Technische Universität Dresden
Fakultät Informatik

Diplomarbeit

zum Thema

Ein massiv paralleles Berechnungsmodell für normale logische Programme

von
Yvonne Kalinke
geboren am 06.03.1971 in Rodewisch

Eingereicht am 09. Juli 1994

Betreuer und Verantwortlicher Hochschullehrer: Prof. Dr. S. Hölldobler

Inhaltsverzeichnis

1. Einleitung

Das imperative Programmierparadigma der 70er Jahre orientierte sich stark an den technischen Voraussetzungen einer Programmberechnung. Für viele Aufgabenstellungen ist es aber sinnvoll, das Problem in einer Sprache zu beschreiben, die stärker den menschlichen Denkabläufen entspricht. Dies führte zur Entwicklung deklarativer Sprachen. Zu Beginn der 80er Jahre fanden deshalb eine Vielzahl neuer Programmiersprachen eine größere Verbreitung. PROLOG ist eine deklarative Sprache, die es erlaubt, ein Problem mit den Mitteln der Prädikatenlogik in Form eines normalen logischen Programms zu beschreiben.

Ein wichtiges Problem ist bei diesem Ansatz die eindeutige Bestimmung einer Semantik, damit konkrete Implementationen auf realen Rechnern hinsichtlich ihrer Korrektheit geprüft werden können.

Unser Interesse gilt deshalb den Aussagen, die als logische Konsequenzen aus dem Programm abgeleitet werden können. Diese beschreiben die Bedeutung, die *deklarative Semantik*, des Programms. Ein Ziel, das heißt eine Anfrage an ein Programm, soll genau dann positiv beantwortet werden, falls das Ziel eine logische Konsequenz aus der Menge der Programmklauseln ist. Anderenfalls soll die Anfrage fehlschlagen.

Die Menge der logischen Konsequenzen entspricht dem kleinsten Modell für ein logisches Programm, so daß sich die Problemstellung auf die Ermittlung des kleinsten Modelles für das Programm reduziert. Eine der bekanntesten Methoden zur Bestimmung der Semantik eines Programms $\mathcal{P}$ in einer zweiwertigen Logik ist die Nutzung eines Operators, der gewöhnlich mit $\mathcal{T}_{\mathcal{P}}$ notiert wird. Für definite Programme, also Programme, in denen keine Negation erlaubt ist, kann man mit Hilfe der Verbandstheorie zeigen, daß $\mathcal{T}_{\mathcal{P}}$ monoton und stetig ist. Aufgrund dieser Eigenschaften besitzt die Abbildung $\mathcal{T}_{\mathcal{P}}$ einen eindeutig bestimmten kleinsten Fixpunkt, der dem kleinsten Modell für $\mathcal{P}$ entspricht[1]. Für definite Programme liefert uns $\mathcal{T}_{\mathcal{P}}$ also ein Berechnungsmodell zur Bestimmung der Semantik. Erweitert man die Betrachtung auf normale logische Programme, das heißt, läßt man im Körper der Programmklauseln negierte Literale zu, kann die Monotonie von $\mathcal{T}_{\mathcal{P}}$ und somit die Existenz eines Fixpunktes für $\mathcal{T}_{\mathcal{P}}$ mit Hilfe der Verbandstheorie nicht mehr gezeigt werden.

Für ein normales logisches Programm können mehrere minimale Modelle existieren. Wel-

[1] siehe [Llo87]

ches Modell beschreibt dann die Semantik dieses Programms?

Als eine Erweiterung der von Clark vorgeschlagenen *Programmvervollständigung*[2] schlägt Fitting in [Fit85] vor, das Programm in eine dreiwertige Logik zu transformieren. Er zeigt, daß das Programm in einer dreiwertigen Logik ein eindeutig bestimmtes minimales Herbrand-Modell besitzt. Unter Nutzung eines dreiwertigen Fixpunktoperators $\Phi_{\mathcal{P}}$ kann dann die Semantik des Programms in der dreiwertigen Logik ermittelt werden. Transformiert man den so ermittelten dreiwertigen Fixpunkt zurück in die zweiwertige Logik, erhält man einen zweiwertigen Fixpunkt. Unter verschiedenen Voraussetzungen kann man zeigen, daß der so ermittelte Fixpunkt der gesuchte Fixpunkt des Problems in der zweiwertigen Logik ist.

Diese Methode führt jedoch in einigen Fällen zu unerwünschten Resultaten. So existieren normale logische Programme mit einem Fixpunkt in der zweiwertigen Logik, der nicht als Fixpunkt des Operators $\Phi_{\mathcal{P}}$ in der dreiwertigen Logik ermittelt werden kann[3].

Eine weitere Möglichkeit zur Bestimmung der Semantik normaler Programme liefern sogenannte *kanonische Modelle*. Für eine möglichst große Klasse von Programmen wird versucht, aus den minimalen Modellen für ein Programm das Modell auszuwählen, welches die vom Programmierer beabsichtigte Semantik[4] beschreibt. So führten Van Gelder etal. in [GRS88] *wohlfundierte Modelle* ein. Przymusinski zeigt in [Prz88] für die Klasse *lokal stratifizierter* Programme, daß diese ein *perfektes Modell* besitzen. In [GL88] schlagen Gelfond und Lifschitz ebenfalls eine neue deklarative Semantik für normale logische Programme vor, die *stabile Modellsemantik*. Für alle angegebenen Semantiken findet man jedoch Programme, deren kanonisches Modell nicht mit der beabsichtigten Semantik übereinstimmt.

Mehrfach wurden auch metrische Methoden vorgeschlagen, so von Batarekh in [Bat89]. In [Fit93] schlägt Fitting eine neue metrische Methode vor, die zur Berechnung der Semantik *akzeptabler* Programme verwendet werden kann. Diese beseitigt viele der in den bisher bekannten Methoden aufgetretenen Probleme. Die Idee besteht darin, einem Programm $\mathcal{P}$ einen vollständigen metrischen Raum $(\mathcal{M}_{\mathcal{P}}, d_{\mathcal{P}})$ zuzuordnen, so daß $\mathcal{T}_{\mathcal{P}}$ die Eigenschaft einer Kontraktion in $(\mathcal{M}_{\mathcal{P}}, d_{\mathcal{P}})$ besitzt. Banachs Kontraktionstheorem sichert dann die Existenz und Eindeutigkeit eines Fixpunktes für $\mathcal{T}_{\mathcal{P}}$.

Basierend auf der von Hornik etal. in [HSW89] formal bewiesenen Aussage, daß ein mehrschichtiges neuronales Netz jede (Borel-)meßbare Funktion hinreichend exakt berechnen kann, entstand die Idee, die Funktion $\mathcal{T}_{\mathcal{P}}$ durch ein mehrschichtiges neuronales Netz berechnen zu lassen. Die von Fitting vorgeschlagene metrische Methode erscheint dabei direkt übertragbar auf ein neuronales Netz. Dies liefert ein neues Berechnungsmodell für die Semantik logischer Programme und eine enge Beziehung zwischen logischer Programmierung und neuronaler Informationsverarbeitung.

In der vorliegenden Arbeit soll für den aussagenlogischen Fall untersucht werden, wie

[2] siehe [Cla87]
[3] [AP93]
[4] intended semantics, siehe [Llo87]

dieses Berechnungsmodell realisiert werden kann. Desweiteren soll gezeigt werden, daß ein Perzeptron nicht ausreicht, um $\mathcal{T}_\mathcal{P}$ für definite Programme zu berechnen. Außerdem wollen wir zeigen, wie zu einem aussagenlogischen Programm ein Netz konstruiert werden kann, das $\mathcal{T}_\mathcal{P}$ in zwei Schritten berechnet. Es soll gezeigt werden, daß die Iteration des Netzes die Iteration von $\mathcal{T}_\mathcal{P}$ berechnet. Außerdem wollen wir die Gültigkeit der Umkehrung dieser Aussage untersuchen.

Dieses neue, massiv parallele und modell-generierende Berechnungsmodell für aussagenlogische Programme unterscheidet sich wesentlich von herkömmlichen parallelen Verarbeitungsmodellen, die im allgemeinen UND-, ODER-, oder Unifikations-Parallelismus implementieren. Außerdem umfaßt die Klasse der akzeptablen Programme eine sehr große Klasse von Programmen, für die unser Modell in der Lage ist, die Semantik zu berechnen. Ein weiterer Vorteil des Berechnungsmodelles liegt in seiner einfachen Adaptierbarkeit auf mehrwertige Logiken.

Im ersten Kapitel wird eine Einführung in die verwendete Syntax und Semantik gegeben. Im ersten Abschnitt erläutern wir wichtige Definitionen der logischen Programmierung. Es folgt eine kurze Einführung zu Begriffen aus der Topologie und dem Gebiet der neuronalen Netze.

Im folgenden Kapitel wird die von Fitting vorgeschlagene metrische Methode vorgestellt.

Im vierten Kapitel wird die der Arbeit zugrundeliegende Idee eines neuen Berechnungsmodelles für logische Programme erläutert.

Im fünften Kapitel beschränken wir unsere Betrachtungen auf den aussagenlogischen Fall. Es sollen Aussagen darüber getroffen werden, welche Arten von Netzen benötigt werden, um den Operator $\mathcal{T}_\mathcal{P}$ für ein aussagenlogisches Programm zu berechnen, wie für ein Programm ein Netz gefunden wird, das den Operator $\mathcal{T}_\mathcal{P}$ für das Netz berechnet und wie man zu einem Netz ein Programm findet, so daß das Netz die Funktion $\mathcal{T}_\mathcal{P}$ für das Programm berechnet. Mit Hilfe der von Fitting getroffenen Aussagen über die Konvergenz der Berechnung von $\mathcal{T}_\mathcal{P}$ für ein akzeptables Programm $\mathcal{P}$, kann eine Aussage über die Konvergenz des Netzes getroffen werden. Eine umgekehrte Aussage kann ebenfalls präzisiert werden. Zu einem gegebenen 3–schichtigen Netz, dessen Iteration unabhängig von einer Anfangsbelegung konvergiert, findet man eine Abbildung, deren Iteration ebenfalls konvergiert.

Anschließend sollen mehrwertige Logiken hinsichtlich der Anwendbarkeit des neuen Berechnungsmodelles untersucht werden.

Im siebten Kapitel betrachten wir Programme mit genau einem stabilen Modell und wollen Aussagen über Beziehungen dieser Programme zu akzeptablen Programmen treffen.

2. Grundlagen

2.1 Semantik logischer Programme

Es wird die Standardnotation und Terminologie von Lloyd[1] verwendet. Eine ausführlichere Darstellung der logischen Grundlagen findet man zum Beispiel in [Rei94].

Definition 2.1 *Das* Alphabet *der Prädikatenlogik erster Stufe besteht aus den folgenden Symbolen:*

(i) *Variablen*
(ii) *Konstanten*
(iii) *Funktionssymbole*
(iv) *Prädikatszeichen*
(v) *Junktoren* $\neg$ $\wedge$ $\vee$
(vi) *Quantoren* $\exists$ $\forall$
(vii) *Begrenzer* ( ) ,

Wir bezeichnen im folgenden Variablen mit Großbuchstaben X, Y, Z, Konstanten mit kleinen Buchstaben $a, b, c, \ldots$, Funktionszeichen mit f, g, h und Prädikatszeichen mit p, q, r. Aussagenlogische Variable bezeichnen wir mit den Großbuchstaben $A, B, C, \ldots$.

Definition 2.2 *Ein* Term *ist*

(i) *eine Variable oder*
(ii) *eine Konstante oder*
(iii) $f(t_1, \ldots, t_n)$*, falls* f *ein* n*–stelliges Funktionssymbol und* $t_1, \ldots, t_n$ *Terme sind.*

Definition 2.3 *Eine* Formel *ist induktiv definiert wie folgt*

(i) *Sei* p *ein* n*–stelliges Prädikatszeichen und* $t_1, \ldots, t_n$ *seien Terme, dann ist* $p(t_1, \ldots, t_n)$ *eine Formel (genannt* atomare Formel *oder kürzer* Atom*).*
(ii) *Seien* F *und* G *Formeln. Dann sind auch* $\neg F$*,* $F \wedge G$ *und* $F \vee G$ *Formeln.*
(iii) *Sei* F *eine Formel und* X *sei eine Variable. Dann sind auch* $\forall X.F$ *und* $\exists X.F$ *Formeln.*

[1] [Llo87]

Als abkürzende Schreibweisen führen wir ein:

$$F \rightarrow G \text{ für } (\neg F \vee G)$$

$$F \leftrightarrow G \text{ für } (F \leftarrow G) \wedge (F \rightarrow G)$$

Definition 2.4 *Für jede Formel F definieren wir die Menge $var(F)$ der* freien Variablen *in F induktiv durch:*

$$\begin{aligned}
&var(p(t_1, \ldots, t_n)) = \textstyle\bigcup_{i=1}^{n} var(t_i),\\
&var(\neg F) = var(F),\\
&var(F \wedge G) = var(F) \cup var(G),\\
&var(F \vee G) = var(F) \cup var(G),\\
&var(\exists X.F) = var(F) - \{X\},\\
&var(\forall X.F) = var(F) - \{X\}
\end{aligned}$$

mit $var(X) = \{X\}$, $var(c) = \emptyset$ und $var(f(t_1, \ldots, t_n)) = \bigcup_{i=1}^{n} var(t_i)$.

Definition 2.5 *Einen Term t mit $var(t) = 0$ wird als* Grundterm *bezeichnet.*

Definition 2.6 *Ein atomare Formel A mit $var(A) = 0$ wird als* Grundatom *bezeichnet.*

Definition 2.7 *Eine Formel F heißt* abgeschlossene Formel, *wenn $var(F) = \emptyset$.*

Definition 2.8 *Ein* Literal *ist ein Atom oder die Negation eines Atomes. Ein* positives Literal *ist ein Atom. Ein* negatives Literal *ist die Negation eines Atomes.*

Definition 2.9 *Eine* Programmklausel *ist eine Klausel der Form*

$$A \leftarrow L_1, \ldots, L_m$$

wobei A ein Atom ist und $L_1, \ldots, L_m$ Literale. Wir bezeichnen A als Kopf der Klausel und $L_1, \ldots, L_m$ als Körper der Klausel.

Definition 2.10 *Eine* definite Programmklausel *ist eine Klausel der Form*

$$A \leftarrow B_1, \ldots, B_m$$

wobei $A, B_1, \ldots, B_m$ Atome sind.

Definition 2.11 *Ein* normales Programm *ist eine endliche Menge von Programmklauseln.*

Definition 2.12 *Ein* definites Programm *ist eine endliche Menge von definiten Programmklauseln.*

Bemerkung Zur Abkürzung der Schreibweise bezeichnen wir im folgenden normale logische Programme als Programme und definite logische Programme als definite Programme.

In einem normalen Programm $\mathcal{P}$ sind negierte Literale im Körper der Klauseln erlaubt. Für die Berechnung der Semantik dieser Programme ist bisher kein allgemeines Verfahren bekannt. Für die eingeschränkte Menge der definiten Programme liefert die Verbandstheorie eine Methode zur Bestimmung der Semantik. Dabei hat sich gezeigt, daß man nicht alle Interpretationen für $\mathcal{P}$ betrachten muß. Es ist ausreichend, die Menge aller Herbrand–Interpretationen für $\mathcal{P}$ zu betrachten.

Definition 2.13 *Das* Herbrand–Universum $U_\mathcal{P}$ *für ein Programm* $\mathcal{P}$ *ist die Menge aller Grundterme, die mit den Konstanten und Funktionssymbolen, die in* $\mathcal{P}$ *vorkommen, gebildet werden können. Falls in* $\mathcal{P}$ *keine Konstanten vorkommen, definieren wir eine beliebige Konstante, um Grundterme formen zu können.*

Definition 2.14 *Die* Herbrand–Basis $B_\mathcal{P}$ *für ein Programm* $\mathcal{P}$ *ist die Menge aller Formeln* $p(t_1, \ldots, t_n)$*, die mit allen in* $\mathcal{P}$ *vorkommenden* n*-stelligen Prädikatszeichen* p *und den Termen* $t_1, \ldots, t_n$ *des Herbrand-Universums* $U_\mathcal{P}$ *gebildet werden können.*

Beispiel 2.15 Wir betrachten das definite logische Programm $\mathcal{P}$

$$\begin{array}{lcl} p(X) & \leftarrow & q(f(X)). \\ r(Y). & & \end{array}$$

und ermitteln das Herbrand–Universum U_P und die Herbrand–Basis B_P:
$U_P = \{a, f(a), f(f(a)), f(f(f(a))), \ldots\}$
$B_P = \{p(a), r(a), q(a), p(f(a)), r(f(a)), q(f(a)), \ldots\}$ ◦

Definition 2.16 *Die Elemente der Menge* $\{0, 1\}$ *werden* Wahrheitswerte *genannt. Eine* Herbrand–Interpretation I *für ein Programm* $\mathcal{P}$ *ist eine Abbildung* $I : B_P \to \{0, 1\}$ *der Elemente der Herbrand–Basis von* $\mathcal{P}$ *auf die Wahrheitswerte 0 für falsch bzw. 1 für wahr.*

Die Schreibweise $A \in I$ bzw. $A \notin I$ $(A \in B_P)$ sei äquivalent zu $I(A) = 1$ bzw. $I(A) = 0$. Wir vereinbaren außerdem $I(\neg A) = 1$ gdw. $I(A) = 0$.

Beispiel 2.17 Für Beispiel 2.15 ist $I = \{r(a), p(a)\}$ eine Herbrand–Interpretation für $\mathcal{P}$, die die Grundatome $r(a)$ und $p(a)$ nach *wahr* und alle anderen Grundatome der Herbrand–Basis nach *falsch* abbildet. ◦

Wird jedes Vorkommen einer Variable in einer Programmklausel durch einen Grundterm ersetzt, so bezeichnen wir die erhaltene Klausel als eine *Grundinstanz* der Programmklausel. Mit ground($\mathcal{P}$) bezeichnen wir die Menge aller Grundinstanzen von Programmklauseln aus $\mathcal{P}$.

Für jedes Programm $\mathcal{P}$ gibt es eine äquivalente quantorenfreie Normalform[2].

Bisher haben wir nur Interpretationen von Atomen betrachtet. Die Interpretation I^* ist eine Erweiterung von I auf Formeln. Seien F und G Formeln. Es gilt:

$$\begin{aligned}
I^*(F) &= I(F) \text{ für } F \in B_{\mathcal{P}} \\
I^*((F \wedge G)) &= \begin{cases} 1 & \text{falls} \quad I^*(F) = 1 \text{ und } I^*(G) = 1 \\ 0 & \text{sonst} \end{cases} \\
I^*((F \vee G)) &= \begin{cases} 1 & \text{falls} \quad I^*(F) = 1 \text{ oder } I^*(G) = 1 \\ 0 & \text{sonst} \end{cases} \\
I^*(\neg F) &= \begin{cases} 1 & \text{falls} \quad I^*(F) = 0 \\ 0 & \text{sonst} \end{cases}
\end{aligned}$$

Definition 2.18 *Sei I eine Herbrand–Interpretation und F eine abgeschlossene Formel. Dann ist I ein* Modell *für F (notiert mit $I \models F$), falls $I^*(F) = 1$.*

Definition 2.19 *Sei I eine Herbrand–Interpretation und S eine Menge von Formeln. Dann ist I ein* Modell *für S, falls I ein Modell für jedes Element von S ist.*

Definition 2.20 *Sei S eine Menge abgeschlossener Formeln und F eine abgeschlossene Formel. Wir bezeichnen F als eine* logische Konsequenz *aus S, falls für jede Interpretation I gilt:*

$$I \models S \quad \Rightarrow \quad I \models F$$

Definition 2.21 *Sei $\mathcal{P}$ ein normales logisches Programm. Ein* Herbrand–Modell *für $\mathcal{P}$ ist eine Herbrand–Interpretation, die Modell für $\mathcal{P}$ ist.*

Beispiel 2.22 Für das Programm $\mathcal{P}$ aus Beispiel 2.15 sind die Interpretationen $I_1 = \{r(Y) \mid Y \in U_P\}$ und $I_2 = \{r(Y) \mid Y \in U_P\} \cup \{q(f(a)), p(a)\}$ Herbrand–Modelle. ◦

Einem Programm $\mathcal{P}$ soll eindeutig ein Modell zugeordnet werden, das die *deklarative Semantik* von $\mathcal{P}$ beschreibt. Für ein definites Programm $\mathcal{P}$ wählt man aus der Menge aller Herbrand–Interpretationen das *kleinste Modell $M_{\mathcal{P}}$* bezüglich der Inklusionsrelation $\subseteq$ aus, da dieses eindeutig bestimmt ist. $M_{\mathcal{P}}$ entspricht sowohl dem Durchschnitt über alle Herbrand–Modelle für $\mathcal{P}$, als auch der Menge der logischen Konsequenzen aus $\mathcal{P}$. Die Menge aller logischen Konsequenzen aus $\mathcal{P}$ beschreibt jedoch genau die Menge der Anfragen an ein Programm, die positiv beantwortet werden sollen und erscheint damit geeignet, die Bedeutung eines Programmes zu beschreiben.

Beispiel 2.23 Für das Programm $\mathcal{P}$ aus Beispiel 2.15 ist $I = \{r(Y) \mid Y \in U_{\mathcal{P}}\}$ kleinstes Herbrand–Modell für $\mathcal{P}$. Die leere Menge und jede kleinere Menge, die $r(Y)$ für weniger Elemente des Herbrand–Universums enthält, sind kein Modell. Jedes andere Modell

[2] siehe [Rei94]

würde noch weitere Grundatome enthalten und wäre bezüglich der Inklusionsrelation nicht kleiner als I $\circ$

Theorem 2.24 [Llo87] *Sei $\mathcal{P}$ ein definites Programm. Es gilt $M_{\mathcal{P}} = \{ A \in B_P \mid A$ ist logische Konsequenz aus $\mathcal{P}$ $\}$.*

Um das kleinste Modell für ein Programm $\mathcal{P}$ zu berechnen, nutzen wir eine Funktion $\mathcal{T}_{\mathcal{P}}$, die über der Potenzmenge der Herbrand–Basis für $\mathcal{P}$ definiert ist. Für ein definites Programm $\mathcal{P}$ kann man zeigen, daß der kleinste Fixpunkt von $\mathcal{T}_{\mathcal{P}}$ genau der Menge $M_{\mathcal{P}}$ entspricht.

Definition 2.25 *Sei $\mathcal{P}$ ein definites Programm. Die Abbildung $\mathcal{T}_{\mathcal{P}} : 2^{B_{\mathcal{P}}} \to 2^{B_{\mathcal{P}}}$ ist wie folgt definiert. Sei I eine Herbrand–Interpretation für $\mathcal{P}$. Dann gilt $\mathcal{T}_{\mathcal{P}}(I) = \{A \in B_{\mathcal{P}} \mid A \leftarrow A_1, A_2, \ldots, A_n$ ist Grundinstanz einer Klausel in $\mathcal{P}$ und $I(A_1) = I(A_2) = \cdots = I(A_n) = wahr\}$.*

Mit Hilfe der Verbandstheorie können wir zeigen, daß die Menge aller Herbrand–Interpretationen für $\mathcal{P}$ einen vollständigen Verband bildet und die Funktion $\mathcal{T}_{\mathcal{P}}$ monoton ist. Aufgrund dieser Eigenschaften ist der kleinste Fixpunkt von $\mathcal{T}_{\mathcal{P}}$ eindeutig bestimmt.

Definition 2.26 *Sei $T : L \to L$ eine Abbildung der Menge L auf sich selbst. Wir bezeichnen a $(a \in L)$ als* Fixpunkt *von T, falls $T(a) = a$.*

Sei L ein vollständiger Verband. Wir bezeichnen mit $\top$ das größte Element von L, das heißt, die kleinste obere Schranke[3] $\mathrm{lub}(L)$ von L und mit $\perp$ das kleinste Element von L, das heißt, die größte untere Schranke[4] $\mathrm{glb}(L)$ von L.

Definition 2.27 *Sei L ein vollständiger Verband und $T : L \to L$ eine Abbildung. T ist* monoton*, falls $T(x) \leq T(y)$ für $x \leq y$ gilt.*

Definition 2.28 *Sei L ein vollständiger Verband und X eine Teilmenge $X \subseteq L$. Die Menge X ist* gerichtet*, falls jede endliche Teilmenge von X eine obere Schranke in X besitzt.*

Definition 2.29 *Sei L ein vollständiger Verband und $T : L \to L$ eine Abbildung. T ist* stetig*, falls $T(\mathrm{lub}(X)) = \mathrm{lub}(T(X))$ für jede gerichtete Teilmenge X von L.*

Stetigkeit ist eine schärfere Eigenschaft als Monotonie. Die Monotonie kann man aus der Stetigkeit folgern.

[3] engl. least upper bound
[4] engl. greatest lower bound

Definition 2.30 *Sei L ein vollständiger Verband und $T : L \rightarrow L$ sei monoton. Wir definieren*

$$\begin{array}{lcl} T \uparrow 0 & = & \bot \\ T \uparrow \alpha & = & T(T \uparrow (\alpha - 1)) \textit{ falls } \alpha \textit{ Nachfolger einer Ordinalzahl} \\ T \uparrow \alpha & = & \mathrm{lub}\{T \uparrow \beta \mid \beta < \alpha\} \textit{ falls } \alpha \textit{ eine Limeszahl} \end{array}$$

Proposition 2.31 [Tar55] *Sei L ein vollständiger Verband und $T : L \rightarrow L$ sei monoton. Dann hat T einen kleinsten Fixpunkt*[5] $\mathrm{lfp}(T)$ *und einen größten Fixpunkt*[6] $\mathrm{gfp}(T)$.

Proposition 2.32 [Llo87] *Sei L ein vollständiger Verband und $T : L \rightarrow L$ sei stetig. Dann gilt* $\mathrm{lfp}(T) = T \uparrow \omega$.

Da man die Stetigkeit des in Definition 2.25 definierten Operators $\mathcal{T}_\mathcal{P}$ auf dem vollständigen Verband der Herbrand–Interpretationen für ein definites Programm $\mathcal{P}$ zeigen kann[7], liefert $\mathcal{T}_\mathcal{P} \uparrow \omega$ eine Möglichkeit, den kleinsten Fixpunkt von $\mathcal{T}_\mathcal{P}$ für ein definites Programm zu ermitteln. Der Operator $\mathcal{T}_\mathcal{P}$ liefert somit die Verbindung zwischen der deklarativen und der prozeduralen Semantik eines definiten Programmes.

