Modulare Datentypdefinitionen und ihre Beziehungen zur Logik erster Stufe

Dissertation
zur Erlangung des Grades
des Doktors der Naturwissenschaften
der Technischen Fakultät
der Universität des Saarlandes

von

Ralf Treinen

Saarbrücken
1991

Tag des Kolloquiums: 5. Dezember 1991

Vorsitzender:	Prof. Dr.-Ing. R. Maurer
Berichterstatter:	Prof. Dr.-Ing. J. Loeckx
	Prof. Dr. rer. nat. G. Smolka

Inhaltsverzeichnis

1 Einführung

Die vorliegende Arbeit hat einige grundsätzliche Probleme der Verifikation modularisierter Software zum Gegenstand. Dabei interessieren wir uns für die semantischen Aspekte im Zusammenhang mit der Top-Down Entwicklung und der Wiederverwertbarkeit von Software-Modulen.

Der Begriff der Modularisierung ist in der Softwareentwicklung eng mit dem Konzept der Abstraktion verknüpft ([Bis86]). Ausgangspunkt hierfür war das Konzept der *prozeduralen* Abstraktion ([Par72]), wie sie beispielsweise durch Prozeduren in der Programmiersprache Pascal ([JW78]) realisiert werden. Insbesondere seit den 70er Jahren entwickelte sich, nicht zuletzt ausgelöst durch die Arbeiten über abstrakte Datentypen ([GTW78]), ein starkes Interesse an *Daten*abstraktion. Dahinter steckt die Überlegung, daß ein Datentyp aus Datenstrukturen *und* Operationen besteht und diese daher als eine Einheit zu behandeln sind. Diese Ideen fanden Eingang in die Entwicklung moderner Programmier- und Spezifikationssprachen.

Als Beispiele von Modularisierungskonzepten in Programmiersprachen erwähnen wir hier nur die "clusters" von CLU ([LG86], [LAB+81]), die "packages" von Ada[1] ([ASCII83]), die "modules" von Modula-2([Wir85]) und die "structures" von ML ([Mac86]). Generische Datentypen wie in ML oder in Miranda[2] ([Tur85]) ermöglichen es, bei der Definition komplexer Datenstrukturen von den zugrundeliegenden Basissorten zu abstrahieren. Spezifikationssprachen beruhen ebenfalls in starkem Maße auf Modulkonzepten, dies gilt für operationelle, axiomatische (insbesondere algebraische) und konstruktive (insbesondere algorithmische) Spezifikationssprachen ([EM90]). Eine sehr unvollständige Liste von Spezifikationssprachen mit Modulkonzepten besteht aus Alphard ([Sha81]), CLEAR ([BG80]), ACT ONE ([EM85]) und OBSCURE ([LL87], [LL90]).

Was verstehen wir nun — aus Sicht der Semantik — unter einem *Modul* einer (Programmier- oder Spezifikations-) Sprache? Um diese Frage zu beantworten, müssen wir zunächst einmal die semantischen Objekte der betrachteten Sprache festlegen. Darunter verstehen wir die mathematischen Konzepte, die wir mit unserer Sprache definieren und die wir in der Syntax der Sprache bezeichnen können. Semantische Objekte können z. B. Werte aus bestimmten Wertebereichen, Funktionen, Zustandstransformationen oder Algebren sein. In der Programmiersprache Pascal beispielsweise werden Werte durch Terme entsprechenden Typs, Funktionen durch "functions" und Zustandstransformationen durch Prozeduren beschrieben. Sprachen mit Datenabstraktion (also insbesondere Spezifikationssprachen) zählen auch *Algebren* zu ihren semantischen Objekten.

Unter einem *Modul* verstehen wir einen wohlgeformten Ausdruck der Sprache, der seiner "Umgebung" (dem durch das umfassende Sprachkonstrukt gebildeten Kontext) einige dieser semantischen Objekte zur Verfügung stellt. Die wesentliche Einschränkung ist hierbei, daß diese semantischen Objekte die Semantik des Moduls bereits vollständig beschreiben. Wir können innerhalb eines Kontextes ein Modul durch ein anderes, das die gleichen semantischen Objekte definiert, ersetzen, ohne daß sich die Semantik des Kontextes dadurch ändert. Es spielt also

[1] Ada ist ein eingetragenes Warenzeichen des United States Departement of Defense.

[2] Miranda ist ein eingetragenes Warenzeichen von Research Software Ltd.

keine Rolle, *wie* diese Objekte innerhalb eines Moduls definiert werden, der Kontext hat keinen Zugriff auf die Implementierungsdetails. Dieses unter dem Stichwort "information hiding" bekannte Prinzip bildet eine wesentliche Grundlage der Entwicklung großer Softwarepakete.

Wir können unseren Modulbegriff auf *parametrisierte Module* erweitern. Dabei wird in der Syntax eines Moduls eine Menge von Bezeichnern für semantische Objekte, die in dem Modul benutzt, aber nicht definiert werden, ausgezeichnet. Wir nennen diese Bezeichner dann die (formalen) *Parameter* des Moduls. Die Semantik eines solchen parametrisierten Moduls ist eine Funktion, die die Denotationen der formalen Parameter (d. h. die aktuellen Parameter) auf die von dem Modul unter der gegebenen Parameterbelegung definierten semantischen Objekte abbildet. Wir werden in dieser Arbeit unter Modulen stets parametrisierte Module verstehen.

Diese Definition eines Modulkonzeptes für eine Sprache ist natürlich sehr allgemein, sie umfaßt beispielsweise auch einzelne Funktionsanwendungen in einer funktionalen Programmiersprache. Tatsächlich sind wir aber — und dies entspricht dem eigentlichen Sinn des Begriffes "Softwaremodul" in der Informatik — an Modulen interessiert, deren semantische Objekte eine reiche mathematische Struktur aufweisen. Die in dieser Arbeit betrachteten Module dienen zur Definition von *Algebren*, die Semantik eines Moduls ist also eine Abbildung von Algebren auf Algebren.

Geht man davon aus, daß man die Semantik eines Programms als eine Algebra auffassen kann, so sehen wir hier die Bedeutung der Module insbesondere in den beiden folgenden Punkten:

1. Ein Modul kann innerhalb eines Softwarepaketes mehrfach eingesetzt werden, indem es auf verschiedene aktuelle Parameter angewandt wird.

2. Module können zur Top-Down Entwicklung von Algebren benutzt werden. Um eine Algebra zu programmieren (spezifizieren), konstruiert man dazu ein Modul, das für geeignete Parameteralgebren die gewünschte Ergebnisalgebra liefert. Anschließend verfährt man ebenso für die Parameteralgebra.

Der springende Punkt bei dieser Art der Top-Down Entwicklung liegt natürlich in der Beschreibung der "geeigneten" Parameteralgebren. Um dies zu präzisieren, müssen wir zunächst eine Beschreibungssprache für Algebren festlegen. Wir wählen hier die Prädikatenlogik erster Stufe, da sie einen guten Kompromiß zwischen Ausdruckskraft einerseits und "Beherrschbarkeit" (sowohl aus Sicht der Beweistheorie als auch der Modelltheorie) andererseits bietet.

Mit der Prädikatenlogik erster Stufe als Beschreibungssprache an der Hand können wir das Problem der Top-Down Entwicklung von Modulen nun genauer fassen: gegeben ist eine prädikatenlogische Formel w und ein Modul m, so daß w Ergebnisalgebren von m beschreibt. Gesucht ist eine Formel v, die genau diejenigen Parameteralgebren beschreibt, für die das Ergebnis der Anwendung der Semantik von m die Formel w erfüllt. Man beachte hierbei, daß die Formeln nur Eigenschaften der Algebren festlegen und somit nicht mehr einzelne Algebren eindeutig und vollständig beschreiben.

Im Zusammenhang mit der Wiederverwertbarkeit von Modulen ergibt sich ein ähnliches Problem. Hier möchten wir oft für ein Modul m durch eine Formel w Eigenschaften der Ergeb-

nisalgebren von m beschreiben, die für alle Parameteralgebren erfüllt sein sollen. Gesucht ist eine (schwächste, d. h. möglichst allgemeine) Bedingung v an die Parameteralgebren, die diese garantiert.

In beiden Fällen nennen wir v eine m-schwächste Parameterbedingung von w. Wir werden nun das Problem weiter präzisieren und dabei einen Überblick über die in dieser Arbeit vorgestellten Ergebnisse geben. Der Aufbau der folgenden Abschnitte orientiert sich an der Gliederung der gesamten Arbeit in Kapitel.

1.1 Module

Im Kapitel 3 stellen wir eine idealisierte Sprache zur Definition von Modulen vor. Ein solches Modul besitzt zwei "Schnittstellen" zu seiner Außenwelt: eine Parametersignatur und eine Exportsignatur. Diese legen somit den Definitions- und den Wertebereich der dem Modul zugeordeneten Semantik fest. Einem Modul m mit Parametersignatur Σ_P und Exportsignatur Σ_E, oder anders ausgedrückt einem Modul m mit Signatur (Σ_P, Σ_E), ordnen wir als Semantik einen Funktor zu:

$$[\![m]\!]: \text{Klasse der } \Sigma_P\text{-Algebren} \quad \rightarrow \quad \text{Klasse der } \Sigma_E\text{-Algebren}$$

Dabei handelt es sich um einen "totalen Funktor", das heißt zu jeder Parameteralgebra gehört eine Exportalgebra.

Das Modul selbst enthält die Definitionen der Sorten und Funktionen von Σ_E, die nicht Bestandteil von Σ_P sind. Sorten und Funktionen, die sowohl in der Parametersignatur als auch in der Exportsignatur enthalten sind, werden durch die Semantik des Moduls nicht verändert.

Die in einem Modul zu kreierenden Sorten werden durch die Angabe von Konstruktorfunktionen als eine formale Sprache definiert. Dieses Verfahren ist aus Programmiersprachen wie ML oder Miranda sowie von Algorithmischen Spezifikationen her bekannt. Zusätzlich zu den Konstruktorfunktionen können wir weitere Funktionen durch Angabe rekursiver Programme definieren. Die Syntax unser Sprache ist elementar — sie enthält nur Bestandteile, die wir zur Realisierung dieser Definitionsprinzipien benötigen. Syntaktischen Zucker wie "Pattern Matching" oder ein *let*-Konstrukt haben wir, obwohl dieser für die praktische Anwendung einer Sprache wichtig ist, hier nicht aufgenommen.

Ein erstes Beispiel einer Moduldefinition findet man in Abbildung 1. Tatsächlich enthalten die Parametersignatur Σ_P sowie die Exportsignatur Σ_E mehr Sorten und Operationen, als aus der Moduldefinition in Abbildung 1 direkt ersichtlich ist. Jede der betrachteten Signaturen enthält einen "Standardteil", der aus einer Konstante $\perp_s$ zu jeder Sorte s, einer Sorte *bool* sowie unter anderem einem Funktionszeichen `ifthenelse`: $bool, s, s \rightarrow s$ zu jeder Sorte s besteht. Die Algebren (wir werden dann von "Standardalgebren" sprechen) weisen $\perp_s$ jeweils ein "undefiniertes" Element, d. h. ein minimales Element einer flachen cpo, und den restlichen Standardkomponenten ihre übliche Bedeutung zu. Das undefinierte Element benötigen wir zur Definition der Semantik der durch ein rekursives Programm beschriebenen Funktionen. Den Datentyp *bool*

PAR **SORTS** $elem$
OPNS $+ : elem, elem \to elem$
$f : elem \to elem$
BODY **SORTS** $list$
CONS $atom : elem \to list$
$cons : elem, list \to list$
FCTS $suml : list \to elem$
$sumr : list \to elem$
PROG $sumr(l) \Leftarrow \texttt{if is}_{atom}\texttt{?}(l)\ \texttt{then select}^1_{atom}(l)$
$\texttt{else } sumr(\texttt{select}^2_{cons}(l)) + \texttt{select}^1_{cons}(l)$
$suml(l) \Leftarrow \texttt{if is}_{atom}\texttt{?}(l)\ \texttt{then select}^1_{atom}(l)$
$\texttt{else select}^1_{cons}(l) + suml(\texttt{select}^2_{cons}(l))$

Abbildung 1: Ein Beispiel einer Moduldefinition.

mit den zugehörigen Funktionen, insbesondere `ifthenelse`, werden wir oft in den rekursiven Programme benutzen.

Die Semantik des Moduls aus Abbildung 1 bildet eine Algebra, die eine Sorte *elem*, eine unäre Funtion f und eine binäre Operation + enthält, auf eine Algebra ab, die im wesentlichen aus den folgenden Komponenten besteht:

- die Sorte *elem* und die Operationen f und + aus der Parameteralgebra,
- eine Sorte *list*, bestehend aus den endlichen nichtleeren Listen von definierten Elementen der Sorte *elem* der Parameteralgebra, sowie einem undefinierten Element der Sorte *list*
- Konstruktorfunktionen *cons* und *atom* zur Konstruktion von Listen,
- Funktionen wie $\texttt{is}_{atom}\texttt{?}$ oder $\texttt{select}^1_{cons}$ zum Zugriff auf Teilstrukturen von Listen und
- zwei rekursiv definierte Funktionen *suml* und *sumr*.

Im Gegensatz zur "klassischen Programmverifikation" ([LS87]) untersuchen wir die Eigenschaften der Programme und Datentypdefinitionen nicht in einzelnen, fest gewählten Interpretationen der Parameterfunktionen. Statt dessen betrachten wir, gemäß dem funktionalen Charakter der Semantik, das semantische Verhalten der Module in einer großen *Klasse* von Interpretationen. Diese Sichtweise entspricht also eher dem "Geiste" der Arbeiten über Programmschemata ([Gre75]). So erhält man aus unserer Definition der Modulsprache die beispielsweise in [Gre75] untersuchten rekursiven Programmschemata, indem man die Möglichkeit zur Definition neuer Datentypen wegstreicht. In diesem Sinne könnten wir also auch von "Datentypschemata" sprechen.

1.2 Logik

In Kapitel 4 werden wir die Prädikatenlogik erster Stufe dazu benutzen, Algebren zu beschreiben. Die "klassische" Prädikatenlogik" liefert hierzu eine Syntax von Formeln und eine Gültigkeitsrelation $\models$ zwischen Formeln und Algebren. Genauer existiert zu jeder Signatur Σ eine Menge von Σ-Formeln, die wir als Aussagen über Σ-Algebren ansehen können. $\models$ ist also für jede Signatur Σ jeweils eine Relation zwischen der Menge der Σ-Formeln und der Klasse der Σ-Algebren.

Mit den Formeln der Prädikatenlogik erster Stufe können wir aber im allgemeinen nur bestimmte Eigenschaften der Algebren spezifizieren, zu einer Formel gibt es stets eine *Klasse* von Algebren, in denen diese Formel gilt. Die in dieser Klasse enthaltenen Algebren können dabei durchaus strukturell verschiedene (d. h. nicht isomorphe) Algebren enthalten, die Aussagekraft der Prädikatenlogik erster Stufe ist also beschränkt.

Wir haben nun die beiden syntaktischen Konstruktionen, in deren Rahmen sich die hier behandelte Problematik bewegt, kennengelernt: Module und prädikatenlogische Formeln. Die Formeln können dabei sowohl zur Beschreibung der Parameteralgebren als auch der Exportalgebren benutzt werden. Die zugehörigen semantischen Kategorien sind dann:

- Eine Formel der Signatur Σ_P beschreibt eine Klasse von Σ_P-Algebren.
- Ein Modul der Signatur (Σ_P, Σ_E) beschreibt einen Funktor, der Σ_P-Algebren auf Σ_E-Algebren abbildet.
- Eine Formel der Signatur Σ_E beschreibt eine Klasse von Σ_E-Algebren.

Da das Symbol $\bot$ zum Standardteil jeder Signatur gehört, können wir in den prädikatenlogischen Formeln auch Terminierungsaussagen über einzelne Funktionen treffen. Typische Formeln über der Paramtersignatur Σ_P für das Modul m aus Abbildung 1 sind

$$\begin{aligned} v_1 &:= \forall x, y : elem \,.\, x + y = y + x \\ v_2 &:= \exists x : elem \,.\, \forall y : elem \,.\, x + y = y \wedge y + x = y \\ v_3 &:= \forall x, y : elem.(x \neq \bot_{elem} \wedge y \neq \bot_{elem}) \supset x + y \neq \bot_{elem} \end{aligned}$$

Beispiele für Formeln über der Exportsignatur Σ_E sind

$$\begin{aligned} w_1 &:= \forall l : list \,.\, suml(l) = sumr(l) \\ w_2 &:= \forall l : listl \neq \bot_{list} \supset (suml(l) \neq \bot \wedge sumr(l) \neq \bot) \end{aligned}$$

Oft möchten wir das Verhalten eines Modules gar nicht in der Klasse *aller* Standardalgebren zu einer Parametersignatur untersuchen, sondern in einer "sinnvollen" Teilklasse. Hierzu definieren wir den Begriff eines *Domainoperators*. Ein Domainoperator $\Im$ bildet eine Signatur Σ auf eine Klasse $\Im_\Sigma$ von Σ-Standardalgebren ab, wobei bestimmte Bedingungen erfüllt sein müssen. Wichtige Beispiele von Domainoperatoren sind

- $\Im^f$ bildet eine Signatur Σ auf die Klasse *aller* Σ-Standardalgebren ab,
- $\Im^{st}$ bildet eine Signatur Σ auf die Klasse derjenigen Σ-Algebren ab, in denen alle Funktionen strikt und total sind.

Die oben angeführten schwächsten Parameterbedingungen können wir nun genauer definieren:

> Gegeben sei ein Modul m der Signatur (Σ_P, Σ_E), ein Domainoperator $\Im$ und eine Formel w der Signatur Σ_E. Eine Σ_P-Formel v ist eine $\Im, m$-schwächste Parameterbedingung von w (abgeküzt *spab*), falls für alle Σ_P-Algebren $A \in \Im_{\Sigma_P}$ gilt:
>
> $$A \models v \quad \text{genau dann, wenn} \quad [\![m]\!](A) \models w$$

So ist in obigem Beispiel v_1 eine $\Im^f, m$-spab sowie auch eine $\Im^{st}$-spab von w_1. Eine $\Im^f$-spab von w_2 ist v_3, eine $\Im^{st}$-spab von w_2 ist die Formel `True`.

Es wird sich aber recht schnell herausstellen, daß es im allgemeinen zu einem Modul m und zu einer Formel w *keine* m-schwächste Parameterbedingung geben kann. Der Grund hierfür liegt in der beschränkten Aussagekraft der Prädikatenlogik erster Stufe. So können wir beispielsweise ein Modul m der Signatur (Σ_P, Σ_E) und eine Σ_E-Formel w angeben, so daß $[\![m]\!](A) \models w$ genau für alle endlichen Σ_P-Algebren A gilt. Es ist aber aus prinzipiellen Gründen nicht möglich, mit einer Formel der Prädikatenlogik erster Stufe die Klasse aller endlichen Algebren einer Signatur zu beschreiben. In dem Beispiel aus Abbildung 1 hat die folgende Formel keine $\Im^f, m$-spab, wie wir später zeigen werden:

$$\exists l : list \,.\, l \neq \bot_{list} \wedge f(suml(l)) = f(sumr(l))$$

1.3 Modelltheoretische Einordnung

Um die logischen Eigenschaften der Modulsemantik bewerten zu können, nehmen wir nun einen anderen Standpunkt ein: Wir können die Modulsemantik mit der Prädikatenlogik erster Stufe über Exportalgebren als eine neue Logik ansehen, die Aussagen über Parameteralgebren trifft. Hierzu holen wir nun etwas weiter aus und stellen unsere Betrachtungen der Logik in den Kontext der abstrakten Modelltheorie. Der Gegenstand der abstrakten Modelltheorie ([BF85]) ist der Vergleich verschiedener Logiken nach ihrer "Stärke" und ihren modelltheoretischen Eigenschaften. Diese Begriffe müssen natürlich präzise gefaßt werden. Was soll man zunächst überhaupt unter "einer Logik" verstehen?

Wir wollen den Begriff der *Logik* hier nicht formal definieren, es seien nur einige wichtige Punkte festgehalten (siehe [Ebb85] für vollständige Definitionen): Eine Logik $\mathcal{L}$ ordnet jeder prädikatenlogischen Sprache τ, d. h. jeder wohlgeformten Menge von Sorten, Funktions- und Prädikatenzeichen

- eine Menge $\mathcal{L}(\tau)$ von τ-*Sätzen* und
- eine *Gültigkeitsrelation* $\models_{\mathcal{L}}$ zwischen τ-Modellen und τ-Sätzen

zu. Dabei müssen bestimmte Bedingungen, die beispielsweise ein Wohlverhalten bezüglich Isomorphismen von Modellen oder Umbenennungen von prädikatenlogischen Sprachen garantieren, erfüllt sein. Wichtig ist hierbei, daß der Modellbegriff von *erster Stufe* (first order) ist: Ein Modell enthält zu jeder Sorte eine Trägermenge und zu jedem Funktions- und Prädikatszeichen eine Funktion, bzw. eine Relation entsprechender Stelligkeit auf diesen Trägermengen. Die Modelle enthalten also keine höheren Funktionen oder Relationen auf Mengen. Dies schließt aber nicht aus, daß solche Objekte "intern" in einer Logik benutzt werden, um Aussagen über Modelle zu treffen. Neben der Prädikatenlogik erster Stufe, die man mit $\mathcal{L}_{\omega\omega}$ bezeichnet, sind wichtige Beispiele von Logiken:

1. die infinitäre Logik $\mathcal{L}_{\omega_1\omega}$, die $\mathcal{L}_{\omega\omega}$ insofern erweitert, als sie Disjunktionen und Konjunktionen abzählbar vieler Formeln erlaubt ([Kei71], [Nad85]),

2. die Logik $\mathcal{L}(Q_1)$, die $\mathcal{L}_{\omega\omega}$ um einen Quantor "es gibt überabzählbar viele Elemente, für die ... gilt" erweitert ([Kau85]) und

3. die Prädikatenlogik zweiter Stufe $\mathcal{L}^2$, die $\mathcal{L}_{\omega\omega}$ um Quantifizierungen über Relationen erweitert. Man beachte, daß der Modellbegriff auch hier von erster Stufe ist.

Alle diese Beispiellogiken haben die Eigenschaft, daß sie die Prädikatenlogik erster Stufe umfassen, und daß in ihnen die Konzepte Negation, Konjunktion und Existenzquantifizierung von Formeln ausdrückbar sind. Solche Logiken bezeichnet man als *reguläre Logiken*, für exakte Definitionen verweisen wir wieder auf [Ebb85].

Man kann nun, und dies ist ein zentrales Anliegen der abstrakten Modelltheorie, die Aussagekraft verschiedener Logiken vergleichen. Die entscheidende Frage ist dabei, welche Klassen von Modellen in den jeweiligen Logiken durch einzelne Sätze beschrieben werden können. Eine Logik $\mathcal{L}^*$ ist *mindestens so stark wie* eine Logik $\mathcal{L}$, in Zeichen $\mathcal{L} \leq \mathcal{L}^*$, falls gilt

> Für jede Klasse C von τ-Modellen, für die es einen Satz $w \in \mathcal{L}(\tau)$ gibt mit $C = \{M \mid M \models_{\mathcal{L}} w\}$, gibt es auch einen Satz $v \in \mathcal{L}^*(\tau)$ mit $C = \{M \mid M \models_{\mathcal{L}^*} v\}$.

Die Logiken der obigen Beispiele sind alle mindestens so stark wie $\mathcal{L}_{\omega\omega}$, man sieht leicht ein, daß sie sogar jeweils echt stärker als $\mathcal{L}_{\omega\omega}$ sind.

Weiterhin kann man nun die aus der Modelltheorie der Prädikatenlogik erster Stufe ([CK90] ist hier *die* Referenz) bekannten Sätze wie z. B. Kompaktheit, aufwärts und abwärts gerichtete Skolem-Löwenheim-Tarski Eigenschaft, Craigscher Interpolationssatz und Robinson Konsistenz, in diesem allgemeinen Rahmen untersuchen. Es stellt sich dabei heraus, daß bei den drei oben angegebenen Logiken stets einige der modelltheoretischen Eigenschaften der Prädikatenlogik erster Stufe verloren gehen.

Die bahnbrechende Erkenntnis von Lindström ([Lin69], siehe auch [CK90], [Flu85]) war es nun, daß bei regulären Logiken ein starker Zusammenhang zwischen der Verstärkung der Ausdruckskraft und dem Verlust wünschenswerter modelltheoretischer Eigenschaften besteht:

$\mathcal{L}_{\omega\omega}$ ist bezüglich $\leq$ maximal unter den regulären Logiken, die abzählbare Kompaktheit und $\downarrow$LST erfüllen.

Diese Eigenschaften bedeuten im einzelnen:

1. Eine Logik $\mathcal{L}$ ist abzählbar *kompakt*, wenn für jede abzählbare Menge T von $\mathcal{L}$-Sätzen gilt: Falls jede endliche Teilmenge von T ein Modell hat, dann hat auch T ein Modell.

2. Eine Logik $\mathcal{L}$ hat die *abwärts gerichtete Skolem-Löwenheim-Tarski Eigenschaft*, abgekürzt $\downarrow$LST, falls jeder $\mathcal{L}$-Satz, der ein unendliches Modell hat, auch ein abzählbares Modell hat.

Man kann den Satz von Lindström auch so lesen: Jede reguläre Logik, die echt stärker als $\mathcal{L}_{\omega\omega}$ ist, verletzt mindestens eine der Eigenschaften Kompaktheit und $\downarrow$LST. In unseren Beispielen erhalten wir ([Ebb85]):

1. $\mathcal{L}_{\omega_1\omega}$ ist $\downarrow$LST, aber nicht kompakt.

2. $\mathcal{L}(Q_1)$ ist kompakt, aber nicht $\downarrow$LST.

3. $\mathcal{L}^2$ ist weder kompakt noch $\downarrow$LST.

Wie paßt unsere Modulsemantik nun in dieses allgemeine Rahmenwerk?

Ein erster Ansatz könnte darin bestehen, zu einer Signatur Σ die Klasse aller Formeln $[m]w$ zu betrachten, wobei m ein Modul mit Signatur (Σ, Σ_E) und w ein Σ_E Satz der Prädikatenlogik erster Stufe ist. Die Erfüllbarkeitsrelation $\models$ würden wir dann durch

$$A \models [m]w \quad :\Longleftrightarrow \quad [\![m]\!](A) \models w$$

definieren. Auf Formeln dieser Bauart können wir leicht Negation, Konjunktion und Existenzquantifizierung erklären. Abgesehen von den eher unwesentlichen Einschränkungen, daß wir hier keine Prädikatszeichen und nur Standardalgebren betrachten, erhalten wir somit eine reguläre Logik in obigem Sinne. Diese Logik ist echt stärker als die Prädikatenlogik erster Stufe, denn wir können in ihr beispielsweise die Klasse der endlichen Algebren beschreiben. In Satz 1 werden wir zeigen, daß diese Logik die $\downarrow$LST Eigenschaft hat, nach dem Satz von Lindström muß also die Kompaktheit verletzt sein.

Mit dieser Erkenntnis können wir uns natürlich nicht zufrieden geben, denn bei dieser Logik handelt es sich um eine *exogene Logik*. In [KT90] wurde dieser Begriff im Zusammenhang mit Modallogiken benutzt, hier meinen wir damit, daß die Module selbst Bestandteil der Sätze der Logik sind. Die obige Betrachtung betrifft also die Logik, die die Klasse *aller* Sätze und damit insbesondere auch *aller* Module umfaßt. Unser Ausgangspunkt waren aber die logischen Probleme, die im Zusammenhang mit der Verifikation *einzelner* Module auftreten. Aus diesem Grunde gehen wir hier anders vor und definieren für jedes *einzelne* Modul m mit Signatur

(Σ_P, Σ_E) eine Menge von Formeln $\mathcal{S}en_m$, diese ist identisch mit der Menge der prädikatenlogischen Sätze über Σ_E. Das Modul m geht nun in die Definition der Gültigkeitsrelation ein. Für eine Σ_P-Algebra A definieren wir

$$A \, [m| \!= \, w \quad \Longleftrightarrow \quad [\![m]\!](A) \models w$$

Hierbei handelt es sich nun um eine *endogene Logik* im Sinne von [KT90].

In Satz 2 werden wir einen detaillierteren Zusammenhang zwischen der Kompaktheit von $[m|\!=$ und der Existenz von spabs zeigen, als ihn uns der Satz von Lindström liefert. Zu diesem Zwecke müssen wir *Erweiterungen* eines Moduls um Mengen von zusätzlichen Konstantenzeichen betrachten. Wir erhalten die Erweitrung des Modules m um die Menge C von Konstantenzeichen, indem wir C sowohl zur Paramtersignatur von m als auch zur Exportsignatur von m hinzufügen. Wir zeigen die Äquivalenz der beiden folgenden Aussagen für beliebige Module m und Domainoperatoren $\Im$:

- Für alle Erweiterungen m' von m um Konstanten ist $[m'|\!=$ kompakt, wobei die betrachtete Klasse von Modellen durch den Domainoperator $\Im$ bestimmt ist
- Zu jeder Erweiterung m' von m um Konstanten und zu jeder Formel w über der Exportsignatur von m' gibt es eine $\Im, m'$-spab

Mit den Sätzen 3 und 4 werden wir zeigen, daß die Existenz von spabs eine nicht entscheidbare Eigenschaft der Module ist. Genauer werden wir zeigen, daß die beiden folgenden Probleme für nichttriviale Domainoperatoren $\Im$ weder semientscheidbar noch kosemientscheidbar sind:

- Gibt es zu einem Modul m und einer Formel w über der Exportsignatur von m eine $\Im, m$-spab?
- Hat ein Modul m die Eigenschaft, daß es zu jeder Formel w über der Exportsignatur von m eine $\Im, m$-spab gibt?

1.4 Berechnung von spabs

Nachdem wir uns in Kapitel 5 mit Bedingungen für die Existenz von spabs beschäftigt haben, suchen wir in Kapitel 6 nach Möglichkeiten zur Berechnung von spabs. Dazu werden wir in Abschnitt 6.1 zunächst ein allgemeines Resultat (Satz 5) zeigen, das einen Zusammenhang herstellt zwischen der Berechenbarkeit von m-spabs einerseits und der Entscheidbarkeit der Theorie von $[\![m]\!](A)$ relativ zur Theorie von A für alle Parameteralgebren A andererseits.

Dabei ist, allgemein gesprochen, eine Menge M relativ zu einer Menge N entscheidbar, falls es einen Algorithmus zur Entscheidung von M gibt, der aber auf das vollständige Wissen um die Menge N zurückgreifen kann. Wie in der "normalen" Berechnungstheorie gibt es für diesen Begriff verschiedene Formalisierungen, einer davon ist der einer *Orakelmaschine*. Der Begriff der Orakelmaschine ist eine Erweiterung des Begriffes der Turingmaschine. Zusätzlich zu den

Berechnungschritten einer Turingmaschine kann eine Orakelmaschine in bestimmten Zuständen Fragen der Form "ist x in der Orakelmenge?" an sein Orakel stellen. Der weitere Verlauf der Berechnung hängt dann von der Antwort des Orakels ab.

Unser Resultat in Satz 5 liefert nun für beliebige Module m, Mengen W von Formeln über der Exportsignatur von m und Domainoperatoren, die sich aus Sicht der Beweistheorie ähnlich verhalten wie $\Im^f$, die Äquivalenz der beiden folgenden Aussagen:

- Es gibt einen Algorithmus, der zu jeder Formel aus W eine $\Im, m$-spab berechnet
- Für jede Paramteralgebra $A \in \Im_{\Sigma_P}$ ist $Th([\![m]\!](A))$ relativ zu $Th(A)$ entscheidbar.

Dieses Resultat legt es nahe, bekannte Entscheidbarkeitsresultate auf ihre Verwendbarkeit für die Berechnung von spabs zu untersuchen. In Abschnitt 6.2 werden wir, ausgehend von dem in [CL88] vorgestellten Entscheidungsverfahren für die Theorie einer Grundtermalgebra, eine Methode zur Berechnung von spabs für eine eingeschränkte Klasse von Modulen angeben (Satz 6). Wir geben zunächst eine kurze Übersicht über das in [CL88] vorgestellte Verfahren und erläutern anschließend unsere Erweiterungen.

Das Verfahren von [CL88] liefert ein Entscheidungsverfahren für die Theorie einer von einer Menge K erzeugten Grundtermalgebra G_K. Der Kern des Verfahrens besteht aus einem Regelsystem zur Eliminierung von Quantoren. Dieses Regelsystem transformiert eine Formel der Form

$$\exists x_1, \ldots, x_n \forall y_1, \ldots, y_m \,.\, w$$

wobei w keine Quantoren enthält, in eine in G_K äquivalente Formel

$$\exists x_1, \ldots, x_l \,.\, w'$$

wobei w' ebenfalls keine Quantoren enthält. Im allgemeinen gilt dabei $l \geq n$, d. h. es wurden neue existenzquantifizierte Variablen eingeführt. Zu den Regeln gehören neben allgemeingültigen prädikatenlogischen Transformaionen

1. Regeln zur Vereinfachung von Gleichungen und Ungleichungen zwischen Termen wie beispielsweise
$$k(t_1, \ldots, t_n) = k(r_1, \ldots, r_n) \perp\!\!\!\rightarrow t_1 = r_1 \wedge \ldots \wedge t_n = r_n$$
2. eine Regel zur Fallunterscheidung nach den möglichen Werten einer nicht zu $y_1, \ldots, y_n$ gehörigen Variablen:
$$\forall \vec{y} \,.\, P \perp\!\!\!\rightarrow \bigvee_{k \in K} \exists \vec{x} \forall \vec{y} \,.\, x = k(\vec{x}) \wedge P$$
falls $x \notin \vec{y}$. Diese Regel (explosion rule) ist die Quelle der neu entstehenden existenzquantifizierten Variablen.
3. eine Regel zur Elimierung gewisser Disjunktionen von Gleichungen, in vereinfachter Form:
$$\forall \vec{y} \,.\, z_1 = u_1 \vee \ldots \vee z_n = u_n \perp\!\!\!\rightarrow \texttt{False}$$

falls jedes z_i eine Variable ist, u_i jeweils nicht identisch mit z_i ist und jede Gleichung $z_i = u_i$ mindestens eine Variable aus $\vec{y}$ enthält.

Der Beweis dieser Regel (U4) beruht auf der Tatsache, daß eine solche Disjunktion von Gleichungen nur endlich viele Lösungen haben kann. [Mal71] enthält ein ähnliches Lemma.

Unsere Erweiterung bezieht sich nun auf Module, die höchstens eine durch ein rekursives Programm definierte Funktion aufweisen. Diese Funktion f muß dabei einige zusätzliche Bedingungen erfüllen:

- die Ergebnissorte von f ist eine Parametersorte, f hat nur ein Argument und die Argumentsorte ist eine neue Sorte
- f ist symbolisch partiell auswertbar, d. h. für jeden nur mit Konstruktoren und Variablen aufgebauten Term t kann man $f(t)$ zu einem äquivalenten Term reduzieren, in dem f nur noch mit einer Variablen als Argument auftaucht. Die Funktionen *suml* und *sumr* aus Abbildung 1 erfüllen diese Bedingung, so können wir beispielsweise die folgende Reduktion vornehmen:
$$suml(cons(x_1, cons(x_2, y)))) \perp\!\!\!\rightarrow x_1 + (x_2 + y)$$
- Die wichtigste Bedingung ist die Existenz einer vollständigen Menge für f.

  Im allgemeinen entsteht bei der Suche nach spabs das Problem, daß die Menge der Werte, die man aus einem Term $f(t)$ durch Auswertung unter allen Variablenbelegungen erhält, nicht mehr durch eine Formel der Prädikatenlogik erster Stufe beschrieben werden kann. In diesem Fall wird im allgemeinen zu einer Formel w, die einen solchen Teilausdruck $f(t)$ enthält, keine spab existieren.

  Die Existenz eine vollständigen Menge für f löst dieses Problem. Dabei heißt eine endliche Menge C von Termen *vollständig* für f, falls

  – alle Terme in C nur aus Konstruktoren und Variablen von Sorten aus der Parametersignatur aufgebaut sind und
  – für jede Parameteralgebra A und jedes a aus $[\![m]\!](A)$ von entsprechender Sorte ein $t \in C$ und eine Variablenbelegung α existiert, so daß die Anwendung von f auf a in $[\![m]\!](A)$ den gleichen Wert ergibt wie die Auswertung von $f(t)$ in $[\![m]\!](A)$ unter der Variablenbelegung α.

  Im Beispiel von Abbildung 1 ist die einelementige Menge $\{atom(x)\}$ eine vollständige Menge sowohl für *suml* als auch *sumr*. Die Begründung dafür ist, daß für jede Parameteralgebra A und jeden Wert der Sorte *list* in der Form
$$a = cons(a_1, \ldots, cons(a_{n-1}, atom(a_n)) \ldots)$$
  für jede Variablenbelegung α mit α mit
$$\alpha(x) = (a_1 + \ldots + (a_{n-1} + a_n) \ldots)$$

die Anwendung von *suml* auf a in $[\![m]\!](A)$ den gleichen Wert ergibt wie die Auswertung des Termes $suml(atom(x))$ in der Belegung α.

Dies bedeutet, daß eine Formel

$$\forall y : s_1 \,.\, P(f(y))$$

wobei P irgendeine Formel ist, die y nur im Kontext $f(y)$ enthält, für alle Paramteralgebren A äquivalent ist zu

$$\bigwedge_{t \in C} \forall \mathcal{V}ar\langle t \rangle \,.\, P(f(t))$$

Dabei steht $\forall \mathcal{V}ar\langle t \rangle$ für eine Allquantifizierung über alle freien Variablen von t. Da f nun partiell symbolisch auswertbar ist, können wir $f(t)$ dann jeweils zu einem Term ohne Vorkommen einer Variablen von neuer Sorte auswerten. Wir haben somit insgesamt die allquantifizierte Variable y von neuer Sorte durch mehrere allquantifizierte Variablen über Parametersorte ersetzt. Der Beweis von Satz 6 besteht nun im wesentlichen darin, eine Formel in eine Form zu bringen, in der die oben erklärte Argumentation anwendbar ist. Das Problem dabei ist, daß die Bedingung "y kommt in P nur in der Form $f(y)$ vor" im allgemeinen nicht erfüllt ist.

Das Modul aus Abbildung 1 erfüllt die Bedingungen für die Anwendbarkeit unseres Berechnungsverfahren nicht, da es zwei durch rekursive Programme definierte Funktionen enthält. Streicht man jedoch eine der beiden Funktionen *suml* oder *sumr* aus dem Modul weg, so sind die Bedingungen erfüllt. Wir werden am Ende des Kapitels 6 zeigen, daß bezüglich des Modules aus Abbildung 1 (mit beiden Funktionen *suml* und *sumr*) im allgemeinen zu einer Formel w eine spab nicht existiert. Dies zeigt, daß zumindest ohne eine weitere Verschärfung der Bedingungen an die rekursiven Funktionen einer Verallgemeinerung unserer Methode nicht möglich ist.

1.5 Parameterbedingungenprobleme und Reduktionen

In Kapitel 7 werden wir die Probleme der Existenz und Berechenbarkeit von spabs formaler fassen. Dazu definieren wir den Begriff des *Parameterbedingungenproblems*, abgekürzt *PbP*. Ein PbP ist ein Tripel, bestehend aus

- einem Domainoperator $\Im$
- einem Modul m und
- einer Menge W von Formeln über der Exportsignatur von m.

Eine *Lösung* eines solchen PbP besteht aus einem Algorithmus, der zu jeder Formel $w \in W$ eine $\Im, m$-spab berechnet. PbP's haben drei "Freiheitsgrade", durch Änderung des Domainoperators, des Moduls und natürlich der Menge W können wir die Lösbarkeit eines PbP entscheidend verändern. Wir führen deshalb einen Begriff von "Reduktion" eines PbP auf ein anderes an. Dabei ist ein PbP P_1 auf ein anderes PbP P_2 reduzierbar, falls wir aus einer Lösung von P_2

unmittelbar eine Lösung von P_1 konstruieren können. Das Problem P_1 ist also "höchstens so schwierig wie" das Problem P_2.

In Abschnitt 7.2 werden wir zeigen, daß sich eine PbP $P_1 = (\Im^f, m, W)$ stets auf ein geeignetes PbP $P_2 = (\Im^{st}, m', W')$ reduzieren lässt (Satz 7). Dabei sind die in m' durch rekursive Programme definierten Funktionen von einer einfachen Rekursionsstruktur. Da darüber hinaus P_2 als Domainoperator $\Im^{st}$ benutzt, werden dann alle Funktionen in $[\![m']\!](A)$ für alle Paramteralgebren $A \in \Im^{st}_{\Sigma_P}$ total sein. Dies bedeutet, daß in dem PbP P_2 der undefinierte Wert $\perp$ für die Logik keine Rolle mehr spielt.

Insgesamt besagt Satz 7 also, daß die Möglichkeit der Nichtterminierung von Berechnungen von neuen Funktionen, bzw. von undefinierten Werten bei der Auswertung von Funktionen der Parameteralgebra, das Problem des Auffindens von spabs nicht erschwert.

1.6 Eine Methode für Unentscheidbarkeitsbeweise

In Abschnitt 8 werden wir eine allgemeine Methode zum Beweis der Unentscheidbarkeit einer Theorie entwickeln. Aus der Literatur sind verschiedene andere Methoden zum Beweis der Unentscheidbarkeit von prädikatenlogischen Theorien bekannt. In [Tar53a] wurde gezeigt, daß eine Theorie T unentscheidbar ist, falls eine wesentlich unentscheidbare (essentially undecidable) und endlich axiomatisierbare Theorie T' relativ schwach interpretierbar (relatively weak interpretable) in T ist. Dabei heißt eine Theorie *wesentlich unentscheidbar*, falls jede ihrer konsistenten Erweiterungen unentscheidbar ist. Der Nachweis der relativen schwachen Interpretierbarkeit von T' in T besteht, grob gesagt, in der Konstruktion von Formeln, die das Universum und die Operationen von T' in einer geeigneten konsistenten Erweiterung von T beschreiben. Die Reduktion wird also in der Sprache der Prädikatenlogik erster Stufe selbst ausgedrückt, man benötigt aber eine geeignete endlich axiomatisierbare und wesentlich unentscheidbare Theorie. Die Theorie Q, das ist im wesentlichen die Peano-Arithmetik ohne Induktion, hat diese Eigenschaften ([TMR53a]). Diese Methode kann dazu benutzt werden, die Unentscheidbarkeit unvollständiger Theorien (z. B. die von einem Axiomensystem beschriebene Theorie) zu zeigen. In [Tar53b] wurde dies benutzt, um die Unentscheidbarkeit der Gruppentheorie zu beweisen.

Die in [Rab65] beschriebene Methode erfordert hingegen nicht die endliche Axiomatisierbarkeit der zur Rückführung benutzten unentscheidbaren Theorie. Um nach [Rab65] die Unentscheidbarkeit einer Theorie nachzuweisen, muß man, wiederum vereinfacht ausgedrückt, folgendermaßen vorgehen: Finde eine geeignete unentscheidbare Theorie T' und Formeln über der Sprache von T, die für jedes Modell von T neue Operationen in einem Teiluniversum dieses Modells ausdrücken. Zeige nun eine eins-zu-eins Korrespondenz zwischen den auf diese Weise neu konstruierten Modellen und den Modellen von T'. Die Rückführung selbst wird also auch hier wieder in der Prädikatenlogik ausgedrückt, der Beweis benutzt aber eine Aussage über die Modelle der verschiedenen Theorien.

Unsere Methode beruht hingegen auf einem anderen Prinzip: Wir nutzen wesentlich die Eigenschaften der *Modelle* der betrachteten Theorien aus, im Gegensatz zu Eigenschaften der

<table>
<tr><th></th><th>Unifikation</th><th>Σ_1</th><th>Σ_2</th><th>Σ_3</th></tr>
<tr><td>1 Konstante
+1 AC</td><td colspan="2" rowspan="2"></td><td colspan="2">$\surd$ [Com88], [Pre29]</td></tr>
<tr><td>n Konstanten
+1 AC</td><td colspan="2">$\surd$ Satz 17 dieser Arbeit</td></tr>
<tr><td>1 Konstante
+ mind. 1 Funktion
+ mind. 1 AC</td><td>$\surd$ [Sti81]
$\surd$ [Kir85]
$\surd$ [Fag84]</td><td>$\surd$ [Com88]</td><td>?</td><td>$\bigcirc$ Satz 9 dieser Arbeit</td></tr>
</table>

In der Tabelle steht "Funktion" für eine mindestens einstellige Funktion, dies schließt auch die Möglichkeit einer binären AC Funktion ein. $\surd$ bedeutet entscheidbar, $\bigcirc$ unentscheidbar.

Abbildung 2: Übersicht der Resultate zu Gleichungsproblemen modulo AC

Theorien selbst. Die Rückführung geschieht nun auf der Ebene der Modelle, nicht auf der Ebene der Theorie.

Unsere allgemeine Methode illustrieren wir mit einigen konkreten Unentscheidbarkeitsbeweisen, die teilweise neue Ergebnisse liefern. Wir geben hier einen Überblick über die einzelnen Resultate und vergleichen diese mit bekannten Ergebnissen aus der Literatur.

Das erste Anwendungsfeld unsere Methode sind Gleichungsprobleme (equational problems), wie sie in [Com90a] und [BSS89] eingeführt wurden. Gleichungsprobleme stellen eine Verallgemeinerung von Unifikationsproblemen (siehe [Sie89] für einen Überblick über Unifikationsprobleme) dar. Dabei handelt es sich um die Theorie der initialen, bzw. der freien Algebra einer Gleichungsspezifikation (equational specification). Die initiale Algebra ist der Quotient einer Grundtermalgebra nach der von einer endlichen Gleichungsmenge erzeugten Kongruenzrelation, während bei der freien Algebra der Quotient auf die von einer unendlichen Menge von zusätzlichen Konstanten, die aber nicht Bestandteil der Sprache der Logik sind, erzeugte freie Algebra angewandt wird.

Eine wichtige Anwendung betrifft die Theorie einer Grundtermalgebra modulo den Axiomen für Assoziativität und Kommutativität, abgekürzt AC, (mindestens) eines binären Funktionssymbols. Eine Übersicht über die Resultate zur Entscheidbarkeit von Gleichungsproblemen modulo AC findet man in Abbildung 2. Unifikationsalgorithmen für AC Funktionen bei Anwesenheit von freien Funktionszeichen wurden in [Sti81] (die Terminierung konnte aber erst in [Fag84] bewiesen werden) und [Kir85] gegeben. In [Com88] wurde die Entscheidbarkeit des existentiellen Fragments bewiesen. Dort wurde außerdem festgestellt, daß der Fall einer Konstanten und eines AC Funktionssymbols eine Teiltheorie der Presburger Arithmetik darstellt und daher ([Pre29]) entscheidbar ist. Wir werden in Satz 17 zeigen, daß sich diese Argumentation leicht

auf den Fall von endlich vielen Konstanten und einem AC Funktionssymbol erweitern läßt. Wir zeigen in Satz 9, daß das Σ_3-Fragment unentscheidbar ist, falls die Signatur mindestens eine Konstante, ein mindestens einstelliges Funktionszeichen sowie ein AC Funktionszeichen enthält. In [Com88] und [Les88] wurde dies als offenes Problem herausgestellt. Unser Unentscheidbarkeitsresultat kann leicht auf freie Algebren und auf Gleichungsprobleme modulo ACI, d. h. Assoziativität, Kommutativität und Idempotenz erweitert werden. Die Verallgemeinerung auf freie Algebren widerlegt die in [BSS89] geäußerte Vermutung, daß Gleichungsprobleme in freien Termalgebren (general equational problems) entscheidbar sind, falls Unifikation mit freien Funktionszeichen entscheidbar ist.

Eine zweite Anwendung aus dem Gebiet der Gleichungsprobleme ist die Theorie einer Grundtermalgebra modulo Assoziativität. In [Qui46] wurde die Unentscheidbarkeit der Theorie der Stringkonkatenation gezeigt, dies entspricht der Grundtermalgebra eines binären Funktionszeichens und zweier Konstanten modulo der Assoziativität. Der dort gegebene Beweis benutzt eine Rückführung der vollständigen Zahlentheorie auf die Theorie der Stringkonkatenation, es wird aber nicht versucht, die Unentscheidbarkeit für ein möglichst kleines Fragment zu zeigen. Benutzt man für die Rückführung Hilberts zehntes Problem, von dem wir heute wissen, daß es unentscheidbar ist ([Mat70]), so erhalten wir aus dem Beweis von [Qui46] die Unentscheidbarkeit des Σ_6-Fragments. In [Plo72] wurde ein Unifikationsalgorithmus für Termalgebren modulo Assoziativität angegeben, der die (möglicherweise unendliche) Menge von Lösungen (most general unifiers) erzeugt. Die Entscheidbarkeit der Unifizierbarkeit zweier Terme modulo der Assoziativität wurde in [Plo72] vermutet und in [Mak77] bewiesen. Wir zeigen in Satz 16, daß das Σ_2-Fragment unentscheidbar ist, falls die Signatur mindestens eine Konstante, ein nicht konstantes Funktionszeichen und ein assoziatives Funktionszeichen enthält. Im Vergleich zum Unentscheidbarkeitsresultat für AC fällt auf, das wir bei Assoziativität alleine Unentscheidbarkeit für Σ_2, im AC Fall aber nur für Σ_3 beweisen konnten. Einen Hinweis auf die Gründe hierfür liefert die Unifikationshierarchie von [BHSS87], in der AC finitär ist, Assoziativität aber den Unifikationstyp ∞ hat. Dies bedeutet, daß bereits das Unifikationsproblem im assoziativen Falle in einem schwächeren Sinne lösbar ist als im AC Fall.

Ein weiteres wichtiges Anwendungsgebiet umfaßt die Theorie einer Grundtermalgebra mit einer Ordnung auf Termen. Die Unentscheidbarkeit des Σ_2-Fragmentes der Theorie der Teiltermrelation wurde in [Ven87] für Signaturen mit mindestens einer Konstanten, zwei unären Funktionszeichen und einem ternären Funktionszeichen gezeigt. Wir zeigen die Unentscheidbarkeit des Σ_2-Fragments für Signaturen, die mindestens eine Konstante und ein binäres Funktionszeichen enthalten. Unser Beweis umfaßt nicht nur das Unentscheidbarkeitsresultat von [Ven87], der Beweis ist auch konzeptionell einfacher, wie wir später noch ausführen werden. Damit ist die Fallunterscheidung für die Theorie der Teiltermrelation komplett (siehe auch Abbildung 3): Die Entscheidbarkeit des Σ_1-Fragmentes wurde in [Ven87] bewiesen. [Rab69] zeigt die Entscheidbarkeit der Theorie für den Fall einer Konstanten und endlich vieler Funktionszeichen, dies läßt sich sehr einfach auf den Fall endlich vieler Konstanten zusammen mit endlich vielen unären Funktionszeichen verallgemeinern.

Wir erweitern unser Unentscheidbarkeitsresultat auf die Algebra der unendlichen Bäume mit der Teiltermrelation. Die Entscheidbarkeit des existentiellen Fragments dieser Theorie wurde

	Σ_1	Σ_2
1 Konstante +1 unäre Funktion		$\surd$ [Pre29]
n Konstanten +n unäre Funktionen		$\surd$ [Rab69]
1 Konstante + mind. 1 binäre Funktion		◯ Satz 16 dieser Arbeit
1 Konstante +2 unäre Funktionen +1 ternäre Funktion	$\surd$ [Ven87]	◯ [Ven87]

$\surd$ bedeutet entscheidbar, ◯ unentscheidbar.

Abbildung 3: Übersicht der Resultate zur Theorie der Teiltermrelation

in [Tul90] gezeigt.

In [Com88] wurde die Frage nach der Entscheidbarkeit einer totalen Simplifikationsordnung (total simplification ordering) gestellt. Die Entscheidbarkeit des existentiellen Fragments einer totalen lexikographischen Pfadordnung wurde in [Com90b], die einer totalen Multimengen Pfadordnung in [JO91] bewiesen. In Satz 11 werden wir die Unentscheidbarkeit des Σ_4-Fragments einer partiellen lexikographischen, bzw. Multimengen Pfadordnung zeigen. Damit bleiben in diesem Fall noch einige Lücken zwischen den Entscheidbarkeits- und den Unentscheidbarkeitsresultaten (siehe auch Abschnitt 8.3).

1.7 Beziehungen zwischen $\models$ und $[m]\models$

Nachdem wir uns bisher mit der Existenz schwächster Parameterbedingungen beschäftigt haben, wollen wir in Kapitel 9 andere mögliche Zusammenhänge zwischen der Prädikatenlogik erster Stufe und $[m]\models$ untersuchen. Wie wir gesehen haben, existiert aus Gründen, die in der Ausdruckskraft der Prädikatenlogik erster Stufe liegen, in vielen Fällen keine schwächste Parameterbedingung zu einem Satz w. Andererseits ist aber trivialerweise der Satz `False` eine Parameterbedingung eines jeden Satzes w. `False` ist dabei die stärkste Parameterbedingung, da sie in keinem Modell gültig ist und somit jede andere Parameterbedingung impliziert. Eine Parameterbedingung `False` hilft uns natürlich nicht weiter, wir können uns aber fragen, ob es vielleicht eine *schwächste ausdrückbare* Parameterbedingung eines Satzes gibt. Diese schwächste ausdrückbare Parameterbedingung ist dann eine hinreichende Bedingung für die

Gültigkeit eines Satzes bezüglich $[m]\models$, mit der wir uns aber unter Umständen zufrieden geben können. Wir werden zeigen, daß im allgemeinen aber auch eine schwächste ausdrückbare Parameterbedingung nicht existiert.

Den Namen "schwächste ausdrückbare Parameterbedingung" wählen wir dabei in Anlehnung an eine vergleichbare Situation in der Hoare-Logik für die partielle Korrektheit imperativer Programme. Dort liefert der Begriff der schwächsten ausdrückbaren Vorbedingung (weakest expressible precondition) eine Abschwächung des Begriffes der schwächsten Vorbedingung (weakest precondition), so daß die Existenz von schwächsten ausdrückbaren Vorbedingungen (precondition completness) noch ausreicht zum Beweis der relativen Vollständigkeit des Hoare-Kalküls ([Sie82]).

Anschließend werden wir einige weitere Beziehungen zwischen der Prädikatenlogik erster Stufe und unserer Logik $[m]\models$ untersuchen.

Ich möchte an dieser Stelle Herrn Professor Loeckx für die Anregung zu dieser Arbeit sowie den folgenden Personen für Diskussionen, ihre Ermutigungen und technische Unterstützung danken: Peter Buhmann, Thomas Lehmann, Joachim Philippi, Ralf Scheidhauer, Kurt Sieber, Gert Smolka, Stephan Uhrig und insbesondere Hubert Comon.

2 Grundlagen

In diesem Kapitel werden wir einige grundlegende Begriffe und Sätze aus Informatik, Logik und Mathematik aufführen. Dieses Kapitel sollte dem Leser in erster Linie als Referenz dienen. In vielen Fällen sind die hier erwähnten Begriffe aus der englischen Literatur bekannt, wir geben deshalb bei der Einführung eines deutschen Begriffes manchmal das englische Analogon in Klammern an.

2.1 Mengen, Multimengen und Funktionen

Wir setzen die Notationen der elementaren Mengenlehre voraus (siehe z.B. Anhang A in [CK90]). Die (unsymmetrische) *Mengendifferenz* wird bezeichnet durch

$$A \setminus B := \{x \in A \mid x \notin B\}$$

$\mathcal{P}(M)$ ist die *Potenzmenge* der Menge M.

Eine *Multimenge* (multiset, bag) über einer Grundmenge M ist eine Funktion $m : M \to \mathbb{N}$. Man kann sich eine Multimenge m als eine Menge von Elementen von M vorstellen, wobei aber endlich viele mehrfache Vorkommen der Elemente möglich sind. Eine Multimenge m heißt *endlich*, falls es nur endlich viele x gibt mit $m(x) \neq 0$. Wir benutzen auch für Multimengen die Mengenschreibweise, also ist z.B. $\{1, 1, 1, 5, 8\}$ eine Multimenge über $\mathbb{N}$. $\mathcal{M}(M)$ ist die Menge der endlichen Multimengen über M. Zwei Multimengen m_1, m_2 sind gleich, falls $m_1(x) = m_2(x)$ für alle x der Grundmenge. Die Vereinigung $m_1 \cup m_2$ zweier Multimengen ist definiert durch $(m_1 \cup m_2)(x) = m_1(x) + m_2(x)$. Die Differenz zweier Multimengen ist

$$(m_1 \setminus m_2)(x) = \begin{cases} m_1(x) \dot{-} m_2(x) & \text{falls } m_1(x) \geq m_2(x) \\ 0 & \textit{sonst} \end{cases}$$

Falls $\Phi \subseteq A \times B$ eine Relation zwischen A und B ist, dann bezeichnet für $a \in A$ $a\Phi := \{b \in B \mid a\Phi b\}$ und analog für $b \in B$ $\Phi b := \{a \in A \mid a\Phi b\}$.

Wir betrachten stets totale Funktionen. $(A \to B)$ ist die Menge aller Funktionen mit Definitionsbereich A und Wertebereich B. Für $f \in (A \to B)$ und $M \subseteq A$ ist $f(M) := \{f(m) \mid m \in M\}$. Falls $f\colon A_1, \ldots, A_n \to A$ eine Funktion ist und $B_i \subseteq A_i$ für $i = 1, \ldots, n$, dann bezeichnet $f\,|_{B_1, \ldots, B_n}$ die *Einschränkung* (restriction) von f auf $B_1 \times \ldots \times B_n$. Falls $f\colon M \to N$ eine Funktion ist, $m_1, \ldots, m_l$ paarweise verschiedene Elemente von M und $n_1, \ldots, n_l$ beliebige Elemente von N, dann ist die Funktion

$$f[m_1 \mapsto n_1, \ldots, m_l \mapsto n_l]\colon M \to N$$

gegeben durch

$$f[m_1 \mapsto n_1, \ldots, m_l \mapsto n_l](x) = \begin{cases} n_i & \text{falls } x = m_i \\ f(x) & \text{sonst} \end{cases}$$

Die *Komposition* zweier Funktionen $g: A \to B$ und $f: B \to C$ ist die Funktion $f \circ g: A \to C$, definiert durch $(f \circ g)(x) = f(g(x))$ für alle $x \in A$.

Unendliche Produkte sind Funktionen, deren Wertebereich von ihrem Argument abhängt. Falls M eine Menge ist und falls R_i eine Menge ist für jedes $i \in M$, dann ist

$$\prod_{i \in M} R_i = \{f: M \to \bigcup_{i \in M} R_i \mid f(i) \in R_i \text{ für alle } i \in M\}$$

$\mathbb{N}$ bezeichnet die Menge der natürlichen Zahlen inklusive der Zahl 0. Für eine Menge M ist M^* die Menge der endlichen Folgen über M, d.h.

$$M^* = \{m: \{1, \ldots, n\} \to M \mid n \in \mathbb{N}\}$$

Falls $m \in M^*$ dann schreiben wir auch m_i für $m(i)$ und notieren die Folge m in der Form $(m_i)_{i \in \{1,\ldots,n\}}$. In diesem Fall ist $n = lth(m)$ die *Länge* der Liste m. ϵ bezeichnet die leere Folge, das ist die Funktion mit leerem Definitionsbereich. Weiterhin ist M^+ die Menge der nichtleeren Folgen $M^+ := M^* \setminus \{\epsilon\}$. wv bezeichnet die *Konkatenation* der beiden Listen w und v, das heißt

$$wv = w_1 \cdots w_{lth(w)} v_1 \cdots v_{lth(v)}$$

In diesem Fall ist w ein *Präfix* von wv, in Zeichen $w \triangleleft wv$.

2.2 Signaturen und Algebren

Eine ausführliche Behandlung dieser Begriffe und ihrer Eigenschaften kann man beispielsweise in [EM85] finden. Wir werden uns hier in den verwendeten Notationen meist an den in [DJ91] definierten Standard halten (siehe auch [DJ90]).

Eine *Signatur* ist ein Paar von Mengen

$$\Sigma = (S, F)$$

In diesem Fall heißen die Elemente von S die *Sorten* von Σ und die Elemente von F die *Funktionszeichen* von Σ. Weiterhin assoziiert jede Signatur $\Sigma = (S, F)$ zu jedem Funktionszeichen $f \in F$ eine *Stelligkeit* (arity, rank)

$$rank(f) \in S^+$$

F heißt dann auch *S-sortiert*. Es ist üblich, diese Zuordnung in der folgenden Art darzustellen: Statt "$f \in F$ mit $rank(f) = (s_1, \ldots, s_n, s_{n+1})$" schreiben wir:

$$f: s_1, \ldots, s_n \to s_{n+1} \in F$$

Die Mengen S und F dürfen dabei durchaus unendlich sein. Eine S-sortierte Menge C von Konstantenzeichen ist eine *Umbenennung* einer S-sortierten Menge C' von Konstantenzeichen, falls es eine Bijektion $\phi: C \to C'$ gibt mit $rank(c) = rank(\phi(c))$ für alle $c \in C$.

Eine Signatur $\Sigma_1 = (S_1, F_1)$ ist eine *Teilsignatur* einer Signatur $\Sigma_2 = (S_2, F_2)$, i.Z. $\Sigma_1 \subseteq \Sigma_2$, falls $S_1 \subseteq S_2$, $F_1 \subseteq F_2$ und alle Funktionszeichen aus F_1 haben in Σ_2 die gleiche Stelligkeit wie in Σ_1. Falls $\Sigma = (S, F_1)$ eine Signatur ist und F_2 eine Menge von Funktionszeichen mit $rank(f) \in S^+$ für alle $f \in F_2$ und $F_1 \cap F_2 = \emptyset$, dann ist die *Erweiterung* von Σ um F_2 die Signatur $\Sigma \cup F_2 := (S, F_1 \cup F_2)$.

Es sei nun $\Sigma = (S, F)$ eine Signatur. Eine Σ-*Algebra* A besteht aus

- einer Menge s^A zu jeder Sorte $s \in S$, genannt die *Trägermenge* von A der Sorte s und
- einer Funktion $f^A: s_1^A, \ldots, s_n^A \to s^A$ zu jedem Funktionszeichen $f: s_1, \ldots, s_n \to s \in F$.

Die Elemente der Trägermengen nennen wir auch *Träger*, die Funktion f^A ist die *Denotation* des Funktionszeichens f. Für eine Signatur $\Sigma = (S, F)$ heißt eine Σ-Algebra A eine *Teilalgebra* (subalgebra) einer Σ-Algebra B, i.Z. $A \subseteq B$, falls gilt:

- Für alle $s \in S$ ist $s^A \subseteq s^B$
- Für alle $f: s_1, \ldots, s_n \to s \in F$ ist $f^A = f^B \mid_{s_1^A, \ldots, s_n^A}$

Falls Σ_1 eine Teilsignatur von Σ_2 ist und A eine Σ_2-Algebra, dann ist $A \mid_{\Sigma_1}$ die *Einschränkung* (restriction) von A auf Σ_1, das ist die Algebra, die wir erhalten, wenn wir alle nicht zu Σ_1 gehörenden Komponenten vergessen.

Ein *Homomorphismus* zwischen zwei (S, F)-Algebren A und B ist eine Familie $(\phi_s)_{s \in S}$ von Funktionen $\phi_s: s^A \to s^B$ mit

$$\phi_s(f^A(a_1, \ldots, a_n)) = f^B(\phi_{s_1}(a_1), \ldots, \phi_{s_n}(a_n))$$

für alle $f: s_1, \ldots, s_n \to s \in F$ und $a_i \in s_i^A$. Falls dabei alle ϕ_s Bijektionen sind, dann bezeichnen wir A und B als *isomorph*.

Die Kardinalität $\#(A)$ einer Algebra A ist die Summe der Kardinalitäten ihrer Trägermengen. Insbesondere bezeichnen

- $\aleph_0 = \#(\mathbb{N})$ die abzählbar unendliche Kardinalität und
- $\aleph_1 = 2^{\aleph_0} = \#\mathcal{P}(\mathbb{N})$ die kleinste überabzählbare Kardinalität.[3]

Eine kompakte Darstellung der Kardinalzahlen findet man in Anhang A von [CK90].

[3] Wir setzen also die Kontinuumshypothese voraus.

2.3 Variablen und Terme

Eine *Variablenfamilie* für eine Signatur $\Sigma = (S, F)$ ist eine Familie $X = (X_s)_{s \in S}$ von Mengen von Variablensymbolen, die jeweils paarweise disjunkt sind und disjunkt zu F.

Es sei nun $\Sigma = (S, F)$ eine Signatur und $X = (X_s)_{s \in S}$ eine Variablenfamilie für Σ. Wir definieren durch simultane induktive Definition

- die Menge $\mathcal{T}(\Sigma, X)_s$ für $s \in S$ der Σ, X-Terme der Sorte s und damit auch $\mathcal{T}(\Sigma, X) := \bigcup_{s \in S} \mathcal{T}(\Sigma, X)_s$,
- die Funktion $\mathcal{V}ar\langle\rangle : \mathcal{T}(\Sigma, X) \to \mathcal{P}(X)$, die jeden Term auf die Menge seiner freien Variablen abbildet und
- die Teiltermrelation $\trianglelefteq$ auf $\mathcal{T}(\Sigma, X)$

Dabei benutzen wir eine notationelle Konvention, die wir in dieser Arbeit noch öfters einsetzen werden. Neben der "Objekt"-Termsprache $\mathcal{T}(\Sigma, X)$ betrachten wir auch Terme auf einer "Meta"-Ebene, wie z. B. den "Meta"-Term $\mathcal{V}ar\langle t\rangle$. Dies kann zu einigen Verwirrungen führen, insbesondere wenn später solche Meta-Terme benutzt werden, um Objektterme zu bezeichnen (in diesem Fall haben wir dann eine Berechnungsvorschrift für solche Metaterme zur Verfügung). Um die beiden Ebenen besser auseinanderhalten zu können, verwenden wir für die Funktionsanwendung auf der Objektebene runde Klammern (,) und für die Anwendung von Meta-Funktionen spitze Klammern $\langle$, $\rangle$.

