Rheinisch-Westfälische Technische Hochschule Aachen
Fachbereich 1
Lehrstuhl für Informatik II

Denotationelle und operationelle Semantiken für konstruktorbasierte funktionale Programmiersprachen erster Ordnung

Diplomarbeit
Olaf Chitil
Matrikelnummer 171864

Gutachter:
Prof. Dr. Klaus Indermark
Prof. Dr. Rita Loogen

Betreuer:
Dipl.-Inform. Thomas Noll

Aachen, den 8. Februar 1995

Hiermit versichere ich, daß ich die Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen benutzt sowie Zitate kenntlich gemacht habe.

Aachen, den

Olaf Chitil

Inhaltsverzeichnis

Kapitel 1

Einleitung

Eine Programmiersprache besteht aus zwei Komponenten: ihrer Syntax und ihrer Semantik.
Die **Syntax** beschreibt die Struktur eines wohlgeformten Programms, d. h. sie besagt, welcher Text überhaupt ein Programm ist. Diese Eigenschaft muß entscheidbar sein.
Die **Semantik** ordnet jedem Programm seine Bedeutung zu. Hierbei ist zu beachten, daß einerseits diese Zuordnung selbst als Semantik der Programmiersprache, andererseits aber auch die Bedeutung eines Programms als Semantik des Programms bezeichnet wird. Im allgemeinen besitzen mehrere verschiedene Programme die gleiche Semantik.
Lehrbücher vermitteln die Semantik einer Programmiersprache durch Erklärungen und Beispiele. Zum Erlernen einer neuen Programmiersprache ist eine derartige informale Beschreibung sicherlich gut geeignet. Eine vollständige, eindeutige und möglichst nicht zu umfangreiche Definition der Semantik einer Programmiersprache ist so jedoch nicht erreichbar. Die eingeschränkte Portabilität von Programmen — auch bei standardisierten Programmiersprachen wie Fortran, Pascal oder C — zeigt das Problem in der Praxis auf. Einige Aspekte werden in solchen Semantiken oft auch überhaupt nicht definiert, wie beispielsweise das Verhalten beim Zugriff auf ein Array-Element mit einem Index außerhalb des vorgegebenen Bereichs. Dies widerspricht jedoch unserer Forderung, daß die Semantik jedem syntaktisch korrekten Programm eine Bedeutung zuordnen soll. Da das Vorhandensein eines fehlerhaften Array-Zugriffs auch nicht entscheidbar ist, kann ein solches Programm auch nicht als syntaktisch fehlerhaft betrachtet werden.
Dagegen bauen **formale Semantiken** auf bekannten, wohlverstandenen mathematischen Strukturen auf. Sie ermöglichen die eindeutige und vollständige Definition einer Semantik zur Standardisierung einer Programmiersprache. Damit unterstützen sie die Entwicklung korrekter Interpreter und Compiler. Im Zusammenhang mit der Implementierung stehen auch semantikerhaltende Transformationen eines Programms zu Optimierungszwecken, deren Korrektheit erst mit formalen Semantiken beweisbar ist. Zur formalen Verifikation von Programmen sind formale Semantiken unverzichtbar. Schließlich führt eine formale Semantik auch zu einem viel größeren Verständnis einer Programmiersprache.

Imperative Programmiersprachen sind am Modell der Von-Neumann-Maschine orientiert. Ihre Semantiken lassen sich zwar formal beschreiben — insbesondere mit den sogenannten axiomatischen Semantiken —, aber in der Praxis sind derartige Semantiken sehr aufwendig und daher kaum einsetzbar.
Funktionale Programme sind dagegen maschinenunabhängige Spezifikationen. Wir wollen hier auf die Schilderung der sich daraus ergebenden Vorteile bezüglich der Anwendungsnähe und flexibleren Implementierbarkeit verzichten. Da funktionale Programmiersprachen aus mathematischen

Kalkülen, hauptsächlich dem λ-Kalkül, entstanden sind, sind sie formalen Beschreibungsmethoden weitaus zugänglicher. Insbesondere gestatten sie die Definition relativ einfacher formaler Semantiken, die praktisch einsetzbar sind.
Logische Programme sind ebenso maschinenunabhängig und in gewissem Sinne noch abstrakter als funktionale. Mit ihnen sind allerdings auch nicht-berechenbare Funktionen spezifizierbar. Auch die Verwendung von Semi-Entscheidungsverfahren erweist sich noch als ziemlich ineffizient, so daß Prolog-Implementierungen zum Beispiel statt der vollständigen, semantisch korrekten Breitensuche Tiefensuche im SLD-Baum verwenden.
Dagegen sind die funktionalen Programmiersprachen zugrunde liegenden Kalküle entworfen worden, um genau die berechenbaren Funktionen spezifizieren zu können. Es existiert auch ein universeller Algorithmus, konkret ein Interpreter oder Compiler, der jede durch ein Programm einer funktionalen Programmiersprache spezifizierte Funktion berechnen kann. Hierbei ist allerdings zu beachten, daß im Church'schen Sinne universelle Programmiersprachen auch partielle Funktionen spezifizieren. Da nicht entscheidbar ist, welche Argumente im Definitionsbereich liegen, sind nichtterminierende Berechnungen unvermeidbar. Die Definition einer Programmiersprache, die genau die Spezifikation der totalen berechenbaren Funktionen gestattet, ist unter den genannten Voraussetzungen unmöglich. Ist die Syntax einer Programmiersprache entscheidbar, so sind die Programme der Programmiersprache aufzählbar. Die Menge der totalen berechenbaren Funktionen ist jedoch bekanntlich nicht aufzählbar.

Wie sieht nun die Semantik eines funktionalen Programms aus? Betrachten wir dazu das folgende Beispielprogramm.

$$\begin{array}{lcl} \texttt{mult(x,y)} & = & \text{if y} = 0 \text{ then 0 else add(x, mult(x, y } \dot{-} 1)) \\ \text{add(x,y)} & = & \text{if y} = 0 \text{ then x else add(x, y } \dot{-} 1) + 1 \end{array}$$

Es ist intuitiv ersichtlich, daß dieses die Multiplikations- und die Additionsfunktion spezifiziert.
Formale Semantiken zeichnen meist eine Gleichung aus, üblicherweise die erste, und bezeichnen die dort spezifizierte mathematische Abbildung als Semantik des Programms. Diese Verwendung einer einzigen Abbildung als Semantik des gesamten Programms vernachläßigt jedoch vollkommen die innere, kompositionelle Struktur desselbigen. Wir betrachten daher ein Programm nicht als die Spezifikation einer Abbildung, sonderen eines Datentyps. Ein **Datentyp** besteht aus einer Menge von Daten, dem Träger, im Beispiel die natürlichen Zahlen $\mathbb{N}$, und einer Menge von Operationen, mathematischen Abbildungen über dem Träger. Ein Datentyp ist also mathematisch gesehen eine Algebra.

$$\text{Datentyp} = \text{Algebra} = \langle \text{Träger, Operationen} \rangle$$

Ein Teil des Datentyps ist schon durch die Semantik der Programmiersprache und nicht erst durch das konkrete Programm gegeben. In unserem Beispiel haben wir vorausgesetzt, daß die Menge der natürlichen Zahlen der Träger ist, und Symbole wie `+ 1`, $\dot{-}$`1` und `if-then-else` Inkrementierung, Dekrementierung und bedingte Verzweigung bezeichnen. Nur die Bedeutungen der Symbole `mult` und `add` ergeben sich erst durch das Programm. Die Semantik der Programmiersprache definiert also einen **Basisdatentyp**:

$$\text{Basisdatentyp} = \langle \text{Träger, Basisoperationen} \rangle$$

Die Semantik eines konkreten Programms ist dann eine Erweiterung dieses Basisdatentyps um zusätzliche Operationen:

$$\text{Datentyp eines Programms} = \langle \text{Träger, Basisoperationen} \cup \text{definierte Operationen} \rangle$$

Die Bedeutung der Basisdatentypen wird in der Theorie der Programmschemata [Ind94] besonders deutlich, die Schemata von Programmen unabhängig von konkreten Basisdatentypen — dort Interpretationen genannt — betrachtet.

Funktionale Programmiersprachen besitzen üblicherweise einige Standardbasisdatentypen, beispielsweise ganze Zahlen (Integer), Fließkommazahlen, Zeichen (Character), logische Werte (Boolean) und Listen. Um auch Bäume und Matrizen direkt repräsentieren zu können, ermöglichen moderne funktionale Programmiersprachen wie Miranda und Haskell dem Programmierer, neue, eigene Basisdatentypen zu definieren: die sogenannten algebraischen Datentypen. Ein algebraischer Datentyp der natürlichen Zahlen läßt sich folgendermaßen in Miranda definieren:

```
nat ::= Zero | Succ(nat)
```

`Zero` und `Succ` heißen **Konstruktoren**, und Konstruktorterme (`Zero`, `Succ(Zero)`, ...) bilden den Träger des algebraischen Datentyps. Der Mechanismus des **Patternmatchings** ermöglicht nun die einfache und intuitive Definition von Operationen über dem algebraischen Datentyp.

```
add(x,Zero)      =  x
add(x,Succ(y))   =  Succ(add(x,y))

mult(x,Zero)     =  Zero
mult(x,Succ(y))  =  add(x,mult(x,y))
```

Die Konstruktoren repräsentieren die Basisoperationen des algebraischen Datentyps, und das Patternmatching verbindet zusätzlich eine Testoperation mit einer Komponentenselektion.
Algebraische Datentypen stellen ein äußerst mächtiges Konzept dar, da sie unter anderem die Definition von Aufzählungstypen, markierten Vereinigungen von Typen und kartesischen Produkten von Typen ermöglichen. Wie schon das obige Beispiel verdeutlicht, sind selbst die Standardbasisdatentypen als algebraische Datentypen definierbar (auf Probleme mit den ganzen und Fließkommazahlen gehen wir später noch ein). Dies rechtfertigt, im weiteren funktionale Programmiersprachen zu betrachten, die nur algebraische Datentypen besitzen. Die Standardbasisdatentypen lassen sich schlicht als effizientere Realisierungen entsprechender algebraischer Datentypen auffassen. In dieser Arbeit untersuchen wir, wie sich für derartige konstruktorbasierte funktionale Programmiersprachen sinnvoll Semantiken definieren lassen, ähnlich wie dies in [Vui74b] für Funktionsschemata getan wurde.

Funktionale Programmiersprachen zeichnen sich durch die Möglichkeit aus, daß Funktionen höherer Ordnung, sogenannte Funktionale, spezifizierbar sind. Wird der λ-Kalkül oder werden Kombinatorsysteme als Grundlage funktionaler Programme verwendet, so ergeben sich Funktionale praktisch automatisch. Sie sind jedoch kein notwendiger Bestandteil einer funktionalen Programmiersprache. Auch bei einem Verzicht auf Funktionale besitzt man noch eine vollwertige (bezüglich der natürlichen Zahlen mit Sicherheit universelle) Programmiersprache. Es existieren auch reale funktionale Programmiersprachen wie beispielsweise Sisal, die nur die Spezifikation von Funktionen erster Ordnung ermöglichen. Auch rekursive Funktionsschemata sind im allgemeinen erster Ordnung ([Ind94]). Die hier zu untersuchende semantische Beschreibung auf Konstruktoren basierender Basisdatentypen und des Patternmatchings ist unabhängig von dem Vorhandensein von Funktionalen. Semantiken funktionaler Programmiersprachen höherer Ordnung sind mathematisch aufwendiger, da aufgrund der Funktionale ein spezifizierter Datentyp keine einfache Algebra mehr sein kann. Daher beschränken wir uns auf konstruktorbasierte funktionale Programme **erster Ordnung**.

Die hier vorgestellten Definitionen und Sätze lassen sich wegen der einfacheren Semantik zwar nicht direkt für funktionale Programmiersprachen höherer Ordnung übernehmen, aber sowohl die Vorgehensweise als auch die meisten Ergebnisse sind übertragbar.

Es gibt viele verschiedene Methoden, formale Semantiken zu definieren. Für funktionale Programmiersprachen werden am häufigsten denotationelle und operationelle Semantiken eingesetzt.
Eine **operationelle Semantik** betrachtet die Ausführung eines Programms und ist gewissermaßen eine Berechnungsvorschrift. Sie beschreibt die Auswertung eines Eingabeausdrucks durch Übergänge zwischen teilweise ausgewerteten Ausdrücken. Letztendlich erhält man als operationelle Semantik eines funktionalen Programms eine Eingabe-Ausgabe-Abbildung mit beispielsweise

$$\texttt{mult}(\texttt{add}(\texttt{0},\texttt{0}),\texttt{0}) \mapsto \texttt{0}$$

bzw.

$$\texttt{mult}(\texttt{add}(\texttt{Zero},\texttt{Zero}),\texttt{Zero}) \mapsto \texttt{Zero}$$

Eine **denotationelle Semantik** übersetzt dagegen ein ganzes Programm in ein mathematisches Objekt. Jedem syntaktischen Teilobjekt eines Programms wird ein mathematisches, semantisches Objekt zugeordnet. Wir sagen, daß das syntaktische Objekt das semantische Objekt bezeichnet oder **denotiert**. Auf diese Weise können wir auch eine Algebra, den spezifizierten Datentyp, als Semantik eines funktionalen Programms erhalten.
Aus dem Datentyp ergibt sich auch unmittelbar die Eingabe-Ausgabe-Abbildung des Programms. Der umgekehrte Schluß ist nur bedingt möglich. Der entscheidende Unterschied sind jedoch die völlig verschiedenen Sichtweisen: Bezeichnen wir Programme als ausführbare Spezifikationen von Datentypen, so befassen sich operationelle Semantiken mit der Ausführbarkeit und denotationelle mit dem abstrakten Datentyp.
Auch in ihren Anwendungen ergänzen sich die beiden Semantiken gegenseitig. Eine operationelle Semantik dient als Grundlage einer konkreten Implementierung. Eine denotationelle Semantik wird dagegen für Beweise von Programm- und Spracheigenschaften eingesetzt. Auch wir werden denotationelle Semantiken derart verwenden.
Unser Ziel besteht somit darin, Semantiken sowohl operationell als auch denotationell zu definieren. Wir müssen dann jeweils beweisen, daß die operationelle und die denotationelle Semantik übereinstimmen.