Beispiel 2.33 Sei $\mathcal{P}$ das definite logische Programm:

$$\begin{array}{lcl} p(a). & & \\ p(b). & & \\ q(X) & \leftarrow & p(X). \end{array}$$

Das Herbrand–Universum, die Herbrand–Basis und die Potenzmenge der Herbrand–Basis für $\mathcal{P}$ ermitteln wir mit:

$$\begin{array}{lcl} U_\mathcal{P} & = & \{a, b\} \\ B_\mathcal{P} & = & \{p(a), p(b), q(a), q(b)\} \\ 2^{B_\mathcal{P}} & = & \{\emptyset, \{p(a)\}, \{p(b)\}, \{q(a)\}, \{q(b)\}, \{p(a), p(b)\}, \{p(a), q(a)\}, \{p(a), q(b)\}, \\ & & \{p(b), q(a)\}, \{p(b), q(b)\}, \{q(a), q(b)\}, \{p(a), q(a), q(b)\}, \ldots\} \end{array}$$

Mit der in Definition 2.30 angegebenen Berechnungsvorschrift ermitteln wir:

$$\begin{array}{lcl} \mathcal{T}_\mathcal{P} \uparrow 0 & = & \bot = \emptyset \\ \mathcal{T}_\mathcal{P} \uparrow 1 & = & \mathcal{T}_\mathcal{P}(\emptyset) = \{p(a), p(b)\} \\ \mathcal{T}_\mathcal{P} \uparrow 2 & = & \mathcal{T}_\mathcal{P}(\{p(a), p(b)\}) = \{p(a), p(b), q(a), q(b)\} \\ \mathcal{T}_\mathcal{P} \uparrow 3 & = & \mathcal{T}_\mathcal{P}(\{p(a), p(b), q(a), q(b)\}) = \{p(a), p(b), q(a), q(b)\} = \mathcal{T}_\mathcal{P} \uparrow 2 \end{array}$$

woraus $\mathrm{lfp}(\mathcal{T}_\mathcal{P}) = \{p(a), p(b), q(a), q(b)\}$ folgt. ∘

[5] engl. least fixpoint

[6] engl. greatest fixpoint

[7] siehe [Llo87]

Für ein normales Programm $\mathcal{P}$ kann der Operator $\mathcal{T}_{\mathcal{P}}$ analog zum definiten Fall definiert werden. Jedoch kann die Monotonie von $\mathcal{T}_{\mathcal{P}}$ auf dem Verband der Herbrand–Interpretationen für $\mathcal{P}$ nicht gezeigt werden. Die Existenz eines kleinsten Fixpunktes für $\mathcal{T}_{\mathcal{P}}$ ist demzufolge nicht gesichert. Für ein normales Programm können auch mehrere minimale Herbrand–Modelle oder keines existieren.

Beispiel 2.34 Das Programm $\mathcal{P}$:

$$\begin{array}{lcl} p & \leftarrow & \neg q. \\ q & \leftarrow & \neg p. \end{array}$$

hat zwei minimale Herbrand–Modelle $I_1 = \{p\}$ und $I_2 = \{q\}$. ○

Beispiel 2.35 Das Programm $\mathcal{P}$:

$$p \quad \leftarrow \quad \neg p.$$

hat kein Herbrand–Modell. Die Anwendung des Operators $\mathcal{T}_{\mathcal{P}}$ liefert

$\mathcal{T}_{\mathcal{P}} \uparrow 0 = \emptyset$
$\mathcal{T}_{\mathcal{P}} \uparrow 1 = \{p\}$
$\mathcal{T}_{\mathcal{P}} \uparrow 2 = \emptyset$
$\mathcal{T}_{\mathcal{P}} \uparrow 3 = \{p\}$
usw. ○

Um die Frage zu beantworten, welches der minimalen Modelle, falls eines existiert, die Semantik des Programms charakterisiert, wurde eine Anzahl von Ansätzen vorgeschlagen. Dabei wurden zwei Hauptrichtungen verfolgt. Zum einen die *Programmvervollständigung* und zum anderen die Verwendung sogenannter *kanonischer Modelle*.

2.1.1 Semantik der Programmvervollständigung

Die Anwendung der Programmvervollständigung geht auf Clark[8] zurück. Ein Interpreter beziehungsweise Compiler vervollständigt ein Programm, indem er alle Belegungen eines Prädikatszeichens als falsch interpretiert, falls sie nicht ausdrücklich durch den Programmtext gelten müssen. Wir unterstellen zunächst, daß so die vom Programmierer beabsichtigte Semantik beschrieben wird.

Definition 2.36 *Die* Definition eines Prädikatszeichens *p in einem normalen Programm $\mathcal{P}$ ist die Menge aller Programmklauseln in $\mathcal{P}$ mit einem Vorkommen von p im Kopf der Klausel.*

[8] siehe [Cla87]

Sei $p(t_1, \ldots, t_n) \leftarrow L_1 \wedge \cdots \wedge L_m$ eine Programmklausel in einem normalen Programm $\mathcal{P}$. Wir führen ein neues Prädikatszeichen $=$ ein, das nicht in $\mathcal{P}$ vorkommt und dessen Interpretation der Gleichheitsrelation entspricht. Wir transformieren die gegebene Klausel nach

$$p(X_1, \ldots, X_n) \leftarrow (X_1 = t_1) \wedge \cdots \wedge (X_n = t_n) \wedge L_1 \wedge \cdots \wedge L_m$$

wobei $X_1, \ldots, X_n$ Variablen bezeichnen, die nicht in der Klausel vorkommen. Die so erhaltene Klausel transformieren wir, falls $Y_1, \ldots, Y_d$ Variable in der Originalklausel sind, nach

$$p(X_1, \ldots, X_n) \leftarrow \exists Y_1 \ldots \exists Y_d \quad (X_1 = t_1) \wedge \cdots \wedge (X_n = t_n) \wedge L_1 \wedge \cdots \wedge L_m.$$

Diese Transformation führen wir für alle Klauseln, die Elemente der Definition von p sind, durch und erhalten $k \geq 1$ transformierte Klauseln der Form

$$\begin{array}{rcl} p(X_1, \ldots, X_n) & \leftarrow & E_1 \\ & \vdots & \\ p(X_1, \ldots, X_n) & \leftarrow & E_k \end{array}$$

wobei jedes der E_i von der allgemeinen Form

$$\exists Y_1 \ldots \exists Y_d \quad (X_1 = t_1) \wedge \cdots \wedge (X_n = t_n) \wedge L_1 \wedge \cdots \wedge L_m$$

ist. Die *vervollständigte Definition* von p ist dann die folgende Klausel

$$\forall X_1, \ldots, X_n \quad (p(X_1, \ldots, X_n) \longleftrightarrow E_1 \vee \cdots \vee E_k).$$

In $\mathcal{P}$ kann es auch Prädikatszeichen geben, die nicht im Kopf einer Klausel von $\mathcal{P}$ auftreten. Für jedes dieser Prädikatszeichen q fügen wir explizit die folgende Klausel

$$\forall X_1, \ldots, X_n \quad \neg q(X_1, \ldots, X_n)$$

hinzu. Diese Definition entspricht der implizit durch das Programm gegebenen Definition eines solchen Prädikatszeichens q. Die so erhaltene Klausel bezeichnen wir ebenfalls als die Definition des Prädikatszeichens q.

Definition 2.37 *Sei $\mathcal{P}$ ein normales Programm. Die* Vervollständigung *von $\mathcal{P}$, bezeichnet mit* comp($\mathcal{P}$)*, ist die Menge aller vervollständigten Definitionen von Prädikatszeichen in $\mathcal{P}$ zusammen mit der Gleichungstheorie.*

Die folgende Gleichungstheorie ist ausreichend für unseren Zweck und soll mit EQ bezeichnet werden. Wir nutzen die Standardnotation $\neq$ für syntaktische Ungleichheit.

1. $c \neq d$ für alle Paare c, d unterschiedlicher Konstanten.
2. $\forall (f(X_1, \ldots, X_n) \neq g(Y_1, \ldots, Y_m))$ für alle Paare f, g unterschiedlicher Funktionssymbole.

3. $\forall(f(X_1,\ldots,X_n) \neq c)$ für jede Konstante c und jedes Funktionssymbol f.

4. $\forall(t[X] \neq X)$ für jeden Term $t[X]$, der X enthält und verschieden von X ist.

5. $\forall((X_1 \neq Y_1) \vee \cdots \vee (X_n \neq Y_n) \rightarrow f(X_1,\ldots,X_n) \neq f(Y_1,\ldots Y_n))$ für jedes Funktionssymbol f.

6. $\forall(X = X)$.

7. $\forall((X_1 = Y_1) \wedge \cdots \wedge (X_n = Y_n) \rightarrow f(X_1,\ldots,X_n) = f(Y_1,\ldots Y_n))$ für jedes Funktionssymbol f.

8. $\forall((X_1 = Y_1) \wedge \cdots \wedge (X_n = Y_n) \rightarrow p(X_1,\ldots,X_n) = p(Y_1,\ldots Y_n))$ für jedes Prädikatszeichen p (einschließlich $=$).

Beispiel 2.38 Sei $\mathcal{P}$ das Programm:

$$\begin{array}{rcl} p & \leftarrow & q, \neg r. \\ p & \leftarrow & r, \neg s. \\ q & \leftarrow & p. \\ r & \leftarrow & q. \end{array}$$

Die Vervollständigung faßt die Regeln für p zusammen, fügt eine Regel hinzu, die s als $falsch$ auszeichnet, ersetzt $\leftarrow$ durch $\longleftrightarrow$ und liefert somit:

$$\begin{array}{rclcl} \mathrm{comp}(\mathcal{P}) & = & \{\ p & \longleftrightarrow & ((q \wedge \neg r) \vee (r \wedge \neg s)), \\ & & q & \longleftrightarrow & p, \\ & & r & \longleftrightarrow & q, \\ & & \neg s\ \} & \cup & \{\ EQ\ \} \end{array}$$

$\circ$

Die Modelle der Vervollständigung eines Programms sind auch Modelle des ursprünglichen Programms[9]. In diesem Sinn ist die Programmvervollständigung korrekt. Es gibt aber selbstverständlich Modelle des ursprünglichen Programms, die nicht Modell der Programmvervollständigung sind. Wie soll ein Interpreter beziehungsweise Compiler, der lediglich mit der Programmvervollständigung arbeitet, auf Anfragen reagieren, die solchen Modellen entsprechen? In diesem Fall darf eine Anfrage an einen Interpreter weder fehlschlagen noch erfolgreich beantwortet werden.

In [Fit85] schlägt Fitting eine explizite 3–wertige Semantik vor[10]. Während die Formel $(p \vee \neg p)$ in der 2–wertigen Logik *wahr* sein muß, kann sie in der 3–wertigen Logik den Wahrheitswert **u** für *undefiniert* besitzen. Zusätzlich wird der bei der Programmvervollständigung erzeugte Operator $\longleftrightarrow$ als Operator mit der Bedeutung „hat denselben Wahrheitswert“ interpretiert, so daß $\mathbf{u} \longleftrightarrow \mathbf{u}$ als *wahr* interpretiert wird. Diese Veränderungen beseitigen einige Anomalien der von Clark vorgeschlagenen Methode.

[9] siehe [Cla87]

[10] Die Semantik der Junktoren $\neg$, $\wedge$ und $\vee$ für die dreiwertige Logik ist in Tabelle 6.1 auf Seite 65 definiert.

Beispiel 2.39 Sei $\mathcal{P}$ das folgende Programm mit nur einer Programmklausel:

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

Die Vervollständigung von $\mathcal{P}$ lautet

$$\text{comp}(\mathcal{P}) = \{p \longleftrightarrow (\neg p \wedge \neg q), \neg q\} \cup \{EQ\}$$

und besitzt kein 2–wertiges Modell. Fügen wir jedoch die „sinnlose" Regel

$$q \leftarrow q$$

hinzu, so ist $\{\neg p, q\}$ ein eindeutiges 2-wertiges Modell der Vervollständigung, da die Klausel $\neg q$ in der Vervollständigung von $\mathcal{P}$ entfällt. In beiden Versionen besitzt $\mathcal{P}$ ein 3–wertiges Modell mit $p = \mathbf{u}$. Ein Programmierer, der das Programm $\mathcal{P}$ schreibt, hat nicht im Sinn, daß q als $falsch$ interpretiert wird. Die Programmvervollständigung arbeitet jedoch explizit mit einer Klausel, die q als $falsch$ interpretiert, da q nicht im Kopf einer Klausel von $\mathcal{P}$ auftritt. ∘

Ein grundlegendes Resultat der Fittingschen Arbeit ist die Aussage, daß die Vervollständigung eines Programmes ein eindeutig bestimmtes minimales 3–wertiges Herbrand–Modell besitzt. Fitting schlägt vor, dieses Modell als die Semantik des Programms zu betrachten. In einigen Beispielen stimmt dieses Modell jedoch nicht mit dem Modell überein, das man unter Nutzung von Regeln des gesunden Menschenverstandes erhält.

2.1.2 Semantik kanonischer Modelle

Die Idee eines kanonischen Modells besteht darin, ein bestimmtes minimales Herbrand–Modell eines Programmes (falls ein solches existiert) als das durch das Programm beabsichtigte Modell, das heißt, das Modell, welches Programmierer und Nutzer im Sinn haben, zu bestimmen. Ein Problem dieser Anwendung liegt darin, daß es Programme gibt, für die man kein kanonisches Modell bestimmen kann, da keines oder mehrere Modelle als kanonisches Modell ausgewählt werden. Die Forschung auf diesem Gebiet konzentriert sich darauf, eine kanonische Modellsemantik für eine möglichst große Klasse von Programmen zu ermitteln.

Die Klasse der *stratifizierten* Programme und die Klasse der *lokal stratifizierten* Programme werden unter Nutzung einer Stufenabbildung definiert.

Definition 2.40 *Eine* Stufenabbildung *für ein normales Programm $\mathcal{P}$ ist eine Abbildung $|\_|: B_{\mathcal{P}} \rightarrow \mathbb{N}$ von Grundatomen auf natürliche Zahlen. Wir bezeichnen den Wert von $A \in B_{\mathcal{P}}$ bezüglich der Stufenabbildung als die Stufe von A.*

Beispiel 2.41 Wir betrachten das definite aussagenlogische Programm $\mathcal{P}$:

$$\begin{array}{lll} A. & & \\ A & \leftarrow & B. \\ A & \leftarrow & C, D. \\ A & \leftarrow & E, F. \end{array}$$

Die Abbildung

$$\begin{array}{rcrcl} |\_| & : & A & \mapsto & 0 \\ & & B & \mapsto & 1 \\ & & C & \mapsto & 3 \\ & & D & \mapsto & 1 \\ & & E & \mapsto & 5 \\ & & F & \mapsto & 1 \end{array}$$

ist eine Stufenabbildung für $\mathcal{P}$. ◦

Wir erweitern eine Stufenabbildung auf natürliche Weise zu einer Abbildung von Grundliteralen auf natürliche Zahlen, indem wir $|\neg A| = |A|$ setzen.
Wird nur eine Teilmenge der Herbrand–Basis durch eine Stufenabbildung abgebildet, so erhalten wir eine partielle Stufenabbildung.

Definition 2.42 *Eine* partielle Stufenabbildung *für ein normales Programm $\mathcal{P}$ ist eine Abbildung $|\_|: B_0 \rightarrow I\!N$ von Grundatomen auf natürliche Zahlen, wobei B_0 eine beliebige Teilmenge der Herbrand–Basis für $\mathcal{P}$ ist.*

Bemerkung Eine Stufenabbildung, die über Prädikatszeichen aus $\mathcal{P}$ definiert ist, bezeichnen wir mit $|\_|_{PZ}$. Eine Stufenabbildung $|\_|$ über Grundatomen der Herbrand–Basis von $\mathcal{P}$ kann durch eine Stufenabbildung $|\_|_{PZ}$ über Prädikatszeichen definiert werden:

$$|p(X)| := |p|_{PZ} \quad \forall X \in U_{\mathcal{P}}$$

Eine Stufenabbildung $|\_|_{PZ}$ über Prädikatszeichen kann im allgemeinen nicht durch eine Stufenabbildung $|\_|$ über Grundatomen definiert werden.

Definition 2.43 *Ein normales Programm $\mathcal{P}$ ist* stratifiziert, *falls eine Stufenabbildung $|\_|_{PZ}$ gefunden werden kann, so daß in jeder Programmklausel $A \leftarrow L_1, \ldots, L_n$ die Stufe der Prädikatszeichen aller positiven Literale im Körper kleiner oder gleich der Stufe des Prädikatszeichens in A ist und die Stufe der Prädikatszeichen aller negativen Literale im Körper kleiner als die Stufe des Prädikatszeichens in A ist.*

Ein Programm $\mathcal{P}$ ist also genau dann stratifiziert, falls sein Abhängigkeitsgraph keine Zyklen mit negativen Kanten enthält. Ein Abhängigkeitsgraph $\mathcal{G}$ für ein Programm $\mathcal{P}$ ist ein Graph, dessen Knoten Prädikatszeichen aus $\mathcal{P}$ repräsentieren. Seien p und q Prädikatszeichen in $\mathcal{P}$, dann existiert in $\mathcal{G}$ eine gerichtete Kante von p nach q, falls in $\mathcal{P}$ eine Klausel mit einem Vorkommen von p im Kopf der Klausel und einem Vorkommen von q im Körper der Klausel existiert. Tritt q negiert auf, bezeichnen wir die Kante als negativ.

Definition 2.44 *Ein normales Programm $\mathcal{P}$ ist* lokal stratifiziert, *falls eine Stufenabbildung $|\_|$ gefunden werden kann, so daß in jeder Programmklausel die Stufe aller* *positiven Literale im Körper kleiner oder gleich der Stufe des Atomes im Kopf der Klausel und die Stufe aller negativen Literale im Körper kleiner als die Stufe des Atomes im Kopf der Klausel ist.*

Beispiel 2.45 Wir betrachten das definite Programm $\mathcal{P}$:

```
member(E,[E|Liste]).
member(E,[X|Liste])   ←   member(E,Liste).
```

$\mathcal{P}$ ist lokal stratifiziert, da eine Stufenabbildung

$$|\_| \;:\; \texttt{member(E,Liste)} \mapsto \text{Länge}(\texttt{Liste})$$

gefunden werden kann, die die geforderte Eigenschaft besitzt.
$\mathcal{P}$ ist stratifiziert mit der Stufenabbildung $|\_|_{PZ}\colon \texttt{member} \mapsto 0$. ◦

Beispiel 2.46 Das Programm $\mathcal{P} = \{p(a); p(b) \leftarrow \neg p(a)\}$ ist nicht stratifiziert. Da das Prädikatszeichen p im Körper negiert auftritt, müßte eine Stufenabbildung gefunden werden, so daß $|p|_{PZ} > |p|_{PZ}$ gilt. Das ist aber ein Widerspruch. $\mathcal{P}$ ist lokal stratifiziert, da eine Stufenabbildung über Grundatomen $|p(a)| = 1$ und $|p(b)| = 2$ gefunden werden kann, so daß die Bedingung aus Definition 2.44 erfüllt ist. ◦

Die Menge der lokal stratifizierten Programme ist eine Obermenge der Menge der stratifizierten Programme. Für lokal stratifizierte Programme kann ein kanonisches Modell definiert werden, indem aus der Menge der minimalen Fixpunkte, unter Verwendung der Stufenfunktion, ein Modell ausgewählt wird. Dieses Modell wird nach [Prz86] ein *perfektes Modell* genannt. Für die Definition dieses Begriffes benötigen wir einige Vorbereitungen.

Definition 2.47 *Sei $\mathcal{P}$ ein Programm. Eine* Prioritätsrelation *ist auf Elementen der Herbrand-Basis für $\mathcal{P}$ definiert. Für $A, B, C \in B_{\mathcal{P}}$ gilt:*

- *$C < B$ falls eine Grundinstanz einer Klausel in $\mathcal{P}$ mit einem Vorkommen von $\neg B$ im Körper und C im Kopf existiert.*
- *$C \leq A$ falls eine Grundinstanz einer Klausel in $\mathcal{P}$ mit einem Vorkommen von A im Kopf und C im Körper existiert.*
- *Sowohl $<$ als auch $\leq$ sind transitiv. Falls $A < B$, so gilt auch $A \leq B$.*

Definition 2.48 *Seien N und M zwei Modelle ($N \neq M$) für ein Programm $\mathcal{P}$. Das Modell N ist* besser als *M ($N \ll M$), falls für jedes Grundatom $A \in N \setminus M$ ein Grundatom $B \in M \setminus N$ existiert, so daß $A < B$ gilt.*

Definition 2.49 *Ein Modell M für ein Programm $\mathcal{P}$ ist* perfekt*, falls kein Modell für $\mathcal{P}$ existiert, das besser als M ist.*

Przymusinski zeigt die Existenz und Eindeutigkeit eines perfekten Modelles für die Klasse der lokal stratifizierten Programme.

Es existiert jedoch eine sehr große Klasse von Programmen, die nicht lokal stratifiziert sind.

Beispiel 2.50 Sei $\mathcal{P}$ das Programm:

$$\begin{array}{lcl} \texttt{e(s(X))} & \leftarrow & \neg\texttt{e(X)}. \\ \texttt{e(0)} & \leftarrow & \texttt{e(X)}. \end{array}$$

Versuchen wir, eine Stufenabbildung für $\mathcal{P}$ zu finden, liefert die erste Klausel

$$|e(s(0))| > |e(0)|$$

Die zweite Klausel liefert für $X = s(0)$

$$|e(0)| \geq |e(s(0))|$$

Damit muß gelten

$$|e(0)| \geq |e(s(0))| > |e(0)|$$

was zu einem Widerspruch führt. $\mathcal{P}$ ist also nicht lokal stratifiziert. ∘

In [GRS88] führen Van Gelder etal. ein anderes kanonisches Modell ein, das sie als *wohlfundierte Semantik* bezeichnen. Für lokal stratifizierte Programme stimmt dieses Modell mit dem perfekten Modell überein. Außerdem zeigen sie, daß es Programme gibt, die nicht lokal stratifiziert sind, aber ein wohlfundiertes Modell besitzen. Im weiteren beschäftigen wir uns nicht näher mit der wohlfundierten Semantik. Der Leser wird auf [GRS88] verwiesen. Die später verwendete *stabile Modellsemantik*[11] ist ohnehin allgemeiner als die wohlfundierte Semantik.

2.1.3 Metrische Methoden zur Modellauswahl

Mehrfach wurden auch metrische Methoden zur Auswahl eines kleinsten Modelles vorgeschlagen[12]. In Kapitel 3 beschäftigen wir uns mit einer von Fitting vorgeschlagenen metrischen Methode. Für die Klasse der *akzeptablen* Programme kann der Operator $\mathcal{T}_{\mathcal{P}}$ verwendet werden, um die Semantik des Programmes $\mathcal{P}$ zu ermitteln. Ein Fixpunkttheorem für Metriken sichert die Konvergenz von $\mathcal{T}_{\mathcal{P}}$.

[11] siehe [GL88]
[12] siehe [BS89]

2.2 Topologie metrischer Räume

Metrische Räume liefern ein alternatives Hilfsmittel für die Bestimmung der Semantik unterschiedlichster Probleme. Die Grundlage dafür ist die Existenz eines Fixpunkttheorems für Kontraktionen auf vollständigen metrischen Räumen. Ein Vorteil dieses Theorems liegt darin, daß ein *eindeutig bestimmter* Fixpunkt ermittelt wird. Diesen Ansatz nutzt Fitting zur Bestimmung der Semantik von Programmen. Dabei transformiert er Interpretationen und Stufenabbildungen für ein logisches Programm in einen metrischen Raum. Im folgenden sollen die wichtigsten Definitionen angeführt werden. Für eine umfassendere Darstellung sei auf [Smy92] verwiesen.

Definition 2.51 *Eine* Metrik (Abstandsfunktion) *über einer Menge $\mathcal{M}$ ist gegeben, wenn jedem Paar x, y von Elementen aus $\mathcal{M}$ eine reelle Zahl $d(x,y) \geq 0$ zugeordnet werden kann, so daß die folgenden Bedingungen erfüllt sind.*

$$\begin{array}{rcll} d(x,y) & = & 0 \quad \textit{gdw.} \quad x = y & \\ d(x,y) & = & d(y,x) & \\ d(x,y) & \leq & d(x,z) + d(z,y) & \quad \textit{(Dreiecksungleichung)} \end{array}$$

Definition 2.52 *Eine* Pseudometrik *über einer Menge $\mathcal{M}$ ist gegeben, wenn jedem Paar x, y von Elementen aus $\mathcal{M}$ eine reelle Zahl $d(x,y) \geq 0$ zugeordnet werden kann, so daß die folgenden Bedingungen erfüllt sind.*

$$\begin{array}{rcll} d(x,y) & = & d(y,x) & \\ d(x,y) & \leq & d(x,z) + d(z,y) & \quad \textit{(Dreiecksungleichung)} \end{array}$$

Definition 2.53 *Eine Menge $\mathcal{M}$ zusammen mit einer Metrik d über $\mathcal{M}$ heißt* metrischer Raum $(\mathcal{M}, d)$.

Beispiel 2.54 Die Menge der reellen Zahlen mit der Metrik

$$d(x,y) = |x - y|$$

bildet den metrischen Raum $\mathbb{R}^1 = (\mathbb{R}, d)$. ◦

Definition 2.55 *Eine Menge $\mathcal{M}$ zusammen mit einer Pseudo–Metrik d über $\mathcal{M}$ heißt* pseudo–metrischer Raum $(\mathcal{M}, d)$.

Im folgenden betrachten wir Folgen von Elementen eines (pseudo–)metrischen Raumes und definieren Eigenschaften dieser Folgen.

Definition 2.56 *Eine Folge von Elementen $(s_n)_{n \in \mathbb{N}}$ eines (pseudo–)metrischen Raumes $(\mathcal{M}, d)$ ist* Cauchy, *falls*

$$\forall \epsilon > 0 . \exists k . \forall m, n \geq k. \quad d(s_m, s_n) \leq \epsilon$$

gilt.

Das heißt, daß es für einen beliebig kleinen, aber positiven Abstand ϵ immer einen Index k gibt, für den alle nachfolgenden Elemente höchstens ϵ weit voneinander entfernt sind.

Definition 2.57 *Ein Element s ist* Grenzwert *einer Folge $(s_n)_{n \in \mathbb{N}}$, falls die Elemente der Folge gegen s konvergieren, das heißt, es gilt $s = \lim_{n \to \infty} s_n$, falls*

$$\forall \epsilon > 0.\, \exists k.\, \forall n \geq k. \quad d(s_n, s) \leq \epsilon$$

gilt.

Weniger formal können wir dafür schreiben $d(s_n, s) \to 0$ für $n \to \infty$.

Definition 2.58 *Ein metrischer Raum $(\mathcal{M}, d)$ ist* vollständig*, wenn jede Cauchy–Folge von Elementen aus $(\mathcal{M}, d)$ einen Grenzwert in $(\mathcal{M}, d)$ besitzt.*

Beispiel 2.59 Der in Beispiel 2.54 angegebene metrische Raum $\mathbb{R}^1$ ist vollständig, da der Grenzwert jeder Cauchy–Folge von reellen Zahlen, das heißt, jeder Folge, die konvergiert, wieder eine reelle Zahl ist. $\circ$

Definition 2.60 *Ein pseudo–metrischer Raum $(\mathcal{M}, d)$ ist* vollständig*, wenn für je zwei Grenzwerte s' und s'' einer Cauchy-Folge $(s_n)_{n \in \mathbb{N}}$ aus $(\mathcal{M}, d)$ die Beziehung $d(s', s'') = 0$ gilt.*

Bemerkung Für $d(s', s'') = 0$ kann $s' \neq s''$ gelten, da in einem pseudo–metrischen Raum die Bedingung $d(x, y) = 0$ gdw. $x = y$ nicht gelten muß.