1. Falls $x \in X_s$, dann ist
   - $x \in \mathcal{T}(\Sigma, X)_s$,
   - $\mathcal{V}ar\langle x\rangle = \{x\}$ und
   - $x \trianglelefteq x$.
2. Falls $t_i \in \mathcal{T}(\Sigma, X)_{s_i}$ für $i = 1, \ldots, n$ und $f : s_1, \ldots, s_n \to s \in F$, dann ist
   - $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, X)_s$,
   - $\mathcal{V}ar\langle f(t_1, \ldots, t_n)\rangle = \bigcup_{i=1}^{n} \mathcal{V}ar\langle t_i\rangle$
   - $f(t_1, \ldots, t_n) \trianglelefteq f(t_1, \ldots, t_n)$ und außerdem $r \trianglelefteq f(t_1, \ldots, t_n)$ falls $r \trianglelefteq t_i$ für ein i.

Wir definieren nun die *simultane Substitution* in Termen. Sei Σ eine Signatur, X eine Variablenfamilie für Σ, $t \in \mathcal{T}(\Sigma, X)$, $t_i \in \mathcal{T}(\Sigma, X)_{s_i}$ und $x_i \in X_{s_i}$ für $i = 1, \ldots, n$. Dann ist

- ${x_i}^{t_1 \ldots t_n}_{x_1 \ldots x_n} = t_i$, falls $1 \leq i \leq n$
- $x^{t_1 \ldots t_n}_{x_1 \ldots x_n} = x$, falls $x \notin \{x_1, \ldots, x_n\}$
- $f(r_1, \ldots, r_m)^{t_1 \ldots t_n}_{x_1 \ldots x_n} = f({r_1}^{t_1 \ldots t_n}_{x_1 \ldots x_n}, \ldots, {r_m}^{t_1 \ldots t_n}_{x_1 \ldots x_n})$

Falls es Variablen $x_1, \ldots, x_n$ und Terme $t_1, \ldots, t_n$ gibt, so daß $u = r^{t_1 \ldots t_n}_{x_1 \ldots x_n}$, dann *subsumiert* der Term r den Term u, in Zeichen $r \trianglelefteq u$. Zwei Terme t_1, t_2 heißen *unifizierbar*, falls es Variablen $x_1, \ldots, x_m$ und Terme $r_1, \ldots, r_m$ gibt mit $t_1{}^{r_1 \ldots r_m}_{x_1 \ldots x_m} = t_2{}^{r_1 \ldots r_m}_{x_1 \ldots x_m}$.
Weiterhin benutzen wir für ein unäres Funktionszeichen $f: s \to s$, $t \in \mathcal{T}(\Sigma, X)$ und $n \in \mathbb{N}$ die abkürzende Schreibweise $f^n(t)$, wobei

$$f^n(t) := \begin{cases} t & \text{falls } n = 0 \\ f^{n-1}(f(t)) & \text{falls } n \geq 0 \end{cases}$$

Die Funktion $|: \mathcal{T}(\Sigma, X), \mathbb{N} \rightsquigarrow \mathcal{T}(\Sigma, X)$ liefert uns einen Teilterm eines Termes an einer bestimmten Position. Dabei handelt es sich um eine partielle Funktion, da eine Position in einem Term möglicherweise nicht existiert. Wir schreiben $|\,(t, o)$ in Infixschreibweise $t\,|_o$ und definieren:

$$x\,|_o \quad \begin{cases} = x & \text{falls } o = \epsilon \\ \text{ist undefiniert} & \text{anderenfalls} \end{cases} \qquad x \in X$$

$$f(t_1, \ldots, t_n)\,|_o \quad \begin{cases} = t_i\,|_p & \text{falls } o = ip \text{ für ein } i \in \{1, \ldots, n\} \text{ und } p \in \mathbb{N}^* \\ \text{ist undefiniert} & \text{anderenfalls} \end{cases}$$

Für eine Variablenfamilie $X = (X_s)_{s \in S}$ und eine (S, F)-Algebra A bezeichnet $?_{X,A}$ die Menge der A-Variablenbelegungen:

$$?_{X,A} = \{\bigcup_{s \in S} \gamma_s \mid \gamma_s \in (X_s \to s^A)\}$$

Eine Σ-Algebra A induziert eine Interpretationsfunktion für Σ-Terme, die wir dann ebenfalls mit A bezeichnen:

$$A: \mathcal{T}(\Sigma, X) \to (?_{X,A} \to A)$$

Die Interpretation ist induktiv definiert durch

- $A(x)(\alpha) = \alpha(x)$ für $x \in X$
- $A(f(t_1, \ldots, t_n))(\alpha) = f^A(A(t_1)(\alpha), \ldots, A(t_n)(\alpha))$

Es gilt der folgende Substitutionssatz (siehe z.B. [LS87]):

$$A(t^{t_1 \ldots t_n}_{x_1 \ldots x_n})(\alpha) = A(t)(\alpha[x_1 \mapsto A(t_1)(\alpha), \ldots, x_n \mapsto A(t_n)(\alpha)])$$

2.4 Quotiententermalgebren

Für eine beliebige binäre Relation, die durch $\to$ oder ähnlich bezeichnet wird, ist

1. $\to^*$ der *reflexive transitive Abschluß* von $\to$, das ist die kleinste reflexive und transitive Relation die $\to$ enthält.

2. $\leftrightarrow^*$ der *reflexive symmetrische transitive Abschluß* von $\rightarrow$, das ist die kleinste reflexive, symmetrische und transitive Relation die $\rightarrow$ enthält.

3. $\rightarrow^!$ die Normalisierung von $\rightarrow$, d.h. $s \rightarrow^! t$ genau dann, wenn $s \rightarrow^* t$ und es gibt kein u mit $t \rightarrow u$. In diesem Falle ist t eine $\rightarrow$-*Normalform* von s.

Eine binäre Relation $\otimes$ auf $\mathcal{T}((S,F),X)$ heißt *abgeschlossen unter Kontextanwendung*, falls für alle $f\colon s_1,\ldots,s_n \rightarrow s \in F$ gilt:

$$t_1 \otimes t_1' \wedge \ldots \wedge t_n \otimes t_n' \Rightarrow f(t_1,\ldots,t_n) \otimes f(t_1',\ldots,t_n)$$

Falls E eine Menge von $(S,F),X$-Gleichungen ist, also

$$E \subseteq \bigcup_{s\in S} \mathcal{T}((S,F),X)_s \times \mathcal{T}((S,F),X)_s$$

dann bezeichnet $=_E$ den *Kongruenzabschluß* von $\otimes_E$, also die kleinste reflexive, symmetrische, transitive und unter Kontextanwendung abgeschlossene Relation, die $\otimes_E$ enthält. Dabei ist $t_1 \otimes_E t_2$ genau dann, wenn es eine Gleichung $(l,r) \in E$, Variablen $x_1,\ldots,x_m$ und Terme $r_1,\ldots,r_m$ gibt mit

$$l^{r_1\ldots r_m}_{x_1\ldots x_m} = t_1 \quad \text{und} \quad r^{r_1\ldots r_m}_{x_1\ldots x_m} = t_2$$

Die $=_E$-Klasse eines Termes $t \in \mathcal{T}(\Sigma,X)$ bezeichnen wir dann mit

$$[t]_E = \{r \in \mathcal{T}(\Sigma,X) \mid t =_E r\}$$

$\mathcal{T}(\Sigma,\emptyset)/E$ ist dann die *Quotientermalgebra* von Σ modulo E. Ihre Signatur ist Σ, und sie ist definiert durch

- $s^{\mathcal{T}(\Sigma,\emptyset)/E} = \{[t]_E \mid t \in \mathcal{T}(\Sigma,X)_s\}$
- $f^{\mathcal{T}(\Sigma,\emptyset)/E}([t_1]_E,\ldots,[t_n]_E) = [f(t_1,\ldots,t_n)]_E$

Näheres hierzu findet man in [EM85], wo auch die Wohldefiniertheit dieser Konstruktion gezeigt ist.

2.5 Ordnungen

Eine binäre Relation $\bowtie$ auf einer gegebenen Menge M ist eine *strikte partielle Ordnung*, falls $\bowtie$ irreflexiv und transitiv ist. $\bowtie$ heißt *partielle Ordnung*, falls $\bowtie$ reflexiv, transitiv und antisymmetrisch ist. Falls zusätzlich für alle $x,y \in M$ gilt: $x \bowtie y$ oder $y \bowtie x$, dann nennen wir $\bowtie$ auch eine *lineare Ordnung*. Falls $\bowtie$ eine partielle Ordnung auf M ist und $=_M$ die Gleichheitsrelation auf M, dann ist der *strikte Anteil* von $\bowtie$ die Ordnung $\bowtie \setminus =_M$. Hier ist die folgende Schreibweise nützlich: partielle Ordnungen werden durch ein Relationensymbol mit Unterstrich (z.B. $\underline{\leq}$, $\underline{\preceq}$, $\underline{\sqsubseteq}$, $\underline{\lhd}$, etc.) bezeichnet. Durch Weglassung dieses Unterstriches erhalten wir eine Bezeichnung für den strikten Anteil der Relation, also $<$, $\prec$, $\sqsubset$, $\lhd$, etc. Das gespiegelte Relationensymbol

($\leq$ wird zu $\geq$, $\lhd$ wird zu $\rhd$, etc.) steht für das Inverse der Ordnung, d.h. $x \geq y$ genau dann, wenn $y \leq x$.

Eine partielle Ordnung $\leq$ heißt *noethersch* (well-founded), falls es keine unendliche Kette

$$x_0 > x_1 > \ldots > x_i > \ldots$$

gibt. Beispiele für noethersche Ordnungen sind $\leq$ auf $\mathbb{N}$ und die Teiltermrelation $\trianglelefteq$ auf einer Menge $\mathcal{T}(\Sigma, X)$ von Termen. Wir geben nun zwei wichtige Methoden an, um aus gegebenen noetherschen Ordnungen neue zu konstruieren:

1. Falls $\preceq_i$ jeweils eine noethersche Ordnung auf Mengen M_i, $i = 1, \ldots, n$ ist, dann ist ihre *lexikographische Kombination*, definiert durch

$$(a_1, \ldots, a_n) \preceq (b_1, \ldots, b_n) \quad \Leftrightarrow \quad a_i = b_i \text{ für alle } i \text{ oder ex. } i \leq n \text{ mit } a_i \prec_i b_i \text{ und } a_j = b_j \text{ für alle } j < i$$

   eine nothersche Ordnung auf $M_1 \times \ldots \times M_n$ ([LS87]). Falls für alle i $M_i = M$ und $\preceq_i = \preceq$ für feste M, $\preceq$, dann sprechen wir auch von der *lexikographischen Erweiterung* von $\preceq$ und bezeichnen diese dann mit $\preceq^n_{lex}$.

2. Falls $\preceq$ eine noethersche Ordnung auf einer Menge M ist, dann ist die *Multimengen Erweiterung*, definiert durch

$$A \preceq_{mul} B \quad \Leftrightarrow \quad M = N \text{ oder ex. } X, Y \in \mathcal{M}(M) \text{ mit } A = (B \setminus X) \cup Y \text{ und für alle } y \in Y \text{ ex. } x \in X \text{ mit } y \prec x$$

   eine noethersche Ordnung auf $\mathcal{M}(M)$ ([DM79]). Intuitiv ist $A \preceq_{mul} B$, falls man A aus B konstruieren kann, indem man sukzessive jeweils ein Element aus B durch endlich viele kleinere ersetzt.

Eine wichtige Ordnung auf Termen ist die *Multimengen Pfadordnung*[4]. Sei (S, F) eine Signatur und $\prec$ eine partielle Ordnung auf F. Dann definieren wir die Ordnung $\preceq_{mpo}$ auf $\mathcal{T}((S, F), \emptyset)$ rekursiv durch

$$t = g(t_1, \ldots, t_n) \prec_{mpo} f(s_1, \ldots, s_n) = s$$

falls einer der folgenden Fälle gilt:

1. $t \preceq_{mpo} s_i$ für ein i
2. $g \prec f$ und $t_j \prec_{mpo} s$ für alle j
3. $f = g$ und $\{t_1, \ldots, t_n\} \preceq_{mpo_{mul}} \{s_1, \ldots, s_n\}$

[4] Diese Ordnung hieß ursprünglich ([Der82]) "recursive path ordering". Wir schliessen uns hier dem Vorschlag von [DJ91] an und reservieren diesen Namen für eine allgemeinere Konstruktion.

Es gilt ([Der82]): Falls $\preceq$ noethersch ist (man beachte, daß F durchaus eine unendliche Menge sein kann), dann ist auch $\preceq_{mul}$ noethersch.

Eine wichtige Abwandlung der Multiset Pfadordnung ist die *lexikographische Pfadordnung* $\preceq_{lpo}$ ([Der87]). Sie ist analog zur Multimengen Pfadordnung definiert:

$$t = g(t_1, \ldots, t_n) \prec_{lpo} f(s_1, \ldots, s_n) = s$$

gilt genau dann, falls einer der folgenden Fälle gilt:

1. $t \preceq_{lpo} s_i$ für ein i
2. $g \prec f$ und $t_j \prec_{lpo} s$ für alle j
3. $f = g$ und $\{t_1, \ldots, t_n\} \preceq_{lpo_{mul}} \{s_1, \ldots, s_n\}$ und $t_i \preceq_{lpo} s$ für alle i

Auch hier gilt, daß $\preceq_{lpo}$ noethersch ist, falls $\preceq$ noethersch ist.

Eine Übersicht über noethersche Ordnungen, wie sie für Terminierungsbeweise von Termersetzungssystemen benutzt werden, findet man in [Der87].

2.6 Prädikatenlogik erster Stufe

Wir wollen in diesem Abschnitt die Syntax und einige semantische Eigenschaften der Prädikatenlogik erster Stufe einführen. Dabei richten wir die Definitionen und Sätze an dem in dieser Arbeit behandelten Spezialfall aus: Wir betrachten nur Formeln, in denen als einziges Prädikatzeichen das Gleichheitssymbol "=" vorkommt. In diesem Fall entsprechen die Signaturen den prädikatenlogischen mehrsortigen Basen. Das Gleichheitssymbol wird stets als die Gleichheit in der zugehörigen Menge interpretiert, so daß wir die Bedeutung von = nicht eigens in der Definition eines semantischen Bereiches angeben müssen. Aus diesem Grunde können wir die Algebren als semantische Bereiche für unsere Formeln ansehen.

Es sei nun Σ eine Signatur und X eine Variablenfamilie für Σ. Wir definieren mit simultaner Induktion die Menge $\mathcal{W}ff(\Sigma, X)$ der Σ, X Formeln sowie die Erweiterung der Funktion $\mathcal{V}ar\langle\rangle$ auf $\mathcal{W}ff(\Sigma, X)$:

- Falls $t_1, t_2 \in \mathcal{T}(\Sigma, X)_s$, dann ist
  - $(t_1 = t_2) \in \mathcal{W}ff(\Sigma, X)$
  - $\mathcal{V}ar\langle(t_1 = t_2)\rangle = \mathcal{V}ar\langle t_1\rangle \cup \mathcal{V}ar\langle t_2\rangle$
- Falls $w_1, w_2 \in \mathcal{W}ff(\Sigma, X)$, dann ist
  - $(w_1 \wedge w_2) \in \mathcal{W}ff(\Sigma, X)$
  - $\mathcal{V}ar\langle(w_1 \wedge w_2)\rangle = \mathcal{V}ar\langle w_1\rangle \cup \mathcal{V}ar\langle w_2\rangle$
- Falls $w \in \mathcal{W}ff(\Sigma, X)$, dann ist

- $\neg w \in \mathcal{W}ff(\Sigma, X)$
- $\mathcal{V}ar\langle\neg w\rangle = \mathcal{V}ar\langle w\rangle$

- Falls $x \in X_s$ und $w \in \mathcal{W}ff(\Sigma, X)$, dann ist
  - $\exists x : s \,.\, w \in \mathcal{W}ff(\Sigma, X)$
  - $\mathcal{V}ar\langle\exists x : s \,.\, w\rangle = \mathcal{V}ar\langle w\rangle \setminus \{x\}$

Die Menge der Σ, X-Sätze $\mathcal{S}en(\Sigma, X)$ besteht aus den Formeln ohne freie Variablen:

$$\mathcal{S}en(\Sigma, X) = \{w \in \mathcal{W}ff(\Sigma, X) \mid \mathcal{V}ar\langle w\rangle = \emptyset\}$$

Wir definieren außerdem die folgenden Abkürzungen:

$(t_1 \neq t_2)$	steht für	$(\neg(t_1 = t_2))$
$(w_1 \vee w_2)$	steht für	$(\neg((\neg w_1) \wedge (\neg w_2)))$
$(w_1 \supset w_2)$	steht für	$((\neg w_1) \vee w_2)$
$(w_1 \mathrel{\text{⫤}} w_2)$	steht für	$(w_1 \supset w_2) \wedge (w_2 \supset w_1)$
$(\forall x : s \,.\, w)$	steht für	$(\neg(\exists x : s \,.(\neg w)))$

Weitere abkürzende Vereinbarungen zur Notation von Formeln findet man auf Seite 31.

Wir erweitern nun die Interpretationsfunktion für Terme zu einer Interpretation von Formeln

$$A\colon \mathcal{W}ff(\Sigma, X) \to (?_{X,A} \to \{\textsc{True}, \textsc{False}\})$$

per Induktion. Seien X, Σ wie oben und A eine Σ-Algebra.

1. $A(t_1 = t_2)(\alpha) = \begin{cases} \textsc{True} & \text{falls } A(t_1)(\alpha) = A(t_2)(\alpha) \\ \textsc{False} & \text{anderenfalls} \end{cases}$

2. $A(w_1 \wedge w_2)(\alpha) = \begin{cases} \textsc{True} & \text{falls } A(w_1)(\alpha) = A(w_2)(\alpha) = \textsc{True} \\ \textsc{False} & \text{anderenfalls} \end{cases}$

3. $A(\neg w)(\alpha) = \begin{cases} \textsc{True} & \text{falls } A(w)(\alpha) = \textsc{False} \\ \textsc{False} & \text{anderenfalls} \end{cases}$

4. $A(\exists x : s \,.\, w)(\alpha) = \begin{cases} \textsc{True} & \text{falls } A(w)(\alpha[x \mapsto a]) = \textsc{True} \text{ für ein } a \in s^A \\ \textsc{False} & \text{anderenfalls} \end{cases}$

Wir schreiben auch $A, \alpha \models w$ für $A(w)(\alpha) = \textsc{True}$. Falls $w \in \mathcal{S}en(\Sigma, X)$, dann hängt der Wert $A(w)(\alpha)$ nicht von α ab, und wir schreiben dann auch $A \models w$ und nennen A ein *Modell* von w. Wir verallgemeinern diese Schreibweise auf Mengen von Sätzen durch die Festlegung

$$A \models W \quad \Leftrightarrow \quad A \models w \quad \text{for all } w \in W$$

Falls C eine Klasse von Modellen von w ist, dann schreiben wir auch $C \models w$. Die *Theorie* einer Σ-Algebra A ist die Menge der Sätze, die in A gültig sind:

$$Th(A) = \{w \in \mathcal{S}en(\Sigma, X) \mid A \models w\}$$

Zwei Σ-Algebren A, B heißen *elementar äquivalent*, falls A und B mittels der Logik erster Stufe nicht unterscheidbar sind, d. h. wenn $Th(A) = Th(B)$. Insbesondere isomorphe Algebren sind stets elementar äquivalent. Für eine Klasse $\mathcal{C}$ von Σ-Algebren heißt eine Menge $T \subseteq \mathcal{S}en(\Sigma, X)$ *konsistent* in $\mathcal{C}$, falls es ein $A \in \mathcal{C}$ gibt mit $A \models T$. $w \in \mathcal{S}en(\Sigma, X)$ ist konsistent in $\mathcal{C}$, falls $\{w\}$ konsistent in $\mathcal{C}$ ist. T heißt *vollständig* in $\mathcal{C}$, falls für jedes $w \in \mathcal{S}en(\Sigma, X)$ höchstens eine der Mengen $T \cup \{w\}$ und $T \cup \{\neg w\}$ konsistent in $\mathcal{C}$ ist. Falls $R \subseteq T \subseteq \mathcal{S}en(\Sigma, X)$ und T vollständig in $\mathcal{C}$ ist, dann heißt T eine *Vervollständigung* von R in $\mathcal{C}$.

Die Substitution in Formeln ist folgendermaßen definiert:

1. $(r_1 = r_2)^{t_1 \ldots t_n}_{x_1 \ldots x_n} = ({r_1}^{t_1 \ldots t_n}_{x_1 \ldots x_n} = {r_2}^{t_1 \ldots t_n}_{x_1 \ldots x_n})$
2. $(w_1 \wedge w_2)^{t_1 \ldots t_n}_{x_1 \ldots x_n} = ({w_1}^{t_1 \ldots t_n}_{x_1 \ldots x_n} \wedge {w_2}^{t_1 \ldots t_n}_{x_1 \ldots x_n})$
3. $(\neg w)^{t_1 \ldots t_n}_{x_1 \ldots x_n} = \neg w^{t_1 \ldots t_n}_{x_1 \ldots x_n}$
4. $(\exists x : s \,.\, w)^{t_1 \ldots t_n}_{x_1 \ldots x_n} = \exists y : s \,.\, (w^y_x)^{t_1 \ldots t_n}_{x_1 \ldots x_n}$ wobei $y \notin \{x_1, \ldots, x_n\} \cup \bigcup_{i=1}^{n} \mathcal{V}ar\langle t_i \rangle$.

Auch hier gilt wieder ein Substitutionssatz ([LS87]):

$$A(w^{t_1 \ldots t_n}_{x_1 \ldots x_n})(\alpha) = A(w)(\alpha[x_1 \mapsto A(t_1)(\alpha), \ldots, x_n \mapsto A(t_n)(\alpha)]$$

In manchen Fällen ist eine andere Schreibweise für Substitutionen nützlich (siehe z.B. [CK90]): Falls $w \in \mathcal{W}ff(\Sigma, X)$ mit $\mathcal{V}ar\langle w \rangle \subseteq \{x_1, \ldots, x_n\}$, dann schreiben wir w auch als $w(x_1, \ldots, x_n)$. Diese Schreibweise dient dazu, eine Ordnung der freien Variablen von w festzulegen (die Menge der Variablen ist ansonsten ungeordnet). Sobald diese Ordnung fixiert ist, können wir die Schreibweise $w(t_1, \ldots, t_n)$ als Abkürzung für $w^{t_1 \ldots t_n}_{x_1 \ldots x_n}$ benutzen.

Wenn die Ordnung der freien Variablen von w in der Form $w(x_1, \ldots, x_n)$ festgelegt worden ist, benutzen wir manchmal auch eine andere Notation für die Semantik der Formel w. Falls A eine Σ-Algebra ist und $r_i \in s_i^A$ für $i = 1, \ldots, n$, wobei s_i jeweils die Sorte von x_i ist, und $\alpha \in {}^{?}X,A$, dann schreiben wir $A \models w[r_1, \ldots, r_n]$ als Abkürzung für

$$A, \alpha[x_1 \mapsto r_1, \ldots, x_n \mapsto r_n] \models w$$

Die Variablenbelegung α taucht in dieser abkürzenden Schreibweise nicht mehr auf. Dies wird durch den *Koinzidenzsatz* ([LS87]) gerechtfertigt, der besagt:

> Falls $\alpha(x) = \beta(x)$ für alle $x \in \mathcal{V}ar\langle w \rangle$, dann ist $A(w)(\alpha) = A(w)(\beta)$.

Ferner erlauben wir, wo es schreibtechnisch angebracht erscheint, für die obigen Abkürzungen auch eine Infix- oder auch Mixfixschreibweise wie z. B. $(x)w(y)$ statt $w(x, y)$ oder $[r_1]w[r_2]$statt $w[r_1, r_2]$.

Eine Formel $w \in \mathcal{W}ff(\Sigma, X)$ ist in *Prenexnormalform*, falls w von der Form

$$Q_1 x_1 : s_1 \ldots Q_n x_n : s_n \,.\, p$$

ist, wobei $Q_i \in \{\exists, \forall\}$ und $p \in \mathcal{W}ff(\Sigma, X)$ keine Quantoren enthält. Die *Anzahl der Quantoralternierungen* in w ist die Anzahl der i, $1 \leq i < n$ mit $Q_i \neq Q_{i+1}$. Falls q diese Anzahl ist und $n \geq 1$, dann ist

$$w \in \begin{cases} \Sigma_{q+1}\mathcal{W}ff(\Sigma_E, X) & \text{falls } Q_1 = \exists \\ \Pi_{q+1}\mathcal{W}ff(\Sigma_E, X) & \text{falls } Q_1 = \forall \end{cases}$$

$\Sigma_0\mathcal{W}ff(\Sigma, X) = \Pi_0\mathcal{W}ff(\Sigma, X)$ bezeichnet die Menge der der quantorenfreien Formeln. Falls $W \subseteq \mathcal{W}ff(\Sigma, X)$, dann heißt $\Sigma_n\mathcal{W}ff(\Sigma, X) \cap W$ das Σ_n-*Fragment* und $\Pi_n\mathcal{W}ff(\Sigma, X) \cap W$ das Π_n-*Fragment* von W.

Es gibt eine berechenbare Funktion $p: \mathcal{W}ff(\Sigma, X) \to \mathcal{W}ff(\Sigma, X)$, die jede Formel w auf eine äquivalente Formel $p(w)$ in Prenexnormalform abbildet ([Gal86]).

2.7 Modelltheoretische Sätze

Dieser Abschnitt enthält einige wichtige Definitionen und Sätze der Modelltheorie der Prädikatenlogik erster Stufe. Eine ausführliche Darstellung mit Beispielen und Beweisen findet man beispielsweise in [CK90].

in Abschnitt 2.2 haben wir den Begriff der Teilalgebra eingeführt, um Teilstrukturen in der Klasse der Σ-Algebren zu beschreiben. Von einer Beziehung "A ist eine Teilstruktur von B" verlangen wir dabei, daß sich bezüglich einer geeignete Menge von Aussagen, die in beiden Algebren sinnvoll sind, A und B gleich verhalten. Solange man nur an Berechnungen, d. h. Auswertungen von Termen, in Algebren interessiert ist, ist hierfür die Teilalgebrabeziehung $A \subseteq B$ adäquat, denn es gilt dann für alle Terme $t \in \mathcal{T}(\Sigma, X)$ und alle Belegungen $\alpha \in ?_{A,X}$:

$$A(t)(\alpha) = B(t)(\alpha)$$

Die Logik erster Stufe ermöglicht aber stärkere Aussagen über Algebren, bezüglich derer gleiches Verhalten von der Teilalgebrarelation nicht mehr garantiert werden kann, da z. B. $Th(A) = Th(B)$ *nicht* aus $A \subseteq B$ folgt. Der Begriff der elementaren Teilalgebra[5] ist eine Verschärfung des Teilalgebrabegriffes: Wir fordern jetzt zusätzlich , daß alle Elemente von A in A wie in B die gleichen "Eigenschaften erster Stufe" haben.

Es seien Σ eine Signatur und A, B Σ-Algebren. A heißt eine *elementare Teilalgebra* von B, i. Z. $A \preceq B$, falls

- $A \subseteq B$
- Für alle $w \in \mathcal{W}ff(\Sigma, X)$ und $\alpha \in ?_{X,A}$ ist $A(w)(\alpha) = B(w)(\alpha)$.

[5] Im allgemeineren Kontext von [CK90] ist von *elementaren Teilmodellen* (elementary submodels) die Rede.

Insbesondere ist also $Th(A) = Th(B)$, falls $A \preceq B$. Eine Klasse $\mathcal{C}$ von Σ-Algebren heißt *abgeschlossen unter elementaren Teilalgebren*, falls für alle $B \in \mathcal{C}$ und $A \preceq B$ auch $A \in \mathcal{C}$ gilt. Es gilt der verschärfte abwärts gerichtete Löwenheim-Skolem-Tarski Satz (strengthend downward Löwenheim-Skolem-Tarski theorem in [CK90], Theorem 3.1.6):

> Zu jeder abzählbaren Signatur Σ und Σ-Algebra B mit $\#(B) \geq \aleph_0$ gibt es eine elementare Teilalgebra $A \preceq B$ mit $\#(A) = \aleph_0$.

Der "klassische" abwärts gerichtete Löwenheim-Skolem-Tarski Satz (downward Löwenheim-Skolem-Tarski theorem in [CK90], Corollary 2.1.4) ist eine Folgerung hieraus:

> Zu jeder abzählbaren Signatur Σ und Σ-Algebra B mit $\#(B) \geq \aleph_0$ gibt es eine Σ-Algebra A mit $\#(A) = \aleph_0$ und $Th(A) = Th(B)$.

Ein anderer wichtiger Satz der Modelltheorie ist der Kompaktheitssatz:

> Eine Menge von Sätzen $W \subseteq \mathcal{S}en(\Sigma, X)$ hat ein Modell genau dann wenn jede endliche Teilmenge von W ein Modell hat.

Wir zeigen nun ein einfaches Korollar aus dem Kompaktheitssatz, der Beweis dient uns als Musterbeispiel für die Anwendungen des Kompaktheitssatzes (vergleiche Corollary 2.1.5 in [CK90]):

> Wenn eine Menge $W \subseteq \mathcal{S}en(\Sigma, X)$ von Sätzen Modelle beliebiger endlicher Kardinalität hat, dann hat W auch ein unendliches Modell.

Beweis: W habe Modelle, bei denen die Trägermengen der Sorte s beliebige endliche Größe annehmen. Betrachte die Signatur $\Sigma' = \Sigma \cup \{c_i : \rightarrow s \mid i \in \mathbb{N}\}$ und

$$W' = W \cup \{c_i \neq c_j \mid i \neq j\} \subseteq \mathcal{S}en(\Sigma', X)$$

Nach Voraussetzung hat jede endliche Teilmenge von W' ein Σ'-Modell, aus dem Kompaktheitssatz folgt, daß auch ganz W' ein Σ'-Modell A hat. Damit ist $A|_\Sigma$ ein unendliches Σ-Modell von W. □

Ein wichtiges Beispiel verschiedener Modelle einer Theorie liefert die Zahlentheorie. Die Signatur Σ_N besteht aus der Sorte *nat* sowie den Funktionszeichen 0, 1, + und $*$ der üblichen Stelligkeit. Eine mögliche Σ_N-Algebra ist die Algebra N der natürlichen Zahlen, wir nennen diese Algebra das *Primmodell der Arithmetik*. Ihre Theorie $Th(N)$ bildet die *vollständige Zahlentheorie*. Es gibt nun aber, wie man mit dem Kompaktheitssatz zeigen kann, Modelle der vollständigen Zahlentheorie, in denen nicht jeder Träger in der Form $1 + \cdots + 1$ darstellbar ist. Wir nennen diese Modelle der vollständigen Zahlentheorie *Nichtprimmodelle der Arithmetik*. Wir haben uns hier für das Begriffspaar Prim/Nichtprimmodell im Gegensatz zum üblichen Standard/Nonstandardmodell entschieden, um eine Verwechslung mit dem später in dieser Arbeit definierten Begriff der Standardisierung einer Algebra zu vermeiden.

Regelsystem WE:

$(\wedge \perp \text{com})$	$(w_1 \wedge w_2)$	$\perp\!\!\!\rightarrow$	$(w_2 \wedge w_1)$
$(\wedge \perp \text{ass})$	$(w_1 \wedge (w_2 \wedge w_3))$	$\perp\!\!\!\rightarrow$	$((w_1 \wedge w_2) \wedge w_3)$
$(\vee \perp \text{com})$	$(w_1 \vee w_2)$	$\perp\!\!\!\rightarrow$	$(w_2 \vee w_1)$
$(\vee \perp \text{ass})$	$(w_1 \vee (w_2 \vee w_3))$	$\perp\!\!\!\rightarrow$	$((w_1 \vee w_2) \vee w_3)$
$(\forall \perp \text{com})$	$(\forall x_1 : s_1 \,.\, \forall x_2 : s_2 \,.\, w)$	$\perp\!\!\!\rightarrow$	$(\forall x_2 : s_2 \,.\, \forall x_1 : s_1 \,.\, w)$
$(\exists \perp \text{com})$	$(\exists x_1 : s_1 \,.\, \exists x_2 : s_2 \,.\, w)$	$\perp\!\!\!\rightarrow$	$(\exists x_2 : s_2 \,.\, \exists x_1 : s_1 \,.\, w)$

Abbildung 4: Das Regelsystem zur Relation WE

N ist ein elementares Teilmodell jedes Modelles der vollständigen Zahlentheorie, unsere Namesgebung ist daher konsistent mit der in der Modelltheorie üblichen Verwendung des Begriffes *Primmodell* (prime model, [CK90]).

2.8 Kongruenzen und Ersetzungen auf Formeln

Eine binäre Relation $\otimes$ auf $\mathcal{W}ff(\Sigma, X)$ heißt *abgeschlossen unter Kontextanwendung*, falls gilt:

1. Falls $w \otimes v$, dann $(\neg w) \otimes (\neg v)$
2. Falls $w_1 \otimes v_1$ und $w_2 \otimes v_2$, dann $(w_1 \wedge w_2) \otimes (v_1 \wedge v_2)$
3. Falls $w \otimes v$, dann $(\exists x : s \,.\, w) \otimes (\exists x : s \,.\, v)$

Falls R eine binäre Relation auf $\mathcal{W}ff(\Sigma, X)$ ist, dann bezeichnet $\rightarrow_R$ ihren *Abschluß unter Kontextanwendung*, also die kleinste unter Kontextanwendung abgeschlossene Relation die R enthält.

Eine solche Relation R geben wir oft durch ein Formelschema an. Falls $R_1, \ldots, R_n$ Relationen auf $\mathcal{W}ff(\Sigma, X)$ sind, dann definieren wir

$$\rightarrow_{R_1,\ldots,R_n} \quad := \quad \rightarrow_{R_1} \cup \ldots \cup \rightarrow_{R_n}$$

Wir werden solche Mengen von Relationen dann oft wieder unter einem symbolischen Namen zusammenfassen. So ist durch Abbildung 4 beispielsweise die Relation

$$\rightarrow_{WE} \quad = \quad \rightarrow_{\forall\text{-com}} \cup \ldots \cup \rightarrow_{\exists\text{-com}}$$

definiert. Die Hintereinanderausführung solcher Relationen definieren wir durch

$$x \rightarrow_A \circ \rightarrow_B y \quad :\Leftrightarrow \quad \text{ex. } y \;\; \text{mit } x \rightarrow_A y \rightarrow_B z$$

Somit gilt $w_1 \leftrightarrow^*_{WE} w_2$, falls w_1 und w_2 gleich sind modulo der Assoziativität und Kommutativität von $\wedge$, $\vee$ und der Vertauschung gleichartiger Quantoren. Wir nennen in diesem

Fall w_1 und w_2 *kongruent*, es gilt dann offenbar $A(w_1)(\alpha) = A(w_2)(\alpha)$ für alle A, α. Aus diesem Grunde identifizieren wir meist kongruente Formeln und unterdrücken Klammerpaare, die nur zur Unterscheidung kongruenter Formeln dienen. Weiterhin vereinbaren wir folgende Prioritäten:

Formeloperator	Priorität
$\neg$	5
$\wedge$	4
$\vee$	3
$\supset, \not\subset$	2
$\exists x : s, \forall x : s$	1

2.9 Semantik rekursiver Programme

Die Definition der denotationellen sowie der algebraischen Semantik rekursiver Programe basiert auf vollständigen partiellen Ordnungen. Sie bilden die semantischen Bereiche für die verschiedenen syntaktischen Einheiten der Programmiersprache. Da unsere Sprache keine höheren Funktionen enthält, benötigen wir in dieser Arbeit nur einen wichtigen Spezialfall: die flachen vollständigen partiellen Ordnungen. Eine allgemeinere Einführung in vollständige partielle Ordnungen findet man in [LS87]. Die hier betrachtete Klasse der flachen vollständigen partiellen Ordnungen ist identisch mit der Klasse der diskreten Interpretationen in [CG78].

Sei D eine Menge und $\sqsubseteq$ eine partielle Ordnung auf D. $(D, \sqsubseteq)$ heißt *flache vollständige partielle Ordnung*, abgekürzt fcpo von "flat complete partial order", falls es ein $\bot^D \in D$ gibt, so daß für alle $d_1, d_2 \in D$ gilt:

$$d_1 \sqsubseteq c_2 \quad \Leftrightarrow \quad (d_1 = d_2 \vee d_1 = \bot^D)$$

$\bot^D$ heißt dann auch das *kleinste Element* von $(D, \sqsubseteq)$. Im Zusammenhang mit der Semantik von Programmen nennen wir $\bot^D$ auch das "undefinierte Element" von D, es repräsentiert dann die Tatsache, daß (noch) keine Information über den Wert einer Berechnung vorliegt.

Seien nun $(D_1, \sqsubseteq_1), \ldots, (D_{n+1}, \sqsubseteq_{n+1})$ fcpo's und f eine Funktion $f\colon D_1, \ldots, D_n \to D_{n+1}$. f heißt *monoton*, falls für alle $x_i, y_i \in D_i$, $i = 1, \ldots, n$ gilt:

$$x_1 \sqsubseteq_1 y_1 \wedge \ldots \wedge x_n \sqsubseteq_n y_n \supset f(x_1, \ldots, x_n) \sqsubseteq_{n+1} f(y_1, \ldots, y_n)$$

f heißt *strikt*, falls für alle $x_i \in D_i$ gilt:

$$x_1 = \bot^{D_1} \vee \ldots \vee x_n = \bot^{D_n} \supset f(x_1, \ldots, x_n) = \bot^D$$

Jede strikte Funktion ist natürlich monoton.

Eine *Kette* in einer fcpo $(D, \sqsubseteq)$ ist eine Folge $(d_i)_{i \in \mathbb{N}}$ mit $d_i \sqsubseteq d_{i+1}$ für alle i. Ihre *kleinste obere Schranke* $\bigsqcup_{i \geq 0} d_i$ ist eindeutig definiert durch die Eigenschaft

$$\text{für alle } y\colon \left(\bigsqcup_{i \geq 0} d_i \leq y \quad \Leftrightarrow \quad \text{für alle } i\colon d_i \sqsubseteq y \right)$$

2.10 Berechenbarkeit

Wir setzen die elementaren Begriffe der Berechenbarkeitstheorie ([Rog87]) wie z.B. berechenbare Funktionen, Entscheidbarkeit und Semientscheidbarkeit von Mengen voraus. Eine Menge ist *kosemientscheidbar*, falls ihr Komplement seminentscheidbar ist.

Unser Berechnungsbegriff ist dabei nicht auf Berechnungen über natürlichen Zahlen beschränkt. Berechnungen können in beliebigen Mengen, die symbolisch darstellbar sind (also Teilmenge eines S^* für einen geeigneten endlichen Zeichenvorrat S) und die effektiv durch natürliche Zahlen kodierbar sind (siehe [Rog87]), stattfinden.

3 Module

In diesem Kapitel definieren wir Syntax und Semantik unserer idealisierten Programmiersprache.

In Abschnitt 3.1 werden wir zunächst zwei technische Konzepte, die wir zur Definition von Syntax und Semantik der Module benötigen werden, vorstellen: Standardsignaturen und Standardalgebren. Eine Standardsignatur ist eine Signatur, die gewisse Sorten- und Funktionssymbole enthält, die wir in jeder betrachteten Signatur voraussetzen möchten. Eine Standardalgebra weist diesen Symbolen dann eine festgelegte Semantik zu. Die Definitionen von Standardsignatur und -algebra legen einen "Kern" der betrachteten Algebren fest und sind somit sowohl für die Mächtigkeit unserer Modulsprache als auch für die Aussagekraft der zugehörigen Logik von Bedeutung.

Wir werden dann in Abschnitt 3.2 die Syntax unserer Modulsprache definieren. Wir erläutern hier zunächst informal die einzelnen Teile eines Modules und ihren Beitrag zur Semantik. Abbildung 5 auf Seite 39 enthält ein konkretes Beispiel einer Moduldefinition.

Ein Modul besteht aus

- einer Parametersignatur, diese muß eine Standardsignatur sein. Die Sorten- und Funktionsdefinitionen des Moduls bauen auf der Parametersignatur auf. Die Klasse der Standardalgebren über der Parametersignatur bildet den Definitionsbereich der Semantik eines Modules.
- der Definition der neuen Sorten und Funktionen des Modules. Diese teilt sich auf in:
  - die Angabe der neuen Sorten
  - eine Liste von Konstruktorsymbolen für die neuen Sorten. Die Stelligkeiten der Konstruktorsymbole enthalten im allgemeinen selbst die neuen Sorten (anderenfalls ist die Sortendefinition trivial) und auch Sorten der Parametersignatur. Die Konstruktoren definieren, für eine gegebene Parameteralgebra, die Trägermengen der neuen Sorten: diese bestehen aus den Termen, die man unter Beachtung der Stelligkeiten mit den Konstruktoren aus den Elementen der Parameteralgebra konstruieren kann. Diese Konstruktion hat ein Vorbild in der Sprache LISP ([Ste90]). Dort wird auf ähnliche Weise eine "Sorte" *list* mit Hilfe der Konstruktoren *nil* und *cons* definiert[6]. Die Angabe von Konstruktoren impliziert die Definition von Selektorfunktionen und Testfunktionen. Selektorfunktionen extrahieren Teilterme aus einem Term (wie *car* und *cdr* in der LISP-Analogie), und Testfunktionen überprüfen, ob ein Term ein bestimmtes Konstruktorsymbol an der Wurzel hat (vgl. *consp* in LISP).

    Außerdem ergänzen wir jede neu definierte Trägermenge um ein undefiniertes Element, um die Nichtterminierung von Funktionen zu modellieren.
  - eine Liste von neuen Funktionszeichen, deren Stelligkeit beliebig aus Sorten der Parametersignatur und neuen Sorten zusammengesetzt sein darf.

[6] Die Analogie ist sehr grob, da LISP kein Typenkonzept hat

  - ein rekursives Programm für die neuen Funktionszeichen. Hier erlauben wir beliebige Rekursion (nicht nur primitive Rekursion, wie in dem Beispiel) und auch simultane Rekursion der Funktionen.

- eine Liste von exportierten Funktionen, die nicht exportierten Funktionen haben dann den Status von Hilfsfuntionen und sind nach außen nicht sichtbar. Die Klasse der Standardalgebren über sämtlichen Sorten des Moduls und den exportierten Funktionen bildet den Wertebereich der Semantik des Moduls.

Eine formale Beschreibung der Semantik werden wir in Abschnitt 3.3 angeben. In Abschnitt 3.4 schließlich untersuchen wir einige Eigenschaften der Semantik, die wir bei den Betrachtungen der Logik in den darauffolgenden Kapitel benötigen werden.

Wir werden in diesem und in den folgenden Kapiteln die Wahl einer geeigneten Menge von Variablen nicht mehr erwähnen. Es bezeichne daher stets $X = (X_s)_{s \in S}$ eine "passende" Variablenfamilie.

3.1 Standardsignaturen und Standardalgebren

Da wir in den rekursiven Programmen bedingte Ausdrücke verwenden möchten, gehört zu einer Standardsignatur insbesondere eine Sorte *bool* mit entsprechenden Funktionszeichen. Von den Standardalgebren forden wir, daß alle Trägermengen fcpo's sind und die Funktionen monoton bezüglich der Ordnungen auf den Trägermengen. Dies ist notwendig, um die Semantik der rekursiven Programme mit einem algebraischen (oder denotationellen) Ansatz definieren zu können.

Wir haben uns hier dafür entschieden, in der Modulsprache und in der Logik den gleichen "Kern" an Sorten- und Funktionssymbolen zu benutzen. Dies vereinheitlicht die Notationen, führt aber andererseits zu einigen Redundanzen. So ist z. B. die Verfügbarkeit eines Zeichens $\perp$ für ein nicht definiertes Datenobjekt für unsere Logik essentiell, es beeinflußt aber nicht die Mächtigkeit der Modulsprache, da eine nichtterminierende Berechnung stets durch den Aufruf einer Funktion $loop(x) \Leftarrow loop(x)$ erreicht werden kann. Aus diesem Grunde wurde in [Loe87] das Symbol $\perp$ in der Modulsprache nicht erlaubt. Auf der anderen Seite sind beispielsweise die Funktionen `ifthenelse` notwendig zur Formulierung bedingter Ausdrücke in den Funktionsdefinitionen, in der Logik können diese Ausdrücke aber durch Transformation der Formeln eliminiert werden (siehe Lemma 26).

Wir führen zunächst die Begriffe Semi-Standardsignatur und Semi-Standardalgebra ein. Dabei handelt es sich um Signaturen, bzw. Algebren, die man zu Standardsignaturen, bzw. Standardalgebren erweitern kann. Diese Begriffe sind in erster Linie technischer Natur.

Definition 1 *Eine Signatur $\Sigma = (S, F)$ heißt* Semi-Standardsignatur, *falls für alle $s \in S$:*

falls $\perp_s \in F$	*dann mit Stelligkeit* $\rightarrow s$
falls $=_s \in F$	*dann mit Stelligkeit* $s, s \rightarrow bool$ *und* $bool \in S$
falls $\texttt{ifthenelse}_s \in F$	*dann mit Stelligkeit* $bool, s, s \rightarrow s$ *und* $bool \in S$

und entweder

- $bool \notin S$ *und* $\{\bot_s, \texttt{ifthenelse}_s, =_s \mid s \in S\} \cap F = \emptyset$ *oder*
- $(\{bool\}, \{true: \rightarrow bool, false: \rightarrow bool\}) \subseteq \Sigma$ *und falls* $\bot_s \in F$ *für ein* $s \in S$ *dann* $\bot_{bool} \in F$

Definition 2 *Sei* $\Sigma = (S, F)$ *eine Semi-Standardsignatur. Die* Standardisierung *von* Σ *ist die Signatur* (S', F') *mit*

$$\begin{array}{rcl} S' & := & S \cup \{bool\} \\ F' & := & F \cup \{true: \rightarrow bool, false: \rightarrow bool\} \\ & & \cup \{\bot_s \mid s \in S \cup \{bool\}\} \\ & & \cup \{=_s \mid s \in S \cup \{bool\}\} \\ & & \cup \{\texttt{ifthenelse}_s \mid s \in S \cup \{bool\}\} \end{array}$$

Definition 3 *Eine Signatur heißt* Standardsignatur, *falls sie eine Semi-Standardsignatur und identisch mit ihrer eigenen Standardisierung ist.*

Eine Standardsignatur enthält also die in Definition 2 aufgeführten Symbole.

Definition 4 *Sei* A *eine* (S, F)*-Algebra. Jeder Sorte* $s \in S$ *ordnen wir eine partielle Ordnung* $\sqsubseteq_s^A$ *auf* s^A *zu durch:*

$$x \sqsubseteq_s^A y :\Longleftrightarrow \begin{cases} x = y & \text{falls } \bot_s \notin F \\ x = y \vee x = \bot_s^A & \text{falls } \bot_s \in F \end{cases}$$

Definition 5 *Sei* $\Sigma = (S, F)$ *eine Semi-Standardsignatur und* A *eine* Σ*-Algebra.* A *ist eine* Semi-Standardalgebra, *falls*

- *Für alle* $f: s_1, \ldots, s_n \rightarrow s \in F$, $x_1, y_1 \in s_1^A, \ldots, x_n, y_n \in s_n^A$*:*

$$f^A(x_1, \ldots, x_n) \sqsubseteq_s^A f^A(y_1, \ldots, y_n) \text{ falls } x_1 \sqsubseteq_{s_1}^A y_1 \text{ und } \ldots \text{ und } x_n \sqsubseteq_{s_n}^A y_n$$

- *Falls* $bool \in S$, *dann*

$$\begin{array}{rcll} bool^A & = & \begin{cases} \{\underline{\text{true}}, \underline{\text{false}}\} & \text{falls } \bot_{bool} \notin F \\ \{\underline{\text{true}}, \underline{\text{false}}, \bot_{bool}\} & \text{falls } \bot_{bool} \in F \end{cases} \\ true^A & = & \underline{\text{true}} & \\ false^A & = & \underline{\text{false}} & \\ \bot_{bool}^A & = & \bot_{bool} & \text{falls } \bot_{bool} \in F \end{array}$$

- *Für alle* $s \in S$ *mit* $\texttt{ifthenelse}_s \in F$ *und alle* $x_1, x_2 \in s^A$*:*

$$\begin{aligned} \texttt{ifthenelse}_s^A(\underline{\text{true}}, x_1, x_2) &= x_1 \\ \texttt{ifthenelse}_s^A(\underline{\text{false}}, x_1, x_2) &= x_2 \end{aligned}$$

  und, falls $\bot_{bool} \in F$*:*

$$\texttt{ifthenelse}_s^A(\bot_{bool}, x_1, x_2) = \bot_s^A$$

- *Für alle* $s \in S$ *mit* $\bot_s \notin F$ *und alle* $x_1, x_2 \in s^A$*:*

$$x_1 =_s^A x_2 = \begin{cases} \underline{\text{true}} & \textit{falls } x_1 = x_2 \\ \underline{\text{false}} & \textit{anderenfalls} \end{cases}$$

- *Für alle* $s \in S$ *mit* $\bot_s \in F$ *und alle* $x_1, x_2 \in s^A$*:*

$$x_1 =_s^A x_2 = \begin{cases} \underline{\text{true}} & \textit{falls } x_1 = x_2 \textit{ und } x_1 \neq \bot_s^A \neq x_2 \\ \underline{\text{false}} & \textit{falls } x_1 \neq x_2 \textit{ und } x_1 \neq \bot_s^A \neq x_2 \\ \bot_{bool} & \textit{falls } x_1 = \bot_s^A \textit{ oder } x_2 = \bot_s^A \end{cases}$$

Definition 6 *Sei* $\Sigma = (S, F)$ *eine Semi-Standardalgebra und* Σ' *ihre Standardisierung. Die* Standardisierung *einer* Σ *Semi-Standardalgebra* A *ist die* Σ'*-Algebra* B*, definiert durch*

- *Für alle* $s \in S$ *mit* $\bot_s \in \Sigma$ *ist* $s^B = s^A$*. Falls* $\bot_s \notin \Sigma$*, dann ist* $s^B = s^A \cup \{\bot_s\}$ *wobei wir o.B.d.A. annehmen, daß* $\bot_s \notin s^A$*. Definiere* $\bot_s^B = \bot_s$*.*
- $bool^B = \{\underline{\text{true}}, \underline{\text{false}}, \bot_{bool}\}$ *und* $true^B = \underline{\text{true}}$*,* $false^B = \underline{\text{false}}$*,* $\bot_{bool}^B = \bot_{bool}$
- *Für alle* $s \in S$ *und* $x_1, x_2 \in s^B$*:*

$$\begin{aligned} \texttt{ifthenelse}_s^B(\underline{\text{true}}, x_1, x_2) &= x_1 \\ \texttt{ifthenelse}_s^B(\underline{\text{false}}, x_1, x_2) &= x_2 \\ \texttt{ifthenelse}_s^B(\bot_{bool}, x_1, x_2) &= \bot_s^B \end{aligned}$$

- *Für alle* $s \in S$ *und alle* $x_1, x_2 \in s^B$*:*

$$x_1 =_s^B x_2 = \begin{cases} \underline{\text{true}} & \textit{if } x_1 = x_2 \textit{ und } x_1 \neq \bot_s^B \neq x_2 \\ \underline{\text{false}} & \textit{if } x_1 \neq x_2 \textit{ und } x_1 \neq \bot_s^B \neq x_2 \\ \bot_{bool} & \textit{if } x_1 = \bot_s^B \textit{ oder } x_2 = \bot_s^B \end{cases}$$

- *Für alle Funktionssymbole* $f\colon s_1, \ldots, s_n \to s \in F$ *und alle* $x_i \in s_i^B$*:*

$$f^B(x_1, \ldots, x_n) = \begin{cases} f^A(x_1, \ldots, x_n) & \textit{falls } x_i \in s_i^A \textit{ für alle } i \\ \bot_s^B & \textit{anderenfalls} \end{cases}$$

Definition 7 *Eine Algebra über einer Standardsignatur ist eine* Standardalgebra, *wenn sie eine Semi-Standardalgebra und identisch mit ihrer eigenen Standardisierung ist. Alg_Σ bezeichnet die Klasse aller Σ-Standardalgebren.*

Damit ist also insbesondere jede Trägermenge einer Standardalgebra eine fcpo, und alle Funktionen sind monoton.

Lemma 1 *Sei B die Standardisierung der Σ-Semi-Standardalgebra A. dann ist*

$$A \subseteq B\,|_\Sigma$$

A ist eine echte Teilalgebra von$B\,|_\Sigma$, falls B zusätzliche undefinierte Träger enthält.

Definition 8 *Für eine Variablenfamilie $X = (X_s)_{s\in S}$ und (S,F)-Standardalgebra A bezeichnet $?^+_{X,A}$ die Menge der positiven A-Variablenbelegungen:*

$$?^+_{X,A} = \{\bigcup_{s\in S} \gamma_s \mid \gamma_s \in (X_s \to s^A \setminus \{\bot^A_s\})\}$$

Das folgende Lemma drückt aus, daß die Theorie einer Algebra die Theorie ihrer Standardisierung eindeutig bestimmt. Den einfachen Beweis führen wir hier nicht.

Lemma 2 *Es seien Σ eine Semistandardsignatur und A, B Σ-Semistandardalgebren. Σ' sei die Standardisierung von Σ und A', bzw. B' die Standardisierung von A, bzw. B. Falls $Th(A) = Th(B)$, dann ist auch $Th(A') = Th(B')$.*

Lemma 3 *Sei Σ eine Semistandardsignatur und Σ' die Standardisierung von Σ. Dann gibt es zu jedem Satz $w' \in \mathcal{S}en(\Sigma', X)$ ein $w \in \mathcal{S}en(\Sigma, X)$, so daß für alle strikten Σ'-Standardalgebren gilt: $A \models w \leftrightarrows w'$*

Wir benötigen nun noch eine Definition einer sehr einfachen Standardsignatur und ihrer einzigen Standardalgebra:

Definition 9 *Σ_{BOOL} ist die Standardisierung der leeren Signatur.*
$BOOL$ bezeichnet die einzige Standardalgebra der Signatur Σ_{BOOL}.

Σ_{BOOL} besteht also nur aus der Sorte *bool* sowie aus den Konstanten $true$, $false$, $\bot_{bool}$ der Sorte *bool* und den beiden Funktionen $=_{bool}$ und $\texttt{ifthenelse}_{bool}$. Die Algebra $BOOL$ enthält den "boolschen Kern" einer jeden Standardalgebra.

3.2 Syntax

Eine informale Übersicht der Syntax unserer Modulsprache haben wir bereits auf Seite 33 gegeben. Wir definieren nun die Syntax formal, ein Beispiel in einer lesefreundlichen Form enthält Abbildung 5.

Definition 10 *Ein* Modul *ist ein Tupel*

$$(PS_m, PF_m, NS_m, K_m, NF_m, EF_m, PR_m)$$

wobei PS_m, PF_m, NS_m, K_m und NF_m paarweise disjunkte Mengen sind und

- *(PS_m, PF_m) ist eine Standardsignatur, wir bezeichnen diese auch mit Σ_{Pm}. Die Elemente von PS_m heißen die* Parametersorten*, die Elemente von PF_m die* Parameterfunktionen *von m.*
- *$(PS_m \cup NS_m, PF_m \cup K_m \cup NF_m)$ ist eine Signatur und der Wertebereich aller Funktionssymbole aus K_m liegt in NS_m. Die Standardisierung dieser Signatur, erweitert um*

  $$\{\mathtt{is}_k\mathtt{?}\colon s \to \mathit{bool} \mid s \in NS_m\} \cup \{\mathtt{select}_k^j\colon s \to s_j \mid c \in K_m\ ,\ c\colon s_1, \ldots, s_j, \ldots, s_n \to s\}$$

  bildet dann die Signatur $\Sigma_{Am} = (AS_m, AF_m)$
- *PR_m ist ein rekursives Programm der Form*

  $$\begin{pmatrix} f_1(x_{1,1}, \ldots, x_{1,l_1}) & \Leftarrow & t_1 \\ \vdots & & \vdots \\ f_n(x_{n,1}, \ldots, x_{n,l_n}) & \Leftarrow & t_n \end{pmatrix}$$

  wobei $NF_m = \{f_1, \ldots, f_n\}$ und für $f_i\colon s_1, \ldots, s_n \to s \in NF_m$
  - *$x_{i,j} \in X_{s_j}$ und $x_{i,j} \neq x_{i,k}$ für $i \neq k$ und*
  - *$t_i \in \mathcal{T}(\Sigma_{Am}, \{x_{i,1}, \ldots, x_{i,l_i}\})_s$*
- *$EF_m \subseteq AF_m$ und $\Sigma_{Em} := (AS_m, EF_m)$ ist eine Standardsignatur.*

Wir nennen $(\Sigma_{Pm}, \Sigma_{Em})$ auch die Signatur *des Moduls m.*
Darüber hinaus bezeichnen wir mit $K_{m,s}$ für $s \in NS_m$ die Menge der Konstruktoren mit Zielsorte s.

Abbildung 5 enthält ein Beispiel einer Moduldefinition in einer besser lesbaren Syntax. Wir werden einzelne Module stets in dieser Form angeben, in formalen Definitionen aber auf Definition 10 zurückgreifen. Für die Umsetzung der formalen Syntax von Definition 10 in die saloppe Schreibweise gemäß Abbildung 5 benutzen wir folgende Konventionen:

PAR **SORTS** $elem$
OPNS $0{:} \rightarrow elem$
$+{:} elem, elem \rightarrow elem$

BODY **SORTS** $list$
CONS $nil{:} \rightarrow list$
$cons{:} elem, list \rightarrow list$
FCTS $app{:} list, list \rightarrow list$
$sum{:} list \rightarrow elem$
PROG $app(l_1, l_2) \Leftarrow \texttt{if is}_{nil}\texttt{?}(l_1)\ \texttt{then}\ l_2$
$\texttt{else}$
$cons(\texttt{select}^1_{cons}(l_1), app(\texttt{select}^2_{cons}(l_1), l_2))$
$sum(l) \Leftarrow \texttt{if is}_{nil}\texttt{?}(l)\ \texttt{then}\ 0$
$\texttt{else select}^1_{cons}(l) + sum(\texttt{select}^2_{cons}(l))$

Abbildung 5: Das Modul $list1$.

- In Σ_P und EF erwähnen wir die Sorte *bool* sowie die in der Definition von "Standardalgebra" verlangten Funktionszeichen nicht.
- EF wird, falls identisch mit AF, nicht erwähnt.
- Die Sortenindizes der Funktionszeichen, wie z. B. bei $=_s$, entfallen.
- Wir benutzen, wo üblich, eine Infix oder Mixfix Schreibweise für Terme.

Auf das Beispiel in Abbildung 5 angewandt, bedeutet dies

$$
\begin{aligned}
PS &= \{bool, elem\} \\
PF &= \{true, false, \bot_{bool}, ifthenelse_{bool}, =_{bool}, 0, +, \bot_{elem}, ifthenelse_{elem}, =_{elem}\} \\
NS &= \{list\} \\
K &= \{nil, cons\} \\
NF &= \{app, sum\} \\
AS &= \{bool, elem, list\} \\
AF &= \{true, false, \bot_{bool}, ifthenelse_{bool}, =_{bool}, 0, +, \bot_{elem}, ifthenelse_{elem}, =_{elem}, \\
&\quad nil, cons, \bot_{list}, ifthenelse_{list}, =_{list}, app, sum, \texttt{is}_{list}\texttt{?}, \texttt{select}^1_{list}, \texttt{select}^2_{list}\} \\
EF &= AF
\end{aligned}
$$

Die hier vorgestellte Syntax stellt eine vereinfachte Version der in [Loe87] vorgestellten Algorithmischen Spezifikationssprache dar. Von der in [Loe87] (siehe auch [LWF+91]) angegebenen

Syntax sind wir in den Teilen abgewichen, die einerseits für unsere Betrachtung der Logik nicht wesentlich sind, andererseits aber den Formalismus belasten. Die in Definition 10 angegebene Syntax enthält insbesondere *nicht*:

- überladene Funktionen. Die in [LWF+91] erlaubte Überladung von Funktionen kann leicht statisch aufgelöst werden.
- case-Ausdrücke. Diese wurden in [Loe87] benutzt, um die neuen Funktionen mittels einer eingeschränkten Form von pattern-matching zu definieren. Diese Terme führen aber semantisch zu einigen Komplikationen, da sie gebundene Vorkommen von Variablen einführen (siehe auch [Tre86]). In [Loe87] wurde gezeigt, wie case-Ausdrücke durch Terme mit Test- und Selektorfunktionen ersetzt werden können.
- Subset- und Quotientenformeln, bzw. -operationen. Diese sind zwar für die Ausdruckskraft der Modulsprache als Spezifikationsmethode wichtig, nicht aber für die Logik (Theorem 3 von [Loe87]).
- Einschränkungen der exportierten Sorten. Unsere Syntax erlaubt nur die Einschränkung der exportierten Funktionen. Da unsere Logik über Sorten, zu denen keine Funktionszeichen zur Verfügung stehen, nur sehr einfache Aussagen treffen kann (nämlich die der "pure idendity language", siehe [CK90]), ist eine Einschränkung der exportierten Sorten nicht von Bedeutung.

Wir beschließen den Abschnitt über die Syntax der Module mit zwei Modulkonstruktionen, die wir später noch benötigen werden.

Definition 11 *Sei m ein Modul und C eine Menge von Konstantensymbolen, die disjunkt ist zu allen Komponenten von m. Dann ist die* Erweiterung von m um C *das Modul*

$$(PS_m, PF_m \cup C, NS_m, K_m, NF_m, EF_m \cup C, PR_m)$$

Definition 12 *Für jede Standardsignatur $\Sigma = (S, F)$ definieren wir das* triviale Modul

$$1_\Sigma = (S, F, \emptyset, \emptyset, \emptyset, F, \emptyset)$$

Ein triviales Modul exportiert also genau seine Parametersorten und -funktionen, es definiert keine neuen Sorten oder Funktionen.

3.3 Semantik

Die Semantik eines Moduls m mit Signatur (Σ_P, Σ_E) ist ein Funktor

$$[\![m]\!] : Alg_{\Sigma_P} \to Alg_{\Sigma_E}$$

Um $[\![\cdot]\!]$ zu definieren, geben wir für ein beliebiges Modul m und eine Parameteralgebra A die Konstruktion von $[\![m]\!](A)$ an. Wir gehen dabei in mehreren Schritten vor und definieren zunächst drei "Zwischenalgebren" $A^{[1]}$, $A^{[2]}$ und $A^{[3]}$. Diese sind nur für die Konstruktion von $[\![m]\!](A)$ von Bedeutung, und wir werden nach der Definition von $[\![m]\!](A)$ keinen Bezug mehr auf sie nehmen.

Die Algebra $A^{[1]}$ erweitert die Parameteralgebra A um die Trägermengen der neuen Sorten (die undefinierten Träger $\perp_s$ nehmen wir allerdings erst im nächsten Schritt hinzu:)

Definition 13 *Es sei m ein Modul mit Signatur (Σ_P, Σ_E) und $A \in Alg_{\Sigma_P}$. Für die Definition der Trägermengen der neuen Sorten benötigen wir die folgende Menge von Konstantensymbolen:*

$$C_A := \{c_a : \rightarrow s \mid s \in PS_m, a \in s^A, a \neq \perp_s^A\}$$

Wir definieren eine Algebra $A^{[1]}$ mit Signatur $(PS_m \cup NS_m, PF_m)$ wie folgt:

- $s^{A^{[1]}} = s^A$ *falls* $s \in PS_m$
- $s^{A^{[1]}} = \mathcal{T}((PS_m \cup NS_m, K_m \cup C_A), \emptyset)_s$ *falls* $s \in NS_m$
- $f^{A^{[1]}} = f^A$ *für alle* $f \in PF_m$

Die Trägermengen der neuen Sorten sind somit als eine formale Sprache definiert. Zu diesem Zweck mußten wir die Algebra A in eine entsprechende Menge C_A von Konstantenzeichen umwandeln. Wir werden aber im folgenden nicht zwischen den Trägern von A und den zugehörigen Konstantenzeichen unterscheiden.