In Kapitel 2 sind die im weiteren benötigten Grundlagen kurz zusammengestellt, insbesondere der Kalkül der Termersetzungssysteme, der der Ausgangspunkt für unsere operationellen Semantiken sein wird.
In Kapitel 3 definieren wir dann die Syntax zweier einfacher, konstruktorbasierter funktionaler Programmiersprachen erster Ordnung, die wir als Grundlage unserer semantischen Untersuchungen verwenden werden: Die erste Programmart setzt Patternmatching zur Funktionsspezifikation ein, während die zweite stärker an rekursiven Funktionsschemata orientiert ist, und explizite Test- und Dekompositionsfunktionen für Konstruktoren verwendet. Diese Programme sind schlicht spezielle Arten von Termersetzungssystemen. Damit läßt sich deren Theorie, insbesondere die Reduktion, auf die Programme übertragen. Die naive Verwendung der Reduktion zur Definition einer operationellen Semantik für unsere Programme führt jedoch zu unerwünschten Resultaten. Daraufhin fordern wir eine wichtige Eigenschaft, die alle Semantiken funktionaler Programmiersprachen erfüllen sollen: die Invarianz.

Um derartige Semantiken zu finden, betrachten wir in Kapitel 4 die zwei in funktionalen Programmiersprachen allgemein verwendeten Semantiken: die call-by-value und die call-by-name Semantik. Wir definieren jeweils sowohl eine operationelle Semantik, die auf der Reduktion der Termersetzungssysteme beruht, als auch eine denotationelle Semantik. Bei letzterern handelt es sich um auf ω-vollständigen Halbordnungen und dem Fixpunktsatz von Tarski beruhende Fixpunktsemantiken. Für die call-by-value Semantik erweist sich auch die Übereinstimmung der operationellen und der denotationellen Semantik als relativ einfach beweisbar.
Diese Betrachtungen führen uns in Kapitel 5 zu dem Konzept der erzwungenen Striktheit und den allgemeineren ς-Semantiken. Die call-by-value und die call-by-name Semantik sind nur zwei, allerdings ausgezeichnete Vertreter dieser ς-Semantiken. Wir definieren für jede ς-Semantik eine (denotationelle) Fixpunktsemantik und zwei (operationelle) Reduktionssemantiken, wobei die eine Reduktionssemantik mehr von theoretischem Interesse ist, während die zweite implementationsnäher ist. Anschließend beweisen wir die Übereinstimmung aller drei Semantiken.
In Kapitel 6 stellen wir Beziehungen zu anderen Semantikarten her. Im Kalkül der Termersetzungssysteme und der algebraischen Spezifkationen ist das initiale Modell (Quotientenmodell) als semantische Grundlage sehr beliebt. Wir begründen, warum dieses initiale Modell für die Semantik unserer funktionalen Programmiersprachen ungeeignet ist. Wir zeigen jedoch auch einen Zusammenhang zwischen dem initialen Modell und den ς-Semantiken auf. Deklarative Semantiken beruhen auf dem Erfüllbarkeits- und Modellbegriff der Logik. Wir stellen fest, daß von unseren ς-Semantiken allein die call-by-name Semantik deklarativ definierbar ist. Durch die Verwendung partieller Algebren läßt sich aber auch die call-by-value Semantik deklarativ formulieren.
Wir haben zwei Programmarten definiert, eine, die Patternmatching verwendet, und eine, die zusätzliche Hilfsfunktionen einführt. Es stellt sich somit die Frage, ob diese beiden gleich mächtig sind und sich Programme der einen Art in Programme der anderen Art semantikerhaltend effektiv übersetzen lassen. In Kapitel 7 stellen wir fest, daß Patternmatching echt mächtiger als die Hilfsfunktionen ist. Die das Scheidekriterium darstellende Eigenschaft der Sequentialität gibt uns weitere Einblicke in die Semantik der Programmiersprachen und auch Anregungen für effizientere Reduktionssemantiken.
Mit den ς-Semantiken sind nur sogenannte freie Datentypen direkt darstellbar. In Kapitel 8 beschäftigen wir uns mit Semantiken für nicht-freie Datentypen und zeigen einige dabei auftretende Probleme auf. Daraufhin stellen wir eine Programmiermethode vor, die eine adäquate Darstellung nicht-freier Datentypen auch bei Verwendung der ς-Semantiken gestattet.
Schließlich fassen wir in Kapitel 9 die erzielten Erkenntnisse kurz zusammen. Außerdem weisen wir noch auf interessante weiterführende Fragestellungen und Ideen hin.

An dieser Stelle möchte ich all jenen danken, die mich durch Vorschläge, Kommentare, Hinweise auf Literatur und auch Ratschläge für das Schreiben mit LaTeX unterstützt haben. Insbesondere danke ich diesbezüglich meinem Betreuer Thomas Noll.

Kapitel 2

Grundlagen

Hier werden in kompakter Form die grundlegenden Begriffe und deren Eigenschaften dargestellt, die wir in den weiteren Kapiteln brauchen werden. Leider ist die in der Literatur verwendete Notation sehr uneinheitlich, und häufig werden auch verschiedene Dinge gleich bezeichnet. Es wurde jedoch versucht, sich an wenigen Standardwerken zu orientieren, und nur für neue Konzepte, die hauptsächlich natürlich erst in den folgenden Kapiteln eingeführt werden, eine eigene Notation einzuführen. Auf diese Werke sowie auf ausführlichere Einführungen in die Gebiete und auch Beweise der angegebenen Eigenschaften wird jeweils hingewiesen.
Der Leser kann diese Kapitel also kurz überfliegen und im weiteren — mit Hilfe des Indexes — zum Nachschlagen verwenden. Es sei allerdings darauf hingewiesen, daß einige Begriffe der Termersetzungssysteme und insbesondere der beinahe orthogonalen hier in einer ungewöhnlichen und etwas allgemeineren Form als üblich definiert werden, was sich bei ihrer späteren Verwendung als nützlich erweisen wird.
Viele Abhängigkeiten definierter Begriffe von anderen Begriffen werden durch entsprechende Indizes ausgedrückt. So ist beispielsweise $\xrightarrow[R]{}$ die Reduktionsrelation eines Termersetzungssystemes R. Im weiteren Gebrauch werden diese Indizes jedoch oft weggelassen ($\longrightarrow$). Sie ergeben sich dann aus dem Zusammenhang.

2.1 Allgemeines

Die Grundlagen der „naiven“ Mengentheorie, wie sie von Georg Cantor geprägt wurden, werden als bekannt vorausgesetzt. Jedoch ist dieser Mengenbegriff problematisch, da seine konsequente Durchführung zu Widersprüchen führt. In dieser Arbeit erscheinen einige „Kollektionen von Objekten“, deren Bezeichnung als Mengen genau zu diesen Widersprüchen führen würde. Diese werden darum **Klassen** genannt. Jede Menge ist auch eine Klasse, jedoch nicht umgekehrt. Auf Klassen können nicht mehr alle Operationen der „naiven“ Mengenlehre angewendet werden.
Konkret betrifft dies hier die in 2.3 eingeführten Algebren. Vereinfacht gesagt besteht eine Algebra aus einer beliebigen nicht-leeren Menge A und einer bestimmten Abbildung α. Die Kollektion aller Σ-Algebren Alg_Σ ist eine Klasse aber keine Menge, denn wäre sie eine Menge, ergäbe sich folgende Variante von Russels Paradoxon:
Definiere $R := \{\mathfrak{A} = \langle A, \alpha\rangle \in \mathrm{Alg}_\Sigma \mid \mathfrak{A} \notin A\}$. Ist nun eine Algebra $\langle R, \alpha_R\rangle \in R$? Wenn ja, so folgt aus der Eigenschaft aller Element der Menge R, daß $\langle R, \alpha_R\rangle \notin R$. Wenn nein, so muß nach Definition $\langle R, \alpha_R\rangle \in R$ sein. Beide Möglichkeiten führen also zu Widersprüchen.
Hier soll jedoch nicht weiter darauf eingegangen werden. Eine Einführung in die axiomatische

Mengentheorie befindet sich im Anhang 1 von [We92], eine ausführliche Darstellung in [Shoen77].

Im weiteren bezeichnet $\mathbb{N}$ die natürlichen Zahlen, $\mathbb{N}_+$ die positiven natürlichen Zahlen, $[n]$ die Zahlenmenge $\{i \mid 1 \leq i \leq n\}$, $\mathbb{B}$ die Menge der Wahrheitswerte $\mathtt{tt}$ und $\mathtt{ff}$, $\mathcal{P}(M)$ die Potenzmenge einer Menge M und M^* die Menge aller (endlichen) Wörter über dem Alphabet (der Menge) M. Seien A und B Mengen. Ist φ eine (totale) Abbildung von A nach B, so schreiben wir $\varphi : A \to B$; ist φ eine partielle Abbildung von A nach B, so schreiben wir $\varphi : A \rightharpoonup B$. Jede totale Abbildung ist natürlich auch eine partielle Abbildung. Wir ziehen hier übrigens den Begriff „Abbildung“ dem Begriff „Funktion“ vor, da letzterer im Bereich der Programmiersprachen ein äußerst vieldeutig verwendeter Begriff ist. Ist $\varphi : A \to B$ oder $\varphi : A \rightharpoonup B$ und $T \subseteq A$, so verwenden wir $\varphi(T) := \{\varphi(a) \mid a \in T\}$.
Eine Menge M heißt genau dann **abzählbar**, wenn eine surjektive Abbildung $\varphi : \mathbb{N} \to M$ existiert, sie also höchstens soviele Elemente wie $\mathbb{N}$ besitzt. Eine Menge M heißt genau dann **aufzählbar**, wenn eine berechenbare surjektive Abbildung $\varphi : \mathbb{N} \to M$ existiert (§2.2 in [Hermes78]).

2.2 Halbordnungen

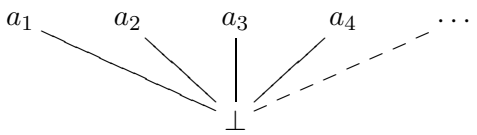
Im folgenden werden Halbordnungen und die zugehörigen Begriffe eingeführt (vgl. [Ind90], [We92]).

Eine **Halbordnung** $\mathfrak{A} = \langle A, \leq_{\mathfrak{A}} \rangle$ besteht aus einer nicht-leeren Menge A und einer Relation $\leq_{\mathfrak{A}} \subseteq A \times A$ mit den 3 Eigenschaften:

- Reflexivität: $a \leq_{\mathfrak{A}} a$
- Antisymmetrie: $a \leq_{\mathfrak{A}} b \wedge b \leq_{\mathfrak{A}} a \implies a = b$
- Transitivität: $a \leq_{\mathfrak{A}} b \wedge b \leq_{\mathfrak{A}} c \implies a \leq_{\mathfrak{A}} c$

Ist $T \subseteq A$, $a \in A$, so schreiben wir $T \leq a :\Leftrightarrow \forall t \in A.\ t \leq a$ und $a \leq T :\Leftrightarrow \forall t \in A.\ a \leq t$. Wir schreiben $a < b :\Leftrightarrow a \leq b \wedge a \neq b$, $a \geq b :\Leftrightarrow b \leq a$ und $a > b :\Leftrightarrow b < a$. Außerdem verwenden wir noch die Symole $\preceq$, $\sqsubseteq$, $\trianglelefteq$ und deren Abwandlungen für Halbordnungsrelationen. Auch die Teilmengenrelation $\subseteq$ ist eine Halbordnungsrelation.

Eine Halbordnung läßt sich durch ein **Hasse-Diagramm** veranschaulichen, wie im folgenden für die **flache Halbordnung** $\langle A_\perp, \leq \rangle$ ($A_\perp = \{\perp, a_1, a_2, \ldots\}$, $a \leq b \Leftrightarrow a = \perp$ oder $a = b$) gezeigt:



Ist $\mathfrak{B} = \langle B, \leq_{\mathfrak{B}} \rangle$ eine Halbordnung und A eine Menge, so ist die kanonische Halbordnung des Abbildungsraums (Funktionsraums), $\mathfrak{C} := \langle (A \to B), \preceq_{\mathfrak{C}} \rangle$, gegeben durch die punktweise Ordnungsrelation

$$\varphi \preceq_{\mathfrak{C}} \psi \ :\Longleftrightarrow\ \forall a \in A.\ \varphi(a) \leq_{\mathfrak{B}} \psi(a).$$

Sind A und B Mengen, so ist die kanonische Halbordnung der partiellen Abbildungen, $\mathfrak{C} := \langle (A \rightharpoonup B), \preceq_{\mathfrak{C}} \rangle$, gegeben durch

$$\varphi \preceq_{\mathfrak{C}} \psi \ :\Longleftrightarrow\ \forall a \in A.\ \varphi(a) \text{ undefiniert } \vee\ \varphi(a) = \psi(a).$$

Ist $\mathfrak{A} = \langle A, \leq \rangle$ eine Halbordnung und $T \subseteq A$ eine beliebige Teilmenge, so heißt $a \in A$ genau dann

- **obere Schranke** von T, wenn $T \leq a$;
- **kleinstes Element** von T, geschrieben $\mathrm{Min}_{\mathfrak{A}}(T)$, wenn $a \in T$ und $a \leq T$;
- **kleinste obere Schranke** von T, geschrieben $\bigsqcup_{\mathfrak{A}} T$, wenn es kleinstes Element der Menge $\{b \in A \mid T \leq b\}$ aller oberen Schranken von T ist.