Um mit pseudometrischen ebenso wie mit metrischen Räumen umgehen zu können, definieren wir eine Äquivalenzrelation wie folgt.

Definition 2.61 *Zwei Elemente x und y eines pseudometrischen Raumes $(\mathcal{M}, d)$ bezeichnen wir als* äquivalent *$(x \sim y)$, falls $d(x, y) = 0$.*

Eine Einteilung des pseudometrischen Raumes $(\mathcal{M}, d)$ in Äquivalenzklassen bezüglich der angegebenen Äquivalenzrelation liefert eine Metrik $\bar{d}$ auf dem Raum der Äquivalenzklassen. Bezeichne $[x]$ die Klasse mit dem Repräsentanten x. Dann ist die Abbildung $\bar{d}$ definiert durch $\bar{d}([x], [y]) = d(x, y)$. Mit $\bar{\mathcal{M}} = \mathcal{M}/\sim$ erhalten wir einen metrischen Raum $(\bar{\mathcal{M}}, \bar{d})$.

Beispiel 2.62 Sei $\mathcal{P}$ das Programm:

$$\begin{array}{lll} \texttt{win(X)} & \leftarrow & \texttt{move(X,Y), } \neg\texttt{, win(Y).} \\ \texttt{move(a,b).} & & \forall (a, b) \in \mathcal{G} \end{array}$$

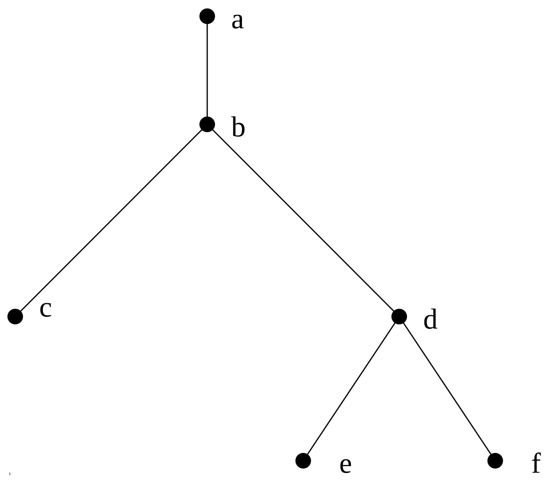

Abbildung 2.1: Ein Graph $\mathcal{G}$ für das Spielprogramm $\mathcal{P}$. Die Knoten repräsentieren Zustände in einem Spiel. Die Kanten sind, ausgehend von der Wurzel des Graphen, in Richtung der Blattknoten gerichtet und repräsentieren mögliche Zustandsübergänge die von den Spielern realisiert werden können. Gelangt ein Spieler in einen Zustand, der einem Wurzelknoten entspricht, so kann er im nächsten Schritt keine Bewegung mehr ausführen und hat das Spiel verloren.

Abbildung 2.1 zeigt einen konkreten Graph $\mathcal{G}$. Eine Stufenabbildung[13] $|\_|$ definieren wir wie folgt: $|win(p)|$ entspricht der Höhe des Teilbaumes von $\mathcal{G}$ mit der Wurzel p. Also es gilt: $|win(a)| = 1$, $|win(b)| = 2$, $|win(c)| = |win(d)| = 3$ und $|win(e)| = |win(f)| = 4$. Seien I_1 und I_2 zwei Interpretationen. Eine Metrik d definieren wir wie folgt

$$d(I_1, I_2) = \begin{cases} 0 & \text{falls } I_1 = I_2 \\ (\frac{1}{2})^n & \text{falls sich } I_1 \text{ und } I_2 \text{ in der Interpretation eines beliebigen Grunda-} \\ & \text{tomes } A \text{ mit Stufe } n \text{ unterscheiden, jedoch in der Interpretation} \\ & \text{aller Grundatomen mit niedrigerer Stufe übereinstimmen} \end{cases}$$

Betrachtet man zum Beispiel die Interpretationen $I_1 = \{win(a), win(c), win(e)\}$ und $I_2 = \{win(a), win(d)\}$, stimmen I_1 und I_2 in der Interpretation aller Grundatome mit Stufen 1 und 2 ($win(a), win(b)$) überein, unterscheiden sich jedoch in der Interpretation von $win(c)$. Daraus ermitteln wir $d(I_1, I_2) = (\frac{1}{2})^3$.

Es soll gezeigt werden, daß d eine Metrik auf dem Raum von Interpretationen für $\mathcal{P}$ ist. Die Bedingungen $d(x, y) = 0$ gdw. $x = y$ und $d(x, y) = d(y, x)$ sind laut Definition erfüllt. Zu zeigen ist, daß die Dreiecksungleichung $d(x, y) \leq d(x, z) + d(z, y)$ gilt. Seien x, y und z Interpretationen für $\mathcal{P}$. Falls $x = y = z$, ist die Bedingung erfüllt. Anderenfalls

[13] siehe Definition 2.40

können wir o.B.d.A. annehmen, daß $d(x,y) = (\frac{1}{2})^n$, $d(x,z) = (\frac{1}{2})^m$ und $d(z,y) = (\frac{1}{2})^k$ gilt. Dann unterscheiden sich x und z in einem Grundatom mit Stufe m, stimmen aber in allen Grundatomen mit einer Stufe kleiner als m überein. Analog stimmen z und y in allen Grundatomen mit einer Stufe kleiner als k überein. O.B.d.A. nehmen wir $m \leq k$ an. Dann müssen x und y in allen Grundatomen mit einer Stufe kleiner als m übereinstimmen und es gilt $d(x,y) \leq (\frac{1}{2})^m$. Damit folgern wir $d(x,y) \leq (\frac{1}{2})^m + (\frac{1}{2})^k = d(x,z) + d(z,y)$.

∘

Definition 2.63 *Sei $(\mathcal{M}, d)$ ein metrischer Raum. Eine Abbildung $f : \mathcal{M} \to \mathcal{M}$ ist eine* Kontraktion, *falls es eine Zahl $0 < k < 1$ gibt, so daß für alle $x, y \in \mathcal{M}$:*

$$d(f(x), f(y)) \leq k \cdot d(x,y)$$

Theorem 2.64 (Banachs–Kontraktions–Theorem)
Eine Kontraktion f auf einem vollständigen metrischen Raum hat genau einen Fixpunkt. Die Folge $x, f(x), f(f(x)), \ldots$ konvergiert für ein beliebiges x gegen diesen Fixpunkt.

Der Beweis kann zum Beispiel in [Wil70] nachgeschlagen werden. Eine Anwendung des Kontraktionstheorems und ein Beispiel findet man im Kapitel 3.

2.3 Künstliche Neuronale Netze

Definition 2.65 *Eine* Netzarchitektur *ist ein Tupel (E, N, A, T) mit*

E	*Menge von n_e Eingabeeinheiten $\{e_1, \ldots, e_{n_e}\}$*
N	*Menge von n verborgenen Einheiten*
A	*Menge von n_a Ausgabeeinheiten $\{a_1, \ldots, a_{n_a}\}$*
T	*Menge gewichteter und gerichteter Kanten.*

Eine gewichtete und gerichtete Kante ist ein Tupel (k_j, k_k, w_{kj}) mit $k_j \in E \cup N$, $k_k \in N \cup A$, $w_{kj} \in \mathbb{R}$. Das Tupel (k_j, k_k, w_{kj}) bezeichnet eine Kante von der Einheit j zur Einheit k mit dem Gewicht w_{kj}.

Jede Einheit in der Netzarchitektur ist gekennzeichnet durch ihr *Potential* $p \in \mathbb{R}$, ihren *Wert* v und ihren *Eingabevekor* $\vec{i} = (i_1, \ldots, i_n)$. Dabei beschreibt $i_j = w_{kj} v_j$ die Eingabe, die die Einheit k von der Einheit j erhält.

Jede Einheit berechnet die Funktionskomposition $\psi\phi : \mathbb{R}^n \to \mathbb{R}$, wobei $\psi : \mathbb{R}^n \to \mathbb{R}$ als Aktivierungsfunktion und $\phi : \mathbb{R} \to \mathbb{R}$ als Ausgabefunktion bezeichnet wird.

Als Aktivierungsfunktionen werden häufig *sigmoide* Funktionen verwendet. Das sind Funktionen, die im gesamtem Definitionsbereich differenzierbar sind und abhängig von einem Wert ϵ für $\epsilon \to \infty$ gegen die Schwellwert-Funktion

$$f(x) = \begin{cases} 1 & \text{falls } x \geq \theta \\ 0 & \text{sonst} \end{cases}$$

konvergieren, wobei θ für einen beliebigen Schwellwert steht.

Beispiel 2.66 Die Funktion $s_\epsilon : \mathbb{R} \to (0,1)$ mit

$$s_\epsilon(x) = \frac{1}{1 + xe^{-\epsilon x}}$$

liefert für $\epsilon \to \infty$ die oben angegebene Funktion f für den Schwellwert $\theta = 0$. $\circ$

Betrachten wir nur diskrete Werte, heißt eine Einheit E_k mit dem Schwellwert θ_k *binäre Schwellwerteinheit*, falls ihr Potential p_k und ihr Wert v_k nach der folgenden Aktivierungs– und Ausgabefunktion berechnet werden:

$$p_k = \psi(\vec{i}) = \sum_j w_{kj} v_j \quad v_j \in \{0, 1\}$$

$$v_k = \phi(p_k) = \begin{cases} 1 & \text{falls } p_k \geq \theta_k \\ 0 & \text{sonst} \end{cases}$$

wobei j alle Indizes von Einheiten umfaßt, die Ausgangspunkt einer Kante zur Einheit E_k sind.

Eine *Schichtenarchitektur* liegt vor, falls die Menge N von verborgenen Einheiten in Gruppen $N_1, \ldots, N_l$ aufgeteilt werden kann, so daß nur Verbindungen $(e_i \in E) \to (E_1 \in N_1), (E_1 \in N_1) \to (E_2 \in N_2), \ldots, (E_l \in N_l) \to (a_i \in A)$ zugelassen sind, das heißt, von den Einheiten jeder Schicht sind nur Verbindungen zu den Einheiten der in Richtung der Ausgabeschicht direkt folgenden Schicht zugelassen. Die Gruppen bezeichnen wir als Schichten. Die Menge der Einheiten in der Schicht E bezeichnen wir als Eingabeschicht. Die Menge der Einheiten in der Schicht A bezeichnen wir als Ausgabeschicht. Die Gruppen $N_1, \ldots, N_l$ bezeichnen wir als *verborgene oder interne Schichten.* Die Einheiten innerhalb einer Schicht sind nicht miteinander verbunden.

Definition 2.67 *Ein n–*schichtiges vorwärtsgerichtetes Netz $\mathcal{F}$ *besteht aus einer Eingabeschicht, $n-2$ verborgenen Schichten und einer Ausgabeschicht ($n \geq 2$). Jede Einheit in der i–ten Schicht ist mit jeder Einheit in der $(i+1)$–ten Schicht verbunden.*

Ein *mehrschichtiges vorwärtsgerichtetes Netz mit binären Schwellwerteinheiten* berechnet die Funktion $f_\mathcal{F} : \{0,1\}^{n_e} \to \{0,1\}^{n_a}$, indem der Eingabevektor $\vec{i}$ zum Zeitpunkt t_1 an der Eingabeschicht angelegt und durch die verborgenen Schichten zur Ausgabeschicht propagiert wird. Zu jedem Zeitpunkt $t_j (1 \leq j \leq m)$ werden das Potential p_k und der Wert v_k einer jeden Einheit E_k des Netzes synchron aktualisiert. Abhängig von der Anzahl m der Schichten wird der Ausgabevektor $\vec{a}$ zum Zeitpunkt t_m an der Ausgabeschicht abgelesen.

Abbildung 2.2: Skizze eines 3–schichtigen rekurrenten Netzes. Jede Einheit in der Eingabeschicht ist mit jeder Einheit in der verborgenen Schicht und jede Einheit in der verborgenen Schicht ist mit jeder Einheit in der Ausgabeschicht verbunden. Die Ein– und die Ausgabeschicht sind identisch. Jede Einheit in der Ausgabeschicht ist mit der zu ihr identischen Einheit in der Eingabeschicht verbunden.

Definition 2.68 *Ein n–*schichtiges rekurrentes Netz *$\mathcal{N}$ besteht aus einem n–schichtigen vorwärtsgerichteten Netz mit derselben Anzahl von Einheiten in der Ein– und Ausgabeschicht. Jede Einheit an der k–ten Position in der Ausgabeschicht wird mit der k–ten Einheit in der Eingabeschicht durch eine mit dem Gewicht 1 bewertete Kante verbunden, wobei $1 \leq k \leq m$ und $m = n_e = n_a$ ist die Anzahl der Einheiten in der Ein– bzw. Ausgabeschicht.*

Abbildung 2.2 zeigt ein 3–schichtiges rekurrentes Netz.

Definition 2.69 *Ein* einfaches Perzeptron *ist ein 2–schichtiges vorwärtsgerichtetes Netz mit n binären Schwellwerteinheiten in der Eingabeschicht mit Eingabewerten x_1, x_2, $\ldots, x_n$ ($x_i \in \{0, 1\}$) und einer Einheit in der Ausgabeschicht. Die Kanten von den Einheiten $x_1, x_2, \ldots, x_n$ in der Eingabeschicht zur Einheit in der Ausgabeschicht sind mit den reellen Gewichten $w_1, w_2, \ldots, w_n$ bewertet. Für einen reellen Schwellwert θ*

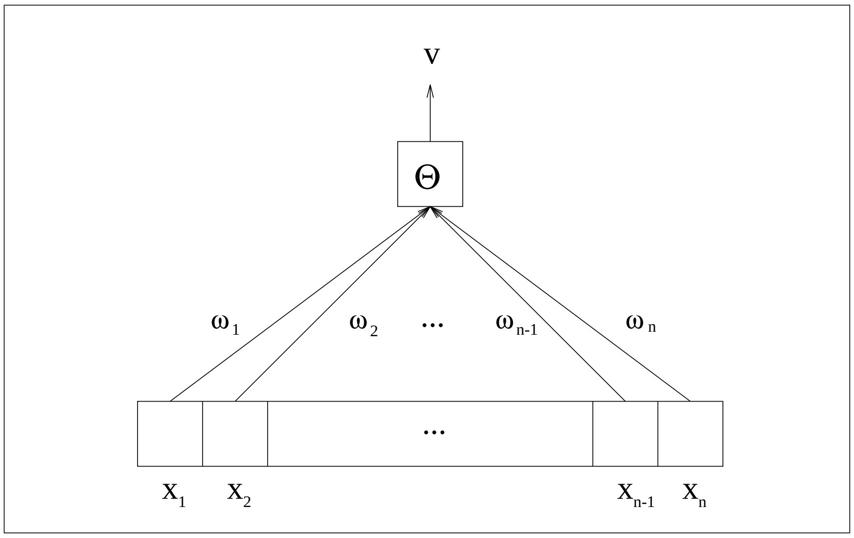

Abbildung 2.3: Ein einfaches Perzeptron

der Einheit in der Ausgabeschicht gibt das Perzeptron nur dann eine Eins aus, wenn $\sum_{i=1}^{n} w_i x_i \geq \theta$ *gilt. Anderenfalls wird eine Null ausgegeben.*

Abbildung 2.3 zeigt ein einfaches Perzeptron.

Definition 2.70 *Ein Perzeptron hat im Vergleich zu einem einfachen Perzeptron* $n \geq 1$ *Einheiten in der Ausgabeschicht, die von beliebig vielen Einheiten in der Eingabeschicht Eingaben erhalten. Dabei arbeitet jede der Einheiten in der Ausgabeschicht wie ein einfaches Perzeptron.*

Ein Perzeptron kann nur linear separable Funktionen berechnen.

Definition 2.71 *Zwei Mengen* A *und* B *von Punkten im* n*–dimensionalen Raum sind* linear separabel, *falls reelle Zahlen* $w_1, \ldots, w_n$ *und* γ *existieren, so daß für jeden Punkt* $(x_1, \ldots, x_n) \in A$

$$\sum_{i=1}^{n} w_i x_i \geq \gamma$$

und für jeden Punkt $(x_1, \ldots, x_n) \in B$

$$\sum_{i=1}^{n} w_i x_i < \gamma$$

gilt.

Abbildung 2.4: Abbildung (a) zeigt zwei linear separable Mengen A und B. Abbildung (b) zeigt zwei Mengen A und B, die nicht linear separiert werden können.

Definition 2.72 *Eine binäre Funktion* $f : \mathbb{R}^n \to \{0, 1\}$ *heißt* linear separabel, *falls die Menge der Punkte, für die* $f(x_1, \ldots, x_n) = 1$ *gilt, von der Menge der Punkte, für die* $f(x_1, \ldots, x_n) = 0$ *gilt, linear separiert werden kann.*

Anschaulich bedeutet lineare Separierbarkeit im 2–dimensionalen Fall, daß eine Gerade gefunden werden kann, die zwei Punktmengen in der Ebene voneinander trennt. Abbildung 2.4 (a) zeigt einen solchen Fall. Im n–dimensionalen Fall muß eine $(n-1)$-dimensionale Hyperebene gefunden werden, die zwei Punkträume voneinander trennt.

Proposition 2.73 [MP72] *Ein Perzeptron kann nur linear separable Funktionen berechnen.*

Bemerkung Um Schreibaufwand zu vermeiden werden die folgenden Vereinbarungen betreffs der Abbildung von Netzen getroffen.

- Werte in den Einheiten bezeichnen den Schwellwert der Einheit
- Werte an den Einheiten bezeichnen den Eingabewert beziehungsweise stehen für einen Bezeichner, mit dem die Einheit versehen wird
- Werte an den Kanten bezeichnen die Gewichte der Kanten
- Aus Gründen der Übersichtlichkeit werden nur Kanten mit Gewichten verschieden von Null dargestellt. Alle nicht gezeigten Kanten sind mit Null bewertet.

3. Eine metrische Methode

In [Fit93] schlägt Fitting vor, metrische Methoden zur Berechnung der Semantik zu verwenden[1]. Die Idee besteht darin, einem Programm $\mathcal{P}$ einen vollständigen metrischen Raum $(\mathcal{M}_{\mathcal{P}}, d_{\mathcal{P}})$ zuzuordnen, so daß $\mathcal{T}_{\mathcal{P}}$ die Eigenschaft einer Kontraktion in $(\mathcal{M}_{\mathcal{P}}, d_{\mathcal{P}})$ besitzt. Das Banachsche Fixpunkttheorem für Kontraktionen in vollständigen metrischen Räumen sichert dann die Existenz und Eindeutigkeit eines Fixpunktes für $\mathcal{T}_{\mathcal{P}}$. Ein großer Vorteil dieser Methode liegt darin, daß der Fixpunkt unabhängig von einer Anfangsinterpretation ermittelt werden kann.

3.1 Interpretationen und Metriken

Fitting führt eine Metrik auf der Menge von Interpretationen für ein Programm $\mathcal{P}$ ein. Dabei wird eine Stufenabbildung $|\_|$ genutzt.

Definition 3.1 *Sei $|\_|$ eine Stufenabbildung, I_1 und I_2 seien Interpretationen. Eine Abbildung d definieren wir wie folgt.*

$$d(I_1, I_2) = \begin{cases} 0 & \text{falls } I_1 = I_2 \\ (\frac{1}{2})^n & \text{falls sich } I_1 \text{ und } I_2 \text{ in der Interpretation eines beliebigen Grundatomes } A \text{ mit Stufe } n \text{ unterscheiden, jedoch in der Interpretation aller Grundatome mit niedrigerer Stufe übereinstimmen} \end{cases}$$

Satz 3.2 [Fit93]
Die Abbildung d ist eine Metrik.

Definition 3.3 *Sei $|\_|$ eine partielle Stufenabbildung, I_1 und I_2 seien Interpretationen. Eine Abbildung d_{part} definieren wir wie folgt.*

$$d_{part}(I_1, I_2) = \begin{cases} 0 & \text{falls } I_1 \text{ und } I_2 \text{ im Wertebereich von } |\_| \text{ übereinstimmen} \\ (\frac{1}{2})^n & \text{falls sich } I_1 \text{ und } I_2 \text{ in der Interpretation eines beliebigen Grundatomes } A \text{ mit Stufe } n \text{ unterscheiden, jedoch in der Interpretation aller Grundatome mit niedrigerer Stufe übereinstimmen} \end{cases}$$

[1] siehe Abschnitt 2.1.3

Satz 3.4 [Fit93]
Die Abbildung d_{part} ist eine Pseudo–Metrik.

Abhängig von der Stufenabbildung können auf den Interpretationen für ein Programm $\mathcal{P}$ durch obige Definition unterschiedliche Metriken bestimmt werden.

Proposition 3.5 [Fit93]
Die Menge von Interpretationen für ein Programm $\mathcal{P}$, bezeichnet mit $\mathcal{M}_{\mathcal{P}}$, und eine, auf einer Stufenabbildung für das Programm basierende Metrik $d_{\mathcal{P}}$ bilden einen vollständigen metrischen Raum $(\mathcal{M}_{\mathcal{P}}, d_{\mathcal{P}})$.

3.2 Semantik normaler logischer Programme

Die Semantik eines definiten Programms $\mathcal{P}$ kann mit Hilfe des Operators $\mathcal{T}_{\mathcal{P}}$ ermittelt werden[2]. Auch die von Fitting vorgestellte metrische Methode verwendet die Abbildung $\mathcal{T}_{\mathcal{P}}$. Falls man zeigen kann, daß $\mathcal{T}_{\mathcal{P}}$ eine Kontraktion[3] auf dem vollständigen metrischen Raum der Herbrand–Interpretationen für ein normales Programm $\mathcal{P}$ ist, kann man $\mathcal{T}_{\mathcal{P}}$ auch zur Bestimmung der Semantik dieses normalen Programmes verwenden.

Intuitiv soll die Metrik über den Interpretationen den folgenden Sachverhalt widerspiegeln: Eine Interpretation I_1 unterscheidet sich stark von einer Interpretation I_2, wenn sie Grundatome mit niedriger Stufe unterschiedlich bewertet. Dann soll auch der metrische Abstand groß sein. Eine Kontraktion verringert den Abstand zwischen zwei Interpretationen, das heißt, zwei Interpretationen unterscheiden sich nach Anwendung der Kontraktion in der Interpretation von Grundatomen einer höheren Stufe als vorher. Die Iteration einer Kontraktion, angewendet auf zwei Interpretationen I_1 und I_2, verringert also sukzessive den Abstand zwischen I_1 und I_2. Mit dem Banachschen Fixpunkttheorem kann man zeigen, daß dieser Abstand gegen Null konvergiert, das heißt, in einem vollständigen metrischen Raum konvergieren die Folgen $I_1, f(I_1), f(f(I_1)), \ldots$ und $I_2, f(I_2), f(f(I_2)), \ldots$ gegen einen gemeinsamen Grenzwert.

Falls nun der Operator $\mathcal{T}_{\mathcal{P}}$ eine Kontraktion auf dem vollständigen metrischen Raum der Herbrand–Interpretationen für das Programm $\mathcal{P}$ ist, hat $\mathcal{T}_{\mathcal{P}}$ einen eindeutig bestimmten Fixpunkt. Man erhält eine Berechnungsvorschrift, die unabhängig vom Anfangswert der Berechnung einen eindeutig bestimmten Fixpunkt von $\mathcal{T}_{\mathcal{P}}$ liefert.

Damit reduziert sich die Suche nach einem Fixpunkt des Operators $\mathcal{T}_{\mathcal{P}}$ auf das Auffinden einer Metrik auf der Menge von Interpretationen, die $\mathcal{T}_{\mathcal{P}}$ zu einer Kontraktion macht. Unter Nutzung von Definition 3.1 reduziert sich das Auffinden einer solchen Metrik auf das Auffinden einer Stufenabbildung $|\ \_\ |$, so daß $\mathcal{T}_{\mathcal{P}}$ eine Kontraktion auf dem vollständigen metrischen Raum $(\mathcal{M}_{\mathcal{P}}, d_{\mathcal{P}})$ ist.

[2] siehe Definition 2.25
[3] siehe Definition 2.63

Bisher haben wir nur zweiwertige Interpretationen betrachtet. Man kann jedoch zeigen, daß alle in diesem Abschnitt getroffenen Aussagen analog für dreiwertige Interpretationen gelten. Den auf der Menge der dreiwertigen Interpretationen definierten Operator bezeichnen wir mit $\Phi_{\mathcal{P}}$[4]. In vielen Fällen kann man eine Stufenabbildung finden, so daß $\Phi_{\mathcal{P}}$ Kontraktion auf dem vollständigen metrischen Raum der dreiwertigen Interpretationen für $\mathcal{P}$ ist. Entsprechend hat dann auch $\Phi_{\mathcal{P}}$ einen eindeutig bestimmten Fixpunkt. Falls für ein Programm $\mathcal{P}$ jeder Fixpunkt von $\mathcal{T}_{\mathcal{P}}$ auch Fixpunkt für $\Phi_{\mathcal{P}}$ ist und umgekehrt, stimmen die Semantik für ein Programm $\mathcal{P}$ in der zweiwertigen und in der dreiwertigen Logik überein. Dieses Problem wird in Abschnitt 3.4 betrachtet.

Der dreiwertige Operator $\Phi_{\mathcal{P}}$ für ein normales Programm $\mathcal{P}$ ist monoton auf der Ordnung der dreiwertigen Interpretationen für $\mathcal{P}$. Aufgrund dieser Eigenschaft kann man zeigen, daß $\Phi_{\mathcal{P}}$ gegen einen Fixpunkt konvergiert, falls man die Iteration mit dem kleinsten Element $(\emptyset, \emptyset)$ startet. Im allgemeinen benötigt $\Phi_{\mathcal{P}}$ jedoch mehr als ω Schritte, um seinen Fixpunkt zu erreichen. Mit der oben angegebenen Methode kann gesichert werden, daß $\Phi_{\mathcal{P}}$ in ω Schritten konvergiert und die Iteration kann mit einer beliebigen Anfangsinterpretation starten.

3.3 Ein Beispiel

Wir betrachten das Programm $\mathcal{P}$, daß sich aus $\mathcal{P}_1$ und $\mathcal{P}_2$ zusammensetzt.