Betrachte beispielsweise das Modul $list1$ aus Abbildung 5, Seite 39. N bezeichne die Standardisierung des Primmodells der Arithmetik. Dann ist, bezüglich $list1$, die neue Trägermenge der Sorte $list$ in $N^{[1]}$:

$$\begin{aligned} list^{N^{[1]}} = \{&nil, cons(0, nil), cons(1, nil), \ldots, \\ &cons(0, cons(0, nil)), cons(1, cons(0, nil)), \ldots, \\ &cons(0, cons(1, nil)), cons(1, cons(1, nil)), \ldots\} \end{aligned}$$

Definition 14 *Seien A und m wie in Definition 13. Die Algebra $A^{[2]}$ ist die Standardisierung von $A^{[1]}$.*

Man beachte, daß durch die Standardisierung die Algebra $A^{[1]}$ nicht nur um neue Funktionen erweitert wird, sondern daß auch die Trägermengen der neuen Sorten um die undefinierten Träger ergänzt werden. Im oben begonnenen Beispiel heißt das:

$$list^{N^{[2]}} = list^{N^{[1]}} \cup \{\perp_{list}\}$$

Wir erweitern die Algebra $A^{[2]}$ nun um die Konstruktoren, die Selektoren und die Testfunktionen und erhalten so die Algebra $A^{[3]}$:

Definition 15 *Seien A und m wie in Definition 13. Wir erweitern die Algebra $A^{[2]}$ zur Algebra $A^{[3]}$ mit Signatur $(AS_m, AF_m \setminus NF_m)$ durch:*

- $s^{A^{[3]}} = s^{A^{[2]}}$ *für alle* $s \in AS_m$
- $f^{A^{[3]}} = f^{A^{[2]}}$ *für alle* $f \in AF_m \setminus (NF_m \cup K_m \cup \{\mathtt{select}_k^j, \mathtt{is}_k\mathtt{?} \mid k \in K_m\})$
- *Für alle* $k\colon s_1, \ldots, s_n \to s \in K_m$*:*

$$k^{A^{[3]}}(x_1, \ldots, x_n) := \begin{cases} k(x_1, \ldots, x_n) & \text{falls } x_i \neq \bot_{s_i}^{A^{[2]}} \text{ für alle } i \\ \bot_s^{A^{[2]}} & \text{anderenfalls} \end{cases}$$

- *Für alle* $k\colon s_1, \ldots, s_n \to s \in K_m$*:*

$$\mathtt{is}_k\mathtt{?}^{A^{[3]}}(x) := \begin{cases} \underline{\text{true}} & \text{falls } x = k(x_1, \ldots, x_n) \text{ für geeignete } x_i \\ \bot_{bool} & \text{falls } x = \bot_s^{A^{[2]}} \\ \underline{\text{false}} & \text{anderenfalls} \end{cases}$$

- *Für alle* $k\colon s_1, \ldots, s_n \to s \in K_m$ *und* $j = 1, \ldots, n$*:*

$$\mathtt{select}_k^{j\,A^{[3]}}(x) := \begin{cases} x_j & \text{falls } x = k(x_1, \ldots, x_j, \ldots, x_n) \\ & \quad \text{für geeignete } x_i \neq \bot_{s_i}^{A^{[2]}}, i = 1 \ldots n \\ \bot_{s_j}^{A^{[2]}} & \text{anderenfalls} \end{cases}$$

Auf obiges Beispiel angewandt, erhalten wir

$$\begin{aligned} cons^{N^{[3]}}(91, cons(47, nil)) &= cons(91, cons(47, nil)) \\ \mathtt{select}_{cons}^{1\;N^{[3]}}(cons(47, nil)) &= 47 \\ \mathtt{is}_{cons}\mathtt{?}^{N^{[3]}}(cons(47, nil)) &= \underline{\text{true}} \end{aligned}$$

Lemma 4 *Wenn A eine Standardalgebra ist, dann auch $A^{[3]}$.*

Beweis: Nach Konstruktion ist $A^{[2]}$ eine Standardalgebra. Die zusätzlichen Funktionen in $A^{[3]}$ sind strikt und daher monoton, also ist auch $A^{[3]}$ eine Standardalgebra. □

Zur vollständigen Definition von $[\![m]\!](A)$ fehlt uns jetzt nur noch die Angabe der Semantik der rekursiven Funktionszeichen. Hierzu sind aus der Literatur verschiedene Ansätze bekannt: operationelle, denotationelle und algebraische Semantik. [Gue79] enthält einen Vergleich dieser drei Methoden und zeigt, daß alle drei im Endeffekt die gleiche Semantik liefern. Die Auswahl einer Methode hängt also davon ab, für welchen Aspekt der Semantik man sich interessiert.

Unser Interesse liegt insbesondere in den semantischen Eigenschaften von Programmen, für die wir keine spezielle Parameteralgebra zugrunde legen wollen. Für Betrachtungen dieser Art ist die algebraische Semantik von Vorteil, da sie aus zwei getrennten Teilen besteht: der eine Teil reflektiert die Rekursionsstruktur der betrachteten Programme, während der andere Teil die

Eigenschaften der benutzten Basisfunktionen berücksichtigt. Aus diesem Grunde haben wir uns hier, im Gegensatz zu [Loe87], für die algebraische Semantik entschieden.

Zunächst benötigen wir einige Definitionen. Dabei sind wir nur an der Funktion kleene$\langle , \rangle$ interessiert, unfold$\langle\rangle$ und drain$\langle\rangle$ sind bloß Hilfsfunktionen zur Definition von kleene$\langle , \rangle$.

Definition 16 *Es sei m ein Modul. Wir definieren einige Funktionen auf Termen. Dabei sei jeweils x eine Variable, $f \in AF_m \setminus NF_m$ und $F \in NF_m$ mit $(F(x_1, \ldots, x_n) \Leftarrow body) \in PR_m$. Den Index m an den Funktionen werden wir im allgemeinen nicht schreiben.*

- $\text{unfold}_m : \mathcal{T}(\Sigma_{Am}, X) \to \mathcal{T}(\Sigma_{Am}, X)$

$$\begin{aligned} \text{unfold}\langle x \rangle &:= x \\ \text{unfold}\langle f(t_1, \ldots, t_n) \rangle &:= f(\text{unfold}\langle t_1 \rangle, \ldots, \text{unfold}\langle t_n \rangle) \\ \text{unfold}\langle F(t_1, \ldots, t_n) \rangle &:= body_{x_1 \ldots x_n}^{\text{unfold}\langle t_1 \rangle \ldots \text{unfold}\langle t_n \rangle} \end{aligned}$$

- $\text{unfold}_m : \mathbb{N}, \mathcal{T}(\Sigma_{Am}, X) \to \mathcal{T}(\Sigma_{Am}, X)$

$$\begin{aligned} \text{unfold}\langle 0, t \rangle &:= t \\ \text{unfold}\langle n+1, t \rangle &:= \text{unfold}\langle \text{unfold}\langle n, t \rangle \rangle \end{aligned}$$

- $\text{drain} : \mathcal{T}(\Sigma_{Am}, X) \to \mathcal{T}(\Sigma_{Am} \setminus NF_m, X)$

$$\begin{aligned} \text{drain}\langle x \rangle &:= x \\ \text{drain}\langle f(t_1, \ldots, t_n) \rangle &:= f(\text{drain}\langle t_1 \rangle, \ldots, \text{drain}\langle t_n \rangle) \\ \text{drain}\langle F(t_1, \ldots, t_n) \rangle &:= \bot \end{aligned}$$

- $\text{kleene} : \mathbb{N}, \mathcal{T}(\Sigma_{Am}, X) \to \mathcal{T}(\Sigma_{Am} \setminus NF_m, X)$

$$\text{kleene}\langle n, t \rangle := \text{drain}\langle \text{unfold}\langle n, t \rangle \rangle$$

Lemma 5

$$\text{unfold}\langle t_{x_1 \ldots x_n}^{t_1 \ldots t_n} \rangle = \text{unfold}\langle t \rangle_{x_1 \ldots x_n}^{\text{unfold}\langle t_1 \rangle \ldots \text{unfold}\langle t_n \rangle}$$

Beweis: Das ist eine einfache strukturelle Induktion über t. □

Lemma 6

$$\text{unfold}\langle m, t_{x_1 \ldots x_n}^{t_1 \ldots t_n} \rangle = \text{unfold}\langle m, t \rangle_{x_1 \ldots x_n}^{\text{unfold}\langle m, t_1 \rangle \ldots \text{unfold}\langle m, t_n \rangle}$$

Beweis: durch Induktion nach m unter Berücksichtigung von Lemma 5. □

Lemma 7

$$\text{drain}\langle t_{x_1 \ldots x_n}^{t_1 \ldots t_n} \rangle = \text{drain}\langle t \rangle_{x_1 \ldots x_n}^{\text{drain}\langle t_1 \rangle \ldots \text{drain}\langle t_n \rangle}$$

Beweis: Das ist eine einfache strukturelle Induktion über t. □

Lemma 8 *Es sei* $F \in NF_m$ *und* $(F(x_1, \ldots, x_n) \Leftarrow body) \in PR_m$.

$$\text{kleene}\langle m+1, F(t_1, \ldots, t_n)\rangle = (\text{kleene}\langle m, body\rangle)^{\text{kleene}\langle m+1,t_1\rangle \ldots \text{kleene}\langle m+1,t_n\rangle}_{x_1 \ldots x_n}$$

Beweis: folgt direkt aus Lemma 6 und Lemma 7. □

Die Folge kleene$\langle 0, t\rangle$, kleene$\langle 1, t\rangle$, kleene$\langle 2, t\rangle, \ldots$ heißt in [Gue79] die *Kleene Folge* für t.

In dem Beispiel von Abbildung 5, Seite 39, erhalten wir beispielsweise für den Term $l = cons(91, cons(47, nil))$:

$$\begin{aligned}
\text{kleene}\langle 0, sum(l)\rangle &= \bot_{elem} \\
\text{kleene}\langle 1, sum(l)\rangle &= \texttt{if is}_{nil}\texttt{?}(l)\texttt{ then } 0 \\
&\quad\ \texttt{else select}^1_{cons}(l) + \bot_{elem} \\
\text{kleene}\langle 2, sum(l)\rangle &= \texttt{if is}_{nil}\texttt{?}(l)\texttt{ then } 0 \\
&\quad\ \texttt{else select}^1_{cons}(l) + \texttt{if is}_{nil}\texttt{?(select}^2_{cons}(l))\texttt{ then } 0 \\
&\quad\ \qquad\qquad\qquad\qquad\quad \texttt{else select}^1_{cons}(\texttt{select}^2_{cons}(l)) + \bot_{elem}
\end{aligned}$$

Wir haben oben davon gesprochen, daß die algebraische Semantik aus zwei Teilen besteht. Die obige Definition der Kleene Folge bildet den einen Teil, der die Rekursionsstruktur der Programme reflektiert. Die Basisfunktionen, d.h. in unserem Falle die Parameteralgebren, kommen erst jetzt zum Zuge:

Definition 17 *Es sei m ein Modul mit Signatur* (Σ_P, Σ_E). *Die* Semantik *von m ist der Funktor*

$$[\![m]\!] \colon Alg_{\Sigma_P} \to Alg_{\Sigma_E}$$

definiert für alle $A \in Alg_{\Sigma_P}$ *durch*

- $[\![m]\!](A) \mid_{(AS_m, EF_m \setminus NF_m)} := A^{[3]} \mid_{(AS_m, EF_m \setminus NF_m)}$
- *für alle* $f : s_1, \ldots, s_n \to s \in NF_m \cap EF_m$, $a_i \in s_i^{A^{[3]}}$:

$$f^{[\![m]\!](A)}(a_1, \ldots, a_n) := \bigsqcup_{i \geq 0} A^{[3]}(\text{kleene}\langle i, f(x_1, \ldots, x_n)\rangle)(\alpha[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n])$$

wobei $\alpha \in ?_{A^{[3]}}$ *eine beliebige Variablenbelegung ist.*

Lemma 9 $[\![m]\!]$ *ist wohldefiniert, und für alle Standardalgebren A über der Parametersignatur von m ist* $[\![m]\!](A)$ *eine Standardalgebra.*

Beweis: Für die Wohldefiniertheit ist nur zu zeigen, daß $[\![m]\!](A)$ nicht von der Wahl der Belegung α abhängt, dies folgt aber direkt aus

$$\mathcal{V}ar\langle \text{kleene}\langle i, f(x_1, \ldots, x_n)\rangle\rangle \subseteq \{x_1, \ldots, x_n\}$$

Die Denotationen der neuen Funktionszeichen sind selbst wieder monotone Funktionen (siehe [Gue79]), aus Lemma 4 folgt dann, daß $[\![m]\!](A)$ eine Standardalgebra ist. □

In dem Listenbeispiel von Abbildung 5, Seite 39, erhalten wir, wobei N wieder die Standardisierung der Algebra der natürlichen Zahlen bezeichnet:

$$\begin{aligned}
N^{[3]}(\text{kleene}\langle 0, sum(cons(91, cons(47, nil)))\rangle) &= \bot^N_{elem} \\
N^{[3]}(\text{kleene}\langle 1, sum(cons(91, cons(47, nil)))\rangle) &= 91 +^N \bot^N_{elem} \\
&= \bot^N_{elem} \\
N^{[3]}(\text{kleene}\langle 2, sum(cons(91, cons(47, nil)))\rangle) &= 91 +^N (47 +^N \bot^N_{elem}) \\
&= \bot^N_{elem} \\
N^{[3]}(\text{kleene}\langle 3, sum(cons(91, cons(47, nil)))\rangle) &= 91 +^N (47 +^N 0) \\
&= 138
\end{aligned}$$

Also ist

$$sum^{[\![list1]\!](N)}(cons(91, cons(47, nil))) = 138$$

3.4 Eigenschaften der Semantik

Die Semantik der Module ist persistent, d.h. die Trägermengen und Funktionen der Parameteralgebren werden durch die Anwendung der Semantik eines Moduls nicht verändert.

Lemma 10 *Es sei m ein Modul mit Signatur (Σ_P, Σ_E) und $A \in Alg_{\Sigma_P}$. Dann ist*

$$[\![m]\!](A) \mid_{\Sigma_P \cap \Sigma_E} = A \mid_{\Sigma_P \cap \Sigma_E}$$

Die beiden nächsten Lemmata spiegeln den konstruktiven Aspekt der Modulsemantik wider. Sie drücken aus, daß sich die in der Modulsemantik definierten Strukturen, nämlich die Trägermengen der neuen Sorten und die Denotationen der neuen Funktionszeichen, jeweils als Grenzwerte eines Iterationsprozesses beschreiben lassen. Im Falle der Trägermengen ist dies die Vereinigung aller endlichen Mengen von Konstruktortermen, im Falle der neuen Funktionen die kleinste obere Schranke einer Kette in einer fcpo. Später werden wir sehen, daß sich trotz dieser Analogie die beiden Strukturen unterschiedlich auf die Logik auswirken.

In Definition 13 haben wir die Trägermengen der neuen Sorten als Termsprache definiert, gebildet aus den Konstruktoren und den Trägern der Parameteralgebra. Wir können nun, unter Verwendung von Variablen, eine Darstellung der neuen Trägermengen angeben, die die Rolle der Konstruktoren und der Parameteralgebra voneinander trennt (man vergleiche hierzu auch unsere Motivation zur Verwendung der algebraischen Semantik, Seite 42).

Wir bezeichnen hier und im folgenden, jeweils im Kontext eines fest gewählten Modules m, mit $XP = (XP_s)_{s \in S}$ die "Teilfamilie" von X, die genau aus den Variablen von Parametersorte besteht:

$$XP_s = \begin{cases} X_s & \text{falls } s \in PS_m \\ \emptyset & \text{falls } s \in NS_m \end{cases}$$

Wir erinnern hier auch an unsere Konvention bezüglich X (Seite 34).

Definition 18 *Für ein Modul m ist die Menge der* Konstruktorterme *zur Sorte s:*

$$\mathcal{K}t_m(s) = \mathcal{T}((AS_m, K_m), XP)_s$$

Im Beispiel von Abbildung 5, Seite 39, ist also

$$\begin{aligned} \mathcal{K}t_{list1}(elem) &= X_{elem} \\ \mathcal{K}t_{list1}(list) &= \{nil, cons(x, nil), cons(x, cons(x, nil)), cons(x, cons(y, nil)), \ldots\} \end{aligned}$$

Lemma 11 *Es sei m ein Modul der Signatur (Σ_P, Σ_E) und $A \in Alg_{\Sigma_P}$ eine Parameteralgebra. Dann ist für alle Sorten $s \in AS_m$:*

$$s^{[\![m]\!](A)} = \{[\![m]\!](A)(t)(\alpha) \mid t \in \mathcal{K}t_m(s), \alpha \in ?^{+}_{XP, [\![m]\!](A)}\} \cup \{\bot_s^{[\![m]\!](A)}\}$$

Lemma 12 *Es sei m ein Modul mit Signatur (Σ_P, Σ_E), $A \in Alg_{\Sigma_P}$, $f: s_1, \ldots, s_n \to s \in EF \cap NF_m$ und $a_i \in s^{[\![m]\!](A)}$. Weiter sei $\alpha \in ?_{X, [\![m]\!](A)}$ mit $\alpha(x_i) = a_i$ für alle $i = 1, \ldots, n$. Dann gilt*

- *entweder $f^{[\![m]\!](A)}(a_1, \ldots, a_n) = \bot_s^{[\![m]\!](A)}$ und*

  $$[\![m]\!](A)(\text{kleene}\langle j, f(x_1, \ldots, x_n)\rangle)(\alpha) = \bot_s^{[\![m]\!](A)}$$

  für alle $j \in \mathbb{N}$

- *oder $f^{[\![m]\!](A)}(a_1, \ldots, a_n) = c \neq \bot_s^{[\![m]\!](A)}$ und es gibt ein $j_0 \in \mathbb{N}$ so daß*

  $$[\![m]\!](A)(\text{kleene}\langle j, f(x_1, \ldots, x_n)\rangle)(\alpha) = c$$

  für alle $j \geq j_0$

Beweis: Dies ist eine triviale Folgerung aus der Definition der Semantik von m und aus den Eigenschaften von Ketten in fcpo's. □

Das folgende Lemma zeigt eine Fixpunkteigenschaft der neuen Funktionen. Dies bedeutet, daß wir später in der Logik ein neues Funktionszeichen stets durch den dazugehörigen Funktionsrumpf ersetzen dürfen.

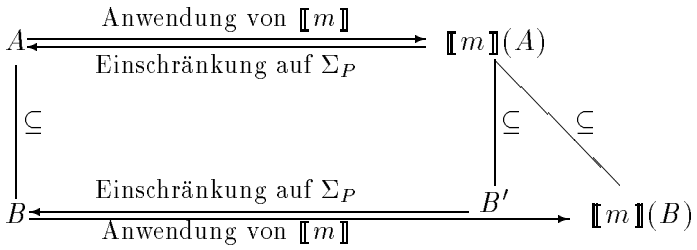


Abbildung 6: Beziehung der Algebren in Lemma 15

Lemma 13 *Es sei m ein Modul mit Signatur (Σ_P, Σ_E), $A \in Alg_{\Sigma_P}$, $t \in \mathcal{T}(\Sigma_E, X)$ und $\alpha \in ?_{X,[\![m]\!](A)}$ eine Belegung. Dann ist*

$$[\![m]\!](A)(t)(\alpha) = [\![m]\!](A)(\mathrm{unfold}\langle t \rangle)(\alpha)$$

Beweis: Siehe [Gue79]. □

Die beiden nächsten Lemmata zeigen eine Verträglichkeit zwischen der Semantik der Module und der Teilalgebrarelation auf. Man beachte hierzu aber auch Lemma 28, Seite 66.

Lemma 14 *Es sei m ein Modul mit Signatur (Σ_P, Σ_E), $A, B \in Alg_{\Sigma_P}$ und $A \subseteq B$.*
Dann ist $[\![m]\!](A) \subseteq [\![m]\!](B)$.

Beweis: Die Teilmengenbeziehung zwischen den Trägermengen ist offensichtlich. Die Übereinstimmung der Denotationen der neuen Funktionszeichen auf den Trägermengen von $[\![m]\!](A)$ folgt direkt aus der Definition der Semantik. □

Lemma 15 *Es sei m ein Modul mit Signatur (Σ_P, Σ_E), $\Sigma_P \subseteq \Sigma_E$, $A \in Alg_{\Sigma_P}$ und $B' \in Alg_{\Sigma_E}$ mit $B' \subseteq [\![m]\!](A)$.*
Dann ist $B' = [\![m]\!](B'\,|_{\Sigma_P})$.

Beweis: (Siehe auch Abbildung 6.) Wir definieren die Abkürzung $B := B'\,|_{\Sigma_P}$. Da auf Grund der Persistenz von $[\![m]\!]$ (Lemma 10) $A = [\![m]\!](A)\,|_{\Sigma_P}$, folgt $B \subseteq A$ sofort aus $B' \subseteq [\![m]\!](A)$. Wegen Lemma 14 gilt dann $[\![m]\!](B) \subseteq [\![m]\!](A)$. Wir zeigen nun, daß B' und $[\![m]\!](B)$ die gleichen Trägermengen haben. Da beide Algebren Teilalgebren von $[\![m]\!](A)$ sind, müssen dann auch die Denotationen der Funktionszeichen übereinstimmen, und somit $B' = [\![m]\!](B)$.

Für die Parametersorten folgt die Gleichheit der Trägermengen aus der Persistenz von $[\![m]\!]$ (Lemma 10).

$s^{[\![m]\!](B)} \subseteq s^{B'}$ für $s \in NS$:

folgt per struktureller Induktion aus $B \subseteq A$.

$s^{B'} \subseteq s^{[\![m]\!](B)}$ für $s \in NS$:

Wir nehmen an, daß es einen Träger von neuer Sorte in B' gibt, der kein Träger in $[\![m]\!](B)$ ist. Es sei x ein bezüglich der Teiltermordnung $\trianglelefteq$ minimaler Träger von B' mit dieser Eigenschaft. Ein solches x muß unter unserer Annahme existieren, da $\trianglelefteq$ eine noethersche Ordnung ist.

Nach Konstruktion der Trägermengen ist x dann von der Form $k(x_1, \ldots, x_n)$ für einen geeigneten Konstruktor k. Da B' als Teilalgebra von $[\![m]\!](A)$ abgeschlossen ist unter den Denotationen der Selektoren, sind die x_i Träger von B' und auf Grund der Minimalität von x dann auch Träger von $[\![m]\!](B)$. Dann ist aber auch $x = k(x_1, \ldots, x_n)$ ein Träger von $[\![m]\!](B)$, dies widerspricht unserer Annahme. □

4 Logik

In diesem Kapitel werden wir die Prädikatenlogik erster Stufe in eine Beziehung zur Semantik der Module setzen. In Abschnitt 4.1 führen wir den für diese Arbeit grundlegenden Blickwinkel ein: Die Prädikatenlogik erster Stufe, zusammen mit der Semantik der Module, liefert eine neue Logik, die die Prädikatenlogik erster Stufe erweitert.

Wir werden dann in Abschnitt 4.2 den Begriff des Domainoperators untersuchen, der es uns erlauben wird, unsere Untersuchungen in einem allgemeineren Rahmen als dem aller (Standard-) Algebren zu einer Signatur durchzuführen. Domainoperatoren ordnen Signaturen jeweils Klassen von Algebren zu, die essentiell die gleichen angenehmen modelltheoretischen Eigenschaften wie die Klasse *aller* Algebren zu einer Signatur haben. Wir zeigen, daß alle "vernünftigen" Klassen von Algebren durch einen Domainoperator beschrieben werden können. Hierzu zählt allerdings *nicht* die Klasse der termgenerierten Algebren. Diese Klasse können wir aber auch getrost aus unseren Betrachtungen ausschließen, da wir im Zusammenhang mit unserem Ausgangsproblem der Wiederverwertbarkeit und der Top-Down Entwicklung von Modulen nicht davon ausgehen können, daß die Parametersignatur alle zur Generierung der Träger notwendigen Funktionen enthält.

In Abschnitt 4.3 definieren wir schwächste Parameterbedingungen und illustrieren diesen Begriff an Hand einiger Beispiele. Den für ein späteres Kapitel wichtigen Mechanismus der Ersetzungssysteme auf Formeln werden in Abschnitt 4.4 vorstellen. In Abschnitt 4.5 schließlich untersuchen wir den Begriff der symbolischen Auswertbarkeit von Funktionen.

4.1 Die Logik $[m|=$

Wir möchten die Logik erster Stufe dazu benutzen, Eigenschaften der Parameter- und der Exportalgebren auszudrücken. Da wir nur an Eigenschaften von Algebren und nicht an denen einzelner Träger interessiert sind, können wir unsere Betrachtungen im allgemeinen auf Sätze beschränken. Wir erinnern nochmals daran, daß X stets eine passende Variablenfamilie bezeichnet.

Definition 19 *Für ein Modul m mit Signatur (Σ_P, Σ_E) bezeichnet*

$$\begin{aligned} \mathcal{S}en_m &= \mathcal{S}en(\Sigma_E, X) \\ \mathcal{P}Sen_m &= \mathcal{S}en(\Sigma_P, X) \end{aligned}$$

Wir vereinbaren einige Notationen. Hierbei nutzen wir die Tatsache aus, daß sowohl die Parameter- als auch die Exportsignatur eine Standardsignatur ist.

$$\begin{array}{rll} (t_1 \sqsubseteq t_2) & \text{steht für} & (t_1 = \bot \vee t_1 = t_2) \\ \forall x \in s \,.\, w & \text{steht für} & \forall x : s \,.\, x = \bot_s \vee w \\ \exists x \in s \,.\, w & \text{steht für} & \exists x : s \,.\, x \neq \bot_s \wedge w \end{array}$$

$$\begin{aligned} \texttt{True} &\text{ steht für } true = true \\ \texttt{False} &\text{ steht für } true = false \end{aligned}$$

Wir werden die Junktoren $\wedge$ und $\vee$ auch auf Familien von Formeln $(w_i)_{i \in I}$, wobei I eine endliche Indexmenge ist, anwenden. Zu diesem Zweck definieren wir

$$\begin{aligned} \bigwedge_{i \in I} w_i &= \begin{cases} (w_j \wedge \bigwedge_{i \in I \setminus \{j\}} w_j) & \text{falls } I \neq \emptyset \text{ und } j \in I \\ \texttt{True} & \text{falls } I = \emptyset \end{cases} \\ \bigvee_{i \in I} w_i &= \neg \bigwedge_{i \in I} \neg w_i \end{aligned}$$

Diese Definitionen sind etwas salopp, da sie von der Auswahl der $j \in I$ abhängen. Da aber alle hierbei möglichen Resultate kongruent sind, stört diese Unexaktheit nicht weiter. Auf ähnliche Art und Weise definieren wir eine Allquantifizierung über einer endlichen Menge von Variablen. Sei M eine endliche Teilmenge von X, dann definieren wir

$$\begin{aligned} \exists M \,.\, w &= \begin{cases} \exists x \in s \,.\, \exists M \setminus \{x\} \,.\, w & \text{falls } M \neq \emptyset \text{ und } x \in M \cap X_s \\ w & \text{falls } M = \emptyset \end{cases} \\ \forall M \,.\, w &= \neg \forall M \,.\, \neg w \end{aligned}$$

Auch hier ist die so definierte Formel nur eindeutig bis auf Kongruenz. Eine typische Anwendung dieser Konstruktion ist

$$\forall \mathcal{V}ar\langle w \rangle \,.\, w$$

Dies entspricht dem universellen Abschluß von w, d.h. alle freien Variablen von w werden durch einen Allquantor gebunden. Man beachte aber, daß bei dieser Form der Quantifizierung der undefinierte Träger ausgeschlossen bleibt. Ferner werden wir eine Folge von Quantifizierungen der Art

$$\forall x_1 : s_1, \forall x_2 : s_2, \ldots, \forall x_n : s_n$$

manchmal durch die "vektorisierte Form" $\forall \bar{x} : \bar{s}$ abkürzen.

Wir können die Logik erster Stufe nun direkt dazu benutzen, um Aussagen über Parameteralgebren zu treffen. Wie aber können wir Aussagen über Exportalgebren formulieren? Zunächst können wir hierzu natürlich ebenfalls die Logik erster Stufe benutzen. Der Modellbegriff der Logik erster Stufe ist hierfür allerdings zu allgemein, da nun als Modelle nur noch Algebren aus dem Wertebereich der Semantik eines Moduls in Frage kommen. Aus diesem Grunde definieren wir eine neue Logik, die diese Tatsache berücksichtigt. Die Formeln dieser neuen Logik werden über der Exportsignatur eines Moduls gebildet, die Klasse der möglichen Modelle besteht aber aus den möglichen Parameteralgebren des Moduls, d.h. also den Standardalgebren über der Parametersignatur. Das eigentlich "Neue" an dieser Logik ist die Gültigkeitsrelation $[m|=$ zwischen Algebren und Formeln, sie berücksichtigt nämlich die Semantik des Moduls m. Zur Definition von $[m|=$ benutzen wir die Logik erster Stufe, so daß wir auf die Eigenschaften der Logik erster Stufe zurückgreifen können, um Eigenschaften unserer neuen Logik zu beweisen.

Definition 20 *Es sei m ein Modul mit Signatur (Σ_P, Σ_E), $A \in Alg_{\Sigma_P}$ und $w \in \mathcal{S}en_m$. Wir definieren*

$$A\ [m|\!=\ w \qquad \Leftrightarrow \qquad [\![m]\!](A) \models w$$

Auch hier benutzen wir wieder einige nützliche Konventionen. Für eine Klasse $C \subseteq Alg_{\Sigma_P}$ schreiben wir $C\ [m|\!=\ w$, falls $A\ [m|\!=\ w$ für alle $A \in C$. Insbesondere schreiben wir $[m|\!=\ w$ für $Alg_{\Sigma_P}\ [m|\!=\ w$. Falls $W \subseteq \mathcal{S}en_m$, dann bedeutet $A\ [m|\!=\ W$: $A\ [m|\!=\ w$ für alle $w \in W$, und analog für Klassen von Modellen. Die m-Theorie von A und analog die m-Theorie einer Klasse $C \subseteq Alg_{\Sigma_P}$ sind definiert als

$$\begin{aligned} Th_m(A) &:= \{w \in \mathcal{S}en_m \mid A\ [m|\!=\ w\} \\ Th_m(C) &:= \{w \in \mathcal{S}en_m \mid C\ [m|\!=\ w\} \end{aligned}$$

Als Beispiel betrachten wir wieder das Modul $list1$ aus Abbildung 5, Seite 39. N sei wiederum die Standardisierung des Primmodells der Arithmetik. Dann gilt:

$$N\ [m|\!=\ \forall l_1, l_2 : list\,.\, sum(app(l_1, l_2)) = sum(l_1) + sum(l_2)$$

Als unmittelbare Folge der Persistenz der Modulsemantik (Lemma 10) erhalten wir:

Lemma 16 *Es sei m ein Modul mit Signatur (Σ_P, Σ_E), $A \in Alg_{\Sigma_P}$ und $w \in \mathcal{S}en(\Sigma_P \cap \Sigma_E, X)$. Dann gilt*

$$A \models w \qquad \Leftrightarrow \qquad A\ [m|\!=\ w$$

Dies bedeutet, daß unsere neue Logik mindestens so ausdrucksstark ist wie die Logik erster Stufe. Im allgemeinen ist sie aber echt stärker als die Logik erster Stufe, wie wir in Abschnitt 4.3 sehen werden.

4.2 Domainoperatoren

Wir sind oft nicht an der gesamten Klasse Alg_{Σ_P} von Modellen interessiert, sondern wollen unsere Betrachtungen auf Teilklassen von Alg_{Σ_P} einschränken. Dabei müssen wir, um auch für solche Teilklassen Eigenschaften beweisen zu können, "vernünftige" Einschränkungen an diese Teilklassen machen.

Wir nennen eine Klasse von Algebren kompakt, wenn in ihr der Kompaktheitssatz der Prädikatenlogik erster Stufe gilt:

Definition 21 *Eine Klasse $\mathcal{C} \subseteq Alg_{\Sigma_P}$ heißt* kompakt, *wenn für jede Menge $W \subseteq \mathcal{S}en(\Sigma, X)$ von Formeln gilt:*

> *Wenn jede endliche Teilmenge von W ein Modell in $\mathcal{C}$ hat,*
> *dann hat W ein Modell in $\mathcal{C}$.*

Die Auswahl einer Klasse von Algebren zu einer Parametersignatur wird durch den Begriff des Domainoperators formalisiert:

Definition 22 *Ein* Domainoperator $\Im$ *bildet jede Standardsignatur* Σ *in eine Klasse* $\Im_\Sigma \subseteq Alg_\Sigma$ *ab, wobei gilt:*

1. $\Im_\Sigma$ *ist kompakt.*
2. $\Im_\Sigma$ *ist unter elementaren Teilalgebren abgeschlossen.*
3. *Es gilt die* Erweiterungseigenschaft*: Zu jeder Menge* C *von neuen Konstantensymbolen gibt es eine Umbenennung* C' *mit:*

$$\Im_{\Sigma\cup C'} = \{A \in Alg_{\Sigma\cup C'} \mid A|_\Sigma \in \Im_\Sigma\}$$

Die Erweiterungseigenschaft drückt also aus, daß wir stets neue Konstantenzeichen einführen können, die dann an beliebige Träger der entsprechenden Sorte gebunden werden können. Die Umbenennung benötigen wir nur für den Fall, daß $\Im$ die Bedeutung von bestimmten Konstantenzeichen festlegt. Im allgemeinen wird jedoch keine Umbenennung notwendig sein, und wir werden sie deshalb nicht extra erwähnen.

Die folgenden Abbildungen sind *keine* Domainoperatoren:

1. Die Abbildung $\Im^{faststrikt}$, die jede Standardsignatur Σ in die Klasse $\Im_\Sigma^{faststrikt}$ der Standardalgebren abbildet, in denen alle Funktionen für fast alle Tupel von definierten Argumenten (d.h. alle bis auf endlich viele) einen definierten Wert liefern. Diese Definition verletzt die Kompaktheitsforderung, denn die Menge

$$W = \left\{ \exists\{x_i^j \mid 1 \le i \le n, 1 \le j \le m\} \,.\, \bigwedge_{j=1}^{m} f(x_1^j, \ldots, x_n^j) = \bot \wedge \bigwedge_{\substack{j,k\le n \\ j\neq k}} \bigvee_{i=1}^{n} x_i^j \neq x_i^k \mid m \in \mathbb{N} \right\}$$

   hat kein Modell in $\Im_\Sigma^{faststrikt}$, wohl aber jede endliche Teilmenge von W.

2. Die Abbildung, die jede Standardsignatur in die Klasse der Standardalgebren mit Kardinalität $> \aleph_0$ abbildet, verletzt den Abschluß unter elementaren Teilalgebren, da nach dem $\downarrow$LST-Satz der Prädikatenlogik jede Algebra unendlicher Kardinalität eine elementare Teilalgebra der Kardinalität $\aleph_0$ hat.

3. Die Abbildung, die jede Standardsignatur Σ in die Klasse der Σ-generierten Algebren abbildet, ist kein Domainoperator, da sie die Erweiterungseigenschaft nicht erfüllt. Dabei heißt eine Algebra $A \in Alg_\Sigma$ Σ-generiert, wenn es für jedes $a \in s^A$ ein $t \in \mathcal{T}(\Sigma,\emptyset)_s$ gibt mit $a = A(t)(\alpha)$ für eine geeignete Belegung $\alpha \in ?_{A,X}$.

Andererseits zeigt das nächste Lemma, daß eine wichtige Klasse von Abbildungen die Bedingungen von Definition 22 erfüllt:

(1) $\forall x{:}\mathit{bool}\,.\,(x = \bot_{bool} \vee x = true \vee x = false)$
(2) $\bot_{bool} \neq true$
(3) $\bot_{bool} \neq false$
(4) $true \neq false$
(5) $\forall x, y{:}s\,.\,[(x =_s y) = true \mathrel{\mathfrak{X}} (x = y \wedge x \neq \bot_s)]$
(6) $\forall x, y{:}s\,.\,[(x =_s y) = false \mathrel{\mathfrak{X}} (x \neq y \wedge x \neq \bot_s \wedge y \neq \bot_s)]$
(7) $\forall x, y{:}s\,.\,[(x =_s y) = \bot_{bool} \mathrel{\mathfrak{X}} (x = \bot_s \vee y = \bot_s)]$
(8) $\forall x, y{:}s\,.\,\texttt{if}\ true\ \texttt{then}\ x\ \texttt{else}\ y\ \texttt{fi} = x$
(9) $\forall x, y{:}s\,.\,\texttt{if}\ false\ \texttt{then}\ x\ \texttt{else}\ y\ \texttt{fi} = y$
(10) $\forall x, y{:}s\,.\,\texttt{if}\ \bot_{bool}\ \texttt{then}\ x\ \texttt{else}\ y\ \texttt{fi} = \bot_s$
(11) $\forall \vec{x}_i, \vec{y}_i{:}\vec{s}.\ [\bigwedge_{i=1\ldots n}(x_i \sqsubseteq y_i) \supset f(x_1, \ldots, x_n) \sqsubseteq f(y_1, \ldots, y_n)]$

Abbildung 7: Axiomenschemata für Standardalgebren.

Lemma 17 *Die Abbildung ι bilde jede Standardsignatur $\Sigma = (S, F)$ auf eine Menge von Sätzen $\iota(\Sigma) \subseteq \mathcal{S}en(\Sigma, X)$ ab, wobei in $\iota(\Sigma)$ nur endlich viele Konstantenzeichen vorkommen.*

Dann ist die Abbildung, die jede Standardsignatur Σ auf die Klasse der Σ-Algebren A mit $A \models \iota(\Sigma)$ abbildet, ein Domainoperator.

Beweis: Zunächst beachte man, daß die Klasse der (S, F)-Standardalgebren identisch ist mit der Klasse der (S, F)-Algebren, die Modelle der von den Axiomschemata in Abbildung 7 beschriebenen Axiome sind, wobei die f die Menge der Funktionszeichen F und die s die Menge der Sorten S durchlaufen. Die Axiomschemata (7) und (10) sind dabei "fast redundant": sie folgen bereits mit der Monotoniebedingung (11) aus (5) und (6), bzw. aus (8) und (9), falls die Trägermengen mindestens zwei definierte Träger enthalten. Aus diesem Grunde sind die Kompaktheit und der Abschluß unter elementaren Teilalgebren einfache Folgerungen aus dem verschärften abwärts gerichteten Löwenheim-Skolem-Tarski Satz (Seite 29), bzw. dem Kompaktheitssatz (Seite 29) der Prädikatenlogik erster Stufe. Der Beweis der dritten Bedingung von Definition 22 ist trivial. □

Als Folgerung von Satz 17 sind die folgenden Abbildung Domainoperatoren:

1. Die Abbildung $\Im^f$, die jede Standardsignatur Σ auf die gesamte Klasse Alg_Σ der Σ-Standardalgebren abbildet. Wähle hierzu $\iota(\Sigma) = \emptyset$.

2. Die Abbildung $\Im^{strikt}$, die jede Standardsignatur Σ auf die Klasse der Σ-Standardalgebren abbildet, in denen die Denotationen aller Funktionszeichen außer den $\texttt{ifthenelse}_s$ strikt sind. Hier wählen wir

$$\begin{aligned}\iota(S, F) \quad := \quad & \{\forall x_1 : s_1, \ldots, x_n : s_n\,.\,x_1 = \bot \vee \ldots \vee x_n = \bot \supset f(x_1, \ldots, x_n) = \bot \mid \\ & f \in F \setminus \{\texttt{ifthenelse}_s \mid s \in S\}\}\end{aligned}$$

3. Die Abbildung $\Im^{st}$, die jede Standardsignatur Σ auf die Klasse der Σ-Standardalgebren abbildet, in denen die Denotationen aller Funktionszeichen außer den $\texttt{ifthenelse}_s$ strikt

und total sind:

$$\iota(S,F) \;:=\; \{\forall x_1 : s_1, \ldots, x_n : s_n \,.\, x_1 = \bot \vee \ldots \vee x_n = \bot \;\mathfrak{D}\; f(x_1, \ldots, x_n) = \bot \mid f \in F \setminus \{\mathtt{ifthenelse}_s \mid s \in S\}\}$$

4. Die Abbildung $\Im^{seq}$, die jede Standardsignatur Σ auf die Klasse der Σ-Standardalgebren abbildet, in denen alle Funktionen sequentiell ([Vui74]) sind. Die Untersuchung sequentieller Funktionen wird durch die Betrachtung der *Berechnungen* von Funktionswerten motiviert: Intuitiv gesprochen heißt eine Funktion *sequentiell*, falls ihre Argumente stets nacheinander ausgewertet werden können, ohne daß dadurch eine vermeidbare Nichtterminierung eintritt. Die Auswahl des nächsten auszuwertenden Argumentes kann dabei von den bisher ermittelten Argumentwerten abhängen. Beispielsweise ist die sequentielle $ifthenelse$ Funktion, wie wir sie in Standardalgebren voraussetzen sequentiell: Zuerst wird das erste Argument ausgewertet, in Abhängigkeit vom Ergebnis dann das zweite oder dritte. Der Funktionswert kann nun angegeben werden, ohne daß hierzu das verbleibende Argument ausgewertet werden muß. Im Gegensatz dazu ist die parallele *oder*-Funktion, die true zurückliefert, sobald eines ihrer Argument zu true ausgewertet werden kann (selbst wenn die Auswertung des anderen Argumentes nicht terminiert), nicht sequentiell. Die Bedeutung der sequentiellen Funktionen liegt darin begründet, daß alle strikten Funktionen sequentiell sind (die Auswertungsreihenfolge der Argumente ist dann sogar irrelevant), und daß die Sequentialität unter Fixpunkten von rekursiven Programmen abgeschlossen ist ([Vui74]). Nach [Vui74] werden die sequentiellen Funktionen durch folgendes Axiom beschrieben:

$$\iota(S,F) := \left\{ \forall \bar{x} \in \bar{s} \,.\, \bigvee_{i=1}^{n} \forall \bar{y} : \bar{s} \,.\, (\bar{x} \sqsubseteq \bar{y} \wedge x_i = y_i \supset f(\bar{x}) = f(\bar{y})) \mid f \in F \right\}$$

5. Die Abbildung $\Im^n$ für $n \in \mathbb{N}$, die jede Standardsignatur Σ auf die Klasse der Σ-Standardalgebren A mit $\#(A) \leq n$ abbildet. Falls Σ l Sorten enthält, dann wählen wir hierzu

$$\iota(\Sigma) := \left\{ \bigvee_{\substack{n_1 + \ldots \\ + n_l = n}} \bigwedge_{i=1}^{l} \exists x_{i,1}, \ldots, x_{i,n_i} : s_i \,.\, \forall y_i \in s_i \,.\, \bigvee_{j=1}^{n_i} y_i = x_{i,j} \right\}$$

Weiterhin bezeichnen wir für einen Domainoperator $\Im$ mit $\Im^+$ die Beschränkung von $\Im$ auf Algebren, die zu allen Sorten ein definiertes Element enthalten:

$$\Im^+_{(S,F)} = \{A \in \Im_{(S,F)} \mid s^A \neq \{\bot^A_s\} \quad \text{für alle } s \in S$$

Lemma 18 *Falls $\Im$ ein Domainoperator ist, dann auch $\Im^+$.*

Beweis: Betrachte den Satz

$$P_S := \bigwedge_{s \in S} \exists x_s : s \,.\, x_s \neq \bot_s$$

Es ist $\Im^+_{(S,F)} = \{A \in \Im_{(S,F)} \mid A \models P_S\}$. Falls nun jede endliche Teilmenge von $W \subseteq \mathcal{S}en(\Sigma, X)$ eine Modell in $\Im^+_{(S,F)}$ hat, dann hat auch jede jede endliche Teilmenge von $W \cup P_S$ ein Modell in $\Im^+_{(S,F)}$ und damit in $\Im_{(S,F)}$. Wegen der Kompaktheit von $\Im_{(S,F)}$ hat dann auch $W \cup \{P_S\}$ ein Modell in $\Im_{(S,F)}$, das also auch in $\Im^+_{(S,F)}$ liegt.

Falls $B \in \Im^+_\Sigma$ und $A \preceq B$, dann ist wegen der Abschlußeigenschaft von $\Im_\Sigma$ unter elementaren Teilalgebren $A \in \Im_\Sigma$. Da elementare Teilmodelle die gleiche Theorie haben, folgt $A \models P_S$ aus $B \models P_S$, also $A \in \Im^+_\Sigma$.

Die Erweiterungseigenschaft ist trivialerweise erfüllt. □

$\Im^n$ ist im Vergleich zu den übrigen Domainoperatoren der obigen Aufstellung trivial, da $\Im^n_\Sigma$ keine "interessanten" Algebren enthält. Dies wird später näher präzisert werden. Mit dem Begriff der Reichhaltigkeit eines Domainoperators können wir solche trivialen Domainoperatoren wie $\Im^n$ ausschließen. Wir nennen einen Domainoperator $\Im$ reichhaltig, wenn $\Im_\Sigma$ mindestens die Algebren enthält, die mit gewissen algebraischen Spezifikationen definiert werden können:

Definition 23 *Ein Domainoperator $\Im$ heißt* reichhaltig, *falls für jede Semistandardsignatur Σ mit $\bot \notin \Sigma$ und endliche Σ-Gleichungsmenge E mit der Eigenschaft*

$$\mathcal{T}(\Sigma, \emptyset)/E \quad \textit{ist eine Semistandardalgebra}$$

die Standardisierung von $\mathcal{T}(\Sigma, \emptyset)/E$ ein Element von $\Im_{\Sigma^*}$ ist, wobei Σ^* die Standardisierung von Σ bezeichnet.

Hierbei interessiert uns insbesondere der Fall der $\Sigma = (S, F)$ mit

$$\begin{aligned} F \cap \{\bot_s, \texttt{ifthenelse}_s, =_s \mid s \in S\} &= \emptyset \\ bool &\in S \\ true, false &\in F \end{aligned}$$

In diesem Fall bedeutet die Reichhaltigkeitsbedingung für eine Σ-Gleichungsmenge E:

1. $bool^{\mathcal{T}(\Sigma,\emptyset)/E} = \{[true], [false]\}$
2. $true^{\mathcal{T}(\Sigma,\emptyset)/E} \neq false^{\mathcal{T}(\Sigma,\emptyset)/E}$

In der Terminologie von [WPP+83] heißt dies, daß die Spezifikation $(\Sigma, E, BOOL)$ *Hierarchie-persistent* (hierarchy-persistent) ist.

Der Domainoperator $\Im^n$ ist also nicht reichhaltig, da er (für nichttriviale Signaturen) die von der leeren Gleichungsmenge erzeugte Termalgebra nicht enthält.

Andererseits sind alle Domainoperatoren, die den strikten totalen Domainoperator umfassen, reichhaltig:

Lemma 19 *Es sei $\Im$ ein Domainoperator, so daß für alle Standardsignaturen Σ gilt: $\Im^{st}_{\Sigma} \subseteq \Im_{\Sigma}$. Dann ist $\Im$ reichhaltig.*

Damit sind also die Domainoperatoren $\Im^{f}$, $\Im^{strict}$, $\Im^{st}$ und $\Im^{seq}$ reichhaltig.

Man beachte, daß wir von den Domainoperatoren nicht verlangt haben, daß sie unter der Anwendung der Semantik eines Moduls abgeschlossen sind. Wir verlangen also *nicht*, daß $[\![m]\!](A) \in \Im_{\Sigma_E}$ für alle $A \in \Im_{\Sigma_P}$.

4.3 Parameterbedingungen

Der Begriff der Parameterbedingung verbindet die Logik $[m|=$ mit der Logik der ersten Stufe $\models$.

Definition 24 *Es sei m ein Modul mit Signatur (Σ_P, Σ_E), $\Im$ ein Domainoperator und $w \in \mathcal{S}en_m$. Ein Satz $v \in \mathcal{P}Sen_m$ ist eine $\Im, m$-*Parameterbedingung *von w, falls für alle $A \in \Im_{\Sigma_P}$ gilt:*

$$A \models v \qquad \Longrightarrow \qquad A\ [m|\!= w$$

*Ein Satz $v \in \mathcal{P}Sen_m$ ist eine $\Im, m$-*schwächste Parameterbedingung*, abgekürzt* spab*, von w falls für alle $A \in \Im_{\Sigma_P}$ gilt:*

$$A \models v \qquad \Longleftrightarrow \qquad A\ [m|\!= w$$

Die folgenden Eigenschaften folgen unmittelbar aus der obigen Definition:

Lemma 20 *Spabs sind bis auf Äquivalenz eindeutig, das heißt: Sei m ein Modul mit Signatur (Σ_P, Σ_E), $\Im$ ein Domainoperator und $w \in \mathcal{S}en_m$. Dann gilt für alle spabs $v_1, v_2 \in \mathcal{P}Sen_m$: $\Im_{\Sigma_P} \models v_1 \between v_2$*

Lemma 21 *Sei m ein Modul mit Signatur (Σ_P, Σ_E), $\Im$ ein Domainoperator, $w_1, w_2 \in \mathcal{S}en_m$ und $v_1, v_2 \in \mathcal{P}Sen_m$.*

- *v_1 ist eine $\Im, m$-spab von w_1 genau dann, wenn $\neg v_1$ eine $\Im, m$-spab von $\neg w_1$ ist.*
- *Falls jeweils v_i eine $\Im, m$-spab von w_i $(i = 1, 2)$ ist, dann ist $v_1 \wedge v_2$ eine $\Im, m$-spab von $w_1 \wedge w_2$.*
- *Falls jeweils v_i eine $\Im, m$-spab von w_i $(i = 1, 2)$ ist, dann ist $v_1 \vee v_2$ eine $\Im, m$-spab von $w_1 \vee w_2$.*

Falls also w_1 und w_2 spabs haben, dann haben auch $w_1 \wedge w_2$ und $w_1 \vee w_2$ spabs. Die Umkehrung gilt im allgemeinen nicht: Sei w ein Satz, der keine spab hat (siehe unten für Beispiele). Dann haben trotzdem $w \wedge \neg w$ ($\between$ `False`) und $w \vee \neg w$ ($\between$ `True`) spabs, nämlich `False`, bzw. `True`.

Wir illustrieren den wichtigen Begriff der schwächsten Parameterbedingung mit einigen Beispielen:

PAR **SORTS** $elem$
OPNS $0 : \rightarrow elem$
$pred : elem \rightarrow elem$
BODY **FCTS** $isstandard : elem \rightarrow bool$
PROG $isstandard(x) \Leftarrow \texttt{if } x = 0 \texttt{ then } true$
$\texttt{else } isstandard(pred(x))$

Abbildung 8: Das Modul *sta* zur Unterscheidung von Prim- und Nichtprimmodellen.

PAR **SORTS** $elem$
BODY **SORTS** $list$
CONS $nil : \rightarrow list$
$cons : elem, list \rightarrow list$
FCTS $isin : elem, list \rightarrow bool$
PROG $isin(e, l) \Leftarrow \texttt{if is}_{nil}\texttt{?}(l) \texttt{ then } false$
$\texttt{else if select}^1_{cons}(l) = e \texttt{ then } true$
$\texttt{else } isin(e, \texttt{select}^2_{cons}(l))$

Abbildung 9: Das Modul *fin* zur Unterscheidung endlicher von unendlichen Algebren.

1. Es sei $list1$ das Modul von Abbildung 5, Seite 39. Der Satz

$$(\forall x \in elem \,.\, 0 + x = x \wedge x + 0 = x) \wedge$$
$$(\forall l_1, l_2 \in list \,.\, sum(app(l_1, l_2)) = sum(l_1) + sum(l_2))$$

   hat die $\Im^{strict}, list1$-spab:

$$(\forall x \in elem \,.\, 0 + x = x \wedge x + 0 = x) \wedge$$
$$(\forall x_1, x_2, x_3 \in elem \,.\, x_1 + (x_2 + x_3) = (x_1 + x_2) + x_3)$$

2. Betrachte das Modul *sta* aus Abbildung 8. Der Satz

$$w := \forall x \in elem . isstandard(x) = true$$

   hat keine $\Im^{strict}, sta$-spab, da das Prim- und das Nichtprimmodell der Arithmetik die gleiche Theorie haben, während nur eine dieser Algebren W erfüllt.

3. Das letzte Beispiel zeigt, daß auch bei Modulen mit primitiv rekursiven Funktionen eine spab nicht notwendig existiert. Betrachte das Modul fin aus Abbildung 9. Die Klasse der Standardalgebren A, für die $[\![fin]\!](A)$ den Satz

$$w := \exists l \in list \,.\, \forall x \in elem \,.\, isin(x, l) = true$$

erfüllt, ist gerade die Klasse der endlichen Σ_{P_m}-Standardalgebren. Da die Klasse der endlichen Algebren nicht durch einen Satz der Logik erster Stufe beschrieben werden kann (siehe Seite 29), hat w keine $\Im^f, fin$-spab.

Speziell dieses letzte Beispiel zeigt uns, daß unsere neue Logik $[m|=$ echt ausdrucksstärker als die Prädikatenlogik erster Stufe ist. Im nächsten Kapitel werden wir die modelltheoretischen Eigenschaften von $[m|=$ untersuchen, die zu diesem Gewinn an Ausdruckskraft führen.

Die Existenz von spabs hängt natürlich von dem zugrundeliegenden Domainoperator ab. Die beiden folgenden Lemmata setzen die spabs bezüglich verschiedener Domainoperatoren zueinander in Beziehung:

Lemma 22 *$\Im$ und $\Im'$ seien Domainoperatoren, so daß es für alle Standardsignaturen Σ einen Satz $w_\Sigma \in \mathcal{S}en(\Sigma, X)$ gibt mit mit*

$$\Im'_\Sigma = \{A \in \Im_\Sigma \mid A \models w_\Sigma\}$$

Falls m ein Modul mit Signatur (Σ_P, Σ_E) ist und $w \in \mathcal{S}en_m$, dann gilt für alle $r \in \mathcal{P}Sen_m$:

$$r \text{ ist eine } \Im', m\text{-spab von } v \quad \Leftrightarrow \quad r \wedge w_\Sigma \text{ ist eine } \Im, m\text{-spab von } v \wedge w_\Sigma$$

Beweis:

r ist eine $\Im', m$-spab von v
$\Leftrightarrow$ für alle $A \in \Im'_{\Sigma_P}$ gilt $A\ [m|= r \between v$
(nach Definition)
$\Leftrightarrow$ für alle $A \in \Im_{\Sigma_P}$ mit $A \models w_{\Sigma_P}$ gilt $A\ [m|= r \between v$
(nach Voraussetzung)
$\Leftrightarrow$ für alle $A \in \Im_{\Sigma_P}$ gilt $A\ [m|= w_{\Sigma_P} \supset (r \between v)$
$\Leftrightarrow$ für alle $A \in \Im_{\Sigma_P}$ gilt $A\ [m|= (r \wedge w_{\Sigma_P}) \between (v \wedge w_{\Sigma_P})$
(Aussagenlogik !)
$\Leftrightarrow$ $r \wedge w_{\Sigma_P}$ ist eine $\Im, m$-spab von $v \wedge w_{\Sigma_P}$
(nach Definition) □

Lemma 23 *$\Im$ und $\Im'$ seien Domainoperatoren, so daß $\Im'_\Sigma \subseteq \Im_\Sigma$ für alle Standardsignaturen Σ gilt. Dann ist für jedes Modul m und $w \in \mathcal{S}en_m$ jede $\Im, m$-spab von w auch eine $\Im', m$-spab von w.*

Beweis: Wenn $A\ [m|= v \between w$ für alle $A \in \Im_\Sigma$ gilt, dann gilt es natürlich auch für alle $A \in \Im'_\Sigma$. □

Wir haben den Domainoperator $\Im^n$ als trivial bezeichnet, diese Bemerkung wird jetzt durch das folgende Lemma untermauert:

Lemma 24 *Für alle Module m, Sätze $w \in \mathcal{S}en_m$ und $n \in \mathbb{N}$ gibt es eine $\Im^n, m$-spab von w.*

Beweis: Für jede Standardsignatur Σ und $n \in \mathbb{N}$ gibt es eine endliche Menge $V_{\Sigma,n} \subseteq \mathcal{S}en(\Sigma, X)$, die in dem folgenden Sinne eine Aufteilung von $\Im^n_\Sigma$ in Isomorphieklassen liefert:

1. Für jedes $A \in \Im^n_\Sigma$ gibt es ein $v \in V_{\Sigma,n}$ mit $A \models v$
2. Falls $A, B \models v \in V_{\Sigma,n}$, dann sind A und B isomorph.

Die Konstruktion von $V_{\Sigma,n}$ beruht auf der Tatsache, daß es für eine endliche Kardinalität nur endlich viele mögliche Funktionsgraphen in der Klasse der Algebren mit dieser Kardinalität gibt. Diese sind selbst durch Axiome der Logik erster Stufe beschreibbar (siehe auch [CK90]). Da die Gültigkeitsrelation $[m]\models$ invariant unter Isomorphismen von Algebren ist, erhalten wir als eine $\Im^n, m$-spab von w die Disjunktion aller $w_i \in W_{\Sigma,n}$, für die es ein $A \in \Im^n_\Sigma$ gibt mit $A \models w_i$ und $A\ [m]\models w$. □

Wir wollen hier einen wichtigen Spezialfall erwähnen. Das hier angeführte Resultat wird später (Korollar 6) bewiesen werden:

> Es sei m ein Modul mit $NF_m = \emptyset$ und $\Im$ ein Domainoperator. Dann gibt es für jeden Satz $w \in \mathcal{S}en_m$ eine $\Im, m$-spab. Darüber hinaus gilt sogar, daß die $\Im, m$-spab eines Satzes berechenbar ist.

Bezüglich des trivialen Modules 1_Σ ist die Existenz von spabs eine Trivialität:

Lemma 25 *Sei Σ eine Standardsignatur und $\Im$ ein Domainoperator. Dann ist für jeden Satz $w \in \mathcal{S}en_{1_\Sigma}$ w eine $\Im, 1_\Sigma$-spab von sich selbst.*

4.4 Ersetzungssysteme auf Formeln

Oft wollen wir durch eine Reihe von Transformationsschritten eine gegebene Formel in eine andere äquivalente Formel mit gewissen wünschenswerten Eigenschaften überführen. Ein solches Transformationsverfahren kann man natürlich in einer schematischen Programmiersprache ("pidgin-Algol") formulieren. Dies hat den Vorteil, daß die Formulierung des Verfahrens bereits sehr nahe an einer Programmierung in einer realen Programmiersprache liegt. Andererseits bietet eine solche Formulierung aber relativ wenige Einblicke in die logische Natur des Problems, denn mit der Angabe des Kontrollflusses des Programmes ist das Problemlösungsverfahren im Allgemeinen überspezifiziert. Wir folgen deshalb hier dem von Kowalski in [Kow79] vorgeschlagenem Prinzip

$$\text{Algorithm} = \text{Logik} + \text{Control}$$

und teilen das Problem in einen Logikteil und einen Kontrollteil auf. Der Logikteil besteht aus einer Menge von Prozeduren (im weitesten Sinne), die unter bestimmten Bedingungen aktiviert werden können. Die Bedingungen werden dabei so liberal wie möglich gefaßt, sie sollen hier nur eine partielle Korrektheit des Verfahrens garantieren. Der Kontrollteil sorgt durch eine geeignete Auswahlstrategie der zu aktivierenden Regeln für eine Terminierung und insbesondere auch für eine möglichst hohe Effizienz des Verfahrens. Diese Aufteilung erleichtert gegenüber

einem kompletten Programm das Verständnis und die Verifikation. Im logischen (siehe [Apt90] für eine Übersicht) und prädikativen Programmieren ([Bib75]) überläßt man den Kontrollteil sogar weitgehend dem Ausführungssystem der Programmiersprache.

In dieser Arbeit sind wir an Fragen der Effizienz nicht interessiert. Aus diesem Grunde werden wir den Kontrollteil nur soweit spezifizieren, wie dies für die Terminierung des Verfahrens notwendig ist. Zur abstrakten Beschreibung von Algorithmen in diesem Sinne haben sich *Regelsysteme* (siehe z. B. [CL88], [Der89] und [JK91]) bewährt.

Den logischen Teil definieren wir dabei als Abschluß $\rightarrow_R$ einer durch ein Regelsystem gegebenen Relation R unter Kontextanwendung. Um Berechnungen auf der Basis einer solchen Relation durchführen zu können, muß die Relation natürlich in einem gewissen Sinne berechenbar sein. In unserem Fall ist R durch elementare syntaktische Umformungen gegeben, so daß die Funktion $\phi: x \mapsto \{y \mid x \rightarrow_R y\}$ stets berechenbar sein wird. Für den Kontrollteil geben wir nur einige zusätzliche einschränkende Bedingungen an R an, die die Terminierung des Verfahrens garantieren sollen.

Um zu beweisen, daß eine solche Relation $\rightarrow$ einen korrekten Algorithmus zur Transformation von Formeln in äquivalente Formeln einer bestimmten Gestalt darstellt, muß folgendes bewiesen werden:

1. $\rightarrow_R$ ist noethersch.
2. Falls $w \rightarrow_R v$, dann sind w und v äquivalent
3. Jede $\rightarrow_R$-Normalform ist in der gewünschten Form

Dabei ist natürlich im Einzelfall der Begriff der Äquivalenz und die angestrebte Endform zu präzisieren. Insbesonderen sind wir an den beiden folgenden Formen von Äquivalenz interessiert:

Definition 25 *Sei m ein Modul mit Signatur (Σ_P, Σ_E), $\Im$ ein Domainoperator und $\rightarrow$ eine binäre Relation auf $\mathcal{W}ff(\Sigma_E, X)$.*

1. *$\rightarrow$ heißt* korrekt, *falls für alle $w, v \in \mathcal{W}ff(\Sigma_E, X)$ mit $w \rightarrow v$ gilt: $\Im_{\Sigma_P}\ [m| = w \leftrightarrows v$*
2. *$\rightarrow$ heißt* korrekt für positive Belegungen, *falls für alle $w, v \in \mathcal{W}ff(\Sigma_E, X)$ mit $w \rightarrow v$, alle $A \in \Im_{\Sigma_P}$ und alle $\alpha \in ?^+_{[\![m]\!](A)}$ gilt:*

$$[\![m]\!](A)(w)(\alpha) = [\![m]\!](A)(v)(\alpha)$$

Wir wollen nun ein solches System angeben, daß aus einer gegebenen Formel einige für die Logik überflüssige Funktionszeichen eliminiert.

Lemma 26 *Es sei m ein Modul der Signatur (Σ_P, Σ_E) und*

$$\Sigma := (AS_m, EF_m \setminus (\{\mathtt{is}_k?, \mathtt{select}^j_k \mid k \in K_m\} \cup \{\mathtt{ifthenelse}_s, =_s \mid s \in S\}))$$

Regelsystem el:

$$
\begin{array}{lll}
(=_s) & (w(t =_s u)) & \perp\!\!\!\rightarrow \quad ((t = u \wedge t \neq \perp_s \wedge w(true)) \vee (t \neq u \wedge t \neq \perp_s \wedge w(false)) \vee \\
 & & \qquad (t = \perp_s \wedge u = \perp_s \wedge w(\perp_s))) \\
(if_s) & (w(\texttt{if } b \texttt{ then } t \texttt{ else } u)) & \\
 & & \perp\!\!\!\rightarrow \quad (b = true \wedge w(t)) \vee (b = false \wedge w(u)) \vee \\
 & & \qquad (b = \perp_{bool} \wedge w(\perp_{bool})) \\
(sel) & (w(\texttt{select}_k^j(t))) & \perp\!\!\!\rightarrow \quad (t = \perp \wedge w(\perp)) \vee \\
 & & \qquad (\exists x_1 \in s_1, \ldots, x_n \in s_n \,.\, t = k(x_1, \ldots, x_n) \wedge w(x_j)) \\
(is) & (w(\texttt{is}_k\texttt{?}(t)) & \perp\!\!\!\rightarrow \quad (t = \perp \wedge w(\perp)) \vee \\
 & & \qquad (\exists x_1 \in s_1, \ldots, x_n \in s_n \,.\, t = k(x_1, \ldots, x_n) \wedge w(true)) \vee \\
 & & \qquad \bigvee_{k' \in K_k} \exists x_1 \in s_1, \ldots, x_n \in s_n \,.\, t = k'(x_1, \ldots, x_n) \wedge w(false))
\end{array}
$$

Dabei ist für $k\colon s_1, \ldots, s_n \to s$ $K_k := \{k'\colon s'_1, \ldots, s'_m \to s \in K_m \mid k' \neq k\}$

Abbildung 10: Das Regelsystem *el* zur Eliminierung einiger Funktionssymbole

Dann gibt es eine totale berechenbare Funktion $\phi\colon \mathcal{S}en_m \to \mathcal{S}en(\Sigma, X)$*, so daß für alle* $w \in \mathcal{S}en_m$*:* $[m\models w \leftrightarrows \phi(x)$*.*

Beweis: Betrachte die Relation $\rightarrow_{el}$ gemäß Abbildung 10. Offenbar ist jede $\rightarrow_{el}$-Normalform ein Element von $\mathcal{S}en(\Sigma, X)$, und es gilt $[m\models w \leftrightarrows v$ falls $w \rightarrow_{el} v$.

Zum Beweis der Terminierung benutzen wir eine Multisetordnung: Für eine Formel w sei $\psi_1(w) \subseteq \mathcal{T}(\Sigma_E, X)$ der Multimenge der in w vorkommenden Terme. $\psi_2\colon \mathcal{T}(\Sigma_E, X) \to \mathbb{N}$ bilde jeden Term t auf die Anzahl der in t vorkommenden Funktionszeichen aus der Menge

$$\{\texttt{is}_k\texttt{?}, \texttt{select}_k^j \mid k \in K_m\} \cup \{\texttt{ifthenelse}_s, =_s \mid s \in S\}$$

ab. Man sieht leicht, daß $\psi_2(\psi_1(v)) <_{mul} \psi_2(\psi_1(w))$, falls $w \rightarrow_{el} v$. Da $\leq_{mul}$ eine noethersche Ordnung auf $\mathcal{M}(\mathbb{N})$ ist, ist somit auch $\rightarrow_{el}$ noethersch. □

4.5 Symbolische Auswertung von Funktionen

Die Funktionen *app* und *sum* aus Abbildung 5, Seite 39, verhalten sich wesentlich angenehmer als beispielsweise die Funktion *isstandard* in Abbildung 8, Seite 57. Im ersten Fall kann ein Term wie z. B. $sum(cons(x, nil))$ ohne Berücksichtigung der Parameteralgebra zu einem äquivalenten Term ohne neue Funktionszeichen, in diesem Fall $x + 0$, reduziert werden. Der Grund hierfür ist die Tatsache, daß die Kontrolle der Rekursion in dem Programm für *sum* nur Eigenschaften von Konstruktoren, Selektoren und Testfunktionen, die von der Parameteralgebra unabhängig sind, benutzt. Im Gegensatz dazu ist die Auswertung des Termes $isstandard(x)$ gemäß Abbildung 8 zu einem Term ohne Vorkommen des Funktionszeichen *isstandard* nicht

PAR **SORTS** $elem$

OPNS $0: \rightarrow elem$

$f: elem \rightarrow elem$

BODY **SORTS** $list$

CONS $nil: \rightarrow list$

$cons: elem, list \rightarrow list$

FCTS $F: list \rightarrow bool$

$ack: list, list \rightarrow list$

PROG

$$F(l) \Leftarrow \texttt{if is}_{nil}\texttt{?}(l)\ \texttt{then}\ true\ \texttt{else if select}^1_{cons}(l) = 0\ \texttt{then}\ true\ \texttt{else}\ F(cons(f(\texttt{select}^1_{cons}(l)), \texttt{select}^2_{cons}(l)))$$

$$ack(l_1, l_2) \Leftarrow \texttt{if is}_{nil}\texttt{?}(l_1)\ \texttt{then}\ cons(0, l_2)\ \texttt{else if is}_{nil}\texttt{?}(l_2)\ \texttt{then}\ ack(\texttt{select}^1_{cons}(l_1), cons(0, nil))\ \texttt{else}\ ack(\texttt{select}^1_{cons}(l_1), ack(l_1, \texttt{select}^2_{cons}(l_2)))$$

Abbildung 11: Beispiele zur symbolischen Auswertung von Funktionen.

ohne Kenntnis der Parameteralgebra möglich. Diese Unterscheidung wird durch den Begriff der symbolischen Auswertung präzisiert:

Definition 26 *Es sei m ein Modul der Signatur (Σ_P, Σ_E) und $f: s_1, \ldots, s_n \rightarrow s \in NF_m$. f heißt* symbolisch auswertbar, *wenn für alle Terme $t_i \in \mathcal{K}t_m(s_i)$, $i = 1, \ldots, n$, ein $j \in \mathbb{N}$ existiert mit*

$$\Im^{st}_{\Sigma_P}\ [m] \models \forall \mathcal{V}ar\langle f(t_1, \ldots, t_n)\rangle\ .\ \text{kleene}\langle j, f(t_1, \ldots, t_n)\rangle \neq \bot_s$$

Man beachte in obiger Definition, daß der Wert von j wohl von den Argumenttermen t_i, nicht aber von einer speziellen Parameteralgebra abhängt.

