \leftarrow \neg t.$
$t \leftarrow \neg r.$

bekommt. Diese Art, ein Programm unter der Annahme eines Wahrheitswertes für ein Grundliteral zu vereinfachen, wird nun im Folgenden formalisiert:

Definition (Relation $\xrightarrow{I}$) Sei Π ein variablenfreies Programm und U eine Menge, die alle in Π auftauchenden Grundatome enthält. Dann sei die Relation $\xrightarrow{I}$ definiert wie folgt:

$$(\Pi, U) \xrightarrow{I} (\Pi', U')$$

genau dann, wenn

- I eine nicht-leere Interpretation ist, für die
  - $Pos(I)$ die Menge aller Grundatome A ist, für die $A \in \Pi$ ein Fakt ist und
  - $Neg(I) = \{A \in U | \nexists R \in \Pi : R = A \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_n\}$ die Menge der Grundatome ist, die nicht im Kopf einer Regel aus Π auftreten.
- $U' = U - (Pos(I) \cup Neg(I))$
- Π' entsteht aus Π durch Löschen
  - jeder Regel mit einem Literal L im Körper, das nicht in I gilt,
  - jeden Literals L in den Körpern der verbleibenden Regeln, falls L in I gilt und letztendlich
  - jeder Regel mit einem Kopf $A \in Pos(I)$.

(Π, U) heißt *reduziert*, falls es keine Interpretation I mit $(\Pi, U) \xrightarrow{I} (\Pi', U')$ für beliebige Π' und U' gibt.

Wenn (Π, U) nicht reduziert ist, dann existiert genau eine nicht leere Interpretation I und genau ein Paar (Π', U'), so dass $(\Pi, U) \xrightarrow{I} (\Pi', U')$.
Ist (Π, U) reduziert, so entspricht U der Menge aller Grundatome aus Π.

Definition (Reduktion) Sei Π ein Variablenfreies Programm und $\mathcal{U}(\Pi)$ die Menge aller Grundatome aus Π. Π heißt *reduziert zu einer Interpretation I und einem Programm* Π' genau dann, wenn für ein $n \geq 0$, $\Pi_0 = \Pi$, $U_0 = \mathcal{U}(\Pi)$ und $\Pi_n = \Pi'$

$$(\Pi_0, U_0) \xrightarrow{I_1} (\Pi_1, U_1) \xrightarrow{I_2} \ldots \xrightarrow{I_n} (\Pi_n, U_n)$$

(Π_n, U_n) reduziert und $I = \bigcup_{1 \leq i \leq n} I_i$.

Lemma 3 Sei Π ein endliches variablenfreies Programm und Π zu einer Interpretation I^* und einem Programm Π^* reduziert. Dann gilt für jedes stabile Modell I von Π: $I = I^* \cup J$ für ein stabiles Modell J von Π^* und umgekehrt für gilt für jedes stabile Modell J von Π^*: $I = I^* \cup J$ für ein stabiles Modell I von Π. Falls Π ein definites Programm ist, dann gilt: $Pos(I^*) = Pos(\mathcal{M}(\Pi))$.

Beweis Angenommen $(\Pi_k, U_k) \xrightarrow{I_{k+1}} (\Pi_{k+1}, U_{k+1})$. Laut Definition von $\xrightarrow{I}$ enthält $Pos(I_{k+1})$ alle jene Grundatome A, die als Fakt in Π_k auftreten, und $Neg(I_{k+1})$ all jene Grundatome $A \in U_k$, die in keiner Regel $R \in \Pi_k$ im Kopf auftreten.

Nun sei I eine Interpretation mit $Pos(I) \cup Neg(I) = U_k$ und $I = I_{k+1} \cup J$ für eine Interpretation J mit $Pos(J) \cup Neg(J) = U_{k+1}$. Dann ist $\mathcal{M}((\Pi_k)_I) = I_{k+1} \cup \mathcal{M}((\Pi_{k+1})_J)$.

Für eine beliebige Interpretation I mit $Pos(I) \cup Neg(I) = U_k$ ist I stabiles Modell von Π_k genau dann, wenn $I = \mathcal{M}((\Pi_k)_I)$, also genau dann, wenn $I = I_{k+1} \cup J$ für ein J mit $Pos(J) \cup Neg(J) = U_{k+1}$ und $J = \mathcal{M}((\Pi_{k+1})_J)$, also J stabiles Modell von Π_{k+1} ist.