$\mathcal{P}_1$:
```
diff([E|Liste1],Liste2,[E|Liste])  ←  ¬ member(E,Liste2),
                                       diff(Liste1,Liste2,Liste).
diff([E|Liste1],Liste2,Liste)      ←  member(E,Liste2),
                                       diff(Liste1,Liste2,Liste).
```

$\mathcal{P}_2$:
```
member(E,[E|Liste]).
member(E,[X|Liste]) ← member(E,Liste).
element(a). ∀a ∈ A
```

Wir betrachten Listen mit Elementen, die wir mit den Konstanten $a, b, \ldots$ bezeichnen, wobei $\mathcal{A}$ die Menge der genutzten Bezeichner umfaßt. Eine Ziel $\leftarrow$ `diff(X,Y,Z)` soll erfolgreich sein, falls `Z` die Differenz der Listen `X` und `Y` repräsentiert. Das ist der Fall, falls `Z` alle Elemente aus `X` umfaßt, die nicht in `Y` vorkommen. Die Klausel `element(a)` sichert, daß alle von uns verwendeten Bezeichner in der Herbrand–Basis von $\mathcal{P}$ vorkommen.

In $\mathcal{P}$ treten drei Prädikatszeichen auf. Dabei ist das Prädikat `element` nicht problematisch. Alle Instanzen von `element` sind Fakten, so daß wir bei der Iteration von $\mathcal{T}_{\mathcal{P}}$, falls wir mit unterschiedlichen Interpretationen starten, nach einem Zyklus eine Interpretation erhalten, die alle Instanzen von `element` liefert. Folglich können wir das Verhalten von Interpretationen bezüglich des Prädikates `element` vernachlässigen. Wir betrachten

[4] siehe Kapitel 6

also im folgenden die Prädikate `diff` und `member`, wobei `diff` von `member` abhängt. Um diese Abhängigkeit behandeln zu können, teilen wir die Spezifikation einer Metrik in zwei Teile, so daß jeder Teil eines der Programme $\mathcal{P}_1$ beziehungsweise $\mathcal{P}_2$ behandelt.

Da `member` auch unabhängig von `diff` existiert, diskutieren wir zuerst das Verhalten von `member`.

Definition 3.6 *Als* Listen *bezeichnen wir Grundterme der Form* [] *und* $[a|b]$.
Wir definieren den Begriff Länge *wie folgt.*

$$Länge(c) = \begin{cases} 0 & falls\ c = [] \\ 1 + \ Länge(b) & falls\ c = [a|b] \end{cases}$$

Wir definieren den Begriff Element *einer Liste wie folgt. Falls $c = []$, dann enthält c keine Elemente. Die Elemente von $[a|b]$ sind a und die Elemente der Liste b.*

Wir definieren nun eine Stufenabbildung $|\_|$ auf Grundatomen der Relation `member`. Da $|\_|$ nur auf einer Teilmenge der Herbrand–Basis von $\mathcal{P}$ definiert ist, erhalten wir eine partielle Stufenabbildung.

Definition 3.7 *Eine partielle Stufenabbildung $|\_|$ auf Grundatomen von* `member` *definieren wir wie folgt:*

$$|\_| \quad : \quad \texttt{member(x,y)} \mapsto Länge(\texttt{y})$$

Die Abbildung $|\_|$ ist nicht auf Grundatomen der Form `diff(X,Y,Z)` *definiert.*
Die Abstandsfunktion d_1 definieren wir als die durch diese partielle Stufenabbildung induzierte Pseudo–Metrik.

Die Abstandsfunktion d_1 ist eine Pseudo–Metrik.

Um das Banachsche Kontraktionstheorem anwenden zu können, sind wir an einem *metrischen* Raum interessiert. Eine Faktorisierung des pseudometrischen Raumes bezüglich der Äquivalenzrelation $\sim$ liefert einen metrischen Raum $(\bar{\mathcal{M}}, \bar{d})$[5].

Nun wollen wir die Semantik von `member` bestimmen. `member(x,y)` gilt genau dann, wenn `y` eine Liste ist und `x` eines ihrer Elemente.

Definition 3.8 *Sei I eine Interpretation, so daß $I($*`member(x,y)`*$)$ = wahr gdw.* `y` *eine Liste und* `x` *ein Element von* `y`. *Auf Grundatomen der Form* `diff(x,y,z)` *ist I beliebig. Seien I_1 und I_2 Interpretationen. Wir definieren eine Abstandsfunktion d_2 als*

$$d_2(I_1, I_2) = \max\{d_1(I_1, I), d_1(I_2, I)\}.$$

[5] siehe Seite 21

Satz 3.9 *Die Abstandsfunktion d_2 ist eine Pseudo–Metrik.*

Beweis Seien I_1, I_2 und I_3 beliebige Interpretationen. $d_2(I_1, I_2) = d_2(I_2, I_1)$ gilt laut Definition. Zu zeigen ist, daß die Dreiecksungleichung $d_2(I_1, I_2) \leq d_2(I_1, I_3) + d_2(I_3, I_2)$ gilt.

$$\begin{aligned} d_2(I_1, I_2) &= \max\{d_1(I_1, I), d_1(I_2, I)\} \\ &\leq \max\{d_1(I_1, I), d_1(I_2, I), d_1(I_3, I)\} \\ &\leq \max\{d_1(I_1, I), d_1(I_3, I)\} + \max\{d_1(I_3, I), d_1(I_2, I)\} \\ &= d_2(I_1, I_3) + d_2(I_3, I_2) \end{aligned}$$

■

Wir betrachten nun das Prädikat `diff`.

Definition 3.10 *Sei* `diff(x,y,z)` *grundinstantiiert. Wir definieren eine partielle Stufenabbildung* $||\_||$ *mit:*

$$||\_|| \quad : \quad \texttt{diff(x,y,z)} \mapsto \mathit{Länge}(\texttt{x}) + \mathit{Länge}(\texttt{y}) + \mathit{Länge}(\texttt{z})$$

Auf Grundatomen der Form `member(x,y)` *ist* $||\_||$ *nicht definiert. Die Pseudo–Metrik* d_3 *ist die durch die partielle Stufenabbildung* $||\_||$ *induzierte Abstandsfunktion.*

Abschließend definieren wir eine Abbildung d, die die beiden Pseudo–Metriken d_2 und d_3 kombiniert.

Definition 3.11 *Für zwei Interpretationen I_1 und I_2, setze*

$$d(I_1, I_2) = \max\{d_2(I_1, I_2), d_3(I_1, I_2)\}.$$

Die Abbildung d ist eine Metrik und liefert zusammen mit der Menge aller Herbrand–Interpretationen für unser Programm $\mathcal{P}$ einen vollständigen metrischen Raum.

Proposition 3.12 *Die Abbildung $\mathcal{T}_\mathcal{P}$ ist eine Kontraktion.*

Beweis Seien I_1 und I_2 Interpretationen. Wir wollen zeigen, daß $\mathcal{T}_\mathcal{P}$ Kontraktion ist, also

$$d(\mathcal{T}_\mathcal{P}(I_1), \mathcal{T}_\mathcal{P}(I_2)) \leq k \cdot d(I_1, I_2)$$

mit $0 < k \leq 1$. Für $d(I_1, I_2) = 0$ gilt $I_1 = I_2$ und mit $\mathcal{T}_P(I_1) = \mathcal{T}_P(I_2)$ gilt:

$$d(\mathcal{T}_P(I_1), \mathcal{T}_P(I_2)) = 0$$

Wir nehmen $d(I_1, I_2) = (\frac{1}{2})^n$ an.

- Mit $d(I_1, I_2) = (\frac{1}{2})^n$ gilt $d_2(I_1, I_2) \leq (\frac{1}{2})^n$ und damit $d_1(I_1, I) \leq (\frac{1}{2})^n$, d.h. I_1 und I stimmen in allen Grundatomen `member(a,l)` überein, für die $|$`member(a,l)`$| < n$ gilt, das heißt, `l` ist eine Liste mit Länge(`l`)$< n$.

- Die Anwendung von $\mathcal{T}_\mathcal{P}$ verlängert die Listen in den Grundatomen von `member` um 1 Element. Daraus folgt, daß $\mathcal{T}_\mathcal{P}(I_1)$ und $\mathcal{T}_\mathcal{P}(I)$ in allen Grundatomen übereinstimmen, die Listen der Länge kleiner als $n+1$ beinhalten. Also ist $d_1(\mathcal{T}_P(I_1), \mathcal{T}_P(I)) \leq (\frac{1}{2})^{n+1}$.

- Analog muß $d_1(\mathcal{T}_\mathcal{P}(I_2), I) \leq (\frac{1}{2})^{n+1}$ gelten.

- Mit unserer Annahme $d(I_1, I_2) = (\frac{1}{2})^n$ gilt dann

$$d_1(\mathcal{T}_\mathcal{P}(I_1), I) \leq \frac{1}{2} \cdot d(I_1, I_2)$$

  Und analog für I_2

$$d_1(\mathcal{T}_\mathcal{P}(I_2), I) \leq \frac{1}{2} \cdot d(I_1, I_2)$$

  woraus folgt

$$d_2(\mathcal{T}_\mathcal{P}(I_1), \mathcal{T}_\mathcal{P}(I_2)) \leq \frac{1}{2} \cdot d(I_1, I_2) \tag{3.1}$$

- Wir betrachten jetzt `diff` und nehmen ebenfalls $d(I_1, I_2) = (\frac{1}{2})^n$ an.
  Sei `diff(x,y,z)` grundinstantiiert und $||$`diff(x,y,z)`$|| < n$. Wir wollen zeigen, daß falls $\mathcal{T}_\mathcal{P}(I_1)$ `diff(x,y,z)` als *wahr* interpretiert auch $\mathcal{T}_\mathcal{P}(I_2)$ `diff(x,y,z)` als *wahr* interpretiert. Wir nehmen an, daß $\mathcal{T}_\mathcal{P}(I_1)$ `diff(x,y,z)` als *wahr* interpretiert.

- Mit unserer Annahme $d(I_1, I_2) = (\frac{1}{2})^n$ gilt $d(I_1, I) \leq (\frac{1}{2})^n$ und $d(I_2, I) \leq (\frac{1}{2})^n$, das heißt, I_1, I_2 und I stimmen in allen Grundatomen `member(a,l)` überein, in denen `l` eine Liste der Länge kleiner als n ist. Falls also `l` eine Liste mit Länge(`l`) $< n$ ist, gilt $I_1($`member`$(a, l)) = I_2($`member(a,l)`$) = I$ und $I = wahr$, gdw. `a` ein Element von `l` ist.

- Wenn also $\mathcal{T}_\mathcal{P}(I_1)$ `diff(x,y,z)` als *wahr* interpretiert wird, so weist eine der Klauseln aus $\mathcal{P}_1$ `diff(x,y,z)` den Wert *wahr* zu.

  1. Die erste Klausel von $\mathcal{P}_1$ weist `diff(x,y,z)` den Wert *wahr* zu.
     - Dann muß es eine Grundinstanz dieser Klausel geben wie folgt:

       `diff(x,y,z)` $\leftarrow \neg$`member(e,y),diff(x\e,y,z\e).`

       und $\neg$ `member(e,y)` und `diff(x\e,y,z\e)` werden von $\mathcal{T}_\mathcal{P}(I_1)$ als *wahr* interpretiert.

– `x\e` steht für die Liste `x` ohne das Element `e`. Die Länge der Liste `x\e` bestimmen wir wie folgt.

$$\text{Länge}(\texttt{x\textbackslash e}) = \begin{cases} \text{Länge}(\texttt{x}) & \text{falls } \texttt{e} \text{ nicht Element von } \texttt{x} \\ \text{Länge}(\texttt{x}) - 1 & \text{sonst} \end{cases}$$

– Es gilt $n > ||\texttt{diff(x,y,z)}||$ und

$$||\texttt{diff(x,y,z)}|| \geq \text{Länge}(\texttt{x\textbackslash e}) + \text{Länge }(\texttt{y}) + \text{Länge}(\texttt{z\textbackslash e})$$

Aus

$$I_1(\neg\texttt{member(e,y)}) = \mathit{wahr}$$

folgt, daß `e` kein Element der Liste `y` ist. Damit gilt

$$I_2(\neg\texttt{member(e,y)}) = \mathit{wahr}$$

und

$$\begin{aligned} ||\texttt{diff(x\textbackslash e,y,z\textbackslash e)}|| &= \text{Länge}(\texttt{x\textbackslash e}) + \text{Länge }(\texttt{y}) + \text{Länge}(\texttt{z\textbackslash e}) \\ &\leq ||\texttt{diff(x,y,z)}|| \\ &< n \end{aligned}$$

– Mit $d_3(I_1, I_2) \leq (\frac{1}{2})^n$ gilt nun, daß I_1 und I_2 in der Interpretation von `diff(x\e,y,z\e)` übereinstimmen und damit $I_2(\texttt{diff(x\textbackslash e,y,z\textbackslash e)}) = \mathit{wahr}$. Unter Anwendung der ersten Klausel von $\mathcal{P}_1$ muß also $\mathcal{T}_P(I_2)$, wie gefordert, `diff(x,y,z)` den Wert *wahr* zuweisen.

2. Die zweite Klausel von $\mathcal{P}_1$ weist `diff(x,y,z)` den Wert *wahr* zu. Dann gelten alle Aussagen analog zu (1.).

- Daraus folgt

$$d_3(\mathcal{T}_\mathcal{P}(I_1), \mathcal{T}_\mathcal{P}(I_2)) \quad \leq \quad (1/2)^{n+1}$$

und damit

$$d_3(\mathcal{T}_\mathcal{P}(I_1), \mathcal{T}_\mathcal{P}(I_2)) \quad \leq \quad d(I_1, I_2) \tag{3.2}$$

- Aus (3.1) und (3.2) folgt mit

$$\begin{aligned} d(I_1, I_2) &= \max\{d_2(I_1, I_2), d_3(I_1, I_2)\} \\ d(\mathcal{T}_\mathcal{P}(I_1), \mathcal{T}_\mathcal{P}(I_2)) &\leq \frac{1}{2} \cdot d(I_1, I_2) \end{aligned}$$

das heißt, wir haben ein k mit $0 < k \leq 1$ gefunden so daß $\mathcal{T}_\mathcal{P}$ eine Kontraktion ist.

∎

3.4 Akzeptable Programme

In [AP93] zeigen Apt und Predeschi, daß für eine eingeschränkte Klassen von Programmen die Definition der Semantik eines Programms in einer zweiwertigen und in einer dreiwertigen Logik übereinstimmt. Dabei betrachten sie die Menge der *akzeptablen* Programme.

Der Operator $\Phi_{\mathcal{P}}$, in einer dreiwertigen Logik, ist auf dreiwertigen Interpretationen $I = (I^+, I^-)$ definiert. Die Menge I^+ enthält alle Grundatome, die als wahr interpretiert werden und die Menge I^- alle Grundatome, die als falsch interpretiert werden. Ein Grundatom A ist undefiniert, falls weder $A \in I^+$ noch $A \in I^-$ gilt. Apt und Predeschi zeigen, daß für eine totale[6] dreiwertige Interpretation $I = (I^+, B_{\mathcal{P}} - I^+)$ die Menge I^+ ein Fixpunkt für $\mathcal{T}_{\mathcal{P}}$ ist gdw. I Fixpunkt von $\Phi_{\mathcal{P}}$ ist. Falls $\Phi_{\mathcal{P}}$ genau einen Fixpunkt besitzt und I total ist, so ist I^+ der eindeutig bestimmte Fixpunkt von $\mathcal{T}_{\mathcal{P}}$. Betrachtet man nun ein *akzeptables* Programm $\mathcal{P}$, kann man zeigen, daß $\Phi_{\mathcal{P}} \uparrow \omega$ der eindeutig bestimmte Fixpunkt von $\Phi_{\mathcal{P}}$ ist und außerdem ist $\Phi_{\mathcal{P}} \uparrow \omega$ total.

In seinem Artikel zeigt Fitting diese Übereinstimmung direkter, als in [AP93], indem er die in den vorangegangenen Abschnitten besprochene metrische Methode nutzt. Er zeigt, daß alle für den zweiwertigen Fall getroffenen Aussagen analog für den dreiwertigen Fall gelten.

Wenn man also zeigen kann, daß nicht nur $\mathcal{T}_{\mathcal{P}}$, sondern auch $\Phi_{\mathcal{P}}$ eine Kontraktion auf dem metrischen Raum der zwei- bzw. dreiwertigen Interpretationen für $\mathcal{P}$ ist, so hat auch $\Phi_{\mathcal{P}}$ einen eindeutig bestimmtem Fixpunkt. Da jeder Fixpunkt von $\mathcal{T}_{\mathcal{P}}$ auch Fixpunkt von $\Phi_{\mathcal{P}}$ ist[7], koinzidieren klassische und dreiwertige Semantik dann für $\mathcal{P}$.

Die Definition eines akzeptablen Programmes übernimmt Fitting direkt von Apt. Dafür benötigen wir noch die folgenden Definitionen.

Definition 3.13 *Sei $\mathcal{P}$ ein normales Programm und seien p und q Prädikatszeichen.*

1. p bezieht sich auf q ($p \Leftarrow q$) *gdw. es eine Klausel in $\mathcal{P}$ mit einem Vorkommen von p im Kopf der Klausel und q im Rumpf der Klausel gibt.*
2. *p ist* abhängig von q ($p \stackrel{*}{\Leftarrow} q$) *gdw. sich das Paar (p, q) in der reflexiven, transitiven Hülle der Relation $\Leftarrow$ befindet.*
3. *$NEG_{\mathcal{P}}$ bezeichnet die Menge von Prädikatszeichen in $\mathcal{P}$, die in einem negativen Literal im Rumpf einer Klausel von $\mathcal{P}$ auftreten.*
4. *$NEG^*_{\mathcal{P}}$ bezeichnet die Menge von Prädikatszeichen in $\mathcal{P}$, von denen die Prädikatszeichen in $NEG_{\mathcal{P}}$ abhängen, also $NEG^*_{\mathcal{P}} = \{q \mid p \stackrel{*}{\Leftarrow} q \wedge p \in NEG_{\mathcal{P}}\}$.*
5. *$\mathcal{P}^-$ bezeichnet die Menge der Klauseln in $\mathcal{P}$, in deren Kopf ein Prädikatszeichen aus $NEG^*_{\mathcal{P}}$ vorkommt.*

[6] siehe Seite 65
[7] siehe [AP93]

Beispiel 3.14 Wir betrachten das normale aussagenlogische Programm $\mathcal{P}$

$$\begin{array}{lll} A. & & \\ B & \leftarrow & A, \neg C. \\ C & \leftarrow & \neg D. \\ D & \leftarrow & \neg A, E. \end{array}$$

und ermitteln die Relationen $\Leftarrow$ und $\stackrel{*}{\Leftarrow}$.

$$\begin{array}{lll} \Leftarrow & = & \{ \text{(B,A), (B,C), (C,D), (D,A),(D,E)} \} \\ \stackrel{*}{\Leftarrow} & = & \{ \text{(A,A), (B,B), (C,C), (D,D), (E,E), B,A), (B,C),} \\ & & \text{(C,D), (D,A), (D,E), (B,D), (C,A), (C,E), (B,E)} \} \end{array}$$

Entsprechend erhalten wir

$$\begin{array}{lll} NEG_{\mathcal{P}} & = & \{A, C, D\} \\ NEG^*_{\mathcal{P}} & = & \{A, C, D, E\} \\ \mathcal{P}^- & = & \{A; C \leftarrow \neg D; D \leftarrow \neg A, E\} \end{array}$$

∘

Definition 3.15 *Die* Einschränkung *einer Menge A von Grundatomen auf eine Menge B von Relationssymbolen ($A \downarrow_B$) ist der Durchschnitt $A \cap \overline{B}$ mit $\overline{B}$ der Menge aller Grundatome der Form $p(t_1, \ldots, t_n)$ mit $p \in B$.*

Definition 3.16 *Sei $\mathcal{P}$ ein normales Programm, $|\_|$ eine Stufenabbildung für $\mathcal{P}$ und I ein Modell für $\mathcal{P}$, dessen Einschränkung auf Relationen aus $NEG^*_{\mathcal{P}}$ ein Modell für comp($\mathcal{P}^-$) ist. $\mathcal{P}$ heißt* akzeptabel *bezüglich $|\_|$ und I, wenn für jede Klausel $A \leftarrow L_1, \ldots, L_m$ in* ground($\mathcal{P}$) *für $1 \leq i \leq m$ gilt:*

$$I \models L_1 \wedge \cdots \wedge L_{i-1} \quad \Rightarrow \quad |A| > |L_i|.$$

Ein Programm $\mathcal{P}$ heißt akzeptabel, *wenn es bezüglich einer Stufenabbildung und eines Modelles akzeptabel ist.*

Treten in $\mathcal{P}$ negierte Atome auf, sichert die Bedingung $I \models \text{comp}(P^-)$ die Existenz einer Widerlegung für die Literale $L_1, \ldots, L_{i-1}$. Sei C Grundinstanz einer Klausel in $\mathcal{P}$ und $\neg A$ ein negatives Literal im Körper von C mit $I \not\models \neg A$. Da die Einschränkung von I auf Relationen aus $NEG^*_{\mathcal{P}}$ ein Modell für comp(P^-) ist, gilt comp(P^-) $\not\models \neg A$. Diese Bedingung schließt die Existenz einer Widerlegung für $\neg A$ aus und entsprechend ist gesichert, daß es keine Notwendigkeit mehr gibt, die Literale in C, die rechts von $\neg A$ im Körper von C auftreten zu untersuchen.

Lokale Stratifiziertheit eines Programmes $\mathcal{P}$ bedeutet, daß für jede Programmklausel $A \leftarrow L_1, \ldots, L_n$ aus $\mathcal{P}$ für alle Grundinstanzen aller Literale L_i

$$|A| \geq |L_i| \text{ falls } L_i \text{ positiv}$$

und

$$|A| > |L_i| \text{ falls } L_i \text{ negativ}$$

gelten muß.

Das Konzept der Akzeptabilität hingegen fordert für Grundinstanzen von Klauseln für die eine Interpretation I die Literale $L_1, \ldots, L_n$ wahr macht, daß $|A| > |L_i|$ gelten muß. Das heißt, $|A| > |L_i|$ muß nur für die Grundinstanzen gezeigt werden, die auch Element der Interpretation I sind, das heißt, nur für die Klauseln, die wirklich von der Funktion $\mathcal{T}_\mathcal{P}$ behandelt werden.

Die Klasse der akzeptablen Programme ist genau so definiert, daß eine Stufenabbildung gefunden werden kann, so daß $\mathcal{T}_\mathcal{P}$ eine Kontraktion auf dem vollständigen metrischen Raum der Herbrand–Interpretationen für $\mathcal{P}$ ist.

Satz 3.17 [Fit93] *Sei $\mathcal{P}$ ein akzeptables Programm. Auf der Menge von Interpretationen $\mathcal{M}_\mathcal{P}$ für $\mathcal{P}$ existiert eine Metrik $d_\mathcal{P}$, so daß $\mathcal{T}_\mathcal{P}$ eine Kontraktion auf dem vollständigen metrischen Raum $(\mathcal{M}_\mathcal{P}, d_\mathcal{P})$ ist.*

Analog kann zu jedem akzeptablen Programm in einer dreiwertigen Logik eine Stufenabbildung gefunden werden, so daß der dreiwertige Operator $\Phi_\mathcal{P}$ eine Kontraktion auf dem vollständigen metrischen Raum der dreiwertigen Interpretationen für $\mathcal{P}$ ist. Da jeder Fixpunkt von $\mathcal{T}_\mathcal{P}$ auch Fixpunkt von $\Phi_\mathcal{P}$ ist[8], muß die Semantik in der zweiwertigen mit der Semantik in der dreiwertigen Logik übereinstimmen.

Die Klasse der akzeptablen Programme umfaßt auch Programme, die nicht lokal stratifiziert sind. Als Beispiel führt Fitting das bereits in Beispiel 2.62 erwähnte Programm $\mathcal{P}(\mathcal{G})$ an. Sei $\mathcal{G}$ ein azyklischer endlicher Graph, dessen Positionen wir mit den Konstanten $a, b, c, \ldots$ bezeichnen. Die Paare $(a, b), (b, c), \ldots$ repräsentieren die Kanten des Graphen. Das Prädikat *move* beschreibt alle zulässigen Bewegungen der Spieler entlang von Kanten des Graphen. Ein Spieler hat das Spiel genau dann verloren, wenn er keine Bewegung mehr ausführen kann.

$$\begin{array}{lllll} \mathcal{P}(\mathcal{G}): & \text{win(X)} & \leftarrow & \text{move(X,Y) , } \neg \text{ win(Y).} \\ & \text{move(a,b).} & & \forall (a, b) \in \mathcal{G} \end{array}$$

Proposition 3.18 *$\mathcal{P}(\mathcal{G})$ ist nicht lokal stratifiziert.*

Beweis Für eine Grundinstanz $\text{win}(a) \leftarrow \text{move}(a, a), \neg\text{win}(a)$ der ersten Klausel in $\mathcal{P}(\mathcal{G})$ und eine beliebige Stufenabbildung $|\_|$ erhalten wir $|\text{win}(a)| = |\neg\text{win}(a)|$. ■

$\mathcal{P}$ ist jedoch akzeptabel, da eine Stufenabbildung $|\_|$ und eine Interpretation I gefunden werden können, so daß I Modell für $\mathcal{P}(\mathcal{G})$ ist und die oben angegebene Instanziierung von $\mathcal{P}$ verhindert.

[8] siehe [AP93]

Die Stufenabbildung $|\_|$ nutzt eine Abbildung f mit:

$$f(a) = \begin{cases} 0 & \text{falls für kein } b\text{: } (a,b) \in \mathcal{G} \\ 1 + \max\{f(b) | (a,b) \in \mathcal{G}\} & \text{sonst} \end{cases}$$

und eine Abbildung g mit:

$$g(a) = \begin{cases} 0 & \text{falls für kein } b\text{: } (a,b) \in \mathcal{G} \\ 1 - \min\{g(b) | (a,b) \in \mathcal{G}\} & \text{sonst} \end{cases}$$

Dabei beschreibt die Funktion $f(a)$ die Tiefe des Teilgraphen an einem Knoten a. Die Funktion $g(a)$ liefert den Wert 0, falls von a bis zum tiefsten Knoten des Graphen eine ungerade Anzahl von Kanten durchlaufen werden muß, anderenfalls liefert sie den Wert 1.

Proposition 3.19 *Die Interpretation*

$$I = \{\text{move}(a,b) | (a,b) \in \mathcal{G}\} \cup \{\text{win}(a) | g(a) = 1\}$$

ist Modell für $\mathcal{P}(\mathcal{G})$.