Wie man leicht sieht, sind die Funktion *app* und *sum* aus Abbildung 5 symbolisch auswertbar, nicht aber die Funktion *isstandard* aus Abbildung 8. *app* und *sum* kann man sogar in einem Sinne, den wir hier nicht weiter präzisieren wollen, als *primitiv rekursiv* bezeichnen. Aus Abbildung 11 erhält man weitere Beispiele: *ack* ist symbolisch auswertbar, kann aber auf keinen Fall als primitiv rekursiv bezeichnet werden (siehe [Her71]). Die Funktion F wiederum ist nicht symbolisch auswertbar.

Lemma 27 *Sei m ein Modul mit Signatur (Σ_P, Σ_E) und $f: s_1, \ldots, s_n \rightarrow s \in NF_m$.*

F ist symbolisch auswertbar genau dann, wenn für alle $A \in \Im^{st}_{\Sigma_P}$ $f^{[\![m]\!](A)}$ eine totale Funktion ist.

Beweis:

"$\Rightarrow$" Folgt direkt aus Lemma 11 und Lemma 12.

"$\Leftarrow$" Wir nehmen an, daß f nicht symbolisch auswertbar ist. Dann gibt es Terme $t_i \in \mathcal{K}t_m(s_i)$, so daß für alle $j \in \mathbb{N}$ ein $A \in \Im^{st}_{\Sigma_P}$ existiert mit

$$A\ [m]\models \exists \mathcal{V}ar\langle f(t_1,\ldots,t_n)\rangle\ .\ \mathrm{kleene}\langle j, f(t_1,\ldots,t_n)\rangle = \bot_s$$

Es sei nun $\mathcal{V}ar\langle f(t_1,\ldots,t_n)\rangle = \{x_1,\ldots,x_m\}$ und $\{c_1,\ldots,c_m\}$ eine Menge von neuen Konstantenzeichen, wobei c_i jeweils die gleiche Sorte wie x_i hat. Wähle $\Sigma' := \Sigma_P \cup \{c_1,\ldots,c_m\}$ und

$$T = \left\{ \left(\bigwedge_{l=1}^{m} c_l \neq \bot \right) \wedge (\mathrm{kleene}\langle j, f(t_1,\ldots,t_n)\rangle)^{c_1\ \ldots c_m}_{x_1\ \ldots x_m} = \bot_s \mid j \in \mathbb{N} \right\}$$

Nach Voraussetzung hat jede endliche Teilmenge von T ein Modell in $\Im^{st}_{\Sigma'}$, also hat auf Grund der Kompaktheit von $\Im^{st}$ auch T ein Modell $B \in \Im^{st}_{\Sigma'}$. Dann ist aber $f^{[\![m]\!](B\ |_{\Sigma_P})}$ nicht total. □

Später (Satz 7) werden wir sehen, daß wir zu jedem Modul ein in einem einem gewissen Sinne äquivalentes Modul konstruieren können, in dem alle Funktionen symbolisch auswertbar sind.

5 Modelltheoretische Eigenschaften

Wir untersuchen nun die modelltheoretischen Eigenschaften unserer Logik $[m|=$. Zunächst zeigen wir in Abschnitt 5.1, daß auch für durch Domainoperatoren definierte Klassen von Algebren die wichtigen modelltheoretischen Sätze der Prädikatenloik erster Stufe gelten. Danach wenden wir uns der Logik $[m|=$ zu und untersuchen die beiden Eigenschaften $\downarrow$LST und Kompaktheit. In Abschnitt 1.3 haben wir bereits die Bedeutung dieser beiden Sätze dargelegt. In Abschnitt 5.2 zeigen wir, daß der $\downarrow$LST Satz für die Logik $[m|=$ gilt. In Abschnitt 5.3 untersuchen wir den Zusammenhang zwischen der Kompaktheit von $[m|=$ und der Existenz von m-schwächsten Parameterbedingungen. Wir werden dann in Abschnitt 5.4 zeigen, daß die Existenz von m-schwächsten Parameterbedingung eine unentscheidbare Eigenschaft der Module ist.

5.1 Modelltheoretische Eigenschaften der Domainoperatoren

In Abschnitt 2.7 haben wir einige wichtige modelltheoretische Eigenschaften der Prädikatenlogik erster Stufe angegeben. Diese Eigenschaften beziehen sich jeweils auf einen Modellbegriff, der *alle* Algebren einer gegebenen Signatur umfaßt. Wir sind hier jedoch an Einschränkungen der betrachteten Modellklasse, ausgedrückt durch verschieden Domainoperatoren, interessiert. Wir wollen nun untersuchen, welche modelltheoretischen Eigenschaften die Klassen von Algebren haben, die durch einen Domainoperator beschrieben werden. Zwei wesentliche Eigenschaften haben wir in der Definiton der Domainoperatoren festgeschrieben: Kompaktheit und Abschluß unter elementaren Teilmodellen. Die hier angegeben Korollare sind einfache Folgerungen aus diesen beiden grundlegenden Eigenschaften. Wir formulieren die Korollare hier nur für abzählbare oder endliche Signaturen, eine Verallgemeinerung auf Signaturen höherer Kardinalität ist aber leicht möglich.

Korollar 1 *Es sei $\Im$ ein Domainoperator und Σ eine abzählbare Standardsignatur. Wenn $T \subseteq \mathcal{S}en(\Sigma, X)$ in $\Im_\Sigma$ ein Modell unendlicher Kardinalität hat, dann hat T auch ein Modell in $\Im_\Sigma$ der Kardinalität $\aleph_0$.*

Beweis: Dies folgt direkt aus dem Abschluß von $\Im_\Sigma$ unter elementaren Teilmodellen. □

Das nächste Korollar drückt die aufwärts gerichtete Löwenheim-Skolem-Tarski Eigenschaft aus:

Korollar 2 *Es seien $\Im$ ein Domainoperator, Σ eine endliche Standardsignatur und α, β unendliche Kardinalzahlen mit $\beta \geq \alpha$. Falls $T \subseteq \mathcal{S}en(\Sigma, X)$ in $\Im_\Sigma$ ein Modell der Kardinalität α hat, dann hat T auch ein Modell B in $\Im_\Sigma$ mit $\#B \geq \beta$.*

Beweis: Da T ein Modell $A \in \Im_\Sigma$ mit $\#A = \alpha \geq \aleph_0$ hat, gibt es eine Sorte $s \in S$ mit $\#s^A = \alpha$. Zu einer Kardinalzahl $\beta \geq \alpha$ definieren wir nun eine zu F disjunkte Menge C_β von Konstanten

$$C_\beta := \{c_i : \rightarrow s \mid i \in \beta\}$$

und weiter

$$\begin{aligned} \Sigma' &:= \Sigma \cup C_\beta \\ T_\beta &:= \{c_i \neq c_j \mid i \neq j,\, i,j \in \beta\} \end{aligned}$$

Jede endliche Teilmenge N von $T \cup T_\beta$ enthält nur endlich viele Ungleichungen aus T_β, wir können daher mit Hilfe der Erweiterungseigenschaft von $\Im$ A zu einer Algebra $A_N \in \Im_{\Sigma'}$ mit $A_N \models N$ erweitern. Wegen der Kompaktheit von $\Im_{\Sigma'}$ hat dann auch $T \cup T_\beta$ ein Modell $B \in \Im_{\Sigma'}$, für daß dann $\#B \geq \beta$ gelten muß. Auf Grund der Erweiterungseigenschaft von $\Im$ ist dann $B \mid_\Sigma \in \Im_\Sigma$ ein Modell von T, und es gilt natürlich $\#B \mid_\Sigma = \#B \geq \beta$. □

Lemma 1 und Lemma 2 kann man nun zusammenfassen:

Korollar 3 *Es sei $\Im$ ein Domainoperator und Σ eine abzählbare Standardsignatur. Wenn eine Menge $T \subseteq \mathcal{S}en(\Sigma, X)$ ein unendliches Modell in $\Im_\Sigma$ hat, dann hat sie Modelle beliebiger unendlicher Kardinalität in $\Im_\Sigma$.*

Das nächste Korollar ist aus der Modelltheorie der Prädikatenlogik erster Stufe als *Łoś-Vaught Test* bekannt (siehe z. B. Proposition 3.1.7 in [CK90]).

Korollar 4 *Sei $\Im$ ein Domainoperator, Σ eine abzählbare Standardalgebra und $T \subseteq \mathcal{S}en(\Sigma, X)$ konsistent in $\Im_\Sigma$. Falls*

1. *alle Modelle von T in $\Im_\Sigma$ unendliche Kardinalität haben und*
2. *für eine geeignete unendliche Kardinalzahl α alle Modelle von T in $\Im_\Sigma$ der Kardinalität α isomorph sind,*

dann ist T vollständig in $\Im_\Sigma$.

Beweis: Es seien $A, B \in \Im_\Sigma$ Modelle von T, nach Voraussetzung (1) haben A und B also unendliche Kardinalität. Durch Anwendung von Korollar 3 auf $Th(A)$, bzw. $Th(B)$, erhalten wir Modelle $A', B' \in \Im_\Sigma$ mit Kardinalität α, so daß A und A', bzw. B und B' elementar äquivalent sind. Dann sind nach Voraussetzung (2) A' und B' isomorph, also elementar äquivalent, und damit sind auch A und B elementar äquivalent. Da alle Modelle von T in $\Im_\Sigma$ elementar äquivalent sind, ist T vollständig in $\Im_\Sigma$. □

5.2 Abwärts Löwenheim-Skolem-Tarski Eigenschaft

In diesem Abschnitt werden wir zeigen, daß $[m|=$ die abwärts gerichtete Löwenheim-Skolem-Tarski Eigenschaft, die wir im folgenden mit ↓LST abkürzen, hat. Ein analoges Resultat wurde in [Tiu81] für die "Logic of Effective Definitions" durch Übersetzung in die Logik $L\omega_1\omega$ ([Kei71]) bewiesen. Im Gegensatz dazu benutzen wir nur Eigenschaften elementarer Teilalgebren, so daß wir nicht auf Sätze der unendlichen Logik zurückgreifen müssen.

Bereits in Abschnitt 2.7 und in Korollar 1 haben wir gesehen, daß die ↓LST-Eigenschaft aus dem Abschluß der betrachteten Modellklasse unter elementaren Teilalgebren folgt. Leider verhält sich die Semantik von Modulen etwas problematisch zur elementaren Teilalgebrarelation: Während $[\![m]\!]$ die Teilalgebrarelation respektiert, d. h. $A \subseteq B \Rightarrow [\![m]\!](A) \subseteq [\![m]\!](B)$ (siehe Lemma 14), gilt dies im allgemeinen für die elementare Teilalgebrarelation nicht mehr:

Lemma 28 *Es sei sta das Modul aus Abbildung 8, Seite 57, mit Signatur* (Σ_P, Σ_E).
Es gibt Algebren $A, B \in Alg_{\Sigma_P}$ *mit* $A \preceq B$ *und* $[\![m]\!](A) \not\preceq [\![m]\!](B)$.

In dem Beweis benutzen wir einige elementare modelltheoretische Begriffe aus [CK90]:

Beweis: Es sei A die Standardisierung der Algebra N, definiert durch

$$\begin{aligned} elem^N &= \mathbb{N} \\ 0^N &= 0 \\ pred^N(x) &= \begin{cases} x \perp 1 & \text{falls } x \geq 1 \\ 0 & \text{falls } x = 0 \end{cases} \end{aligned}$$

Man sieht leicht, daß A ein atomares Modell ist (siehe [CK90], Seite 97). Betrachte nun $\Sigma' := \Sigma_P \cup \{c{:}\rightarrow elem\}$ und

$$T' := \{pred^i(c) \neq 0 \mid i \in \mathbb{N}\}$$

Jede endliche Teilmenge von $Th(A) \cup T'$ hat ein Modell in $\Im^f_{\Sigma'}$, also gibt es wegen der Kompaktheit von $\Im^f$ ein $B \in \Im^f_{\Sigma'}$ mit

$$B \models Th(A) \cup T'$$

also insbesondere $Th(A) = Th(B\mid_{\Sigma_P})$. Da A atomar ist, folgt dann aus Theorem 2.3.4 von [CK90]: $A \preceq B\mid_{\Sigma_P}$
Andererseits gilt aber

$$\begin{aligned} [\![sta]\!](A) &\models \forall x \in elem\,.\,isstandard(x) = true \\ [\![sta]\!](B\mid_{\Sigma_P}) &\models \exists x \in elem\,.\,isstandard(x) = \perp \end{aligned}$$

also $Th([\![sta]\!](A)) \neq Th([\![sta]\!](B\mid_{\Sigma_P}))$, und somit $[\![sta]\!](A) \not\preceq [\![sta]\!](B\mid_{\Sigma_P})$. □

Um ↓LST für unsere neue Logik zu beweisen, können wir allerdings elementare Teilalgebren der Exportalgebren benutzen:

Satz 1 *Es sei m ein Modul mit Signatur* (Σ_P, Σ_E), $\Im$ *ein Domainoperator und* $T \subseteq \mathcal{S}en_m$. *Falls es ein* $A \in \Im_{\Sigma_P}$ *gibt mit* $A\ [m\models T$ *und* $\#A \geq \aleph_0$, *dann gibt es auch ein* $B \in \Im_{\Sigma_P}$ *mit* $B\ [m\models T$ *und* $\#B = \aleph_0$.

Beweis: O.B.d.A. sei $\Sigma_P \subseteq \Sigma_E$. Nach dem verschärften abwärts gerichteten Löwenheim-Skolem-Tarski Satz (Seite 29) gibt es ein $B' \preceq [\![m]\!](A)$ mit $\#B' = \aleph_0$. Wähle $B := B'\mid_{\Sigma_P}$, nach Lemma 15 ist $B' = [\![m]\!](B)$. Da B' eine elementare Teilalgebra von $[\![m]\!](A)$ ist, ist

auch B eine elementare Teilalgebra von A. Wegen des Abschlusses von $\Im_{\Sigma_P}$ unter elementaren Teilalgebren ist dann $B \in \Im_{\Sigma_P}$. Schließlich hat, da $B \preceq A$, B unendliche Kardinalität, also $\#B = \aleph_0$ □

Korollar 5 *Es sei m ein Modul mit Signatur* (Σ_P, Σ_E) *und* $A \in \Im_{\Sigma_P}$ *mit unendlicher Kardinalität. Dann gibt es ein* $B \in \Im_{\Sigma_P}$ *mit* $\#B = \aleph_0$ *und* $Th_m(A) = Th_m(B)$.

Beweis: Folgt sofort aus Satz 1. □

5.3 Kompaktheit

Aus der Modelltheorie ist bekannt, daß die Anwendungen des Kompaktheitssatzes fast immer die Verwendung zusätzlicher Konstantenzeichen verlangen. So mußten wir in dem Beweis des Korollares auf Seite 29 neue Konstantenzeichen verwenden, um die Träger der zu konstruierenden Algebra bezeichnen zu können. In einem gewissen Sinne ermöglicht der Kompaktheitssatz auf diese Art und Weise die Konstruktion eines Modelles, das eine unendliche Konjunktion von Formeln erfüllt. Durch Verwendung neuer Konstantenzeichen kann man eine Existenzquantifizierung dieser unendlichen Konjunktion simulieren (vergleiche hierzu auch Proposition 2.2.7 in [CK90]).

In der Prädikatenlogik erster Stufe stellt die Einführung neuer Konstantenzeichen keine Schwierigkeit dar, da die modelltheoretischen Eigenschaften unabhängig von der zugrundeliegenden Sprache (d. h. in unserem Falle der Signatur) betrachtet werden. Jetzt sind wir aber an den modelltheoretischen Eigenschaften *einzelner* Module mit festgelegter Signatur interessiert. Wir müssen deshalb explizit eine Möglichkeit, neue Konstanten einzuführen, einräumen.

Definition 27 *Es sei m ein Modul mit Signatur* (Σ_P, Σ_E) *und* $\Im$ *ein Domainoperator. m ist* $\Im$-kompakt, *falls für alle* $W \subseteq \mathcal{S}en_m$ *gilt:*

> *Wenn es zu jeder endlichen Teilmenge* $F \subseteq W$ *ein* $A \in \Im_{\Sigma_P}$ *gibt mit* $A\ [m|\!=\ F$, *dann gibt es auch ein* $B \in \Im_{\Sigma_P}$ *mit* $B\ [m|\!=\ W$.

Satz 2 *Es sei m ein Modul mit Signatur* (Σ_P, Σ_E) *und* $\Im$ *ein Domainoperaor. Die folgenden Aussagen sind äquivalent:*

1. *Jede Erweiterung von m um eine endliche Menge von Konstanten ist* $\Im$*-kompakt.*
2. *Für jede Erweiterung* m' *von m um eine endliche Menge von Konstanten gibt es zu jedem Satz in* $\mathcal{S}en_{m'}$ *eine* $\Im, m'$*-spab.*

Beweis:

(1) $\Longleftarrow$ (2)

Es sei m' eine Erweiterung von m um eine endliche Menge von Konstanten, (Σ'_P, Σ'_E) die Signatur von m' und $W \subseteq \mathcal{S}en_{m'}$. Für jedes $w \in W$ bezeichne $\bar{w} \in \mathcal{P}\mathcal{S}en_{m'}$ eine $\Im, m'$-spab von w, und $\bar{V} := \{\bar{v} \mid v \in V\}$ für $V \subseteq W$.

Es gebe nun für jede endliche Teilmenge $F \subseteq W$ ein $A \in \Im_{\Sigma'_P}$ mit $A\ [m|\!=\ F$, also auch $A \models \bar{F}$. Wegen der Kompaktheit von $\Im$ gibt es dann ein $B \in \Im_{\Sigma'_P}$ mit $B \models \bar{W}$, also $B\ [m|\!=\ W$.

(1) $\Longrightarrow$ (2)

Es sei jede Erweiterung von m um eine endliche Menge von Konstanten $\Im$-kompakt. Wir definieren W als die Menge aller Formeln über den Exportsignaturen der Ereiterungen von m. Um mengentheoretischen Schwierigkeiten aus dem Weg zu gehen, kann man hierbei annehmen, daß alle Konstantenzeichen aus einer festen Grundmenge gewählt werden.

$$W := \bigcup_{\substack{m' \text{erweitert } m \\ \text{um endliches C}}} \mathcal{S}en_{m'}$$

Für $w \in W$ definieren wir

- $\phi_1(w)$ als die Anzahl von Vorkommen von Existenzquantoren über neuer Sorte in w und
- $\phi_2(w)$ als die Summe der Anzahl der Vorkommen von Existenzquantoren über Parametersorte, der Anzahl der Vorkommen von $\neg$ und der Anzahl der Vorkommen von $\wedge$ in w.
- und schließlich $\phi(w) := (\phi_1(w), \phi_2(w))$.

Wir nehmen nun an, daß es eine Erweiterung m' und einen Satz $w \in \mathcal{S}en_{m'}$ gibt, so daß w keine $\Im, m'$-spab hat. Da die Ordnung $\leq^2_{lex}$ eine noethersche Ordnung auf $\mathbb{N} \times \mathbb{N}$ ist, können wir sogar annehmen, daß es kein w' mit dieser Eigenschaft und $\phi(w') <^2_{lex} \phi(w)$ gibt. Wir werden diese Annahme nun zum Widerspruch führen.

Zunächst einmal bemerken wir, daß die Existenz einer spab zu einem Satz w nicht davon abhängt, welche Konstanten zusätzlich zu denen von w und m noch in dem Modul vorhanden sind:

> Falls m', m'' Erweiterungen von m sind und $w \in \mathcal{S}en_{m'} \cap \mathcal{S}en_{m''}$, dann hat w eine $\Im, m'$-spab genau dann, wenn w eine $\Im, m''$-spab hat.

Wir sagen dann "$w \in W$ hat eine $\Im, m$-spab", falls w eine $\Im, m_w$-spab hat, wobei m_w die Erweiterung von m um die kleinste Menge C mit $w \in \mathcal{S}en(\Sigma_E \cup C, X)$ ist.

Wir machen nun eine Fallunterscheidung nach der Struktur von w. Σ sei die Parametersignatur von m_w.

a) $w = \exists x : s \,.\, v$ mit $s \in PS_m$

Es sei c ein Konstantensymbol, das weder in m noch in w vorkommt. Auf Grund der Minimalitätsannahme an w hat v_x^c dann eine $\Im, m$-spab r. In Abwandlung unserer Notation für Substitutionen bezeichnen wir mit r_c^y die Formel, die wir aus r erhalten, indem wir alle Vorkommen von c durch die Variable y ersetzen. Wir zeigen nun, daß $\exists y : s \,.\, r_c^y$, wobei $y \notin \mathcal{V}ar\langle r \rangle$, eine $\Im, m$-spab von w ist.

Es sei m' die Erweiterung von m_w um $\{c : \rightarrow s\}$. Für eine Σ-Algebra A und $a \in s^A$ bezeichne (A, a) die $\Sigma \cup \{c\}$-Algbra mit $(A, a)\,|_\Sigma = A$ und $c^{(A,a)} = a$. Dann gilt für alle $A \in \Im_\Sigma$:

$A \models \exists y : s \,.\, r_c^y$
$\Leftrightarrow$ es gibt ein $a \in s^A$ mit $(A, a) \in \Im_{\Sigma \cup \{c\}}$ und $(A, a) \models r$
(wegen der Erweiterungseigenschaft von $\Im$)
$\Leftrightarrow$ es gibt ein $a \in s^A$ mit $(A, a) \in \Im_{\Sigma \cup \{c\}}$ und $(A, a)\,[m' \models v_x^c$
(da r eine $\Im, m$-spab von v_x^c ist)
$\Leftrightarrow$ $A\,[m_w \models \exists x : s \,.\, v$
(wegen der Erweiterungseigenschaft von $\Im$)
$\Leftrightarrow$ $A\,[m_w \models w$

b) $w = \exists x : s \,.\, v$ mit $s \in NS_m$

Auf Grund der Minimalitätsannahme an w hat für jedes $t \in \mathcal{K}t_m(s) \cup \{\bot_s\}$ der Satz $\exists \mathcal{V}ar\langle t \rangle \,.\, v_x^t$ eine $\Im, m$-schwächste Parameterbedingung r_t. Man beachte hierzu, daß alle Variablen in $\mathcal{V}ar\langle t \rangle$ von Parametersorte sind. Nach Lemma 11 ist dann für jede endliche Teilmenge $F \subseteq \mathcal{K}t_m(s) \cup \{\bot_s\}$ der Satz $\bigvee_{t \in F} r_t$ eine $\Im, m$-Parameterbedingung von w. Nach Annahme kann dies aber keine $\Im, m$-*schwächste* Parameterbedingung von w sein, also gibt es für jede endliche Teilmenge $F \subseteq \mathcal{K}t_m(s) \cup \{\bot_s\}$ ein $A \in \Im_\Sigma$ mit

$$A\,[m_w \models \{\exists x : s \,.\, v\} \cup \{\neg r_t \mid t \in F\}$$

Auf Grund der $\Im$-Kompaktheit von m_w gibt es dann auch ein $A \in \Im_\Sigma$ mit

$$A\,[m_w \models \{\exists x : s \,.\, v\} \cup \{\neg r_t \mid t \in \mathcal{K}t_m(s)\}$$

also im Widerspruch zu Lemma 11:

$$A\,[m_w \models \{\exists x : s \,.\, v\} \cup \{\forall \mathcal{V}ar\langle t \rangle \,.\, \neg v_x^t \mid t \in \mathcal{K}t_m(s) \cup \{\bot_s\}\}$$

c) $w = \neg v$

Wegen der Minimalität von w hat v eine $\Im, m$-spab r. Nach Lemma 21 ist dann aber $\neg r$ eine $\Im, m$-spab von w.

d) $w = v_1 \vee v_2$

Wegen der Minimalität von w haben dann auch v_1 und v_2 jeweils ein $\Im, m$-spab r_1, bzw. r_2. Nach Lemma 21 ist dann aber $r_1 \vee r_2$ eine $\Im, m$-spab von w.

e) $w = (t_1 = t_2)$

Wir können dann w auch in der Form

$$\underbrace{(t_1 = t_2 \wedge t_1 \neq \bot)}_{v_1} \vee \neg (\underbrace{t_1 \neq \bot}_{v2} \vee \underbrace{t_2 \neq \bot}_{v_3})$$

schreiben. Wie in den Fällen (c) und (d) können wir auch hier folgern, daß mindestens eine der drei Formeln v_1, v_2 und v_3 keine $\Im, m$-spab hat. Wir nehmen an, daß v_1 keine $\Im, m$-spab hat, der Beweis für die beiden anderen Fälle verläuft analog.

Wie wir in Korollar 6 zeigen werden, hat für jedes $n \in \mathbb{N}$ der Satz

$$\text{kleene}\langle n, t_1\rangle = \text{kleene}\langle n, t_2\rangle \wedge \text{kleene}\langle n, t_1\rangle \neq \bot$$

eine $\Im, m$-spab r_n. Damit ist nach Lemma 12 für jede endliche Teilmenge $F \subseteq \mathbb{N}$ der Satz $\bigvee_{n\in F} r_n$ eine $\Im, m$-Parameterbedingung von w, aber nach Annahme keine $\Im, m$-*schwächste* Parameterbedingung von w. Also gibt es für jede endliche Teilmenge $F \subseteq \mathbb{N}$ ein $A \in \Im_\Sigma$ mit

$$A\ [m|\!=\ \{v_1\} \cup \{\neg r_n \mid n \in F\}$$

also gibt es auf Grund der $\Im$-Kompaktheit von m_w auch ein $B \in \Im_\Sigma$ mit

$$B\ [m|\!=\ \{v_1\} \cup \{\neg r_n \mid n \in \mathbb{N}\}$$

d. h. nach Konstruktion:

$$B\ [m|\!=\ \{t_1 = t_2 \wedge t_1 \neq \bot\} \cup \{\text{kleene}\langle n, t_1\rangle \neq \text{kleene}\langle n, t_2\rangle \vee \text{kleene}\langle n, t_1\rangle = \bot \mid n \in \mathbb{N}\}$$

Dies widerspricht aber Lemma 12. □

Man beachte, wie in diesem Beweis die Iteration der Trägermenger und die der rekursiven Funktionen in ganz analoger Weise eingesetzt wurden.

Der obige Satz zeigt, daß die Nichtexistenz einer spab stets mit der Nicht-Kompaktheit der Logik $[m|\!=$ einhergeht. Aus diesem Grunde werden immer, um die Nichtexistenz einer spab zu beweisen, den Kompaktheitssatz benutzen.

5.4 Unentscheidbarkeit der Existenz von spabs

Wir zeigen nun, daß die Existenz einer spab unentscheidbar ist. Genauer sind, für "interessante" Domainoperatoren $\Im$, folgende Fragen unentscheidbar:

- Gibt es zu einem Modul m und einem Satz $w \in \mathcal{S}en_m$ eine $\Im, m$-spab ?
- Hat ein Modul m die Eigenschaft, daß es zu jedem Satz $w \in \mathcal{S}en_m$ eine $\Im, m$-spab gibt?

Wir benutzen ein Unentscheidbarkeitsresultat über Zweikopfautomaten, daß sich im Zusammenhang mit ähnlichen Fragen bei Programmschemata als nützlich herausgestellt hat. Die Verwendung von Zweikopfautomaten liegt hier nahe, da sie keine speziellen Datentypen außer Bitfolgen (diese können durch bool-wertige Funktionen simuliert werden) und Zuständen (diese werden direkt in dem Modul kodiert) benötigen.

Wir geben zunächst die Definition von Zweikopfautomaten und das zugehörige Unentscheidbarkeitsresultat an, eine detailierte Darstellung findet man in [LPP70] und [Gre75]. Hier beschränken wir uns gegenüber der Originalliteratur auf ein fest gewähltes binäres Alphabet $\{0,1\}$.

Definition 28 ([LPP70]) *Ein* Zweikopfautomat *(abgekürzt ZKA) ist ein Tupel*

$$(Q_1, Q_2, q_0, q_a, q_r, \delta)$$

wobei

- Q_1 *and* Q_2 *endliche Mengen sind,*
- q_0, q_a *und* q_r *paarweise verschieden und jeweils weder in* Q_1 *noch in* Q_2 *enthalten sind,*
- *und* δ *eine Übergangsfunktion ist*

$$\delta : (Q_1 \cup Q_2 \cup \{q_0\}) \times \{0,1\} \to Q_1 \cup Q_2 \cup \{q_a, q_r\}$$

Die Eingabe eines solchen Automaten ist eine *unendliche* Folge über $\{0,1\}$. Der ZKA arbeitet analog zum endlichen Automaten, besitzt aber nun zwei Leseköpfe, die sich unabhängig voneinander über das gleiche Eingabeband bewegen. Der Zustand des Automaten wird durch die Übergangsfunktion aus dem alten Zustand (oder dem Startzustand im ersten Berechnungsschritt) und der von einem Kopf gelesenen Eingabe berechnet. Die Auswahl des Kopfes geschieht dabei über die Zustände: falls der alte Zustand in Q_1 liegt (und ebenso beim ersten Berechnungsschritt) wird die Eingabe vom ersten Kopf gelesen, sonst vom zweiten. Ebenso bestimmt der neu berechnete Zustand, welcher der beiden Köpfe um ein Feld auf dem Band nach vorne bewegt wird. Im Gegensatz zum endlichen Automaten gibt es nun *drei* Möglichkeiten für das Gesamtverhalten eines ZKA auf einer Eingabe:

- Der Automat *akzeptiert* seine Eingabe, falls er den Zustand q_a erreicht.
- Der Automat *verwirft* seine Eingabe, falls er den Zustand q_r erreicht.
- Der Automat *divergiert* auf seiner Eingabe, falls er niemals die Zustände q_a oder q_r erreicht.

In den ersten beiden Fällen "sieht" der ZKA also nur ein endliches Anfangsstück des Eingabebandes. $\mathcal{L}_A$ ist die Menge der von A akzeptierten Eingaben und $\mathcal{D}_A$ ist die Menge der Eingaben, auf denen A divergiert. Das grundlegende Resultat über ZKAs, das wir hier benutzen wollen, ist das folgende:

PAR **SORTS** $tapeposition$
$value$

OPNS $start: \rightarrow tapeposition$
$next: tapeposition \rightarrow tapeposition$
$contents0: tapeposition \rightarrow bool$
$a: \rightarrow value$
$f: value \rightarrow value$
$test: value \rightarrow bool$

BODY **FCTS** $H: \rightarrow bool$
$F_q: tapeposition, tapeposition, value \rightarrow bool$
für alle Zustände q des Automaten

PROG $H \Leftarrow F_{q_0}(start, start, a)$
$F_q(p_1, p_2, x) \Leftarrow$ `if` $contents0(p_1)$
`then` $F_{\delta(q,0)}(next(p_1), p_2, x)$
`else` $F_{\delta(q,1)}(next(p_1), p_2, x)$
für alle Zustände $q \in Q_1$
$F_q(p_1, p_2, x) \Leftarrow$ `if` $contents0(p_2)$
`then` $F_{\delta(q,0)}(p_1, next(p_2), x)$
`else` $F_{\delta(q,1)}(p_1, next(p_2), x)$
für alle Zustände $q \in Q_2$
$F_r(p_1, p_2, x) \Leftarrow F_r(p_1, p_2, x)$
$F_a(p_1, p_2, x) \Leftarrow$ `if` $test(x)$
`then` $true$
`else` $F_{q_0}(start, start, f(x))$

EXPORT $H: \rightarrow bool$

Σ_P bezeichen die Parametersignatur von sim_A.

Abbildung 12: Das Modul sim_A zur Simulation eines ZKA A für Satz 3.

Lemma 29 ([LPP70]) *Die beiden folgenden Mengen sind nicht semientscheidbar:*

- *Die Menge aller ZKAs A, so daß $\mathcal{L}_A = \emptyset$.*
- *Die Menge aller ZKAs A, so daß $\mathcal{D}_A \neq \emptyset$.*

Das Modul sim_A in Abbildung 12 simuliert einen ZKA A. Um diese Simulation zu präzisieren, stellen wir zunächst einmal eine Verbindung zwischen den Eingabebändern eines ZKA A und den Parameteralgebren des Modules sim_A aus Abbildung 12 her: Zu einer Algebra $B \in Alg_{\Sigma_P}$

ist die unendliche Folge $tape_B\colon \mathbb{N} \to \{0,1\}$ definiert durch:

$$tape_B(i) := \begin{cases} 0 & \text{falls } B(contents0(next^i(start))) = true \\ 1 & \text{falls } B(contents0(next^i(start))) = false \end{cases}$$

Wir können jedes Eingabeband t als ein $tape_B$ für eine geeignete Algebra $B \in Alg_{\Sigma_P}$ darstellen.

Lemma 30 *Es sei A ein ZKA und sim_A das gemäß Abbildung 12 zugehörige Modul. Dann gilt für alle $B \in Alg_{\Sigma_P}$ und $\alpha \in {?}_B$:*

$$\begin{aligned} tape_B \in \mathcal{L}_A \quad &\Rightarrow \quad [\![m]\!](B)(F_{q_0}(start, start, x))(\alpha) = \\ &\qquad [\![m]\!](B)(\texttt{if } test(x) \texttt{ then } true \texttt{ else } F_{q_0}(start, start, f(x)))(\alpha) \\ tape_B \notin \mathcal{L}_A \quad &\Rightarrow \quad [\![m]\!](B)(F_{q_0}(start, start, x))(\alpha) = \bot \end{aligned}$$

Beweis: folgt sofort aus den Definitionen. □

Lemma 31 *Es sei $\Im$ ein Domainoperator, A ein ZKA und sim_A das zugehörige Modul gemäß Abbildung 12. Falls $\mathcal{L}_A = \emptyset$, dann hat jedes $w \in \mathcal{S}en_{sim_A}$ eine $\Im, sim_A$-spab.*

Beweis: Da $\mathcal{L}_A = \emptyset$, gilt nach Lemma 30: $\Im_{\Sigma_P}\ [sim_A|\!=\ (H = \bot)$. Für $w \in \mathcal{S}en_{sim_A}$ sei $w_H^- \in \mathcal{P}Sen_{sim_A}$ die Formel, die wir erhalten, wenn wir in w jedes Vorkommen von H durch $\bot_{bool}$ ersetzen. Nach der obigen Bemerkung gilt dann $\Im_{\Sigma_P}\ [sim_A|\!=\ w \between w_H^-$, also ist w_H^- eine $\Im, m$-spab von w. □

Lemma 32 *Es sei $\Im$ ein reichhaltiger Domainoperator, A ein ZKA und sim_A das zugehörige Modul gemäß Abbildung 12. Falls $\mathcal{L}_A \neq \emptyset$, dann hat der Satz $(H \neq \bot) \in \mathcal{S}en_{sim_A}$ keine $\Im, sim_A$-spab.*

Beweis: Sei $t \in \mathcal{L}_A$ und n die letzte von A "gesehene" Position von t. Wir nehmen an, daß v eine $\Im, sim_A$-spab von $(H \neq \bot)$ ist.

Denn relevanten Teil von t (d. h. das Anfangsstück von T bis zur Position n) können wir nun durch ein endliches Gleichungssytem beschreiben:

$$e_t := \bigwedge_{i=0}^{n} \begin{cases} contents0(next^i(start)) = true & \text{falls } t(i) = 0 \\ contents0(next^i(start)) = false & \text{falls } t(i) = 1 \end{cases}$$

Es ist dann $H^{[\![sim_A]\!](B)} = H^{[\![sim_A]\!](C)}$ für alle $B, C \in Alg_{\Sigma_P}$ mit $B\ [sim_A|\!=\ e_t$ und $C\ [sim_A|\!=\ e_t$. Also folgt aus Lemma 30, daß für alle $B \in Alg_{\Sigma_P}$ mit $B \models e_t$ gilt:

$$B\ [sim_A|\!=\ H \neq \bot \ \Leftrightarrow\ \text{es gibt ein } i \text{ mit } B \models test(f^i(a)) = true \qquad (1)$$

Wegen der Reichhaltigkeit von $\Im$ hat jede Menge der Form

$$\{v, e_t\} \cup \{test(f^i(a)) = false \mid i \leq n_0\}$$

ein Modell in $\Im_{\Sigma_P}$, nämlich die Standardisierung der Algebra $\mathcal{T}(\Sigma_P, \emptyset)/E$, wobei

$$\begin{aligned} E &= \{e_t\} \\ &\cup \{test(f^i(a)) = false \mid i \leq n_0\} \\ &\cup \{test(f^{n_0+1}(x)) = true\} \end{aligned}$$

Auf Grund der Kompaktheit von $\Im$ gibt es dann auch ein $B \in \Im_{\Sigma_P}$ mit

$$B \models \{v, e_t\} \cup \{test(f^i(a)) = false \mid i \in \mathbb{N}\}$$

d. h., da v eine $\Im, sim_A$-spab von $(H \neq \bot)$ ist:

$$B \models e_t \text{ und } B\ [sim_A]\models H \neq \bot \text{ und für alle } i\text{: } B \models test(f^i(a)) = false$$

Dies ist ein Widerspruch zu (1). □

Damit erhalten wir unser erstes Unentscheidbarkeitsresultat:

Satz 3 *Für jeden reichhaltigen Domainoperator $\Im$ sind die folgenden Mengen nicht semientscheidbar:*

- *Die Menge aller Module m, für die jeder Satz $w \in \mathcal{S}en_m$ eine $\Im, m$-spab hat.*
- *Die Menge aller Paare (w, m), wobei m ein Modul ist und $w \in \mathcal{S}en_m$, für die w eine $\Im, m$-spab hat.*

Beweis: folgt mit Lemma 30 und Lemma 31 aus Lemma 29. □

Um zu zeigen, daß die Mengen von Satz 3 nicht kosemientscheidbar sind, benutzen wir ebenfalls eine Reduktion auf eine nicht semientscheidbare Eigenschaft von ZKAs. Das Modul in Abbildung 13 ist in einem gewissen Sinne eine duale Version des Moduls aus Abbildung 12. Jetzt lassen wir die Funktion H terminieren, falls die Eingabe akzeptiert oder verworfen wird. Eine unbeschränkte Folge von Aufrufen von *test* wird durchgeführt, falls der Automat auf seiner Eingabe divergiert. Der Beweis verläuft analog zu dem von Satz 3, deshalb geben wir nur die beiden Lemmata und den Satz an:

Lemma 33 *Es sei $\Im$ ein Domainoperator, A ein ZKA und $cosim_A$ das zugehörige Modul gemäß Abbildung 13. Falls $\mathcal{D}_A = \emptyset$, dann hat jedes $w \in \mathcal{S}en_{cosim_A}$ eine $\Im, cosim_A$-spab.*

Lemma 34 *Es sei $\Im$ ein reichhaltiger Domainoperator, A ein ZKA und $cosim_A$ das zugehörige Modul gemäß Abbildung 13. Falls $\mathcal{D}_A \neq \emptyset$, dann hat der Satz $(H \neq \bot) \in \mathcal{S}en_{cosim_A}$ keine $\Im, cosim_A$-spab.*

PAR **SORTS** $tapeposition$
$value$
OPNS $start: \rightarrow tapeposition$
$next: tapeposition \rightarrow tapeposition$
$contents0: tapeposition \rightarrow bool$
$a: \rightarrow value$
$f: value \rightarrow value$
$test: value \rightarrow bool$

BODY **FCTS** $H: \rightarrow bool$
$F_q: tapeposition, tapeposition, value \rightarrow bool$
für alle Zustände q des Automaten
PROG $H \Leftarrow F_{q_0}(start, start, a)$
$F_q(p_1, p_2, x) \Leftarrow$ `if` $test(x)$ `then` $true$
`else if` $contents0(p_1)$
`then` $F_{\delta(q,0)}(next(p_1), p_2, f(x))$
`else` $F_{\delta(q,1)}(next(p_1), p_2, f(x))$
für alle Zustände $q \in Q_1$
$F_q(p_1, p_2, x) \Leftarrow$ `if` $test(x)$ `then` $true$
`else if` $contents0(p_2)$
`then` $F_{\delta(q,0)}(p_1, next(p_2), f(x))$
`else` $F_{\delta(q,1)}(p_1, next(p_2), f(x))$
für alle Zustände $q \in Q_2$
$F_r(p_1, p_2, x) \Leftarrow true$
$F_a(p_1, p_2, x) \Leftarrow true$

EXPORT $H: \rightarrow bool$

Abbildung 13: Das Modul $cosim_A$ zur Simulation eines ZKA A für Satz 4.

Satz 4 *Für jeden reichhaltigen Domainoperator $\Im$ sind die folgenden Mengen nicht semientscheidbar:*

- *Die Menge aller Module m, für die es einen Satz $w \in \mathcal{S}en_m$ gibt, der keine $\Im, m$-spab hat.*
- *Die Menge aller Paare (w, m), wobei m ein Modul ist und $w \in \mathcal{S}en_m$, für die w keine $\Im, m$-spab hat.*

PAR	**SORTS**	$\bot\,\bot\,\bot$
	OPNS	$\bot\,\bot\,\bot$
BODY	**SORTS**	nat
	CONS	$0\colon \rightarrow nat$
		$succ\colon nat \rightarrow nat$
	FCTS	$+\colon nat, nat \rightarrow nat$
		$*\colon nat, nat \rightarrow nat$
		$\leq\colon nat, nat \rightarrow bool$
	PROG	$x + y \Leftarrow$ `if` $\mathtt{is}_0?(x)$ `then` y
		`else` $succ(\mathtt{select}^1_{succ}(x) + y)$
		$x * y \Leftarrow$ `if` $\mathtt{is}_0?(x)$ `then` 0
		`else` $y + (\mathtt{select}^1_{succ}(x) * y)$
		$x \leq y \Leftarrow$ `if` $\mathtt{is}_0?(x)$ `then` $true$
		`else` `if` $\mathtt{is}_0?(y)$ `then` $false$
		`else` $\mathtt{select}^1_{succ}(x) \leq \mathtt{select}^1_{succ}(y)$

Abbildung 14: Das Modul *nat* definiert die natürlichen Zahlen.

6 Berechnung von spabs

Wir wollen uns nun mit Berechnungsverfahren für spabs beschäftigen. Selbst wenn für ein Modul m und einen Domainoperator $\Im$ eine $\Im, m$-spab zu jedem Satz in $\mathcal{S}en_m$ existiert, so heißt dies noch lange nicht, daß diese auch berechenbar ist, wie der folgende Fall zeigt:

Lemma 35 *nat sei das Modul aus Abbildung 14, Seite 76.*

1. *Jeder Satz $w \in \mathcal{S}en_m$ hat eine $\Im^f, nat$-spab.*
2. *Es gibt keine totale berechenbare Funktion $\phi\colon \mathcal{S}en_m \rightarrow \mathcal{P}\mathcal{S}en_m$, so daß für alle $w \in \mathcal{S}en_m$ $\phi(w)$ eine $\Im^f, nat$-spab von w ist.*

Beweis: Die Parametersignatur von *nat* ist die Signatur Σ_{BOOL}, die einzige Parameteralgebra von *nat* ist also $BOOL$. Somit gilt für jeden Satz $w \in \mathcal{S}en_{nat}$ entweder $[nat] \models w$ oder $[nat] \models \neg w$, und damit ist für jeden Satz $w \in \mathcal{S}en_{nat}$ entweder `True` oder `False` eine $\Im^f, nat$-spab.

B ist eine endliche Algebra, $Th(B)$ ist also entscheidbar. Falls es nun eine totale berechenbare Funktion gäbe, die zu jedem $w \in \mathcal{S}en_m$ eine $\Im^f, nat$-spab berechnet, dann könnte man also auch zu jedem $w \in \mathcal{S}en_m$ entscheiden, ob `True` oder `False` eine $\Im^f, nat$-spab von w ist. Dies steht im Widerspruch zur Unentscheidbarkeit der vollständigen Zahlentheorie ([End72]). □

Es gibt sogar für jede Erweiterung m' von *nat* um endliche Konstanten zu jedem Satz $w \in \mathcal{S}en_{m'}$ eine $\Im^f, m'$-spab, da die Konstanten jeweils nur Werte aus dem endlichen Bereich $\{true, false, \bot_{bool}\}$ annehmen können. Nach Satz 2 ist also jede endliche Erweiterung von *nat* $\Im^f$-kompakt.

6.1 Berechenbarkeit von spabs

In diesem Abschnitt wollen wir eine Beziehung zwischen der Berechenbarkeit von spabs einerseits und der Entscheidbarkeit von $Th_m(A)$ relativ zu $Th(A)$ andererseits herstellen. Intuitiv heißt dabei $Th_m(A)$ entscheidbar relativ zu $Th(A)$, falls es einen Algorithmus gibt, der zu jeder Parameteralgebra A $Th_m(A)$ entscheiden kann, wobei der Algorithmus das Wissen um $Th(A)$ zur Verfügung hat. Eine mögliche Formalisierung dieses relativen Entscheidbarkeitsbegriffes ist die Erweiterung des Konzeptes der Turingmaschine zu einer *Orakelmaschine*. Eine Orakelmaschine wird auf einem Eingabeband gestartet und zusätzlich mit einer Menge X, dem Orakel, versehen. Die Maschine kann die Berechnungsschritte einer Turingmaschine ausführen, zusätzlich aber auch "Fragen" der Form "ist $x \in X$?" an sein Orakel stellen. Welche Fragen dabei an das Orakel gestellt werden, hängt vom Vorlauf der Berechnung für ein gegebenes Eingabeband und von den bisher vom Orakel gegebenen Antworten ab und kann nicht vor Beginn der Berechnung bestimmt werden (eine Vorausberechnung der an das Orakel zu stellenden Fragen führt zu einem eingeschränkten Begriff von relativer Berechenbarkeit). Eine ausführliche Darstellung findet man in [Rog87].

Hier benutzen wir ein auf unsere Erfordernisse angepaßtes Maschinenmodell: Zu einem Modul m betrachten wir Orakelmaschinen, deren

- Eingabebereich die Menge $\mathcal{S}en_m$ aller Exportformeln,
- Ausgabebereich die Menge $\{\texttt{True}, \texttt{False}\}$,
- und deren Orakel jeweils eine Teilmenge von $\mathcal{P}Sen_m$ ist.

Die Semantik einer solchen Maschine T ist eine Funktion

$$\varphi^T : \mathcal{P}(\mathcal{P}Sen_m) \to (\mathcal{S}en_m \rightsquigarrow \{\texttt{True}, \texttt{False}\})$$

definiert durch

$$\varphi^T(X)(w) \begin{cases} = \texttt{True} & \text{falls } T \text{ mit Eingabe } w \text{ und Orakel } X \\ & \quad \text{mit dem Ausgabewert } \texttt{True} \text{ terminiert} \\ = \texttt{False} & \text{falls } T \text{ mit Eingabe } w \text{ und Orakel } X \\ & \quad \text{mit dem Ausgabewert } \texttt{False} \text{ terminiert} \\ \text{ist undefiniert} & \text{falls } T \text{ mit Eingabe } w \text{ und Orakel } X \text{ nicht terminiert} \end{cases}$$

In Anlehnung an [Rog87] definieren wir das *Diagramm* der Berechnung von T mit Eingabe w als einen (möglicherweise unendlichen) Baum, der *alle* Berechnungen von T mit Eingabe w und beliebigem Orakel repräsentiert. Jeder Knoten n des Baumes ist mit einer Konfiguration $konf(n)$ der Maschine T und einem Satz $bed(n) \in \mathcal{P}Sen_m$ markiert. Der Baum ist induktiv definiert durch

1. Es gibt einen Knoten *start* mit $bed(start) = \texttt{True}$ und $konf(start)$ ist die Startkonfiguration von T bei Eingabe w.

2. Falls n ein Knoten ist, so daß $konf(n)$ keine Endkonfiguration ist und im nächsten Berechnungsschritt das Orakel nicht befragt wird, dann gibt es auch einen Knoten n' mit $bed(n') = bed(n)$, $konf(n')$ ist die Nachfolgekonfiguration von $konf(n)$, und eine Kante von n zu n'.

3. Es sei n ein Knoten, so daß T in Konfiguration $konf(n)$ an das Orakel die Frage x stellt. Dann gibt es zwei Knoten n' und n'' mit $bed(n') = bed(n) \wedge x$, $bed(n'') = bed(n) \wedge \neg x$, $konf(n')$ (bzw. $konf(n'')$) ist die Nachfolgekonfiguration von $konf(n)$ bei positiver (bzw. negativer) Antwort des Orakels, und Kanten von n zu n' und n''.

Wir können jetzt den Satz über den Zusammenhang von relativer Entscheidbarkeit von Th_m und der Berechenbarkeit von spabs formulieren:

Satz 5 *Es sei m ein Modul mit Signatur (Σ_P, Σ_E) und $W \subseteq \mathcal{S}en_m$. $\Im$ sei ein Domainoperator, so daß die Menge der in $\Im_{\Sigma_P}$ konsistenten Sätze kosemientscheidbar ist. Die beiden folgenden Aussagen sind äquivalent:*

1. *Es gibt eine partielle berechenbare Funktion $p: \mathcal{S}en_m \rightsquigarrow \mathcal{P}Sen_m$, so daß für alle $w \in W$:*
   - *$p(w)$ definiert ist und*
   - *$p(w)$ eine $\Im, m$-spab von w ist.*
2. *Es gibt eine Orakelmaschine T, so daß für alle $w \in W$ und alle $A \in \Im_{\Sigma_P}$:*
   - *$\varphi^T(Th(A))(w)$ definiert ist und*
   - *A $[m|\!=$ w genau dann gilt, wenn $\varphi^T(Th(A))(w) =$ `True`*

Dabei können wir aus einem Algorithmus zur Berechnung von p gemäß (1) effektiv die Orakelmaschine T gemäß (2) konstruieren, und umgekehrt.

Die Kosemientscheidbarkeit der in $\Im_{\Sigma_P}$ konsistenten Sätze ist auf jeden Fall gegeben, falls $\Im_{\Sigma_P}$ durch eine endliche Menge von Axiomen im Sinne von Lemma 17 beschrieben wird. Das folgt aus der Semientscheidbarkeit der Menge der in der Prädikatenlogik erster Stufe allgemeingültigen Formeln ([End72], [Rob65]). Insbesondere trifft sie auf die Domainopertaren $\Im^f$, $\Im^{strikt}$, $\Im^{st}$ usw. zu.

Beweis:

(1) → (2): Die Orakelmaschine T berechnet zur Eingabe w zunächst $p(w)$ und befragt dann ihr Orakel über den so erhaltenen Satz. Die algorithmische Beschreibung von T kann man also insbesondere effektiv aus der Beschreibung des Algorithmusses zur Berechnung von p konstruieren.

(2) → (1): Wir betrachten den Teilgraphen G des Diagramms der Berechnung von T mit Eingabe w, der von der Menge aller Knoten n, für die $bed(n)$ konsistent in $\Im_{\Sigma_P}$ ist, erzeugt wird. G ist dann ebenfalls ein Baum, da stets $w \wedge v$ inkonsistent in $\Im_{\Sigma_P}$ ist, falls w inkonsistent in $\Im_{\Sigma_P}$ ist. Wir zeigen zunächst, daß G ein *endlicher* Baum ist.

Wir nehmen an, G wäre ein unendlicher Baum. Nach Königs Unendlichkeits Lemma (Königs Infinity Lemma, [Mal79]) gibt es dann einen unendlichen Pfad p in G. Auf Grund der Konstruktion des Diagramms gibt es für jede endliche Menge $\{n_1, \ldots, n_i\}$ von Knoten in p (man beachte, daß p ein Pfad ist) ein $j \in \{1, \ldots, i\}$ mit $\models bed(n_j) \supset bed(n_l)$ für alle $l \in \{1, \ldots, i\}$. Also ist, da jedes $bed(n_j)$ konsistent in $\Im_{\Sigma_P}$ ist, auch jede endliche Teilmenge von

$$R := \{bed(n) \mid n \in p\}$$

konsistent in $\Im_{\Sigma_P}$. Auf Grund der Kompaktheit von $\Im_{\Sigma_P}$ gibt es dann auch ein $A \in \Im_{\Sigma_P}$ mit $A \models R$. Dann ist aber, da p unendlich ist, im Widerspruch zur Annahme $\varphi^T(Th(A))(w)$ nicht definiert.

Es sei g nun die Tiefe des endlichen Baumes G. Wir können eine $\Im, m$-spab nun folgendermaßen finden: Wir konstruieren zunächst bis zur Tiefe g das Diagramm der Berechnung von T mit Eingabe w. N^+ sei die Menge aller Blätter dieses Baumes, für die $konf(n)$ eine Endkonfiguration mit Ausgabewert `False` ist. Dann ist der Satz

$$\bigwedge_{n \in N^+} konf(n)$$

eine $\Im, m$-spab von w, denn nach Definition von G können keine Knoten, die mit einem Orakel der Form $Th(A)$ erreichbar sind, im Diagramm tiefer als g liegen.

Das Problem hierbei ist aber, daß wir zu gegebenem Eingabesatz w den Wert g *berechnen* müssen. Hierbei nutzen wir nun die Kosemientscheidbarkeit der in $\Im_{\Sigma_P}$ konsistenten Formeln aus. Der Algorithmus für p arbeitet dann folgendermaßen:

Starte die Konstruktion des Diagramms zur Berechnung von T mit Eingabe w. Die Entwicklung des Baumes verläuft dabei jeweils an allen Blättern in Parallele ("breadth-first-search"). An jedem neu konstruiertem Knoten n des Diagramms wird eine Turingmaschine zum Beweis der Inkonsistenz von $bed(n)$ gestartet. Alle diese Turingmaschinen arbeiten parallel zur Konstruktion des Baumes. Falls an einem Knoten die Inkonsistenz erkannt wurde, wird der gesamte Unterbaum dieses Knotens gelöscht, so daß an diesem Unterbaum auch keine Berechnungen mehr durchgeführt werden.

Für alle Knoten in Tiefe $g+1$ ist $bed(n)$ inkonsistent in $\Im_{\Sigma_P}$. Da es nur endlich viele Knoten in Tiefe $g+1$ gibt und da die Inkonsistenz in $\Im, m$ semientscheidbar ist, terminieren irgendwann sämtliche Turingmaschinen zum Nachweis der Inkonsistenz der Formeln $bed(n)$ in Tiefe $g+1$. Zu diesem Zeitpunkt haben wir ein "Anfangsstück" des Diagramms konstruiert, das von G subsumiert wird. Aus diesem können wir nun die $\Im, m$-spab wie gehabt extrahieren. □

6.2 Ein Berechnungsverfahren für einen Spezialfall

Wir geben nun für eine spezielle Klasse von Modulen einen Algorithmus zur Berechnung von spabs an. Damit unser Verfahren auf ein Modul anwendbar ist, müssen die Funktionsdefinitionen der Module bestimmte "angenehme" Eigenschaften haben. Die Abbildungen 15 und 16 enthalten Beispiele von Modulen, die diese Bedingungen erfüllen. Bei den beiden folgenden

PAR **SORTS** $elem$
OPNS $+: elem, elem \to elem$
BODY **SORTS** $list$
CONS $atom: elem \to list$
$cons: elem, list \to list$
FCTS $sum2: list \to elem$
PROG $sum2(l) \Leftarrow \texttt{if is}_{atom}\texttt{?}(l)\ \texttt{then select}^1_{atom}(l)$
$\texttt{else select}^1_{cons}(l) + sum2(\texttt{select}^2_{cons}(l))$

Abbildung 15: Das Modul $list2$.

PAR **SORTS** $elem$
OPNS $a: \to elem$
$b: \to elem$
$+: elem, elem \to elem$
BODY **SORTS** $list$
CONS $nila: \to list$
$nilb: \to list$
$cons: elem, list \to list$
FCTS $sum3: list \to elem$
PROG $sum3(l) \Leftarrow \texttt{if is}_{nila}\texttt{?}(l)\ \texttt{then}\ a$
$\texttt{else if is}_{nilb}\texttt{?}(l)\ \texttt{then}\ b$
$\texttt{else select}^1_{cons}(l) + sum3(\texttt{select}^2_{cons}(l))$

Abbildung 16: Das Modul $list3$.

Definitionen beachte man, daß sie sich nur auf einstellige Funktionen beziehen, deren Argumentsorte eine neue Sorte und deren Ergebnissorte eine Parametersorte ist. Die erste dieser Eigenschaften drückt aus, daß eine Funktion in einem gewissen Sinne primitiv rekursiv ist.

Definition 29 *Sei m ein Modul mit Signatur (Σ_P, Σ_E) und $f: s_1 \to s_2 \in NF_m$, wobei $s_1 \in NS_m$ und $s_2 \in PS_m$.*

f heißt symbolisch partiell auswertbar, *falls für alle $k: \sigma_1, \ldots, \sigma_n \to s_1 \in K_m$ und Variablen $x_i \in X_{\sigma_i}$ ein Term $t_k^{(x_1,\ldots,x_n)} \in \mathcal{T}(PF_m \cup \{f\}, \{x_1, \ldots, x_n\})$ existiert mit*

1. $[m] \models \forall x_1 : \sigma_1, \ldots, x_n : \sigma_n \,.\, f(k(x_1, \ldots, x_n)) = t_k^{(x_1,\ldots,x_n)}$ *und*
2. *falls $f(t') \trianglelefteq t_k^{(x_1,\ldots,x_n)}$, dann ist $t' \in \{x_1, \ldots, x_n\}$*

und falls außerdem $[m\models f(\bot_{s_1}) = \bot_{s_2}$.

Die Bedingung (2) bedeutet, daß die einzigen Vorkommen von f in $t_k^{(x_1,\ldots,x_n)}$ von der Form $f(x)$ sind, wobei $x = x_i$ für ein geeignetes i. Die Funktionen aus den Abbildungen 15 und 16 sind symbolisch partiell auswertbar, so ist z. B. in Abbildung 15:

$$\begin{aligned} t_{atom}^{(x)} &= x \\ t_{cons}^{(x,y)} &= x + sum(y) \end{aligned}$$

Die Funktion F aus Abbildung 11, Seite 62 hingegen ist nicht partiell symbolisch auswertbar.

Die zweite Eigenschaft hängt außer von dem Modul auch noch von dem betrachteten Domainoperator ab.

Definition 30 *Sei m ein Modul mit Signatur (Σ_P, Σ_E), $\Im$ ein Domainoperator und $f\colon s_1 \to s_2 \in NF_m$, wobei $s_1 \in NS_m$ und $s_2 \in PS_m$.*

Eine Menge $C \subseteq \mathcal{K}t_m(s_1)$ heißt vollständig *für f in $\Im$, falls C endlich ist und für alle $A \in \Im_{\Sigma_P}$ und $a \in s_1^{[\![m]\!](A)} \setminus \{\bot_{s_1}^{[\![m]\!](A)}\}$ ein $t \in C$ und eine Belegung $\alpha \in ?^+_{X,A}$ existiert, so daß*

$$[\![m]\!](A)(f(t))(\alpha) = f^{[\![m]\!](A)}(a)$$

Im Beispiel des Moduls *list*2 aus Abbildung 15 ist $\{atom(x)\}$ vollständig für *sum*2 in $\Im^f$, denn für jede Parameteralgebra A und $a \in list^A$ mit $a \neq \bot_{list}^A$ gilt für eine beliebige Belegung α mit $\alpha(x) = sum2^{[\![list2]\!](A)}(a)$:

$$\begin{aligned} [\![list2]\!](A)(sum2(atom(x)))(\alpha) &= sum2^{[\![list2]\!](A)}(atom^{[\![list2]\!](A)}(sum2^{[\![list2]\!](A)}(a))) \\ &= sum2^{[\![list2]\!](A)}(a) \end{aligned}$$

Für die Funktion *sum*3 aus Abbildung 16 gibt es hingegen keine in $\Im^f$ vollständige Menge, wie wir nun zeigen werden:

Σ_P bezeichne die Parametersignatur von *list*3. Wir nehmen an, es gäbe eine in $\Im^f$ vollständige Menge C für f. m sei die maximale Anzahl von Vorkommen von *cons* in einem Term aus C. Wir definieren

$$r := \underbrace{cons(a, cons(a, \ldots, cons(a, atom(a))\ldots))}_{m+1 \text{ mal } cons}$$

Betrachte nun die Algebra $A = \mathcal{T}(\Sigma_P, \emptyset)/\emptyset$. Es ist dann

$$sum3^{[\![list3]\!](A)}(r) = \underbrace{a + (a + \ldots + (a + a)\ldots)}_{m+1 \text{ mal } +} \tag{2}$$

aber für alle $t' \in C$ und $\alpha \in ?^+_{X,A}$ ist wegen der Maximalität von m

$$[\![list3]\!](A)(sum3(t'))(\alpha) = \underbrace{u_1 + (u_2 + \ldots + (u_k + c)\ldots)}_{\leq m \text{ mal } +} \tag{3}$$

für geeignete Terme u_i und $c \in \{a, b\}$, und die Terme (2) und (3) können in $\mathcal{T}(\Sigma_P, \emptyset)/\emptyset$ niemals gleich sein. Um eine vollständige Menge für $sum3$ zu bekommen, müssen wir uns auf Parameteralgebren beschränken, in denen Terme wie (2) und (3) unter Umständen zu identischen Werten ausgewertet werden können.

Sei beispielsweise

$$e := (\forall x, y \in elem\,.\, x+(y+a) = (x+y)+a) \wedge (\forall x, y, z \in elem\,.\, x+(y+(z+b)) = (z+y)+(x+b))$$

und der Domainoperator $\Im'$ definiert durch

$$\Im'_\Sigma = \begin{cases} \{A \in Alg_\Sigma \mid A \models e\} & \text{falls } \Sigma_P \subseteq \Sigma \\ Alg_{\Sigma_P} & \text{anderenfalls} \end{cases}$$

Eine vollständige Menge für $sum3$ in $\Im'$ ist nun beispielsweise

$$\{nila, cons(x, nila), nilb, cons(x, nilb), cons(x, cons(y, nilb))\}$$

Satz 6 *Sei $\Im$ ein Domainoperator, m ein Modul mit Signatur (Σ_P, Σ_E) und $NF_m = \{f : s_1 \to s_2\}$ mit $s_1 \in NS$ und $s_2 \in PS$. Falls*

- *f symbolisch partiell auswertbar ist und*
- *es eine vollständige Menge für f in $\Im$ gibt,*

dann gibt es eine totale berechenbare Funktion ϕ: $\mathcal{S}en_m \to \mathcal{PS}en_m$, so daß für alle w $\phi(w)$ eine $\Im, m$-spab von w ist.

Beweis: Nach Lemma 26 können wir uns auf Sätze w ohne Vorkommen von Funktionszeichen aus der Menge

$$\{\mathtt{is}_k\mathtt{?}, \mathtt{select}_k^j \mid k \in K_m\} \cup \{\mathtt{ifthenelse}_s, =_s \mid s \in NS_m\}$$

beschränken, das heißt alle in w vorkommenden Funktionszeichen sind

- entweder aus PF_m
- oder f
- oder Konstruktoren
- oder $\perp_s$ für ein $s \in NS_m$.

Wir werden nun, um die weiteren Berechnungen zu vereinfachen, einige weitere Transformationen der Formeln vornehmen. Zunächst spalten wir die Terme soweit auf, daß unsere Formeln nur noch Terme enthalten, die in einer der beiden folgenden Klassen liegen:

$$\mathcal{P} := \bigcup_{s \in PS_m} \mathcal{T}((AS_m, PF_m \cup \{f\}), X)_s$$

$$\mathcal{K} := \bigcup_{s \in NS_m} \mathcal{T}((AS_m, K_m \cup \{\perp_s \mid s \in NS_s\}), X)_s$$

Regelsystem K-P:

$$
\begin{array}{lrcl}
(K) & w(k(t)) & \perp\!\!\!\rightarrow & \exists x : s \,.\, w(k(x)) \wedge x = t \\
 & & & \text{falls } top(t) \in PF_m \cup \{f\} \text{ und } x \notin \mathcal{V}ar\langle w \rangle \\
(P) & w(f(k(t_1, \ldots, t_n))) & \perp\!\!\!\rightarrow & w\left(\left(t_k^{(x_1,\ldots,x_n)}\right)_{x_1 \ldots x_n}^{t_1 \ldots t_n}\right) \\
 & & & \text{falls } k \in K_m \\
(BF) & w(f(\perp_{s_1})) & \perp\!\!\!\rightarrow & w(\perp_{s_2})
\end{array}
$$

Abbildung 17: Das Regelsystem *K-P* zur Separierung der Terme in $\mathcal{K}$- und $\mathcal{P}$-Terme.