□

3.4.2 Assume-and-Reduce Algorithmus

Chen und Warren konstruieren anhand der Beobachtungen 1-3 den *Assume-and-Reduce Algorithmus*, um alle stabilen Modelle eines endlichen und variablenfreien Logikprogramms zu suchen:

```
Eingabe: ein endliches Grundprogramm Π
Ausgabe: ein stabiles Modell oder Fehler
begin
    Sei P reduziert zu einer Interpretation I_0 und einem Programm Π_0;
    DI := I_0;
    Pgm := Π_0;
    AI := ∅;
    N := 𝒩(Pgm);
    while N ≠ ∅ do begin
        Lösche ein beliebiges Element A aus N
        if A ∉ DI and ¬A ∉ DI then begin
            choice(A,L); /* Wähle entweder L := A oder L := ¬A */
            AI := AI ∪ L;
            Sei Pgm_L reduziert zu I* und Π*;
            DI := DI ∪ I*;
            Pgm := Π*;
            if DI ∪ AI inkonsistent then
              Fehler → Backtracking
        end
    end
    if A ∈ AI and A ∉ DI für ein Grundatom A then
      Fehler → Backtracking
    else begin
        Für jedes A, dass in Pgm auftritt, füge ¬A zu DI hinzu;
        Ausgabe: AI ∪ DI als stabiles Modell von Π;
    end
end
```

Im Algorithmus bezeichnet DI die Menge der Grundliterale, die zu einem Zeitpunkt als wahr abgeleitet werden können. AI beschreibt die Menge derjenigen Grundliterale, für die der Algorithmus eine Annahme bezüglich des Wahrheitswertes getroffen hat. Der Algorithmus durchläuft nicht-deterministisch und mit Backtracking einen Suchbaum für stabile Modelle. Jeder Knoten dieses Suchbaums wird durch ein Tripel (AI, DI, Pgm) repräsentiert.

Der Algorithmus muss an zwei Stellen „nicht-deterministische“ Entscheidungen treffen. Die erste ist diejenige, an der der Algorithmus aus der verbleibenden Menge von in dem Programm negiert vorkommenden Atomen eines auswählt. Diesen

Nicht-Determinismus kann man bei einer Implementation des Algorithmus dadurch beseitigen, dass man die Menge (die offensichtlich endlich ist) ordnet, und dann nach Backtracking das jeweils nächste Element aus dieser Ordnung auswählt. Die zweite Stelle, an der Nicht-Determinismus auftritt, ist diejenige, an der ausgewählt wird, ob das gewählte Atom A wahr oder falsch sein soll in der zu betrachtenden Interpretation. Da es nur diese beiden Möglichkeiten gibt, ist der Nicht-Determinismus ganz einfach dadurch aufzulösen, dass man zunächst A als wahr annimmt und nach Backtracking mit der Annahme A sei falsch weitermacht oder umgekehrt.

Satz (Assume-and-Reduce Algorithmus) Sei Π ein endliches und variablenfreies Logikprogramm. Dann ist eine Interpretation I stabiles Modell von Π genau dann, wenn I das Ergebnis eines Laufs des Assume-and-Reduce Algorithmus ist.

Beweis

Termination Der Algorithmus terminiert für endliche variablenfreie Programme, da die Anzahl der Elemente von N bei jedem Schleifendurchlauf abnimmt und für $N = \emptyset$ der Algorithmus anhält.

Vollständigkeit des Suchraums Die Wurzel des Suchbaumes ist $(AI, DI, Pgm) = (\emptyset, I_0, \Pi_0)$, mit Π reduziert zu I_0 und Π_0. Nach Lemma 3 ist I ein stabiles Modell von Π genau dann, wenn $I = I_0 \cup J$ und J ist ein stabiles Modell von Π_0, also $I = AI \cup DI \cup J$ und J ist stabiles Modell von Pgm.

Für einen Knoten $v = (AI, DI, Pgm)$ sei $A \in \mathcal{N}(\Pi)$, so dass $A, \neg A \notin AI \cup DI$. Dann muss $\neg A$ noch in Pgm vorkommen. Nun gibt es zwei Auswahlmöglichkeiten, entweder ist A oder $\neg A$ wahr. Laut Lemma 2. ist eine Interpretation J ein stabiles Modell von Pgm genau dann, wenn entweder A in J gilt und J ein stabiles Modell von Pgm_A ist, oder $\neg A$ in J gilt und J ein stabiles Modell von $Pgm_{\neg A}$ ist. Sei Pgm_A ($Pgm_{\neg A}$) reduziert zu einer Interpretation I^*_A ($I^*_{\neg A}$) und einem Programm Π^*_A ($\Pi^*_{\neg A}$). Dann ist, wieder laut Lemma 3, J stabiles Modell von Pgm_L mit L entweder A oder $\neg A$, genau dann, wenn $J = I^*_L \cup J^*_L$ und J^*_L stabiles Modell von Π^*_L. Dann hat v zwei Kinder, einmal $(AI \cup \{A\}, DI \cup I^*_A, \Pi^*_A)$ und einmal $(AI \cup \{\neg A\}, DI \cup I^*_{\neg A}, \Pi^*_{\neg A})$. Nach den Argumenten oben gilt für eine Interpretation I: $I = AI \cup DI \cup J$ für ein stabiles Model J von Pgm genau dann, wenn für ein Literal L das entweder A oder $\neg A$ sein kann, $L \in J$ und $J = I^*_L \cup J^*_L$ für ein stabiles Modell J^*_L von Π^*_L, genau dann, wenn $I = AI \cup \{L\} \cup DI \cup I^*_L \cup J^*_L$ für ein stabiles Modell J^*_L von Π^*_L.