Beweis
Betrachtet man eine Grundinstanz der ersten Klausel

$$\text{win}(a) \leftarrow \text{move}(a,b), \neg\text{win}(b)$$

und nimmt man $I \models \text{move}(a,b) \wedge \neg\text{win}(b)$ an, gilt $(a,b) \in \mathcal{G}$ und $g(b) = 0$ und damit $g(a) = 1$ woraus $I \models \text{win}(a)$ folgt. Außerdem ist I Modell für alle Grundinstanzen von move–Klauseln. ∎

Theorem 3.20 *Das Programm $\mathcal{P}(\mathcal{G})$ ist akzeptabel.*

Beweis Es gilt $\mathcal{P}(\mathcal{G}) = \mathcal{P}(\mathcal{G})^-$. Wir wollen zeigen, daß I ein Modell für $\text{comp}(\mathcal{P}(\mathcal{G}))$ und damit auch für $\text{comp}(\mathcal{P}(\mathcal{G})^-)$ ist. Wir betrachten ein Grundatom $\text{win}(a) \in I$. Mit $g(a) = 1$ für ein beliebiges $b \in \mathcal{G}$ erhalten wir $(a,b) \in \mathcal{G}$ und $g(b) = 0$, woraus $I \models \text{move}(a,b) \wedge \neg\text{win}(b)$ folgt.

Wir betrachten eine Grundinstanz der ersten Klausel unseres Programmes:

$$\text{win}(a) \leftarrow \mathit{move}(a,b), \neg\text{win}(b)$$

Wir definieren eine Stufenabbildung $|\_|$ mit:

$$\begin{array}{lll} |\text{move}(a,b)| & \mapsto & f(a) \quad \forall (a,b) \in \mathcal{G} \quad \text{und} \\ |\text{win}(a)| & \mapsto & f(a) + 1 \quad \forall a \in \mathcal{G}. \end{array}$$

Laut Definition gilt dann

$$|\mathrm{win}(a)| = f(a) + 1 > f(a) = |\mathrm{move}(a, b)|.$$

Gelte $I \models \mathrm{move}(a, b)$. Dann gilt $(a, b) \in \mathcal{G}$ und also $f(a) > f(b)$, woraus folgt

$$|\mathrm{win}(a)| = f(a) + 1 > f(b) + 1 = |\neg\mathrm{win}(b)|$$

■

Dieses Programm ist eines der Beispiele, die zur Definition neuer Semantiken für normale Programme führten, so der wohlfundierten Modellsemantik und der stabilen Modellsemantik. Im Abschnitt 7 soll näher auf die Beziehung zwischen akzeptablen Programmen und Programmen mit genau einem stabilen Modell eingegangen werden.

Es gibt jedoch auch lokal stratifizierte Programme, die nicht akzeptabel sind. Insbesondere sind definite Programme, die lokal stratifiziert sind, nicht immer akzeptabel.

Beispiel 3.21 Sei $\mathcal{P}$ das definite Programm:

$$\begin{array}{lcl} p(X). & & \\ q(X) & \leftarrow & p(X). \\ r(X) & \leftarrow & q(X). \\ p(X) & \leftarrow & r(X). \end{array}$$

mit $U_{\mathcal{P}} = \{a\}$ und $B_{\mathcal{P}} = \{p(a), q(a), r(a)\}$. $\mathcal{P}$ ist definit und damit ist der kleinste Fixpunkt von $\mathcal{T}_{\mathcal{P}}$ ermittelbar durch $\mathcal{T}_{\mathcal{P}} \uparrow \omega = M_{\mathcal{P}} = \{p(a), q(a), r(a)\} = B_{\mathcal{P}}$. Wegen $B_{\mathcal{P}} = M_{\mathcal{P}}$ ist $\{p(a), q(a), r(a)\}$ nicht nur kleinstes sondern auch einziges Herbrand-Modell für $\mathcal{P}$. Für dieses Modell finden wir jedoch keine Stufenabbildung mit der geforderten Eigenschaft $(|p(a)| > |q(a)| > |r(a)| > |p(a)|)$, so daß $\mathcal{P}$ nicht akzeptabel ist. ∘

Seien p, q und r Prädikatszeichen in einem definiten Programm $\mathcal{P}$. Falls in $\mathcal{P}$ Zyklen der Form $p \Leftarrow q \Leftarrow r \Leftarrow \cdots \Leftarrow p$ auftreten, so ist $\mathcal{P}$ nicht akzeptabel. Für diese Programme kann man nur eine Stufenabbildung mit $|p| = |q| = |r| = \cdots = |p|$ finden. Dann ist $\mathcal{P}$ lokal stratifiziert. Für eine solche Stufenabbildung kann die Funktion $\mathcal{T}_{\mathcal{P}}$ aber nicht die Anforderungen an eine Kontraktion erfüllen. Gilt aber nun, daß die Stufe des Prädikatszeichens im Kopf gleich der Stufe des Prädikatszeichens im Körper einer Klausel ist, vergrößert sich die Stufe der Prädikatszeichen in den Interpretationen nicht, daß heißt, ihr Abstand wird nicht geringer. Damit kann die Konvergenz der Funktionsanwendung von $\mathcal{T}_{\mathcal{P}}$ nicht mehr gezeigt werden.

Abbildung 3.1 zeigt die Beziehungen zwischen den von uns betrachteten Klassen von Programmen.

Abbildung 3.1: Beziehungen zwischen den von uns betrachteten Klassen von Programmen.

4. Logische Programme und rekurrente Netzwerke

4.1 Die Idee eines neuen Berechnungsmodelles

In der vorliegenden Arbeit soll untersucht werden, ob metrische Methoden verwendet werden können, um eine Beziehung zwischen logischer Programmierung und konnektionistischen Berechnungsmodellen zu finden.

In [HSW89] wird formal gezeigt, daß mehrschichtige vorwärtsgerichtete Netze mit nur einer verborgenen Schicht in der Lage sind, jede (Borel-)meßbare Funktion beliebig genau zu approximieren. Entsprechend sollte man ein Netz realisieren können, daß den mit dem logischen Programm $\mathcal{P}$ verbundenen Operator $\mathcal{T}_{\mathcal{P}}$ berechnet.

Mehrschichtige vorwärtsgerichtete Netze sind eines der am meisten verwendeten konnektionistischen Modelle, deren Eigenschaften recht gut untersucht worden sind. Effektive Lernalgorithmen für dieses Modell wurden theoretisch untersucht und werden oft in der Praxis angewendet. Verschiedene Varianten rekurrenter mehrschichtiger Netze wurden in unterschiedlichen Arbeiten untersucht, so von Jordan [Jor86], Elman [Elm90] und Pollack [Pol89]. In [Pol89] zeigt Pollack, daß solche Netze in der Lage sind rekursive Strukturen zu repräsentieren, was eine bis dahin stark kritisierte Unzulänglichkeit konnektionistischer Modelle beseitigt.

Die Konstruktion eines Netzes zur Berechnung von $\mathcal{T}_{\mathcal{P}}$ ergibt sich direkt aus dem Programm $\mathcal{P}$. Da $\mathcal{T}_{\mathcal{P}}$ Interpretationen auf Interpretationen abbildet, repräsentieren die Ein- und Ausgabeschicht des Netzes jeweils Interpretationen für $\mathcal{P}$. Die Werte der Einheiten in der Eingabeschicht repräsentieren die Interpretation I_n und die Werte der Einheiten in der Ausgabeschicht die Interpretation $I_{n+1} = \mathcal{T}_{\mathcal{P}}(I_n)$. Erweitert man das Netz durch Rückführung der Werte der Ausgabeeinheiten zu den Eingabeeinheiten zu einem rekurrenten Netz (siehe Abbildung 2.2), so berechnet dieses Netz die Iteration von $\mathcal{T}_{\mathcal{P}}$. Diese startet mit einer beliebigen Aktivierung der Einheiten in der Eingabeschicht, d.h. mit einer beliebigen Anfangsinterpretation. Falls man zeigen kann, daß $\mathcal{T}_{\mathcal{P}}$ eine Kontraktion auf dem metrischen Raum von Interpretationen für das Programm $\mathcal{P}$ ist, sichert Banachs Kontraktionstheorem die Konvergenz der Iteration und damit die Konvergenz des Netzes. Die Iteration endet in einem stabilen Zustand des Netzes, der das kleinste Modell

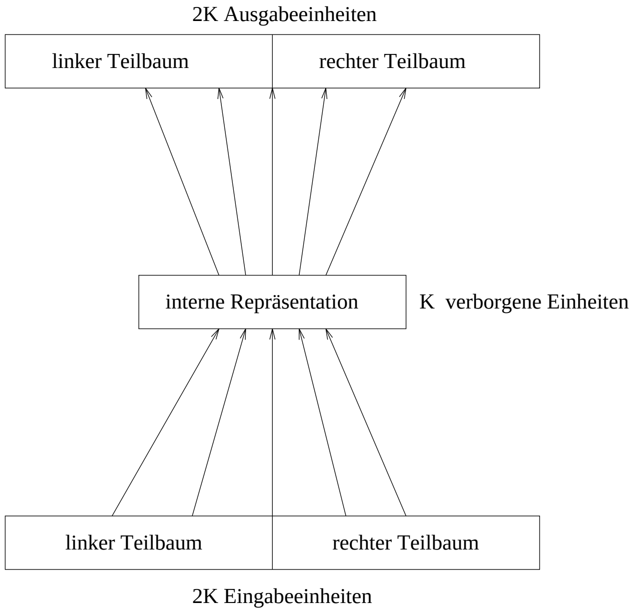

Abbildung 4.1: Ein 3–schichtiges vorwärtsgerichtetes Netz, wie es von Pollack verwendet wird. Die Schichten sind voll vernetzt. Durch Rückführung der Werte der Ausgabeeinheiten zu den Eingabeeinheiten erhalten wir ein rekurrentes Netz, daß angelernt werden kann, für ein Muster der Länge k eine interne Repräsentation der Länge k zu erzeugen.

für das Programm $\mathcal{P}$ repräsentiert.

Die Anwendung der von Fitting vorgeschlagenen metrischen Methode liefert uns somit ein neues massiv paralleles Berechnungsmodell für die Semantik logischer Programme.

4.2 Anwendungen mehrschichtiger rekurrenter Netze

Bei der Modellierung konnektionistischer Architekturen für Probleme auf einer sehr hohen Abstraktionsebene, wie zum Beispiel der Sprachverarbeitung, erschien eine adäquate Repräsentation der verwendeten Datenstrukturen als eines der größten Probleme konnektionistischer Modelle. Besonders die Repräsentation variabel–großer rekursiver Datenstrukturen, wie zum Beispiel Bäume und Listen variabler Länge, erschien als ein Schwachpunkt konnektionistischer Modelle. In [Pol89] zeigt Pollack eine konnektionistische Architektur, die sowohl eine kompakte verteilte Repräsentation für zusammengesetzte symbolische Strukturen, im Beispiel Sätze einer Sprache, als auch effiziente Zugriffsmechanismen auf diese entwickelt.

Abbildung 4.1 zeigt ein 3–schichtiges vorwärtsgerichtetes Netz, wie es von Pollack ver-

Abbildung 4.2: Beispiel eines binären Baumes

wendet wird.

Durch Rückpropagierung der Werte der Ausgabeeinheiten auf die Eingabeschicht wird das Netz darauf trainiert, ein an der Eingabeschicht anliegendes Muster an der Ausgabeschicht zu reproduzieren. Um dies zu erlernen, entwickelt das Netz in den verborgenen Schichten einen komprimierten Kode (k anstelle von $2k$ Einheiten) für jedes Eingabemuster. Trainiert man zum Beispiel ein (8-3-8)–Netz darauf, eine Folge von 8 Bit zu reproduzieren, so entwickelt es einen binären 3–Bit Kode.

Beispiel 4.1 Wollen wir eine Kodierung für den in Abbildung 4.2 gezeigten Baum erzeugen, müssen die Einheiten der Ausgabeschicht 1 : 1 auf die Einheiten der Eingabeschicht rückgekoppelt werden. Seien A, B, C und D Muster der Länge k Bit, wird das Netz wie folgt trainiert:

Eingabeschicht		Verborgene Schicht		Ausgabeschicht
$(A \quad B)$	$\rightarrow$	$R_1(t)$	$\rightarrow$	$(A'(t) \quad B'(t))$
$(C \quad D)$	$\rightarrow$	$R_2(t)$	$\rightarrow$	$(C'(t) \quad D'(t))$
$((R_1(t) \quad R_2(t))$	$\rightarrow$	$R_3(t)$	$\rightarrow$	$(R_1(t)' \quad R_2(t)')$

t repräsentiert die Trainingszeit. Nehmen wir an, daß die Iteration des Netzes nach endlicher Zeit konvergiert, so verringert sich mit zunehmender Zeit t die Differenz zwischen der gewünschten und der tatsächlichen Ausgabe des Netzes und es gilt: $A' = A; B' = B; C' = C; D' = D, R'_1 = R_1; R'_2 = R_2$. Das heißt, das Netz findet eine Repräsentation für $((A \quad B)(C \quad D))$. $\circ$

Die Architektur dieses Netzes entspricht der von uns verwendeten Architektur zur Berechnung der Funktion $\mathcal{T_P}$ für ein Programm $\mathcal{P}$. Wir beschränken vorerst unsere Betrachtungen auf den aussagenlogischen Fall. Um aussagenlogische Variable zu repräsentieren, erweisen sich die Einheiten des Netzes als ausreichend. Erweitern wir unsere Betrachtungen auf den prädikatenlogischen Fall, erlangt die Repräsentation der von uns darzustellenden Strukturen jedoch Bedeutung, da eine Repräsentation für Formeln mit Variablen gefunden werden muß.

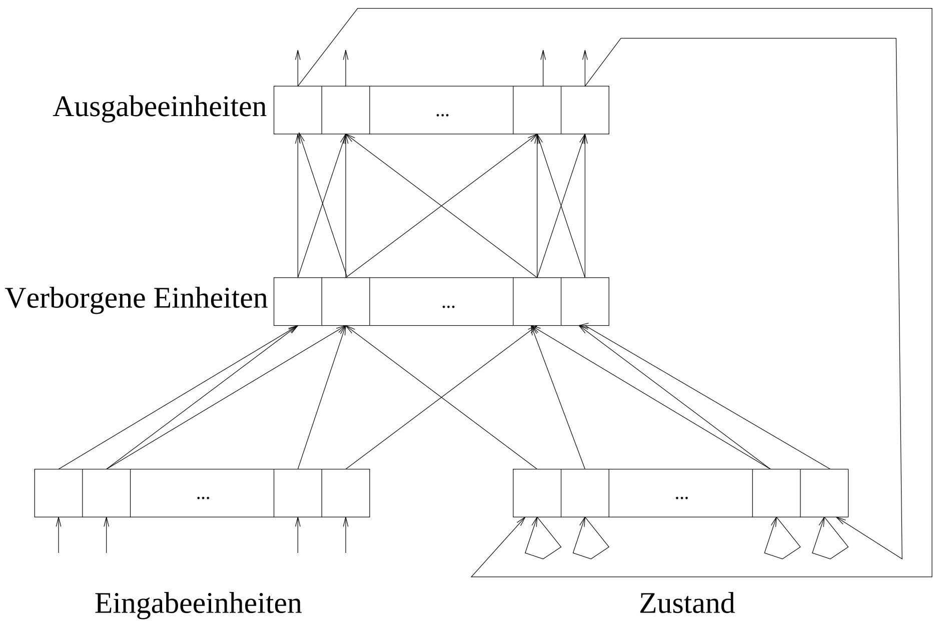

Abbildung 4.3: Ein 3–schichtiges vorwärtsgerichtetes Netz, wie es von Jordan verwendet wird. Die Einheiten in der Ausgabeschicht werden 1 : 1 mit dem Gewicht 1 auf die Zustandseinheiten abgebildet. Aus Gründen der Übersichtlichkeit sind nicht alle Verbindungen angegeben.

In [Jor86] untersucht Jordan das Problem der Parallelität der Sprachproduktion und schlägt eine konnektionistische Architektur vor, die ein ähnliches Verhalten, wie es in der Sprache auftritt, darstellen kann. Die von ihm benutzen Netztypen ähneln ebenfalls den von uns untersuchten 3-schichtigen rekurrenten Netzen. Abbildung 4.3 zeigt ein solches Netz. Um Parallelität darzustellen, werden Zustandseinheiten eingeführt. Diese erhalten Eingaben aus der Ausgabeschicht. Die Einheiten in der verborgenen Schicht erhalten sowohl Eingaben von den Einheiten in der Eingabeschicht, als auch Eingaben aus den Zustandseinheiten. Diese repräsentieren aber gerade den aktuellen Zustand des Netzes. Damit wird implizit das zeitliche Verhalten berücksichtigt.

Vernachlässigt man die Betrachtung der Eingabeeinheiten des von Jordan verwendeten Netzes, erhält man ein Netz, wir wir es zur Iteration der Funktion $\mathcal{T}_{\mathcal{P}}$ verwenden.

Elman schlägt in [Elm90] ebenfalls eine konnektionistische Architektur vor, in der Zeit repräsentiert werden soll. Abbildung 4.4 zeigt ein solches 3–schichtiges Netz. Elman benutzt Kontexteinheiten, um den letzten internen Zustand des Netzes zu „speichern“. Dieser dient dann zusammen mit den Ausgabewerten der Einheiten in der Eingabeschicht als Eingabe der verborgenen Schicht. Die interne Repräsentation, die von den Einheiten in der verborgenen Schicht durch Anwendung von Lernalgorithmen entwickelt wird, ist damit abhängig vom zeitlichen Kontext.

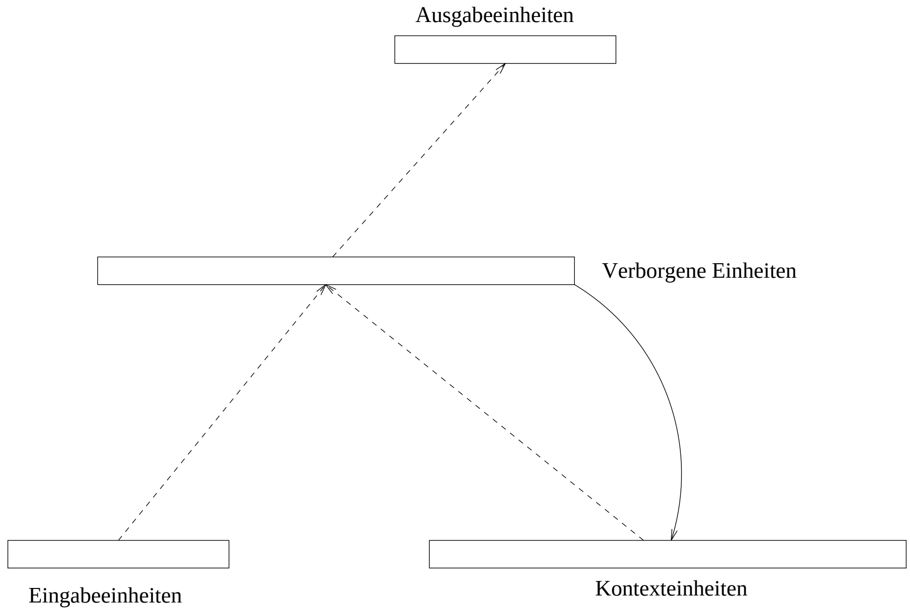

Abbildung 4.4: Ein 3–schichtiges vorwärtsgerichtetes Netz, wie es von Elman verwendet wird. Die Einheiten in der verborgenen Schicht und die Einheiten in der Ausgabeschicht sind voll vernetzt. Die Einheiten in der verborgenen Schicht werden 1 : 1 mit dem Gewicht 1 auf die Kontexteinheiten abgebildet.

Die Iteration des von Elman verwendeten Netzes wird nur über zwei Schichten realisiert. Wir werden im folgenden zeigen, daß ein Netz, daß die Funktion $\mathcal{T}_\mathcal{P}$ für ein akzeptables Programm $\mathcal{P}$ berechnet, mindestens eine verborgene Schicht besitzen muß.

4.3 Mehrschichtige rekurrente Netze als universelle Appoximatoren

Theorem 4.2 [HSW89] *Ein n-schichtiges vorwärtsgerichtetes Netz* $(n > 2)$ *ist in der Lage, jede Borel-meßbare Funktion hinreichend genau zu approximieren.*

Der Begriff der Borel-Meßbarkeit kommt aus der Messungstheorie. Die Klasse der Borelmeßbaren Funktionen kann als die kleinste Klasse charakterisiert werden, welche die stetigen Funktionen enthält und die bezüglich der Summe, der Differenz und des Produktes endlich vieler dieser Funktionen und bezüglich des Grenzüberganges einer überall konvergenten Folge dieser Funktionen abgeschlossen ist. Mit anderen Worten, diese Klasse besteht aus den stetigen Funktionen und denjenigen Funktionen, die man aus ihnen durch wiederholte Grenzprozesse gewinnen kann.

In [Bil79] zeigt Billingsley, daß nicht Borel-meßbare Funktionen existieren, diese jedoch pathologisch sind.

Für ein definites Programm $\mathcal{P}$ kann man zeigen, daß die Funktion $\mathcal{T}_{\mathcal{P}}$ stetig ist[1]. Da jede stetige Funktion Borel-meßbar ist, gilt der folgende Satz.

Satz 4.3 *Sei $\mathcal{P}$ ein definites Programm. Die Funktion $\mathcal{T}_{\mathcal{P}}$ ist Borel-meßbar.*

Für die Klasse der akzeptablen Programme $\mathcal{P}$ wird vermutet, daß $\mathcal{T}_{\mathcal{P}}$ ebenfalls Borel-meßbar ist.

Im folgenden soll kurz darauf eingegangen werden, was wir in Theorem 4.2 mit *hinreichend genau* bezeichnen. Dazu benötigen wir die folgende Definition.

Definition 4.4 *Eine Teilmenge S eines metrischen Raumes $(\mathcal{M}, d)$ heißt d-*dicht *in einer Teilmenge T, falls für jedes $\epsilon > 0$ und für jedes $t \in T$ ein $s \in S$ existiert, so daß*

$$d(s,t) < \epsilon$$

gilt.

Bezeichne f die Funktion, die wir mit einem mehrschichtigen vorwärtsgerichteten Netz berechnen wollen und g, die von diesem Netz berechnete Funktion. Bilden wir den Wertebereich dieser Funktionen in einen metrischen Raum ab, so beschreibe die Menge S den Wertebereich von f und die Menge T den Wertebereich von g. Die Funktion g berechnet die Funktion f hinreichend genau, wenn S in T d-dicht ist. Im Beispiel reeller Funktionen werden die Funktionswerte von f und g in den metrischen Raum $\mathbb{R}^1$ abgebildet[2]. Die Metrik d ist dann der Betrag des Abstandes der beiden Funktionen.

[1] siehe [Llo87]
[2] siehe Beispiel 2.54

5. Aussagenlogische Programme

Im folgenden bezeichne $\mathcal{P}$ ein aussagenlogisches Programm. Für den aussagenlogischen Fall soll untersucht werden, wie zu einem Programm $\mathcal{P}$ ein mehrschichtiges vorwärtsgerichtetes Netz $\mathcal{F}$ konstruiert werden kann, so daß $\mathcal{F}$ den Operator $T_{\mathcal{P}}$ berechnet.

Weiterhin wird gezeigt, daß zu einem gegebenen mehrschichtigen rekurrenten Netz $\mathcal{N}$ ein Programm $\mathcal{P}$ gefunden werden kann, so daß $\mathcal{N}$ gerade den Operator $T_{\mathcal{P}}$ für $\mathcal{P}$ berechnet. Konvergiert die Berechnung von $\mathcal{N}$, so wollen wir zeigen, daß die Funktion $T_{\mathcal{P}}$ eine Kontraktion auf dem metrischen Raum der Interpretationen für $\mathcal{P}$ ist. Ob es sich bei $\mathcal{P}$ in diesem Fall um ein akzeptables Programm handelt, kann nicht gezeigt werden, wird aber vermutet.

In den ersten Abschnitten des Kapitels wollen wir zeigen, daß für die Funktion XOR, die nicht durch ein Perzeptron realisiert werden kann, ein 3–schichtiges vorwärtsgerichtetes Netz gefunden werden kann, daß die Abbildung $T_{\mathcal{P}}$ für die Funktion XOR berechnet. Daraus resultiert die Frage, ob für die eingeschränkte Klasse definiter Programme ein Netz mit weniger Schichten, ein Perzeptron, für die Berechnung von $T_{\mathcal{P}}$ verwendet werden kann. Im zweiten Abschnitt wird gezeigt, daß dies nicht der Fall ist, das heißt, auch für die Klasse definiter Programme wird ein Netz mit mindestens einer verborgenen Schicht benötigt, um $T_{\mathcal{P}}$ zu berechnen.

5.1 Die Funktion XOR

Satz 5.1 **[MP72]** *Die binäre logische Funktion XOR ist nicht linear separabel.*

Damit kann diese Funktion nicht durch ein Perzeptron realisiert werden (siehe Proposition 2.73). Im folgenden wollen wir die Eigenschaften eines normalen Programms untersuchen, daß die Funktion XOR berechnet.

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Bringt man die Wertetabelle für XOR in Klauselform, so erhält man

$$C = A \text{ xor } B = (A \wedge \neg B) \vee (\neg A \wedge B)$$

und damit das folgende Programm

$$\begin{array}{llll} \mathcal{P}: & C & \leftarrow & A \wedge \neg B. \\ & C & \leftarrow & \neg A \wedge B. \end{array}$$

mit der Herbrand-Basis $B_{\mathcal{P}} = \{A, B, C\}$.

Abbildung 5.1 zeigt ein 3-schichtiges vorwärtsgerichtetes Netz $\mathcal{N}$, daß wir zur Berechnung des Operators $\mathcal{T}_{\mathcal{P}}$ für unser Programm $\mathcal{P}$ benutzen wollen.[1]. Die Einheiten in der Ein- und in der Ausgabeschicht des Netzes repräsentieren die Grundatome A, B und C. Die Wertebelegung dieser Einheiten repräsentiert die Interpretation dieser Grundatome. Sei I die an den Einheiten der Eingabeschicht angelegte Interpretation. An den Einheiten der Ausgabeschicht kann nach 2 Schritten die Interpretation $\mathcal{T}_{\mathcal{P}}(I)$ abgelesen werden.

Die Wertetabelle zeigt die vom Netz realisierte Berechnung und zeigt, daß das Netz $\mathcal{N}$ den Operators $\mathcal{T}_{\mathcal{P}}$ für unser Programm $\mathcal{P}$ berechnet.

$I(A)$	$I(B)$	$I(C)$	i_1	i_2	$\mathcal{T}_{\mathcal{P}}(I)(A)$	$\mathcal{T}_{\mathcal{P}}(I)(B)$	$\mathcal{T}_{\mathcal{P}}(I)(C)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	1	0	0	1
0	1	1	0	1	0	0	1
1	0	0	1	0	0	0	1
1	0	1	1	0	0	0	1
1	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0

Die Interpretation $I = \{A, C\}$ ist Modell für $\mathcal{P}$. Wir ermitteln $NEG^*_{\mathcal{P}} = \{A, B\}$ und $\mathcal{P}^- = \emptyset$. Folglich gilt $I \upharpoonright_{NEG^*_{\mathcal{P}}} \models \text{comp}(\mathcal{P}^-)$.