Sind $\mathfrak{A} = \langle A, \leq_{\mathfrak{A}} \rangle, \mathfrak{B} = \langle B, \leq_{\mathfrak{B}} \rangle$ Halbordnungen, so heißt die Abbildung $\varphi : A \to B$ genau dann **monoton**, wenn

$$a \leq_{\mathfrak{A}} a' \implies \varphi(a) \leq_{\mathfrak{B}} \varphi(a')$$

für alle $a, a' \in A$.

Ist $\mathfrak{A} = \langle A, \leq \rangle$ eine Halbordnung, dann heißt eine Teilmenge $T \subseteq A$ genau dann **gerichtet**, wenn

$$\forall a, b \in T.\ \exists c \in T.\ a \leq c \wedge b \leq c$$

Ein Spezialfall der gerichteten Teilmengen sind die total geordneten **Ketten** $K \subseteq A$ mit:

$$\forall a, b \in K.\ a \leq b \ \vee \ b \leq a$$

Nicht-leere Ketten K mit abzählbar vielen Elementen werden **ω-Ketten** genannt und lassen sich auch als Folge schreiben:

$$K = (a_i)_{i \in \mathbb{N}} \text{ mit } a_0 \leq a_1 \leq a_2 \leq \ldots$$

Monotone Abbildungen erhalten die Gerichtetheit bzw. die Ketteneigenschaft, d. h. ist T gerichtet (Kette) und φ monoton, so ist $\varphi(T)$ gerichtet (Kette).

Eine Halbordnung $\mathfrak{A} = \langle A, \leq \rangle$ heißt genau dann **ω-vollständig** , wenn jede abzählbare gerichtete Menge $T \subseteq A$ in A eine kleinste obere Schranke hat, d. h. $\bigsqcup T$ existiert und $\bigsqcup T \in A$. Schon äquivalent dazu ist die Eigenschaft, daß jede abzählbare Kette $K \subseteq A$ eine kleinste obere Schranke in A besitzt. Diese Äquivalenz wird in [Mar76] bewiesen und beruht auf folgender Idee: Jede Kette ist eine gerichtete Menge, und umgekehrt läßt sich zu jeder abzählbaren gerichteten Menge $T = \{a_0, a_1, a_2, \ldots\}$ eine abzählbare Kette $K = \{a_0, \bigsqcup\{a_0, a_1\}, \bigsqcup\{\bigsqcup\{a_0, a_1\}, a_2\}, \ldots$ mit $\bigsqcup T = \bigsqcup K$ konstruiert.
Da insbesondere $T = \emptyset$ bzw. $K = \emptyset$ sein kann[1], besitzt jede ω-vollständige Halbordnung $\mathfrak{A}$ ein **kleinstes Element** $\bot_{\mathfrak{A}} := \bigsqcup \emptyset \in A$. Wenn $\mathfrak{A}$ sich aus dem Zusammenhang ergibt, wird oft auch nur $\bot$ geschrieben.

Flache Halbordnungen sind immer ω-vollständig.
Ist M eine Menge, so ist $\langle \mathcal{P}(M), \subseteq \rangle$ eine ω-vollständige Halbordnung mit $\bigcup T$ als kleinster oberer Schranke einer beliebigen(!) Teilmenge $T \in \mathcal{P}(M)$.
Ist $\mathfrak{A} = \langle A, \leq \rangle$ eine ω-vollständige Halbordnung und B eine Menge, so ist die kanonische Halbordnung des Abbildungsraumes, $\langle (B \to A), \preceq \rangle$, ω-vollständig, wobei für beliebige abzählbare gerichtete Mengen $T \subseteq (B \to A)$ die kleinste obere Schranke $\bigsqcup T$ durch

$$(\bigsqcup T)(b) := \bigsqcup\{\varphi(b) \mid \varphi \in T\}$$

[1] In der Literatur werden gerichtete Mengen und Ketten oft als nicht-leer definiert. In diesem Fall wird für die ω-Vollständigkeit zusätzlich die Existenz eines kleinsten Elementes gefordert.

für alle $b \in B$ gegeben ist.

Ist $\langle A, \leq \rangle$ eine Halbordnung, so heißt eine Menge $T \subseteq A$ genau dann **kofinal** in eine Menge $T' \subseteq A$, wenn $\forall t \in T.\ \exists t' \in T'.\ t \leq t'$. Ist T kofinal in T' und T' kofinal in T, so heißen T und T' **gegenseitig kofinal**.

Lemma 2.1 Kofinalität und kleinste obere Schranken
Sei $\langle A, \leq \rangle$ eine ω-vollständige Halbordnung und $M, M', T, T' \subseteq A$, wobei T und T' gerichtet und abzählbar seien. Es gilt:

1. $M \subseteq M' \implies M$ kofinal in M'.
2. M kofinal in $T \implies M \leq \bigsqcup T$.
3. T' kofinal in $T \implies \bigsqcup T' \leq \bigsqcup T$.
4. M und T gegenseitig kofinal $\implies \bigsqcup M$ existiert und $\bigsqcup M = \bigsqcup T$.
5. T und T' gegenseitig kofinal $\implies \bigsqcup T = \bigsqcup T'$.

Beweis:

1. Trivial.
2. Sei $a \in M$ beliebig. Da M kofinal in T, existiert $a' \in T$ mit $a \leq a'$. Es ist $a' \leq \bigsqcup T$ und somit $a \leq \bigsqcup T$.
3. Folgt aus Aussage 2, weil $\bigsqcup T'$ die kleinste obere Schranke von T' ist.
4. Nach Aussage 2 ist $M \leq \bigsqcup T$, $\bigsqcup T$ also eine obere Schranke von M. Sei c eine beliebige obere Schranke von M, d. h. $M \leq c$. Da T kofinal in M ist, ist $T \leq c$. Es folgt $\bigsqcup T \leq c$. Also ist $\bigsqcup T$ auch die kleinste obere Schranke von M.
5. Folgt direkt aus Aussage 4.

□

Seien $\mathfrak{A} = \langle A, \leq_{\mathfrak{A}} \rangle, \mathfrak{B} = \langle B, \leq_{\mathfrak{B}} \rangle$ ω-vollständige Halbordnungen. Eine Abbildung $\varphi : A \to B$ heißt genau dann ω-stetig, wenn für alle ω-Ketten $K \subseteq A$ die kleinste obere Schranke von $\varphi(K)$ existiert und $\bigsqcup \varphi(K) = \varphi(\bigsqcup K)$ ist.
Die Menge aller ω-stetigen Funktionen $\varphi : A \to B$ wird mit $[A \to B]$ bezeichnet. Wir hätten in der obigen Definition auch nicht-leere abzählbare Ketten oder nicht-leere abzählbare gerichtete Mengen betrachten können, ohne den definierten Begriff ω-Stetigkeit zu verändern.
Es ist jedoch zu beachten, daß diese Mengen nicht-leer sind, denn andernfalls müßte

$$\varphi(\bot) = \varphi(\bigsqcup \emptyset) = \bigsqcup \varphi(\emptyset) = \bigsqcup \emptyset = \bot$$

sein. Eine Abbildung mit dieser Eigenschaft heißt **strikt**. Für ω-Stetigkeit wollen wir Striktheit jedoch nicht fordern.
Jede ω-stetige Abbildung φ ist monoton, denn wenn $a \leq a'$, ist $\varphi(a') = \varphi(\bigsqcup\{a, a'\}) = \bigsqcup\{\varphi(a), \varphi(a')\}$ und somit $\varphi(a) \leq \varphi(a')$. Monotone Abbildungen über flachen Halbordnungen sind auch immer ω-stetig.

BEMERKUNG 2.1: ω-Stetigkeit und Monotonie
In der Literatur wird in Definitionen der ω-Stetigkeit die Monotonie meistens vorausgesetzt. Die Existenz von $\bigsqcup \varphi(K)$ ist damit direkt gegeben, und es muß nur noch die Gleichheit $\bigsqcup \varphi(K) = \varphi(\bigsqcup K)$ gefordert werden. Diese Definition ist äquivalent zu der gegebenen, erweckt aber den Eindruck, strenger zu sein. Beim Beweisen der ω-Stetigkeit einer Abbildung erweist sich unsere Definition auch als praktischer, da andernfalls die Monotonie und die Gleichheit $\bigsqcup \varphi(K) = \varphi(\bigsqcup K)$ in zwei getrennten, aber sehr ähnlichen Beweisen gezeigt werden müßten. □

Sind $\mathfrak{A} = \langle A, \leq_{\mathfrak{A}} \rangle, \mathfrak{B} = \langle B, \leq_{\mathfrak{B}} \rangle$ ω-vollständige Halbordnungen, so ist der Abbildungsraum der ω-stetigen Abbildungen, $[\mathfrak{A} \to \mathfrak{B}] := \langle [A \to B], \preceq \rangle$, mit der kanonischen punktweisen Ordnungsrelation $\preceq$ eine ω-vollständige Halbordnung.

Sein φ eine Abbildung von einer Menge A in sich selbst. Ein Element $a \in A$ mit $\varphi(a) = a$ heißt ein **Fixpunkt** von φ. Die denotationellen Semantiken, die wir später definieren werden, basieren alle auf dem folgenden zentralen Satz:

Satz 2.2 Fixpunktsatz von Tarski
Sei $\mathfrak{A} = \langle A, \leq \rangle$ ω-vollständig und $\varphi : A \to A$ ω-stetig. Dann besitzt φ einen **kleinsten Fixpunkt** in A, nämlich

$$\mathrm{Fix}(\varphi) := \bigsqcup \{\varphi^i(\bot) \mid i \in \mathbb{N}\}.$$

Beweis:
Anhang B.4 in [Fie&Har88], Abschnitt 1.5.2 in [We92]. □

Bisher wurden nur Abbildungen mit je einem Argument betrachtet. Die Begriffe Monotonie und ω-Stetigkeit lassen sich jedoch auch auf mehrstellige Abbildungen kanonisch fortsetzen, ohne daß diese als einstellige Abbildungen mit einem kartesischen Produktraum als Definitionsbereich aufgefaßt werden müssen.
Seien $\langle A_1, \leq_1 \rangle, \ldots, \langle A_n, \leq_n \rangle$ und $\langle A, \leq \rangle$ ω-vollständig. Die Funktion $\varphi : A_1 \times A_2 \times \ldots \times A_n \to A$ ist ω-stetig
gdw. $\forall i \in [n]$. $\forall \omega$-Ketten $T_i \subseteq A_i$

$$\varphi(\bigsqcup T_1, \ldots, \bigsqcup T_n) = \bigsqcup \varphi(T_1, \ldots, T_n)$$

gdw. $\forall a_1 \in A_1, \ldots, a_n \in A_n$. $\forall i \in [n]$. $\forall \omega$-Ketten $T_i \subseteq A_i$

$$\varphi(a_1, \ldots, a_{i-1}, \bigsqcup T_i, a_{i+1}, \ldots, a_n) = \bigsqcup \varphi(a_1, \ldots, a_{i-1}, T_i, a_{i+1}, \ldots, a_n).$$

Die Monotonie mehrstelliger Abbildungen ergibt sich analog.

Sei $\mathfrak{A} = \langle A, \leq \rangle$ eine ω-vollständige Halbordnung. Ein Element a aus A heißt genau dann **ω-kompakt**, wenn für jede ω-Kette $K \subseteq A$

$$a \leq \bigsqcup K \implies \exists a' \in K.\ a \leq a'$$

Für jedes Element a von A heißt ein ω-kompaktes Element a' von A mit $a' \leq a$ **endliche Approximation** von a und wir definieren

$$\begin{aligned} \mathrm{Fin}(a) &:= \{a' \in A \mid a' \leq a \text{ und } a' \text{ ist } \omega\text{-kompakt}\} \\ \mathrm{Fin}(T) &:= \bigcup \{\mathrm{Fin}(a') \mid a' \in T\} \qquad \text{für } T \subseteq A \end{aligned}$$

$\mathrm{Fin}(A)$ ist somit die Menge aller ω-kompakten Elemente von A.
$\mathfrak{A}$ heißt genau dann **ω-induktiv**, wenn für jedes Element a von A eine ω-Kette $K \subseteq A$ von ω-kompakten Elementen mit $a = \bigsqcup K$ existiert, d. h. jedes Element durch eine Kette endlicher (ω-kompakter) Elemente approximierbar ist.
Die Halbordnung der natürlichen Zahlen mit ∞, $\langle \mathbb{N} \dot\cup \{\infty\}, \leq \rangle$, ist ω-induktiv. Es ist $\mathrm{Fin}(\mathbb{N} \dot\cup \{\infty\}) = \mathbb{N}$, und es gilt $\bigsqcup\{0, 1, 2, \ldots\} = \infty$.
Für jedes Element a einer ω-induktiven Halbordnung gilt $a = \bigsqcup \mathrm{Fin}(a)$.
Für ω-induktive Halbordnungen gilt auch die Umkehrung des Lemmas 2.1 über Kofinalität und kleinste obere Schranken: Sind $T, T' \subseteq A$ abzählbare Ketten und besteht T nur aus ω-kompakten Elementen, so impliziert $\bigsqcup T \leq \bigsqcup T'$, daß T kofinal in T' ist.
Alle im weiteren betrachteten ω-vollständigen Halbordnungen sind ω-induktiv, auch wenn dies meistens nicht explizit erwähnt wird.

Bemerkung 2.2: ω-Vollständigkeit und ω-Stetigkeit
In der Literatur über Halbordnungen (insbes. [Mar76], [Cou80]) werden neben den hier definierten abzählbaren gerichteten Mengen, abzählbaren Ketten und ω-Ketten noch viele weitere Systeme von Teilmengen des Trägers A einer Halbordnung $\langle A, \leq \rangle$ betrachtet, z.B.:

$\bigsqcup$-Mengen	=	nicht-leere Mengen
$\bigsqcup_\alpha$-Mengen	=	$\bigsqcup$-Mengen einer Kardinalität kleiner-gleich α
Δ-Mengen	=	gerichtete Mengen
Δ_α-Mengen	=	Δ-Mengen einer Kardinalität kleiner-gleich α
Φ-Mengen	=	nicht-leere Ketten
Φ_α-Mengen	=	Φ-Mengen einer Kardinalität kleiner-gleich α
$\dot\bigsqcup$-Mengen	=	$\bigsqcup$-Mengen, die jeweils eine obere Schranke in A besitzen
PC-Mengen	=	$\bigsqcup$-Mengen, in denen jedes Elementenpaar eine obere Schranke in A besitzt.