Der Grund für diese Partitionierung ist, daß diese beiden Arten von Termen nun unterschiedlich behandelt werden müssen: Die Terme aus $\mathcal{K}$ "gehören" zu den in m neu definierten Sorten und müssen daher eliminiert werden. Die Terme aus $\mathcal{P}$ hingegen sind schon fast Terme über der Parametersignatur, es stören nur noch die Vorkommen von f, deren Argument dann allerdings eine Variable sein muß. Unser Ziel wird es nun sein, die Formeln so zu transformieren, daß wir diese Vorkommen von f aus den $\mathcal{P}$-Termen eliminieren können. Um die Terme wie oben beschrieben zu separieren, benutzen wir das Regelsystem aus Abbildung 17. Für jeden Satz $w \in \mathcal{S}en_m$ mit $w \rightarrow_K v$ gilt bereits auf Grund der Semantik der Prädikatenlogik $[m] \models w \leftrightarrows v$. Für $w \rightarrow_{P,BF} v$ folgt $[m] \models w \leftrightarrows v$ mit dem Substitutionssatz aus der Definition der partiellen symbolischen Auswertbarkeit.

Da die Argumentsorten einer Parameterfunktion Parametersorten sind, kann ein Term t mit $top(t) \in K_m$ nicht als Argument einer Parameterfunktion auftauchen. Aus diesem Grunde enthalten alle $\rightarrow_{K-P}$-Normalformen nur Terme aus $\mathcal{K}$ und $\mathcal{P}$. Wegen der Sorten der $\mathcal{K}$- und $\mathcal{P}$-Terme können darüber hinaus keine Atome $t_1 = t_2$ mit $t_1 \in \mathcal{K}$ und $t_2 \in \mathcal{P}$ auftauchen. Also enthalten alle $\rightarrow_{K-P}$-Normalformen nur Atome $t_1 = t_2$, für die entweder $t_1, t_2 \in \mathcal{K}$ oder $t_1, t_2 \in \mathcal{P}$ ist. Man beachte allerdings, daß ein Atom $t_1 = t_2$ auch im Kontext eines Negationszeichens , d. h. als "Ungleichung" $\neg(t_1 = t_2)$ in einer Formel auftreten kann.

Die Terminierung von $\rightarrow_{K-P}$ ist mit Hilfe einer geeigneten Multimengenordnung leicht einzusehen.

Beispiel: In diesem wie auch in den folgenden Beispielen beziehen wir uns auf das Modul *list*2 aus Abbildung 15, Seite 80.

$$
\begin{array}{rl}
 & \forall x \in list. \exists y \in elem. cons(y + sum2(x), x) \neq x \vee sum2(cons(sum2(x), x)) = y \quad (4) \\
\rightarrow_P & \forall x \in list. \exists y \in elem. cons(y + sum2(x), x) \neq x \vee sum2(x) + sum2(x) = y \\
\rightarrow_K & \forall x \in list. \exists y \in elem. (\exists y' : elem. cons(y', x) \neq x \wedge y' = y + sum2(x)) \quad (5) \\
 & \qquad \vee sum2(x) + sum2(x) = y
\end{array}
$$

Im nächsten Schritt sorgen wir dafür, daß kein in w vorkommender Term von neuer Sorte zu $\perp$ ausgewertet werden kann. Das in Abbildung 18 angegebene Regelsystem ist offensichtlich terminierend und korrekt. Man beachte, daß wir den Allquantor $\forall$ nur als syntaktische Abkürzung für $\neg\exists\neg$ definiert haben, eine Regel wie (BQ) kann also ebenso gut auf einen Allquantor ange-

Regelsystem BQ:

$(BQ) \quad \exists x : s \,.\, w \quad \bot\!\!\!\rightarrow \quad (\exists x \in s \,.\, w) \vee w_x^{-s}$

Abbildung 18: Das Regelsystem B zur Eleminierung von Quantoren $\exists x : s$

wandt werden. Das in Abbildung 19 angegebene Regelsystem BE ist ebenfalls terminierend, korrekt aber nur für positive Belegungen. Wir wenden daher $\rightarrow_{BE}$ nur auf $\rightarrow_{BQ)}$-Normalformen von Sätzen an und definieren:

$$\rightarrow_{prim} := (\rightarrow^{!}_{K-P} \circ \rightarrow^{!}_{BQ}) \circ \rightarrow_{BE}$$

Beispiel: (Fortsetzung)

$$\begin{aligned}
(5) \quad \rightarrow_{BQ} \quad & \forall x \in list.\exists y \in elem.(\exists y' \in elem.cons(y', x) \neq x \wedge y' = y + sum2(x)) \\
& \vee (cons(\bot_{elem}, x) \neq x \wedge \bot_{elem} = y + sum2(x)) \\
& \vee sum2(x) + sum2(x) = y \\
\rightarrow_{BE1} \quad & \forall x \in list.\exists y \in elem.(\exists y' \in elem.cons(y', x) \neq x \wedge y' = y + sum2(x)) \\
& \vee (\bot_{elem} \neq x \wedge \bot_{elem} = y + sum2(x)) \\
& \vee sum2(x) + sum2(x) = y \\
\rightarrow_{BE3} \quad & \forall x \in list.\exists y \in elem.(\exists y' \in elem.cons(y', x) \neq x \wedge y' = y + sum2(x)) \qquad (6) \\
& \vee (\neg \texttt{False} \wedge \bot_{elem} = y + sum2(x)) \\
& \vee sum2(x) + sum2(x) = y
\end{aligned}$$

Die Teilformel $\neg \texttt{False} \wedge w$ ist natürlich äquivalent zu w. Da wir aber nur an der prinzipiellen Berechenbarkeit interessiert sind, haben wir keine Regeln zur aussagenlogischen Vereinfachung der Formeln in das Regelsystem aufgenommen. Es gilt also insgesamt:

$$(4) \rightarrow^{!}_{prim} (6)$$

Die $\rightarrow_{prim}$-Normalformen von Sätzen enthalten also sogar nur Gleichungen zwischen $\mathcal{P}$-Termen oder zwischen $\mathcal{K}$-Termen, die kein Vorkommen von $\bot$ enthalten. Da alle Variablen mit $\exists \cdot \in \cdot$, bzw. $\forall \cdot \in \cdot$ quantifziert sind, brauchen wir darüber hinaus im folgenden die Korrektheit der Regeln nur noch für positive Belegungen zu betrachten.

Wir werden nun ein Verfahren angeben, um einen Satz $w \in \mathcal{S}en_m$ in einen äquivalenten Satz in $\mathcal{P}Sen_m$ zu überführen. Dabei gehen wir schrittweise vor und eliminieren von innen nach außen die Quantoren von neuer Sorte. Die Quantoren von importierter Sorte werden in den zu konstruierenden $\mathcal{P}Sen_m$-Satz eingehen.

In Anlehnung an die Fragmente der Prädikatenlogik erster Stufe partitionieren wir nun die Menge $\mathcal{W}ff(\Sigma_E, X)$ in m-Fragmente. Im Gegensatz zur Prädikatenlogik erster Stufe interessieren wir uns jetzt aber nur für die Quantoren von neuer Sorte. Eine Formel $w \in \mathcal{W}ff(\Sigma_E, X)$ ist in *m-Prenexnormalform*, falls w von der Form

$$Q_1 x_1 \in s_1 \ldots Q_n x_n \in s_n \,.\, p$$

Regelsystem BE:

$(BE1)$	$w(k(t_1,\ldots,\bot,\ldots,t_n))$	$\bot\!\!\rightarrow$	$w(\bot)$
			falls $k \in K_m$
$(BE2)$	$\bot = \bot$	$\bot\!\!\rightarrow$	`True`
$(BE3)$	$t = \bot$	$\bot\!\!\rightarrow$	`False`
			falls $t \in \mathcal{K}$ und $\bot$ kommt nicht in t vor

Abbildung 19: Das Regesystem *BE* zur Eliminierung undefinierter Träger

ist, wobei $Q_i \in \{\exists, \forall\}$ und $p \in \mathcal{W}ff(\Sigma_E, X)$ keine Quantoren über neuer Sorte enthält. Die *Anzahl der m-Quantoralternierungen* in w ist die Anzahl der i, $1 \leq i < n$ mit $Q_i \neq Q_{i+1}$. Falls q diese Anzahl ist und $n \geq 1$, dann ist

$$w \in \begin{cases} \Sigma^m_{q+1}\mathcal{W}ff(\Sigma_E, X) & \text{falls } Q_1 = \exists \\ \Pi^m_{q+1}\mathcal{W}ff(\Sigma_E, X) & \text{falls } Q_1 = \forall \end{cases}$$

$\Sigma^m_0\mathcal{W}ff(\Sigma_E, X) = \Pi^m_0\mathcal{W}ff(\Sigma_E, X)$ bezeichnet die Menge der Formeln, für die $n = 0$.

Beispiel: (Fortsetzung) Die Formel (6) auf Seite 84 hat eine m-Quantoralternierung und ist also ein Element von $\Pi^m_2\mathcal{W}ff(\Sigma_E, X)$.

Wie in der Prädikatenlogik kann man jede Formel $w \in \mathcal{W}ff(\Sigma_E, X)$ in eine (sogar im Sinne der Prädikatenlogik) äquivalente m-Prenexnormalform transformieren. Hierzu genügt es beispielsweise, die Prenexnormalform (im Sinne der Prädikatenlogik) zu berechnen.

Lemma 36 *Sei $\Im$ ein Domainoperator, m ein Modul mit Signatur (Σ_P, Σ_E) und $NF_m = \{f : s_1 \rightarrow s_2\}$ mit $s_1 \in NS$ und $s_2 \in PS$. Falls*

- *f symbolisch partiell auswertbar ist und*
- *es eine vollständige Menge C für f in $\Im$ gibt,*

dann gibt es eine totale berechenbare Funktion

$$\phi\colon \Sigma^m_2\mathcal{W}ff(\Sigma_E, X) \cap \downarrow_{prim} \rightarrow \Sigma^m_1\mathcal{W}ff(\Sigma_E, X) \cap \downarrow_{prim}$$

so daß $\mathcal{V}ar\langle\phi(w)\rangle \subseteq \mathcal{V}ar\langle w\rangle$ und für alle $A \in \Im_{\Sigma_P}$ und $\alpha \in ?^{+}_{X,A}$ ist $A(w)(\alpha) = A(\phi(w))(\alpha)$.

Lemma 37 *Sei $\Im$ ein Domainoperator, m ein Modul mit Signatur (Σ_P, Σ_E) und $NF_m = \{f : s_1 \rightarrow s_2\}$ mit $s_1 \in NS$ und $s_2 \in PS$. Falls*

- *f symbolisch partiell auswertbar ist und*
- *es eine vollständige Menge C für f in $\Im$ gibt,*

Regelsystem DNF:

$$
\begin{array}{lrcl}
(DNF1) & \neg(w \wedge v) & \perp\!\!\!\rightarrow & \neg w \vee \neg v \\
(DNF2) & \neg(w \vee v) & \perp\!\!\!\rightarrow & \neg w \wedge \neg v \\
(DNF3) & \neg\neg w & \perp\!\!\!\rightarrow & w \\
(DNF4) & (w_1 \wedge w_2) \vee w_3 & \perp\!\!\!\rightarrow & (w_1 \vee w_3) \wedge (w_2 \vee w_3) \\
(DNF5) & \forall x\,.(w_1 \wedge w_2) & \perp\!\!\!\rightarrow & (\forall x\,.\,w_1) \wedge (\forall x\,.\,w_2)
\end{array}
$$

Abbildung 20: Das Regelsystem *DNF*

dann gibt es eine totale berechenbare Funktion

$$\psi\colon (\Pi_1^m \mathcal{W}ff(\Sigma_E, X) \cup \Pi_0^m \mathcal{W}ff(\Sigma_E, X)) \cap \mathcal{S}en_m \cap \downarrow_{prim} \rightarrow \mathcal{P}Sen_m$$

so daß $\Im_{\Sigma_P}\,[m| \models \psi(w) \leftrightarrows w$.

Bevor wir diese beiden Lemmata beweisen, zeigen wir zuerst, wie Satz 6 aus den Lemmata 36 und 37 folgt. Zu einem Satz w berechnen wir zunächst eine $\rightarrow_{prim}$-Normalform w^*. Anschließend transformieren wir w^* in eine m-Prenexnormalform w', w' ist dann ebenfalls in $\rightarrow_{prim}$-Normalform. Wir zeigen per Induktion nach der Anzahl der m-Quantoralternierungen von w', daß wir w' effektiv in einen m-äquivalenten Satz aus $\mathcal{P}Sen_m$ transformieren können.

Falls $w' \in \Pi_1^m \mathcal{W}ff(\Sigma_E, X) \cup \Pi_0^m \mathcal{W}ff(\Sigma_E, X)$, dann berechne $\psi(w)$ gemäß Lemma 37. Falls $w' \in \Sigma_1^m \mathcal{W}ff(\Sigma_E, X)$, dann sei zunächst $w'' \in \Pi_1^m \mathcal{W}ff(\Sigma_E, X)$ eine m-Prenexnormalform von $\neg w'$. $\neg\psi(w'')$ leistet dann das Gewünschte.

Es habe nun w' $n \geq 1$ m-Quantoralternierungen. Dann gibt es für w' die beiden folgenden Möglichkeiten:

$$
\begin{array}{rcl}
w' & = & Q_1 z_1 \in z_1 \ldots Q_n z_n \in z_n \underbrace{\exists x_1 \in \sigma_1 \ldots \exists x_m \in \sigma_n \forall y_1 \in \sigma_1' \ldots \forall y_k \in \sigma_k'\,.\,p}_{v} \\
w' & = & Q_1 z_1 \in z_1 \ldots Q_n z_n \in z_n \underbrace{\forall x_1 \in \sigma_1 \ldots \forall x_m \in \sigma_n \exists y_1 \in \sigma_1' \ldots, \exists y_k \in \sigma_k'\,.\,p}_{v}
\end{array}
$$

Im ersten Fall ersetzen wir in w' die Teilformel v durch $\phi(v)$ gemäß Lemma 36 und erhalten so eine m-äquivalente Formel mit $n \perp 1$ m-Quantoralternierungen, auf die wir dann die Induktionsannahme anwenden können. Im zweiten Fall gehen wir, wie beim Induktionsanfang, analog für die m-Prenexnormalform der Negation von w' vor. □

Es bleiben noch die Lemmata 36 und 37 zu beweisen. Wir zeigen hier nur Lemma 36, der Beweis von Lemma 37 verläuft analog hierzu.

Beweis: [von Lemma 36] Im folgenden werden wir aus Gründen der Übersichtlichkeit die Sortenspezifikationen der Quantoren nicht mehr schreiben. Da wir von $\rightarrow_{prim}$-Normalformen ausgehen, sind alle Quantoren von der Form $Qx \in s$ (und nicht $Qx : s$), so daß der Zusammenhang jeweils klar ist. Außerdem betrachten wir nun Äquivalenzklassen von Formeln bezüglich $\leftrightarrow_{WE}^*$.

Wie können wir nun die in Abschnitt 4.4 Ersetzungssysteme auf Äquivalenzklassen von Formeln anwenden? Hierzu definieren wir nun zu einer Regel R eine Relation $\stackrel{\sim}{\rightarrow}_R$, bei der wir zunächst einen Übergang zu einer bezgl. $\leftrightarrow^*_{WE}$ äqivalenten Formel erlauben und anschließend einen $\rightarrow_R$-Schritt durchführen:

$$w \stackrel{\sim}{\rightarrow}_R v \quad \Leftrightarrow \quad \text{es gibt ein } w' \text{ mit } w \leftrightarrow^*_{WE} w' \rightarrow_R v$$

In der Terminologie von [DJ90] ist dies die Relation $\rightarrow_{R/WE}$. Da wir sowieso bei Anwendung de Regeln einen Übergang zu einer kongruenten Formel erlauben, brauchen wir die Formelschemata in den Regeln auch nur noch bis auf Kongruenz anzugeben. Wir können also die gleichen Konventionen, die wir bisher bei der Semantik der Formeln benutzt haben, auch hier einsetzen.

Wir gehen jetzt so vor, daß wir ein neues Ersetzungssystem durch Angabe einiger Ersetzungsregeln definieren, wobei wir aber manchmal nach einem Ersetzungsschritt eine Normalisierung bzgl. des bisher definierten Ersetzungssystems verlangen. Auf diese Weise wird die Menge der Normalformen der einzelnen Ersetzungssysteme immer weiter eingegrenzt. In der letzten Stufe sind die Normalformen dann genau die gewünschten Formeln.

Da wir von $\rightarrow_{prim}$-Normalformen ausgehen, brauchen wir von den Regeln nur Korrektheit bezüglich positiver Belegungen (statt allgemeiner Korrektheit) zu fordern.

1.) Disjunktive Normalformen Betrachte die Relation $\stackrel{\sim}{\rightarrow}_{DNF}$ gemäß Abbildung 20. $\stackrel{\sim}{\rightarrow}_{DNF}$ ist bekanntlich korrekt (nicht nur bzgl. positiver Belegungen) und noethersch. Die Anwendung von $\stackrel{\sim}{\rightarrow}_{DNF}$ zerstört nicht die $\stackrel{\sim}{\rightarrow}_{prim}$-Normalformen. Die $\stackrel{\sim}{\rightarrow}_{DNF}$-Normalformen von $\Sigma_2^m\mathcal{W}ff(\Sigma_E, X)$-Formeln in $\stackrel{\sim}{\rightarrow}_{prim}$-Normalform sind von der Form

$$\exists \vec{x}\,(\,\forall \vec{y}_1\,.\,l_{1,1} \vee \ldots \vee l_{1,n_1}) \wedge \ldots \wedge (\,\forall \vec{y}_k\,.\,l_{k,1} \vee \ldots \vee l_{k,n_k}\,)$$

wobei die l_j von einer der folgenden Formen sind:

- $t_1 = t_2$ für Terme t_1, t_2,
- $\neg(t_1 = t_2)$ für Terme t_1, t_2 oder
- $Q_1 x_1 \cdots Q_n . x_n . P$, wobei alle x_i von importierter Sorte sind und P keine Quantoren von neuer Sorte enthält.

Die $\stackrel{\sim}{\rightarrow}_{DNF}$-Normalformen von Formeln aus $\Sigma_2^m\mathcal{W}ff(\Sigma_E, X)$ in $\rightarrow_{prim}$-Normalform nennen wir auch *disjunktive Normalformen*.

Beispiel:

$$\begin{aligned} &\exists x_1, x_2 \in elem . \forall y_1, y_2 \in list. \\ &\qquad atom(x_1) \neq atom(x_2) \\ &\qquad \vee (cons(x_1, y_1) = cons(x_2, y_2) \wedge atom(x_1) = cons(x_1, y_1)) \\ \stackrel{\sim}{\rightarrow}_{DNF4}\ &\exists x_1, x_2 \in elem . \forall y_1, y_2 \in list. \end{aligned} \tag{7}$$

Regelsystem analyze:

(Tr)	$x = x$	$\bot\!\!\!\rightarrow$	`True`
(Oc)	$x = t$	$\bot\!\!\!\rightarrow$	`False`
			falls $t \in \mathcal{K} \setminus X$ und $x \in \mathcal{V}ar\langle t \rangle$
(De)	$k(t_1, \ldots, t_n) = k(u_1, \ldots, t_n)$	$\bot\!\!\!\rightarrow$	$(t_1 = u_1) \wedge \ldots \wedge (t_n = u_n)$
			falls $k \in K_m$
(Cl)	$k_1(t_1, \ldots, t_n) = k_2(u_1, \ldots, u_m)$	$\bot\!\!\!\rightarrow$	`False`
			falls $k_1, k_2 \in K_m$, $k_1 \neq k_2$
(P)	$w(f(k(t_1, \ldots, t_n)))$	$\bot\!\!\!\rightarrow$	$w\left(\left(t_k^{(x_1,\ldots,x_n)}\right)_{x_1 \ldots x_n}^{t_1 \ldots t_n}\right)$
			falls $k \in K_m$

Abbildung 21: Das Regelsystem *analyze*

$$
\begin{aligned}
& (atom(x_1) \neq atom(x_2) \vee cons(x_1, y_1) = cons(x_2, y_2)) \\
& \wedge (atom(x_1) \neq atom(x_2) \vee atom(x_1) = cons(x_1, y_1)) \\
\overset{\sim}{\rightarrow}{}^{*}_{DNF5}\ & \exists x_1, x_2 \in elem. \\
& (\forall y_1, y_2 \in list.atom(x_1) \neq atom(x_2) \vee cons(x_1, y_1) = cons(x_2, y_2)) \\
& \wedge (\forall y_1, y_2 \in list.atom(x_1) \neq atom(x_2) \vee atom(x_1) = cons(x_1, y_1))
\end{aligned}
\tag{8}
$$

Es gilt also (7) $\overset{\sim}{\rightarrow}{}^{!}_{DNF}$ (8)

2.) Vereinfachte Normalformen Nun werden wir, mit dem Wissen um die Semantik der Konstruktoren und der partiellen symbolischen Auswertbarkeit von f, die disjunktiven Normalformen weiter zerlegen. Dazu benutzen wir die Regeln aus Abbildung 21. Die Anwendung dieser Regeln berührt nicht die $\overset{\sim}{\rightarrow}_{prim}$-Normalisierung der Formeln, bei Anwendung der Regel (De) kann aber unter Umständen die disjunktive Normalform der Formel zerstört werden. Aus diesem Grunde führen wir, um die Betrachtungen auch weiterhin auf disjunktive Normalformen beschränken zu können, nach jedem $\overset{\sim}{\rightarrow}_{analyze}$-Ersetzungsschritt eine $\overset{\sim}{\rightarrow}_{DNF}$-Normalisierung durch. Wir betrachten also die Relation

$$\overset{\sim}{\rightarrow}_{simplify} \quad := \quad \overset{\sim}{\rightarrow}_{analyze} \circ \overset{\sim}{\rightarrow}{}^{!}_{DNF}$$

Die $\overset{\sim}{\rightarrow}_{simplify}$-Normalformen von disjunktiven Normalformen nennen wir auch *vereinfachte Normalformen*. $\overset{\sim}{\rightarrow}_{analyze}$ ist offenbar korrekt für positive Belegungen. Man beachte daß die Regeln (Cl) und (De) bei Betrachtung *aller* Belegungen nicht korrekt wären. Die Terminierung von $\overset{\sim}{\rightarrow}_{simplify}$ ist leicht zu zeigen. Eine vereinfachte Normalform einer $\Sigma_2^m\mathcal{W}ff(\Sigma_E, X)$-Formel hat nun die Gestalt

$$\exists \vec{x}\,(\,\forall \vec{y}_1\,.\,l_{1,1} \vee \ldots \vee l_{1,n_1}) \wedge \ldots \wedge (\forall \vec{y}_k\,.\,l_{k,1} \vee \ldots \vee l_{k,n_k})$$

wobei die l_j von einer der folgenden Formen sind:

- $z = t$ für einen Term $t \in \mathcal{K}$ und eine Variable $z \notin \mathcal{V}ar\langle t\rangle$ oder
- $\neg(z = t)$ für einen Term $t \in \mathcal{K}$ und eine Variable $z \notin \mathcal{V}ar\langle t\rangle$ oder
- eine Gleichung oder Ungleichung zwischen zwei Termen aus $\mathcal{P}$ oder
- $Q_1 x_1 \cdots Q_n x_n.P$, wobei alle x_i von importierter Sorte sind und P keine Quantoren von neuer Sorte enthält. Alle Vorkommen von f in v sind von der Form $f(z)$ für eine Variable z von neuer Sorte.

Beispiel: (Fortsetzung)

$$
\begin{aligned}
(8) \quad &\stackrel{\sim}{\rightarrow}_{De} \exists x_1, x_2 \in elem. && (9)\\
&\qquad (\forall y_1, y_2 \in list.x_1 \neq x_2 \vee cons(x_1, y_1) = cons(x_2, y_2))\\
&\qquad \wedge(\forall y_1, y_2 \in list.atom(x_1) \neq atom(x_2) \vee atom(x_1) = cons(x_1, y_1))\\
&\stackrel{\sim}{\rightarrow}_{De} \exists x_1, x_2 \in elem.\\
&\qquad (\forall y_1, y_2 \in list.x_1 \neq x_2 \vee cons(x_1, y_1) = cons(x_2, y_2))\\
&\qquad \wedge(\forall y_1, y_2 \in list.x_1 \neq x_2 \vee atom(x_1) = cons(x_1, y_1))\\
&\stackrel{\sim}{\rightarrow}_{Cl} \exists x_1, x_2 \in elem.\\
&\qquad (\forall y_1, y_2 \in list.x_1 \neq x_2 \vee cons(x_1, y_1) = cons(x_2, y_2))\\
&\qquad \wedge(\forall y_1, y_2 \in list.x_1 \neq x_2 \vee \texttt{False}\\
&\stackrel{\sim}{\rightarrow}_{De} \exists x_1, x_2 \in elem.\\
&\qquad (\forall y_1, y_2 \in list.x_1 \neq x_2 \vee (x_1 = x_2 \wedge y_1 = y_2))\\
&\qquad \wedge(\forall y_1, y_2 \in list.x_1 \neq x_2 \vee \texttt{False}\\
\stackrel{\sim}{\rightarrow}^{!}_{DNF} &\exists x_1, x_2 \in elem.\\
&\qquad (\forall y_1, y_2 \in list.x_1 \neq x_2 \vee x_1 = x_2)\\
&\qquad \wedge(\forall y_1, y_2 \in list.x_1 \neq x_2 \vee y_1 = y_2)\\
&\qquad \wedge(\forall y_1, y_2 \in list.x_1 \neq x_2 \vee \texttt{False}) && (10)
\end{aligned}
$$

Also gilt (9) $\stackrel{\sim}{\rightarrow}^{!}_{simplify}$ (10)

3.) Separierte Normalformen Die ersten beiden Formen der l_j in obiger Aufstellung sind also schon recht weit zerlegt. Allerdings können die Formeln P noch Terme aus $\mathcal{K}$ enthalten. Diesen Fall wollen wir in diesem Schritt ausschließen, so daß eine Formel P in der obigen Situation also nur Terme aus $\mathcal{P}$ enthalten kann. Dazu benutzen wir das Regelsystem PQ aus Abbildung 22.

Die Korrektheit der Regeln PQA und PQE bezüglich positiver Belegungen folgt unmittelbar aus Lemma 11. Zur Korrektheit der Regel PQD beachte man, daß die Formeln $P \wedge (R \vee w)$ und $(w \wedge P) \vee (\neg w \wedge (P \wedge R))$ bereits im Sinne der Aussagenlogik äquivalent sind. Man zeigt nun mit Hilfe der elementaren Eigenschaften der Prädikatenlogik, daß für alle Formeln A, B und w mit $x \notin \mathcal{V}ar\langle w\rangle$ die folgenden beiden Formeln allgemeingültig sind:

Regelsystem PQ:

(PQA) $\quad \forall \vec{y}, y\,.\,P \vee Q\vec{\gamma}\,.\,R \quad \perp\!\!\!\rightarrow \bigwedge_{k \in K_{m,s}} \forall \vec{y} \setminus \{y\} \cup \{y_1, \ldots, y_n\}.P_y^{k(y_1,\ldots,y_n)} \vee Q\vec{\gamma}.R_y^{k(y_1,\ldots,y_n)}$

falls
- y ist von neuer Sorte und
- alle $\gamma \in \vec{\gamma}$ sind von Parametersorte und
- $(\vec{\gamma} \cup \vec{y}) \cap \{y_1, \ldots, y_n\} = \emptyset$ und
- R enthält eine Gleichung $y = t$ mit $\mathcal{V}ar\langle t \rangle \cap \vec{\gamma} \neq \emptyset$

(PQE) $\quad \exists \vec{x}\,.\,(T \wedge \forall \vec{y}\,.\,(P \vee Q\vec{\gamma}.R)) \quad \perp\!\!\!\rightarrow \bigvee_{k \in K_{m,s}} \exists \vec{x}, \vec{z}\,.\,z = k(\vec{z}) \wedge T_z^{k(\vec{z})} \wedge \forall \vec{y}\,.\,(P_z^{k(\vec{z})} \vee Q\vec{\gamma}.R_z^{k(\vec{z})})$

falls
- z ist von neuer Sorte und
- alle $\gamma \in \vec{\gamma}$ sind von Parametersorte und
- $(\vec{\gamma} \cup \vec{y} \cup \vec{x}) \cap \vec{z} = \emptyset$ und
- R enthält eine Gleichung $y = t$ mit $\mathcal{V}ar\langle t \rangle \cap \vec{\gamma} \neq \emptyset$

(PQD) $\quad Q\vec{x}.(P \wedge (R \vee w)) \quad \perp\!\!\!\rightarrow \quad (w \wedge Q\vec{x}.P) \vee (\neg w \wedge Q\vec{x}.(P \wedge R))$

falls
- $\vec{x} \cap \mathcal{V}ar\langle w \rangle = \emptyset$ und
- alle $x \in \vec{x}$ sind von Parametersorte

Abbildung 22: Das Regelsystem PQ zur Eliminierung von $\mathcal{K}$-Termen

- $\exists x.((w \wedge A) \vee (\neg w \wedge B)) \mathrel{\mathfrak{X}} (w \wedge \exists x.A) \vee (\neg w \wedge \exists x.B)$
- $\forall x.((w \wedge A) \vee (\neg w \wedge B)) \mathrel{\mathfrak{X}} (w \wedge \forall x.A) \vee (\neg w \wedge \forall x.B)$

Wir definieren nun

$$\overset{\sim}{\rightarrow}_{separate} := \overset{\sim}{\rightarrow}_{PQA,PQE,PQD} \circ \overset{\sim}{\rightarrow}^{!}_{simplify}$$

Insbesondere wird somit nach jeder Anwendung von PQA oder PQE eine der Regeln De oder Cl angewandt, was dazu führt, daß ein Vorkommen einer Variablen γ in einem $\mathcal{K}$-Term entweder verschwindet (bei Anwendung von Cl) oder "nach oben rutscht" (bei Anwendung von De). Die Terminierung ist daher leicht einzusehen.

Als *separierte P-Formeln* bezeichnen wir nun die Formeln,

- die keine Quantoren über neuer Sorte enthalten und
- die nur Terme aus $\mathcal{P}$ enthalten und
- für die alle Vorkommen von f in P von der Form $f(z)$ sind für eine Variable z von neuer Sorte.

Die $\stackrel{\sim}{\rightarrow}_{separate}$-Normalformen von $\stackrel{\sim}{\rightarrow}_{simplify}$-Normalformen nennen wir auch *separierte Normalformen*. Eine separierte Normalform einer $\Sigma_2^m\mathcal{W}ff(\Sigma_E, X)$-Formel ist nun einer Disjunktion von Formeln der Gestalt

$$\exists\vec{x}\,(\,\forall\vec{y}_1\,.(l_{1,1}\vee\ldots\vee l_{1,n_1}\vee P_1)\wedge\ldots\wedge\forall\vec{y}_k\,.(l_{k,1}\vee\ldots\vee l_{k,n_k}\vee P_k)\,)$$

wobei die P_i separierte P-Formeln sind und die l_j von einer der folgenden Formen sind:

- $z = t$ für einen Term $t \in \mathcal{K}$ und eine Variable $z \notin \mathcal{V}ar\langle t\rangle$ oder
- $\neg(z = t)$ für einen Term $t \in \mathcal{K}$ und eine Variable $z \notin \mathcal{V}ar\langle t\rangle$

Beispiel:

$$\begin{aligned}
& \forall y_1, y_2 \in list.y_1 = y_2 \vee \exists z \in elem.y_1 \neq cons(z, y_2) && (11)\\
\stackrel{\sim}{\rightarrow}_{PQA}\quad & (\forall x' \in elem, y_2 \in list.atom(x') = y_2 \vee \exists z \in elem.atom(x') \neq cons(z, y_2))\\
& \wedge(\forall x' \in elem, y', y_2 \in list.cons(x', y') = y_2)\\
& \qquad\qquad \vee(\exists z \in elem.cons(x', y') \neq cons(z, y_2))\\
\stackrel{\sim}{\rightarrow}^{!}_{simplify}\quad & (\forall x' \in elem, y_2 \in list.atom(x') = y_2 \vee \exists z \in elem.\neg\texttt{False})\\
& \wedge(\forall x' \in elem, y', y_2 \in list.cons(x', y') = y_2)\\
& \qquad\qquad \vee(\exists z \in elem.x' \neq z \vee y' \neq y_2)\\
\stackrel{\sim}{\rightarrow}^{!}_{PDQ}\quad & (\forall x' \in elem, y_2 \in list.atom(x') = y_2 \vee \exists z \in elem.\neg\texttt{False})\\
& \wedge(\forall x' \in elem, y', y_2 \in list.cons(x', y') = y_2)\\
& \qquad\qquad \vee(y' \neq y_2 \wedge \exists z \in elem.\texttt{True})\\
& \qquad\qquad \vee(y' = y_2 \wedge \exists z \in elem.x' \neq z)\\
\stackrel{\sim}{\rightarrow}_{DNF}\quad & (\forall x' \in elem, y_2 \in list.atom(x') = y_2 \vee \exists z \in elem.\neg\texttt{False}) && (12)\\
& \wedge(\forall x' \in elem, y', y_2 \in list.cons(x', y') = y_2 \vee y' \neq y_2 \vee y' = y_2)\\
& \wedge(\forall x' \in elem, y', y_2 \in list.cons(x', y') = y_2 \vee y' \neq y_2 \vee \exists z \in elem.x' \neq z)\\
& \wedge(\forall x' \in elem, y', y_2 \in list.cons(x', y') = y_2 \vee (\exists z \in elem.\texttt{True}) \vee y' = y_2)\\
& \wedge(\forall x' \in elem, y', y_2 \in list.cons(x', y') = y_2 \vee (\exists z \in elem.\texttt{True}) \vee \exists z \in elem.x' \neq z)
\end{aligned}$$

Also ist (11) $\stackrel{\sim}{\rightarrow}^{!}_{seperate}$ (12).

4.) Positive Normalformen Die in $\vec{y}_i$ enthaltenen Variablen *von neuer Sorte* nennen wir auch die *Parameter* von $l_{i,j}$. Im nächsten Schritt werden wir die Ungleichungen $\neg(z = t)$ mit $t \in \mathcal{K}$, die einen Parameter enthalten, eliminieren. Dabei müssen wir darauf achten, ob ein solches Vorkommen der Variablen z selbst ein Parameter ist oder nicht. Hierzu verwenden wir die in Abbildung 23 angegebenen Regeln. Auch hier müssen wir darauf achten, daß die in den vorherigen Schritten erzielten Normalformen nicht zunichte gemacht werden, denn durch die

Regelsystem El,Ex:

$$(El) \qquad \forall \vec{y}\,.\,(y \neq t \vee P) \quad \perp\!\!\!\rightarrow \quad \forall \vec{y} \setminus \{y\}\,.\,P_y^t$$
falls $y \in \vec{y}$ und $t \in \mathcal{K}$

$$(Ex) \quad \exists \vec{x}\,.\,(R \wedge \forall \vec{y}\,.\,(z \neq t \vee P)) \quad \perp\!\!\!\rightarrow \quad \bigvee_{k \in K_{m,s}} \exists \vec{x}, \vec{z}\,.\,z = k(\vec{z}) \wedge R_z^{k(\vec{z})} \wedge \forall \vec{y}\,.\,(k(\vec{z}) \neq t \vee P_z^{k(\vec{z})})$$
falls
- x ist von neuer Sorte und
- $\mathcal{V}ar\langle t \rangle \cap \vec{y} \neq \emptyset$ und
- $x \notin \vec{y}$

Abbildung 23: Das Regelsystem *El,Ex* zur Eliminierung von Ungleichungen

Ersetzung einer Variablen durch einen Term wird im allgemeinen die separierte Normalform zerstört. Aus diesem Grunde betrachten wir die Relation

$$\overset{\sim}{\rightarrow}_{positiv} \quad := \quad \overset{\sim}{\rightarrow}_{El,Ex} \circ \overset{\sim}{\rightarrow}^{!}_{separate}$$

Offenbar ist $\overset{\sim}{\rightarrow}_{positiv}$ eine für positive Belegungen korrekte Berechnungsregel. Die Terminierung von $\overset{\sim}{\rightarrow}_{positiv}$ sieht man wie folgt: Für ein Literal l sei $\zeta(l)$ der Multiset der Tiefen von Vorkommen von Parametern in l. Für eine Formel

$$w = \forall \vec{y}\,.\,l_1 \vee \ldots \vee l_n \vee P$$

gemäß unserer Darstellung der separierten Normalformen sei $\xi(w)$ das Paar, bestehend aus der Anzahl der Parameter (d. h. der Mächtigkeit von $\vec{y}$) und dem Multiset $\{\zeta(l_i) \mid i = 1 \ldots n\}$. Es sei $\preceq$ die lexikographische Kombination von $\leq$ und $\leq_{mul}$, $\preceq$ ist dann eine noethersche Ordnung auf $\mathbb{N} \times \mathcal{M}(\mathbb{N})$. Eine separierte Normalform d schließlich bilden wir ab auf $\eta(d)$, definiert als der Multiset der $\xi(w)$ für alle in d enthaltenen allquantifizierten Formeln w. Falls nun $d_1 \overset{\sim}{\rightarrow}_{positiv} d_2$ für eine zerlegte Normalform, d_1, dann ist $\eta(d_2) \prec_{mul} \eta(d_1)$.

Die $\overset{\sim}{\rightarrow}_{positiv}$-Normalformen von $\Sigma_2^m \mathcal{W}ff(\Sigma_E, X)$-Formeln nennen wir nun *positive Normalformen*. Diese sind Disjunktionen von Formeln der folgenden Gestalt:

$$\exists \vec{x}\,(\,(\forall \vec{y}_1\,.\,l_{1,1} \vee \ldots \vee l_{1,n_1} \vee P_1) \wedge \ldots \wedge (\forall \vec{y}_k\,.\,l_{k,1} \vee \ldots \vee l_{k,n_k} \vee P_k)\,)$$

wobei die P_i separierte P-Formeln sind und die $l_{i,j}$ von der Form sind:

- $z = t$ für einen Term $t \in \mathcal{K}$ und eine Variable $z \notin \mathcal{V}ar\langle t \rangle$ oder
- $z = t$ oder $z \neq t$, wobei $t \in \mathcal{K}$ und $z \notin \vec{y}$ und $\mathcal{V}ar\langle t \rangle \cap \vec{y} = \emptyset$

Beispiel:

$$\forall y_2 \in list.y_1 \neq cons(x, y_2) \vee sum(y_2) = x \tag{13}$$

$$
\begin{array}{ll}
\overset{\sim}{\rightarrow}_{Ex} & (\exists x_1' \in elem.y_1 = atom(x_1') \\
& \qquad \wedge \forall y_2 \in list.atom(x_1) \neq cons(x, y_2) \vee sum(y_2) = x) \\
& \vee (\exists x_1' \in elem, y_1' \in list.y_1 = cons(x_1', y_1') \\
& \qquad \wedge \forall y_2 \in list.cons(x_1', y_1') \neq cons(x, y_2) \vee sum(y_2) = x) \\
\overset{\sim}{\rightarrow}^{!}_{seperate} & (\exists x_1' \in elem.y_1 = atom(x_1') \\
& \qquad \wedge \forall y_2 \in list.\neg \texttt{False} \vee sum(y_2) = x) \\
& \vee (\exists x_1' \in elem, y_1' \in list.y_1 = cons(x_1', y_1') \\
& \qquad \wedge \forall y_2 \in list.x_1' \neq x \vee y_1' \neq y_2 \vee sum(y_2) = x) \\
\overset{\sim}{\rightarrow}_{El} & (\exists x_1' \in elem.y_1 = atom(x_1') \\
& \qquad \wedge \forall y_2 \in list.\neg \texttt{False} \vee sum(y_2) = x) \\
& \vee (\exists x_1' \in elem, y_1' \in list.y_1 = cons(x_1', y_1') \\
& \qquad \wedge x_1' \neq x \vee sum(y_1') = x)
\end{array}
\tag{14}
$$

Also ist (13) $\overset{\sim}{\rightarrow}^{!}_{positive}$ (14).

5.) Eliminierung der Parameter: Wir werden nun die Allquantoren über neuer Sorte elimieren. Ausgehend von einer $\Sigma_2^m \mathcal{W}ff(\Sigma_E, X)$-Formel in positiver Normalform erhalten wir somit eine $\Sigma_1^m \mathcal{W}ff(\Sigma_E, X)$-Formel. Die Literale, die selbst keine Parameter enthalten, werden uns dabei nicht weiter stören. Um eine Terminierung unseres Regelsystems zu erreichen, müssen wir die Möglichkeiten des Kontrollteiles erweitern. Zu diesem Zweck führen wir eine partielle Ordnung $\sqsubseteq$ auf den Parametern ein, die wir zur Formulierung der Bedingungen in den Regeln benutzen. Wir legen die am Anfang zu wählende Ordnung nicht fest, sondern fordern nur, daß alle in einem Literal vorkommenden Parameter in der Ordnung jeweils vergleichbar sind. Zu Anfang können wir also beispielsweise eine beliebige totale Ordnung auf den Parametern wählen.

Einige Regeln verändern die Parameter, so daß wir auch die Ordnung der Parameter entsprechend modifizieren müssen. Es bezeichnen

$$
\begin{array}{rcl}
\sqsubseteq_y^{y_1,\ldots,y_n} & = & \{(x_1, x_2) \mid x_1 \sqsubseteq x_2, x_1 \neq y \neq x_2\} \\
& \cup & \{(x_1, y_i) \mid x_1 \sqsubseteq y\} \\
& \cup & \{(y_i, x_2) \mid y \sqsubseteq x_2\} \\
\sqsubseteq \perp\{y\} & = & \{(x_1, x_2) \mid x_1 \sqsubseteq x_2, x_1 \neq y \neq x_2\}
\end{array}
$$

Man sieht zunächst, daß die folgende Eigenschaft, die wir ja zu Anfang von unserer Ordnung $\sqsubseteq$ gefordert haben, invariant ist unter Anwendung der Regeln $Ex1$, $Ex2$, $Ax1$, $Ax2$, Gr und Co:

Alle in einer Gleichung vorkommenden Parameter sind bezüglich $\sqsubseteq$ vergleichbar

Die Korrektheit der Regeln $Ex1$, $Ex2$, $Ax1$, $Ax2$ und Gr bezüglich positiver Belegungen folgt wiederum aus Lemma 11. Wir zeigen nun die Korrektheit von Co bezüglich positiver Belegungen. Hierzu betrachten wir eine Instanz von Co und eine Algebra $A \in \Im_{\Sigma_P}$, $B := [\![m]\!](A)$,

Regelsystem Ex1,Ex2:

$(Ex1)\quad \exists\vec{x}\,.(R \wedge \forall\vec{y}\,.(z = t \vee P)) \quad \perp\!\!\!\rightarrow \bigvee_{k\in K_{m,s}} \exists\vec{x},\vec{z}\,.z = k(\vec{z}) \wedge R_z^{k(\vec{z})} \wedge \forall\vec{y}\,.(k(\vec{z}) = t \vee P_z^{k(\vec{z})})$

falls
- z ist von neuer Sorte s und
- $\mathcal{V}ar\langle t\rangle \cap \vec{y} \neq \emptyset$
- $z \notin \vec{y}$

$(Ex2)\quad \exists\vec{x}\,.(R \wedge \forall\vec{y}\,.(y = t \vee P)) \quad \perp\!\!\!\rightarrow \bigvee_{k\in K_{m,s}} \exists\vec{x},\vec{z}\,.z = k(\vec{z}) \wedge R_z^{k(\vec{z})} \wedge \forall\vec{y}\,.(y = t_z^{k(\vec{z})} \vee P_z^{k(\vec{z})})$

falls
- y ist von neuer Sorte und $y \in \vec{y}$ und
- z ist von neuer Sorte und $z \in \mathcal{V}ar\langle t\rangle$ und
- $z \notin \vec{y}$ und
- es gibt ein $r \in C$ mit $t \trianglelefteq r$

Regelsystem Ax1,Ax2:

$(Ax1)\quad \forall y,\vec{y};\sqsubseteq .y = t \vee P \quad \perp\!\!\!\rightarrow \bigwedge_{k\in K_{m,s}} \forall\vec{y} \cup \{y_1,\ldots,y_n\};\sqsubseteq_y^{y_1,\ldots,y_n} .k(y_1,\ldots,y_n) = t \vee P_y^{k(y_1,\ldots,y_n)}$

falls
- y ist von neuer Sorte und
- $t \notin X$
- es gibt ein $y' \in \mathcal{V}ar\langle t\rangle \cap \vec{y}$ mit $y' \sqsubset y$

$(Ax2)\quad \forall y,\vec{y};\sqsubseteq .y' = t \vee P \quad \perp\!\!\!\rightarrow \bigwedge_{k\in K_{m,s}} \forall\vec{y} \cup \{y_1,\ldots,y_n\};\sqsubseteq_y^{y_1,\ldots,y_n} .y' = t_y^{k(y_1,\ldots,y_n)} \vee P_y^{k(y_1,\ldots,y_n)}$

falls
- y ist von neuer Sorte und $y \in \mathcal{V}ar\langle t\rangle$ und
- y' ist von neuer Sorte und $y' \sqsubset y$
- $t \notin X$ und es gibt ein $r \in \mathcal{K}$ mit $t \trianglelefteq r$

Abbildung 24: Die Regelsysteme *Ex1,Ex2* und *Ax1,Ax2*

$\alpha \in ?^+_{X,B}$. Die Informationen über die partielle Ordnung der Parameter benötigen wir in diesem Beweisschritt nicht.

$\supseteq$

Es sei

$$B,\alpha \models \forall y,\vec{y}.y = t_1 \vee \ldots \vee y = t_n \vee P \vee Q(f(y))$$

Man beachte, daß auf Grund der Bedingungen in Co y nicht frei in P, nicht frei in den t_i und in Q nur als Argument von f vorkommt. Es sei s die Sorte von y. Dann gilt insbesondere für alle $t \in \mathcal{K}t_m(s)$:

$$B,\alpha \models \forall\vec{y}\forall\mathcal{V}ar\langle t\rangle.t = t_1 \vee \ldots \vee t = t_n \vee P \vee Q(f(t))$$

Da nach Voraussetzung die t_i nicht mit t unifizierbar sind, gilt somit für alle $t \in \mathcal{K}t_m(s)$:

$$B,\alpha \models \forall\vec{y}.\mathcal{V}ar\langle t\rangle.P \vee Q(f(t))$$

Regelsystem Gr:

$$(Gr) \quad \forall y, \vec{y}; \sqsubseteq .y = t \vee P \quad \perp\!\!\!\rightarrow \bigwedge_{k \in K_{m,s}} \forall \vec{y} \cup \{y_1, \ldots, y_n\}; \sqsubseteq_y^{y_1,\ldots,y_n} .k(y_1, \ldots, y_n) = t \vee P_y^{k(y_1,\ldots,y_n)}$$

falls
- y ist von neuer Sorte s und
- $t \in \mathcal{K}t_m(s)$

Abbildung 25: Die Regel Gr

Regelsystem Co:

$$(Co) \quad \forall y, \vec{y}; \sqsubseteq .y = t_1 \vee \ldots \vee y = t_n \vee P \vee Q(f(y))$$
$$\perp\!\!\!\rightarrow \quad \forall \vec{y}; \sqsubseteq \perp\{y\} . P \vee \bigwedge_{t \in C} \forall \mathcal{V}ar\langle t \rangle . Q(f(t))$$

falls
- $y \notin \mathcal{V}ar\langle P \rangle$ und $y \notin \mathcal{V}ar\langle t_i \rangle$ für alle i und
- Q ist eine separierte P-Formel und
- für alle i ist t_i mit keinem Term aus C unifizierbar

Abbildung 26: Die Regel Co

Also gilt, da $C \subseteq \mathcal{K}t_m(S)$:

$$B, \alpha \models \forall \vec{y}.P \vee \bigwedge_{t \in C} \forall \mathcal{V}ar\langle t \rangle . Q(f(t))$$

$\underline{\supseteq}$

Es sei

$$B, \alpha \models \forall \vec{y}.P \vee \bigwedge_{t \in C} \forall \mathcal{V}ar\langle t \rangle . Q(f(t))$$

Da C vollständig für f in $\Im$ ist und da y in $Q(f(t))$ nicht frei vorkommt, gilt für alle $\beta \in ?_{X,B}$:

$$B, \beta \models (\bigwedge_{t \in C} \forall \mathcal{V}ar\langle t \rangle . Q(f(t))) \quad \supset \quad (\forall y \in s . Q(f(y)))$$

Also gilt auch

$$B, \alpha \models \forall \vec{y}.P \vee \forall y . Q(f(y))$$

und somit:

$$B, \alpha \models \forall y, \vec{y} . y = t_1 \vee \ldots \vee y = t_n \vee P \vee Q(f(y))$$

Wir werden die Regeln $Ex1$, $Ex2$, $Ax1$, $Ax2$, Gr und Co nun schrittweise zu einer noetherschen Ersetzungsrelation kombinieren. Dabei betrachten wir stets positive Nomalformen. Diese bestehen, wir wir oben gesehen haben, aus *Klauseln* der folgenden Gestalt:

$$\forall \vec{y}; \sqsubseteq .l_1 \vee \ldots \vee l_n \vee P$$

wobei die l_i von der Form $y_i = t_i$ oder $y_i \neq t_i$ sind mit $t_i \in \mathcal{K}$ und P eine separierte P-Formel ist. Für eine gegebene Klausel w und einen Parameter y von w ist der *Rang* von y $rg(y)$ definiert als die Länge der längsten Kette

$$y_0 \sqsubset y_1 \sqsubset \ldots \sqsubset y_n = y$$

Der *Durchmesser* von w $dm(w)$ ist der höchste Rang eines Parameters von w. Man beachte, daß sich durch Anwendung der Regeln der Rang der Klauseln nicht erhöht.

Beispiel: Die Klausel

$$\forall y_1, y_2, y_3 . y_1 = cons(x, y_2) \vee y_2 = cons(x, y_3) \vee y_2 = cons(x, y_1) \tag{15}$$

hat unter der Ordnung $y_1 \sqsubset y_2 \sqsubset y_3$ den Durchmesser 3.

Wir zeigen nun zunächst die Terminierung der Relation

$$\overset{\sim}{\rightarrow}_{E1} := \overset{\sim}{\rightarrow}_{Ex1,Ex2,Ax1,Ax2} \circ \overset{\sim}{\rightarrow}^{!}_{positive}$$

Dazu definieren wir für $i \leq dm(w)$ die folgenden Multimengen:

$$\begin{aligned} fehl(i) &:= \{lth(o) \mid y' = t \in w \wedge t|_o = y, o \neq \epsilon \text{ für ein } y \text{ mit } rg(y) = i \wedge (y \sqsubset y' \vee y' \notin \vec{y})\} \\ match(i) &:= \{u \in C \mid y = t \in w \wedge rg(y) = i \wedge t \trianglelefteq u \wedge t \notin C\} \end{aligned}$$

$fehl(i)$ gibt also die Menge der Tiefen von "ungünstigen Vorkommen" eines Parameters von Rang i an. Dabei ist ein Vorkommen eines Parameters y ungünstig, falls es auf der rechten Seite einer Gleichung steht, deren linke Seite ein *größerer* Parameter oder gar kein Parameter ist.

Beispiel: (Fortsetzung) Für die Klausel (15) ist

$$\begin{aligned} fehl(1) &= \{1\} \\ match(1) &= \emptyset \\ fehl(2) = fehl(3) &= \emptyset \\ match(2) = match(3) &= \emptyset \end{aligned}$$

Es sei nun die Ordnung $\preceq_1$ auf $\mathcal{M}(\mathbb{N}) \times \mathcal{M}(C)$ definiert als die lexikographische Kombination der Multimengenerweiterung von $\leq$ und der Teilmengenrelation. Da $\mathcal{M}(C)$ nur *endliche* Multimengen enthält, ist $\preceq_1$ eine noethersche Relation.

Eine Klausel w mit Durchmesser n bilden wir nun ab auf

$$\Phi_1(w) := ((fehl(1), match(1)), \ldots, (fehl(n), match(n)))$$

Beispiel: (Fortsetzung) Für die Klausel 15 ist der Wert von Φ_1:

$$((\{1\}, \emptyset), (\emptyset, \emptyset), (\emptyset, \emptyset))$$

Da $\preceq_1$ noethersch ist, ist nun auch die Relation $\preceq_1 lex^n$ noethersch. Die Regeln $Ex1$, $Ex2$, $Ax1$ und $Ax2$ ersetzen in einer positiven Normalform, unter Umständen nach Normalisierung bzgl. $\overset{\sim}{\rightarrow}_{positive}$, eine Klausel durch (endlich viele) mehrere Klauseln. Wir werden nun zeigen, daß in diesem Falle *jede* der Ergebnisklauseln bzgl. $\preceq_1 lex^n$ kleiner ist als die Ausgangsklausel. Mit Hilfe eines Multimengen-Arguments erhalten wir damit leicht die Terminierung der Relation $\overset{\sim}{\rightarrow}_{E1}$. Diese Argumentation werden wir weiter unten noch präzisieren.

Bei Anwendung der Regeln $Ex1$ und $Ex2$ können die $fehl$- sowie die *match*-Komponenten nicht anwachsen. Dafür nimmt aber im Falle von $Ex1$ zumindest eine $fehl$-Komponente und im Falle von $Ex2$ zumindest eine *match*-Komponente ab, also verringert sich auch $\Phi_1(w)$.

Bei den Regeln $Ax1$ und $Ax2$ ist die Argumentation nicht so einfach. Wir müssen nun im Detail untersuchen, was mit den einzelnen Gleichungen $y_i = t_i$ passiert, wenn wir einen Parameter y durch $k(y_1, \ldots, y_n)$ ersetzen.

Im Falle von $Ax1$ verringert sich die $fehl$-Komponente auf der $rg(y')$-ten Position von $\Phi_1(w)$, bei $Ax2$ verringert sich die *match*-Komponente auf der $rg(y')$-ten Position. Allerdings können sich die Komponenten auf der $rg(y)$-ten Position erhöhen. Wir zeigen nun, daß *nur* Komponenten auf Positionen $\geq rg(y) > rg(y')$ anwachsen können. Nach der Definition der lexikographischen Erweiterung einer Ordnung nimmt dann Φ_1 bezügich $\preceq_1{}^n_{lex}$ ab.

Eine j-te Komponente kann sich nur verändern, falls es in P eine Gleichung l gibt und einen Parameter y^* mit $rg(y^*) = j$ und $\{y, y^*\} \subseteq \mathcal{V}ar\langle l \rangle$.

Wir nehmen also an, daß $y, y^* \in \mathcal{V}ar\langle l \rangle$ und $rg(y^*) < rg(y)$, d. h., da alle Parameter in einer Gleichung vergleichbar sind, $y^* \sqsubset y$. Wir untersuchen, welchen Einfluß die einzelnen Gleichungen auf eine Änderung von Φ_1 bei Anwendung von $\overset{\sim}{\rightarrow}_{E1}$ haben können. Die folgenden Fälle sind möglich:

- $y^* = t(y)$: Die $fehl$-Komponente von y^* bleibt unverändert, die *match*-Komponente kann sich sogar verringern (da t "grösser wird"), aber auf keinen Fall vergrössern.
- $y = t(y^*)$: Die $fehl$-Komponente von y^* verringert sich, die *match*-Komponente von y^* kann jedoch anwachsen, falls aus $y = t(y^*)$ nach Normalisierung bzgl. $\overset{\sim}{\rightarrow}_{positive}$ eine Gleichung $y' = y^*$ entsteht.
- $y^{**} = t(y, y^*)$: Beide Komponenten von y^* bleiben unverändert.

Beispiel: (Fortsetzung)

$$
\begin{aligned}
(15)\quad & \overset{\sim}{\rightarrow}_{Ax1} \circ \overset{\sim}{\rightarrow}^{!}_{positive} \\
& (\forall y_1, x_2^1, y_3.y_1 = cons(x, atom(x_2^1)) \vee \texttt{False} \vee \texttt{False}) && (16)\\
& \wedge(\forall y_1, x_2^1, y_2^1, y_3.y_1 = cons(x, cons(x_2^1, y_2^1)) \vee x_2^1 = x \vee x_2^1 = x) && (17)\\
& \wedge(\forall y_1, x_2^1, y_2^1, y_3.y_1 = cons(x, cons(x_2^1, y_2^1)) \vee y_2^1 = y_3 \vee x_2^1 = x) && (18)\\
& \wedge(\forall y_1, x_2^1, y_2^1, y_3.y_1 = cons(x, cons(x_2^1, y_2^1)) \vee x_2^1 = x \vee y_2^1 = y_1) && (19)\\
& \wedge(\forall y_1, x_2^1, y_2^1, y_3.y_1 = cons(x, cons(x_2^1, y_2^1)) \vee y_2^1 = y_3 \vee y_2^1 = y_1) && (20)
\end{aligned}
$$

Für die Werte von Φ_1 gilt:

$$\begin{aligned}
\Phi_1((16)) &= ((\emptyset,\emptyset),(\emptyset,\emptyset),(\emptyset,\emptyset)) \\
\Phi_1((17)) &= ((\emptyset,\emptyset),(\emptyset,\emptyset),(\emptyset,\emptyset)) \\
\Phi_1((18)) &= ((\emptyset,\emptyset),(\emptyset,\{atom(x)\}),(\emptyset,\{atom(x)\})) \\
\Phi_1((19)) &= ((\emptyset,\{atom(x)\}),(\emptyset,\{atom(x)\}),(\emptyset,\emptyset)) \\
\Phi_1((20)) &= ((\emptyset,\{atom(x)\}),(\emptyset,\{atom(x),atom(x)\}),(\emptyset,\{atom(x)\}))
\end{aligned}$$

(16) und (17) sind bereits in $\overset{\sim}{\rightarrow}_{E1}$-Normalform. Wir erhalten dann weiter:

(18) $\overset{\sim}{\rightarrow}_{Ax2} \circ \overset{\sim}{\rightarrow}_{positive}$

$$(\forall y_1, x_2^1, y_2^1, x_3^1.y_1 = cons(x, cons(x_2^1, y_2^1)) \vee y_2^1 = atom(x_3^1) \vee x_2^1 = x) \quad (21)$$

$$\wedge(\forall y_1, x_2^1, y_2^1, x_3^1, y_3^1.y_1 = cons(x, cons(x_2^1, y_2^1)) \vee y_2^1 = cons(x_3^1, y_3^1) \vee x_2^1 = x) \quad (22)$$

(19) $\overset{\sim}{\rightarrow}_{Ax2} \circ \overset{\sim}{\rightarrow}_{positive}$

$$(\forall y_1, x_2^1, x_2^2, y_3.y_1 = cons(x, cons(x_2^1, atom(x_2^2))) \vee x_2^1 = x \vee y_1 = atom(x_2^2)) \quad (23)$$

$$\wedge(\forall y_1, x_2^1, x_2^2, y_2^2, y_3.y_1 = cons(x, cons(x_2^1, cons(x_2^2, y_2^2))) \quad (24)$$
$$\vee x_2^1 = x \vee y_1 = cons(x_2^2, y_2^2))$$

und es gilt dann

$$\Phi_1((21)) = \Phi_1((22)) = \Phi_1((23)) = \Phi_1((24)) = ((\emptyset,\emptyset),(\emptyset,\emptyset)(\emptyset,\emptyset))$$

(21)-(24) sind in $\overset{\sim}{\rightarrow}_{E1}$-Normalform.

(20) $\overset{\sim}{\rightarrow}_{Ax2} \circ \overset{\sim}{\rightarrow}^{!}_{positive}$

$$(\forall y_1, x_2^1, x_2^2, y_3.y_1 = cons(x, cons(x_2^1, atom(x_2^2))) \quad (25)$$
$$\vee y_3 = atom(x_2^2) \vee y_1 = atom(x_2^2))$$

$$\wedge(\forall y_1, x_2^1, x_2^2, y_2^2, y_3.y_1 = cons(x, cons(x_2^1, cons(x_2^2, y_2^2))) \quad (26)$$
$$\vee y_3 = cons(x_2^2, y_2^2) \vee y_1 = cons(x_2^2, y_2^2))$$

$$\begin{aligned}
\Phi_1((25)) &= ((\emptyset,\emptyset),(\emptyset,\emptyset)(\emptyset,\emptyset)) \\
\Phi_1((26)) &= ((\emptyset,\emptyset),(\{1\},\emptyset)(\emptyset,\emptyset))
\end{aligned}$$

Wir verfolgen jetzt nicht mehr alle Klauseln, sondern rechnen jeweils nur noch mit einer weiter:

(26) $\overset{\sim}{\rightarrow}_{Ax1} \circ \overset{\sim}{\rightarrow}^{!}_{positive}$

$$(\forall y_1, x_2^1, x_2^2, y_2^2, x_3^1.y_1 = cons(x, cons(x_2^1, cons(x_2^2, y_2^2))) \quad (27)$$
$$\vee \texttt{False} \vee y_1 = cons(x_2^2, y_2^2))$$

$$\wedge(\forall y_1, x_2^1, x_2^2, y_2^2, x_3^1, y_3^1.y_1 = cons(x, cons(x_2^1, cons(x_2^2, y_2^2))) \quad (28)$$
$$\vee x_3^1 = x_2^2 \vee y_1 = cons(x_2^2, y_2^2))$$

$$\wedge(\forall y_1, x_2^1, x_2^2, y_2^2, x_3^1, y_3^1.y_1 = cons(x, cons(x_2^1, cons(x_2^2, y_2^2))) \quad (29)$$
$$\vee y_3^1 = y_2^2 \vee y_1 = cons(x_2^2, y_2^2))$$

$$\begin{aligned}\Phi_1((27)) &= ((\emptyset,\emptyset),(\emptyset,\emptyset)(\emptyset,\emptyset))\\ \Phi_1((28)) &= ((\emptyset,\emptyset),(\emptyset,\emptyset)(\emptyset,\emptyset))\\ \Phi_1((29)) &= ((\emptyset,\emptyset),(\emptyset,\{atom(x)\})(\emptyset,\{atom(x)\}))\end{aligned}$$

Und zu guter Letzt:

$$\begin{aligned}(29)\quad &\overset{\sim}{\to}_{Ax2} \circ \overset{\sim}{\to}^{!}_{positive}\\ &(\forall y_1, x_2^1, x_2^2, y_2^2, x_3^1, x_3^2. y_1 = cons(x, cons(x_2^1, cons(x_2^2, y_2^2))) \qquad (30)\\ &\qquad \vee y_2^2 = atom(x_3^2) \vee y_1 = cons(x_2^2, y_2^2))\\ &\wedge(\forall y_1, x_2^1, x_2^2, y_2^2, x_3^1, x_3^2, y_3^2. y_1 = cons(x, cons(x_2^1, cons(x_2^2, y_2^2))) \qquad (31)\\ &\qquad \vee y_2^2 = cons(x_3^2, y_3^2) \vee y_1 = cons(x_2^2, y_2^2))\end{aligned}$$

$$\Phi_1((30)) = \Phi_1((31)) = ((\emptyset,\emptyset),(\emptyset,\emptyset)(\emptyset,\emptyset))$$

Wir wollen darauf hinarbeiten, die Regel Co anwenden zu können. Dazu müssen wir erreichen, daß es für einen Parameter y in w keine Gleichungen $y = t$ mit $t \trianglelefteq r$ für ein $r \in C$ gibt. Die $\overset{\sim}{\to}_{E1}$-Normalformen erfüllen dies allerdings noch nicht, denn sie können immer noch Gleichungen $y = r$ mit $r \in C$ enthalten. Da $r \in C$ keinen Parameter von neuer Sorte enthalten kann, ist in dieser Situation keine der Regeln $Ex2$, $Ax2$ anwendbar.

Hier kommt nun Gr zum Einsatz. Wir zeigen zunächst, daß

$$\overset{\sim}{\to}_{E2} := \overset{\sim}{\to}_{Gr} \circ \overset{\sim}{\to}^{!}_{positive}$$

eine noethersche Relation ist. Dazu betrachten wir wiederum die Klauseln, die durch Anwendung von $\overset{\sim}{\to}_{E1}$ oder $\overset{\sim}{\to}_{E2}$ neu entstehen. Für eine Klausel w in der oben angegebenen Form bezeichnet

$$\Phi_2(w) := \#\{i \mid \exists y \in \mathcal{V}ar\langle l_i\rangle \cap \vec{y},\ y \text{ von neuer Sorte}\}$$

die Anzahl der in w enthaltenen Literale, die einen Parameter von neuer Sorte enthalten.

Beispiel: Betrachte die Formel

$$\forall y_1, y_2. y_1 = cons(z, atom(z')) \vee y_2 = cons(z, y_1) \qquad (32)$$

Es ist $\Phi_2((32)) = 2$.