Die Stufenabbildung

$$\begin{array}{lllll} |\_| & : & B_P & \rightarrow & \mathbb{N} \\ & & A & \mapsto & 1 \\ & & B & \mapsto & 1 \\ & & C & \mapsto & 2 \end{array}$$

und die Interpretation I sichern somit die Akzeptabilität von $\mathcal{P}$.

$\mathcal{P}$ gehört damit zur Menge der akzeptablen Programme, für die in [Fit93] gezeigt wird, daß $\mathcal{T}_{\mathcal{P}}$ Kontraktion ist. Banachs Kontraktionstheorem sichert die Existenz eines eindeutig bestimmten Fixpunktes für $\mathcal{T}_{\mathcal{P}}$.

[1] Die Konstruktion eines solchen Netzes wird in Abschnitt 5.3 beschrieben.

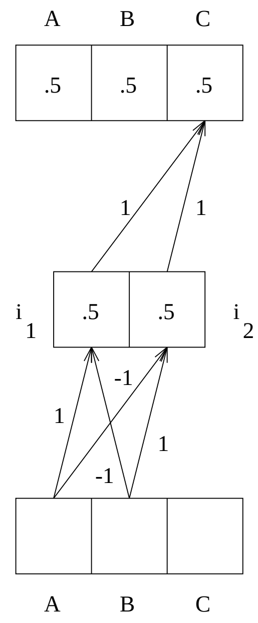

Abbildung 5.1: Ein 3–schichtiges vorwärtsgerichtetes Netz zur Berechnung des Operators $\mathcal{T}_\mathcal{P}$ für das Programm $\mathcal{P} = \{C \leftarrow A \wedge \neg B; C \leftarrow \neg A \wedge B\}$.

5.2 Notwendigkeit verborgener Schichten

In [HSW89] wird gezeigt, daß ein mehrschichtiges vorwärtsgerichtetes Netz jede (Borel-) meßbare Funktion hinreichend genau approximieren kann. Also kann solch ein Netz auch genutzt werden, um den Operator $\mathcal{T}_\mathcal{P}$ für ein Programm $\mathcal{P}$ zu berechnen. Im Kapitel 5.3 werden wir zeigen, daß für aussagenlogische Programme 3–schichtige vorwärtsgerichtete Netze mit binären Schwellwerteinheiten den Operator $\mathcal{T}_\mathcal{P}$ für ein Programm $\mathcal{P}$ berechnen können. In diesem Abschnitt wollen wir untersuchen, ob einfachere Netze in der Lage sind, $\mathcal{T}_\mathcal{P}$ für eine bestimmte Klasse von Programmen zu berechnen.

In [MP72] zeigen Minsky und Papert, daß die Fähigkeit eines Perzeptrons auf die Lösung linear separabler Probleme beschränkt ist. Im folgenden soll gezeigt werden, daß ein Perzeptron nicht in der Lage ist $\mathcal{T}_\mathcal{P}$ für ein definites aussagenlogisches Programm $\mathcal{P}$ zu berechnen.

Proposition 5.2 *Ein einfaches Perzeptron ist nicht in der Lage, den Operator $\mathcal{T}_\mathcal{P}$ für ein definites aussagenlogisches Programm $\mathcal{P}$ zu berechnen.*

Abbildung 5.2: Die allgemeinste Form eines Perzeptrons. Die Kanten, die offensichtlich mit Null bewertet sein müssen, sind weggelassen.

Beweis Sei $\mathcal{P}$ das definite Programm:

$$\begin{array}{rlll} \mathcal{P}: & A & & \\ & A & \leftarrow & B. \\ & A & \leftarrow & C \wedge D. \\ & A & \leftarrow & E \wedge F. \end{array}$$

Sei $\mathcal{F}$ die allgemeinste Form eines Perzeptrons, wie in Abbildung 5.2 gezeigt. Wir nehmen an, daß die Gewichte in $\mathcal{F}$ so bestimmt sind, daß $\mathcal{F}$ den Operator $\mathcal{T}_{\mathcal{P}}$ für unser Programm $\mathcal{P}$ berechnet. Für $i \in \{8, \ldots, 12\}$ seien $w_{ij} = 0$ und $\theta_i > 0$. Damit wird keine der Einheiten 8 bis 12 aktiv, was bedeutet, daß B, C, D, E und F unabhängig von der Belegung der Einheiten in der Eingabeschicht stets als *falsch* bezüglich der Interpretation $\mathcal{T}_{\mathcal{P}}(I)$ bezüglich einer beliebigen Interpretation I interpretiert werden.

Betrachtet man die Einheit 7, so müssen Gewichte w_{7j} gefunden werden, so daß A genau dann als *wahr* interpretiert wird, wenn entweder B oder aber C und D beziehungsweise E und F von der vorliegenden Interpretation als *wahr* interpretiert werden.

Um ein Problem durch ein Perzeptron lösen zu lassen, muß ein Gewichtsvektor $\vec{w}$ gefunden werden, so daß für einen beliebigen Eingabevektor $\vec{i}$

$$\sum_i w_i \cdot i_i \geq \theta$$

wobei θ für den Schwellwert der Ausgabeeinheit des Perzeptrons steht.

Unser Problem kann geschrieben werden als

$$\begin{array}{lll} I(A) = wahr & \text{, falls} & I(A) = wahr \vee \\ & & I(B) = wahr \vee \\ & & (I(C) = wahr \wedge I(D) = wahr) \vee \\ & & (I(E) = wahr \wedge I(F) = wahr) \end{array}$$

Es ist also das folgende *Klassifikationsproblem* zu lösen

$$I(A) = 1 \qquad \Longleftrightarrow \qquad w_1 i_1 + w_2 i_2 + w_3 i_3 + w_4 i_4 + w_5 i_5 + w_6 i_6 \geq \theta$$

Aufgrund der Kommutativität der Konjunktion ($\wedge$) gilt

$$\begin{array}{ll} I(A) = 1 & \\ \Longleftrightarrow & w_1 i_1 + w_2 i_2 + w_4 i_3 + w_3 i_4 + w_5 i_5 + w_6 i_6 \geq \theta \\ \Longleftrightarrow & w_1 i_1 + w_2 i_2 + w_3 i_3 + w_4 i_4 + w_6 i_5 + w_5 i_6 \geq \theta \\ \Longleftrightarrow & 2w_1 i_1 + 2w_2 i_2 + w_4 i_3 + w_3 i_4 + +w_3 i_3 + w_4 i_4 + w_6 i_5 + w_5 i_6 + w_5 i_5 + w_6 i_6 \geq 2\theta \\ \Longleftrightarrow & 2w_1 i_1 + 2w_2 i_2 + (w_4 + w_3) i_3 + (w_4 + w_3) i_4 + (w_6 + w_5) i_5 + (w_6 + w_5) i_6 \geq 2\theta \\ \Longleftrightarrow & 2w_1 i_1 + 2w_2 i_2 + (w_4 + w_3)(i_3 + i_4) + (w_6 + w_5)(i_5 + i_6) \geq 2\theta \end{array}$$

und mit

$$\begin{array}{lll} w_{34} & = & (w_3 + w_4)/2 \\ w_{56} & = & (w_5 + w_6)/2 \end{array}$$

$$I(A) = 1 \quad \text{gdw.} \quad w_1 i_1 + w_2 i_2 + w_{34}(i_3 + i_4) + w_{56}(i_5 + i_6) \geq \theta$$

Aufgrund der Kommutativität der Konjunktion ($\vee$) kann analog ein w gefunden werden, so daß

$$I(A) = 1 \text{ gdw. } w(i_1 + i_2 + i_3 + i_4 + i_5 + i_6) \geq \theta$$

gilt. Mit $x = (i_1 + i_2 + i_3 + i_4 + i_5 + i_6)$ erhält man $wx \geq \theta$. Jetzt gilt aber für $i_3 = i_5 = 1$ und $i_1 = i_2 = i_4 = i_6 = 0$ (C und E werden als *wahr* interpretiert) $wx < \theta$ und $x = 2$. Es gilt aber auch für $i_4 = i_5 = 1$ und $i_1 = i_2 = i_3 = i_6 = 0$ (C und D werden als *wahr* interpretiert) $wx \geq \theta$ und $x = 2$. Das heißt es gilt $wx \neq wx$ für $x = 2$. Da die Multiplikation eine Funktion ist, kann es kein w geben, das diese Ungleichung erfüllt.

Folglich kann es keine Schwellwerteinheit und kein einfaches Perzeptron geben, das das gegebene Problem löst. ■

Damit haben wir gezeigt, daß für definite Programme mindestens eine verborgene Schicht benötigt wird, um den Operator $\mathcal{T}_{\mathcal{P}}$ zu berechnen. Das 3–schichtige vorwärtsgerichtete Netz in Abbildung 5.3 berechnet $\mathcal{T}_{\mathcal{P}}$ für unser Programm $\mathcal{P}$. Die Iteration von $\mathcal{T}_{\mathcal{P}}$ liefert

Abbildung 5.3: Ein 3-schichtiges vorwärtsgerichtetes Netz zur Realisierung der Funktion $\mathcal{T}_\mathcal{P}$ für das Programm $\mathcal{P} = \{A; A \leftarrow B; A \leftarrow C, D; A \leftarrow E, F\}$.

$$\begin{array}{lclcl} \mathcal{T}_\mathcal{P} \uparrow 0 & = & \mathcal{T}_\mathcal{P}(\bot) & = & \emptyset \\ \mathcal{T}_\mathcal{P} \uparrow 1 & = & \mathcal{T}_\mathcal{P}(\emptyset) & = & \{A\} \\ \mathcal{T}_\mathcal{P} \uparrow 2 & = & \mathcal{T}_\mathcal{P}(\{A\}) & = & \{A\} \end{array}$$

und die Iteration des Netzes liefert zum Beispiel für die Anfangsinterpretation $I_0 = \{B, D\}$:

	Anfangsinterpretation I_0	$I_1 = \mathcal{T}_\mathcal{P}(I_0)$	$I_2 = \mathcal{T}_\mathcal{P}(I_1)$
$I_i(A)$	0	1	1
$I_i(B)$	1	0	0
$I_i(C)$	0	0	0
$I_i(D)$	1	0	0
$I_i(E)$	0	0	0
$I_i(F)$	0	0	0

Sowohl als Ergebnis der Berechnung von $\mathcal{T} \uparrow \omega$ als auch der Iteration des Netzes erhalten wir die Interpretation $I = \{A\}$, die kleinstes Modell für unser Programm $\mathcal{P}$ ist. Dabei wird $\mathcal{T}_\mathcal{P}$ wird in konstanter Zeit, genauer gesagt in 2 Schritten, berechnet.

5.3 Aussagenlogische Programme und rekurrente Netzwerke

Im folgenden bezeichnen wir mit $\mathcal{P}$ ein normales aussagenlogisches Programm.

Satz 5.3 *Für jedes Programm $\mathcal{P}$ existiert ein 3–schichtiges vorwärtsgerichtetes Netz, daß den Operator $\mathcal{T}_{\mathcal{P}}$ berechnet.*

Beweis Sei $\mathcal{P}$ ein Programm. Dann besteht $\mathcal{P}$ aus einer endlichen Menge von Klauseln der Form

$$A \leftarrow L_1, \ldots, L_m$$

wobei A ein Atom ist und $L_1, \ldots, L_m$ sind Literale.

Sei I eine beliebige Interpretation von Atomen der Herbrand–Basis. Es gelte $\mathcal{T}_{\mathcal{P}}(I) = I'$.

Im folgenden steht

$\lvert P\rvert$	für die Anzahl der im Programm $\mathcal{P}$ vorkommenden Atome (Anzahl der Elemente der Herbrand–Basis für $\mathcal{P}$),
n	für die Anzahl von Klauseln in $\mathcal{P}$, die die Definition eines Atomes A in $\mathcal{P}$ darstellen
m_j	für die Anzahl von Literalen im Rumpf der j–ten Klausel der Definition eines Atomes.

Ohne Beschränkung der Allgemeinheit wählen wir ein Atom $A \in B_{\mathcal{P}}$. Die Definition von A lautet:

$$\begin{array}{lcl} A & \leftarrow & L_{11}, \ldots, L_{1m_1} \\ A & \leftarrow & L_{21}, \ldots, L_{2m_2} \\ & \vdots & \\ A & \leftarrow & L_{n1}, \ldots, L_{nm_n} \end{array}$$

mit $n \geq 0$ und $m_i \geq 0$.

Es gilt:

$$\begin{array}{lll} A \in \mathcal{T}_{\mathcal{P}}(I) & \text{, falls} & (I(L_{11}) = 1 \wedge \cdots \wedge I(L_{1m_1}) = 1) \vee \\ & & (I(L_{21}) = 1 \wedge \cdots \wedge I(L_{2m_2}) = 1) \vee \\ & & \vdots \\ & & (I(L_{n1}) = 1 \wedge \cdots \wedge I(L_{nm_n}) = 1) \vee \\ A \notin \mathcal{T}_{\mathcal{P}}(I) & \text{, sonst} & \end{array}$$

Im weiteren soll für ein Programm $\mathcal{P}$ ein 3–schichtiges Netz konstruiert werden, das die Funktion $\mathcal{T}_{\mathcal{P}}$ für das Programm realisiert. Dabei sollen die Einheiten des Netzes in der Eingabe– und in der Ausgabeschicht Elemente der Herbrand–Basis repräsentieren. Die Werte der Einheiten der Ausgabeschicht repräsentieren das Ergebnis der Funktionsanwendung von $\mathcal{T}_{\mathcal{P}}(I)$ auf die durch die Werte der Einheiten in der Eingabeschicht repräsentierte Herbrand–Interpretation I.

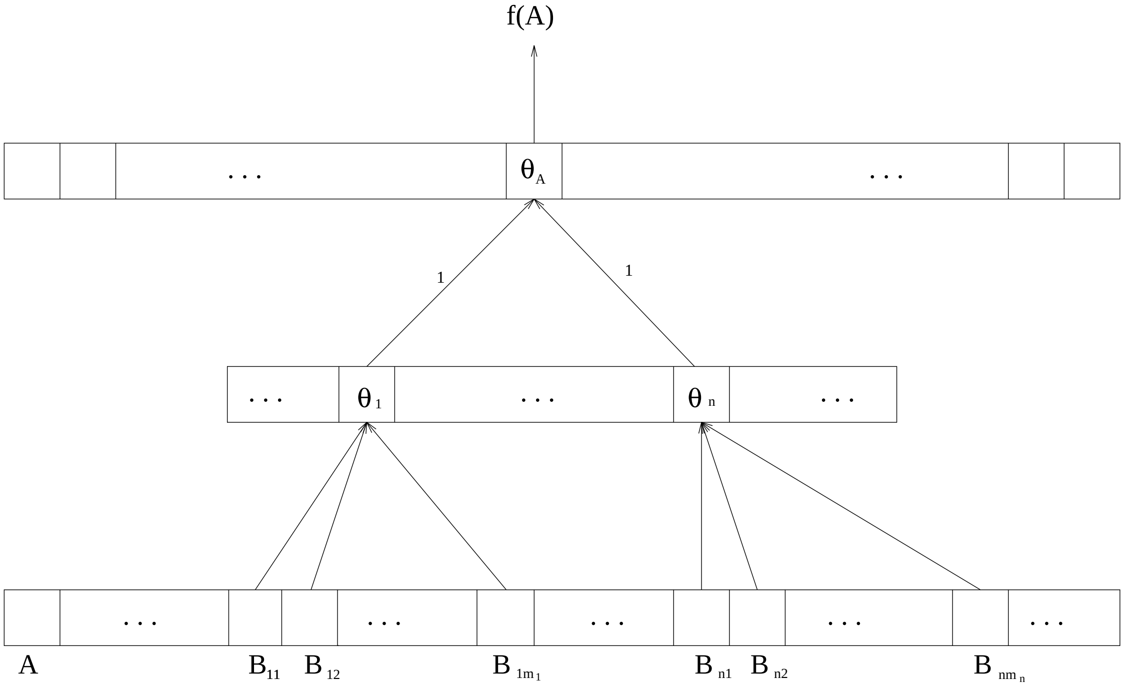

Abbildung 5.4: Konstruktion eines 3–schichtigen Netzes zur Berechnung der Funktion $\mathcal{T}_{\mathcal{P}}$ für das Programm $\mathcal{P}$. Die Gewichte der Kanten zwischen den Einheiten in der Eingabeschicht und der verborgenen Schicht werden in Abhängigkeit davon bestimmt, ob die von den Einheiten in der Eingabeschicht repräsentierten Grundatome im Körper der Klauseln von $\mathcal{P}$ als positive oder negative Literale auftreten.

- Das Netz besteht aus der Eingabeschicht, einer internen Schicht und einer Ausgabeschicht.
- Jede Einheit A in der Eingabe– und Ausgabeschicht repräsentiert ein Atom $A \in B_{\mathcal{P}}$. Die Werte dieser Einheiten repräsentieren eine Interpretation I dieser Atome.
- Gibt es in $\mathcal{P}$ eine Klausel der Form $A_j \leftarrow L_{j1}, \ldots, L_{jm_j} (1 \leq j \leq n)$, so existiert eine Kante zwischen jeder Einheit $L_{jl} (1 \leq l \leq m_j)$ und einer Einheit j in der internen Schicht. Ist L_{ji} ein positives Literal, wird die Kante mit dem Gewicht 1 bewertet. Für negative Literale L_{ji} wird die Kante mit dem Gewicht -1 bewertet.
- Beschränkt sich die Definition von A auf eine Klausel der Form $A_j \leftarrow$ (Fakt), so existiert eine Einheit j in der internen Schicht mit dem Schwellwert $\theta = -0.5$.
- Von jeder Einheit j in der internen Schicht existiert eine Kante zur Einheit A_j in der Ausgabeschicht, die mit dem Gewicht 1 bewertet ist.

- Jede Einheit in der internen und in der Ausgabeschicht realisiert die Funktionskomposition
$$\phi\psi : \{0,1\}^M \rightarrow \{0,1\}$$
mit
$$M = \begin{cases} |P| & \text{für die interne Schicht} \\ n & \text{für die Ausgabeschicht} \end{cases}$$
und
$$\begin{array}{llrcl} \psi & \ldots \text{ Aktivierungsfunktion} & \psi & : & \{0,1\}^M \rightarrow \mathbb{N} \\ & & \psi(\vec{w}, \vec{x}) & = & \sum_{i=1}^{n} \omega_i x_i \\ \phi & \ldots \text{ Ausgabefunktion} & \phi & : & \mathbb{N} \rightarrow \{0,1\} \\ & & \phi(n) & = & 1 \text{ , falls } n \geq \theta \text{ ,} \theta \in \mathbb{R} \\ & & \phi(n) & = & 0 \text{ , sonst} \end{array}$$

- Jede Einheit in der internen Schicht realisiert die Konjunktion (AND), d.h.
Sei $\vec{i} = (i_1, \ldots, i_n)$ der Vektor der Ausgaben der Einheiten $1, \ldots, n$ und $\vec{L} = (L_{j1}, \ldots, L_{jm_j})$ der Vektor der Ausgaben der Einheiten $L_{j1}, \ldots, L_{jm_j}$.
Für alle $j (1 \leq j \leq n)$ gilt
$$\begin{array}{rll} i_j = 1 & \text{gdw.} & \sum_{l=1}^{m_j} \omega_{jl} \cdot L_{jl} \geq \theta_j \\ & \text{gdw.} & L_{j1} = 1 \wedge \cdots \wedge L_{jm_j} = 1 \end{array}$$
mit $(k-1) < \theta_j < k$ und k ist die Anzahl der in der Einheit j endenden Kanten mit positivem Gewicht.

- Jede Einheit in der Ausgabeschicht realisiert die Disjunktion (OR), d.h.
$$\begin{array}{rll} f(A) = 1 & \text{gdw.} & \sum_{j=1}^{n} \omega_j \cdot i_j \geq \theta_a \\ & \text{gdw.} & i_1 = 1 \vee \cdots \vee i_n = 1 \end{array}$$
mit $0 < \theta_a < 1$

Das angegebene Netz realisiert für ein Atom $A \in B_{\mathcal{P}}$ die Funktion $f_{\mathcal{P}} : \{0,1\}^{|\mathcal{P}|} \rightarrow \{0,1\}$

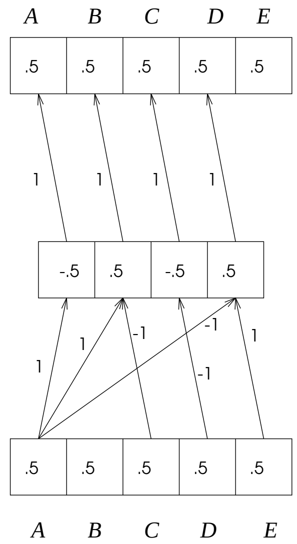

Abbildung 5.5: Konstruktion eines 3–schichtigen Netzes zur Berechnung der Funktion $\mathcal{T}_\mathcal{P}$ für das Programm $\mathcal{P} = \{A; B \leftarrow A, \neg C; C \leftarrow \neg D; D \leftarrow \neg A, E\}$.

$$
\begin{array}{lll}
f_\mathcal{P}(A) = 1 & \text{, falls} & (L_{11} = 1 \wedge \cdots \wedge L_{1m_1} = 1) \vee \\
 & & (L_{21} = 1 \wedge \cdots \wedge L_{2m_2} = 1) \vee \\
 & & \vdots \\
 & & (L_{n1} = 1 \wedge \cdots \wedge L_{nm_n} = 1) \\
f_\mathcal{P}(A) = 0 & \text{, sonst} &
\end{array}
$$

Die Ermittlung des Funktionswertes für alle Elemente der Herbrand–Basis wird durch eine Funktion $F_\mathcal{P} : \{0,1\}^{|\mathcal{P}|} \rightarrow \{0,1\}^{|\mathcal{P}|}$ realisiert, wobei die Berechnung für ein einzelnes Atom $A \in B_\mathcal{P}$ unabhängig von der Berechnung für alle anderen Elemente der Herbrand–Basis nach der Funktion $f_\mathcal{P}$ erfolgt.

Damit erhalten wir:

- $F_N(A) = 1 \equiv I'(A) = 1 \equiv A \in \mathcal{T}_\mathcal{P}(I)$
- $F_N(A) = 0 \equiv I'(A) = 0 \equiv A \notin \mathcal{T}_\mathcal{P}(I)$

■

Abbildung 5.4 zeigt die allgemeinste Form eines, nach der in Beweis von Satz 5.3 angegebenen Konstruktionsvorschrift, konstruierten Netzes für ein Programm $\mathcal{P}$.

Beispiel 5.4 Sei $\mathcal{P}$ das Programm aus Beispiel 3.14:

$$\begin{array}{lll} A. & & \\ B & \leftarrow & A, \neg C. \\ C & \leftarrow & \neg D. \\ D & \leftarrow & \neg A, E. \end{array}$$

Wir konstruieren ein Netz nach der im Beweis von Satz 5.3 angegebenen Vorschrift. Abbildung 5.5 zeigt das Ergebnis. ∘

Die Anwendung von Banachs Kontraktionstheorem, Satz 3.17 und Satz 5.3 liefert die folgende Proposition.

Proposition 5.5 *Sei $\mathcal{P}$ ein akzeptables Programm. Für $\mathcal{P}$ existiert ein 3–schichtiges rekurrentes Netz, so daß jede mit einer beliebigen Anfangsinterpretation startende Berechnung konvergiert und den kleinsten Fixpunkt des Operators $\mathcal{T}_{\mathcal{P}}$ liefert.*

Beispiel 5.6 Sei $\mathcal{P}$ das Programm aus Beispiel 5.4. Eine Stufenabbildung $|\_|$ definieren wir wie folgt:

$$\begin{array}{lllll} |\_| & : & B_{\mathcal{P}} & \rightarrow & \mathbb{N} \\ & & A & \mapsto & 1 \\ & & B & \mapsto & 4 \\ & & C & \mapsto & 3 \\ & & D & \mapsto & 2 \\ & & E & \mapsto & 1 \end{array}$$

Die Stufenabbildung $|\_|$ erfüllt die Anforderung an eine Stufenfunktion für ein akzeptables Programm[2]. Die Interpretation $I = \{A, B, C\}$ ist ein Modell für $\mathcal{P}$. Im Beispiel 3.14 haben wir bereits

$$\begin{array}{lll} NEG^*_{\mathcal{P}} & = & \{A, D, E\} \\ \mathcal{P}^- & = & \{A; D \leftarrow \neg A, E\} \end{array}$$

für unser Programm ermittelt. Die Einschränkung $I\downarrow_{NEG_{\mathcal{P}}^*} = \{A\}$ ist Modell für die Vervollständigung $\text{comp}(\mathcal{P}^-) = \{A; D \leftarrow \neg A, E\}$. Folglich ist das Programm $\mathcal{P}$ akzeptabel bezüglich der Stufenfunktion $|\_|$ und der Interpretation I.
Wenden wir nun Proposition 5.5 an, erhalten wir die Aussage, daß die Iteration des Netzes aus Abbildung 5.5, unabhängig von der Anfangsinterpretation, den kleinsten Fixpunkt von $\mathcal{T}_{\mathcal{P}}$ berechnet, das heißt, die Semantik von $\mathcal{P}$ liefert.

Die Iteration des Netzes mit der beliebig gewählten Anfangsinterpretation $I_0 = \{D\}$ liefert:

[2] siehe Definition 3.16

	Anfangsinterpretation I_0	$I_1 = \mathcal{T}_\mathcal{P}(I)$	$I_2 = \mathcal{T}_\mathcal{P}(I_1)$	$I_3 = \mathcal{T}_\mathcal{P}(I_2)$	$I_4 = \mathcal{T}_\mathcal{P}(I_3)$
$I_i(A)$	0	1	1	1	1
$I_i(B)$	0	0	1	0	0
$I_i(C)$	0	0	1	1	1
$I_i(D)$	1	0	0	0	0
$I_i(E)$	0	0	0	0	0

Als kleinsten Fixpunkt von $\mathcal{T}_\mathcal{P}$ erhalten wir die Interpretation $I = \{A, C\}$, die kleinstes Modell für unser Programm $\mathcal{P}$ ist. ∘

Im folgenden soll untersucht werden, ob eine umgekehrte Aussage zu Satz 5.3 getroffen werden kann. Nehmen wir an, es sei ein 3–schichtiges vorwärtsgerichtetes Netz $\mathcal{F}$ gegeben. Uns interessiert, ob zum Netz $\mathcal{F}$ ein Programm $\mathcal{P}$ gefunden werden kann, so daß $\mathcal{F}$ gerade den Operator $\mathcal{T}_\mathcal{P}$ für $\mathcal{P}$ berechnet.