Es wird dann Z-Vollständigkeit und Z-Stetigkeit ($Z = \bigsqcup, \bigsqcup_\alpha, \Delta, \ldots$) für diese Mengen anstelle der von uns verwendeten (nicht-leeren) abzählbaren gerichteten Mengen definiert. In dem schon erwähnten [Mar76] wird in Theorem 1 & Corollary 1 bewiesen, daß $\bigsqcup_\alpha$-Vollständigkeit und Φ_α-Vollständigkeit für eine feste Kardinalität α äquivalent ist.
Die von uns definierte ω-Vollständigkeit und ω-Stetigkeit ist die Schwächste all dieser Variationen. In der Literatur über Semantiken funktionaler Programmiersprachen werden häufig Δ-Vollständigkeit und Δ-Stetigkeit verwendet, da diese in der Mathematik am geläufigsten sind. Benötigt, und deshalb auch hier verwendet, werden jedoch nur ω-Vollständigkeit und ω-Stetigkeit. Der zentrale Fixpunktsatz von Tarski setzt nur diese voraus. Außerdem kann man vage formulieren, daß bei der Definition von Semantiken von Programmiersprachen nur aus Berechnungen hervorgehende Mengen betrachtet werden müssen, die prinzipiell abzählbar (genaugenommen nur aufzählbar) sind.
Alle im weiteren betrachteten ω-vollständigen Halbordnungen sind jedoch auch Δ-vollständig (und die ω-stetigen Abbildungen auch Δ-stetig), da alle diese Halbordnungen nur abzählbare Ketten besitzen, und die Begriffe somit zusammenfallen. □

2.3 Algebren

Algebren und die dazugehörigen elementaren Begriffe der universellen Algebra werden in fast jeder Abhandlung über rekursive Funktionsschemata oder Termersetzungssysteme definiert. Für eine ausführlichere Behandlung siehe [Ind90] oder [We92].

In [ADJ77] werden zum ersten Mal Algebren und Halbordnungen zu geordneten, ω-vollständigen Algebren zusammengefügt. Diese sind inzwischen als Basis vieler denotationeller Semantiken allgemein akzeptiert ([Cou90]). Eine ausführliche Darstellung, insbesondere der algebraischen Grundlagen, mit vielen Beispielen findet sich in [We92].

2.3.1 Algebren

Eine **Signatur** ist eine Menge Σ, deren Elemente **Operationssymbole** genannt werden, zusammen mit einer Abbildung $\varrho : \Sigma \to \mathbb{N}$, der **Stelligkeitsfunktion**, die jedem Operationssymbol $f \in \Sigma$ seine **Stelligkeit** $\varrho(f)$ zuordnet. Im folgenden werden wir die Stelligkeitsfunktion ϱ nicht mehr explizit erwähnen. Stattdessen schreiben wir $f^{(n)}$ für ein Operationssymbol f der Stelligkeit n und $\Sigma_n := \{f^{(n)} \in \Sigma\}$ für die Menge aller n-stelligen Operationssymbole. Es ist $\Sigma = \bigcup_{n\in\mathbb{N}} \Sigma_n$. Die 0-stelligen Operationssymbole werden auch **Konstanten** genannt.

Ist A eine Menge und $n \in \mathbb{N}$, so heißt eine totale Abbildung $f : A^n \to A$ n-stellige **Operation** auf A. Wir verwenden die Bezeichnungen:

$$\begin{aligned} \mathrm{Ops}_n(A) &:= \{f \mid f : A^n \to A\} \\ \mathrm{Ops}(A) &:= \bigcup_{n\in\mathbb{N}} \mathrm{Ops}_n(A) \end{aligned}$$

Eine **(Σ-)Algebra** $\mathfrak{A} = \langle A, \alpha\rangle$ besteht aus einer nicht-leeren Menge A, dem **Träger** oder **Universum**, und einer **Zuweisung** $\alpha : \Sigma \to \mathrm{Ops}(A)$, die jedem n-stelligen Operationssymbol $f \in \Sigma_n$ eine n-stellige Operation $\alpha(f) \in \mathrm{Ops}_n(A)$ zuweist. Statt $\alpha(f)$ schreiben wir auch oft $f^{\mathfrak{A}}$. Alg_Σ bezeichnet die **Klasse aller Σ-Algebren**.
Wenn nicht anders angegeben, bezeichnen wir im folgenden den Träger einer Algebra $\mathfrak{A}$ immer mit A, den von $\mathfrak{B}$ mit B, den von $\mathfrak{A}'$ mit A' usw. Allerdings werden die Großbuchstaben $A, B, \ldots$ auch noch für diverse andere Zwecke verwendet. Die jeweilige Bedeutung ist dann aus dem Zusammenhang ersichtlich.

Sei A eine Menge und $\sim$ eine Äquivalenzrelation über A. Die Menge aller Elemente, die modulo $\sim$ äquivalent zu einem $a \in A$ ist,

$$[a]_\sim := \{a' \in A \mid a' \sim a\},$$

heißt **Äquivalenzklasse[2] modulo $\sim$** des **Repräsentanten** a. Die Menge aller Äquivalenzklassen von A modulo $\sim$,

$$A/\sim := \{[a]_\sim \mid a \in A\},$$

heißt **Quotientenmenge von A modulo $\sim$**.
Sei $\mathfrak{A}$ eine Σ-Algebra und $\sim$ eine Kongruenz über $\mathfrak{A}$. Die Σ-Algebra

$$\mathfrak{A}/\sim := \langle A/\sim, \alpha/\sim\rangle,$$

mit

$$f^{\mathfrak{A}/\sim}([a_1]_\sim, \ldots, [a_n]_\sim) := [f^{\mathfrak{A}}(a_1, \ldots, a_n)]_\sim$$

für alle $a_1, \ldots, a_n \in A$, $f^{(n)} \in \Sigma$ heißt **Quotientenalgebra** (Faktoralgebra) von $\mathfrak{A}$ modulo $\sim$.

[2] Der Begriff „Klasse“ ist hier aus traditionelle Gründen leider unvermeidbar. Natürlich sind Äquivalenzklassen Mengen.

Seien $\mathfrak{A}, \mathfrak{B} \in \mathrm{Alg}_\Sigma$ Algebren. Eine Abbildung $h : A \to B$ heißt genau dann **Homomorphismus** von $\mathfrak{A}$ nach $\mathfrak{B}$, geschrieben $h : \mathfrak{A} \to \mathfrak{B}$, wenn h mit den Operationen **kompatibel** ist, d. h.

$$h(f^{\mathfrak{A}}(a_1, \ldots, a_n)) = f^{\mathfrak{B}}(h(a_1), \ldots, h(a_n))$$

für alle $a_1, \ldots, a_n \in A$, $f^{(n)} \in \Sigma$.
Ein bijektiver Homomorphismus h von $\mathfrak{A}$ nach $\mathfrak{B}$ heißt **Isomorphismus**. $\mathfrak{A}$ und $\mathfrak{B}$ heißen dann **isomorph**. Natürlich ist in diesem Fall auch h^{-1} ein Isomorphismus.
Sei K eine Klasse von Σ-Algebren und $X \subseteq A$ für eine Algebra $\mathfrak{A} \in K$. $\mathfrak{A}$ heißt genau dann **frei in K relativ zu X**, wenn sich jede Abbildung φ von X zu einer Algebra $\mathfrak{B}$ aus K in eindeutiger Weise zu einem Homomorphismus $\bar{\varphi} : \mathfrak{A} \to \mathfrak{B}$ fortsetzen läßt, also $\bar{\varphi}|_X = \varphi$ ist. Ist $X = \emptyset$, so heißt $\mathfrak{A}$ **initial in K**. In diesem Fall existiert zu jeder Algebra $\mathfrak{B} \in K$ genau ein Homomorphismus $h_{\mathfrak{B}} : \mathfrak{A} \to \mathfrak{B}$.
Alle relativ zu einer festen Menge X freien und alle initialen Algebren einer Klasse K sind jeweils isomorph. Daher unterscheiden wir im weiteren oft nicht mehr zwischen ihnen und sprechen nur noch von der initialen Algebra einer Klasse B.
Wir sprechen von **absolut freien Σ-Algebren** und **absolut intialen Σ-Algebren** (bzw. der absolut ...), wenn $K = \mathrm{Alg}_\Sigma$.

2.3.2 Geordnete Algebren

Sei $\langle A, \leq \rangle$ eine Halbordnung und $\langle A, \alpha \rangle$ eine Σ-Algebra mit demselben Träger, so daß alle Operationen $\alpha(f)$, $f \in \Sigma$, monoton bzgl. der Halbordnung sind. Dann heißt $\mathfrak{A} := \langle A, \leq, \alpha \rangle$ **geordnete (Σ-)Algebra**. Eine geordnete Algebra kann sowohl alleine als Algebra als auch als reine Halbordnung betrachtet werden, wobei jeweils $\leq$ oder α ignoriert werden.
Die **Menge aller geordneten Σ-Algebren zu einer festen Halbordnung** $\langle A, \leq \rangle$ wird mit $\mathrm{Alg}_\Sigma\langle A, \leq \rangle$ bezeichnet. Für diese Menge ist die **kanonische Halbordnung geordneter Σ-Algebren**, $\langle \mathrm{Alg}_\Sigma\langle A, \leq \rangle, \sqsubseteq \rangle$, definiert durch[3]

$$\mathfrak{A} \sqsubseteq \mathfrak{B} \;:\Longleftrightarrow\; \forall f \in \Sigma.\ f^{\mathfrak{A}} \preceq f^{\mathfrak{B}}$$

Für zwei geordnete Algebren $\mathfrak{A}$ und $\mathfrak{B}$ wird ein monotoner Homomorphismus von $\mathfrak{A}$ nach $\mathfrak{B}$ auch **Morphismus** genannt.

Eine geordnete Σ-Algebra $\mathfrak{A} = \langle A, \leq, \alpha \rangle$ heißt genau dann **ω-vollständige Σ-Algebra**[4], wenn $\langle A, \leq \rangle$ ω-vollständig und alle Operationen $f^{\mathfrak{A}}$, $f^{(n)} \in \Sigma$, ω-stetig bezüglich der Halbordnung sind, d. h. $f^{\mathfrak{A}} \in [A^n \to A]$. Die **Klasse aller ω-vollständigen Σ-Algebren** wird durch $\mathrm{Alg}^\infty_{\Sigma,\perp}$ bezeichnet. Die im nächsten Abschnitt definierte Algebra der unendlichen, partiellen Terme stellt ein Beispiel für ω-vollständige Algebren dar.
Seien $\mathfrak{A}, \mathfrak{B} \in \mathrm{Alg}^\infty_{\Sigma,\perp}$. Ein strikter, ω-stetiger Morphismus von A nach B wird **ω-Morphismus** genannt. Man beachte, daß aufgrund der geforderten Striktheit $\perp_{\mathfrak{A}}$ auf $\perp_{\mathfrak{B}}$ abgebildet wird. Bei der Betrachtung von ω-vollständigen Algebren beziehen sich im weiteren die Begriffe **Isomorphie**, **Freiheit** und **Initialität** auf die ω-Morphismen und nicht auf die allgemeineren Homomorphismen.

[3] $\preceq$ ist die kanonische Ordnungrelation der Operationen (über A) zur Ordnung $\langle A, \leq \rangle$.
[4] In der Literatur, zum Beispiel [ADJ77], werden ω-vollständige Σ-Algebren auch ω-stetig genannt.

2.4 Terme

Terme (erster Ordnung) bilden die Grundlage für Termersetzungssysteme und funktionale Programmiersprachen (erster Ordnung). Daher werden sie und die zugehörigen Standardoperationen in fast allen entsprechenden Literaturwerken definiert (z. B. [Der&Jou90], [Huet&Lévy79], [We92]). Wir haben uns hier an der Notation von [Hof&Kut89] orientiert.
Unendliche Terme, auch Bäume genannt, werden in [Gue81] und [We92] für die Semantik von Programmen und Programmiersprachen eingesetzt. Eine ausführliche Darstellung ihrer Eigenschaften ist [Cou83].

2.4.1 Terme

Eine **Variablenmenge** ist eine abzählbare Menge, die disjunkt zu allen verwendeten Signaturen ist. Im folgenden sei Σ eine Signatur mit mindestens einer Konstanten und X eine Variablenmenge.

Die Menge der **(Σ-)Terme über X**, $\mathrm{T}_\Sigma(X)$, ist induktiv definiert als die kleinste Menge mit

- $X \in \mathrm{T}_\Sigma(X)$ und
- $f^{(n)} \in \Sigma, t_1, \ldots, t_n \in \mathrm{T}_\Sigma(X) \implies f(t_1, \ldots, t_n) \in \mathrm{T}_\Sigma(X)$.

Bei Konstanten $a \in \Sigma_0$ schreiben wir anstelle von $a()$ meistens a. $t \in \mathrm{T}_\Sigma(X)$ heißt **Σ-Term über X** oder auch schlicht **Term**. Ein variablenfreier Terme $t \in \mathrm{T}_\Sigma(\emptyset)$ heißt **(Σ-)Grundterm**. $\mathrm{T}_\Sigma := \mathrm{T}_\Sigma(\emptyset)$ ist die **Menge der (Σ-)Grundterme**.