Für ein Blatt $v = (AI, DI, Pgm)$ im Suchbaum für stabile Modelle ist entweder $AI \cup DI$ inkonsistent oder für jedes $A \in \mathcal{N}(\Pi)$ ist A oder $\neg A$ in $AI \cup DI$. Im letzteren Fall enthält Pgm keine negativen Literale. Desweiteren enthält Pgm keine Fakten, sonst könnte man Pgm weiter reduzieren. Also ist das einzige stabile Modell von Pgm gleichzeitig auch das einzige minimale Modell $\mathcal{M}(Pgm)$, und zwar das Modell, in dem jedes Grundatom in Pgm falsch ist. Wenn $A \in AI$ und $A \notin DI$ für ein Grundatom A, muss $\neg A$ entweder in DI oder $\mathcal{M}(Pgm)$ wahr sein, was der Annahme das $A \in AI$ wahr ist widerspricht.

□

3.4.3 Implementationen

Der Assume-and-Reduce Algorithmus ist offensichtlich nur für endliche variablenfreie Programme zu gebrauchen, der exponentielle Aufwand schränkt seine Nützlichkeit zusätzlich ein. Chen und Warren stellen in [2] eine Integration des Algorithmus

mit einem Verfahren zur Berechnung von wohlfundierten Modellen vor, das eine polynomielle Zeitkomplexität hat und auch auf Logikprogrammen mit Variablen funktioniert. Findet dieses Verfahren ein wohlfundiertes Modell für ein Logikprogramm und eine Anfrage, so wissen wir, dass dieses mit dem einzigen stabilen Modell übereinstimmt. Findet das Verfahren kein solches Modell, so liefert es ein sogenanntes *Rest-Programm*, das unter bestimmten Bedingungen dann mit Hilfe des Assume-and-Reduce Algorithmus weiter ausgewertet werden kann, um alle stabilen Modelle des ursprünglichen Programms und der Anfrage zu finden.

Implementiert ist dieses Verfahrens unter Anderem im *SLG-System* (zu finden unter *http://engr.smu.edu/~wchen/slg.html*), einem Prolog-Programm, das als Meta-Interpreter auf existierenden Prolog-Implementationen läuft und im *XSB-System* (zu finden unter *http://xsb.sourceforge.net*), einer quelloffenen Umgebung für Logikprogrammierung und deduktive Datenbanken.

4 Fazit

Gelfond und Lifschitz haben 1988 mit ihrer Definition der Stable Model Semantik einen bedeutenden Beitrag dazu geleistet, Logikprogrammen mit Negation eine Bedeutung zuzuweisen. Ihre Semantik ist dabei für die wichtige Klasse der (lokal) geschichteten Logikprogramme mit anderen verbreiteten Ansätzen Konform, ist aber darüber hinaus auch auf Logikprogramme anwendbar, die sich nicht in diese Klasse einordnen lässt.

Trotz des exponentiellen Aufwandes der Bestimmung von stabilen Modellen gibt es mittlerweile einige Implementationen unterschiedlicher Verfahren, stabile Modelle mit so wenig Rechenschritten wie möglich zu bestimmen. Neben der bereits erwähnten Implementation in XSB scheinen die Programme *Smodels* und *Lparse* der Technischen Universität Helsinki recht ausgereift zu sein. Das Smodels untersucht variablenfreie Logikprogramme hinsichtlich stabiler Modelle und das Programm Lparse generiert aus Logikprogrammen mit Variablen variablenfreie Programme, die dann mit Smodels untersucht werden können.

Desweiteren gibt es auch einige andere Ansätze, die sich mit partiellen stabilen Modellen und 3-wertiger Logik beschäftigen, oder die die Ideen der Stable Model Semantik auf disjunktive Logikprogramme oder auch solche Logikprogramme, die klassische Negation erlauben, übertragen. Die Fülle an Literatur zu diesem Thema und auch ganz allgemein zu deklarativen Semantiken von Logikprogrammen lässt darauf schließen, dass die Suche nach geeigneten deklarativen Semantiken für beliebige Logikprogramme bzw. deduktive Datenbanken nach wie vor ein wichtiges Thema ist.