Satz 5.7 *Für jedes 3–schichtige vorwärtsgerichtete Netz $\mathcal{F}$ existiert ein Programm $\mathcal{P}$, so daß $\mathcal{F}$ den Operator $\mathcal{T}_\mathcal{P}$ berechnet.*

Beweis Sei m die Anzahl der Einheiten in der Eingabeschicht und der Ausgabeschicht des Netzes $\mathcal{F}$. Der i–ten Einheit in der Ein– bzw. Ausgabeschicht sei die aussagenlogische Variable $A_i (1 \leq i \leq m)$ zugeordnet. Für den Eingabevektor $\vec{i}$ der Eingabeschicht existieren dann 2^m verschiedene Eingabevektoren. Sei $\mathcal{P}$ ein leeres Programm. Für jeden Eingabevektor v führe man die folgenden Schritte aus. Wird $\vec{v}$ als Eingabevektor an der Eingabeschicht des Netzes $\mathcal{F}$ angelegt und aktiviert nach Propagierung durch die verborgene Schicht des Netzes die i–te Einheit in der Ausgabeschicht $(1 \leq i \leq m)$, so wird das Programm $\mathcal{P}$ um die Klausel $A_i \leftarrow L_1, \ldots, L_m$ erweitert, wobei $L_k = A_K$, falls $v_k = 1$ und $L_k = \neg A_k$, falls $v_k = 0$. Der Wert v_k bezeichnet dabei die k–te Komponente des Vektors $\vec{v}$ $(1 \leq k \leq m)$. Offensichtlich berechnet $\mathcal{F}$ dann $\mathcal{T}_\mathcal{P}$. ■

Beispiel 5.8 Gegeben sei das in Abbildung 5.6 gezeigte Netz.

Wendet man den oben genannten Algorithmus an, erhält man die Wertebelegung der folgenden Tabelle, in der der Wert der Eingabeeinheiten 1 und 2 sowie das Potential p und den Wert v der Einheiten 3 bis 7 angegeben sind.

1	2	3		4		5		6		7	
		p	v	p	v	p	v	p	v	p	v
0	0	0	0	0	1	0	0	0	1	-1	0
0	1	1.5	1	.3	1	.8	1	1.8	1	.7	1
1	0	1	1	-1	0	$-.5$	0	2	1	.7	1
1	1	2.5	1	$-.7$	0	.3	0	2	1	.7	1

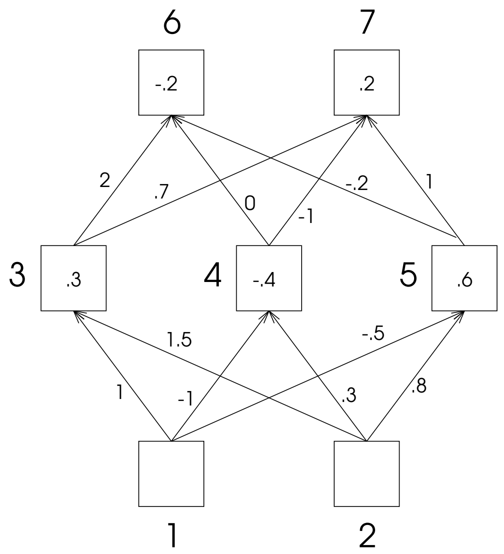

Abbildung 5.6: Das 3–schichtige vorwärtsgerichtete Netz berechnet den Operator $\mathcal{T}_\mathcal{P}$ für das Programm $\mathcal{P} = \{A_1; A_2 \leftarrow A_1; A_2 \leftarrow \neg A_1, A_2\}$. Die Zahlen in den Einheiten stehen für die Schwellwerte der Einheiten.

Das entsprechende Programm

$$\begin{array}{lll} \mathcal{P}: & A_1 \leftarrow \neg A_1, \neg A_2 & \\ & A_1 \leftarrow \neg A_1, A_2 & A_2 \leftarrow \neg A_1, A_2 \\ & A_1 \leftarrow A_1, \neg A_2 & A_2 \leftarrow A_1, \neg A_2 \\ & A_1 \leftarrow A_1, A_2 & A_2 \leftarrow A_1, A_2 \end{array}$$

kann man zum Programm $\mathcal{P} = \{A_1; A_2 \leftarrow A_1; A_2 \leftarrow \neg A_1, A_2\}$ vereinfachen.

Das Programm $\mathcal{P}$ ist akzeptabel bezüglich der Stufenabbildung $|\_|$, die A_1 nach 1 und A_2 nach 2 abbildet und dem Modell $I = \{A_1, A_2\}$. Erweitert man das in Abbildung 5.6 gezeigte Netz zu einem rekurrenten Netz, so konvergiert das Netz, unabhängig von der Belegung des Eingabevektors, gegen den Zustand mit $v_6 = v_7 = 1$ und $v_1 = v_2 = v_3 = v_4 = v_5 = 0$. ∘

Zu einem rekurrenten Netz $\mathcal{N}$, dessen vorwärtsgerichtetes Teilnetz $\mathcal{F}$ die Funktion f berechnet, findet man das zu diesem Netz gehörige Programm $\mathcal{P}$, indem man eine Wertetabelle mit den Belegungen der Einheiten des Netzes aufstellt. Das erhaltene Programm kann in den meisten Fällen noch stark vereinfacht werden (siehe Beispiel 5.8). Das vereinfachte Programm kann man nutzen, um das Netz, daß die Funktion f berechnet, zu minimieren.

Im folgenden wollen wir uns der Fragestellung zuwenden, ob rekurrente Netze, die immer in einen stabilen Zustand gelangen, die Klasse der akzeptablen Programme definieren, das heißt, ob sie $\mathcal{T}_\mathcal{P}$ für ein akzeptables Programm $\mathcal{P}$ berechnen.

Proposition 5.9 *Konvergiert ein 3–schichtiges rekurrentes Netzwerk $\mathcal{N}$ in einen stabilen Zustand, so ist die Funktion f, die durch das 3–schichtige vorwärtsgerichtete Teilnetz $\mathcal{F}$ von $\mathcal{N}$ berechnet wird, eine Kontraktion auf dem metrischen Raum der Interpretationen für $\mathcal{P}$.*

Beweis Sei m die Anzahl der Einheiten in der Ein– bzw. Ausgabeschicht des Netzes $\mathcal{N}$ und f die durch das 3–schichtige Teilnetz $\mathcal{F}$ von $\mathcal{N}$ berechnete Funktion. Sei $\mathcal{M}$ die Menge von binären Vektoren der Länge m.

Wir definieren im folgenden eine Relation $\preceq$ auf Elementen v und v' aus $\mathcal{M}$.

- Eine Relation $\sqsubset$ auf Elementen v und v' aus $\mathcal{M}$ ist wie folgt definiert:

$$v \sqsubset v' \text{ gdw. } f(v) = v' \quad \wedge \quad v \neq v'$$

- Die Relation $\prec$ ist als die transitive Hülle der Relation $\sqsubset$ definiert.
- Die Relation $\preceq$ ist als die reflexive Hülle der Relation $\prec$ definiert.

Aufgrund unserer Konstruktion ist $\preceq$ reflexiv, transitiv und antisymmetrisch, das heißt, $\preceq$ ist eine partielle Ordnung.

Für jedes $v \in \mathcal{M}$ definieren wir die Funktion $g(v) \in \mathbb{N}$ mit folgenden Eigenschaften:

$$\begin{array}{lll} g(v) < g(v') & \text{wenn} & v \prec v' \\ g(v) = g(v') & \text{gdw.} & v = v' \end{array}$$

Den Abstand zweier Elemente aus $\mathcal{M}$ definieren wir als

$$\begin{array}{lcll} d(v, v') & = & 0 & \text{wenn } v = v' \\ d(v, v') & = & (1/2)^{\min\{g(v), g(v')\}} & \text{sonst} \end{array}$$

Da das Netz laut Voraussetzung für jeden Anfangszustand konvergiert, hat jede Folge

$$v_0, f(v_0), f(f(v_0)), \ldots$$

einen Grenzwert, den wir im folgenden mit v^* bezeichnen. Für jede Interpretation $v_0 \neq v^*$ gilt deshalb $g(v) < g(v^*)$.

Um zu zeigen, daß f eine Kontraktion ist, müssen wir für beliebige v und v' aus $\mathcal{M}$ zeigen, daß

$$d(f(v), f(v')) < k \cdot d(v, v')$$

für ein k mit $0 < k \leq 1$ gilt.
Wir unterscheiden die folgenden Fälle.

1. $v = v^*$ und $v' \neq v^*$
   Es gilt
   $$d(f(v^*), f(v')) = (1/2)^{\min\{g(f(v^*)), g(f(v'))\}}$$
   und $g(f(v^*)) > g(f(v'))$, da v^* Fixpunkt ist. Daraus folgt
   $$d(f(v^*), f(v')) = (1/2)^{g(f(v'))}$$
   Außerdem gilt
   $$d(v^*, v') = (1/2)^{\min\{g(v^*), g(v')\}}$$
   und $g(v^*) > g(v')$, da v^* Fixpunkt ist , woraus
   $$d(v^*, v') = (1/2)^{g(v')}$$
   folgt.
   Wegen $v' \neq f(v')$, da v' kein Fixpunkt ist, gilt $g(f(v')) > g(v')$. Daraus folgt
   $$(1/2)^{g(f(v'))} < (1/2)^{g(v')}$$
   das heißt, wir können ein k mit $0 < k \leq 1$ finden, so daß $d(f(v), f(v)') < k \cdot d(v, v')$ gilt.
2. $v' = v^*$ und $v \neq v^*$
   Dieser Fall kann analog zu Fall 1 behandelt werden.
3. $v' \neq v^*$ und $v \neq v^*$
   Da weder v noch v' Fixpunkt ist, gilt $g(v) < g(f(v)$ und $g(v') < g(f(v')$. Folglich gilt $\min\{g(f(v)), g(f(v'))\} > \min\{g(v), g(v')\}$ und damit
   $$d(f(v), f(v')) = (1/2)^{\min\{g(f(v)), g(f(v'))\}} < (1/2)^{\min\{g(v), g(v')\}} = d(v, v')$$

Damit haben wir gezeigt, daß f die Anforderungen an eine Kontraktion erfüllt. ■

Daß es sich bei dem metrischen Raum $(\mathcal{M}, d)$ um einen vollständigen metrischen Raum handelt, konnte nicht gezeigt werden, wird aber vermutet.

Satz 5.10 *Sei $\mathcal{N}$ ein 3–schichtiges rekurrentes Netz, $\mathcal{F}$ sei das 3–schichtige vorwärtsgerichtete Teilnetz von $\mathcal{N}$ und $\mathcal{P}$ sei das zu $\mathcal{F}$ gehörige Programm. Falls $\mathcal{N}$ immer in genau einen stabilen Zustand konvergiert, dann hat $\mathcal{T}_\mathcal{P}$ einen eindeutig bestimmten Fixpunkt.*

Beweis Sei f die Funktion, die vom Netz $\mathcal{F}$ berechnet wird. Mit Satz 5.3 gilt $f(x) = \mathcal{T}_\mathcal{P}(x)$ für ein beliebiges x. Das Netz $\mathcal{N}$ berechnet die Iteration der Funktion f, also die Iteration des Operators $\mathcal{T}_\mathcal{P}$ Wenn das Netz $\mathcal{N}$ unabhängig von einer Anfangsbelegung immer in einen stabilen Zustand v_f konvergiert, konvergiert auch die Iteration von $\mathcal{T}_\mathcal{P}$ unabhängig von einer Anfangsbelegung gegen ein und denselben Wert $v_\mathcal{T} = v_f$. Damit ist $v_\mathcal{T}$ eindeutig bestimmter Fixpunkt von $\mathcal{T}_\mathcal{P}$. ■

In allen von uns untersuchten Beispielen konvergieren nur Netze, die den Operator $\mathcal{T}_{\mathcal{P}}$ für akzeptable oder definite Programme berechnen. Eine entsprechende Aussage wird ebenfalls vermutet.

Bisher liefert unser Berechnungsmodell eine Möglichkeit, die Semantik akzeptabler Programme zu berechnen. Wie bereits im Abschnitt 3.4 erwähnt, definiert diese Klasse eine umfangreiche Klasse von Programmen, für die zur Bestimmung ihrer Semantik der Operator $\mathcal{T}_{\mathcal{P}}$ verwendet werden kann. Auch nicht stratifizierte Programme gehören zu dieser Klasse, z.B. das erwähnte Programm $\mathcal{P}(\mathcal{G})$.

Man kann jedoch nicht zeigen, daß jedes definite Programm akzeptabel ist. Das Beispiel 3.21 auf Seite 39 zeigt ein solches Programm.

Starten wir jedoch die Iteration unseres Berechnungsmodelles nicht mit einer beliebigen Interpretation, sondern mit der Interpretation $I = \emptyset$, so liefert unser Berechnungsmodell auch für jedes definite Programm $\mathcal{P}$ die Semantik. Die Konvergenz des Netzes ist dann durch die Stetigkeit von $\mathcal{T}_{\mathcal{P}}$ für definite Programme $\mathcal{P}$ gesichert.

5.4 Aufwandsabschätzung

Platzkomplexität

Über ein nach der im Abschnitt 5.3 angegebenen Vorschrift konstruiertes Netz $\mathcal{F}$ für ein Programm $\mathcal{P}$ können bezüglich der Größe des Netzes die folgenden Aussagen getroffen werden.

- In der internen Schicht befinden sich genau so viele Einheiten, wie es Klauseln in $\mathcal{P}$ gibt. Damit verhält sich die Anzahl der Einheiten in der verborgenen Schicht linear zur Größe von $\mathcal{P}$
- Die Anzahl von Einheiten in der Eingabe- beziehungsweise in der Ausgabeschicht entspricht der Anzahl von Elementen in der Herbrand-Basis für $\mathcal{P}$.
- Die Anzahl der Kanten zwischen der Eingabeschicht und der verborgenen Schicht entspricht der Anzahl von Grundatomen im Körper der Klauseln von $\mathcal{P}$
- Die Anzahl der Kanten zwischen der verborgenen und der Ausgabeschicht entspricht der Anzahl von Klausel in $\mathcal{P}$.

Bezeichne n die Anzahl von Klauseln und m die Anzahl von Grundatomen, die in $\mathcal{P}$ vorkommen. Dann ist in einem Netz $\mathcal{F}$, das mit dem Programm $\mathcal{P}$ korrespondiert, die Anzahl von Kanten durch $O(m \times n)$ und die Anzahl von Einheiten durch $O(m \times n)$ beschränkt.

Außerdem bestimmt die Anzahl der Kanten des Netzes maßgeblich den Berechnungsaufwand für $\mathcal{T}_{\mathcal{P}}$.

Bemerkung Treten in $\mathcal{P}$ Klauseln der Form

$$A \leftarrow B \qquad (m = 1)$$

oder Fakten

$$A \leftarrow \qquad (m = 0)$$

auf, so kann die Größe des Netzes verringert werden. Für jede Klausel dieser Art kann eine Einheit in der internen Schicht entfernt werden und eine Kante von der Einheit B bzw. A in der Eingabeschicht zur Einheit A in der Ausgabeschicht eingefügt werden. Damit verändert sich die Architektur unseres Netzes. Es liegt keine Schichtenstruktur mehr vor.

Zeitkomplexität

Sei S ein Problem, dessen schnellster sequentieller Algorithmus in einer Zeit proportional zu $T(n)$ läuft. Ein Algorithmus A für S , der in paralleler Zeit $t(n)$ mit $p(n)$ Prozessoren läuft, ist *optimal*[3], falls:

- $t(n) = polylog(n)$ und
- $p(n) \cdot t(n) = O(T(n))$,

Ein optimaler paralleler Algorithmus erreicht einen hohen Grad an Parallelität in optimaler Art und Weise.

Der Operator $\mathcal{T}_{\mathcal{P}}(I)$ für eine an der Eingabeschicht des Netzes anliegende Interpretation I wird in konstanter Zeit, nämlich in zwei Schritten (Propagierung zur verborgenen Schicht und Propagierung zur Ausgabeschicht), berechnet. Bezeichne n die Anzahl von Klauseln und m die Anzahl von Grundatomen, die in $\mathcal{P}$ vorkommen. Dann ist die Zeit, die ein sequentielles Berechnungsmodell benötigt, um $\mathcal{T}_{\mathcal{P}}(I)$ zu berechnen, durch $O(n \times m)$ beschränkt[4]. Unser paralleles Modell benötigt mit $O(n \times m)$ Prozessoren konstante Zeit und ist damit beschränkt durch $O(n \times m)$ und folglich *optimal.*

Die Zeit, die vom Netz benötigt wird, um in einen stabilen Zustand zu gelangen, entspricht im schlechtesten Fall der Zeit, die von einer sequentiellen Maschine benötigt wird, um den kleinsten Fixpunkt von $\mathcal{T}_{\mathcal{P}}$ zu berechnen. Nutzt man zum Beispiel den von Dowling und Gallier in [DG84] (siehe auch [Scu90]) angegebenen linearen Algorithmus zum Test der Erfüllbarkeit aussagenlogischer HORN-Formeln, benötigt die Berechnung $O(n)$ Schritte, wobei n die Anzahl von Vorkommen aussagenlogischer Variable in der Formel bezeichnet.

Die Zeit, die von unserem Modell benötigt wird, um in einem stabilen Zustand zu gelangen ist $3n$ im schlechtesten Fall und folglich linear zur Anzahl von Klauseln, die in $\mathcal{P}$ vorkommen.

[3] siehe [KR90]

[4] Unter der Annahme, daß kein Literal mehr als einmal im Körper einer Klausel auftritt.

6. Mehrwertige Logiken

Im Abschnitt 5.3 wurde ein neues Berechnungsmodell für aussagenlogische Programme in der zweiwertigen Logik vorgestellt. Im folgenden soll gezeigt werden, daß dieses Modell für mehrwertigen Logiken erweitert werden kann.

6.1 Definitionen und Notation

Eine Interpretation ist eine Funktion, die Grundatome eines Programmes $\mathcal{P}$ auf die Menge der Wahrheitswerte W abbildet. Abhängig von der Menge der Wahrheitswerte, auf die die Grundatome abgebildet werden dürfen, erhalten wir unterschiedliche Interpretationen.

Definition 6.1

- *Eine* 2–Interpretation *auf Elementen aus* $B_{\mathcal{P}}$ *ist eine Interpretation, die jedes Atom* $A \in B_{\mathcal{P}}$ *auf die Wahrheitswerte* **t** *und* **f** *abbildet* ($W = \{\mathbf{t}\ , \mathbf{f}\}$).
- *Eine* 3–Interpretation *auf Elementen aus* $B_{\mathcal{P}}$ *ist eine Interpretation, die jedes Atom* $A \in B_{\mathcal{P}}$ *auf die Wahrheitswerte* **t**, **f** *und* **w** *abbildet* ($W = \{\mathbf{t}\ , \mathbf{f}, \mathbf{w}\}$).
- *Eine* 3^*–Interpretation *auf Elementen aus* $B_{\mathcal{P}}$ *ist eine Interpretation, die jedes Atom* $A \in B_{\mathcal{P}}$ *auf die Wahrheitswerte* **t**, **f** *und* **u** *abbildet* ($W = \{\mathbf{t}\ , \mathbf{f}, \mathbf{u}\}$).
- *Eine* 4–Interpretation *auf Elementen aus* $B_{\mathcal{P}}$ *ist eine Interpretation, die jedes Atom* $A \in B_{\mathcal{P}}$ *auf die Wahrheitswerte* **t**, **f**, **w** *und* **u** *abbildet* ($W = \{\mathbf{t}\ , \mathbf{f}, \mathbf{w}, \mathbf{u}\}$).

Dabei repräsentieren die Wahrheitswerte die Bedeutungen:

- **t** *wahr*
- **f** *falsch*
- **u** *undefiniert*
- **w** *widersprüchlich*

A	$\neg A$
t	**f**
f	**t**
u	**u**

A	B	$A \wedge B$
t	**t**	**t**
t	**f**	**f**
t	**u**	**u**
f	**t**	**f**
f	**f**	**f**
f	**u**	**f**
u	**t**	**u**
u	**f**	**f**
u	**u**	**u**

A	B	$A \vee B$
t	**t**	**t**
t	**f**	**t**
t	**u**	**t**
f	**t**	**t**
f	**f**	**f**
f	**u**	**u**
u	**t**	**t**
u	**f**	**u**
u	**u**	**u**

Tabelle 6.1: Definition der Semantik der Junktoren $\neg, \wedge$ und $\vee$ in der 3^*–Logik.

Aufbauend auf diese Interpretationen kann man nun Erfüllbarkeit, 2–, 3–, 3^*– und 4–Folgerung und damit eine 2–, 3–, 3^*– und 4–Logik definieren.

Eine zweiwertige Interpretation eines Grundatomes können wir mit einem Bit kodieren. Der Wert 0 repräsentiert den Wert **t**, der Wert 1 den Wert **f**. Um eine 3–Interpretation zu kodieren benötigen wir 2 Bit. Dabei wird die Belegung 00 verboten. Für die Kodierung einer 3^*–Interpretation benötigen wir ebenfalls 2 Bit. Die Belegung 11 wird verboten. Bei der Kodierung einer 4–Interpretation sind alle Belegungen der 2 Bit erlaubt.

Im folgenden betrachten eine *3^*–Logik*.

Eine Herbrand–Interpretation in dieser Logik (genannt *3^*–wertige Herbrand–Interpretation*) definieren wir als ein Paar (I^+, I^-) disjunkter Mengen von Grundatomen.

- Gegeben sei eine solche Interpretation $I = (I^+, I^-)$. Für ein Grundatom A gilt $I(A) = \mathbf{t}$, falls $A \in I^+$ und $I(A) = \mathbf{f}$ falls $A \in I^-$. Wenn $A \notin I^+$ $A \notin I^-$, dann gilt $I(A) = \mathbf{u}$.
- Jeder binäre Junktor liefert den Wert **t** (bzw. **f**), wenn er diesen Wert in der 2–wertigen Logik für alle möglichen Substitutionen von **u**'s durch **t** oder **f** liefert. Anderenfalls liefert er den Wert **u**.
- Gegeben sei eine Formel Φ und eine 3^*–wertige Herbrand–Interpretation I. Wir erweitern die Interpretation I auf Grundatomen zu einer Interpretation I^* auf Formeln mit der in Tabelle 6.1 definierten Semantik der Junktoren $\neg, \wedge$ und $\vee$ in der 3^*–Logik. Φ ist *wahr*, falls $I^*(\Phi) = \mathbf{t}$ und Φ ist *falsch*, falls $I^*(\Phi) = \mathbf{f}$.
- $I = (I^+, I^-)$ heißt *totale 3^*–wertige Herbrand–Interpretation* für ein Programm P, falls $I^+ \cup I^- = B_P$.
- Gegeben sei ein normales Programm $\mathcal{P}$. Die Menge aller 3^*–wertigen Herbrand–Interpretationen für $\mathcal{P}$ formt einen vollständigen Verband mit der Ordnung $\subseteq$, die definiert ist durch

$$I \subseteq J \quad \text{gdw.} \quad I^+ \subseteq J^+ \quad \text{und} \quad I^- \subseteq J^-.$$

Die Funktion $\Phi_{\mathcal{P}}$ auf dem vollständigen Verband der 3–wertigen Herbrand–Interpretationen für $\mathcal{P}$ ist definiert durch

$$\Phi_{\mathcal{P}}(I) = (I^+, I^-)$$

mit

$$\begin{array}{rcl} I^+ & = & \{A \mid \exists A \leftarrow L_1, \ldots, L_k \text{ in ground}(\mathcal{P}) \text{ mit } (L_1 \wedge \cdots \wedge L_k) \text{ ist } \mathbf{t} \text{ in } I\} \\ I^- & = & \{A \mid \forall A \leftarrow L_1, \ldots, L_k \text{ in ground}(\mathcal{P}) \text{ mit } (L_1 \wedge \cdots \wedge L_k) \text{ ist } \mathbf{f} \text{ in } I\} \end{array}$$

Der kleinste Fixpunkt von $\Phi_{\mathcal{P}}$ liefert die Semantik von $\mathcal{P}$. Die Existenz und Eindeutigkeit eines kleinsten Fixpunktes von $\Phi_{\mathcal{P}}$ auf dem vollständigen Verband der 3^*–wertigen Herbrand–Interpretationen für $\mathcal{P}$ wird von Fitting in [Fit85] gezeigt.

6.2 Ein Berechnungsmodell für mehrwertige Logiken

Die Erweiterung unseres Berechnungsmodelles für eine mehrwertige Logik ist mit geringem Aufwand möglich. Zu einem Programm $\mathcal{P}$ in einer mehrwertigen Logik erhält man ein Netz, daß den Operator $\Phi_{\mathcal{P}}$ berechnet, indem die Ein– und Ausgabeschicht des Netzes, im Vergleich zur zweiwertigen Logik, erweitert werden. Für jede aussagenlogische Variable A, die in der Herbrand–Basis von $\mathcal{P}$ auftritt, werden in der Ein– bzw. Ausgabeschicht des Netzes 2 Einheiten realisiert. Diese repräsentieren die Variable A bzw. ihre Negation $\neg A$. Diese Erweiterung erlaubt die Darstellung der Wahrheitswerte *undefiniert* bzw. *widersprüchlich*. Analog erfolgt eine Erweiterung der verborgenen Schicht.

Für eine 3^*–Logik soll im folgenden die Konstruktion unseres Berechnungsmodelles gezeigt werden.

Satz 6.2 *Für jedes Programm $\mathcal{P}$ in einer 3^*–Logik existiert ein 3–schichtiges vorwärtsgerichtetes Netz, das den Operator $\Phi_{\mathcal{P}}$ berechnet.*

Beweis Sei I eine beliebige 3^*–wertige Herbrand–Interpretation von Atomen der Herbrand–Basis.
Sei $\mathcal{P}$ ein normales Programm in einer 3^*–wertigen Aussagenlogik, das heißt, eine Menge von Klauseln der Form

$$A \leftarrow L_1, \ldots, L_m$$

wobei A ein Atom ist und $L_1, \ldots, L_m$ sind Literale. Ohne Beschränkung der Allgemeinheit wählen wir ein Atom $A \in B_{\mathcal{P}}$. Die Definition von A in $\mathcal{P}$ lautet:

$$\begin{array}{rcl} A & \leftarrow & L_{11}, \ldots, L_{1m_1} \\ A & \leftarrow & L_{21}, \ldots, L_{2m_2} \\ & \vdots & \\ A & \leftarrow & L_{n1}, \ldots, L_{nm_n} \end{array}$$

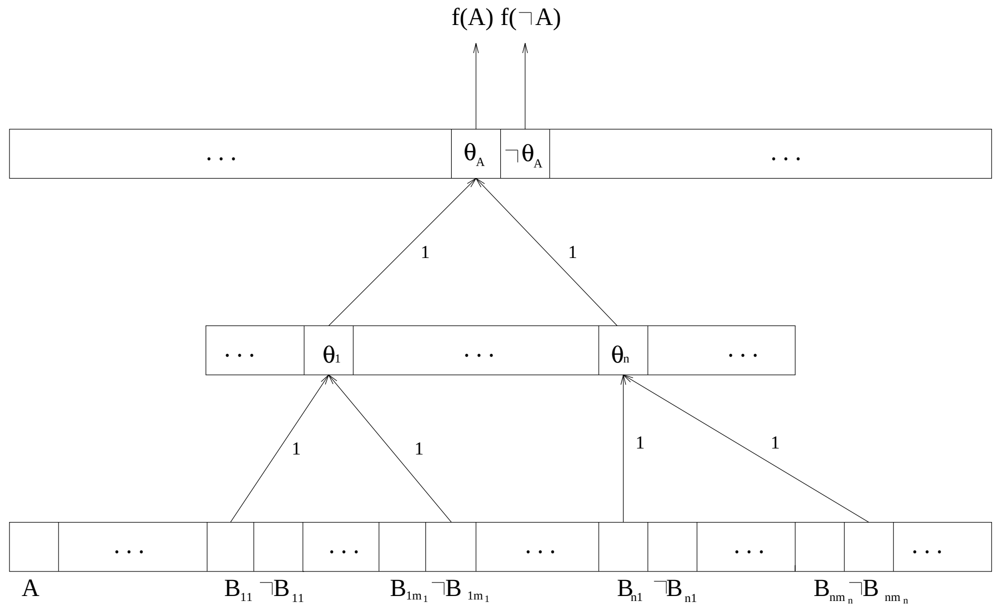

Abbildung 6.1: Konstruktion eines Netzes zur Berechnung der Abbildung $\mathcal{T}_{\mathcal{P}}$ für ein Programm $\mathcal{P}$ in einer dreiwertigen Logik

mit $n \geq 0$ und $m_i \geq 0$.