Die **Σ-Termalgebra** $\mathcal{T}_\Sigma(X) := \langle \mathrm{T}_\Sigma(X), \tau \rangle$ ist gegeben durch die Zuordnung τ:

$$\tau(f)(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$$

($f^{(n)} \in \Sigma$, $t_1, \ldots, t_n \in \mathrm{T}_\Sigma(X)$). $\mathcal{T}_\Sigma := \mathcal{T}_\Sigma(\emptyset)$ heißt **Σ-Grundtermalgebra**.

$\mathcal{T}_\Sigma(X)$ ist absolut frei relativ zu X und $\mathcal{T}_\Sigma$ ist absolut initial. Wir unterscheiden nicht zwischen isomorphen Σ-Algebren und fassen daher im weiteren $\mathcal{T}_\Sigma(X)$ als die relativ zu X absolut freie und $\mathcal{T}_\Sigma$ als die absolut initiale Σ-Algebra auf. Auch wenn wir meistens die oben definierte Präfixnotation mit Klammern für Terme verwenden, sind wir nicht an diese gebunden. Wir können beispielsweise auch graphische Darstellungen, die Infixnotation oder die im weiteren definierte Darstellung als Abbildungen von Stellen verwenden. Wir besitzen somit eine **abstrakte Syntax** von Termen, die allein auf algebraischen Eigenschaften beruht (vgl. mit der Einführung von [ADJ77]).

Die endliche **Menge der in einem Term $t \in \mathrm{T}_\Sigma(X)$ vorkommenden Variablen**, $\mathrm{Var}(t)$, ist induktiv definiert durch

- $\mathrm{Var}(x) := \{x\}$, $x \in X$,
- $\mathrm{Var}(f(t_1, \ldots, t_n)) := \mathrm{Var}(t_1) \cup \ldots \cup \mathrm{Var}(t_n)$.

Für Grundterme $t \in \mathrm{T}_\Sigma$ ist somit $\mathrm{Var}(t) = \emptyset$, und es gilt natürlich für alle Terme $t \in \mathrm{T}_\Sigma(\mathrm{Var}(t))$.

Gegeben seien noch die endliche Variablenmenge $Y = \{y_1, \ldots, y_n\}$ und die Variablenmenge Z. Eine Abbildung $\sigma : Y \to \mathrm{T}_\Sigma(Z)$ heißt **Substitution**. Eine Substitution σ mit $\sigma(y_i) = t_i$ für alle $i \in [n]$ wird dargestellt durch

$$\sigma = [t_1/y_1, \ldots, t_n/y_n].$$

Ist $\sigma(y) = t$ für alle $y \in Y$, so schreiben wir auch

$$\sigma = [t/Y].$$

Eine Substitution $\sigma : Y \to \mathrm{T}_\Sigma$ heißt **Grundsubstitution**.
Ist $Y, Z \subseteq X$, so läßt sich eine Substitution $\sigma : Y \to \mathrm{T}_\Sigma(Z)$ kanonisch fortsetzen zu $\bar{\sigma} : X \to \mathrm{T}_\Sigma(X)$ mit

$$\bar{\sigma}(x) := \begin{cases} \sigma(x) & \text{, falls } y \in Y \\ x & \text{, andernfalls} \end{cases}$$

und weiter zu $\hat{\sigma} : \mathrm{T}_\Sigma(X) \to \mathrm{T}_\Sigma(X)$ mit

$$\begin{array}{lcll} \hat{\sigma}(x) & := & \bar{\sigma}(x) & , x \in X \\ \hat{\sigma}(f(t_1, \ldots, t_n)) & := & f(\hat{\sigma}(t_1), \ldots, \hat{\sigma}(t_n)). & \end{array}$$

Wir schreiben immer nur σ für $\bar{\sigma}$ und $\hat{\sigma}$. Außerdem schreiben wir die Applikation einer Substitution grundsätzlich in Präfixschreibweise; d. h. $y\sigma$ anstelle von $\sigma(y)$. Oft schreiben wir auch $\sigma : X \to \mathrm{T}_\Sigma(Z)$, obwohl die Substitution σ nur auf einer endlichen Teilmenge von X ungleich der Identität ist.
Für alle Terme $t \in \mathrm{T}_\Sigma(X)$ und Grundsubstitutionen $\sigma : Y \to \mathrm{T}_\Sigma$ mit $\mathrm{Var}(t) \subseteq Y$ gilt $t\sigma \in \mathrm{T}_\Sigma$. Für alle Terme $t \in \mathrm{T}_\Sigma(X)$ und Substitutionen $\sigma : Y \to \mathrm{T}_\Sigma(Z)$ $(Y, Z \subseteq X)$ ist $t\sigma = t\sigma|_{\mathrm{Var}(t)} = t\sigma|_{\mathrm{Var}(t) \cap Y}$. $t\sigma$ heißt **Instanz** des Terms t. Ist $t\sigma \in \mathrm{T}_\Sigma$, so heißt es **Grundinstanz** von t.
Zwei Terme $t, t' \in \mathrm{T}_\Sigma(X)$ heißen genau dann **unifizierbar**, wenn zwei Substitutionen $\sigma, \sigma' : X \to \mathrm{T}_\Sigma(X)$ mit $t\sigma = t'\sigma'$ existieren.[5]

Eine Liste von positiven natürlichen Zahlen heißt **Stelle**. $\mathbb{N}_+^*$ ist somit die **Menge aller Stellen**. Die **Menge der in einem Term $t \in \mathrm{T}_\Sigma(X)$ vorkommenden Stellen** (Occurences), $\mathrm{Occ}(t) \subseteq \mathbb{N}_+^*$, ist die kleinste Menge mit

- $\varepsilon \in \mathrm{Occ}(t)$ und
- $t = f(t_1, \ldots, t_n), u \in \mathrm{Occ}(t_i), i \in [n] \implies i.u \in \mathrm{Occ}(t)$.

Der **Teilterm von $t \in \mathrm{T}_\Sigma(X)$ an der Stelle $u \in \mathrm{Occ}(t)$**, t/u, ist gegeben durch

$$\begin{array}{lcll} t/\varepsilon & := & t & \\ f(t_1, \ldots, t_n)/i.u & := & t_i/u & , i \in [n] \end{array}$$

Der durch das **Einsetzen von $t' \in \mathrm{T}_\Sigma(X)$ an der Stelle $u \in \mathbb{N}_+^*$ in $t \in \mathrm{T}_\Sigma(X)$** entstehende Term $t[u \leftarrow t']$ ist definiert durch

$$\begin{array}{lcll} t[\varepsilon \leftarrow t'] & := & t' & \\ f(t_1, \ldots, t_n)[i.u \leftarrow t'] & := & f(t_1, \ldots, t_i[u \leftarrow t'], \ldots, t_n) & , \ i \in [n] \\ t[u \leftarrow t'] & := & t & \text{, andernfalls, d. h. } u \notin \mathrm{Occ}(t). \end{array}$$

Wir definieren zwei Halbordnungen für Stellen: Die **Präfixordnung** $\langle \mathbb{N}_+^*, \leq \rangle$ ist gegeben durch

$$u \leq v \ :\Longleftrightarrow \ \exists w \in \mathbb{N}_+^*.\ u.w = v$$

[5] Anstelle der üblichen einen Substitution verwenden wir zwei: σ und σ'. t und t' können gemeinsame Variablen besitzen ($\mathrm{Var}(t) \cap \mathrm{Var}(t') \neq \emptyset$), aber in unseren Anwendungen der Unifikation sind die Variablen in beiden Termen voneinander bedeutungsunabhängig.

und die **lexikalische Ordnung** $\langle \mathbb{N}_+^*, \leq_{\text{lex}} \rangle$ durch

$$u \leq_{\text{lex}} v \iff u \leq v \vee \exists w, u', v' \in \mathbb{N}_+^*.\ \exists i, j \in \mathbb{N}_+.\ i < j \wedge u = w.i.u' \wedge v = w.j.v'$$

Anschaulich bedeutet in einer Baumdarstellung von Termen $u \leq v$, daß sich u oberhalb (außerhalb) von v befindet, und $u \leq_{\text{lex}} v$, daß u oberhalb oder links von v liegt.
Die Präfixordnung bestimmt die **Abhängigkeit von Stellen**. u und v heißen genau dann **voneinander abhängig**, wenn $u \leq v$ oder $v \leq u$. Entsprechend heißen u und v genau dann **voneinander unabhängig**, geschrieben $u \parallel v$, wenn sie nicht voneinander abhängig sind. Sind $U, V \subseteq \mathbb{N}_+^*$ Stellenmengen, so schreiben wir

- $u \parallel V \ :\iff \forall v \in V.\ u \parallel v$
- $U \parallel V \ :\iff \forall u \in U.\ u \parallel V$

2.4.2 Unendliche Terme

Eine partielle Abbildung von Stellen nach Operationssymbolen und Variablen, $t : \mathbb{N}_+^* \rightharpoonup \Sigma \dot\cup X$, heißt genau dann **unendlicher (Σ-)Term über X** (Σ-Baum über X), wenn ihr Definitionsbereich $\text{Occ}(t)$ die folgenden Eigenschaften hat:

- $\varepsilon \in \text{Occ}(t)$
- $\text{Occ}(t)$ ist präfix-abgeschlossen, d. h.

$$\forall u, v \in \mathbb{N}_+^*.\ u.v \in \text{Occ}(t) \implies u \in \text{Occ}(t)$$

- Die Stelligkeit der Signatursymbole wird beachtet, d. h.

$$\begin{aligned} \forall u \in \text{Occ}(t).\quad & (t(u) \in \Sigma_n \Rightarrow \forall i \in \mathbb{N}_+.\ (u.i \in \text{Occ}(t) \Leftrightarrow i \in [n])) \wedge \\ & (t(u) \in X \Rightarrow \forall i \in \mathbb{N}_+.\ u.i \notin \text{Occ}(t)) \end{aligned}$$

Die **Menge aller unendlichen (Σ-)Terme über X** wird mit $\text{T}_\Sigma^\infty(X)$ bezeichnet. $\text{T}_\Sigma^\infty := \text{T}_\Sigma^\infty(\emptyset)$ ist die **Menge aller unendlichen (Σ-)Grundterme**.
Die **Σ-Algebra unendlicher Terme** $\mathcal{T}_\Sigma^\infty(X) := \langle \text{T}_\Sigma^\infty(X), \tau^\infty \rangle$ ist gegeben durch die Zuordnung τ^∞:

$$\begin{aligned} \text{Occ}(\tau^\infty(f)(t_1, \ldots, t_n)) &:= \{\varepsilon\} \dot\cup \{i.u_i \mid i \in [n], u_i \in \text{Occ}(t_i)\} \\ (\tau^\infty(f)(t_1, \ldots, t_n))(u) &:= \begin{cases} f & \text{, falls } u = \epsilon \\ t_i(u') & \text{, falls } u = i.u',\ i \in [n] \end{cases} \end{aligned}$$

$\mathcal{T}_\Sigma^\infty := \mathcal{T}_\Sigma^\infty(\emptyset)$ heißt **Σ-Algebra der unendlichen Grundterme**.

Sei nun $\perp \notin \Sigma \dot\cup X$ eine neue, ausgezeichnete Konstante. $\text{T}_{\Sigma,\perp}^\infty(X) := T_{\Sigma \dot\cup \{\perp\}}^\infty(X)$ heißt **Menge der partiellen, unendlichen (Σ-)Terme über X**.

Für partielle, unendliche Σ-Terme über X ist die **kanonische Halbordnung** $\langle \text{T}_{\Sigma,\perp}^\infty(X), \trianglelefteq \rangle$ definiert durch

$$t \trianglelefteq t' \ :\iff \begin{array}{l} \text{Occ}(t) \subseteq \text{Occ}(t') \wedge \\ \forall u \in \text{Occ}(t).\ t(u) \neq \perp \ \Rightarrow\ t(u) = t'(u) \end{array}$$

$\langle \text{T}_{\Sigma,\perp}^\infty(X), \trianglelefteq \rangle$ ist die kleinste Halbordnung auf $\text{T}_{\Sigma,\perp}^\infty(X)$,

- für die $\perp$ das kleinste Element ist, d. h.

$$\forall t \in \mathrm{T}^{\infty}_{\Sigma,\perp}(X).\ \perp \trianglelefteq t$$

- und die **invariant** ist, d. h.

$$t' \trianglelefteq t'' \ \Rightarrow\ t[u \leftarrow t'] \trianglelefteq t[u \leftarrow t'']$$

Die kanonische Halbordnung ist ω-vollständig, wobei die kleinste obere Schranke einer abzählbaren gerichteten Menge $T \subseteq \mathrm{T}^{\infty}_{\Sigma,\perp}(X)$ gegeben ist durch

$$\begin{aligned} \mathrm{Occ}(\bigsqcup T) &:= \bigcup \mathrm{Occ}(T) \\ (\bigsqcup T)(u) &:= \begin{cases} f & \text{, falls } t'(u) = f \in \Sigma \cup X \text{ für ein } t' \in T \\ \perp & \text{, falls } t'(u) = \perp \text{ für alle } t' \in T \text{ mit } u \in \mathrm{Occ}(t') \end{cases} \end{aligned}$$

Somit ist $\mathcal{T}^{\infty}_{\Sigma,\perp}(X) := \langle \mathrm{T}^{\infty}_{\Sigma,\perp}(X), \trianglelefteq, \tau^{\infty} \rangle$ eine ω-vollständige Σ-Algebra.