$\overset{\sim}{\to}_{E2}$ (und ebenso auch $\overset{\sim}{\to}_{E1}$) erhöht den Wert von Φ_2 nicht. Der Grund hierfür ist, daß die Parameter von neuer Sorte nur noch in Gleichungen, nicht aber in Ungleichungen vorkommen. Durch Anwendung der Regel De in einem $\overset{\sim}{\to}_{simplify}$-Schritt kann zwar eine Gleichung in n Konjunktionen aufgespalten werden, durch Umwandlung in disjunktive Normalform erhalten wir hieraus aber widerum n Klauseln, die jeweils den gleichen Wert von Φ_2 wie die Ausgangsklausel haben. Es seien nun

$$\begin{aligned}\Psi_1(w) &:= \#\{i \mid l_i = (y_i = t_i), \mathcal{V}ar\langle l_i\rangle \cap \vec{y} \neq \emptyset, t_i \notin \mathcal{K}t_m(s), y_i \text{ von neuer Sorte}\}\\ \Psi_2(w) &:= \{t_i | t_i \in \mathcal{K}t_m(s), y_i = t_i \in w \text{ für ein } y_i \in \vec{y} \text{ neuer Sorte}\}\end{aligned}$$

Beispiel: (Fortsetzung)

$$\begin{aligned}\Psi_1((32)) &= 1\\ \Psi_2((32)) &= \{cons(z, atom(z'))\}\end{aligned}$$

Es sei nun

$$\Phi_3(w) := (\Psi_1(w), \Psi_2(w)) \in \mathbb{N} \times \mathcal{M}(\bigcup_{s \in NS_m} \mathcal{K}t_m(s))$$

und $\preceq_2$ die lexikographische Kombination von $\leq$ und der Teilmengenrelation. $\preceq_2$ ist eine noethersche Relation. Falls aus einer Klausel w durch Anwendung von $\overset{\sim}{\rightarrow}_{E2}$ die Klauseln $w_1, \ldots, w_n$ entstehen, so gilt $w_i \prec_2 w$ für alle i. Falls w' in einer $\overset{\sim}{\rightarrow}_{E2}$-Normalform einer Klausel w vorkommt, dann gilt darüber hinaus $\Phi_2(w') < \Phi_2(w)$.

Beispiel: (Fortsetzung)

$$\begin{aligned}(32) \quad &\overset{\sim}{\rightarrow}_{E2}\\ &(\forall x_1^1, y_2.\mathtt{False} \vee y_2 = cons(z, atom(x_1^1))) && (33)\\ &\wedge(\forall x_1^1, y_1^1, y_2.x_1^1 = z \vee y_2 = cons(z, cons(x_1^1, y_1^1))) && (34)\\ &\wedge(\forall x_1^1, y_1^1, y_2.y_1^1 = atom(z') \vee y_2 = cons(z, cons(x_1^1, y_1^1))) && (35)\end{aligned}$$

Es ist

$$\begin{aligned}\Phi_3((33)) &= (0, \{cons(z, atom(x_1^1))\})\\ \Phi_3((34)) &= (1, \emptyset)\\ \Phi_3((35)) &= (1, \{atom(z')\})\end{aligned}$$

(34) ist in $\overset{\sim}{\rightarrow}_{E2}$-Normalform, und $\Phi_2((34)) = 1$.

$$\begin{aligned}(35) \quad &\overset{\sim}{\rightarrow}_{E2}\\ &(\forall x_1^1, x_1^2, y_2.x_1^2 = z' \vee y_2 = cons(z, cons(x_1^1, atom(x_1^2)))) && (36)\\ &\wedge(\forall x_1^1, x_1^2, y_1^2, y_2.\mathtt{False} \vee y_2 = cons(z, cons(x_1^1, cons(x_1^2, y_1^2)))) && (37)\end{aligned}$$

(37) ist in $\overset{\sim}{\rightarrow}_{E2}$ Normalform, (36) müßte noch weiter reduziert werden, was hier aber nicht mehr durchführen wollen.

$$\begin{aligned}\Phi_2((36)) &= 1\\ \Phi_2((37)) &= 1\\ \Phi_3((36)) &= (0, \{cons(z, cons(x_1^1, atom(x_1^2)))\})\\ \Phi_3((37)) &= (1, \emptyset)\end{aligned}$$

Wir werden nun $\overset{\sim}{\rightarrow}_{E1}$ und $\overset{\sim}{\rightarrow}_{E2}$ zu einer noetherschen Relation kombinieren. Zu einer positiven Normalform v mit den Klauseln $w_1 \cdots w_n$ definieren wir

$$\Phi_4(w) := \{(\Phi_3(w), \Phi_1(w)) \mid 1 \leq i \leq n\}$$

$\preceq_3$ sei die Multiseteerweiterung der lexikographischen Kombination von $\preceq_2$ und $\preceq_1$. Wie wir oben gezeigt haben, gilt nun

- $w \overset{\sim}{\rightarrow}_{E1} w' \Rightarrow \Phi_4(w') = \Phi_4(w) \setminus \{(a,b)\} \cup \{(a_1,b_1),\ldots,(a_n,b_n)\}$, wobei $a_i \preceq_2 a$ und $b_i \prec_1 b$ für alle i
- $w \overset{\sim}{\rightarrow}^{!}_{E2} w' \Rightarrow \Phi_4(w') = \Phi_4(w) \setminus X \cup Y$, wobei für alle $(c,d) \in Y$ ein $(a,b) \in X$ existiert mit $c \prec_2 a$

Dies bedeutet nun, daß die Relation

$$\overset{\sim}{\rightarrow}_{E3} := \overset{\sim}{\rightarrow}_{E1} \cup \overset{\sim}{\rightarrow}^{!}_{E2}$$

noethersch ist.

Die Normalformen von $\overset{\sim}{\rightarrow}_{E3}$ haben nun die Eigenschaft, daß zumindest für alle bzgl. $\sqsubseteq$ minimalen Parameter die Bedinging der Regel Co zutrifft. Eine Anwendung von $\overset{\sim}{\rightarrow}^{!}_{Co}$ auf eine Klausel in $\overset{\sim}{\rightarrow}_{E2}$-Normalform bewirkt also eine Eliminierung zumindest der minimalen Paramter und somit eine Reduktion der Durchmesser der Klausel um mindestens den Wert 1.

Wir bilden also zu guter Letzt die Relation

$$\overset{\sim}{\rightarrow}_{E4} := \overset{\sim}{\rightarrow}^{!}_{E3} \circ (\overset{\sim}{\rightarrow}^{!}_{Co} \circ \overset{\sim}{\rightarrow}^{!}_{simplify})$$

Die Anwendung des Regelsystems $simplify$ ist dabei notwendig, um die disjunktive Normalformen wieder herzustellen, und um den Ausdruck $f(t)$ symbolisch auszuwerten.

Beispiel:

$$\begin{array}{ll} & \forall y_1, y_2 . y_1 = cons(x, y_2) \vee y_1 = cons(x, cons(x, y_2)) \vee y_2 = cons(x, y_3) \vee x \neq sum(y_1) \\ \overset{\sim}{\rightarrow}_{E4} & \forall y_2 . y_2 = cons(x, y_3) \vee \forall x' . x \neq x' \end{array}$$

Zu einer positiven Normalform v mit den Klauseln $w_1,\ldots,w_n$ definieren wir

$$\begin{array}{lcl} \Psi_3(w) & := & \{dm(w_i) \mid 1 \leq i \leq n\} \subseteq \mathcal{M}(\mathbb{N}) \\ \Phi_5(w) & := & (\Psi_3(w), \Phi_4(w)) \end{array}$$

Die Ordnung $\preceq_4$ sei nun die lexikographische Kombination von $\leq$ und $\prec_3$. Wir wir gesehen haben, gilt $w' \prec_4 w$ falls $w \overset{\sim}{\rightarrow}_{E4} w'$, d.h. $\overset{\sim}{\rightarrow}_{E4}$ ist noethersch. Die Normalformen von $\overset{\sim}{\rightarrow}_{E4}$ enthalten nur Klauseln ohne Parameter.

Um nun den Wert der Funktion $\phi(w)$ für eine Formel $w \in \Sigma_2^m \mathcal{W}ff(\Sigma_E, X) \cap \downarrow_{prim}$ zu berechnen, bilden wir zunächst eine positive Normalform w' von w und erzeugen daraus eine Normalform bzgl. $\overset{\sim}{\rightarrow}_{E4}$. Die so erhaltene Formel ist nun eine Disjunktion von existenzquantifizierten Formeln, die wir schließlich in eine m-Prenexnormalform umformen. □

Falls ein Modul keine durch rekursive Programme definierten Funktionszeichen enthält, so liefert die oben beschrieben Methode trivialerweise ebenfalls ein Verfahren zur Berechnung einer spab:

Korollar 6 *Sei $\Im$ ein Domainoperator, m ein Modul mit Signatur (Σ_P, Σ_E) und $NF_m = \emptyset$. Dann gibt es eine totale berechenbare Funktion $\phi: \mathcal{S}en_m \to \mathcal{P}Sen_m$, so daß für alle w $\phi(w)$ eine $\Im, m$-spab von w ist.*

Unser Lösungsverfahren setzt voraus, daß das Modul höchstens eine durch ein rekursives Programm definierte Funktion besitzt. Wir zeigen nun abschließend, daß bei Anwesenheit von *zwei* rekursiv definierten Funktionen, die aber jeweils für sich genommen die beiden Bedingungen von Satz 6 erfüllen, eine spab nicht mehr notwendig existiert.

Es sei m das Modul aus Abbildung 1, Seite 4. Betrachte die Formel

$$w = \exists l \in list.f(suml(l)) = f(sumr(l))$$

Σ_P bezeichne die Parametersignatur von m. Wir nehmen an, v wäre eine $\Im^f, m$-spab von w. Für jeden Term $t \in \mathcal{K}t_m(list)$ der Form

$$t = cons(x_1, cons(x_2, \ldots, cons(x_{n-1}, x_n)\ldots))$$

sei v_t die Formel

$$\exists x_1, \ldots, x_n \in elem.f(x_1 + (x_2 + \ldots + (x_{n-1} + x_n)\ldots)) = f((\ldots(x_n + x_{n-1}) + \ldots + x_2) + x_1)$$

Man sieht nun leicht ein, daß

- jedes v_t eine $\Im^f, m$-spab von w ist und daß
- für $A \in \Im_{\Sigma_P}$ $A\ [m|\!=\ w$ genau dann gilt, wenn $A \models v_t$ für ein $t \in \mathcal{K}t_m(list)$

Es ist nun für jede endliche Teilmenge $N \subseteq \mathcal{K}t_m(list)$ die Menge $\{v\} \cup \{\neg v_t \mid t \in N\}$ konsistent in $\Im_{\Sigma_P}$. Um eine Modell hierfür zu konstruieren, sei m die maximale Anzahl der Vorkommen von *cons* in einem Term aus N. Dann ist die Algebra $\mathcal{T}(\Sigma_P \cup \{0\}, \emptyset)/\{e\}$ ein Modell, wobei e die folgende Gleichung ist:

$$f\underbrace{(0 + (0 + \ldots + (0 + 0)\ldots))}_{m+2 \text{ mal } +} = f\underbrace{((\ldots(0 + 0) + \ldots + 0) + 0)}_{m+2 \text{ mal } +}$$

Auf Grund der Kompaktheit von $\Im_{\Sigma_P}$ muß dann auch die Menge

$$\{v\} \cup \{\neg v_t \mid t \in \mathcal{K}t_m(list)\}$$

ein Modell in $\Im_{\Sigma_P}$ haben, dies widerspricht aber unserer Annahme an v.

7 Reduktionen

Bisher haben wir uns für die Existenz von schwächsten Parameterbedingungen von Sätzen bezüglich gegebener Module und Domainoperatoren interessiert. In diesem Kapitel wollen wir der Frage nachgehen, inwiefern es möglich ist, Module, Domainoperatoren und Sätze so zu transformieren, daß die Frage der Existenz einer schwächsten Parameterbedingung davon nicht berührt wird.

Dazu definieren wir in Abschnitt 7.1 den Begriff des Parameterbedingungenproblems (abgekürzt PbP). Wir geben an, was wir unter einer Lösung eines PbP verstehen, und formulieren das Konzept der Reduktion eines PbP's auf ein anderes. Dies ermöglicht uns einen Vergleich verschiedener PbP's bezüglich ihrer Lösbarkeit.

In Abschnitt 7.2 zeigen wir dann die Reduzierbarkeit eines PbP's auf ein daraus abgeleitetes PbP, in dem alle neuen Funktionen symbolisch auswertbar sind und für das alle Parameterfunktionen strikt sind.

7.1 Grundlegende Definitionen

Definition 31 *Ein* Parameterbedingungen-Problem *(abgekürzt PbP) ist ein Tripel* $(\Im, m, W)$ *wobei* $\Im$ *ein Domainoperator ist,* m *ein Modul und* $W \subseteq \mathcal{S}en_m$.

Mit einem solchen PbP verbinden wir die folgende Aufgabenstellung:

Finde eine partielle berechenbare Funktion $W \rightsquigarrow \mathcal{P}Sen_m$, die zu jedem Satz aus W eine $\Im, m$-spab berechnet, sofern eine existiert.

Definition 32 *Eine* Lösung *eines PbP* $(\Im, m, W)$ *ist eine partielle berechenbare Funktion* $\phi: W \rightsquigarrow \mathcal{P}Sen_m$, *so daß für alle* $w \in \mathcal{S}en_m$:

- *falls* $\phi(w)$ *undefiniert ist,dann hat* w *keine* $\Im, m$*-spab und*
- *falls* $\phi(w)$ *definiert ist, dann ist* $\phi(w)$ *eine* $\Im, m$*-spab von* w.

PbP's sind komplexe Gebilde, die Frage der Lösbarkeit eines PbP's hängt sowohl von den Eigenschaften des Domainoperators, des Moduls und der betrachteten Menge von Formeln ab. Wir möchten daher eine Struktur in diese Klasse von Problemen bringen, die uns einen Vergleich bezüglich der Lösbarkeit ermöglicht. Wir suchen dazu nach einem sinnvollen Begriff der *Reduktion* eines PbP's auf ein anderes. Eine Reduktion eines PbP P_1 auf ein anderes PbP P_2 soll uns dabei eine Konstruktion liefern, mit der wir eine Lösung von P_1 aus einer Lösung für P_2 erhalten können.

Definition 33 *Ein PbP* $(\Im, m, W)$ *heißt* reduzierbar *auf ein PbP* $(\Im', m', w')$ *via* f, g, *in Zeichen* $(\Im, m, W) \leq (\Im', m', W')$, *falls es totale berechenbare Funktionen* $f: W \to W'$ *und* $g: \mathcal{P}Sen_{m'} \to \mathcal{P}Sen_m$ *gibt, so daß*

1. *Falls $f(w)$ keine $\Im', m'$-spab hat, dann hat auch w keine $\Im, m$-spab.*
2. *Falls v' eine $\Im', m'$-spab von $f(w)$ ist, dann ist $g(v')$ eine $\Im, m$-spab von w.*

Falls $P_1 \leq P_2$ und $P_2 \leq P_1$, dann sind P_1 und P_2 äquivalent.

Lemma 38 *Falls die Funktion $\phi: W' \rightsquigarrow \mathcal{P}Sen_{m'}$ eine Lösung des PbP's $(\Im', m', W')$ ist und $(\Im, m, W) \leq (\Im', m', W')$ via f, g, dann ist $g \circ \phi \circ f$ eine Lösung von $(\Im, m, W)$.*

Wir geben nun ein Kriterium für die Reduzierbarkeit eines PbP's auf ein anderes an, das hilfreich ist, wenn wir eine geeignete Beziehung zwischen den Parameteralgebren zur Verfügung haben.

Lemma 39 *Es seien $(\Im, m, W)$ und $(\Im', m', W')$ zwei PbP's und es existieren*

- *eine Relation Φ zwischen $\Im_{\Sigma_{P_m}}$ und $\Im'_{\Sigma_{P_{m'}}}$*
- *ein Satz $p \in \mathcal{P}Sen_{m'}$*
- *eine totale berechenbar Funktion $\mathrm{F}\langle\rangle: W \cup \mathcal{P}Sen_m \to W' \cup \mathcal{P}Sen_{m'}$ mit $\mathrm{F}\langle \mathcal{P}Sen_m \rangle \subseteq \mathcal{P}Sen_{m'}$*
- *eine totale berechenbare Funktion $\mathrm{G}\langle\rangle: \mathcal{P}Sen_{m'} \to \mathcal{P}Sen_m$*

so daß

1. *für alle $B \in \Im'_{\Sigma_{P_{m'}}}$: $B \models p$ genau dann, wenn es ein $A \in \Im_{\Sigma_{P_m}}$ gibt mit $A \Phi B$.*
2. *für alle $A \in \Im_{\Sigma_{P_m}}$ und $w \in W \cup \mathcal{P}Sen_m$:*
   A $[m\models$ w genau dann, wenn für alle $B \in A\Phi$: B $[m'\models$ $\mathrm{F}\langle w \rangle$.
3. *für alle $A \in \Im_{\Sigma_{P_m}}$ und $v' \in \mathcal{P}Sen_{m'}$:*
   $A \models \mathrm{G}\langle v' \rangle$ genau dann, wenn für alle $B \in A\Phi$: $B \models v'$.
4. *für alle $A \in \Im_{\Sigma_{P_m}}$, $B_1, B_2 \in A\Phi$ und $w \in W \cup \mathcal{P}Sen_m$:*
   B_1 $[m'\models$ $\mathrm{F}\langle w \rangle$ genau dann, wenn B_2 $[m'\models$ $\mathrm{F}\langle w \rangle$

Dann ist $(\Im, m, W)$ reduzierbar auf $(\Im', m', W',)$ via f, G, wobei $f\langle w \rangle := \mathrm{F}\langle w \rangle \wedge p$.

Die Bedingung (4) folgt dabei bereits aus (2), falls $\mathrm{F}\langle \neg w \rangle = \neg \mathrm{F}\langle w \rangle$ für alle w gilt.

Beweis: Wir überprüfen die beiden Bedingungen aus Definition 33.

(1): Hier zeigen wir die Kontraposition: Falls v eine $\Im, m$-spab von w ist, dann ist $\mathrm{F}\langle v \rangle \wedge p$ eine $\Im', m'$-spab von $\mathrm{F}\langle w \rangle \wedge p$.

Sei $B \in \Im'_{\Sigma_{P_{m'}}}$ mit $B \models \mathrm{F}\langle v \rangle \wedge p$. Wegen (1) gibt es ein $A \in \Im_{\Sigma_{P_m}}$ mit $A \Phi B$. Damit gilt wegen (4) für alle $B' \in A\Phi$: $B' \models \mathrm{F}\langle v \rangle$, also ist wegen (2) $A \models v$. Wegen der Voraussetzung an v gilt A $[m\models$ w, und wegen (2) B $[m'\models$ $\mathrm{F}\langle w \rangle$, also auch B $[m'\models$ $\mathrm{F}\langle w \rangle \wedge p$.

Sei $B \in \Im'_{\Sigma_{Pm'}}$ mit $B\ [m'\models \mathrm{F}\langle w\rangle \wedge p$. Wegen (1) gibt es ein $A \in \Im_{\Sigma_{Pm}}$ mit $A\Phi B$. Damit gilt wegen (4) für alle $B' \in A\Phi$: $B'\ [m'\models \mathrm{F}\langle w\rangle$, also ist wegen (2) $A\ [m\models w$. Wegen der Voraussetzung an v gilt $A \models v$, und wegen (2) $B \models \mathrm{F}\langle v\rangle$, also auch $B \models \mathrm{F}\langle v\rangle \wedge p$.

(2): Es sei nun v' eine $\Im', m'$-spab von $\mathrm{F}\langle w\rangle \wedge p$. Wir zeigen, daß $\mathrm{G}\langle v'\rangle$ eine $\Im, m$-spab von w ist.

Sei $A \in \Im_{\Sigma_{Pm}}$ mit $A \models \mathrm{G}\langle v'\rangle$. Nach (3) gilt dann für alle $B \in A\Phi$: $B \models v'$, also nach Vorausetzung auch $B\ [m'\models \mathrm{F}\langle w\rangle \wedge p$. Wegen (2) gilt also $A\ [m\models w$.

Andererseits sei $A \in \Im_{\Sigma_{Pm}}$ mit $A\ [m\models w$. Nach (3) gilt dann für alle $B \in A\Phi$: $B\ [m'\models \mathrm{F}\langle w\rangle$, und weiter wegen (1): $B\ [m'\models \mathrm{F}\langle w\rangle \wedge p$. Also gilt nach Voraussetzung für alle $B \in A\Phi$: $B \models v'$, und somit wegen (3): $A \models \mathrm{G}\langle v'\rangle$. □

Unsere erste Anwendung zeigt, daß die Beschränkung auf Algebren von maximal abzählbar unendlicher Kardinalität die Suche nach spabs nicht erleichtert:

Korollar 7 *Für einen Domainoperator $\Im$ sei $\Im^\omega$ die Einschränkung von $\Im$ auf abzählbare Algebren, das heißt:*

$$\Im^\omega_\Sigma := \{A \in \Im_\Sigma \mid \#(A) \leq \aleph_0\}$$

Für jeden Domainoperator $\Im$ und jedes Modul m sind $(\Im, m, \mathcal{S}en_m)$ und $(\Im^\omega, m, \mathcal{S}en_m)$ äquivalent.

Beweis: Wir benutzen Lemma 39.

$(\Im, m, \mathcal{S}en_m) \leq (\Im^\omega, m, \mathcal{S}en_m)$ Wir wählen die Relation Φ als

$$A\Phi B \qquad :\Leftrightarrow \qquad B \preceq A$$

(zur Erinnerung: $B \preceq A$ bedeutet: B ist eine elementare Teilalgebra von A). Weiter wählen wir $p :=$ `True` und für F und G jeweils die identische Abbildung. Bedingung (1) ist erfüllt, da jede Algebra eine elementare Teilalgebra von sich selbst ist. Die restlichen Eigenschaften folgen aus der Tatsache, daß jede Algebra die gleiche Theorie wie ihre elementaren Teilalgebren hat.

$(\Im, m, \mathcal{S}en_m) \geq (\Im^\omega, m, \mathcal{S}en_m)$ Hier wählen wir F, G und p wie oben und $\Phi := \preceq$ als die elementare Teilalgebrarelation. Bedingung (1) ist eine Folgerung aus dem Abschlußß von $\Im$ unter elementaren Teilmodellen. □

Das nächste Korollar drückt aus, daß wir die Berechenbarkeit von spabs auch als Reduktion auf das triviale Modul ausdrücken können:

Korollar 8 *Es $\Im$ ein Domainoperator, m ein Modul und $W \subseteq \mathcal{S}en_m$. Dann ist $(\Im, m, W) \leq 1_{\Sigma_{Pm}}$ genau dann, wenn es eine totale berechenbare Funktion $\phi: W \to \mathcal{P}Sen_m$ gibt, so daß $\phi(w)$ eine $\Im, m$-spab von w ist für alle $w \in W$.*

Beweis: klar nach Definition von Reduzierbarkeit und Lemma 25. □

7.2 Vermeidung der Nichtterminierung

Wir werden nun zeigen, daß wir uns stets auf PbP's zurückziehen können, in denen alle Parameterfunktionen strikt und total sind, und in denen die neuen Funkionen partiell symbolisch auswertbar sind. Insbesondere ist damit der undefinierte Träger für die Betrachtung der Logik nicht mehr wesentlich. Dabei müssen wir zwei mögliche "Ursachen" für die Nicht-Striktheit von Funktionen angehen: Die eine Quelle von Nichtterminierung ist das Auftreten von nicht strikten Funktionen in der Parameteralgebra. Das zweite Problem liegt in der allgemeinen Natur der in einem Modul auftretenden Programme. Diese Programme sind möglicherweise nicht symbolisch auswertbar, das heißt, daß ihre Rekursionsstruktur von den Eigenschaften der aktuellen Parameteralgebra abhängt. Die beiden Probleme sind nicht ganz unabhängig voneinander, denn durch die Verwendung von nicht strikten Parameterfunktionen, insbesondere bei nicht sequentiellen Funktionen wie dem parallelen *oder*, wird die Behandlung der Rekursionsstruktur der Programme erschwert.

Das zweite Problem (möglicherweise unbeschränkte Rekursion der Programme) lösen wir, indem wir die neuen Funktionen um einen Zähler für die Rekursionstiefe anreichern. Bei jedem Aufruf einer neuen Funktion im Rumpf eines rekursiven Programmes wird ein um 1 dekrementierter Wert des Zählers übergeben. Ein Funktionsaufruf mit dem Wert 0 des Rekursionszählers kann selbst keine neuen Funktionen mehr aufrufen. Falls die Auswertung des Programmrumpfes doch einen weiteren Funktionsaufruf notwendig machen sollte, dann liefert die Funktion einen "Unsinns"-Wert zurück und terminiert. Um jetzt zwischen "richtigen" und "unsinnigen" Werten unterscheiden zu können, fügen wir ganz analog Funktionen hinzu, die testen, ob eine der neuen Funktionen in einer vorgegebenen Anzahl von Rekursionsschritten ordnungsgemäß terminiert. Aus der Sicht der Logik müssen wir nun über alle möglichen Werte der Schrittzähler quantifizieren, um eine Aussage über dem alten Modul in eine äquivalente Aussage über dem neu konstruierten Modul zu übersetzen. Diese Vorgehensweise ist ähnlich zur Umwandlung von allgemein rekursiven Programmen in ihre Kleene Normalform (siehe [Rog87]).

Um nun die nicht strikten Funktionen der Parameteralgebren zu eliminieren, führen wir neue *bool*-wertige totale Parameterfunktionen ein, deren Wert jeweils besagt, ob eine Anwendung einer Parameterfunktion auf ein Tupel von Argumenten einen definierten Träger liefert oder nicht. Diese Definition mag auf den ersten Blick verwirrend scheinen, denn eine solche Funktion wird im allgemeinen nicht berechenbar sein. Unser Problem liegt hier aber nur in der Berechnung von neuen Funktionen, deren Programm in dem Modul gegeben ist. Bei den Parameterfunktionen findet keine *Berechnung*, sondern eine *Auswertung* statt. Aus diesem Grunde sind, bei Betrachtung der Parameterfunktionen, die undefinierten Träger nur irgendwelche Werte in der Parameteralgebra (wobei zusätzlich alle Funktionen monoton bezüglich dieser undefinierten Träger sind). Nur aus der Sicht der neuen Funktionszeichen sind die undefinierten Träger wirklich das Resultat einer nicht terminierenden Berechnung.

Satz 7 *Zu jedem Modul m kann man ein Modul m' konstruieren, so daß*

$$(\Im^f, m, \mathcal{S}en_m) \leq (\Im^{st}, m', \mathcal{S}en_{m'})$$

und alle Funktionen $f \in NF_{m'}$ sind symbolisch auswertbar.

Zum Beweis dieses Satzes verwenden wir wiederum Lemma 39. Zunächst konstruieren wir das Modul m' und beweisen einige Eigenschaften, die die Verbindung zwischen m und m' herstellen. Gegeben sei also das Modul

$$m = (PS, PF, NS, K, NF, EF, PR)$$

Dann definieren wir das Modul m' als

$$m' = (PS', PF', NS', K', NF', EF', PR')$$

Wir geben nun die in der Definition von m' verwendeten Komponenten an:

$$\begin{array}{lcl} PS' & := & PS \\ PF' & := & PF \\ & \cup & \{dummy_s : \rightarrow s \mid s \in PS\} \\ & \cup & \{term\bot f_I : s_{i_1}, \ldots, s_{i_k} \rightarrow bool \mid f : s_1, \ldots, s_n \rightarrow s \in PF, f \neq \texttt{ifthenelse}_s, \\ & & \quad I = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}\} \end{array}$$

Beide Module haben die gleichen Parametersorten. Die Parameterfunktionen von m' sind die von m zuzüglich jeweils eines Konstantenzeichen $dummy_s$ zu jeder Sorte s und jeweils einer *bool*-wertigen Funktion $term\bot f_I$ zu jeder Parameterfunktion f von m und jeder Teilmenge I der Menge der Argumentpositionen von f.

Wir werden jetzt bereits die Definition der Relation Φ angeben, um unsere Wahl von PS' und PF' zu motivieren. Die Grundidee, die hinter einer Aussage $A \Phi B$ für $A \in \Im^f{}_{(PS,PF)}$ steckt, ist die folgende: Die Trägermengen von A und B sind die gleichen, daraus folgt, daß auch die zugehörigen Mengen von Variablenbelegungen $?_A$ und $?_B$ gleich sind. Eine Parameterfunktion f wird in B als eine strikte und totale Variante von f^A interpretiert. Dazu definieren wir den Wert von f^B zunächst einmal als $\bot$, falls eines seiner Argumente $\bot$ ist. Um f^B jetzt auch noch total zu machen, wählen wir für den Wert einer Funktionsanwendung von f in B den Wert von $dummy$ in B, falls die entsprechende Funktionsanwendung in A den Wert $\bot$ liefern würde. Dies ist ein Grund für die Einführung der Konstantenzeichen $dummy$: sie dienen dazu, irgendwelche von $\bot$ veschiedenen Werte zu liefern, die wir zur "Totalisierung" der Funktionen verwenden können. Bei diesem Übergang von A zu B wird die Semantik von F allerdings verfälscht, denn wir wissen von einem definierten Wert einer Funktionsanwendung in B zunächst einmal nicht, ob er dem Wert in A entspricht oder durch die "Totalisierung" erzwungen wurde. Aus diesem Grunde führen wir die Funktionen $term\bot f_I$ ein, ihre Interpretation in B gibt uns nun Auskunft darüber, für welche Argumente die Funktion f^A terminiert.

Um dies formal auszudrücken, benötigen wir einige technische Abkürzungen: Es sei $(a_1, \ldots, a_n)$ ein Tupel von Trägern aus A und $(t_1, \ldots, t_n)$ ein Tupel von Termen aus $\mathcal{T}(\Sigma_{Am'}, X)$, wobei s_i jeweils die Sorte von a_i, bzw. t_i ist. K sei ein Teilmenge von $\{1, \ldots, n\}$. Dann bezeichnet für $1 \leq j \leq n$:

$$\dagger_j^K \langle a_j \rangle \quad := \quad \begin{cases} a_j & \text{falls } j \in K \\ \bot^A_{s_j} & \text{anderenfalls} \end{cases}$$

$$\ddagger_j^K\langle t_j\rangle \quad := \quad \begin{cases} t_j & \text{falls } j \in K \\ dummy_{s_j} & \text{anderenfalls} \end{cases}$$

Wir definieren also: $A \Phi B$ genau dann, wenn für alle Sorten $s \in PS = PS'$:

$$\begin{aligned} s^B &:= s^A \\ dummy_s^B &\in s^B \setminus \{\bot_s^B\} \end{aligned}$$

und für alle Funktionszeichen $f\colon s_1, \ldots, s_n \to s \in PF$ mit $f \neq \texttt{ifthenelse}_s$ und Indexmengen $I = \{i_1, \ldots, i_K\} \subseteq \{1, \ldots, n\}$:

$$f^B(x_1, \ldots, x_n) \quad := \quad \begin{cases} f^A(x_1, \ldots, x_n) & \text{falls alle } x_i \neq \bot \\ & \text{und } f^A(x_1, \ldots, x_n) \neq \bot_s^A \\ dummy_s^B & \text{falls alle } x_i \neq \bot \\ & \text{und } f^A(x_1, \ldots, x_n) = \bot_s^A \\ \bot_s^A & \text{anderenfalls} \end{cases}$$

$$term\bot f_I^B(x_{i_1}, \ldots, x_{i_K}) \quad := \quad \begin{cases} true & \text{falls alle } x_{i_j} \neq \bot \text{ und } f^A(\ddagger_1^I\langle x_1\rangle, \ldots, \ddagger_n^I\langle x_n\rangle) \neq \bot \\ false & \text{falls alle } x_{i_j} \neq \bot \text{ und } f^A(\ddagger_1^I\langle x_1\rangle, \ldots, \ddagger_n^I\langle x_n\rangle) = \bot \\ \bot & \text{falls } x_{i_j} = \bot \text{ für ein } i_j \end{cases}$$

Die neuen Sorten von m' sind die von m zuzüglich einer Sorte *counter*, die dazu dienen wird, die Rekursionstiefe einer Berechnung darzustellen.

$$\begin{aligned} NS' &:= NS \cup \{counter\} \\ K' &:= K \cup \{0\colon \to counter, inc\colon counter \to counter\} \end{aligned}$$

In m' sind die neuen Funktionszeichen f aus m nun durch jeweils zwei Funktionen $term\bot f$ und $eval\bot f$ ersetzt. Jede dieser Funktionen erhält einen *counter*-Parameter, n "Daten-Parameter", die den Parametern der zugehörigen Funktion f aus m entsprechen, sowie für jeden der Datenparameter einen zusätzlichen Parameter der Sorte *bool*. Die *bool*-wertige Funktion $term\bot f^{[\![m']\!](B)}(c, \bar{b}, \bar{v})$ gibt uns nun Auskunft darüber, ob die c-te Iteration von f in $[\![m]\!](A)$ terminiert, wobei das i-te Argument von f jeweils v_i ist, falls $d_i = true$, und $\bot$, falls $d_i = false$. Falls dies der Fall ist, dann liefert $eval\bot f^{[\![m']\!](B)}(c, \bar{b}, \bar{v})$ den Wert dieser Funktionsanwendung von f^A, anderenfalls erhalten wir einen nichtssagenden, aber definierten Wert (den wir mit Hilfe der *dummy*-Konstanten konstruieren). Das weiter unten angegebene Lemma 44 gibt eine exakte Formulierung dieses Zusammenhangs.

$$\begin{aligned} NF' &:= \{term\bot f\colon counter, \overbrace{bool, \ldots, bool}^{n}, s_1, \ldots, s_n \to bool \mid f\colon s_1, \ldots, s_n \to s \in NF\} \\ &\cup \{eval\bot f\colon counter, \overbrace{bool, \ldots, bool}^{n}, s_1, \ldots, s_n \to s \mid f\colon s_1, \ldots, s_n \to s \in NF\} \\ EF' &:= EF \setminus NF \\ &\cup \{term\bot f \mid f \in EF \cap NF\} \end{aligned}$$

$$
\begin{array}{rcl}
 & \cup & \{eval\bot f \mid f \in EF \cap NF\} \\
PR & := & \{term\bot f(s,\bar{d},\bar{x}) \Leftarrow \mathrm{t}\langle s, body, \bar{d}, \bar{x}\rangle \mid f(\bar{x}) \Leftarrow body \in PR\} \\
 & \cup & \{eval\bot f(s,\bar{d},\bar{x}) \Leftarrow \mathrm{v}\langle s, body, \bar{d}, \bar{x}\rangle \mid f(\bar{x}) \Leftarrow body \in PR\}
\end{array}
$$

Der eigentliche "Kern" der Definition von m' besteht aus den Funktionen $\mathrm{t}\langle\cdots\rangle$ und $\mathrm{v}\langle\cdots\rangle$, diese sind in den Abbildungen 28 und 29 definiert. Dabei benutzen wir die folgenden Abkürzungen: Es sei I eine endliche Indexmenge, $(b_i)_{i\in I}$ eine Menge von Termen der Sorte *bool* und $(t_i)_{i\in I}$ eine Menge von Termen einheitlicher Sorte s.

$$
\mathrm{and}\langle I,(b_i)_{i\in I}\rangle := \begin{cases} true & \text{falls } I=\emptyset \\ \texttt{if } b_{i_0} \texttt{ then } \mathrm{and}\langle I\setminus\{i_0\},(b_i)_{i\in I}\rangle & \\ \quad \texttt{else } false & \text{falls } i_0 \in I \end{cases}
$$

$$
\mathrm{or}\langle I,(b_i)_{i\in I}\rangle := \begin{cases} false & \text{falls } I=\emptyset \\ \texttt{if } b_{i_0} \texttt{ then } true & \\ \quad \texttt{else } \mathrm{or}\langle I\setminus\{i_0\},(b_i)_{i\in I}\rangle & \text{falls } i_0 \in I \end{cases}
$$

$$
\mathrm{case}\langle I,(b_i)_{i\in I},(t_i)_{i\in I}\rangle := \begin{cases} dummy_s & \text{falls } I=\emptyset \\ \texttt{if } b_{i_0} \texttt{ then } t_{i_0} \texttt{ else} & \\ \quad \mathrm{case}\langle I\setminus\{i_0\},(b_i)_{i\in I},(t_i)_{i\in I}\rangle & \text{falls } i_0 \in I \end{cases}
$$

Weiterhin definieren wir $t_1 and t_2$ als Abkürzung für $\mathrm{and}\langle\{1,2\},(t_i)\rangle$. Ein Tupel von Werten oder Termen wie $(x_1,\ldots,x_n)$ werden wir im folgenden auch in "vektorisierter" Form $\bar{x}$ abkürzen, falls der Zusammenhang klar ist.

Unsere Definitionen von and, or und case legen die Auswahl des Indexes i_0 im Rekursionsfall nicht fest und liefern deshalb auch kein eindeutiges Ergebnis. Wir werden aber keinerlei Annahmen über die Reihenfolge der verschiedenen Teilterme eines mittels and, usw. definierten Termes machen, so daß wir hier keine deterministische Definition benötigen.

Die Definitionen von $\mathrm{t}\langle\cdots\rangle$ und $\mathrm{v}\langle\cdots\rangle$ sind in zweifacher Hinsicht etwas salopp: Zunächst einmal hängen die Definition von $\mathrm{t}\langle\cdots\rangle$ and $\mathrm{v}\langle\cdots\rangle$ natürlich von dem zugrundeliegenden Modul ab, wir unterdrücken den Index m aber, um die Notationen nicht zu überladen. Ein wichtigerer Einwand ist, daß wir $\mathrm{v}\langle\cdots\rangle$ und $\mathrm{t}\langle\cdots\rangle$ genau genommen als *Familien* von Funktionen hätten definieren müssen. Das heißt, falls $\Delta = \Sigma_{Am}$ und falls die x_i paarweise verschiedene Variablen sind, dann sollte beispielsweise eine Funktion $\mathrm{v}_{(x_1,\ldots,x_n)}$ die folgende Stelligkeit haben:

$$
\mathcal{T}(\Delta,X)_{counter}, \mathcal{T}(\Delta,\{x_1,\ldots,x_n\}), (\mathcal{T}(\Delta,X)_{bool})^n, \mathcal{T}(\Delta,X)_{s_1},\ldots,\mathcal{T}(\Delta,X)_{s_n} \to \mathcal{T}(\Delta,X)
$$

wobei s_i jeweils die Sorte der Variablen x_i ist. Der Index $(x_1,\ldots,x_n)$ würde somit eine Reihenfolge der freien Variablen des zweiten Argumentes fixieren. Damit wäre auch der Zusammenhang zwischen den Datenparametern und den boolschen Parametern festgelegt. Auch hier unterdrücken wir wieder zur schreibtechnischen Vereinfachung den Index und setzen eine gegebene Ordnung der freien Variablen voraus.

Wir illustrieren an einem kleinen Beispiel die Konstruktion von m'. Dazu sei m das Modul aus Abbildung 27, Seite 110. Wir zeigen hier nur die Konstruktion des Programmes von $eval\bot g$.

PAR **SORTS** s
OPNS $f\!:\!s \to s$
$p\!:\!s \to bool$
BODY **FCTS** $g\!:\!s \to s$
PROG $g(x) \Leftarrow$ `if` $p(x)$ `then` x
`else` $g(f(x))$

Abbildung 27: Ein Beispiel für die Konstruktion nach Satz 7

Dazu betrachten wir zunächst einige Teilausdrücke, wobei wir bereits triviale Umformungen der Terme vorgenommen haben:

$$
\begin{array}{rcl}
\mathrm{t}\langle c, x, d, x\rangle & = & d \\
\mathrm{v}\langle c, x, d, x\rangle & = & x \\
\mathrm{t}\langle c, p(x), d, x\rangle & = & term{\perp}p_{\emptyset}\ or\ (d\ and\ term{\perp}p_{\{1\}}(x)) \\
\mathrm{v}\langle c, p(x), d, x\rangle & = & \texttt{if}\ term{\perp}p_{\emptyset}\ \texttt{then}\ p(dummy_{bool}) \\
 & & \texttt{else if}\ d\ and\ term{\perp}p_{\{1\}}(x)\ \texttt{then}\ p(x)\ \texttt{else}\ dummy_{bool} \\
\mathrm{t}\langle c, f(x), d, x\rangle & = & term{\perp}f_{\emptyset}\ or\ (d\ and\ term{\perp}f_{\{1\}}(x)) \\
\mathrm{v}\langle c, f(x), d, x\rangle & = & \texttt{if}\ term{\perp}f_{\emptyset}\ \texttt{then}\ f(dummy_s) \\
 & & \texttt{else if}\ d\ and\ term{\perp}p_{\{1\}}(x)\ \texttt{then}\ f(x)\ \texttt{else}\ dummy_s \\
\mathrm{t}\langle c, g(f(x)), d, x\rangle & = & \texttt{if}\ \mathrm{is}_0?(c)\ \texttt{then}\ false \\
 & & \texttt{else}\ term{\perp}g(\mathrm{select}^1_{inc}(c), \mathrm{t}\langle c, f(x), d, x\rangle, \mathrm{v}\langle c, f(x), d, x\rangle) \\
\mathrm{v}\langle c, g(f(x)), d, x\rangle & = & \texttt{if}\ \mathrm{is}_0?(c)\ \texttt{then}\ dummy_s \\
 & & \texttt{else}\ eval{\perp}g(\mathrm{select}^1_{inc}(c), \mathrm{t}\langle c, f(x), d, x\rangle, \mathrm{v}\langle c, f(x), d, x\rangle)
\end{array}
$$

Das vollständige Programm für die Funktion $eval \perp f\!:\!counter, bool, s \to s$ ist in Abbildung 30, Seite 113, angegeben.

Lemma 40 *Alle neuen Funktionen in m' sind symbolisch auswertbar.*

Beweis: Dies folgt aus der Tatsache, daß jeweils für alle Funktionen $term{\perp}f$ und $eval{\perp}f$ in jedem rekursiven Aufruf das erste Argument echt kleiner wird. □

Lemma 41 *Falls $B \in \Im^{st}_{\Sigma_{P'_m}}$, dann sind alle Funktionen in $[\![m']\!](B)$ total.*

Beweis: folgt mit Lemma 27 direkt aus Lemma 40. □

$$
\begin{array}{rcl}
\mathrm{t}\langle s, x_i, \bar{d}, \bar{v}\rangle & := & d_i \qquad \text{falls } x \text{ eine Variable ist} \\
\mathrm{t}\langle s, \bot_s, \bar{d}, \bar{v}\rangle & := & false \\
\mathrm{t}\langle s, true, \bar{d}, \bar{v}\rangle & := & true \\
\mathrm{t}\langle s, false, \bar{d}, \bar{v}\rangle & := & true \\
\mathrm{t}\langle s, \mathtt{if}\ u_1\ \mathtt{then}\ u_2\ \mathtt{else}\ u_3, \bar{d}, \bar{v}\rangle & & \\
 & := & \mathrm{t}\langle s, u_1, \bar{d}, \bar{v}\rangle\ and\ \mathtt{if}\ \mathrm{v}\langle s, u_1, \bar{d}, \bar{v}\rangle \mathtt{then}\ \mathrm{t}\langle s, u_2, \bar{d}, \bar{v}\rangle \\
 & & \mathtt{else}\ \mathrm{t}\langle s, u_3, \bar{d}, \bar{v}\rangle \\
\mathrm{t}\langle s, u_1 =_s u_2, \bar{d}, \bar{v}\rangle & := & \mathrm{t}\langle s, u_1, \bar{d}, \bar{v}\rangle\ and\ \mathrm{t}\langle s, u_2, \bar{d}, \bar{v}\rangle \\
\mathrm{t}\langle s, c(u_1, \ldots, u_n), \bar{d}, \bar{v}\rangle & := & \mathrm{and}\langle\{1, \ldots, n\}, (\mathrm{t}\langle s, u_i, \bar{d}, \bar{v}\rangle)_i\rangle \qquad \text{falls } c \in K \\
\mathrm{t}\langle s, \mathtt{select}_k^j(u), \bar{d}, \bar{v}\rangle & := & \mathrm{t}\langle s, u, \bar{d}, \bar{v}\rangle\ and\ \mathrm{v}\langle s, \mathtt{is}_k?(u), \bar{d}, \bar{v}\rangle \\
\mathrm{t}\langle s, \mathtt{is}_k?(u), \bar{d}, \bar{v}\rangle & := & \mathrm{t}\langle s, u, \bar{d}, \bar{v}\rangle \\
\mathrm{t}\langle s, f(u_1, \ldots, u_n), \bar{d}, \bar{v}\rangle & := & \mathtt{if}\ \mathtt{is}_0?(s)\ \mathtt{then}\ false\ \mathtt{else} \\
 & & term\bot f(\mathtt{select}_{inc}^1(s),\ \mathrm{t}\langle s, u_1, \bar{d}, \bar{v}\rangle, \ldots, \mathrm{t}\langle s, u_n, \bar{d}, \bar{v}\rangle, \\
 & & \qquad \mathrm{v}\langle s, u_1, \bar{d}, \bar{v}\rangle, \ldots, \mathrm{v}\langle s, u_n, \bar{d}, \bar{v}\rangle) \\
 & & \text{falls } f \in NF \\
\mathrm{t}\langle s, f(u_1, \ldots, u_n), \bar{d}, \bar{v}\rangle & := & \mathrm{or}\langle\{I = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}\}, \mathrm{and}\langle I, \mathrm{t}\langle s, u_i, \bar{d}, \bar{v}\rangle\rangle \\
 & & and\ term\bot f_I(\mathrm{v}\langle s, u_{i_1}, \bar{d}, \bar{v}\rangle, \ldots, \mathrm{v}\langle s, u_{i_k}, \bar{d}, \bar{v}\rangle)\rangle \\
 & & \text{falls } f \in PF
\end{array}
$$

Abbildung 28: Die Definition von $\mathrm{t}\langle\cdots\rangle$

Das nächste Lemma besagt, daß wir in Meta-Termen der Art $\mathrm{t}\langle\cdots\rangle$ und $\mathrm{v}\langle\cdots\rangle$ jeweils bezüglich einer Belegung gleiche Terme gegeneinander austauschen dürfen. Dies ist nicht selbstverständlich, da die Gleichheit von Termen bzgl. einer Belegung eine *semantische* Eigenschaft ist, während die Funktionen $\mathrm{t}\langle\cdots\rangle$ und $\mathrm{v}\langle\cdots\rangle$ nur auf der *syntaktischen* Termstruktur arbeiten.

Lemma 42 *Für alle Variablenbelegungen* $\alpha \in {}^?[\![m']\!](B)$, *und alle Terme* $s_1, s_2, u, b_i, v_i, d_i, t_i \in T_\Delta$ *von jeweils entsprechender Sorte mit*

$$
\begin{array}{rcll}
[\![m']\!](B)(s_1)(\alpha) & = & [\![m']\!](B)(s_2)(\alpha) & \\
[\![m']\!](B)(b_i)(\alpha) & = & [\![m']\!](B)(d_i)(\alpha) & \textit{für alle } i \\
[\![m']\!](B)(v_i)(\alpha) & = & [\![m']\!](B)(t_i)(\alpha) & \textit{für alle } i
\end{array}
$$

gilt:

$$
\begin{array}{rcl}
[\![m']\!](B)(\mathrm{t}\langle s_1, u, \bar{b}, \bar{v}\rangle)(\alpha) & = & [\![m']\!](B)(\mathrm{t}\langle s_2, u, \bar{d}, \bar{t}\rangle)(\alpha) \\
[\![m']\!](B)(\mathrm{v}\langle s_1, u, \bar{b}, \bar{v}\rangle)(\alpha) & = & [\![m']\!](B)(\mathrm{v}\langle s_2, u, \bar{d}, \bar{t}\rangle)(\alpha)
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{v}\langle s, x_i, \bar{d}, \bar{v}\rangle & := & v_i \qquad \text{falls } x \text{ eine Variable ist} \\
\mathrm{v}\langle s, \bot_s, \bar{d}, \bar{v}\rangle & := & dummy_s \\
\mathrm{v}\langle s, true, \bar{d}, \bar{v}\rangle & := & true \\
\mathrm{v}\langle s, false, \bar{d}, \bar{v}\rangle & := & false \\
\mathrm{v}\langle s, \mathtt{if}\ u_1\ \mathtt{then}\ u_2\ \mathtt{else}\ u_3, \bar{d}, \bar{v}\rangle & & \\
 & := & \mathtt{if}\ \mathrm{v}\langle s, u_1, \bar{d}, \bar{v}\rangle\ \mathtt{then}\ \mathrm{v}\langle s, u_2, \bar{d}, \bar{v}\rangle\ \mathtt{else}\ \mathrm{v}\langle s, u_3, \bar{d}, \bar{v}\rangle \\
\mathrm{v}\langle s, u_1 =_s u_2, \bar{d}, \bar{v}\rangle & := & \mathrm{v}\langle s, u_1, \bar{d}, \bar{v}\rangle =_s \mathrm{v}\langle s, u_2, \bar{d}, \bar{v}\rangle \\
\mathrm{v}\langle s, c(u_1, \ldots, u_n), \bar{d}, \bar{v}\rangle & := & c(\mathrm{v}\langle s, u_1, \bar{d}, \bar{v}\rangle, \ldots, \mathrm{v}\langle s, u_n, \bar{d}, \bar{v}\rangle) \qquad \text{falls } c \in K \\
\mathrm{v}\langle s, \mathtt{select}_k^j(u), \bar{d}, \bar{v}\rangle & := & \mathtt{select}_k^j(\mathrm{v}\langle s, u, \bar{d}, \bar{v}\rangle) \\
\mathrm{v}\langle s, \mathtt{is}_k?(u), \bar{d}, \bar{v}\rangle & := & \mathtt{is}_k?(\mathrm{v}\langle s, u, \bar{d}, \bar{v}\rangle) \\
\mathrm{v}\langle s, f(u_1, \ldots, u_n), \bar{d}, \bar{v}\rangle & := & \mathtt{if}\ \mathtt{is}_0?(s)\ \mathtt{then}\ dummy\ \mathtt{else} \\
 & & \quad eval\bot f(\mathtt{select}_{inc}^1(s),\ \mathrm{t}\langle s, u_1, \bar{d}, \bar{v}\rangle, \ldots, \mathrm{t}\langle s, u_n, \bar{d}, \bar{v}\rangle, \\
 & & \qquad\qquad \mathrm{v}\langle s, u_1, \bar{d}, \bar{v}\rangle, \ldots, \mathrm{v}\langle s, u_n, \bar{d}, \bar{v}\rangle) \\
 & & \quad \text{falls } f \in NF \\
\mathrm{v}\langle s, f(u_1, \ldots, u_n), \bar{d}, \bar{v}\rangle & := & \mathrm{case}\langle\{I = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}\}, \mathrm{and}\langle I, \mathrm{t}\langle s, u_i, \bar{d}, \bar{v}\rangle\rangle \\
 & & \quad and\ term\bot f_I(\mathrm{v}\langle s, u_{i_1}, \bar{d}, \bar{v}\rangle, \ldots, \mathrm{v}\langle s, u_{i_k}, \bar{d}, \bar{v}\rangle), \\
 & & \quad f(\dagger_1^I\langle \mathrm{v}\langle s, u_1, \bar{d}, \bar{v}\rangle\rangle, \ldots, \dagger_n^I\langle \mathrm{v}\langle s, u_n, \bar{d}, \bar{v}\rangle\rangle)\rangle \quad \text{falls } f \in PF
\end{array}
$$

Abbildung 29: Die Definition von $\mathrm{v}\langle\cdots\rangle$

Beweis: Der Beweis ist eine einfache Induktion über die Struktur von u. Man beachte, daß die Rekursion der beiden Funktionen $\mathrm{t}\langle\cdots\rangle$ und $\mathrm{v}\langle\cdots\rangle$ nur von ihrem zweiten Argument (also in diesem Fall u) abhängt. □

Lemma 43 *Für alle Terme $s, u, t_i, d_i \in T_\Delta$ und Variablen x_i, b_i entsprechender Sorte gilt:*

$$
\begin{array}{rcl}
\mathrm{t}\langle s, u, x_1, \ldots, x_n, b_1, \ldots, b_n\rangle_{x_1 \ldots x_n\, b_1 \ldots b_n}^{t_1 \ldots t_n\, d_1 \ldots d_n} & = & \mathrm{t}\langle s, u, t_1, \ldots, t_n, d_1, \ldots, d_n\rangle \\
\mathrm{v}\langle s, u, x_1, \ldots, x_n, b_1, \ldots, b_n\rangle_{x_1 \ldots x_n\, b_1 \ldots b_n}^{t_1 \ldots t_n\, d_1 \ldots d_n} & = & \mathrm{v}\langle s, u, t_1, \ldots, t_n, d_1, \ldots, d_n\rangle
\end{array}
$$

Beweis: durch Induktion über die Struktur von u. □

Das nächste Lemma gibt nun endlich die Verbindung von Φ zu den Funktionen $\mathrm{t}\langle\cdots\rangle$ und $\mathrm{v}\langle\cdots\rangle$ an. Zunächst benötigen wir aber wieder eine Notation: Für $x \in s^B$ und $b \in \{\underline{\mathrm{true}}, \underline{\mathrm{false}}\} = bool^B \setminus \{\bot_{bool}^B\}$ definieren wir

$$
\left[\begin{array}{c} x \\ b \end{array}\right] := \begin{cases} x & \text{falls } b = \underline{\mathrm{true}} \\ \bot_s^B & \text{falls } b = \underline{\mathrm{false}} \end{cases}
$$

$$
\begin{array}{lll}
eval \perp g(c,d,x) \Leftarrow & \texttt{if} & \texttt{if } term{\perp}p_{\emptyset} \texttt{ then } p(dummy_{bool}) \\
 & & \texttt{else if } d \, and \, term{\perp}p_{\{1\}}(x) \texttt{ then } p(x) \texttt{ else } dummy_{bool} \\
 & \texttt{then} & x \\
 & \texttt{else} & \texttt{if } \mathrm{is}_0?(c) \texttt{ then } dummy_s \\
 & & \texttt{else } eval{\perp}g(\mathrm{select}^1_{inc}(c), \\
 & & \quad term{\perp}g_{\emptyset} \; or \; (d \, and \, term{\perp}f_{\{1\}}(x)), \\
 & & \quad \texttt{if } term{\perp}f_{\emptyset} \texttt{ then } f(dummy_s) \\
 & & \quad \texttt{else if } d \, and \, term{\perp}p_{\{1\}}(x) \texttt{ then } f(x) \texttt{ else } dummy_s
\end{array}
$$

Abbildung 30: Eine Beispielkonstruktion von *eval-g*

Lemma 44 *Es sei* $C \in \Im^f_{(PS,PF)}$, $D \in \Im^{st}_{(PS',PF')}$ *mit* $C \Phi D$ *und* $A := [\![m]\!](C)$, $B := [\![m']\!](D)$. *Für alle Terme* $r, b_1, \ldots, b_n, v_1 \ldots, v_n$, *alle* $m \in \mathbb{N}$ *und alle Variablenbelegungen* $\alpha \in ?^+_B$ *gilt:*

$$\left[\begin{array}{l} B(\mathrm{v}\langle inc^m(0), r, \bar{b}, \bar{v}\rangle)(\alpha) \\ B(\mathrm{t}\langle inc^m(0), r, \bar{b}, \bar{v}\rangle)(\alpha) \end{array} \right] = A(\mathrm{kleene}\langle m, r\rangle) \left(\alpha \left[x_i \leftarrow \left[\begin{array}{l} B(v_i)(\alpha) \\ B(b_i)(\alpha) \end{array} \right] \right]_{i=1,\ldots,n} \right)$$

Beweis: Man beachte zunächst, daß die Terme $\mathrm{t}\langle\cdots\rangle$ und $\mathrm{v}\langle\cdots\rangle$ das Symbol $\perp$ nicht als Teilterm enthalten. Daher gilt mit Lemma 41 für alle $\alpha \in ?^+_B$:

$$B(\mathrm{v}\langle\cdots\rangle)(\alpha) \neq \perp^B_s \quad \text{und} \quad B(\mathrm{t}\langle\cdots\rangle)(\alpha) \neq \perp^B_s$$

Wir beweisen nun die Behauptung mit noetherscher Induktion über (m, r), die benutzte noethersche Ordnung ist dabei die lexikographische Kombination von $\leq_{\mathbb{N}}$ und der Teiltermordnung $\trianglelefteq$. Im folgenden bezeichen wir mit lhs und rhs, die linke, bzw. die rechte Seite der zu beweisenden Gleichung und

$$\alpha' := \left(\alpha \left[x_i \leftarrow \left[\begin{array}{l} B(v_i)(\alpha) \\ B(b_i)(\alpha) \end{array} \right] \right]_{i=1,\ldots,n} \right)$$

$\underline{r = x_i \in X}$: Dies gilt wegen

$$\begin{array}{rcl} lhs & = & \left[\begin{array}{l} B(v_i)(\alpha) \\ B(b_i)(\alpha) \end{array} \right] \\ rhs & = & A(x_i)(\alpha') = \left[\begin{array}{l} B(v_i)(\alpha) \\ B(b_i)(\alpha) \end{array} \right] \end{array}$$

$\underline{r = \perp}$: Dies gilt wegen

$$\begin{array}{rcl} lhs & = & \left[\begin{array}{l} B(dummy)(\alpha) \\ \underline{\mathrm{false}} \end{array} \right] = \perp^B_s = \perp^A_s \\ rhs & = & A(\perp)(\alpha') = \perp^A_s \end{array}$$

$r = cb$ mit $cb \in \{true, false\}$: Dies gilt wegen

$$\begin{aligned} lhs &= \left[\begin{array}{l} B(cb)(\alpha) \\ \underline{\text{true}} \end{array}\right] = cb^B = cb^A \\ rhs &= A(cb)(\alpha') = cb^A \end{aligned}$$

$r = (r_1 =_s r_2)$: Dies gilt wegen

$$\begin{aligned} lhs &= \left[\begin{array}{l} B(\mathrm{v}\langle inc^m(0), r_1, \bar{b}, \bar{v}\rangle = \mathrm{v}\langle inc^m(0), r_2, \bar{b}, \bar{v}\rangle)(\alpha) \\ B(\mathrm{t}\langle inc^m(0), r_1, \bar{b}, \bar{v}\rangle \; and \; \mathrm{t}\langle inc^m(0), r_2, \bar{b}, \bar{v}\rangle)(\alpha) \end{array}\right] \\ rhs &= A(\mathrm{kleene}\langle m, r_1\rangle = \mathrm{kleene}\langle m, r_2\rangle)(\alpha') \end{aligned}$$

Fall a.) $B(\mathrm{t}\langle inc^m(0), r_1, \bar{b}, \bar{v}\rangle \; and \; \mathrm{t}\langle inc^m(0), r_2, \bar{b}, \bar{v}\rangle)(\alpha) = \underline{\text{true}}$.

$$\begin{aligned} lhs &= B(\mathrm{v}\langle inc^m(0), r_1, \bar{b}, \bar{v}\rangle = \mathrm{v}\langle inc^m(0), r_2, \bar{b}, \bar{v}\rangle)(\alpha) \\ &= B(\mathrm{v}\langle inc^m(0), r_1, \bar{b}, \bar{v}\rangle)(\alpha) =^B B(\mathrm{v}\langle inc^m(0), r_2, \bar{b}, \bar{v}\rangle)(\alpha) \\ &= \left[\begin{array}{l} B(\mathrm{v}\langle inc^m(0), r_1, \bar{b}, \bar{v}\rangle)(\alpha) \\ B(\mathrm{t}\langle inc^m(0), r_1, \bar{b}, \bar{v}\rangle)(\alpha) \end{array}\right] =^B \left[\begin{array}{l} B(\mathrm{v}\langle inc^m(0), r_2, \bar{b}, \bar{v}\rangle)(\alpha) \\ B(\mathrm{t}\langle inc^m(0), r_2, \bar{b}, \bar{v}\rangle)(\alpha) \end{array}\right] \\ &= A(\mathrm{kleene}\langle m, r_1\rangle)(\alpha') =^A A(\mathrm{kleene}\langle m, r_2\rangle)(\alpha') \\ &\quad \text{nach Induktionsannahme} \\ &= rhs \end{aligned}$$

Fall b.) $B(\mathrm{t}\langle inc^m(0), r_1, \bar{b}, \bar{v}\rangle \; and \; \mathrm{t}\langle inc^m(0), r_2, \bar{b}, \bar{v}\rangle)(\alpha) = \underline{\text{false}}$.

Wir können ähnlich zu Fall (a) zeigen, daß $lhs = \bot$ und $rhs = \bot$.

$r = \mathtt{if}\ r_1\ \mathtt{then}\ r_2\ \mathtt{else}\ r_3, r = c(r_1, \ldots, r_n), r = \mathtt{select}_k^j(r_1), r = \mathtt{is}_k?(r_1)$:

analog.

$r = f(r_1, \ldots, r_n), f \in NF$:

Fall a.) $m = 0$. Dann ist $lhs = \bot$ und $rhs = \bot$.

Fall b.) $m \geq 1$. Es sei $(f(x_1, \ldots, x_n) \Leftarrow body) \in PR$.

$$\begin{aligned} lhs &= \left[\begin{array}{l} B(\mathtt{if}\ \mathtt{is}_0?(inc^m(0))\ \mathtt{then}\ dummy\ \mathtt{else} \\ \quad eval \bot f(\mathtt{select}^1_{inc}(inc^m(0)), \overline{\mathrm{t}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle}, \overline{\mathrm{v}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle})(\alpha) \\ B(\mathtt{if}\ \mathtt{is}_0?(inc^m(0))\ \mathtt{then}\ false\ \mathtt{else} \\ \quad term \bot f(\mathtt{select}^1_{inc}(inc^m(0)), \overline{\mathrm{t}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle}, \overline{\mathrm{v}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle})(\alpha) \end{array}\right] \\ &\quad \text{nach Definition von t}\langle\rangle \text{ und v}\langle\rangle \\ &= \left[\begin{array}{l} B(eval \bot f(\mathtt{select}^1_{inc}(inc^m(0)), \overline{\mathrm{t}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle}, \overline{\mathrm{v}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle})(\alpha) \\ B(term \bot f(\mathtt{select}^1_{inc}(inc^m(0)), \overline{\mathrm{t}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle}, \overline{\mathrm{v}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle})(\alpha) \end{array}\right] \\ &\quad \text{nach der Semantik von } \mathtt{is}_k? \text{ und wegen } m \geq 1 \\ &= \left[\begin{array}{l} B(\mathrm{v}\langle \mathtt{select}^1_{inc}(inc^m(0)), body, \overline{\mathrm{t}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle}, \overline{\mathrm{v}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle}\rangle)(\alpha) \\ B(\mathrm{t}\langle \mathtt{select}^1_{inc}(inc^m(0)), body, \overline{\mathrm{t}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle}, \overline{\mathrm{v}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle}\rangle)(\alpha) \end{array}\right] \end{aligned}$$

nach Lemma 13 und Lemma 43

$$= \begin{bmatrix} B(\mathrm{v}\langle inc^{m-1}(0), body, \overline{\mathrm{t}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle}, \overline{\mathrm{v}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle}\rangle)(\alpha) \\ B(\mathrm{t}\langle inc^{m-1}(0), body, \mathrm{t}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle, \mathrm{v}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle\rangle)(\alpha) \end{bmatrix}$$

nach Lemma 42

$$= A(\mathrm{kleene}\langle m \perp 1, body\rangle)\left(\alpha\left[x_i \leftarrow \begin{bmatrix} B(\mathrm{v}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle)(\alpha) \\ B(\mathrm{t}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle)(\alpha) \end{bmatrix}\right]_{i=1,\ldots,n}\right)$$

nach Induktionsannahme

$$= A(\mathrm{kleene}\langle m \perp 1, body\rangle)\Big(\alpha[x_i \leftarrow A(\mathrm{kleene}\langle m, r_i\rangle)(\alpha')]_{i=1,\ldots,n}\Big)$$

nach Induktionsannahme (n mal angewandt)

$$= A(\mathrm{kleene}\langle m \perp 1, body\rangle_{x_i}^{\mathrm{kleene}\langle m, r_i\rangle})(\alpha')$$

nach dem Substitutionssatz der Prädikatenlogik

$$= rhs$$

nach Lemma 8

<u>$r = f(r_1, \ldots, r_n), f \in PF$:</u>

Fall a.) $B(\mathrm{t}\langle inc^m(0), f(r_1, \ldots, r_n), \bar{b}, \bar{v}\rangle)(\alpha) = true$.