Es gilt

$$\begin{array}{lll}
A \in I^+ & \text{, falls} & (I(L_{11}) = \mathbf{t} \wedge \cdots \wedge I(L_{1m_1}) = \mathbf{t}) \vee \\
 & & (I(L_{21}) = \mathbf{t} \wedge \cdots \wedge I(L_{2m_2}) = \mathbf{t}) \vee \\
 & & \vdots \\
 & & (I(L_{n1}) = \mathbf{t} \wedge \cdots \wedge I(L_{nm_n}) = \mathbf{t}) \\
A \in I^- & \text{, falls} & (I(L_{11}) = \mathbf{f} \wedge \cdots \wedge I(L_{1m_1}) = \mathbf{f}) \wedge \\
 & & (I(L_{21}) = \mathbf{f} \wedge \cdots \wedge I(L_{2m_2}) = \mathbf{f}) \wedge \\
 & & \vdots \\
 & & (I(L_{n1}) = \mathbf{f} \wedge \cdots \wedge I(L_{nm_n}) = \mathbf{f})
\end{array}$$

Ein Netz wird wie folgt konstruiert:

- Das Netz besteht aus der Eingabeschicht, einer internen Schicht und einer Ausgabeschicht.
- Jeder Wert einer Einheit in der Eingabe– und Ausgabeschicht repräsentiert die

Interpretation eines Literales A bzw. $\neg A$ ($A \in B_{\mathcal{P}}$) bezüglich einer 3^*–wertigen Herbrand–Interpretation $I = (I^+, I^-)$:

$$A = \begin{cases} 1 & \text{falls} \quad I(A) = \mathbf{t} \\ 0 & \text{sonst} \end{cases}$$

$$\neg A = \begin{cases} 1 & \text{falls} \quad I(A) = \mathbf{f} \\ 0 & \text{sonst} \end{cases}$$

- Gibt es in P eine Klausel der Form

  $$A_j \leftarrow B_{j1}, \ldots, B_{jk}, \neg B_{j\,k+1}, \ldots, \neg B_{jm_j},$$

  wobei die B_i Atome repräsentieren, so existiert eine Kante mit dem Gewicht 1 zwischen jeder Einheit $B_1, \ldots, B_k$ bzw. $\neg B_{k+1}, \ldots, B_{m_j}$ und der Einheit j in der internen Schicht.

- Beschränkt sich die Definition von A auf eine Klausel der Form $A_j \leftarrow$ (Fakt), so existiert eine Einheit j in der internen Schicht mit dem Schwellwert $\theta = -0.5$.

- Von jeder Einheit $j (1 \leq j \leq n)$ in der internen Schicht existiert eine Kante zur Einheit A in der Ausgabeschicht, die mit dem Gewicht 1 bewertet ist.

- Jede Einheit in der internen und in der Ausgabeschicht realisiert die Funktionskomposition

  $$\phi\psi : \{0,1\}^M \rightarrow \{0,1\}$$

  mit

  $$M = \begin{cases} |P| & \text{für die interne Schicht} \\ n & \text{für die Ausgabeschicht} \end{cases}$$

  ψ ... Aktivierungsfunktion

  $$\psi : \{0,1\}^M \rightarrow \mathbb{N}$$
  $$\psi\,(\mathbf{w},\mathbf{x}) = \sum_{i=1}^{n} \omega_i x_i$$

  und ϕ ... Ausgabefunktion

  $$\phi : \mathbb{N} \rightarrow \{0,1\}$$
  $$\begin{aligned} \phi(n) &= 1 \text{ falls } n \geq \theta \\ \phi(n) &= 0 \text{ sonst} \end{aligned}$$

- Die Einheiten in der verborgenen Schicht realisieren die Konjunktion (AND), das heißt:

  Sei $\mathbf{i} = (i_1, \ldots, i_n)$ der Vektor der Ausgaben der Einheiten $1, \ldots, n$

  Für alle $j (1 \leq j \leq n)$ gilt

$$
\begin{aligned}
i_j = 1 \quad & \text{gdw.} \quad \sum_{l=1}^{m_j} \omega_{jl} \cdot L_{jl} \geq \theta_j \\
& \text{gdw.} \quad L_{j1} = 1 \wedge \cdots \wedge L_{jm_j} = 1
\end{aligned}
$$

mit $m_j - 1 < \theta_j < m_j$.

- In der Ausgabeschicht gibt es 2 Klassen von Einheiten. Zu jeder Einheit A gibt es eine Einheit $\neg A$.

  Die Einheiten A realisieren die Disjunktion (OR):

$$
\begin{aligned}
f(A) = 1 \quad & \text{gdw.} \quad \sum_{j=1}^{n} \omega_j \cdot i_j \geq \theta_a \\
& \text{gdw.} \quad (L_{11} = 1 \wedge \cdots \wedge L_{1m_1} = 1) \vee \\
& \qquad\quad\; (L_{21} = 1 \wedge \cdots \wedge L_{2m_2} = 1) \vee \\
& \qquad\quad\; \vdots \\
& \qquad\quad\; (L_{n1} = 1 \wedge \cdots \wedge L_{nm_n} = 1)
\end{aligned}
$$

  mit $0 < \theta_a < 1$.

Das so konstruierte Netz realisiert für jedes Atom $A \in B_{\mathcal{P}}$ die Funktion $f_{\mathcal{P}} : \{0,1\}^{|\mathcal{P}|} \rightarrow \{0,1\}$

$$
\begin{aligned}
f_{\mathcal{P}}(A) = 1 \quad & \text{gdw.} \quad (L_{11} = 1 \wedge \cdots \wedge L_{1m_1} = 1) \vee \\
& \qquad\quad\; (L_{21} = 1 \wedge \cdots \wedge L_{2m_2} = 1) \vee \\
& \qquad\quad\; \vdots \\
& \qquad\quad\; (L_{n1} = 1 \wedge \cdots \wedge L_{nm_n} = 1) \\
f_{\mathcal{P}}(A) = 0 \quad & \text{sonst} \\
f_{\mathcal{P}}(\neg A) = 0 &
\end{aligned}
$$

und es gilt

- $f_{\mathcal{P}}(A) = 1 \equiv A \in I^+$
- $f_{\mathcal{P}}(\neg A) = 1 \equiv A \in I^-$

■

Abbildung 6.2: Ein Netz zur Berechnung der Funktion $\Phi_{\mathcal{P}}$ für das Programm $\mathcal{P} = \{C \leftarrow A \wedge \neg B; D \leftarrow C \wedge E; D \leftarrow \neg C\}$ in einer 3^*–wertigen Logik.

Beispiel 6.3 Wir betrachten das Programm $\mathcal{P}$:

$$\begin{array}{lcl} C & \leftarrow & A \wedge \neg B. \\ D & \leftarrow & C \wedge E. \\ D & \leftarrow & \neg C. \end{array}$$

Abbildung 6.2 zeigt das nach der im Beweis von Satz 6.2 angegebenen Konstruktionsvorschrift erstellte Netz. Starten wir die Iteration des Netzes mit der Interpretation $I = (\{A\}, \{C\})$, erhalten wir die folgende Wertetabelle. $\circ$

	v_0	$\Phi_{\mathcal{P}}(v_0)$	$\Phi_{\mathcal{P}}(\Phi_{\mathcal{P}}(v_0))$
$v(A)$	1	0	0
$v(\neg A)$	0	0	0
$v(B)$	0	0	0
$v(\neg B)$	0	0	0
$v(C)$	0	1	0
$v(\neg C)$	1	0	0
$v(D)$	0	1	0
$v(\neg D)$	0	0	0
$v(E)$	0	0	0
$v(\neg E)$	0	0	0

7. Beziehungen zur stabilen Modellsemantik

Im Kapitel 5.3 haben wir rekurrente Netze betrachtet, die in einen eindeutig bestimmten stabilen Zustand gelangen. Offensichtlich gibt es auch rekurrente Netze, die nicht in einen stabilen Zustand gelangen beziehungsweise für die es mehrere stabile Zustande gibt.

Beispiel 7.1 Abbildung 7.1 zeigt ein 3–schichtiges vorwärtsgerichtete Netz. Wenden wir Satz 5.7 an und vereinfachen das erhaltene Programm, erhalten wir

$$\begin{array}{ll} \mathcal{P}: & P \leftarrow \neg Q. \\ & Q \leftarrow \neg P. \end{array}$$

$\mathcal{P}$ ist nicht akzeptabel, da keine Stufenabbildung gefunden werden kann, die die Bedingung $|P| > |Q| > |P|$ erfüllt. Das korrespondierende Netz $\mathcal{F}$ aus Abbildung 7.1 besitzt zwei stabile Zustände, $(0, 1)$ und $(1, 0)$. Alle anderen Zustände sind instabil, da $f_{\mathcal{F}}(1, 1) = (0, 0)$ und $f_{\mathcal{F}}(0, 0) = (1, 1)$ gilt.

Die stabilen Zustände des Netzes für $\mathcal{P}$ repräsentieren Interpretationen, die mit den *stabilen Modellen* für $\mathcal{P}$ übereinstimmen. ◦

Im folgenden wollen wir untersuchen, welche Beziehungen zwischen unserem Berechnungsmodell beziehungsweise der iterierten Fixpunktsemantik akzeptabler Programme und der stabilen Modellsemantik bestehen.

Einführend soll eine kurze Beschreibung der stabilen Modellsemantik gegeben werden.

In [GL88] schlagen Gelfond und Lifschitz eine neue deklarative Semantik für normale Programme vor, die *stabile Modellsemantik*.

Definition 7.2 *Sei $\mathcal{P}$ ein logisches Programm. Jede Klausel, die Variablen enthält wird durch alle ihre Grundinstanzen ersetzt, so daß alle Atome in $\mathcal{P}$ grundinstantiiert sind. Für eine beliebige Menge M von Grundatomen aus $\mathcal{P}$ erhalten wir das Programm $\mathcal{P}_M$ durch Streichen*

- *jeder Klausel aus $\mathcal{P}$ mit einem Vorkommen eines negativen Literals $\neg B$ mit $B \in M$ im Körper der Klausel*

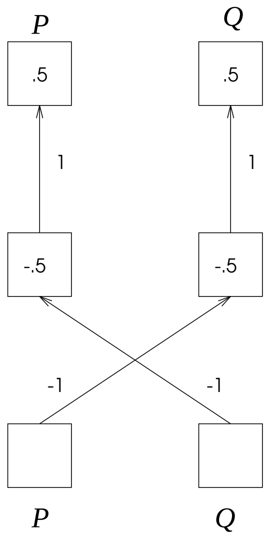

Abbildung 7.1: Ein einfaches vorwärtsgerichtetes Netz, daß die Funktion $\mathcal{T}_\mathcal{P}$ für das Programm $\mathcal{P} = \{P \leftarrow \neg Q; Q \leftarrow \neg P\}$ berechnet.

- *aller negativen Literale im Körper der verbleibenden Klauseln*

$\mathcal{P}_M$ ist ein definites Programm, dessen kleinstes Herbrand-Modell durch $\mathcal{T}_{\mathcal{P}_M} \uparrow \omega$ ermittelt werden kann. Wir definieren $S_\mathcal{P}(M) = \mathcal{T}_{\mathcal{P}_M} \uparrow \omega$. Die Menge M heißt *stabile Menge* von $\mathcal{P}$, falls $S_\mathcal{P}(M) = M$. Aufgrund des folgenden Theorems bezeichnet man stabile Mengen auch als *stabile Modelle.*

Theorem 7.3 [GL88] *Jede stabile Menge eines Programmes $\mathcal{P}$ ist ein minimales Herbrand–Modell für $\mathcal{P}$.*

Die *stabile Modellsemantik* für ein Programm $\mathcal{P}$ ist definiert, falls $\mathcal{P}$ genau ein stabiles Modell besitzt. Falls $\mathcal{P}$ definit ist, sind $\mathcal{P}$ und $\mathcal{P}_M$ für jede Menge M gleich und $S_\mathcal{P}(M)$ entspricht dem kleinsten Herbrand-Modell für $\mathcal{P}$.

Im folgenden sollen Beziehungen zwischen akzeptablen Programmen und Programmen, für die genau ein stabiles Modell existiert, untersucht werden.

Das Programm $\mathcal{P}$ aus Beispiel 3.21 ist nicht akzeptabel. Da $\mathcal{P}$ jedoch definit ist, sind die Menge $S_\mathcal{P}(M)$ und $M_\mathcal{P}$ (kleinstes Herbrand-Modell für $\mathcal{P}$) gleich und $\mathcal{P}$ genau ein stabiles Modell. Es existieren also Programme, die nicht akzeptabel sind, aber ein stabiles Modell besitzen. Aber akzeptable Programme besitzen ein stabiles Modell.

Beispiel 7.4 Sei $\mathcal{P}$ das Programm:

$$\begin{array}{rcl} p(X) & \leftarrow & q(X), \neg r(X). \\ q(X) & \leftarrow & r(X), \neg p(X). \\ r(X) & \leftarrow & p(X), \neg q(X). \end{array}$$

Die Berechnung eines stabilen Modelles für $\mathcal{P}$ liefert

$$M = \emptyset \quad \mathcal{P}_M = \{p(X) \leftarrow q(X); q(X) \leftarrow r(X); r(X) \leftarrow p(X)\} \quad S_{\mathcal{P}}(M) = \emptyset,$$

d.h. $\mathcal{P}$ besitzt genau ein stabiles Modell. $\mathcal{P}$ ist akzeptabel bezüglich der Interpretation $I = \emptyset$. Die Berechnung von $\mathcal{T}_{\mathcal{P}}$ liefert

$$\begin{array}{rcccl} \mathcal{T}_{\mathcal{P}} \uparrow 0 & = & \mathcal{T}_{\mathcal{P}}(\bot) & = & \emptyset \\ \mathcal{T}_{\mathcal{P}} \uparrow 1 & = & \mathcal{T}_{\mathcal{P}}(\emptyset) & = & \emptyset \end{array}$$

$\mathcal{P}$ ist akzeptabel und besitzt genau ein stabiles Modell, das mit dem kleinsten Fixpunkt von $\mathcal{T}_{\mathcal{P}}$ übereinstimmt. ◦

Satz 7.5 *Sei $\mathcal{P}$ ein akzeptables Programm. Der Fixpunkt des Operators $\mathcal{T}_{\mathcal{P}}$ ist ein stabiles Modell für $\mathcal{P}$.*

Beweis

- Für ein akzeptables Programm $\mathcal{P}$ liefert $\mathcal{T}_{\mathcal{P}}$ für eine beliebige Anfangsinterpretation I einen eindeutigen Fixpunkt $\mathrm{lfp}(\mathcal{T}_{\mathcal{P}})$, der Modell für $\mathcal{P}$ ist. Somit gilt auch $\mathrm{lfp}(\mathcal{T}_{\mathcal{P}}) = \mathcal{T}_{\mathcal{P}} \uparrow \omega(I) = \mathcal{T}_{\mathcal{P}} \uparrow \omega(\emptyset)$.
- Bezeichne X die Menge $\mathcal{T}_{\mathcal{P}} \uparrow \omega(\emptyset)$ und Y die Menge $S_{\mathcal{P}}(X) = \mathcal{T}_{\mathcal{P}_X} \uparrow \omega(\emptyset)$.
- Es ist zu zeigen: X ist stabil, also $X = Y$.

($\rightarrow$) Zu zeigen ist $Y \subseteq X$.
Sei $A \in Y$.

Da $\mathcal{P}_X$ definit ist, ist $\mathcal{T}_{\mathcal{P}}$ monoton. Da $Y = \mathcal{T}_{\mathcal{P}_X} \uparrow \omega(\emptyset)$, gilt $A \in \mathcal{T}_{\mathcal{P}_X} \uparrow i(\emptyset)$ für einen bestimmten Index i.

IA: Falls $i = 1$, ist A Fakt in $\mathcal{P}_X$. Es sind zwei Fälle zu betrachten.

1. A ist auch Fakt in $\mathcal{P}$. Daraus folgt $A \in X$.
2. In $\mathcal{P}$ existiert eine Klausel $A \leftarrow \neg B_1, \ldots, \neg B_n$. Daraus folgt $B_j \notin X$, woraus $A \in X$ folgt, da X ein Modell für $\mathcal{P}$ ist.

IV: Für alle $i \leq \alpha$ gilt: Falls $A \in \mathcal{T}_{\mathcal{P}_X} \uparrow i(\emptyset)$ dann gilt $A \in X$.

IB: Sei $A \in \mathcal{T}_{\mathcal{P}_X} \uparrow (\alpha + 1)(\emptyset)$. Es existiert eine Klausel $A \leftarrow C_1, \ldots, C_n \in \mathcal{P}_X$ so daß

$$\mathcal{T}_{\mathcal{P}_X} \uparrow \alpha \models C_1, \ldots, C_n$$

gilt. Also gibt es eine Klausel

$$A \leftarrow C_1, \ldots, C_n, \neg B_1, \ldots, \neg B_m \in \mathcal{P}$$

mit $m \geq 0$ und $B_j \notin X (\forall j = 1, \ldots, m)$. Laut Induktionsprinzip gilt für alle $C_k \in X$ mit $k = 1, \ldots, n$. X ist aber ein Modell für $\mathcal{P}$, also gilt $A \in X$.

($\leftarrow$) Zu zeigen ist $X \subseteq Y$.
Sei $A \in X$. Da $X = \mathcal{T}_{\mathcal{P}} \uparrow \omega(\emptyset)$ gilt $A \in \mathcal{T}_{\mathcal{P}_X} \uparrow i(\emptyset)$ für einen bestimmten Index i.

IA: A ist Fakt in $\mathcal{P}$, daraus folgt, A ist Fakt in $\mathcal{P}_X$. Dann gilt $A \in S_{\mathcal{P}}(X)$, also $A \in Y$.

IV: Für alle $i \leq \alpha$ gilt: Falls $A \in \mathcal{T}_{\mathcal{P}} \uparrow i(\emptyset)$ dann gilt $A \in Y$.

IB: Sei $A \in \mathcal{T}_{\mathcal{P}} \uparrow (\alpha + 1)(\emptyset)$. Es existiert eine Klausel

$$A \leftarrow C_1, \ldots, C_n, \neg B_1, \ldots, \neg B_m \in \mathcal{P}$$

mit $C_1, \ldots, C_n \in \mathcal{T}_{\mathcal{P}} \uparrow (\alpha)(\emptyset)$ und $\neg B_1, \ldots, \neg B_m \notin \mathcal{T}_{\mathcal{P}} \uparrow (\alpha)(\emptyset)$. Daraus folgt es existiert eine Klausel $A \leftarrow C_1, \ldots, C_n$ in $\mathcal{P}_X$ und laut Induktionsvoraussetzung gilt $C_1, \ldots, C_n \in S_{\mathcal{P}}(X)$. Damit gilt aber auch $A \in S_{\mathcal{P}}(X)$, das heißt $A \in Y$.

■

Ermitteln wir die Semantik eines akzeptablen Programms $\mathcal{P}$ mit Hilfe der von Fitting vorgeschlagenen metrischen Methode, so ist das mit Hilfe des Operators $\mathcal{T}_{\mathcal{P}}$ berechnete Modell ein stabiles Modell für $\mathcal{P}$.

8. Zusammenfassung

Die vorliegende Arbeit zeigt, daß eine enge Beziehung zwischen normalen aussagenlogischen Programmen und rekurrenten Netzen mit binären Schwellwerteinheiten besteht. Aufgrund der von Hornik etal. gezeigten Aussage, daß mehrschichtige rekurrente Netze als Approximatoren für (Borel-) meßbare Funktionen verwendet werden können, haben wir vermutet daß sie ebenfalls benutzt werden können, um die Funktion $\mathcal{T}_{\mathcal{P}}$ für ein prädikatenlogisches logisches Programm $\mathcal{P}$ zu berechnen.

Für den aussagenlogischen Fall wird gezeigt, daß man immer ein Netz finden kann, dessen Iteration die Semantik des Programmes berechnet.

Die Anwendung der von Fitting vorgeschlagenen metrischen Methode zur Berechnung der Semantik eines Programmes liefert uns eine Aussage über die Konvergenz des Netzes. Damit konnten wir zeigen, daß für ein akzeptables Programm die Konvergenz des von uns konstruierten Netzes gesichert ist. Somit liefert unser Berechnungsmodell einen eindeutig bestimmten Fixpunkt des Operators $\mathcal{T}_{\mathcal{P}}$, das heißt, eine eindeutige Semantik für das Programm $\mathcal{P}$.

Wir konnten außerdem zeigen, daß zu jedem 3–schichtigen vorwärtsgerichteten Netz ein Programm gefunden werden kann, so daß das Netz gerade den Operator $\mathcal{T}_{\mathcal{P}}$ für das Programm berechnet und außerdem $\mathcal{T}_{\mathcal{P}}$ die Eigenschaften einer Kontraktion besitzt.

Das vorgestellte massiv parallele Berechnungsmodell ermittelt damit für eine sehr umfangreiche Klasse von aussagenlogischen Programmen deren Semantik.

Kann man die Borel-Messbarkeit der Funktion $\mathcal{T}_{\mathcal{P}}$ für ein prädikatenlogisches Programm $\mathcal{P}$ zeigen, so ist gesichert, daß ein Netz existiert, das $\mathcal{T}_{\mathcal{P}}$ berechnet. Bei der Konstruktion eines solchen Netzes ist insbesondere die Repräsentation von Variablen von Interesse, da dieses Problem im aussagenlogischen Fall bisher nicht betrachtet werden mußte. Anregungen dafür sollten die in Abschnitt 4.2 angesprochenen Netze liefern.

Ich danke Prof. Steffen Hölldobler für die sehr hilfreiche Zusammenarbeit und Unterstützung, sowie für die Anregungen, die ich aus vielen Diskussionen erhielt.

Ich möchte auch Ulrich Hensel und Gerald Volkmann für die aufmerksame Lektüre des Manuskriptes und die nützlichen Hinweise zu Form und Inhalt danken.

A. Erklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet, sowie Zitate kenntlich gemacht habe.

Dresden, den 9. Juli 1994

Yvonne Kalinke

Literaturverzeichnis

[AP93] K. R. Apt and D. Pedreschi. Proving termination of general prolog programs. *Information and Computation*, 106:109–157, 1993.

[Bac92] P. Bachmann. *Mathematische Grundlagen der Informatik*. Akademie-Verlag, 1992.

[Bat89] A. Batarekh. *Topological aspects of logic programming*. PhD thesis, Syracuse University, 1989.

[Bat90] Aida Batarekh. Fixpoint techniques for non-monotone maps. In *Algebraic and Logic Programming*, volume 463 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 1990.

[BB89] E.J. Borowski and J.M. Borwein. *Mathematics*. Collins, 1989.

[Bil79] P. Billingsley. Probability and measure. New York:Wiley, 1979.

[Bör92] E. Börger. *Berechenbarkeit, Komplexität, Logik*. Viehweg, 1992.

[Bra91] R. Brause. *Neuronale Netze*. Teubner, 1991.

[BS89] A. Batarekh and V. S. Subrahmanian. Topological model set deformations in logic programming. *Fundamenta Informaticae*, 12:357–400, 1989.

[Cla87] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logik and Data Bases*, pages 293–322. Plenum Press, New York, 1987.

[CS93] M. Cadoli and M. Scherf. On the complexity of entailment in propositional multivalued logics. In *Working notes of the AAAI Spring Symposium Series, AI and NP-Hard Problems*, pages 15–21, 1993.

[DG84] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 1984.

[Elm90] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[Fit85] M. Fitting. A kripe/kleene semantics for logic programming. *Journal of Logic Programming*, 2:295–312, 1985.

[Fit93] M. Fitting. Metric methods – three examples and a theorem. CUNY, Bronx, NY 10468, 1993.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proceedings of the International Joint Conference and Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[GRS88] A. Van Gelder, K.A. Ross, and J.S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings Symposium on Principles of Database Systems*, pages 221–230, 1988.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neuronal Networks*, 2:359–366, 1989.

[Jor86] M. I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Annual Conference of the Cognitive Science Society*, pages 531–546, 1986.

[KR90] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 17, pages 869–941. Elsevier Science Publishers B.V., 1990.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, second edition, 1987.

[MP43] W. S. McCulloch and W. Pitts. A logical calculus and the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MP72] M. Minsky and S. Papert. *Perceptrons.* MIT Press, 1972.

[Pol89] J. B. Pollack. Implications of recursive distributed representations. In Touretzky, editor, *Advances in Neural Information Processing 1*, pages 527–536. Morgan Kaufmann, 1989.

[Prz86] T.C. Przymusinski. On the semantics of stratified deductive databases. In J. Minker, editor, *Workshop on Foundations of Deductive Databases and Logic Programming*, pages 433–443, Washington DC, August 1986.

[Prz88] T.C. Przymusinski. Perfect model semantics. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 1081–1096, 1988.

[Rei90] K. R. Reischuk. *Einführung in die Komplexitätstheorie.* Teubner, 1990.

[Rei94] H. Reichel. Logik für Informatiker. Vorlesungsskript, 1994.

[Roj93] R. Rojas. *Theorie der neuronalen Netze.* Springer, 1993.

[Scu90] M. G. Scutella. A note on Dowling and Gallier's top-down algorithm for propositional Horn satisfiability. *Journal of Logic Programming*, 8:265–273, 1990.

[Smy92] M. B. Smyth. Topology. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, chapter Topology. Oxford Science Publications, 1992.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[Wil70] S. Willard. *General Topology.* Addison-Wesely, 1970.