$\mathcal{T}^{\infty}_{\Sigma,\perp}(X)$ ist sogar frei relativ zu[6] X in $\mathrm{Alg}^{\infty}_{\Sigma,\perp}$ und $\mathcal{T}^{\infty}_{\Sigma,\perp} := \mathcal{T}^{\infty}_{\Sigma,\perp}(\emptyset)$ initial in $\mathrm{Alg}^{\infty}_{\Sigma,\perp}$. Auch hier wollen wir nicht zwischen isomorphen Algebren unterscheiden und fassen daher $\mathcal{T}^{\infty}_{\Sigma,\perp}(X)$ als die relativ zu X freie und $\mathcal{T}^{\infty}_{\Sigma,\perp}$ als die initiale Algebra in $\mathrm{Alg}^{\infty}_{\Sigma,\perp}$ auf.
Es gibt eine direkte Korrespondenz zwischen den Termen über X, $\mathrm{T}_{\Sigma}(X)$, und den unendlichen Termen über X mit endlichem Definitionsbereich $\mathrm{Occ}(t)$, $\mathrm{T}'_{\Sigma}(X) \subseteq \mathrm{T}^{\infty}_{\Sigma}(X)$. Da die zugehörigen Algebren $\mathcal{T}_{\Sigma}(X)$ und $\langle \mathrm{T}'_{\Sigma}(X), \tau^{\infty} \rangle$ isomorph sind, wird auch nicht zwischen ihnen unterschieden, und Terme über X können somit auch als Abbildungen mit den in ihnen vorkommenden Stellen als Definitionsbereich aufgefaßt werden. Alle definierten Mengen von Termen sind somit letztendlich Teilmengen von $\mathrm{T}^{\infty}_{\Sigma,\perp}(X)$.
Eine Übersicht einiger Termbezeichnungen:

T_{Σ}	-	total, endlich
$\mathrm{T}_{\Sigma,\perp}$	-	partiell, endlich
$\mathrm{T}^{\infty}_{\Sigma}$	-	total, unendlich
$\mathrm{T}^{\infty}_{\Sigma,\perp}$	-	partiell, unendlich
$\mathrm{T}_{\Sigma,\perp} \setminus \mathrm{T}_{\Sigma}$	-	echt partiell, endlich
$\mathrm{T}^{\infty}_{\Sigma} \setminus \mathrm{T}_{\Sigma}$	-	total, echt unendlich

Gleiches gilt natürlich auch für Terme über X. Die obigen Bezeichungen entfallen jedoch häufig, genauso wie wir auch Grundterme oft einfach Terme nennen.

Die Menge der in einem unendlichen Term $t \in \mathrm{T}^{\infty}_{\Sigma,\perp}(X)$ vorkommenden Stellen, $\mathrm{Occ}(t)$, ist schon als dessen Definitionsbereich definiert.
Das **Symbol an einer Stelle $u \in \mathrm{Occ}(t)$ in einem Term $t \in \mathrm{T}^{\infty}_{\Sigma,\perp}(X)$**, $t(u)$, ist natürlich insbesondere auch für endliche Terme $t \in \mathrm{T}_{\Sigma}(X)$ gegeben.
Die **Menge der Stellen eines Terms $t \in \mathrm{T}^{\infty}_{\Sigma,\perp}(X)$ an der ein Symbol $g \in \Sigma \dot\cup X \dot\cup \{\perp\}$ steht** ist definiert durch

$$\mathrm{Occ}(g,t) := \{u \in \mathrm{Occ}(t) \mid t(u) = g\}$$

[6] Man beachte, daß X hier nicht die Variablenmenge bezeichnet, sondern die Menge der partiellen, unendlichen Terme, die nur aus einer Variablen bestehen, $\{(\varepsilon \mapsto x) \mid x \in X\}$. Dies erschwert natürlich den Vergleich mit der Algebra $\mathcal{T}_{\Sigma}(X)$, die absolut frei relativ zu der Variablenmenge X ist. Eine saubere, aber auch aufwendigere Definition der Freiheit einer Algebra findet sich in Abschnitt 3.2.2 von [We92].

Kommt jede Variable in einem Term $t \in \mathrm{T}^{\infty}_{\Sigma,\perp}(X)$ höchstens einmal vor, d. h. ist

$$|\mathrm{Occ}(x,t)| \leq 1$$

für alle $x \in X$, so heißt t **linear**.
Der **unendliche Teilterm von $t \in \mathbf{T}^{\infty}_{\Sigma,\perp}(X)$ an der Stelle $u \in \mathbf{Occ}(t)$**, t/u, ist gegeben durch

$$\begin{array}{lcl} \mathrm{Occ}(t/u) & := & \{v \in \mathbb{N}^*_+ \mid u.v \in \mathrm{Occ}(t)\} \\ (t/u)(v) & := & t(u.v) \end{array}$$

Der durch **Einsetzen von $t' \in \mathbf{T}^{\infty}_{\Sigma,\perp}(X)$ an der Stelle $u \in \mathbb{N}^*_+$ in $t \in \mathbf{T}^{\infty}_{\Sigma,\perp}(X)$** entstehende unendliche Term $t[u \leftarrow t']$ ist definiert durch

$$\begin{array}{lcl} \mathrm{Occ}(t[u \leftarrow t']) & := & \{u.v \mid v \in \mathrm{Occ}(t')\} \dot\cup \{w \in \mathrm{Occ}(t) \mid u \not\leq w\} \\ t[u \leftarrow t'](v) & := & \begin{cases} t'(w) & \text{, falls } v = u.w \text{ für ein } w \in \mathrm{Occ}(t') \\ t(v) & \text{, andernfalls} \end{cases} \end{array}$$

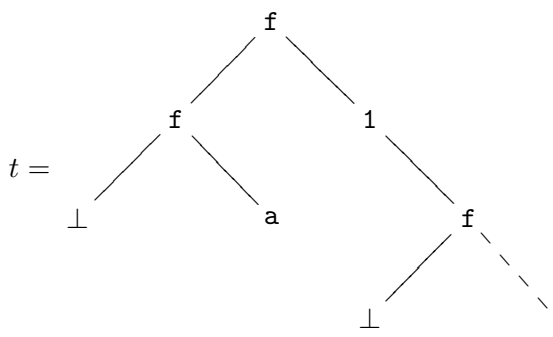
Für endliche Terme $t \in \mathrm{T}_\Sigma(X)$ stimmen die Definitionen mit den vorher gegebenen, einfacheren Definitionen überein. Man beachte, daß wir $\mathrm{Var}(t)$ und Substitutionen nicht für unendliche Terme definieren, da wir stets nur unendliche Grundterme verwenden werden.

Häufig verwenden wir endliche kartesische Produkte von Termen, $(t_1, \ldots, t_n) \in (\mathrm{T}^{\infty}_{\Sigma,\perp}(X))^n$. Wir schreiben diese meist als Vektor $\vec{t} := (t_1, \ldots, t_n)$, wobei wir immer das gleiche Metasymbol (hier t) für den Vektor $\vec{t}$ und die indizierten Elemente t_i verwenden.
Auch für diese definieren wir die Menge der in ihnen vorkommenden Stellen:

$$\mathrm{Occ}(\vec{t}) := \{i.u_i \in \mathbb{N}^*_+ \mid i \in [n], u_i \in \mathrm{Occ}(t_i)\}$$

Die Definitionen für $\vec{t}(u)$, $\mathrm{Occ}(f, \vec{t})$, $\vec{t}/u$ und $\vec{t}[u \leftarrow t']$ ergeben sich kanonisch. Es ist jedoch zu beachten, daß $\varepsilon \notin \mathrm{Occ}(\vec{t})$ ist, und daher auch immer zwischen $\mathrm{T}^{\infty}_{\Sigma,\perp}(X)$ und $(\mathrm{T}^{\infty}_{\Sigma,\perp}(X))^1$ unterschieden werden muß.

Während $\mathrm{T}_\Sigma(X)$ und auch $\mathrm{T}_{\Sigma,\perp}(X)$ abzählbare Mengen sind, besitzen $\mathrm{T}^{\infty}_\Sigma(X)$ und $\mathrm{T}^{\infty}_{\Sigma,\perp}(X)$ die Kardinalität der reellen Zahlen $\mathbb{R}$, wenn die Signatur Σ mindestens zwei 1-stellige oder ein 2-stelliges Operationssymbol besitzt.
Dies wollen wir durch Beispiele verdeutlichen.
$M := \{r \in \mathbb{R} \mid 0 \leq r < 1\}$ ist bekanntlich gleich mächtig wie $\mathbb{R}$, und wir wollen zeigen, daß auch M und $\mathrm{T}^{\infty}_{\Sigma,\perp}(X)$ gleichmächtig sind, wenn $\mathtt{a}^{(0)}, \mathtt{1}^{(1)}, \mathtt{0}^{(1)} \in \Sigma$.
Sei $r \in M$, beispielsweise $r = 0,1001001\ldots$ in Binärdarstellung. r wird dann durch den unendlichen Term `1(0(0(1(0(0(1(...)...)` dargestellt.
Sei nun $t \in \mathrm{T}^{\infty}_{\Sigma,\perp}(X)$, zum Beispiel



t läßt sich nun ebenenweise mit der Codierung $\bot \simeq 1$, $\mathtt{a} \simeq 2$, $\mathtt{f} \simeq 3$, $\mathtt{1} \simeq 4$ und Ebenenwechsel $\simeq 0$ als Zahl $0,3034012301\ldots$ eindeutig darstellen.
Da somit injektive Abbildungen von M nach $\mathrm{T}^{\infty}_{\Sigma,\bot}(X)$ und von $\mathrm{T}^{\infty}_{\Sigma,\bot}(X)$ nach M existieren, sind beide Mengen gleich mächtig.
Man beachte allerdings, daß eine Stelle immer eine endliche Liste positiver natürlicher Zahlen ist, und die Menge der in einem Term $t \in \mathrm{T}^{\infty}_{\Sigma,\bot}(X)$ vorkommenden Stellen, $\mathrm{Occ}(t)$, abzählbar ist.
Die ω-vollständige Halbordnung $\langle \mathrm{T}^{\infty}_{\Sigma,\bot}(X), \trianglelefteq \rangle$ ist ω-induktiv. Die ω-kompakten Terme sind genau die endlichen Terme, d. h. $\mathrm{Fin}(\mathrm{T}^{\infty}_{\Sigma,\bot}(X)) = \mathrm{T}_{\Sigma,\bot}(X)$. Somit existiert zu jedem unendlichem Term $t \in \mathrm{T}^{\infty}_{\Sigma,\bot}(X)$ eine ω-Kette endlicher Terme, $K \subseteq \mathrm{T}_{\Sigma,\bot}(X)$, durch die er approximiert werden kann, d. h. es gilt dann $t = \bigsqcup K$.

2.5 Termersetzungssysteme

Termersetzungssysteme bilden eine Grundlage für funktionale Programmiersprachen, algebraische Spezifikationen und automatisches Beweisen. Dementsprechend handelt es sich um ein bedeutendes, vieluntersuchtes Gebiet. Eine Einführung stellen Kapitel 6 und 7 von [Hof&Kut89] dar. [Huet&Op80], [Klop87], auch [Huet80] und insbesondere [Der&Jou90] sind zusammenfassende Darstellungen des Themas.

2.5.1 Termersetzungssysteme

Ein geordnetes Paar von Termen, welches wir $l \to r$ schreiben, mit **linker Seite** $l \in \mathrm{T}_{\Sigma}(X)$ und **rechter Seite** $r \in \mathrm{T}_{\Sigma}(\mathrm{Var}(t))$ heißt **Termersetzungsregel** oder kurz **Regel**. Eine endliche Menge R von Termersetzungsregeln ist ein **Termersetzungssystem**. Eine linke Regelseite heißt auch **Redexschema**, und die Menge aller Redexschemata eines Termersetzungssystems R wird mit RedS_R bezeichnet. Für $l \in \mathrm{RedS}_R$ und eine Grundsubstitution $\sigma : \mathrm{Var}(l) \to \mathrm{T}_{\Sigma}$ heißt $l\sigma$ **Redex** des Termersetzungssystems R. Red_R ist die Menge aller Redexe von R, und die **Menge aller Redexstellen eines Terms** $t \in \mathrm{T}_{\Sigma}$, $\mathrm{RedOcc}(t) \subseteq \mathrm{Occ}(t)$, ist gegeben durch[7]

$$\mathrm{RedOcc}(t) := \{u \in \mathrm{Occ}(t) \mid t/u \in \mathrm{Red}_R\}$$

Ein Grundterm $t \in \mathrm{T}_{\Sigma}$ heißt genau dann an einer Stelle $u \in \mathrm{Occ}(t)$ durch eine Regel $l \to r \in R$ in einem Schritt zu einem Grundterm $t' \in \mathrm{T}_{\Sigma}$ **reduzierbar**, wenn eine Grundsubstitution $\sigma : \mathrm{Var}(l) \to \mathrm{T}_{\Sigma}$ mit

- $t/u = l\sigma$ und
- $t' = t[u \leftarrow r\sigma]$

existiert.
Man beachte, daß sowohl σ als auch t' durch t, u und $l \to r$ eindeutig bestimmt sind. Das Tupel $A = \langle t, u, l \to r \rangle$ heißt dann **(elementare), (einfache) Reduktion** und wir schreiben $A = (t \xrightarrow[l \to r]{u} t')$ oder schlicht $A = (t \xrightarrow[R]{u} t')$.
Wir haben die Reduktion hier nur für Grundterme definiert, obwohl sie in der Literatur üblicherweise für beliebige Terme $\mathrm{T}_{\Sigma}(X)$ definiert wird. Für Berechnungen mit unseren funktionalen Programmen, die wir in Kapitel 3 definieren werden, benötigen wir jedoch nur die Reduktion auf

[7] In der Literatur wird häufig nicht zwischen den Redexstellen und den Redexen eines Terms unterschieden.

Grundtermen. Wir verwenden Terme mit Variablen nur in den Termersetzungsregeln. Die allgemeine Definition der Reduktion auch für Terme mit Variablen würde einige der weiteren Definitionen unnötig komplizieren.