Dann gibt es nach der Definition von t$\langle\rangle$ eine Menge $J = \{j_1, \ldots, j_{l_J}\} \subseteq \{1, \ldots, n\}$ mit

$$B(\mathrm{and}\langle J, \mathrm{t}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle\rangle)(\alpha) = true$$
$$\text{und} \quad B(term\bot f_J(\mathrm{v}\langle inc^m(0), u_{j_1}, \bar{b}, \bar{v}\rangle, \ldots, \mathrm{v}\langle inc^m(0), u_{j_{l_J}}, \bar{b}, \bar{v}\rangle))(\alpha) = true$$

Also gibt es eine (möglicherweise von J verschiedene) Menge $K = \{k_1, \ldots, k_{l_K}\} \subseteq \{1, \ldots, n\}$ mit

$$B(\mathrm{and}\langle K, \mathrm{t}\langle inc^m(0), r_i, \bar{b}, \bar{v}\rangle\rangle)(\alpha) = true \tag{38}$$
$$\text{und} \quad B(term\bot f_K(\mathrm{v}\langle inc^m(0), u_{k_1}, \bar{b}, \bar{v}\rangle, \ldots, \mathrm{v}\langle inc^m(0), u_{k_{l_K}}, \bar{b}, \bar{v}\rangle))(\alpha) = true \tag{39}$$
$$\text{und} \quad B(\mathrm{v}\langle inc^m(0), f(r_1, \ldots, r_n), \bar{b}, \bar{v}\rangle)(\alpha) = B(f(\overset{+}{\dagger}{}_1^K\langle \mathrm{v}\langle inc^m(0), u_1, \bar{b}, \bar{v}\rangle\rangle, \ldots, \overset{+}{\dagger}{}_n^K\langle \mathrm{v}\langle inc^m(0), u_n, \bar{b}, \bar{v}\rangle\rangle))(\alpha) \tag{40}$$

Damit erhalten wir

$$lhs = B(f(\overset{+}{\dagger}{}_1^K\langle \mathrm{v}\langle inc^m(0), u_1, \bar{b}, \bar{v}\rangle\rangle, \ldots, \overset{+}{\dagger}{}_n^K\langle \mathrm{v}\langle inc^m(0), u_n, \bar{b}, \bar{v}\rangle\rangle))(\alpha)$$

wegen (40)

$$= f^B(B(\overset{+}{\dagger}{}_1^K\langle \mathrm{v}\langle inc^m(0), u_1, \bar{b}, \bar{v}\rangle\rangle)(\alpha), \ldots, B(\overset{+}{\dagger}{}_n^K\langle \mathrm{v}\langle inc^m(0), u_n, \bar{b}, \bar{v}\rangle\rangle)(\alpha))$$

wegen der Semantik der Terme

$$\sqsupseteq f^A(B(\overset{+}{\dagger}{}_1^K\langle \mathrm{v}\langle inc^m(0), u_1, \bar{b}, \bar{v}\rangle\rangle)(\alpha), \ldots, B(\overset{+}{\dagger}{}_n^K\langle \mathrm{v}\langle inc^m(0), u_n, \bar{b}, \bar{v}\rangle\rangle)(\alpha))$$

wegen der Definition von f^B und da alle Argumente von $\perp$ verschieden sind

$$\sqsupseteq f^A(\overset{+}{\dagger}{}_1^K\langle B(\mathrm{v}\langle inc^m(0), u_1, \bar{b}, \bar{v}\rangle)(\alpha)\rangle, \ldots, \overset{+}{\dagger}{}_n^K\langle B(\mathrm{v}\langle inc^m(0), u_n, \bar{b}, \bar{v}\rangle)(\alpha)\rangle) \tag{41}$$

$$
\begin{array}{lll}
 & & \text{wegen der Monotonie von } f^A \\
\sqsubseteq & f^A\left(\left[\begin{array}{c} B(\mathrm{v}\langle inc^m(0), u_1, \bar{b}, \bar{v}\rangle)(\alpha) \\ B(\mathrm{t}\langle inc^m(0), u_1, \bar{b}, \bar{v}\rangle)(\alpha) \end{array}\right], \ldots, \left[\begin{array}{c} B(\mathrm{v}\langle inc^m(0), u_n, \bar{b}, \bar{v}\rangle)(\alpha) \\ B(\mathrm{t}\langle inc^m(0), u_n, \bar{b}, \bar{v}\rangle)(\alpha) \end{array}\right]\right) \\
 & & \text{wegen (38) und der Monotonie von } f^A \\
= & f^A(A(\mathrm{kleene}\langle m, u_1\rangle)(\alpha'), \ldots, A(\mathrm{kleene}\langle m, u_n\rangle)(\alpha')) \\
 & & \text{wegen der Induktionsannahme } (n \text{ mal angewandt}) \\
= & rhs
\end{array}
$$

Auf Grund von (39) und der Definition von B ist (41) verschieden von $\bot$, da wir uns in einer flachen cpo befinden, muß somit in dieser Kette sogar Gleichheit gelten, also $lhs = rhs$.

Fall b.) $B(\mathrm{t}\langle inc^m(0), f(r_1, \ldots, r_n), \bar{b}, \bar{v}\rangle)(\alpha) = false$ Dann ist $lhs = \bot$. Es sei nun

$$K = \{i_1, \ldots, i_K\} := \{i \mid 1 \leq i \leq n \text{ und } B(\mathrm{t}\langle inc^m(0), u_i, \bar{b}, \bar{v}\rangle)(\alpha) = true\}$$

Dann gilt nach der Definition von $\mathrm{t}\langle\rangle$:

$$B(term\bot f_K(\mathrm{v}\langle inc^m(0), u_{i_1}, \bar{b}, \bar{v}\rangle, \ldots, \mathrm{v}\langle inc^m(0), u_{i_K}, \bar{b}, \bar{v}\rangle))(\alpha) = false$$

Somit erhalten wir:

$$
\begin{array}{lll}
rhs & = & f^A(A(\mathrm{kleene}\langle m, u_1\rangle)(\alpha'), \ldots, A(\mathrm{kleene}\langle m, u_n\rangle)(\alpha')) \\
 & = & f^A\left(\left[\begin{array}{c} B(\mathrm{v}\langle inc^m(0), u_1, \bar{b}, \bar{v}\rangle)(\alpha) \\ B(\mathrm{t}\langle inc^m(0), u_1, \bar{b}, \bar{v}\rangle)(\alpha) \end{array}\right], \ldots, \left[\begin{array}{c} B(\mathrm{v}\langle inc^m(0), u_n, \bar{b}, \bar{v}\rangle)(\alpha) \\ B(\mathrm{t}\langle inc^m(0), u_n, \bar{b}, \bar{v}\rangle)(\alpha) \end{array}\right]\right) \\
 & & \text{nach Induktionsannahme } (n \text{ mal angewandt}) \\
 & = & f^A(\dagger_1^K\langle B(\mathrm{v}\langle inc^m(0), u_1, \bar{b}, \bar{v}\rangle)(\alpha)\rangle, \ldots, \dagger_1^K\langle B(\mathrm{v}\langle inc^m(0), u_1, \bar{b}, \bar{v}\rangle)(\alpha)\rangle) \\
 & & \text{nach Definition von } K \\
 & = & \bot \\
 & & \text{nach Definition von } term\bot f_K^B
\end{array}
$$

□

Beweis: [von Satz 7] Die Relation Φ haben wir bereits oben (Seite 107) angegeben. Wir definieren den Satz p als die Konjunktion aller Sätze der folgenden Bauart:

$$\forall x_1, y_1 \in s_1, \ldots, x_n, y_n \in s_n \,.\, term\bot f_I(x_{i_1}, \ldots, x_{i_{l_I}}) = true \wedge x_{i_1} = y_{i_1} \wedge \ldots \wedge x_{i_{l_I}} = y_{i_{l_I}} \supset f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$

$$\forall x_1 \in s_1, \ldots, x_n \in s_n \,.\, term\bot f_I(x_{i_1}, \ldots, x_{i_{l_I}}) = true \supset term\bot f_J(x_{j_1}, \ldots, x_{j_{l_J}}) = true$$

$$\forall x_1 \in s_1, \ldots, x_n \in s_n \,.\, term\bot f_I(x_{i_1}, \ldots, x_{i_{l_I}}) = false \supset f(\dagger_1^I\langle x_1\rangle, \ldots, \dagger_n^I\langle x_n\rangle) = dummy$$

wobei f jeweils eine n-stellige Parameterfunktion und $I = \{i_1, \ldots, i_{l_I}\}$ und $J = \{j_1, \ldots, j_{l_J}\}$ Teilmengen von $\{1, \ldots, n\}$ mit $I \subseteq J$ sind. Wir müssen nun die vier Bedingungen von Lemma 39 nachweisen.

(1): Auf Grund der Eigenschaften monotoner Funktionen und der Definition von Φ folgt nun sofort $B \models p$ für alle $A \in \Im_{(PS,PF)}$ und $A\Phi B$.

Andererseits sei $B \in \Im^{st}_{(PS',PF')}$ mit $B \models p$ Wir konstruieren eine Algebra $A \in \Im^{f}_{(PS,PF)}$ mit $A\Phi B$. Zunächst definieren wir $s^A := s^B$ für alle $s \in PS$. Für $f\colon s_1, \ldots, s_n \to s \in PF$ und $x_1 \in s_1^A, \ldots, x_n \in s_n^A$ sei $I = \{i_1, \ldots, i_k\} := \{i \in I \mid x_i \neq \bot^A_{s_i}\}$. Dann definieren wir

$$f^A(x_1, \ldots, x_n) := \begin{cases} \bot & \text{falls } term\bot f^B_I(x_{i_1}, \ldots, x_{i_k}) = false \\ f^B(\overset{+}{1}{}^I_1\langle x_1\rangle, \ldots, \overset{+}{1}{}^I_n\langle x_n\rangle) & \text{anderenfalls} \end{cases}$$

Man rechnet nun leicht nach, daß alle Funktionen in A monoton sind (dies ist notwendig für $A \in \Im^{f}_{(PS,PF)}$), und daß $A\Phi B$.

(2): Wir definieren nun die Funktion F von Lemma 39 durch Induktion über den Aufbau der Sätze. Ohne Beschränkung der Allgemeinheit betrachten wir nur Sätze mit Quantoren der Form $\forall x. \in s$.

$$\begin{aligned} \mathrm{F}\langle t_1 = t_2\rangle &:= (\forall c \in counter\,.\,\mathrm{t}\langle c, t_1, \overline{true}, \overline{x_1}\rangle = false \wedge \mathrm{t}\langle c, t_2, \overline{true}, \overline{x_2}\rangle = false) \\ &\quad \vee \exists c \in counter\,.(\mathrm{t}\langle c, t_1, \overline{true}, \overline{x_1}\rangle = true \wedge \mathrm{t}\langle c, t_2, \overline{true}, \overline{x_2}\rangle = true \wedge \\ &\qquad \mathrm{v}\langle c, t_1, \overline{true}, \overline{x_1}\rangle = \mathrm{v}\langle c, t_2, \overline{true}, \overline{x_2}\rangle) \\ &\quad \text{wobei } \overline{x_i} = \mathcal{V}ar\langle t_i\rangle \\ \mathrm{F}\langle \neg w\rangle &:= \neg \mathrm{F}\langle w\rangle \\ \mathrm{F}\langle w_1 \wedge w_2\rangle &:= \mathrm{F}\langle w_1\rangle \wedge \mathrm{F}\langle w_2\rangle \\ \mathrm{F}\langle \forall x \in s\,.\,w\rangle &:= \forall x \in s\,.\mathrm{F}\langle w\rangle \end{aligned}$$

Man kann nun leicht mit Hilfe von Lemma 44, Lemma 41 und Lemma 12 zeigen, daß für alle Algebren A, alle Formeln $w \in \mathcal{W}ff(\Sigma_{Em}, X)$ und alle positiven Variablenbelegungen $\alpha \in ?^{+}_{B} = ?^{+}_{A}$ gilt:

$$\text{für alle } B \in A\Phi \quad B, \alpha \models \mathrm{F}\langle w\rangle \quad \Leftrightarrow \quad A, \alpha \models w$$

Man beachte hierzu, daß auf Grund der Lemmata 41 und 44 für alle Terme t mit $\mathcal{V}ar\langle t\rangle = \overline{x}$ und alle positiven Belegungen $\alpha \in ?^{+}_{X,A}$ gilt:

$$\begin{aligned} A(\mathrm{kleene}\langle m, t\rangle)(\alpha) = \bot &\Leftrightarrow B(\mathrm{t}\langle inc^m(0), t, \overline{true}, \overline{x}\rangle)(\alpha) = false \\ A(\mathrm{kleene}\langle m, t\rangle)(\alpha) \neq \bot &\Rightarrow A(\mathrm{kleene}\langle m, t\rangle)(\alpha) = B(\mathrm{v}\langle inc^m(0), t, \overline{true}, \overline{x}\rangle)(\alpha) \end{aligned}$$

(3): Wir können eine Formel $w \in \mathcal{P}Sen_{m'}$ leicht durch Einführung zusätzlicher Quantoren in eine äquivalente Form umwandeln, in der alle Atome von der Form $x = f(x_1, \ldots, x_n)$ für Variablen $x, x_1, \ldots, x_n$ sind. Wir brauchen G also nur für Formeln dieser Bauart zu konstruieren. Zunächst definieren wir eine Hilfsfunktion H. Dabei kürzen wir ein Tupel von Variablen $x_1, \ldots, x_n$ durch $\bar{x}$ ab, $I = \{i_1, \ldots, i_k\}$ bezeichne eine Teilmenge von $\{1, \ldots, n\}$.

$$\begin{aligned} H\langle x = dummy\rangle &= x = dummy \\ H\langle x = \texttt{if}\, x_1\, \texttt{then}\, x_2\, \texttt{else}\, x_3\rangle &= x = \texttt{if}\, x_1\, \texttt{then}\, x_2\, \texttt{else}\, x_3 \end{aligned}$$

$$
\begin{aligned}
H\langle x = f(x_1,\ldots,x_n)\rangle &= ((x_1 = \bot \vee \ldots \vee x_n = \bot) \wedge x = \bot) \vee \\
&\quad (x_1 \neq \bot \wedge \ldots \wedge x_n \neq \bot \wedge ((f(\bar{x}) \neq \bot \wedge x = f(\bar{x})) \\
&\qquad \vee (f(\bar{x}) = \bot \wedge x = dummy))) \\
H\langle x = term\bot f_I(x_{i_1},\ldots,x_{i_k})\rangle &= (\bigvee_{i\in I} x_i = \bot \wedge x = \bot) \vee \\
&\quad (\bigwedge_{i\in I} x_i \neq \bot \wedge ((f(\dagger_1^I\langle x_1\rangle,\ldots,\dagger_n^I\langle x_n\rangle) \neq \bot \wedge x = true) \\
&\qquad \vee (f(\dagger_1^I\langle x_1\rangle,\ldots,\dagger_n^I\langle x_n\rangle) = \bot \wedge x = false))) \\
H\langle \neg w\rangle &= \neg H\langle w\rangle \\
H\langle w_1 \wedge w_2\rangle &= H\langle w_1\rangle \wedge H\langle w_2\rangle \\
H\langle \exists x : s.w\rangle &= \exists x : s.H\langle w\rangle
\end{aligned}
$$

Den Wert von $\mathrm{G}\langle w\rangle$ erhalten wir nun, indem wir in $H\langle w\rangle$ die Konstanten $dummy$ durch neue existenzquantifizierte Variablen ersetzen:

$$\mathrm{G}\langle w\rangle := \forall x_1 \in s_1,\ldots,x_n \in s_n.H\langle w\rangle_{dummy_{s_1}\ldots dummy_{s_n}}^{x_1\ldots x_n}$$

wobei die x_i weder frei noch gebunden in $H\langle w\rangle$ vorkommen dürfen. Die Bedingung (3) folgt dann aus der Definition von Φ.

(4): Dies folgt unmittelbar aus der Definition von $\mathrm{F}\langle\rangle$, da $\mathrm{F}\langle\neg w\rangle = \neg\mathrm{F}\langle w\rangle$ für alle $w \in \mathcal{P}Sen_m$. □

8 Unentscheidbarkeit von Th_m

Wir wollen nun eine Technik vorstellen, mit der wir die Unentscheidbarkeit von $Th_m(BOOL)$ für Module mit Parametersignatur Σ_{BOOL} beweisen können. Die Methode ist dabei nicht auf Theorien der Form $Th_m(BOOL)$ beschränkt, sondern kann dazu benutzt werden, um die Unentscheidbarkeit der Theorie $Th(I)$ irgendeines Modelles I zu beweisen. Da $Th_m(BOOL) = Th(\llbracket m \rrbracket(BOOL))$, erhalten wir dann insbesondere aus der Methode zum Beweis der Unentscheidbarkeit einer Theorie $Th(I)$ eine Methode zum Beweis der Unentscheidbarkeit einer Theorie $Th_m(BOOL)$. Wir können uns also nun auf (vollständige) Theorien im Sinne der Prädikatenlogik erster Stufe konzentrieren. Bisher haben wir uns nur mit prädikatenlogischen Sprachen, deren einziges Prädikatensymbol das Gleichheitssymbol ist, befaßt. Wir konnten daher prädikatenlogische Sprachen mit Signaturen und prädikatenlogische Modelle mit Algebren identifizieren. Diese Einschränkung wollen wir für diesen Abschnitt aufheben und somit unsere Untersuchungen auf einen allgemeineren Modellbegriff ausdehnen. Andererseits betrachten wir, aus Gründen der Vereinfachung der Darstellung, nur die unsortierte Prädikatenlogik. Die hier vorgestellte Technik kann aber leicht auf den mehrsortigen Fall ausgeweitet werden.

Wir geben eine prädikatenlogische Sprache nun als Paar $\tau = (P, F)$ an, wobei F eine Liste von Funktionszeichen der Form $\langle f(n_f), g(n_g), \ldots \rangle$ und P eine Liste von Prädikatszeichen der Form $P = \langle \oplus(n_\oplus), \odot(n_\odot), \ldots \rangle$ ist. Die Zahlen in Klammern geben dabei die Stelligkeit des Funktionszeichen, bzw. Prädikatszeichens an und sind nicht Bestandteil des Zeichens selbst. Wir verlangen nicht, daß das Gleichheitssymbol in P enthalten ist. Falls aber $=(2)$ Bestandteil von P ist, dann wird $=$ in allen Modellen dieser Sprache als das Gleichheitsprädikat interpretiert.

Die Methode basiert auf der Rückführung des Postschen Korrespondenzproblem über dem binären Alphabet $\{a, b\}$ auf die Entscheidbarkeit von $Th(I)$. Eine Instanz P des *Postschen Korrespondenzproblems* über einem endlichen Zeichenvorrat Σ ([Pos46]) ist eine endliche Menge von Paaren von Worten:

$$P = \{(p_i, q_i) \mid 0 \leq i \leq m; p_i, q_i \in \Sigma^+\}$$

Eine *P-Konstruktion* für ein Paar $(u, v) \in \Sigma^* \times \Sigma^*$ ist eine Folge $((u_j, v_j))_{j=1\ldots n}$ mit

- $u_j, v_j \in \Sigma^*$ für alle j,
- $u_1 = v_1 = \epsilon$,
- $u_n = u$ und $v_n = v$ und
- für alle $1 \leq j \leq n \perp 1$ gibt es ein $i \in \{0, \ldots, m\}$ mit $u_{j+1} = u_j p_i$ und $v_{j+1} = v_j q_i$

In diesem Fall ist (u, v) *P-konstruierbar*. P ist *lösbar*, falls es ein $u \in \Sigma^+$ gibt, so daß (u, u) P-konstruierbar ist. Die Menge der lösbaren Instanzen des Postschen Korrespondenzproblems ist unentscheidbar ([Pos46]). Unser Ziel ist es nun, zu einer Instanz P eines Postschen Korrespondenzproblems eine Formel $\underline{\texttt{solvable}}_P$ zu konstruieren, so daß $I \models \underline{\texttt{solvable}}_P$ genau dann gilt, wenn P lösbar ist. Auf Grund des konstruktiven Charakters der Definition von $\underline{\texttt{solvable}}_P$

wird damit die Unentscheidbarkeit desjenigen Fragments von $Th(I)$, das alle Sätze der Form $\underline{\texttt{solvable}}_P$ enthält, folgen. Wir betonen nochmals, daß unsere Methode nur auf Theorien von gegebenen *Modellen* im Gegensatz zu durch Axiomen definierten Theorien anwendbar ist.

Die Grundidee der Rückführung des Postschen Korrespondenzproblems auf die Theorie eines Modelles besteht in der Simulation der beiden Datentypen des Postschen Korrespondenzproblems, nämlich Worte und Folgen von Paaren von Worten, in dem Modell. Die Trägermengen der Datentypen bilden wir mit geeigneten Repräsentationsfunktionen in die Trägermenge (wir betrachten hier ja nur einortige Logik) des Modelles ab. Die Operationen der Datentypen stellen wir durch geeignete Formeln mit freien Variablen, die den Argumenten der Operationen entsprechen, dar. Wir müssen natürlich in einem gewissen Sinne die "Korrektheit" unserer Datentyp-Repräsentation beweisen, zu diesem Zwecke werden wir Bedingungen angegeben, die diese gewährleisten. Die Bedingungen verbinden die Darstellung der Trägermengen mit den Repräsentationen der Operationen. Wir können aus zwei Gründen diese Bedingungen nicht als Sätze der prädikatenlogischen Sprache, die dann in der betrachteten Theorie enthalten sein müßten, formulieren: Zunächst einmal haben wir die Repräsentation der Trägermengen semantisch definiert, dies muß keine Entsprechung in der Logik haben. Wir verlangen insbesondere *nicht*, daß der Bildbereich der Repräsentationsfunktionen durch Formeln der Prädikatenlogik beschrieben werden kann. Der zweite Grund liegt tiefer: eine unsere Bedingungen wird es sein, daß eine bestimmte Relation auf der Trägermenge noethersch ist, und dies kann nicht in der Prädikatenlogik erster Stufe ausgedrückt werden (noch nicht einmal in der unendlichen Logik $L_{\omega_1\omega}$, siehe [Kei71]).

Bei der Konstruktion von $\underline{\texttt{solvable}}_P$ werden wir schrittweise vorgehen und zunächst Teilformeln konstruieren, denen wir dann einen symbolischen Namen (wie z. B. $\underline{\texttt{constr}}_P$) zuordnen. Wir verwenden in der allgemeinen Konstruktion stets einzelne Variablen, erlauben aber in Anwendungen, daß statt dessen Tupel von Variablen verwendet werden. Dies wird sich in einigen Beispielen als nützlich erweisen. Da wir die Unentscheidbarkeit eines möglichst kleinen Fragments beweisen wollen, müßten wir eigentlich *solv* in Prenexnormalform konstruieren. Wir werden allerdings, um das Verständnis zu erleichtern, die Formeln in der allgemeinen Syntax der Prädikatenlogik angeben. Unsere Darstellung *repräsentiert* also streng genommen nur eine Prenexnormalform.

Unsere Anwendungen umfassen insbesondere auch *Gleichungsprobleme*. Eine Übersicht über allgemeine Gleichungsprobleme findet man in [BHSS87]. Dort werden zwei Klassen von Problemen herausgestellt:

- Die Theorie der *initialen Algebra* einer (einsortigen) Signatur Σ und einer Menge E von Σ-Gleichungen ist unserem Formalismus

$$Th(\mathcal{T}(\Sigma,\emptyset)/E)$$

- Die Theorie der *freien Algebra* einer (einsortigen) Signatur und einer Menge E von Σ-Gleichungen ist die Theorie des Quotienten der von einer nicht näher bestimmten, unendlichen Menge von Variablen frei erzeugten Termalgebra nach der von E definierten

Kongruenzrelation. Die erzeugenden Variablensymbole sind allerdings nicht in der prädikatenlogischen Sprache enthalten, sie dienen nur dazu, "junk" in die Algebra einzuführen. Formal ist

$$\mathcal{T}(\Sigma, X)/E := (\mathcal{T}(\Sigma \cup X, \emptyset)/E)\,|_{\Sigma}$$

In diesem Zusammenhang wird in [BSS89] das Π_3-Fragment einer Theorie als *spezielles Gleichungsproblem* (special equational problem) und das Σ_2-Fragment als *spezielles Gleichungsproblem ohne unabhängige Parameter* (special equational problem without independent parameters) bezeichnet.

Wir werden in einem Anwendungsbeispiel auch unendliche Bäume, die wir bisher nicht definiert haben, betrachten. Die Eigenschaften unendlicher Bäume wurden in [Cou83] behandelt.

Eines unserer Anwendungsbeispiele wird die Unentscheidbarkeit des Σ_2-Fragments der vollständigen Zahlentheorie sein. Dies ist natürlich kein neues Ergebnis, da die Unentscheidbarkeit von Hilberts Zehntem Problem und damit des Σ_1-Fragments der vollständigen Zahlentheorie in [Mat70] bewiesen wurde. Wir führen diese Anwendung hier nur an, um einen bestimmten Aspekt unserer Methode zu illustrieren.

Eine Übersicht über unsere Unentscheidbarkeitsresultate und einen Vergleich mit aus der Literatur bekannten Sätzen findet man in Abschnitt 1.6.

Die beiden Datentypen des Postschen Korrespondenzproblems induzieren die folgende Vorgehensweise: In Abschnitt 8.1 stellen wir die Simulation des Datentypes *Worte* dar. In den Anwendungsbeispielen wird dieser Teil stets der einfachere sein. Wir geben einige Beispiele für die Simulation von Worten in konkreten Modellen an, die wir dann später durch Angabe der Simulation von Folgen zu kompletten Beispielen ausbauen werden. Abschnitt 8.2 enthält zwei alternative Methoden zur Simulation des Datentyps *Folgen*. Bei der ersten dieser Methoden betrachten wir die Folgen als Mengen. Dies führt zu einfacheren Beweisen, hat aber den Nachteil, daß wir auf diese Art und Weise unter Umständen nicht die Unentscheidbarkeit des kleinst möglichen Fragments nachweisen können. Die zweite Methode hingegen realisiert unmittelbar den Datentyp *Folge*, was zu stärkeren Resultaten führen kann, was aber auch einen größeren Aufwand erfordert.

8.1 Simulation von Worten

Um den Datentyp *Worte* in der Logik simulieren zu können, benötigen wir zunächst eine Kodierung der Trägermenge:

$$\phi\colon \{a, b\}^* \to I$$

Wir werden das Symbol ϕ auch für die auf Paare von Worten erweiterte Kodierung benutzen:

$$\phi\colon \{a, b\}^* \times \{a, b\}^* \to I^2$$

Die Operationen des Datentypes *Worte*, die wir zur Simulation des Postschen Korrespondenzproblemes benötigen, sind der Vergleich eines Wortes mit dem leeren Wort sowie für jedes fest

gewählte Wort eine einstellige Funktion, die das gegebene Wort an ihr Argument anhängt. Genauer benötigen wir

- $\underline{\texttt{is-}\epsilon}(x)$
- $(y)\underline{v}(x)$ für jedes $v \in \{a,b\}^+$

so daß die folgenden Bedingungen erfüllt sind:

[INJ] ϕ ist injektiv

[EPS] Für alle $r \in I$: gilt $I \models \underline{\texttt{is-}\epsilon}[r]$ genau dann, wenn $r = \phi(\epsilon)$

[CON] Für alle $r \in I$, $v \in \{a,b\}^+$ und $w \in \{a,b\}^*$ gilt $I \models [r]\underline{v}[\phi(w)]$ genau dann, wenn $r = \phi(wv)$

In den Anwendungen müssen wir also sowohl ϕ als auch die Formeln $\underline{\texttt{is-}\epsilon}$ und $\underline{v}$ geeignet definieren. Dises Verfahren enthält eine gewisse Redundanz, eine andere Möglichkeit wäre es gewesen, andere Bedingungen an $\underline{\texttt{is-}\epsilon}$ und $\underline{v}$ zu stellen, so daß die Kodierungsfunktion dann aus den Definitionen dieser Formeln in folgendem Stil abgeleitet werden kann:

$$\begin{aligned} \phi(\epsilon) &:= \text{das eindeutige } r \text{ mit } \models \underline{\texttt{is-}\epsilon}[r] \\ \phi(w) &:= \text{das eindeutige } r \text{ mit } \models [r]\underline{w}[\phi(\epsilon)] \quad (w \neq \epsilon) \end{aligned}$$

Ein Vorteil hiervon wäre es, daß die Bedingungen an die Formeln [INJ], [EPS] und [CON] in diesem Fall nur die *Theorie* des Modells betreffen würden. Wir haben uns gegen diesen Weg entschieden, da eine explizite Angabe von Kodierungsfunktion *und* Formeln zu klareren Beweisen führt. Außerdem müssen wir für die nächste Bedingung sowieso Eigenschaften des Modells, die wir nicht auf Eigenschaften der Theorie des Modells reduzieren können, betrachten:

Definition 34 *Die Relation $\sqsubset$ auf I ist definiert durch: $x \sqsubset y$ genau dann, wenn es ein $v \in \{a,b\}^+$ gibt mit $I \models [y]\underline{v}[x]$. Wie üblich bezeichnen wir mit $\sqsubset^*$ den reflexiven transitiven Abschluß von $\sqsubset$. Weiterhin verallgemeinern wir $\sqsubset$ auf Paare durch die Festlegung $(x_1,x_2) \sqsubset (y_1,y_2)$ genau dann, wenn $x_1 \sqsubset y_1$ und $x_2 \sqsubset y_2$.*

Falls $r_1 = \phi(w_1)$ und $r_2 = \phi(w_2)$, dann drückt $r_1 \sqsubset r_2$ aus, daß w_1 ein Präfix von w_2 ist. Die obige Definition ist jedoch nicht auf den Bildbereich der Kodierungsfunktion beschränkt. Wir müssen die Eigenschaften von $\sqsubset$ auch für Elemente des Modells, die nicht Kodierung eines Wortes sind, betrachten. Die folgenden Bedingungen fordern nämlich, daß $\sqsubset$ noethersch ist nicht nur auf dem Bildbereich der Kodierungsfunktion (dies folgt ja bereits aus der noetherschen Eigenschaft der Präfixrelation), sondern auf einem Teilbereich von M, den wir durch eine Formel charakterisieren können und der alle Kodierungen von Worten enthält:

[NOE] Es gibt keine unendliche Folge $(r_i)_{i\geq 0}$ in I mit $r_{i+1} \sqsubset r_i$ für alle i und $I \models \underline{\texttt{finite}}[r_0]$.

[FIN] Für alle $w \in \{a,b\}^+$ gilt $I \models \underline{\texttt{finite}}[\phi(w)]$

Wir illustrieren die Simulation des Datentyps *Worte* nun an einigen Beispielen, die wir im nächsten Abschnitt durch die Simulation von Folgen zu kompletten Anwendungsbeispielen vervollständigen werden.

Beispiel (1): Die Sprache τ enthalte mindestens die Funktionssymbole $\epsilon(0), a(1), b(1)$ und unter den Prädikatsymbolen mindestens das Gleichheitssymbol $=(2)$. I sei eine Quotiententermalgebra $\mathcal{T}(\Sigma,\emptyset)/E$, wobei Σ die in τ enthaltene Signatur ist und E eine Menge von Σ-Gleichungen, die jeweils nicht die Funktionssymbole a oder b und keine Gleichung der Form $x = t$, wobei x eine Variable ist, enthalten.

Wir definieren nun, wobei wir die Zeichen a und b des Alphabetes mit den unären Funktionszeichen a und b identifizieren:

$$\begin{aligned}
\phi(\sigma_0 \cdots \sigma_n) &:= \sigma_n(\cdots(\sigma_0(\epsilon))\cdots) \\
\underline{\texttt{is-}\epsilon}(x) &:= x = \epsilon \\
(y)\underline{\sigma_0 \cdots \sigma_n}(x) &:= y = \sigma_n(\cdots(\sigma_0(x))\cdots) \\
\underline{\texttt{finite}}(x) &:= \texttt{True}
\end{aligned}$$

So ist beispielsweise $\phi(aab) = b(a(a(\epsilon)))$. Man sieht leicht, daß die Bedingungen [INJ], [EPS], [CON], [FIN] und [NOE] erfüllt sind. Die Einschränkungen an E werden dabei im Beweis von [INJ] benötigt. ◇

Beispiel (2): Die Sprache τ enthalte mindestens die Funktionssymbole $\epsilon(0), f(1), +(2)$ und unter den Prädikatsymbolen mindestens das Gleichheitssymbol $=(2)$. I sei eine Quotiententermalgebra $\mathcal{T}(\Sigma,\emptyset)/E$, wobei Σ die in τ enthaltene Signatur ist und E aus den Axiomen für die Assoziativität, bzw. Assoziativität und Kommutativität von $+$ besteht. Mit Hilfe der Abkürzungen

$$\begin{aligned}
\overline{a}(t) &:= +(\epsilon, f(f(t))) \\
\overline{b}(t) &:= +(f(\epsilon), f(f(t)))
\end{aligned}$$

definieren wir

$$\begin{aligned}
\phi(\sigma_0 \cdots \sigma_n) &:= \overline{\sigma_n}(\cdots(\overline{\sigma_0}(\epsilon))\cdots) \\
\underline{\texttt{is-}\epsilon}(x) &:= x = \epsilon \\
(y)\underline{\sigma_0 \cdots \sigma_n}(x) &:= y = \overline{\sigma_n}(\cdots(\overline{\sigma_0}(x))\cdots) \\
\underline{\texttt{finite}}(x) &:= \texttt{True}
\end{aligned}$$

Die Bedingungen an die Simulation des Datentypes *Worte* sind erfüllt. Dies gilt auch, wenn wir für I die freie Algebra $\mathcal{T}(\Sigma, X)/E$ wählen. ◇

In den ersten beiden Beispielen war die Relation $\sqsubset$ sogar jeweils für das gesamte Modell I noethersch, so daß wir einfach für $\underline{\texttt{finite}}(x)$ die Formel `True` wählen konnten. Das nächste Beispiel enthält eine nichttriviale Formel $\underline{\texttt{finite}}(x)$:

Beispiel (3): Die Sprache τ enthalte mindestens die Funktionssymbole $\epsilon(0), a(1), b(1)$ und unter den Prädikatsymbolen mindestens $=(2), \leq(2)$. Es sei nun $I = \mathcal{T}^\infty(F, \emptyset)$ die Algebra der endlichen *und* unendlichen F-Terme, wobei $\leq$ als die Teilbaumrelation interpretiert wird. Man beachte, daß der auf endliche Bäume eingeschränkte Fall bereits von Beispiel (1) abgedeckt wird. Wir wählen ϕ, $\underline{\texttt{is-}\epsilon}$ und $\underline{v}$ wie in Beispiel (1). Die Menge der endlichen Elemente von I besteht nun aus den Bäumen, die nur mit unären Funktionssymbolen aufgebaut sind und die die Konstante ϵ enthalten.

$$\underline{\texttt{finite}}(x) \quad := \quad \epsilon \leq x \wedge \forall x'.x' \leq x \supset \{x' = \epsilon \vee \exists x''.x' = a(x'') \vee x' = b(x'')\}$$

Falls die Menge F' der *nicht*unären Funktioszeichen endlich ist, so können wir eine Quantoralternierung einsparen:

$$\underline{\texttt{finite}}(x) := \epsilon \leq x \wedge \forall x'.x' \leq x \supset \bigwedge_{f \in B'} \forall \vec{z}.x' \neq f(\vec{z}) \qquad \diamond$$

Unser letztes Beispiel benutzt eine noch weiter eingeschränkte Signatur.

Beispiel (4): Die Sprache τ enthalte mindestens die Funktionssymbole $\epsilon(0)$, $f(2)$ und unter den Prädikatsymbolen mindestens das Gleichheitssymbol $=(2)$. I sei die Grundtermalgebra $\mathcal{T}(\Sigma, \emptyset)$, wobei Σ die in τ enthaltene Signatur ist. Mit Hilfe der Abkürzungen

$$\begin{aligned} \overline{a}(t) &:= f(\epsilon, t) \\ \overline{b}(t) &:= f(f(\epsilon, \epsilon), t) \end{aligned}$$

definieren wir

$$\begin{aligned} \phi(\sigma_0 \cdots \sigma_n) &:= \overline{\sigma_n}(\cdots(\overline{\sigma_0}(\epsilon))\cdots) \\ \underline{\texttt{is-}\epsilon}(x) &:= x = \epsilon \\ (y)\underline{\sigma_0 \cdots \sigma_n}(x) &:= y = \overline{\sigma_n}(\cdots(\overline{\sigma_0}(x))\cdots) \\ \underline{\texttt{finite}}(x) &:= \texttt{True} \end{aligned}$$

Auch hier sind die Bedingungen an die Simulation des Datentyps *Worte* erfüllt. $\diamond$

8.2 Simulation von P-Konstruktionen

Nun können wir die Formel $\underline{\texttt{one-step}}_P$ formulieren. Die beabsichtigte Bedeutung der Formel $\underline{\texttt{one-step}}_P(y_1, y_2, y_3, y_4)$ ist dabei:

> Das von (y_1, y_2) kodierte Paar von Worten geht durch Anwendung eines Konstruktionsschrittes von P aus dem von (y_3, y_4) kodierten Paar von Worten hervor.

Dies ist die einzige Teilformel, bei der die benutzte Instanz P des Postschen Korrespondenzproblems direkt eingeht.

$$\underline{\texttt{one-step}}_P(y_1, y_2, y_3, y_4) \quad := \quad \bigvee_{i=0,\ldots,m} ((y_1)\underline{p_i}(y_3) \wedge (y_2)\underline{q_i}(y_4))$$

wobei $P = \{(p_i, q_i) \mid i = 0, \ldots, m\}$. Von nun an beziehen wir uns auf eine fest gewählte Instanz P des Postschen Korrespondenzproblems.

8.2.1 Simulation von Folgen als Mengen

In diesem Abschnitt werden wir eine Formel $\underline{\texttt{constr}}_P(x)$ angeben, die ausdrückt, daß x eine P-Konstruktion repräsentiert. Dabei wollen wir zunächst die Folgen von Paaren von Worten, die ja eine P-Konstruktion ausmachen, als *Mengen* darstellen. Dies ist deshalb möglich, weil die Paare von Worten in einer P-Konstruktion bezüglich der (lexikographischen Erweiterung der) Teilwortrelation echt wachsen. Die zugrundeliegende P-Konstruktion ist daher durch Angabe der aus ihr abgeleiteten Menge eindeutig bestimmt.

Die Definition von $\underline{\texttt{constr}}_P$ benutzt eine Teilformel $(y_1, y_2)\underline{\texttt{in}}(x)$, die die Elementrelation ausdrücken soll. Während wir den "Rahmen" für die Formel $\underline{\texttt{constr}}_P$ hier allgemein festlegen können, muß die Teilformel $(y_1, y_2)\underline{\texttt{in}}(x)$ in Hinblick auf das betrachtete Modell definiert werden.

$$\begin{aligned}\underline{\texttt{constr}}_P(x) := \forall y_1, y_2.(y_1, y_2)\underline{\texttt{in}}(x) \supset & \\ \{\underline{\texttt{is-}\epsilon}(y_1) \wedge \underline{\texttt{is-}\epsilon}(y_2)\} \vee & \quad (42)\\ \exists y_3, y_4.(y_3, y_4)\underline{\texttt{in}}(x) \wedge \underline{\texttt{one-step}}_P(y_1, y_2, y_3, y_4) & \quad (43)\end{aligned}$$

Obwohl wir die Formel $\underline{\texttt{in}}$ noch gar nicht festgelegt haben, können wir bereits jetzt zeigen:

Lemma 45 *Für alle $r_1, r_2, u, s \in I$ mit $(r_1, r_2) \sqsubset^* (u, u)$ und*

$$\begin{aligned} I &\models \underline{\texttt{finite}}[u] \\ I &\models \underline{\texttt{constr}}_P[s] \\ I &\models [r_1, r_2]\underline{\texttt{in}}[s] \end{aligned}$$

gilt: Falls [INJ], [EPS], [CON] und [NOE] erfüllt sind, dann ist $(r_1, r_2) \in \phi(\Sigma^*) \times \phi(\Sigma^*)$, und das davon kodierte Paar von Worten $\phi^{-1}(r_1, r_2)$ ist P-konstruierbar.

Beweis: Es seien u und s wie in dem Lemma gegeben. Wegen [NOE] gibt es keine unendliche bzgl. $\sqsubset$ absteigende Kette von Paaren (r_1, r_2) mit $(r_1, r_2) \sqsubset^* (u, u)$. Aus diesem Grunde können wir eine noethersche Induktion über (r_1, r_2) durchführen.

Falls $I \models \underline{\texttt{is-}\epsilon}[r_1] \wedge \underline{\texttt{is-}\epsilon}[r_2]$ gilt, dann folgt $(r_1, r_2) = \phi(\epsilon, \epsilon)$ aus [EPS], und wir sind fertig.

Anderenfalls trifft Fall (43) der Definition von $\underline{\texttt{constr}}_P$ ein, also gibt es r_3, r_4 mit $I \models [r_3, r_4]\underline{\texttt{in}}[s]$ und $I \models \underline{\texttt{one-step}}_P[r_1, r_2, r_3, r_4]$. Aus der Definition von $\underline{\texttt{one-step}}_P$ folgt, daß

$$(r_3, r_4) \sqsubset (r_1, r_2) \sqsubset^* (u, u)$$

Dann erhalten wir aus der Induktionsvoraussetzung, daß $(r_3, r_4) \in \phi(\Sigma^*) \times \phi(\Sigma^*)$ und $\phi^{-1}(r_3, r_4)$ ist P-konstruierbar. Wegen [CON] und der Definition von $\underline{\texttt{one-step}}_P$ gilt dies dann auch für (r_1, r_2). □

Wir können nun endlich den Satz $\underline{\texttt{solvable}}_P$ angeben.

$$\underline{\texttt{solvable}}_P \quad := \quad \exists x, y.\underline{\texttt{constr}}_P(x) \wedge \underline{\texttt{finite}}(y) \wedge (y, y)\underline{\texttt{in}}(x) \wedge \neg\underline{\texttt{is-}\epsilon}(y)$$

Aus dem obigen Lemma folgt dann sofort

Korollar 9 *Falls [INJ], [EPS], [CON] und [NOE] erfüllt sind, dann gilt*

$$I \models \underline{\texttt{solvable}}_P \quad \Rightarrow \quad P \text{ ist lösbar}$$

Wir weisen nochmals darauf hin, daß an dieser Stelle noch keinerlei Aussage über den Aufbau von $\underline{\texttt{in}}$ getroffen wurde. Die speziellen Eigenschaften des Modells wurden bisher nur zum Beweis der Bedingungen in Abschnitt 8.1 benutzt. Sobald eine korrekte Simulation des Datentyps *Worte* gefunden ist, erhalten wir Korollar 9 "geschenkt", d. h. ohne daß wir eine Repräsentation des Datentyps *Folge* angeben müßten.

Dies ist dann aber natürlich notwendig, um die Gegenrichtung von Korollar 9 zu beweisen. M bezeichne den Definitionsbereich der zu konstruierenden Kodierungsfunktion ψ:

$$M := \{(u_i, v_i)_{i=1\ldots n} \mid u_i, v_i \in \{a, b\}^*, n \geq 2, u_i \lhd u_{i+1}, v_i \lhd v_{i+1}, (u_1, v_1) = (\epsilon, \epsilon)\}$$

Wir müssen nun eine Kodierungsfunktion $\psi\colon M \to I$ sowie eine Formel $(y_1, y_2)\underline{\texttt{in}}(x)$ finden, so daß gilt:

[IN] Für alle $s \in M$: $I \models [r_1, r_2]\underline{\texttt{in}}[\psi(s)]$ genau dann, wenn es ein $j \in \{1, \ldots, lth(s)\}$ gibt mit $(r_1, r_2) = \phi(s(j))$

Lemma 46 *Wenn [EPS], [CON], [FIN] und [IN] erfüllt sind, dann gilt*

$$P \text{ ist lösbar} \quad \Rightarrow \quad I \models \underline{\texttt{solvable}}_P$$

Der folgende Satz faßt die bisher formulierte Methode zusammen:

Satz 8 *Es sei τ eine prädikatenlogische Sprache und I ein τ-Modell. Falls es Kodierungsfunktionen ϕ, ψ und Formeln $\underline{\texttt{is-}\epsilon}, \underline{v}, \underline{\texttt{finite}}, \underline{\texttt{in}}$ mit [INJ], [EPS], [CON], [FIN], [NOE] und [IN] gibt, dann ist $Th(I)$ unentscheidbar.*

Wir können nun einige der in Abschnitt 8.1 begonnen Beispiele vervollständigen. Zunächst zeigen wir, daß das Σ_3 Fragment der Theorie einer Grundtermalgebra $\mathcal{T}(\Sigma, \emptyset)/AC$, bzw. freien Termalgebra $\mathcal{T}(\Sigma, X)/AC$ modulo Assoziativität und Kommutativität im allgemeinen unentscheidbar ist.

Satz 9 *Die Signatur Σ enthalte mindestens eine Konstante, ein mindestens unäres Funktionszeichen und ein binäres Funktionszeichen $+$. $AC(+)$ bezeichne die Axiome der Assoziativität und Kommutativität von $+$:*

$$\begin{aligned} x + y &= y + x \\ (x + y) + z &= x + (y + z) \end{aligned}$$

Dann ist das Σ_3 Fragment der Theorie $Th(\mathcal{T}(\Sigma,\emptyset)/AC(+))$ sowie das Σ_3 Fragment der Theorie $Th(\mathcal{T}(\Sigma,X)/AC(+))$ unentscheidbar.

Beweis: Wir betrachten zunächst den etwas einfacheren Fall der Signatur

$$\langle \epsilon(0), a(1), b(1), f(2), +(2) \rangle$$

wobei + assoziativ und kommutativ ist. Die Simulation des Datentyps *Worte* nehmen wir aus Beispiel (1). Wie man leicht sieht, erfüllt die folgende Definition die Bedingung [IN] sowohl im Fall der initialen Algebra als auch der freien Algebra:

$$\begin{aligned} \psi((u_i, v_i)_{i=1,\ldots,n}) &:= f(\phi(u_1), \phi(v_1)) + \cdots + f(\phi(u_n), \phi(v_n)) \\ (y_1, y_2)\underline{\mathtt{in}}(x) &:= \exists x'.x = f(y_1, y_2) + x' \end{aligned}$$

Im Falle der Signatur $\langle \epsilon(0), f(1), +(2) \rangle$ nehmen wir nun die Simulation der *Worte* aus Beispiel (2) und

$$\begin{aligned} \psi((u_i, v_i)_{i=1,\ldots,n}) &:= f(f(\phi(u_1)) + f(f(\phi(v_1)))) + \cdots + f(f(\phi(u_n)) + f(f(\phi(v_n)))) \\ (y_1, y_2)\underline{\mathtt{in}}(x) &:= \exists x'.x = f(f(y_1) + f(f(y_2))) + x' \end{aligned}$$

Auch hier zeigt man leicht [IN] für die initiale und für die freie Algebra. In dieser Konstruktion benutzen wir das AC Funktionssymbol + als Konstruktor von Worten *und* als Konstruktor von Folgen. Dies wird durch die Einfügung des unären Funktionszeichens f zwischen die verschiedenen Vorkommen von + in der Kodierung der Worte ermöglicht, diese dienen als "Sperre" gegen unerwünschte Anwendungen der AC Axiome. □

Mit der gleichen Konstruktion können wir zeigen, daß die Theorie einer Grundtermalgebra modulo Assoziativität, Kommutativität und Idempotenz unentscheidbar ist:

Satz 10 *Die Signatur Σ enthalte mindestens eine Konstante, ein mindestens unäres Funktionszeichen und ein binäres Funktionszeichen $+$. $ACI(+)$ bezeichne die Axiome der Assoziativität, Kommutativität und Idempotenz von $+$:*

$$\begin{aligned} x + y &= y + x \\ (x + y) + z &= x + (y + z) \\ x + x &= x \end{aligned}$$

Dann ist das Σ_3 Fragment der Theorie $Th(\mathcal{T}(\Sigma,\emptyset)/ACI(+))$ sowie das Σ_3 Fragment der Theorie $Th(\mathcal{T}(\Sigma,X)/ACI(+))$ unentscheidbar.

Wir zeigen nun, daß das Σ_4 Fragment der Theorie einer partiellen lexikographischen Pfadordnung im allgemeinen unentscheidbar ist.

Satz 11 *Es gibt eine Signatur Σ und eine partielle Ordnung $\preceq$ auf den Funktionszeichen von Σ, so daß das Σ_4 Fragment der Theorie der Grundtermalgebra $\mathcal{T}(\Sigma,\emptyset)/\emptyset$ mit der lexikographischen Pfadordnung $\preceq_{lpo}$ unentscheidbar ist.*

Beweis: Wir betrachten die folgende prädikatenlogische Sprache:

$$\begin{aligned} F &:= \langle \epsilon(0), a(1), b(1), e(1), l(1), h(3) \rangle \\ P &:= \langle =(2), \leq(2) \rangle. \end{aligned}$$

I sei die Grundtermalgebra $\mathcal{T}(\Sigma, \emptyset)/\emptyset$ wobei $\leq$ als die von der folgenden partiellen Ordnung $\prec$ erzeugte lexikographische Pfadordnung $\preceq_{lpo}$ interpretiert wird:

$$\epsilon \prec a \prec b \prec h \prec \begin{cases} l \\ e \end{cases}$$

e und l sind also unvergleichbar bezüglich $\prec$.

ϕ, $\underline{\texttt{is-}\epsilon}$, $\underline{\texttt{finite}}$ und $\underline{v}$ übernehmen wir aus Beispiel (1). Eine P Konstruktion repräsentieren wir nun durch zwei Listen von markierten Worten, jeweils eine für die ersten und die zweiten Komponenten der Konstruktion. Die Markierungen stellen die Verbindung der zugehörigen Komponenten her, sind aber auch für die Realisierung der Elementrelation von Bedeutung.

Wir ordnen zunächst jeder nichtleeren Folge $s = (u_i)_{i=1,\ldots,n}$ den Term $\delta(s)$ zu, wobei δ wie folgt definiert ist (siehe auch Abbildung 31):

$$\delta((u_i)_{i=k\ldots n}) = \begin{cases} \epsilon & \text{falls } k > n \\ h(l(a^{k-1}(\epsilon)), e(\phi(u_n)), \delta((u_{i-1})_{i=k+1\ldots n})) & \text{anderenfalls} \end{cases}$$

und mit Hilfe von δ:

$$\psi((u_i, v_i)_{i=1,\ldots,n}) := (\delta((u_i)_{i=1,\ldots,n}), \delta((v_i)_{i=1,\ldots,n}))$$

Wir benutzen die folgende temporäre Definition:

$$\begin{aligned} (y)\underline{\texttt{in}}(x)\underline{\texttt{at}}(z) &:= h(l(z), e(y), \epsilon) \leq x \wedge & (44) \\ &\quad \forall y'.h(l(z), e(y'), \epsilon) \leq x \supset y' \leq y & (45) \end{aligned}$$

Damit können wir nun $\underline{\texttt{in}}$ definieren:

$$(y_1, y_2)\underline{\texttt{in}}(x_1, x_2) := \exists z.(y_1)\underline{\texttt{in}}(x_1)\underline{\texttt{at}}(z) \wedge (y_2)\underline{\texttt{in}}(x_2)\underline{\texttt{at}}(z)$$

Der Kern des Beweises der Bedingung [IN] ist das folgende Lemma:

Lemma 47 *Es sei $s = (u_i)_{i=1\ldots n}$ eine nichtleere, bzgl. $\triangleleft$ wachsende Folge über $\{a, b\}^*$. Dann sind für alle $t, t_0 \in T(\{\epsilon, a, b\})$ die beiden folgenden Aussagen äquivalent:*

1. $I \models [t]\underline{\texttt{in}}[\delta(s)]\underline{\texttt{at}}[t_0]$
2. *es gibt ein $j \in \{1, \ldots, n\}$ mit $t_0 = a^{n-j}(\epsilon)$ und $t = u_j$*

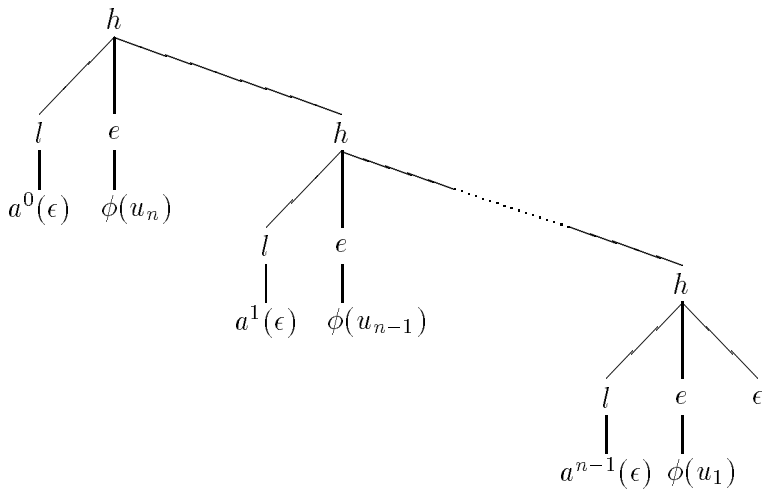


Abbildung 31: Der Term $\delta((u_i)_{i=1\ldots n})$ kodiert die Folge $(u_i)_{i=1\ldots n}$

Beweis: [von Lemma 47] Zunächst benötigen wir eine einfache Eigenschaft der lexikographischen Pfadordnung:

(*) Falls $t_1 \preceq_{\text{lpo}} t_2$, dann gibt es für jedes in t_1 vorkommende Funktionszeichen f ein Funktionszeichen g mit Vorkommen in t_2 und $f \preceq g$.

Es sei nun $h(l(t_0), e(t), \epsilon) \preceq_{\text{lpo}} \delta((u_i)_{i=k\ldots n})$. Gemäß der Definition der lexikographische Pfadornung gibt es vier Möglichkeiten:

1. $h(l(t_0), e(t), \epsilon) \preceq_{\text{lpo}} l(a^{k-1}(\epsilon))$ oder $h(l(t_0), e(t), \epsilon) \preceq_{\text{lpo}} e(u_n)$ oder
2. $h(l(t_0), e(t), \epsilon) \preceq_{\text{lpo}} \delta((u_{i-1})_{i=k+1\ldots n})$ oder
3. $l(t_0) = l(a^{k-1}(\epsilon))$ und $e(t) \preceq_{\text{lpo}} e(u_n)$ oder
4. $l(t_0) \prec_{\text{lpo}} l(a^{k-1}(\epsilon))$ und $e(t) \prec_{\text{lpo}} \delta((u_i)_{i=k\ldots n})$

Die erste Möglichkeit scheidet auf Grund von (*) aus. Aus dem gleichen Grunde ist (4) nur möglich, falls $e(t) \preceq_{\text{lpo}} e(u_j)$ für ein $j \in \{k, \ldots, n\}$. Mit Hilfe von Induktion in Fall (2) und durch Anwendung von (*) erhalten wir speziell für $k = 1$:

(**) Falls $h(l(t_0), e(t), \epsilon) \preceq_{\text{lpo}} \delta((u_i)_{i=1\ldots n})$, dann gibt es i, i' mit $1 \leq i' \leq i \leq n$, so daß $t_0 \preceq_{\text{lpo}} a^{n-i}(\epsilon)$ und $t \preceq_{\text{lpo}} u_{i'}$.

(1) $\Rightarrow$ (2): Es sei $I \models (t)\underline{\texttt{in}}(\delta(s))\underline{\texttt{at}}(t_0)$. Wegen (**) gibt es i, i' mit

$$t_0 \quad \preceq_{\text{lpo}} \quad a^{n-i}(\epsilon)$$

$$t \quad \preceq_{\mathrm{lpo}} \quad u_{i'}$$

und $1 \leq i' \leq i \leq n$. Da es kein nicht-konstantes Funktionszeichen kleiner als a gibt, muß t_0 von der Form $a^{n-j}(\epsilon)$ mit $i \leq j \leq n$ sein. Es ist also

$$t \preceq_{\mathrm{lpo}} u_{i'} \preceq_{\mathrm{lpo}} u_i \preceq_{\mathrm{lpo}} u_j$$

Andererseits folgt aus der Konstruktion von $\delta(s)$, daß

$$h(l(a^{n-j}(\epsilon)), e(u_j), \epsilon) \preceq_{\mathrm{lpo}} \delta(s)$$

und mit (45) folgt $u_j \preceq_{\mathrm{lpo}} t$. Aus der Antisymmetrie von $\preceq_{\mathrm{lpo}}$ erhalten wir $t = u_j$.

(2) $\Rightarrow$ (1): (44) folgt direkt aus der Definition von $\delta(s)$. es sei also

$$h(l(a^{m-j}(\epsilon)), e(t'), \epsilon) \preceq_{\mathrm{lpo}} \delta(s)$$

Wegen (**) gibt es i, i' mit $1 \leq i' \leq i \leq n$ und

$$\begin{array}{rcl} a^{m-j}(\epsilon) & \preceq_{\mathrm{lpo}} & a^{m-i}(\epsilon) \\ t' & \preceq_{\mathrm{lpo}} & u_{i'} \end{array}$$

Dies ist nur möglich, wenn $j \geq i$, und wir erhalten

$$t' \preceq_{\mathrm{lpo}} u_{i'} \preceq_{\mathrm{lpo}} u_i \preceq_{\mathrm{lpo}} u_j = t$$

□

Die Bedingung [IN] folgt sofort aus Lemma 47 und der Definition von `in`. □

Die Aufteilung der P-Konstruktion in zwei Listen war, genau genommen, nicht wesentlich für den Beweis. Wir könnten den Unentscheidbarkeitsbeweis auch mit einer Simulation der P Konstruktion als *einer* Liste von Paaren von Worten führen. Dazu müßten wir aber ein weiteres bzgl. $\prec$ maximales Funktionszeichen einführen, dies würde also zu einer "weniger" totalen Ordnung und somit zu einem schwächeren Resultat führen. Aber auch bei diesem Ansatz wären die Markierungen $l(a^i(\epsilon))$ nicht überflüssig, da wir sie in der Definition von `in` zur Formulierung der Maximalitätsbedingung benötigen.

Es sei hier angemerkt, daß wir mit einem vollkommen analogen Beweis auch die Unentscheidbarkeit des Σ_4 Fragments einer partiellen Multimengen Pfadordnung zeigen können.

Wenn wir die Quantoralternierungen in dem Satz $\texttt{solvable}_P$ betrachten, stellen wir fest, daß $\texttt{solvable}_P$ auf jeden Fall zumindest in dem Σ_3-Fragment enthalten sein muß. Dies ist ein grundsätzliches Problem der hier vorgestellten Methode, da wir $\texttt{solvable}_P$ nach dem Muster

$$\exists s \cdots \forall (s_1, s_2) \in s \cdots \exists (s_3, s_4) \in s \cdots$$

konstruiert haben. Im allgemeinen ist die Formel `in`, gemessen an der Anzahl der Quantoralternierungen, die "teuerste" Formel. Nur falls es uns gelingt, eine passende Formel `in` in Σ_1 zu finden, und falls uns die Formeln des Datentyp *Worte* keinen Strich durch die Rechnung machen, erhalten wir wirklich die Unentscheidbarkeit für das Σ_3 Fragment der Theorie.

8.2.2 Direkte Simulation von Folgen

In einigen Fällen ist es möglich, diese Beschränkung durch eine direkte Simulation der Folgen zu überwinden. Wir müssen dazu aber jetzt drei Operationen des Datentyps *Folgen* in der Logik durch Formeln repräsentieren. Da wir nun mehrere Operationen betrachten, müssen wir nun, anders als im vorhergehenden Abschnitt, weitere Bedingungen an die die Operationen repräsentierenden Formeln stellen.

Die Formeln, die wir für jeden Anwendungsfall finden müssen, sind die folgenden:

- $\underline{\texttt{nonempty}}(x)$
- $(y_1, y_2, x')\underline{\texttt{sub-of}}(x)$
- $(y_1, y_2)\underline{\texttt{head-of}}(x)$

Die beabsichtige Bedeutung der Formel $\underline{\texttt{nonempty}}$ sollte aus dem Namen ersichtlich sein, die Formel $(y_1, y_2, x')\underline{\texttt{sub-of}}(x)$ soll ausdrücken, daß die Folge mit erstem Element (y_1, y_2) und Rest x' ein Suffix von x ist, und $(y_1, y_2)\underline{\texttt{head-of}}(x)$ soll bedeuten, daß (y_1, y_2) das erste Element der Folge x ist.

Die zum vorigen Abschnitt analoge Definition von $\underline{\texttt{constr}}_P$ ist nun:

$$\begin{aligned}\underline{\texttt{constr}}_P(x) := \forall y_1, y_2, x'.(y_1, y_2, x')\underline{\texttt{sub-of}}(x) \supset \\ \{\underline{\texttt{is-}\epsilon}(y_1) \wedge \underline{\texttt{is-}\epsilon}(y_2)\} \vee \\ \{\underline{\texttt{nonempty}}(x') \wedge \forall y_3, y_4.(y_3, y_4)\underline{\texttt{head-of}}(x') \supset \underline{\texttt{one-step}}_P(y_1, y_2, y_3, y_4)\}\end{aligned}$$

und der Satz $\underline{\texttt{solvable}}_P$ ist definiert als:

$$\underline{\texttt{solvable}}_P := \exists x, y.\underline{\texttt{constr}}_P(x) \wedge (y, y)\underline{\texttt{head-of}}(x) \wedge \underline{\texttt{finite}}(y) \wedge \neg\underline{\texttt{is-}\epsilon}(y)$$

Im Gegensatz zu Abschitt 8.2.1, wo wir eine Richtung unseres angestrebten Satzes, nämlich die von Korollar 9, bereits aus den Eigenschaften der Worte bekamen, müssen wir nun mit einigen Bedingungen die Beziehung der verschiedenen Formeln untereinander regeln:

[NH] $I \models \forall x.\underline{\texttt{nonempty}}(x) \supset \exists y_1, y_2.(y_1, y_2)\underline{\texttt{head-of}}(x)$

[HS] $I \models \forall x, y_1, y_2.(y_1, y_2)\underline{\texttt{head-of}}(x) \supset \exists x'.(y_1, y_2, x')\underline{\texttt{sub-of}}(x)$

[HSH] $I \models \forall x, x', y_1, y_2, y_3, y_4.(y_1, y_2, x')\underline{\texttt{sub-of}}(x) \wedge (y_3, y_4)\underline{\texttt{head-of}}(x') \supset$
$\exists x''.(y_3, y_4, x'')\underline{\texttt{sub-of}}(x)$

An diese Stelle könnte man anmerken, daß wir [NH] auch als Definition von $\underline{\texttt{nonempty}}$ hätten nehmen können, indem wir das Implikationzeichen durch eine Äquivalenz ersetzen. In diesem Fall blieben nur noch die Bedingungen [HS] und [HSH] zu beweisen. Es ist natürlich auch in dem oben gewählten Ansatz möglich, $\underline{\texttt{nonempty}}$ einfach durch $\underline{\texttt{head-of}}$ zu definieren. Darüber hinaus erlauben wir aber auch eine separate Definition von $\underline{\texttt{nonempty}}$, diese ermöglicht unter Umständen eine Reduzierung der Quantoralternierungen, wie wir an Satz 16 sehen werden.

Mit Hilfe dieser Eigenschaften können wir nun ein zu Lemma 45 analoges Lemma zeigen:

Lemma 48 *Für alle $r_1, r_2, u, s, s' \in I$ mit $(r_1, r_2) \sqsubset^* (u, u)$ und*

$$I \models \underline{\texttt{finite}}[u]$$
$$I \models \underline{\texttt{constr}}_P[s]$$
$$I \models [r_1, r_2, s']\underline{\texttt{sub-of}}[s]$$

gilt: Falls [INJ], [EPS], [CON], [NOE], [NH] und [HSH] erfüllt sind, dann ist $(r_1, r_2) \in \phi(\Sigma^*) \times \phi(\Sigma^*)$ und $\phi^{-1}(r_1, r_2)$ ist P-konstruierbar.

Beweis: Wir verwenden, mit der gleichen Begründung wie in Lemma 45, noethersche Induktion über (r_1, r_2).

Falls $I \models \underline{\texttt{is-}\epsilon}[r_1] \wedge \underline{\texttt{is-}\epsilon}[r_2]$, dann folgt aus [EPS] daß $(r_1, r_2) = \phi(\epsilon, \epsilon)$.