Sei ϱ eine Menge von Reduktionen, den ϱ-Reduktionen. Wir schreiben eine ϱ-Reduktion $A = t \xrightarrow[l \to r]{u} t'$ auch als $A = t \xrightarrow[l \to r, \varrho]{u} t'$ oder $A = t \xrightarrow[R, \varrho]{u} t'$. Die **$\varrho$-Reduktionsrelation** $\xrightarrow[R,\varrho]{} \subseteq \mathrm{T}_\Sigma \times \mathrm{T}_\Sigma$ ist definiert durch

$$t \xrightarrow[R,\varrho]{} t' \iff \exists u \in \mathrm{RedOcc}_R(t).\ \exists (l \to r) \in R.\ t \xrightarrow[l \to r, \varrho]{u} t' \text{ ist eine } \varrho\text{-Reduktion.}$$

Man beachte, daß $\xrightarrow[R,\varrho]{}$ eine Relation ist, und $t \xrightarrow[R,\varrho]{} t'$ somit sowohl eine Aussage als auch eine Reduktion (ein Tupel) sein kann. In der Literatur wird diese Unterscheidung meistens völlig unterlassen.
Eine deterministische ϱ-Reduktionsrelation $\xrightarrow[R,\varrho]{}$, d. h. bei der für jedes $t \in \mathrm{T}_\Sigma$ höchstens ein $t' \in \mathrm{T}_\Sigma$ mit $t \xrightarrow[R,\varrho]{} t'$ existiert, heißt **sequentielle ϱ-Reduktionsstrategie**.
Genau dann, wenn $t_1 \xrightarrow[R,\varrho]{} t_2, t_2 \xrightarrow[R,\varrho]{} t_3, \ldots$ elementare ϱ-Reduktionen sind, heißt $A = t_1 \xrightarrow[R,\varrho]{} t_2 \xrightarrow[R,\varrho]{} t_3 \longrightarrow \ldots$ **ϱ-Reduktionsfolge**. Reduktionsfolgen können hier sowohl endlich als auch unendlich sein, während in der Literatur oft nur endliche Reduktionsfolgen betrachtet werden ([Huet&Lévy79], [Ber&Lévy77]).

Ist $A = (t = t_1 \xrightarrow[l_1 \to r_1]{u_1} t_2 \xrightarrow[l_2 \to r_2]{u_2} \ldots \xrightarrow[l_n \to r_n]{u_n} t_{n+1} = t')$ eine Reduktionsfolge und $U := \{u_1, \ldots, u_n\} \subseteq \mathrm{RedOcc}_R(t)$ eine Menge voneinander unabhängiger Redexstellen des Terms t, so heißt $A' := \langle t, \{\langle u_i, l_i \to r_i\rangle \mid i \in [n]\}\rangle$ **(elementare), (parallele) Reduktion** und wir schreiben $A' = t \xrightarrow[R]{U} t'$. Aus obigem folgt $t/u_i = l_i\sigma_i$ für Substitutionen σ_i, und daß

$$t' = t[u_i \leftarrow r_i\sigma_i \mid i \in [n]] = t[u_1 \leftarrow r_1\sigma_1] \ldots [u_n \leftarrow r_n\sigma_n]$$

Aufgrund der gegenseitigen Unabhängigkeit der Redexstellen ist die Reihenfolge der u_i bzw. der elementaren, einfachen Reduktionen irrelevant. Man beachte, daß auch $U = \emptyset$ und somit $t = t'$ sein kann.
Die **parallele ϱ-Reduktion** $A' = t \xrightarrow[R,\varrho]{U} t'$, die **parallele ϱ-Reduktionsrelation** $\xrightarrow[R,\varrho]{}$, die **parallele ϱ-Reduktionsstrategie** und die **parallelen ϱ-Reduktionsfolgen** ergeben sich analog zu oben.
In allen obigen Bezeichnungen entfällt ϱ, wenn ϱ die Menge aller einfachen Reduktionen bzw. aller parallelen Reduktionen ist.

Sei T eine Menge und $\xrightarrow[\varrho]{} \subseteq T \times T$ eine beliebige zweistellige Relation über T. Die **reflexive und transitive Hülle** von $\xrightarrow[\varrho]{}$ wird mit $\xrightarrow[\varrho]{*}$ bezeichnet, die **reflexive, transitive und symmetrische Hülle** mit $\xleftrightarrow[\varrho]{*}$.
Ein Element $t \in T$ heißt genau dann **ϱ-Normalform**, wenn kein $t' \in T$ mit $t \xrightarrow[\varrho]{} t'$ existiert. $t' \in T$ heißt genau dann **ϱ-Normalform von** $t \in T$, wenn $t \xrightarrow[\varrho]{*} t'$ und t' eine ϱ-Normalform ist. Ist die Normalform eindeutig, so schreiben wir $t\downarrow_\varrho := t'$.
$\xrightarrow[\varrho]{}$ heißt genau dann **stark konfluent**, wenn für alle $t, t', t'' \in T$ mit $t \xrightarrow[\varrho]{} t'$ und $t \xrightarrow[\varrho]{} t''$ ein $\hat{t} \in T$ existiert, so daß $t' \xrightarrow[\varrho]{} \hat{t}$ und $t'' \xrightarrow[\varrho]{} \hat{t}$ gilt.
$\xrightarrow[\varrho]{}$ heißt genau dann **konfluent**, wenn für alle $t, t', t'' \in T$ mit $t \xrightarrow[\varrho]{*} t'$ und $t \xrightarrow[\varrho]{*} t''$ ein $\hat{t} \in T$ existiert, so daß $t' \xrightarrow[\varrho]{*} \hat{t}$ und $t'' \xrightarrow[\varrho]{*} \hat{t}$ gilt.

$\xrightarrow[\varrho]{}$ heißt genau dann **terminierend**, wenn keine unendliche Folge $(t_i)_{i\in\mathbb{N}_+}$ mit $t_i \xrightarrow[\varrho]{} t_{i+1}$ für alle $i \in \mathbb{N}_+$ existiert ($t_1 \xrightarrow[\varrho]{} t_2 \xrightarrow[\varrho]{} \ldots$).
Starke Konfluenz impliziert Konfluenz, während die Umkehrung im allgemeinen nicht gilt. Wenn $\xrightarrow[\varrho]{}$ konfluent ist, besitzt jedes $t \in T$ höchstens eine ϱ-Normalform $t\downarrow_\varrho$. Wenn $\xrightarrow[\varrho]{}$ terminierend ist, besitzt jedes $t \in T$ mindestens eine ϱ-Normalform. Relationen $\xrightarrow[\varrho]{}$, die sowohl konfluent als auch terminierend sind, heißen **vollständig** oder **konvergent**. Für vollständige $\xrightarrow[\varrho]{}$ besitzt jedes $t \in T$ somit genau eine ϱ-Normalform $t\downarrow_\varrho$.

Die gerade definierten Begriffe und angegebenen Eigenschaften lassen sich nun insbesondere für die Reduktionsrelationen $\xrightarrow[R,\varrho]{}$ und $\xrightarrow[R,\varrho]{}\!\!\!\!\!\rightarrow$ verwenden.
Hierbei gebrauchen wir den Begriff ϱ-Normalform für $t\downarrow_{R,\varrho}$, und, wenn ϱ die Menge aller Reduktionen ist, bezeichnen wir $t\downarrow_R$ schlicht als Normalform. Ein Term $t \in \mathrm{T}_\Sigma$ ist genau dann eine Normalform, wenn $\mathrm{RedOcc}_R(t) = \emptyset$ ist. Ist $\xrightarrow[R]{}$ konfluent, so heißt eine ϱ-Reduktionsstrategie genau dann **normalisierend**, wenn für alle $t \in \mathrm{T}_\Sigma$ $t\downarrow_R$ genau dann existiert, wenn $t\downarrow_{R,\varrho}$ existiert, und im Falle der Existenz $t\downarrow_R = t\downarrow_{R,\varrho}$ ist.
Neben den Relationen $\xrightarrow[R,\varrho]{*}$ und $\xrightarrow[R,\varrho]{*}\!\!\!\!\!\rightarrow$ schreiben wir auch für endliche Reduktionsfolgen $A = t_1 \xrightarrow[R,\varrho]{} \ldots \xrightarrow[R,\varrho]{} t_n$ bzw. $A' = t'_1 \xrightarrow[R,\varrho]{}\!\!\!\!\!\rightarrow \ldots \xrightarrow[R,\varrho]{}\!\!\!\!\!\rightarrow t'_n$ mitunter $A = t_1 \xrightarrow[R,\varrho]{*} t_n$ bzw. $A' = t'_1 \xrightarrow[R,\varrho]{*}\!\!\!\!\!\rightarrow t'_n$.
Ist $\xrightarrow[R]{}$ terminierend/konfluent, so sagen wir auch, daß das Termersetzungssystem R terminierend/konfluent ist.
$\xrightarrow[R,\varrho]{}\!\!\!\!\!\rightarrow$ ist natürlich niemals terminierend. $\xrightarrow[R]{}$ ist im allgemeinen weder terminierend noch konfluent. Es ist unentscheidbar, ob R terminierend ist ([Huet&Lank78]), oder ob R konfluent ist ([Bau&Otto84]).

Im folgenden betrachten wir eine Teilmenge von Termersetzungssystemen, deren Reduktionsrelation immer konfluent und deren parallele Reduktionsrelation sogar stark konfluent ist.

2.5.2 Beinahe orthogonale Termersetzungssyteme

Termersetzungssysteme sind noch eine viel zu allgemeine Grundlage für unsere funktionalen Programmiersprachen. Insbesondere benötigen wir die Konfluenz der Reduktionsrelation, die jedoch im allgemeinen unentscheidbar ist, wie wir gerade festgestellt haben. Auch sind viele Eigenschaften des Reduktionsverhaltens schwer faßbar.
Die syntaktisch eingeschränkteren **beinahe orthogonalen Termersetzungssysteme** stellen daher eine wichtige Teilmenge der allgemeinen Termersetzungssysteme dar. Sie werden im Prinzip in [Ros73] zum ersten Mal betrachtet und in [O'Do77] ausführlich behandelt. Freilich werden in diesen Werken noch keine Termersetzungssysteme in unserem Sinne definiert, worauf wir gleich noch eingehen werden. Von entscheidender Bedeutung ist die nur für beinahe orthogonale Termersetzungssysteme definierbare und auch schon in [Ros73] eingeführte **Residuenabbildung**. Sie

ermöglicht eine einfache Untersuchung des Reduktionsverhaltens beinahe orthogonaler Termersetzungssysteme. Das **zentrale allgemeine Residuenlemma** (2.9) wird die Grundlage fast aller Beweise von Eigenschaften von Reduktionsfolgen sein. Eine triviale Konsequenz des allgemeinen Residuenlemmas ist außerdem die Konfluenz der Reduktionsrelation beinahe orthogonaler Termersetzungssysteme.

Definition 2.1 **Beinahe orthogonales Termersetzungssystem**
Ein Termersetzungssystem R heißt genau dann **beinahe orthogonal** (almost orthogonal), wenn es **linkslinear** ist, d. h. alle Redexschemata $l \in \mathrm{RedS}_R$ linear sind, und die folgende Eindeutigkeitsbedingung erfüllt:

Seien $l \to r, l' \to r' \in R$, $u \in \mathrm{Occ}(l)$ mit $l/u \notin X$, $\sigma, \sigma' : X \to \mathrm{T}_\Sigma(X)$. Dann gilt:

$$(l/u)\sigma = l'\sigma' \implies u = \varepsilon,\ r\sigma = r'\sigma'$$

□

Orthogonale Termersetzungssysteme erfüllen die noch strengere Eindeutigkeitbedingung

$$(l/u)\sigma = l'\sigma' \implies u = \varepsilon,\ (l \to r) = (l' \to r')$$

Die Beschränkung auf diese restriktivere Teilmenge ist jedoch nicht nötig. Fast die gesamte Theorie orthogonaler Termersetzungssysteme ist auf beinahe orthogonale Termersetzungssysteme übertragbar. Die Theorie der needed redexes in [Huet&Lévy79] stellt die einzige bedeutende Ausnahme dar, wie in [An&Mid94] festgestellt wird.
Orthogonale Termersetzungssysteme haben allerdings den Vorteil, daß bei einer Reduktion $t \xrightarrow[l \to r]{u} t'$ die verwendete Regel $l \to r$ durch t und u schon eindeutig bestimmt ist, so daß auf ihre Angabe immer verzichtet werden kann, wie wir dies schon bei der beim Matchen verwendeten Substitution σ machen. Allerdings ist bei beinahe orthogonalen Termersetzungssystemen immerhin noch die verwendete Regelinstanz einer Reduktion, $l\sigma \to r\sigma$, eindeutig durch t und u bestimmt. Dies ist auch für alle weiteren Zwecke ausreichend. In einigen Fällen werden wir allerdings die Unabhängigkeit einer Definition von der konkret verwendeten Regel explizit beweisen müssen.
Linkslineare Termersetzungssysteme, die nur die erweiterte Eindeutigkeitsbedingung

$$(l/u)\sigma = l'\sigma' \implies u \in \mathrm{Occ}(r),\ (r/u)\sigma = r'\sigma'$$

erfüllen, werden **schwach orthogonal** (weakly orthogonal) genannt. Sie sind weitaus schwieriger zu handhaben, da sich die Residuenabbildung und damit die gesamte Thorie beinahe orthogonaler Termersetzungssysteme nicht mehr auf sie übertragen läßt. Derartige teilweise Überlappungen linker Regelseiten sind jedoch in unseren später definierten funktionalen Programmiersprachen durch deren Syntax prinzipiell ausgeschlossen und deshalb auch nicht von Interesse.

Wie schon erwähnt, werden in [Ros73] und [O'Do77] noch keine Termersetzungssysteme in unserem Sinne betrachtet. Stattdessen werden potentiell unendliche Mengen von Grundtermersetzungsregeln, d. h. solchen deren linke und rechte Regelseiten nur aus Grundtermen bestehen, verwendet. Jedoch ist auch die Menge aller Grundinstanzen aller Regeln eines Termersetzungssystems, auch **kanonische Instanz des Termersetzungssystems** genannt, ein solches im allgemeinen unendliches Grundtermersetzungssystem. Die Reduktionen und Reduktionsrelationen dieses unendlichen Grundtermersetzungssystems sind identisch mit denen des Termersetzungssystems.