Anderenfalls gilt $I \models \underline{\texttt{nonempty}}[s]$, also gibt es wegen [NH] $r_3, r_4 \in I$ mit $I \models [r_3, r_4]\underline{\texttt{head-of}}[s]$. Der zweite Fall der Definition von $\underline{\texttt{constr}}_P$ trifft hier zu, also gilt $I \models \underline{\texttt{one-step}}_P[r_1, r_2, r_3, r_4]$, und damit

$$(r_3, r_4) \sqsubset (r_1, r_2) \sqsubset^* (u, u)$$

Wegen [HSH] gibt es ein $s'' \in I$ mit $I \models [r_3, r_4, s'']\underline{\texttt{sub-of}}[s]$, also können wir die Induktionsannahme auf (r_3, r_4) anwenden. Die Behauptung folgt dann aus [CON] und der Definition von $\underline{\texttt{one-step}}_P$. □

Korollar 10 *Wenn [INJ], [EPS], [CON], [NOE], [NH], [HSH] und [HS] erfüllt sind, dann gilt*

$$I \models \underline{\texttt{solvable}}_P \quad \Rightarrow \quad P \text{ ist lösbar}$$

Um die Gegenrichtung von Korollar 10 zu zeigen, könnten wir zunächst wie in Abschnitt 8.2.1 eine Kodierungsfunktion, die M in I abbildet, verlangen. Dies würde auch in den meisten Fällen ausreichen, wir können die Anforderungen an die Kodierung aber noch weiter abschwächen. Es genügt nämlich, eine Funktion zu finden, die jede Folge s auf eine "private" Kodierung der Menge der Teilfolgen von s abbildet:

$$\psi \in \prod_{s \in M} (\{0, \ldots, lth(s)\} \to I)$$

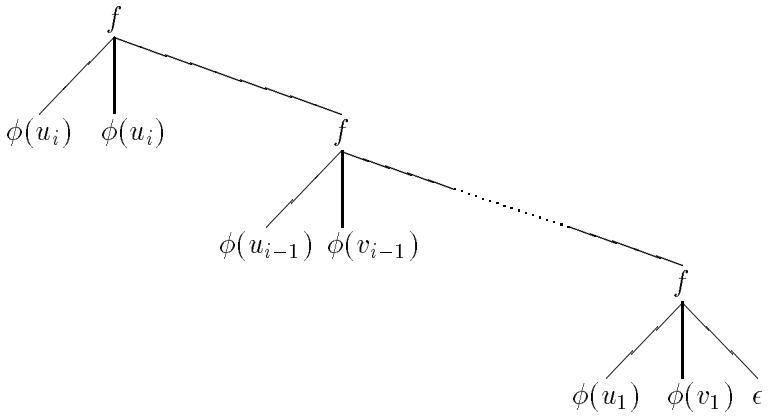
Wir müssen nun noch die Bedingungen angeben, die die Formeln $\underline{\texttt{nonempty}}$, $\underline{\texttt{sub-of}}$ und $\underline{\texttt{head-of}}$ mit der Kodierung in Beziehung setzen:

Für alle $s \in M, n \leq lth(s)$:

[NIL] $I \models \underline{\texttt{nonempty}}[\psi(s)(n)]$ genau dann, wenn $n \neq 0$

[HEA] $I \models [r_1, r_2]\underline{\texttt{head-of}}[\psi(s)(n)]$ genau dann, wenn $n \geq 1$ und $(r_1, r_2) = \phi(s(n))$

[SUB] $I \models [r_1, r_2, t]\underline{\texttt{sub-of}}[\psi(s)(lth(s))]$ genau dann, wenn es ein $i \in \{1, \ldots, lth(s)\}$ gibt mit $(r_1, r_2) = \phi(s(i))$ und $t = \psi(s)(i \perp 1)$



Der Term $\psi((u_j, v_j)_{j=1...n})(i)$ kodiert die Teilfolge $(u_j, v_j)_{j=1...i}$ der Folge $(u_j, v_j)_{j=1...n}$ für $1 \leq i \leq n$

Abbildung 32: Kodierung von Folgen für Satz 13

Lemma 49 *Wenn [EPS], [CON], [FIN], [NIL], [HEA] und [SUB] erfüllt sind, dann gilt*

$$P \text{ ist lösbar} \quad \Rightarrow \quad I \models \underline{\texttt{solvable}}_P$$

Satz 12 faßt die in diesem Abschnitt entwickelte Methode zusammen:

Satz 12 *Es sei τ eine prädikatenlogische Sprache und I ein τ-Modell. Falls es Kodierungsfunktionen ϕ, ψ und Formeln* $\underline{\texttt{is-}\epsilon}$, $\underline{v}$, $\underline{\texttt{nonempty}}$, $\underline{\texttt{finite}}$, $\underline{\texttt{head-of}}$ *und* $\underline{\texttt{sub-of}}$ *gibt, so daß die Bedingungen [INJ], [EPS], [CON], [NOE], [NH], [HS], [HSH], [FIN], [NIL], [HEA] und [SUB] erfüllt sind, dann ist $\mathrm{Th}(I)$ unentscheidbar.*

Zunächst wollen wir demonstrieren, wie das in [Ven87] gegebene Unentscheidbarkeitsresultat in den von uns vorgestellten Rahmen paßt:

Satz 13 ([Ven87]) *Das Σ_2 Fragment der Theorie einer Grundtermalgebra mit dem Teiltermprädikat ist unentscheidbar, falls die Signatur mindestens eine Konstante, zwei unäre und ein ternäres Funktionszeichen enthält.*

Beweis: Wir verwenden die Darstellung der Terme wie in Beispiel (1). Die Kodierungsfunktion $\psi(s)(i)$ wählen wir wie folgt (siehe auch Abbildung 32):

$$\psi(s)(i) := \begin{cases} \epsilon & \text{falls } i = 0 \\ f(\phi(u_i), \phi(v_i), \psi(s)(i \perp 1)) & \text{anderenfalls} \end{cases}$$

und die Formeln für die Operationen auf Folgen:

$$\begin{aligned}(y_1, y_2)\underline{\texttt{head-of}}(x) &:= \exists x'.x = f(y_1, y_2, x')\\ (y_1, y_2, x')\underline{\texttt{sub-of}}(x) &:= f(y_1, y_2, x') \leq x\\ \underline{\texttt{nonempty}}(x) &:= \exists y_1, y_2, x'.x = f(y_1, y_2, x')\end{aligned}$$

Wir können nun eine Quantoralternierung des Satzes $\underline{\texttt{solvable}}_P$ einsparen, indem wir eine zu $\underline{\texttt{nonempty}}$ äquivalente Π_1-Formel angeben:

$$\underline{\texttt{nonempty}}(x) := x \neq \epsilon \wedge \forall x'.x \neq a(x') \wedge x \neq b(x')$$

□

Dieses Ergebnis läßt sich leicht auf die Algebra der endlichen und unendlichen Terme erweitern. Dazu benutzen wir den gleichen Beweis wir für Satz 13, aber mit der Simulation der Worte wie in Beispiel (3):

Satz 14 *Das Σ_2 Fragment der Theorie von $\mathcal{T}^\infty(\Sigma, \emptyset)$ mit dem Teiltermprädikat ist unentscheidbar, falls die Signatur Σ mindestens eine Konstante, zwei unäre und ein ternäres Funktionszeichen enthält.*

Wie bereits angekündigt, können wir das Resultat von Satz 13 verschärfen:

Satz 15 *Das Σ_2 Fragment der Theorie von $\mathcal{T}(\Sigma, \emptyset)$ mit dem Teiltermprädikat ist unentscheidbar, falls die Signatur Σ mindestens eine Konstante und ein mindestens binäres Funktionszeichen enthält.*

Beweis: Die Darstellung der Worte entnehmen wir Beispiel (4). Die Kodierungfunktion für Folgen definieren wir nun als (siehe auch Abbildung 33):

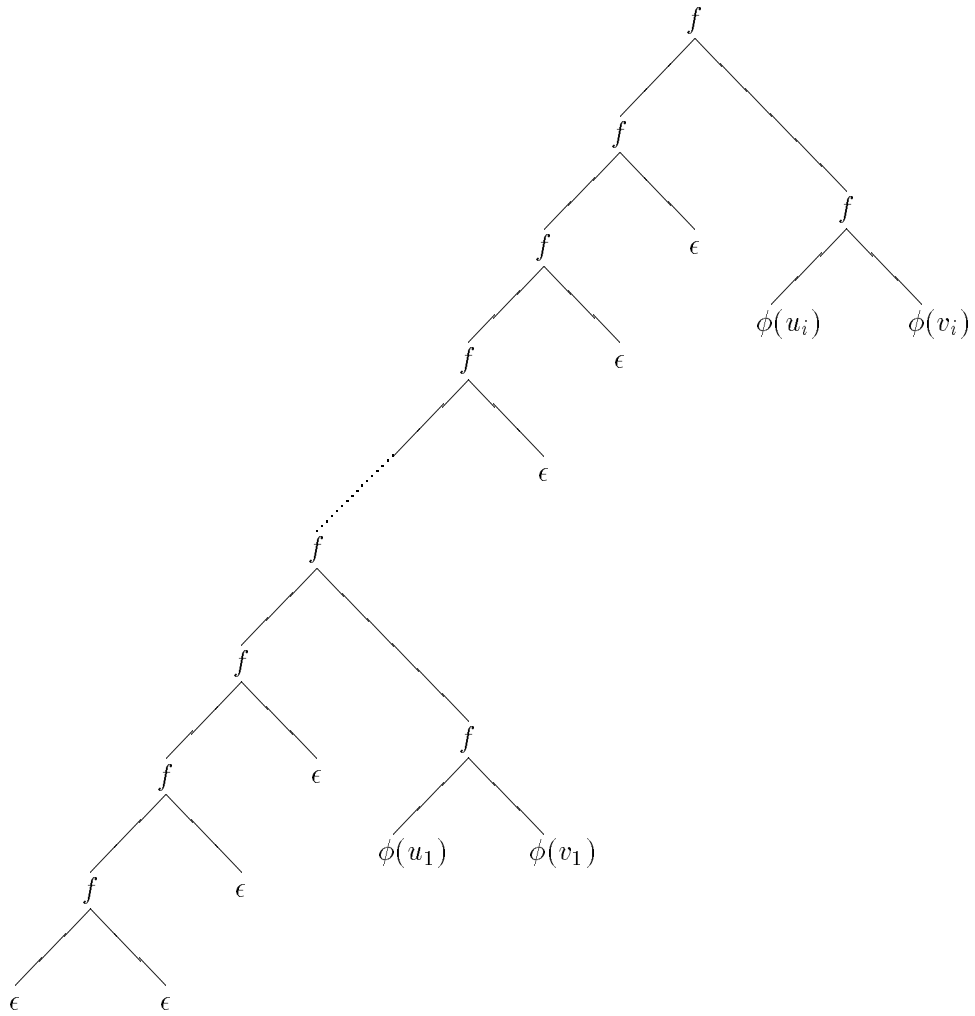
$$\psi(s)(i) := \begin{cases} \epsilon & \text{falls } i = 0\\ f(f(f(f(\psi(s)(i \perp 1), \epsilon), \epsilon), \epsilon), f(\phi(u_i), \phi(v_i))) & \text{anderenfalls}\end{cases}$$

Für die Operationen auf Folgen benutzen wir nun

$$\begin{aligned}(y_1, y_2)\underline{\texttt{head-of}}(x) &:= \exists x'.x = f(f(f(f(x', \epsilon), \epsilon), \epsilon), f(y_1, y_2))\\ (y_1, y_2, x')\underline{\texttt{sub-of}}(x) &:= f(f(f(f(x', \epsilon), \epsilon), \epsilon), f(y_1, y_2)) \leq x\\ \underline{\texttt{nonempty}}(x) &:= \exists y_1, y_2.(y_1, y_2)\underline{\texttt{head-of}}(x)\end{aligned}$$

Auch hier können wir wieder eine Quantoralternierung in $\underline{\texttt{solvable}}_P$ einsparen, indem wir $\underline{\texttt{nonempty}}$ in eine äquivalente Π_1-Formel umwandeln. Eine solche Formel kann beispielsweise mit dem in [CL88] angegebenen Verfahren konstruiert werden. Diese äquivalente Formel ist in unserem Fall aber recht groß und unübersichtlich, wir verzichten deshalb darauf, sie hier explizit anzugeben. □

Wir zeigen nun, daß das Σ_2 Fragment der Theorie einer Grundtermalgebra $\mathcal{T}(\Sigma, \emptyset)/A$ modulo Assoziativität im allgemeinen unentscheidbar ist.



Der Term $\psi((u_j, v_j)_{j=1\ldots n})(i)$ kodiert die Teilfolge $(u_j, v_j)_{j=1\ldots i}$ der Folge $(u_j, v_j)_{j=1\ldots n}$ für $1 \leq i \leq n$

Abbildung 33: Kodierung von Folgen für Satz 15

Satz 16 *Die Signatur Σ enthalte mindestens eine Konstante, ein mindestens unäres Funktionszeichen und ein binäres Funktionszeichen $+$. $A(+)$ bezeichne das Axiom der Assoziativität von $+$:*

$$(x + y) + z \quad = \quad x + (y + z)$$

Dann ist das Σ_3 Fragment der Theorie $Th(\mathcal{T}(\Sigma, \emptyset)/A(+))$ unentscheidbar.

Beweis: Wir betrachten die Signatur $\langle \epsilon(0), f(1), +(2)\rangle$ wobei + assoziativ ist. Die Simulation des Datentyps *Worte* nehmen wir aus Beispiel (2) und ψ ähnlich zu Satz 9:

$$\begin{aligned}
\psi((u_i,v_i)_{i=1,\ldots,m})(j) &:= f(f(\phi(u_j))+f(f(\phi(v_j))))+\cdots \\
&\qquad \cdots+f(f(\phi(u_1))+f(f(\phi(v_1))))+\epsilon \quad (j\geq 1)\\
\psi((u_i,v_i)_{i=1,\ldots,m})(0) &:= \epsilon\\
(y_1,y_2)\underline{\texttt{head-of}}(x) &:= \exists x'.x=f(f(y_1)+f(f(y_2)))+x'\\
(y_1,y_2,x')\underline{\texttt{sub-of}}(x) &:= x=f(f(y_1)+f(f(y_2)))+x'\vee\\
&\qquad \exists x''.x=x''+f(f(y_1)+f(f(y_2)))+x'
\end{aligned}$$

Die Definition der Formel $\underline{\texttt{nonempty}}$ ist etwas umständlich, da wir eine zu

$$\exists y_1,y_2(y_1,y_2)\underline{\texttt{head-of}}(x)$$

in $\mathcal{T}(\Sigma,\emptyset)/A(+)$ äquivalente Π_1-Formel finden müssen. In der folgenden Definition von $\underline{\texttt{nonempty}}$ geben wir jeweils in geschweiften Klammern als Kommentar das Term-Muster an, daß von der bis dahin gegebenen Formel eingegrenzt wird:

$\underline{\texttt{nonempty}}(x) := \forall y_1,y_2,y_3,y_4.$

	$x\neq\epsilon\wedge x\neq f(y_1)\wedge x\neq\epsilon+y_1$	$\{x\sim f(z_1)+z_2\}$
$\wedge$	$x\neq f(\epsilon)+y_1\wedge x\neq f(f(y_1))+y_2$	$\{x\sim f(z_1+z_2)+z_3\}$
$\wedge$	$x\neq f(\epsilon+y_1)+y_2$	$\{x\sim f(f(z_1)+z_2)+z_3\}$
$\wedge$	$x\neq f(y_1+y_2+y_3)+y_4\wedge x\neq f(y_1+\epsilon)+y_2$	$\{x\sim f(f(z_1)+f(z_2))+z_3\}$
$\wedge$	$x\neq f(y_1+f(\epsilon))\wedge x\neq f(y_1+f(y_2+y_3))+y_4$	$\{x\sim f(f(z_1)+f(f(z_2)))+z_3\}$

□

In den bisherigen Anwendungen haben wir jeweils eine einheitliche Kodierung der Folgen benutzt. Unserere letzte Anwendung zeigt die Verwendung von "privaten" Kodierungen der Teilfolgen einer jeweiligen Folge. Wie bereits in der Einführung, Abschnitt 1.6, ausgeführt, ist dies ein künstliches Beispiel, das nur dazu dient, einen Aspekt der hier vorgestellten Methode zu illustrieren.

Es sei $F := \langle 0(0), 1(0), +(2), *(2)\rangle$ und $P := \langle =(2), \leq(2)\rangle$. Wir betrachten das Modell N der natürlichen Zahlen. Bevor wir die Simulation der Worte in den natürlichen Zahlen angeben, benötigen wir zwei abkürende Schreibweisen:

$$\begin{aligned}
\overline{a}(t) &:= t+t\\
\overline{b}(t) &:= t+t+1
\end{aligned}$$

Die Grundidee der Darstellung der Worte besteht darin, ein Wort über einem binären Alphabet als Binärzahldarstellung einer natürlichen Zahl aufzufassen.

$$\begin{aligned}
\phi(\sigma_0\cdots\sigma_n) &:= \overline{\sigma_n}(\cdots\overline{\sigma_0}(1))\cdots)\\
\underline{\texttt{is-}\epsilon}(x) &:= x=1\\
(y)\underline{\sigma_0\cdots\sigma_n}(x) &:= y=\overline{\sigma_n}(\cdots\overline{\sigma_0}(x))\cdots)\\
\underline{\texttt{finite}}(x) &:= \textsc{True}
\end{aligned}$$

Beispielsweise ist $\phi(aaaba) = 34$, das ist 100010 in Binärschreibweise. Um Folgen darzustellen, benutzen wir Gödels β-Prädikat ([Göd31]):

$$\beta(x_1, x_2, l, x) := \exists q.x_1 = q * (1 + (l+1) * x_2) + x \wedge x < 1 + (l+1) * x_2$$

Den Beweis der fundamentalen Eigenschaft des β-Prädikats findet man neben [Göd31] beispielsweise in [End72]:

> Für jede Folge $a_0, \ldots, a_n$ von natürlichen Zahlen gibt es Zahlen c, d, so daß für alle $i \leq n$:
>
> $$I_G \models \beta[c, d, i, x] \quad \Leftrightarrow \quad x = a_i$$

Damit können wir nun die Kodierung der Folgen angeben:

$$\psi((u_i, v_i)_{i=1,\ldots,m})(j) \quad := \quad (c, d, 2 * j + 1)$$

wobei c, d die der Folge

$$(0, 0, \phi(u_1), \phi(v_1), \ldots, \phi(u_m), \phi(v_m))$$

entsprechenden Werte sind.

$$\begin{aligned}
\underline{\texttt{nonempty}}(c, d, n) &:= n \geq 1 \\
(y_1, y_2)\underline{\texttt{head-of}}(c, d, n) &:= \beta(c, d, n + n, y_1) \wedge \beta(c, d, n + n + 1, y_2) \\
(y_1, y_2, (c', d', n'))\underline{\texttt{sub-of}}(c, d, n) &:= c' = c \wedge d' = d \wedge n' < n \wedge (y_1, y_2)\underline{\texttt{head-of}}(c', d', n' + 1)
\end{aligned}$$

Als Ergebnis erhalten wir hier also die Unentscheidbarkeit des Σ_2 Fragments der vollständigen Zahlentheorie. Man beachte, daß der Bildbereich von ϕ die Menge $\mathbb{N} \setminus \{0\}$ ist, damit sind insbesondere ϕ und ψ nicht disjunkt.

8.3 Abschließende Bemerkungen

In den beiden vorhergehenden Abschnitten haben wir zwei Methoden für Unentscheidbarkeitsbeweise vorgestellt. Während die erste Methode für Modelle mit "ungeordneten" Trägermengen wie beispielsweise die Quotiententermalgebra modulo AC angebracht ist, paßt die zweite Methode besser auf Modelle mit einem Ordnungskonzept. Angesichts der Tatsache, daß die zweite Methode Unentscheidbarkeitsresultate von kleineren Fragmenten als die erste Methode ermöglicht, stellt sich hier die Frage, warum wir die zweite Methode nicht für den Beweis von Satz 11 verwandt haben.

Der Grund dafür ist, daß wir die *potentiell* geringere Anzahl von Quantoralternierungen bei der zweiten Methode nur dann wirklich ausschöpfen können, wenn wir wirklich *einfache* Formeln $\underline{\texttt{nonempty}}$, $\underline{\texttt{sub-of}}$ und $\underline{\texttt{head-of}}$ mit den geforderten Eigenschaften finden können. Genauer erhalten wir eine Formel $\underline{\texttt{solvable}}_P$ in Σ_3 genau dann, wenn $\underline{\texttt{nonempty}}$ im $\Pi_2 \cup \Sigma_1$ Fragment und sowohl $\underline{\texttt{sub-of}}$ als auch $\underline{\texttt{head-of}}$ im $\Pi_1 \cup \Sigma_2$ Fragment enthalten sind. Voraussetzung ist

dabei, daß die Formeln $\underline{\texttt{is-}\epsilon}$, $\underline{v}$ und $\underline{\texttt{finite}}$ keine weiteren Quantoralternierungen verursachen, wovon wir im allgemeinen auch ausgehen können. Im Falle von Satz 11 ist es uns jedoch nicht gelungen, hinreichend einfache Teilformeln zu finden, die mit der zweiten Methode zur Unentscheidbarkeit eines kleineren Fragments als Σ_4 geführt hätten.

An dieser Stelle möchten wir herausstellen, daß die hier vorgestellte Methode einige Eigenschaften einer Reduktion, die man vielleicht erwartet hätte, *nicht* erfordert. Damit möchten wir zeigen, daß eine systematische Untersuchung der Rückführungsbeweise zu einer Vereinfachung führt, da sie die kritischen Punkte eines solchen Beweises lokalisiert.

- Wir benötigen *keine* allgemeine binäre Operation für die Konkatenation zweier Worte.
- Die Kodierungen von Worten und Folgen müssen *nicht* disjunkt sein.
- Es ist *nicht* erforderlich, die Bildbereiche der Kodierungen ϕ und ψ durch Formeln zu beschreiben. Insbesondere ist es nicht notwendig, mit einer Formel auszudrücken, daß die Elemente einer Folge tatsächlich Worte sind.
- Eine explizite Charakterisierung der *endlichen* Folgen ist nicht notwendig.

Der Unentscheidbarkeitsbeweis von [Ven87] benutzt hingegen Teilformeln, die *explizit* den Aufbau der Terme, die P Konstruktionen repräsentieren sollen, reglementieren. In dem hier vorgestellten Ansatz ist dies nicht notwendig, wir erhalten daher einen einfacheren Beweis von Venkataramans Resultat.

Der Ausgangspunkt unserer Untersuchungen war das Postsche Korrespondenzproblem. Für Rückführungsbeweise dieser Art kommen natürlich auch andere unentscheidbare Probleme in Frage (siehe [Dav77]). Man könnte beispielsweise das uniforme Halteproblem für Turingmaschinen mit der folgenden Idee auf das Entscheidungsproblem einer Theorie zurückführen: Eine Berechnung einer Turingmaschine terminiert, falls es eine endliche Folge von Konfigurationen gibt, von denen die erste und die letzte von einer ausgezeichneten Form und jeweils zwei benachbarte durch eine Art von lokaler Transformation verknüpft sind. Eine Konfiguration können wir nun als Paar von Worten darstellen, nämlich jeweils als das Wort rechts, bzw. links vom Kopf. Die benötigten Datentypen (Worte und Folgen) wären hier also im Prinzip die gleichen wie im Postschen Korrespondenzproblem. Wir haben uns hier für das Postsche Korrepondenzproblem entschieden, da dies zu einer einfacheren Darstellung führt.

Ein anderes beliebtes unentscheidbares Problem ist die vollständige Zahlentheorie, d. h. die Theorie des Modells der natürlichen Zahlen. Man kann beispielsweise das Resultat von [Mat70] über die Unentscheidbarkeit von Hilberts Zehnten Problem benutzen und das Σ_1 Fragment der vollständigen Zahlentheorie auf die Theorie des fraglichen Modells reduzieren. Tatsächlich wurde eine solche Technik bereits in [Qui46] benutzt. Dort wurde die vollständige Zahlentheorie auf die Theorie der Konkatenation von Worten über dem binären Alphabet $\{a, b\}$ zurückgeführt.

Die Zahl n wird dabei in das aus n a's bestehende Wort kodiert, so daß die Addition zweier Zahlen gerade der Konkatenation zweier Worte entspricht. Zur Darstellung der Multiplikation werden aber Listen von Tupeln von Worten benötigt. Die Idee hierbei ist es in einem gewissen

Sinne, die Multiplikation durch Angabe einer Berechnungsfolge auszudrücken. Somit scheint diese Technik unserer sehr ähnlich zu sein, der springende Punkt bei [Qui46] ist jedoch, daß eine allgemeine binäre Konkatenation in dem betrachteten Modell zur Verfügung steht, so daß keine Quantoren für die Darstellung der Addition benötigt werden. In den hier betrachteten Anwendungen haben wir aber meist nur eine oder mehrere einstellige "Nachfolgerfunktionen" zur Vefügung, so daß wir alleine schon für die Addition eine Listenkonstruktion und Quantoren benötigen würden.

Wir wollen nun abschließend zeigen, daß die Theorie einer Grundtermalgebra modulo AC in einem bestimmten Spezialfall entscheidbar ist:

Satz 17 *Es sei Σ eine Signatur der Form*

$$\Sigma = \langle c_1(0), \ldots, c_n(0), +(2) \rangle$$

und $AC(+)$ die Axiome der Assoziativität und Kommutativität von $+$.
Dann ist $Th(\mathcal{T}(\Sigma, \emptyset)/AC(+))$ entscheidbar.

Der Beweis verallgemeinert die Beobachtung von [Com88], daß diese Algebra im Falle *einer* Konstanten isomorph ist zum Modell der Presburger Arithmetik.

Beweis: Die Algebra $\mathcal{T}(\Sigma, \emptyset)/AC(+)$ ist offenbar zu der folgenden Algebra N^n isomorph:

- die Trägermenge von N^n ist $\mathbb{N}^n$, d. h. die Menge der n-Tupel von natürlichen Zahlen,
- $+$ wird als die komponentenweise Addition von Tupeln von natürlichen Zahlen interpretiert, und
- einer Konstanten c_i wird der Wert

  $$(0, \ldots, 0, \underset{\substack{\uparrow \\ \text{Position } i}}{1}, 0, \ldots, 0)$$

  zugewiesen.

Es sei nun Σ_0 die Signatur der Presburger Arithmetik, d.h. $\Sigma_0 = \langle 0(0), 1(0), +(2) \rangle$, und N das Modell der Presburger Arithmetik. Wir geben nun eine effektive Transformation von $Th(N^n)$ auf $Th(N)$ an. Da $Th(N)$ entscheidbar ist ([Pre29]), folgt hieraus die Entscheidbarkeit von $Th(N^n) = Th(\mathcal{T}(\Sigma, \emptyset)/AC(+)$.

Es sei X eine Σ-Variablenmenge und $X_0 := \{x^i \mid x \in X, 1 \leq i \leq n\}$. Wir definieren $\phi_i : \mathcal{T}(\Sigma, X) \to \mathcal{T}(\Sigma_0, X_0)$ für $1 \leq i \leq n$ und $\psi : \mathcal{S}en(\Sigma, X) \to \mathcal{S}en(\Sigma_0, X)$ durch

$$\begin{aligned} \phi_i(x) &= x^i \\ \phi_i(c_j) &= \begin{cases} 1 & \text{falls } i = j \\ 0 & \text{falls } i \neq j \end{cases} \\ \phi_i(t + r) &= \phi_i(t) + \phi_i(r) \end{aligned}$$

$$\begin{aligned}
\psi(t = r) &= \phi_1(t) = \phi_1(r) \wedge \ldots \wedge \phi_n(t) = \phi_n(r) \\
\psi(\neg w) &= \neg\psi(w) \\
\psi(w_1 \wedge w_2) &= \psi(w_1) \wedge \psi(w_2) \\
\psi(\exists x \,.\, w) &= \exists x^1, \ldots, x^n \,.\, \psi(w)
\end{aligned}$$

Man sieht nun leicht, daß $N^n \models w$ genau dann gilt, wenn $N \models \psi(w)$. □

In Abschnitt 1.6 haben wir einige in Bezug auf diese Arbeit interessante Entscheidbarkeits- und Unentscheidbarkeitsresultate erwähnt. Im Falle der Theorie einer Grundtermalgebra mit dem Teiltermprädikat ist es uns gelungen, einen vollständigen Überblick über die entscheidbaren und die unentscheidbaren Fälle zu bekommen. Bei anderen Theorien bleiben aber noch Lücken, wir nennen hier deshalb einige (Klassen von) Fragmenten von Theorien, für die die Entscheidbarkeit noch ein offenes Problem ist:

- Das Σ_2-Fragment der Theorie einer Grundtermalgebra modulo AC, wobei die Signatur mindestens eine Konstante, ein nicht nullstelliges Funktionszeichen und ein binäres AC Funktionszeichen enthält.
- Fragmente mit mindestens einer Quantoralternierung der Theorie einer Grundtermalgebra mit einer *totalen* lexikographischen Pfadordnung. Die Entscheidbrakeit des Σ_1 Fragments wurde in [Com90b] gezeigt, siehe auch [JO91] für Erweiterungen.
- Fragmente mit höchstens zwei Quantoralternierungen der Theorie einer Grundtermalgebra mit einer *partiellen* lexikographischen Pfadordnung. Unser Resultat (Satz 11) zeigt nur die Unentscheidbarkeit des Σ_4-Fragments.

9 Beziehungen zwischen $\models$ und $[m\models$

In den vergangenen Abschnitten haben wir den Begriff der schwächsten Parameterbedingung eines Satzes bzgl. eines Moduls und eines Domainoperators untersucht. Dabei handelt es sich um eine "starke" Verbindung zwischen unserer Logik und der Prädikatenlogik erster Stufe, denn eine schwächste Parameterbedingung eines Satzes w stellt in gewissem Sinne eine äquivalente Übersetzung von w in die Prädikatenlogik erster Stufe dar. Falls ein Modul m die Eigenschaft hat, daß jeder Satz $w \in \mathcal{S}en_m$ eine $\Im^f, m$-schwächste Parameterbedingung hat, dann besitzt $[m\models$ die gleiche Aussagekraft wie $\models$. Wir haben gesehen, daß dies im allgemeinen nicht der Fall ist. Wir werden deshalb in diesem Kapitel schwächere Beziehungen zwischen $[m\models$ und $\models$ untersuchen.

Im Bereich der Hoare Logik für die partielle Korrektheit von Programmen hat sich der Begriff der schwächsten *ausdrückbaren* Parameterbedingung ([Sie82]) als eine nützliche Abschwächung des Begriffes der schwächsten Parameterbedingung erwiesen. Wir führen deshalb in Abschnitt 9.1 die analoge Verallgemeinerung an unserem Begriff der schwächsten Parameterbedingung durch.

In Abschnitt 9.2 betrachten wir dann einige andere mögliche Zusammenhänge zwischen $\models$ und $[m\models$. Dabei werden wir auch eine hinreichende und notwendige Bedingung für die Existenz von spabs erhalten.

9.1 Schwächste ausdrückbare Parameterbedingungen

Eine schwächste ausdrückbare Parameterbedingung eines Satzes w ist in gewissem Sinne ein minimales Element der Menge der Parameterbedingungen von w:

Definition 35 *Es sei m ein Modul mit Signatur (Σ_P, Σ_E), $\Im$ ein Domainoperator und $w \in \mathcal{S}en_m$. Ein Satz $v \in \mathcal{P}Sen_m$ ist eine $\Im, m$-*scwächste ausdrückbare Parameterbedingung *von w, abgekürzt* sapab, *falls*

- *v eine $\Im, m$-Parameterbedingung von w ist und*
- *$\Im_{\Sigma_P} \models v' \supset v$ für alle $\Im, m$-Parameterbedingungen v' von w gilt.*

Die nächsten beiden Lemmata folgen unmittelbar aus Definition 35:

Lemma 50 *Schwächste ausdrückbare Parameterbedingungen sind bis auf Äquivalenz eindeutig, das heißt: Sei m ein Modul mit Signatur (Σ_P, Σ_E), $\Im$ ein Domainoperator und $w \in \mathcal{S}en_m$. Dann gilt für alle $\Im, m$-schwächsten ausdrückbaren Parameterbedingungen $v_1, v_2 \in \mathcal{P}Sen_m$ von w: $\Im_{\Sigma_P} \models v_1 \mathrel{\text{⫤}} v_2$.*

Der Begriff der schwächsten ausdrückbaren Parameterbedingung verallgemeinert den Begriff der schwächsten Parameterbedingung:

Lemma 51 *Es sei m ein Modul mit Signatur (Σ_P, Σ_E), $\Im$ ein Domainoperator und $w \in \mathcal{S}en_m$. Jede $\Im, m$-spab von w ist auch eine $\Im, m$-sapab von w.*

Wir werden später (Theorem 18) sehen, daß eine sapab auch dann noch existieren kann, wenn es keine spab mehr gibt.

Im Vergleich zu einem Beweis, daß zu einem Satz w eine spab nicht existiert, ist ein Beweis der Nichtexistenz einer sapab mit einer prinzipiellen Schwierigkeit verbunden: Im Fall der spab können wir die Tatsache ausnutzen, daß wir die Klasse der Modelle (im Sinne von $\models$) einer spab von w genau kennen, dies sind nämlich gerade die Modelle (im Sinne von $[m\models$) des Satzes w. Wir können somit die bekannten Eigenschaften (insbesondere die Kompaktheit) der Prädikatenlogik erster Stufe benutzen, um die Nichtexistenz einer spab zu beweisen. Im Falle der sapab ist die Sache nicht so einfach, denn die Klasse der Modelle (im Sinne von $\models$) einer sapab von w ist im allgemeinen nur eine Teilklasse der Klasse der Modelle von w im Sinne von $[m\models$. Um eine echte Teilklasse handelt es sich dabei genau dann, wenn eine spab nicht existiert. Das nächste Lemma gibt ein Beweisprinzip für die *Nicht*existenz einer sapab. Auf Grund von Lemma 51 kann diese Methode auch dazu benutzt werden, die Nichtexistenz einer spab zu beweisen.

Lemma 52 *Es sei m ein Modul mit Signatur (Σ_P, Σ_E), $\Im$ ein Domainoperator und $w \in \mathcal{S}en_m$. m' sei eine Erweiterung von m mit Signatur (Σ_Q, Σ_F) um eine Menge von Konstanten und $T \subseteq \mathcal{P}Sen_{m'}$ mit*

1. *Für jede in $\Im_{\Sigma_Q}$ konsistente Vervollständigung $R \subseteq \mathcal{P}Sen_{m'}$ von T in $\Im_{\Sigma_Q}$ gibt es ein $A' \in \Im_{\Sigma_Q}$ mit $A' [m\models R \cup \{\neg w\}$ und*
2. *für jede endliche Teilmenge $N \subseteq T$ gibt es eine $\Im, m$-Parameterbedingung $v' \in \mathcal{P}Sen_m$ von w, so daß $N \cup \{v'\}$ konsistent in $\Im_{\Sigma_Q}$ ist.*

Dann gibt es keine $\Im, m$-sapab von w.

Beweis: Wir nehmen an, v wäre eine $\Im, m$-sapab von w. Da $\Im_{\Sigma_P} \models v' \supset v$ für jede $\Im, m$-Parameterbedigung v' von w, gilt wegen der Erweiterungseigenschaft von $\Im$ auch $\Im_{\Sigma_Q} \models v' \supset v$. Also ist mit Annahme (2) für jede endliche Teilmenge N von T die Menge $N \cup \{v\}$ konsistent in $\Im_{\Sigma_Q}$. Auf Grund der Kompaktheit von $\Im_{\Sigma_Q}$ ist dann auch $T \cup \{v\}$ konsistent in $\Im_{\Sigma_Q}$, das heißt es gibt ein $A \in \Im_{\Sigma_Q}$ mit $A \models T \cup \{v\}$. Da $Th(A)$ eine in $\Im_{\Sigma_Q}$ konsistente Vervollständigung von T ist, hat nun nach (1) $Th(A)$ ein Modell $A' \in \Im_{\Sigma_Q}$ mit $A' [m'\models \neg w$, und wegen der Erweiterungseigenschaft von $\Im$ gilt dann auch $A' |_{\Sigma_P} [m\models \neg w$.

Dies widerspricht unserer Annahme, daß $v \in Th(A |_{\Sigma_P}) = Th(A' |_{\Sigma_P})$ eine $\Im, m$ Parameterbedingung von w ist. □

Wir demonstrieren diese Beweistechnik an einem Beispiel:

Lemma 53 *m sei das Modul aus Abbildung 8, Seite 57, und*

$$w := \forall x \in elem\,.\,isstandard(x) = true$$

1. *Es gibt keine $\Im^{st}, m$-sapab von w*
2. *Es gibt keine $\Im^{st}, m$-sapab von $\neg w$*

Beweis: Σ_P bezeichne die Parametersignatur von m. Für beide Behauptungen benutzen wir Lemma 52 .

(1): Wir definieren m' als die Erweiterung von m um die Konstante $a :\rightarrow elem$. $\Sigma_Q := \Sigma_P \cup \{a\}$. Für jedes $i \geq 0$ sei

$$r_i := \bigwedge_{j=0}^{i} pred^j(a) \neq 0$$

r_i drückt aus, daß die Auswertung von $isstandard(a)$ nach i rekursiven Aufrufen von $isstandard$ noch nicht terminiert hat. Es sei $N := \bigcup_{i \geq 0} r_i$. In jedem Modell A von N in $\Im^{st}_{\Sigma_Q}$ gilt

$$A\ [m|\!=\ isstandard(a) = \bot$$

und daher auch $A\ [m|\!=\ \neg w$, somit ist insbesondere auch die erste Bedingung von Lemma 52 erfüllt.

Zum Beweis der zweiten Bedingung betrachten wir für ein beliebiges $i \geq 0$

$$v_i := \forall x \in elem\,.\, \bigvee_{j=0}^{i} pred^j(x) = 0$$

Jedes v_i ist eine $\Im^{st}, m$-Parameterbedingung von w. Für eine endliche Teilmenge $T \in N$ sei i der größte Index der in T vorkommenden r_j. Dann ist $N \cup \{v_{i+1}\}$ konsistent in $\Im^{st}_{\Sigma_Q}$, ein Modell ist beispielsweise die Standardisierung der Algebra Nat_{i+1}, definiert durch:

$$\begin{aligned}
elem^{Nat_{i+1}} &:= \{0, \ldots, i+1\} \\
0^{Nat_{i+1}} &:= 0 \\
a^{Nat_{i+1}} &:= i+1 \\
pred^{Nat_{i+1}}(x) &:= \begin{cases} x \perp 1 & \text{falls } x \neq 0 \\ 0 & \text{falls } x = 0 \end{cases}
\end{aligned}$$

(2): Für diesen Teil des Beweises benötigen wir keine Erweiterung des Modules m, es sei also $m' := m$. Wir definieren

$$\begin{aligned}
r_i &:= \quad \forall x \in elem\,.\, x \neq 0 \supset pred^i(x) \neq x \quad (i \geq 1) && (46)\\
A &:= \{ \quad pred(0) = 0\,, && (47)\\
&\quad\ \exists x \in elem . x \neq 0 \wedge pred(x) = 0\,, && (48)\\
&\quad\ \forall x \in elem\,.\, \exists y \in elem\,.\, x = pred(y)\,, && (49)\\
&\quad\ \forall x_1, x_2 \in elem\,.\, (pred(x_1) = pred(x_2) \wedge pred(x_1) \neq 0 \supset x_1 = x_2)\ \} && (50)
\end{aligned}$$

und

$$N := \{A\} \cup \{r_i \mid i \geq 1\}$$

N ist konsistent in $\Im^{st}_{\Sigma_P}$, ein Modell ist beispielsweise die Standardisierung der Algebra der natürlichen Zahlen (mit $pred(0) = 0$). Um zu zeigen, daß N nur unendliche Modelle in $\Im^{st}_{\Sigma_P}$ hat, konstruieren wir in einem beliebigen Modell B von N induktiv eine unendliche Menge $\{e_i \mid i \in \mathbb{N}\}$ von Elementen mit $e_0 = 0^B$, $e_i \neq 0^B$ für $i \geq 1$ und $pred(e_{i+1}) = e_i$ für alle i.

Wir wählen $e_0 = 0^B$ und definieren e_1 als ein von 0^B verschiedenes Element mit $pred(e_1) = 0^B$, das es wegen Axiom (48) geben muß. Falls wir e_i bestimmt haben, wählen wir für e_{i+1} ein Element mit $pred^B(e_{i+1}) = e_i$, ein solches Element muß es nach Axiom (49) geben. Da $e_i \neq 0^B$, ist wegen Axiom (47) auch $e_{i+1} \neq 0^B$. Damit ist nach Konstruktion und Axiom (46) $e_{i+1} \neq e_j$ für alle $j \geq i$.

Eine Untersuchung der Modelle von N ähnlich zu Theorem 31C in [End72] zeigt, daß alle Modelle von N in $\Im^{st}_{\Sigma_P}$ der Kardinalität $\aleph_1$ aus genau einer Kopie der natürlichen Zahlen plus $\aleph_1$ vielen Kopien der ganzen Zahlen, (Z-chains in [End72]) bestehen. Hierbei benutzen wir jetzt auch das Axiom (50). Somit sind alle Modelle von N der Kardinalität $\aleph_1$ in $\Im^{st}_{\Sigma_P}$ isomorph. Aus dem Łoś-Vaught Test (Korollar 4) folgt dann, daß N eine vollständige Theorie ist. Damit hat jede Vervollständigung von N als ein Modell die Standardisierung der Algebra der natürlichen Zahlen, in diesem Modell ist aber der Satz $\neg\neg w$, also w, gültig. Damit ist die erste Bedingung von Lemma 52 bewiesen.

Zum Beweis der zweiten Bedingung von Lemma 52 definieren wir nun für jedes $i \geq 1$

$$v_i := \exists x \in elem\,.\, pred^i(x) = x \wedge \bigwedge_{j=0}^{i-1} pred^j(x) \neq 0$$

Jedes v_i ist eine $\Im^{st}, m$-Parameterbedingung von w. Für eine endliche Teilmenge $T \subseteq N$ sei i der größte Index der in T vorkommenden r_i. Dann hat $N \cup \{v_{i+1}\}$ ein Modell in $\Im^{st}$, nämlich die Standardisierung der folgenden Algebra C:

$$\begin{aligned}
elem^{C_{i+1}} &:= \{n_i \mid i \in \mathbb{N}\} \cup \{c_1, \ldots, c_{i+1}\} \\
0^{C_{i+1}} &:= n_0 \\
pred^{C_{i+1}}(x) &:= \begin{cases} n_{j-1} & \text{falls } x = n_j\,,\ j \geq 1 \\ n_0 & \text{falls } x = n_0 \\ c_{j-1} & \text{falls } x = c_j\,,\ 2 \leq j \leq i+1 \\ c_{i+1} & \text{falls } x = c_1 \end{cases}
\end{aligned}$$

□

9.2 Weitere Beziehungen

Satz 18 *Es sei m ein Modul mit Signatur (Σ_P, Σ_E), $\Im$ ein Domainoperator und $w \in Sen_m$. Betrachte die folgenden Aussagen:*

1. *Es gibt eine $\Im, m$-spab von w.*
2. *Für alle $\mathcal{C} \subseteq \Im_{\Sigma_P}$: $\mathcal{C}\ [m\models w \Longrightarrow Th(\mathcal{C}) \cup Th_m(\Im_{\Sigma_P}) \models w$*
3. *Es gibt eine $\Im, m$-sapab von w.*
4. *Für alle $A \in \Im_{\Sigma_P}$: $A\ [m\models w \Longrightarrow Th(A) \cup Th_m(\Im_{\Sigma_P}) \models w$*
5. *Für alle $A, A' \in \Im_{\Sigma_P}$: $A\ [m\models w \wedge Th(A) = Th(A') \Longrightarrow A'\ [m\models w$*

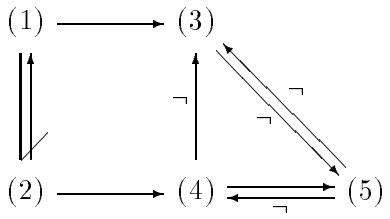
Dann gelten die folgenden Implikationen:

$$(3) \Longleftarrow (1) \Longleftrightarrow (2) \Longrightarrow (4) \Longrightarrow (5)$$

Es gelten keine weiteren Implikationen zwischen den Aussagen (1) - (5), außer denen, die mit der Transitivität aus den obigen Implikationen folgen.

Die Bedeutung von spabs und sapabs kennen wir bereits. (2) sagt aus, daß der in einer Klasse von Algebren bzgl. $[m\models$ gültige Satz w bereits eine prädikatenlogische Folgerung aus der Theorie erster Stufe der Klasse und der Menge der m-allgemeingültigen Sätze ist. Aussage (4) schränkt dies auf einelementige Klassen von Algebren ein. (5) besagt, daß die m-Gültigkeit von w in einer Algebra nur abhängt von der Theorie erster Stufe der Algebra.

Beweis: Das folgende Diagramm stellt den "Plan" unseres Beweises dar. Ein Negationszeichen $\neg$ an einem Pfeil bedeutet dabei, daß die Implikation in dieser Richtung *nicht* gilt, was wir dann jeweils durch Angabe eines Gegenbeispiels beweisen werden.



(1) $\Rightarrow$ (2):

v sei eine $\Im, m$-spab von w. Wegen $\Im_{\Sigma_P}\ [m\models v \supset w$ gilt dann $(v \supset w) \in Th_m(\Im_{\Sigma_P})$. Falls $\mathcal{C}\ [m\models w$, dann gilt wegen der Definition von spab $\mathcal{C} \models v$, also mit Hilfe der Schlußregel der Aussagenlogik ("modus ponens") $Th(\mathcal{C}) \cup Th_m(\Im_{\Sigma_P}) \models w$.

PAR **SORTS** s
OPNS $a: \rightarrow s$
$b: \rightarrow s$
$f: s \rightarrow s$
BODY **FCTS** $g: s \rightarrow bool$
PROG $g(x) \Leftarrow$ `if` $x = a$ `then` $true$
`else` $g(f(x))$

Abbildung 34: Eine Moduldefinition für den Beweis von Satz 18

(2) $\Rightarrow$ (1):
Es sei

$$\mathcal{C} := \{A \in \Im_{\Sigma_P} \mid A\ [m\models\ w\}$$

Nach Annahme gilt also $Th(\mathcal{C}) \cup Th_m(\Im_{\Sigma_P}) \models w$. Auf Grund des Kompaktheitssatzes der Prädikatenlogik erster Stufe gibt es dann eine endliche Teilmenge $V \subseteq Th(\mathcal{C})$ mit $V \cup Th_m(\Im_{\Sigma_P}) \models w$. Wir zeigen, daß $v := \bigwedge_{p \in V} p$ eine $\Im, m$-spab von w ist.

Falls $A \models v$, dann gilt, da $A\ [m\models\ Th_m(\Im_{\Sigma_P})$, nach Konstruktion $A\ [m\models\ w$, also ist v eine $\Im, m$-Parameterbedingung von w. Falls andererseits $A\ [m\models\ w$ für ein $A \in \Im_{\Sigma_P}$, dann gilt wegen der Definition von $\mathcal{C}$: $A \in \mathcal{C}$, also $A \models v$ (da $v \in Th(\mathcal{C})$). Also ist v eine $\Im, m$-spab von w.

(1) $\Rightarrow$ (3):
Dies wurde in Lemma 51 gezeigt.

(2) $\Rightarrow$ (4):
(4) ist nur ein Spezialfall von (2), für eine gegebene Algebra A wählen wir $\mathcal{C} := \{A\}$.

(4) $\Rightarrow$ (5):
Es sei $A\ [m\models\ w$ und $Th(A) = Th(A')$. Nach Annahme gilt $Th(A) \cup Th_m(\Im_{\Sigma_P}) \models w$, also $Th(A') \cup Th_m(\Im_{\Sigma_P}) \models w$ und somit $A'\ [m\models\ w$.

(4) $\not\Rightarrow$ (3):
Betrachte das Modul m gemäß Abbildung 34 und den Satz $w := (g(b) = true)$. Analog zu Lemma 53 können wir auch hier mit Hilfe von Lemma 52 zeigen, daß w keine $\Im^{st}, m$ sapab hat.

Andererseits ist (4) erfüllt, denn $Th_m(\Im^{st}_{\Sigma_P})$ enthält alle Sätze der Art $f^i(b) = a \supset g(b) = true$ für $i \in \mathbb{N}$. Falls nun für eine Algebra $A \in \Im^{st}_{\Sigma_P}$ $A\ [m\models\ w$ gilt, dann gibt es ein $i \in \mathbb{N}$ mit $A \models f^i(b) = a$, also $Th(A) \cup Th_m(\Im_{\Sigma_P}) \models w$.

PAR **SORTS** s
OPNS $0: \rightarrow s$
$f: s \rightarrow s$
$\leq: s, s \rightarrow bool$
BODY **FCTS** $r: s, s \rightarrow bool$
PROG $r(x,y) \Leftarrow$ `if` $x = y$ `then` $true$
`else` $r(f(x), y)$

Abbildung 35: Eine Moduldefinition für den Beweis von Satz 18

(3) $\not\Rightarrow$ (5):

Betrachte das Modul m aus Abbildung 35, Σ sei die Parametersignatur von m und $T \in \mathcal{P}Sen_m$ bezeichne den folgenden Satz:

$$\begin{array}{ll} & \forall y \in s\,.\, y \neq 0 \supset \exists x \in s\,.\, y = f(x) \\ \wedge & \forall x, y \in s\,.\, x \neq f(y) \supset (x \leq f(y)) = (x \leq y) \\ \wedge & \forall x \in s\,.\, x = 0 \vee (x \leq 0) = false \\ \wedge & \forall x, y \in s\,.\, x = y \supset (x \leq y = true) \\ \wedge & \forall x, y \in s\,.\, x \neq y \supset (x \leq y) \neq (y \leq x) \\ \wedge & \forall x, y, z \in s\,.\, (x \leq y) = true \wedge (y \leq z) = true \supset (x \leq z) = true \end{array}$$

Die in den ersten drei Zeilen angegebenen Teilformel geben die Eigenschaften von 0, f und $\leq$ ähnlich wie in der Algebra der natürlichen Zahlen an. Die letzten drei Teilformeln besagen, daß $\leq$ eine lineare Ordnung ist. Im Gegensatz zu Lemma 53 können wir den Łoś-Vaught Test zum Beweis der Vollständigeit von $\{T\}$ nicht benutzen, da für unendliche Kardinalzahlen Modelle dieser Kardinalität nun nicht mehr notwendig isomorph sind. Dies liegt daran, daß nun mit $\leq$ eine "reichere" Struktur der Modelle vorliegt, insbesondere existieren nun auch Relationen zwischen verschiedenen Ketten, während im Beweis von Lemma 53 die einzelnen Ketten vollständig isoliert waren.

Wir können aber mit einigen trivialen Modifikationen des Beweises von Satz 32A in [End72] zeigen, daß $\{T\}$ eine *Quantoreliminierung* für die Klasse $\Im^{st}_{\Sigma}$ erlaubt ([End72],[CK90]). Dies bedeutet insbesondere, daß T in $\Im^{st}_{\Sigma}$ vollständig ist. Es sei nun

$$w := T \wedge \forall x \in s\,.\, r(0,x) = true$$

`False` ist natürlich eine $\Im^{st}, m$-Parameterbedingung von w, wir zeigen daß es sogar bis auf Äquivalenz die einzige $\Im^{st}, m$-Parameterbedingung von w ist. Wir nehmen als an, v sei eine $\Im^{st}, m$-sapab von w und $A \models v$. Dann gilt insbesondere $A \models T$. Es sei nun A' die Standardisierung eines Nichtprimmodelles der Arithmetik. Es gilt $A' \models T$, und da T vollständig in $\Im^{st}_{\Sigma}$ ist, auch $A' \models v$.

Nun gilt aber $A' \,[m|\!=\, \neg w$, dies widerspricht der Annahme an v. Da `False` bis auf Äquivalenz die einzige $\Im^{st}_{\Sigma}$ Parameterbedingung von w ist, ist es also auch eine $\Im^{st}_{\Sigma}$-sapab von w.

Andererseits gilt (5) nicht: Es sei A die Standardisierung des Primmodells und A' die Standardisierung eines Nichtprimmodells der Arithmetik. A und A' haben die gleiche Theorie, aber $A\ [m\models\ w$ und $A'\ [m\models\ \neg w$.

(5) $\not\Rightarrow$ (4):

Es sei m das Modul von Abbildung 9, Seite 57, und Σ die Parametersignatur von m. Betrachte den Domainoperator $\Im^f$ und den Satz

$$w := \forall l \in list\,.\,\exists x \in elem\,.\,isin(x,l) = false$$

Aussage (5) ist erfüllt, denn für jede Algebra $A \in \Im^f_\Sigma$ gilt $A\ [m\models\ w$ genau dann, wenn *keiner* der Sätze

$$\exists x_1,\ldots,x_n \in elem\,.\,\forall y \in elem \bigvee_{i=1}^{n} y = x_i \qquad (i \geq 0)$$

ein Element von $Th(A)$ ist.

Wir nehmen an, Aussage (4) wäre erfüllt. A sei eine Algebra $A \in \Im^f_\Sigma$ mit unendlicher Kardinalität. Dann gilt $A\ [m\models\ w$, also nach (4):

$$Th(A) \cup Th_m(\Im^f_\Sigma) \models w$$

Auf Grund des Kompaktheitssatzes der Prädikatenlogik erster Stufe gibt es dann eine endliche Menge $V \subseteq Th(A)$, so daß

$$V \cup Th_m(\Im^f_\Sigma) \models w$$

Nach Lemma 3 können wir ohne Beschränkung der Allgemeinheit annehmen, daß $V \subseteq \mathcal{S}en(\Sigma', X)$, wobei Σ' nur aus der Sorte *elem* besteht und keine Funktionszeichen enthält. Σ' ist also die (Signatur der) Sprache der *reinen Gleichheitstheorie* (pure identity language, [CK90]). Es sei nun v die Konjunktion der in V enthaltenen Sätze.

Wir können nun zu jeder endliche Menge $N \subseteq \mathbb{N}$ eine Satz $\sigma(N) \in \mathcal{S}en(\Sigma', X)$ angeben, so daß für jede Algebra $A \in Alg_{\Sigma'}$:

$$A \models \sigma(N) \quad \Longleftrightarrow \quad \#(A) \in N$$

Nach Korollar 1.5.8 von [CK90] gibt es eine endliche Teilmenge $N \subseteq \mathbb{N}$, so daß v äquivalent zu $\sigma(N)$ oder zu $\neg\sigma(N)$ ist. In jedem der beiden Fälle hat v ein endliches Modell A' (da N endlich ist), also gilt $A'\ [m\models\ V \cup Th_m(\Im^f_\Sigma)$ und somit nach der Wahl von V: $A'\ [m\models\ w$. Dies ist ein Widerspruch, denn $B\ [m\models\ w$ gilt nur für unendliche Modelle B.

(5) $\not\Rightarrow$ (3):

Es sei m wieder das Modul aus Abbildung 9. Betrachte den Satz

$$w' = \exists l \in list\,.\,\forall x \in elem\,.\,isin(x,l) = true$$

Ebenso wie oben sehen wir auch hier, daß die Aussage (5) erfüllt ist. Andererseits hat w aber keine $\Im^f, m$-spab, wie wir auf Seite 57 gezeigt haben. Also hat w nach Lemma 51 auch keine $\Im^f, m$-sapab. □

Literatur

[ASCII83] American National Standards Institute. *The Programming Language Ada Reference Manual.* LNCS vol. 155. Springer, 1983.

[Apt90] Krzysztof R. Apt. Logic programming. In Jan van Leeuwen, Hrsg., *Handbook of Theoretical Computer Science*, Band 2, Seite 493–574. Elsevier, 1990.

[BF85] J. Barwise und S. Feferman, Hrsg. *Model-Theoretic Logics*. Perspectives in Mathematical Logic. Springer-Verlag, 1985.

[BG80] R. M. Burstall und J. A. Goguen. Semantics of CLEAR, a specification language. In D. Björner, Hrsg., *Abstract Softare Specifications*, Seite 292–332. Springer LNCS, vol. 86, 1980.

[BHSS87] Hans-Jürgen Bürckert, Alexander Herold, und Manfred Schmidt-Schauß. On equational theories, unification and decidability. In Pierre Lescanne, Hrsg., *Rewriting Techniques and Applications*, Seite 204–215. Lecture Notes in Computer Science, vol. 256, Springer, 1987.

[Bib75] Wolfgang Bibel. Prädikatives Programmieren. In H. Brakhage, Hrsg., *Automata Theory and Formal Languages: 2nd GI Conference*, Seite 274–283. Springer Verlag, Lecture Notes in Computer Science vol. 33, 1975.

[Bis86] Judy Bishop. *Data Abstraction in Programming Languages.* Addison–Wesley, 1986.

[BSS89] Hans-Jürgen Bürckert und Manfred Schmidt-Schauß. On the solvability of equational problems. SEKI Report SR-89-07, Universität Kaiserslautern, 1989.

[CG78] Bruno Courcelle und Irène Guessarian. On some classes of interpretations. *Journal of Computer and System Sciences*, 17:388–413, 1978.

[CK90] C. C. Chang und H. J. Keisler. *Model Theory.* Studies in Logic and the Foundations of Mathematics, vol. 73. North-Holland Publishing Company, third edition, 1990.

[CL88] Hubert Comon und Pierre Lescanne. Equational problems and disunification. Rapport de Recherche 904, INRIA CRIN, Nancy, France, September 1988.

[Com88] Hubert Comon. *Unification et Disunification. Théorie et Applications.* Dissertation, Institut National Polytechnique de Grenoble, Grenoble, France, 1988.

[Com90a] Hubert Comon. Disunification:a survey. Rapport de recherche no. 540, LRI, Université de Paris Sud, Januar 1990.

[Com90b] Hubert Comon. Solving inequations in term algebras. In *5th Symposium on Logic in Computer Science*, Seite 62–69. IEEE, 1990.

[Cou83] Bruno Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25(2):95–169, 1983.

[Dav77] Martin Davis. Unsolvable problems. In Jon Barwise, Hrsg., *Handbook of Mathematical Logic*, Kapitel C.2, Seite 567–594. North-Holland, 1977.

[Der82] Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 7:279–301, 1982.

[Der87] Nachum Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.

[Der89] Nachum Dershowitz. Completion and its applications. In Aït-Kaci und Maurice Nivat, Hrsg., *Resolution of Equations in Algebraic Structures*, Band 2 (Rewriting Techniques), Kapitel 2, Seite 31–85. Academic Press, 1989.

[DJ90] Nachum Dershowitz und Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, Hrsg., *Handbook of Theoretical Computer Science*, Band B, Seite 243–320. Elsevier, 1990.

[DJ91] Nachum Dershowitz und Jean-Pierre Jouannaud. Notations for rewriting. *EATCS-Bulletin*, 43:162–172, Februar 1991.

[DM79] Nachum Dershowitz und Zohar Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.

[Ebb85] H.-D. Ebbinghaus. Extended logics: The general framework. In *[BF85]*, Kapitel 2, Seite 25–76. 1985.

[EM85] H. Ehrig und B. Mahr. *Fundamentals of Algebraic Specification, vol. 1.* EATCS-Monographs on Theoretical Computer Science. Springer-Verlag, 1985.

[EM90] H. Ehrig und B. Mahr. *Fundamentals of Algebraic Specification, vol. 2.* EATCS-Monographs on Theoretical Computer Science. Springer-Verlag, 1990.

[End72] Herbert B. Enderton. *Mathematical Introduction to Logic.* Academic Press, 1972.

[Fag84] François Fages. Associative-commutative unification. In R. E. Shostak, Hrsg., *7th International Conference on Automated Deduction*, Seite 194–208, Napa, California, USA, 1984. Lecture Notes in Computer Science vol. 170.

[Fef86] Solomon Feferman, Hrsg. *Kurt Gödel, Collected Works.* Oxford University Press, 1986. 2 volumes.

[Flu85] J. Flum. Characterizing logics. In *[BF85]*, Kapitel 3, Seite 77–120. 1985.

[Göd31] Kurt Gödel. Über Formal Unentscheidbare Sätze der Pricipia Mathematica und Verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931. Reprinted in [Fef86].

[Gal86] Jean H. Gallier. *Logic for Computer Science*. Harper & Row, publishers, 1986.

[Gre75] Sheila A. Greibach. *Theory of Program Structures: Schemes, Semantics, Verification*. Lecture Notes in Computer Science, Vol. 35. Springer Verlag, 1975.

[GTW78] J. A. Goguen, J. W. Thatcher, und E. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, Hrsg., *Current Trends in Programming Methodology*, Band IV, Seite 80–10. Prentice-Hall, 1978.

[Gue79] Irène Guessarian. *Algebraic Semantics*. Lecture Notes in Computer Science, Vol. 99. Springer Verlag, 1979.

[Her71] Hans Hermes. *Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit*. Springer-Verlag, second edition, 1971.

[HMM86] Robert Harper, David MacQueen, und Robin Milner. Standard ML. Technical Report ECS-LFCS-86-2, Edinburgh University, 1986.

[JK91] Jean-Pierre Jouannaud und Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In J. Lassez und G. Plotkin, Hrsg., *Computational Logic: Essays in Honour of Alan Robinson*. MIT Press, 1991. to appear.

[JO91] Jean-Pierre Jouannaud und Mitsuhiro Okada. Satisfiability of systems of ordinal notation with the subterm property is decidable. In *Proceedings of ICALP 91*, 1991.

[JW78] Kathleen Jensen und Niklaus Wirth. *PASCAL User Manual and Report*. Springer-Verlag, second edition, 1978.

[Kau85] M. Kaufmann. The quantifier "there exist uncountably many" and some of its relatives. In *[BF85]*, Kapitel 4, Seite 123–176. 1985.

[Kei71] H. Jerome Keisler. *Model Theory for Infinitary Logic*. Studies in Logic and the Foundations of Mathematics, vol. 62. North-Holland Publishing Company, 1971.

[Kir85] Claude Kirchner. *Méthodes et Outils de Conception Systématique d'Algorithmes d'Unification dans les Théories Équationelles*. Dissertation, Centre de Récherche en Informatique de Nancy, 1985.

[Kow79] Robert A. Kowalski. Algorithm = logic + control. *Communications of the ACM*, 7(22):424–436, 1979.

[KT90] D. Kozen und J. Tiuryn. Logics of programs. In Jan van Leeuwen, Hrsg., *Handbook of Theoretical Computer Science, volume B*, Kapitel 14, Seite 789–840. Elsevier Science Publishers, 1990.

[LAB+81] Barbara Liskov, Rusell Atkinson, Toby Bloom, Eliot Moss, J. Craig Schaffert, Robert Scheifler, und Alan Snyder. *CLU reference manual.* LNCS vol. 114. Springer, 1981.

[Les88] Pierre Lescanne. Open problems from the first UNIF workshop, Val d'Ajol, 1988. Contribution 11 to the "rewriting list" electronic mail forum.

[LG86] Barbara Liskov und John Guttag. *Abstraction and Specification in Program Development.* MIT press, 1986.

[Lin69] P. Lindström. On extension of elementary logic. *Theoria*, 35:1–11, 1969.

[LL87] Thomas Lehmann und Jacques Loeckx. The specification language of OBSCURE. In D. Sannella und A. Tarlecki, Hrsg., *5th Workshop on Specification of Abstract Data Types*, Seite 131–153. Springer LNCS, vol. 332, 1987.

[LL90] Thomas Lehmann und Jacques Loeckx. OBSCURE, a specification language for abstract data types. Technical Report A 19-90, Universität des Saarlandes, 1990.

[Loe87] Jacques Loeckx. Algorithmic specifications: A constructive specification method for abstract data types. *ACM Transactions on Programming Languages and Systems*, 9(4), 1987.

[LPP70] D. C. Luckham, D. M. R. Park, und M. S. Paterson. On formalized computer programs. *Journal of Computer and System Sciences*, 4:220–249, 1970.

[LS87] Jacques Loeckx und Kurt Sieber. *The Foundations of Program Verification.* Wiley/Teubner, 2nd edition, 1987.

[LWF+91] Jacques Loeckx, Markus Wolf, Jürgen Fuchs, Annette Hoffmann, Liane Meiss, Joachim Philippi, Michael Stolz, und Jörg Zeyer. *The OBSCURE Manual.* Universität des Saarlandes, Februar 1991.

[Mac86] David MacQueen. Modules for standard ML. In [HMM86], 1986.

[Mak77] J. Makanin. The problem of solvability of equations in a free semi-group. *Akad. Nauk SSSR*, 232(2), 1977.

[Mal71] Anatoliĭ Ivanovič Malc'ev. Axiomatizable classes of locally free algebras of various type. In Benjamin Franklin Wells, Hrsg., *The Metamathematics of Algebraic Systems: Collected Papers 1936–1967*, Kapitel 23, Seite 262–281. North Holland, 1971.

[Mal79] Jerome Malitz. *Introduction to Mathematical Logic.* Undergraduate Texts in Mathematics. Springer, 1979.

[Mat70] Yu Matijacevič. Enumerable sets are diophantine. *Dokl. Akad. Nauk. SSSR*, 191:279–282, 1970.

[Nad85] M. Nadel. $\mathcal{L}_{\omega_1\omega}$ and admissible fragments. In *[BF85]*, Kapitel 8, Seite 271–316. 1985.

[Par72] D. L. Parnas. A technique for software module specifications with examples. *Communications of the ACM*, 15(5):330–336, 1972.

[Plo72] G. D. Plotkin. Building-in equational theories. In Bernard Meltzer und Donald Michie, Hrsg., *Machine Intelligence 7*, Seite 73–90. Edinburgh University Press, 1972.

[Pos46] Emil L. Post. A variant of a recursively unsolvable problem. *Bulletin of the AMS*, 52:264–268, 1946.

[Pre29] M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchen die Addition als einzige Operation hervortritt. In *Comptes Rendus du I Congres de Mathematiciens des Pays Slaves*, Seite 92–101, 1929.

[Qui46] W. V. Quine. Concatenation as a basis for arithmetic. *Journal of Symbolic Logic*, 11(4):105–114, 1946.

[Rab65] M. O. Rabin. A simple method for undecidability proofs and some applications. In Yehoshua Bar-Hillel, Hrsg., *Logic, Methodology and Philosophy of Science*, Seite 58–68. North-Holland, 1965.

[Rab69] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[Rog87] Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* MIT Press, second edition, 1987.

[Sha81] Mary Shaw, Hrsg. *Alphard: Form and Content.* Springer, 1981.

[Sie82] Kurt Sieber. Weakest expressible preconditions: a new tool for proving completeness results about Hoare-calculi. In *Proceedings of the 6th GI Conference on Theoretical Computer Science*, Seite 325–333. Lecture Notes in Computer Science, Vol. 145, 1982.

[Sie89] Jörg H. Siekmann. Unification theory. a survey. *Journal of Symbolic Logic*, 1989.

[Ste90] Guy Steele. *Common LISP: The Language.* Digital Press, second edition, 1990.

[Sti81] Mark E. Stickel. A unification algorithm for associative-commutative functions. *Journal of the ACM*, 28(3):423–434, 1981.

[Tar53a] Alfred Tarski. A general method in proofs of undecidability. In *[TMR53b]*, Seite 1–35. 1953.

[Tar53b] Alfred Tarski. Undecidability in the elementary theory of groups. In *[TMR53b]*, Seite 75–87. 1953.

[Tiu81] J. Tiuryn. A survey of the logic of effective definitions. In E. Engeler, Hrsg., *Proceedings of the Workshop on Logics of Programs*, Seite 198–245. Springer LNCS, vol. 125, 1981.

[TMR53a] Alfred Tarski, Andrzej Mostowski, und Raphael M. Robinson. Undecidability and essential undecidability in arithmetic. In *[TMR53b]*, Seite 37–74. 1953.

[TMR53b] Alfred Tarski, Andrzej Mostowski, und Raphael M. Robinson. *Undecidable Theories*. North-Holland, 1953.

[Tre86] Ralf Treinen. Ein Kalkül für Algorithmische Spezifikationen. Diplomarbeit, Universität des Saarlandes, 1986.

[Tul90] Sauro Tulipani. Decidability of the existential theory of infinite terms with subterm relation. Unveröffentlichtes Manuskript, Oktober 1990.

[Tur85] David A. Turner. Miranda: A non-strict functional language with polymorphic types. In Jean-Pierre Jouannaud, Hrsg., *IFIP International Conference on Functional Programming Languages and Computer Architecture*, Seite 1–16. Springer LNCS, vol. 201, 1985.

[Ven87] K. N. Venkataraman. Decidability of the purely existential fragment of the theory of term algebra. *Journal of the ACM*, 34(2):492–510, April 1987.

[Vui74] Jean Vuillemin. Correct and optimal implementations of recursion in a simple programming language. *Journal of Computer and System Sciences*, 9:332–354, 1974.

[Wir85] Niklaus Wirth. *Programming in MODULA-2*. Springer, third edition, 1985.

[WPP+83] Martin Wirsing, Peter Pepper, Helmut Partsch, Walter Dosch, und Manfred Broy. On hierarchies of abstract data types. *Acta Informatica*, 20:1–33, 1983.

Index

Abbildungsverzeichnis

Vereichnis der Lemmata, Korollare, Sätze und Definitionen