[Ros73] und [O'Do77] betrachten nun eine spezielle Teilmenge dieser unendlichen Grundtermersetzungssysteme, die sie Subtree-Replacement-Systeme nennen. Die Instanz eines jeden beinahe orthogonalen Termersetzungssystems ist ein solches Subtree-Replacement-System. Daher sind die Begriffe einfach übertragbar, und wir verweisen bei vielen Eigenschaften beinahe orthogonaler Termersetzungssysteme auf die entsprechenden Beweise in [O'Do77].
Da in der Informatik nur endlich beschreibbare Strukturen von Interesse sind, ist die Beschränkung auf beinahe orthogonale Termersetzungssysteme (anstelle beliebiger unendlicher Grundtermersetzungssysteme) sinnvoll; auch in Kapitel VII in [O'Do77] werden kurz endliche Schemata betrachtet, welche Subtree-Replacement-Systeme beschreiben, unter anderem orthogonale Termersetzungssysteme.
Im folgenden seien die Termersetzungssysteme immer beinahe orthogonal.

Es erweist sich im weiteren als nützlich, nicht nur die Menge aller Grundinstanzen der Regeln eines Termersetzungssystems zu betrachten, sondern auch Instanzen eines Termersetzungssystems zu verwenden, die nur bestimmte Grundinstanzen der Regeln des Termersetzungssystems enthalten. Auf jede Regel werden also nur bestimmte Grundsubstitutionen angewendet. Da eine rechte Regelseite nur Variablen enthält, die auf der linken Regelseite vorkommen, ist eine solche Instanz I eines Termersetzungssystems durch die Angabe der Redexmenge $\mathrm{Red}_{R,I} \subseteq \mathrm{Red}_R$ (Menge aller linken Regelseiten des unendlichen Grundtermersetzungssystems) eindeutig festgelegt (vgl. mit dem Instanziierungsprädikat Q der rule schemata in [O'Do77]). Für $\mathrm{Red}_{R,I} = \mathrm{Red}_R$ erhält man die vorher betrachtete kanonische Instanz eines Termersetzungssystems, welches wir zur Wahrung der üblichen Begriffe nicht von dem Termersetzungssystem unterscheiden wollen. Für $t \in \mathrm{T}_\Sigma$ definieren wir:

$$\mathrm{RedOcc}_{R,I}(t) := \{u \in \mathrm{RedOcc}_R(t) \mid t/u \in \mathrm{Red}_{R,I}\}$$

Eine Reduktion $A = t \xrightarrow[R]{u} t'$ heißt genau dann Reduktion in der Instanz I bzw. ***I*-Reduktion**, wenn $u \in \mathrm{RedOcc}_{R,I}(t)$. Die I-Reduktion ist also ein Spezialfall unserer ϱ-Reduktion, wird daher auch als $A = t \xrightarrow[R,I]{u} t'$ geschrieben, und auch die übrigen zugehörigen Begriffe ergeben sich entsprechend.
Alle solche Instanzen eines beinahe orthogonalen Termersetzungssystemes sind Subtree-Replacement-Systeme.

Die in [Ros73] eingeführte **Residuenabbildung** wird schon in [Huet&Lévy79] auf orthogonale Termersetzungssysteme übertragen. Viele interessante Eigenschaften von Reduktionsfolgen beruhen darauf, wie Redexe in den elementaren Reduktionen entstehen, erhalten bleiben und zerstört werden. Eine elementare Reduktion vermittels einer Termersetzungsregel stellt im Grunde eine Umordnung von Teiltermen und damit insbesondere von Redexen dar. Die Residuenabbildung beschreibt nun diesen Umordnungsprozeß, indem für eine Reduktion $t \longrightarrow t'$ jeder Redexstelle u des Terms t die Menge der Redexstellen von t' zugeordnet werden, die Kopien von u sind, die Residuen von u.

Lemma 2.3 Residuen
Sei $t \xrightarrow[l \to r]{u} t'$ eine elementare Reduktion und $v \in \mathrm{RedOcc}(t)$. Dann ist die Menge der **Residuen** von v bezüglich dieser Reduktion wie folgt definiert:

$$v \setminus (t \xrightarrow[l \to r]{u} t') := \begin{cases} \emptyset & \text{, wenn } v = u \\ \{v\} & \text{, wenn } v \parallel u \text{ oder } v < u \\ \{u.w'.v' \mid v = u.w.v', r/w' = l/w \in X\} & \text{, wenn } v > u \end{cases}$$

□

Obwohl in der Definition die bei der Reduktion verwendete Regel $l \to r$ explizit genannt wird, sind die Residuen nur von t und u abhängig. Um dies zu beweisen, brauchen wir allerdings erst die folgende Eigenschaft von Residuen:

Lemma 2.4 Redexe eines Terms und deren Stellen
Sei $t \in T_\Sigma$ und $u, v \in \mathrm{RedOcc}_R(t)$ mit $u < v$. Sei $l \in \mathrm{RedS}_R$ mit $t/u = l\sigma$ für eine Substitution σ. Dann existiert genau ein $w \in \mathbb{N}_+^*$ mit $u \leq u.w \leq v$ und $l/w \in X$.

Beweis:
Da $u < v$, ist $v = u.u'$ mit $\varepsilon \neq u' \in \mathbb{N}_+^*$.
Angenommen, für alle $w \in \mathbb{N}_+^*$ mit $w \leq u'$ ist $l/w \notin X$. Dann ist insbesondere

$$u' \in \mathrm{Occ}_R(l) \setminus \mathrm{Occ}_R(X, l) \tag{1}$$

Da $v \in \mathrm{RedOcc}(t)$, ist t/v ein Redex, d. h. es existiert $\hat{l} \in \mathrm{RedS}_R$ und eine Substitution $\hat{\sigma}$ mit

$$t/v = \hat{l}\hat{\sigma} \tag{2}$$

Somit folgt

$$\hat{l}\hat{\sigma} \overset{(2)}{=} t/v = (t/u)/u' = l\sigma/u' \overset{(1)}{=} (l/u')\sigma$$

Da $u' \neq \varepsilon$, steht dies im Widerspruch zur Eindeutigkeitsbedingung beinahe orthogonaler Termersetzungssysteme.
Also existiert ein $w \in \mathbb{N}_+^*$ mit $u \leq u.w \leq v$ und $l/w \in X$, und trivialerweise ist dieses eindeutig bestimmt. □

Lemma 2.5 Unabhängigkeit der Residuen von der Regel der Reduktion
Seien $A = t \xrightarrow[l \to r]{u} t'$ und $\hat{A} = t \xrightarrow[\hat{l} \to \hat{r}]{u} t'$ Reduktionen und $v \in \mathrm{RedOcc}_{R,I}(t)$. Dann ist

$$v \setminus A = v \setminus \hat{A}$$

Beweis:
Für $v = u$, $v \parallel u$ und $v < u$ ist die Aussage trivial.
Sei $v > u$. Es ist zu zeigen, daß

$$v \setminus A = \{u.w'.v' \mid v = u.w.v', l/w = r/w' \in X\} = \{u.\hat{w}'.\hat{v}' \mid v = u.\hat{w}.\hat{v}', \hat{l}/\hat{w} = \hat{r}/\hat{w}' \in X\} = v \setminus \hat{A}$$

Aus $v > u$ folgt $v = u.u'$ für ein $u' \in \mathbb{N}_+^*$. Gemäß Lemma 2.4 über Redexe eines Terms und deren Stellen läßt sich u' eindeutig zerlegen in

$$u' = w.v' \text{ mit } l/w = x \in X$$

und in

$$u' = \hat{w}.\hat{v}' \text{ mit } \hat{l}/\hat{w} = \hat{x} \in X$$

O. B. d. A. sei $w \leq \hat{w}$, d. h.

$$\hat{w} = w.u'' \text{ und } v' = u''.\hat{v}' \tag{1}$$

Da die Reduktion $t \xrightarrow{u} t'$ sowohl aufgrund von $l \to r$ als auch $\hat{l} \to \hat{r}$ möglich ist, existieren Substitutionen σ, $\hat{\sigma}$ mit

$$l\sigma = t/u = \hat{l}\hat{\sigma} \tag{2}$$

Sei $z \in X$ ($z \notin \mathrm{Var}(t/u)$). Es werden nun zwei modifizierte Substitutionen σ' und $\hat{\sigma}'$ wie folgt definiert:

$$y\sigma' := \begin{cases} (\hat{l}/w)\hat{\sigma}' & \text{, wenn } y = x \\ y\sigma & \text{, andernfalls} \end{cases} \tag{3}$$

$$y\hat{\sigma}' := \begin{cases} z & \text{, wenn } y = \hat{x} \\ y\hat{\sigma} & \text{, andernfalls} \end{cases} \tag{4}$$

für alle $y \in X$.
Dann gilt

$$l\sigma' = \hat{l}\hat{\sigma}' \tag{5}$$

Denn: Sei $u''' \in \mathrm{Occ}(l\sigma')$.

Fall 1: $u''' \parallel w$ oder $u''' < w$.

Da nach (1) $w \leq \hat{w}$, gilt:

$$l\sigma'(u''') = l\sigma(u''') \overset{(2)}{=} \hat{l}\hat{\sigma}(u''') = \hat{l}\hat{\sigma}'(u''')$$

Fall 2: $w \leq u'''$, d. h. $u''' = w.w''$ für ein $w'' \in \mathbb{N}_+^*$.

$$l\sigma'(u''') = (l\sigma'/w)(w'') = x\sigma'(w'') \overset{(3)}{=} (\hat{l}/w)\hat{\sigma}'(w'') = (\hat{l}\hat{\sigma}'/w)(w'') = \hat{l}\hat{\sigma}'(u''')$$

Aus (5) folgt nun gemäß der Eindeutigkeitkeitsbedingung beinahe orthogonaler Termersetzungssysteme:

$$r\sigma' = \hat{r}\hat{\sigma}'$$

Hieraus folgt insbesondere

$$\mathrm{Occ}(z, r\sigma') = \mathrm{Occ}(z, \hat{r}\hat{\sigma}') \tag{6}$$

Aus (1) und (3), (4) folgt

$$\mathrm{Occ}(z, x\sigma') = \{u''\} \text{ und } \mathrm{Occ}(z, \hat{x}\hat{\sigma}') = \{\varepsilon\}$$

und damit gilt:

$$\begin{aligned} \mathrm{Occ}(z, r\sigma') &= \{w'.u'' \mid r/w' = x\} \\ \mathrm{Occ}(z, \hat{r}\hat{\sigma}') &= \{\hat{w}' \mid \hat{r}/\hat{w}' = \hat{x}\} \end{aligned}$$

Zusammen mit (6) folgt dann

$$\{w'.u'' \mid r/w' = x\} = \{\hat{w}' \mid \hat{r}/\hat{w}' = \hat{x}\}$$

Zusammen mit (1) gilt

$$\{w'.v' \mid r/w' = x\} = \{w'.u''.\hat{v}' \mid r/w' = x\} = \{\hat{w}'.\hat{v}' \mid r/\hat{w}' = \hat{x}\}$$

Dies besagt schließlich das gewünschte. □

Wir haben gesagt, daß die Residuenabbildung die Umordnung von Redexen beschreibt. Somit erwarten wir, daß die Redexmenge $\mathrm{Red}_{R,I}$ im folgenden Sinne abgeschlossen ist:

Definition 2.2 Residuale Abgeschlossenheit
Eine Redexmenge $\mathrm{Red}_{R,I}$ heißt genau dann **residual abgeschlossen**, wenn für alle Reduktionen $t \xrightarrow[R,I]{} t'$ und alle Stellen $u \in \mathrm{RedOcc}_{R,I}$ gilt:

$$u \setminus t \xrightarrow[R,I]{} t' \subseteq \mathrm{RedOcc}_{R,I}(t')$$

□

Nicht jede Redexmenge ist residual abgeschlossen. Insbesondere für die Menge aller Redexe eines beinahe orthogonalen Termersetzungssystems ist dies jedoch gegeben.

Lemma 2.6 Residuale Abgeschlossenheit von Red_R
Die Menge aller Redexe eines beinahe orthogonalen Termersetzungssystems R, Red_R, ist residual abgeschlossen.
Insbesondere ist für eine Reduktion $t \xrightarrow{u} t'$ und $v \in \mathrm{RedOcc}_R(t)$ mit $v \not< u$ für alle $\hat{v} \in v \setminus t \xrightarrow{u} t'$ $t/v = t'/\hat{v}$.

Beweis:
Die Reduktion $t \xrightarrow[l \to r]{u} t'$ mit $u \in \mathrm{RedOcc}_R(t)$ erfolge mit der Substitution σ. Sei $v \in \mathrm{RedOcc}_R(t)$ beliebig.

$v = u$: $v \setminus t \xrightarrow{u} t' = \emptyset \subseteq \mathrm{RedOcc}_R(t')$.

$v \parallel u$: Es ist $t'/v = t/v$ und somit $v \setminus t \xrightarrow{u} t' = \{v\} \subseteq \mathrm{RedOcc}_R(t')$.

$v < u$: Da $v \in \mathrm{RedOcc}_R(t)$, existiert eine Regel $\hat{l} \to \hat{r} \in R$ und eine Substitution $[t_1/x_1, \ldots, t_n/x_n] : \mathrm{Var}(l) \to \mathrm{T}_\Sigma(X)$ mit

$$t/v = \hat{l}[t_1/x_1, \ldots, t_n/x_n]$$

Gemäß Lemma 2.4 über Redexe eines Termes und deren Stellen läßt sich u eindeutig zerlegen in

$$u = v.w.u'$$

mit $\hat{l}/w = x_k \in X$ für ein $k \in [n]$. Es gilt

$$t'/v = t/v[w.u' \leftarrow r\sigma] = \hat{l}[t_1/x_1, \ldots, t_n/x_n][w.u' \leftarrow r\sigma] =: e$$

Aus der Linearität von $\hat{l}$ folgt $\mathrm{Occ}(x_k, \hat{l}) = \{w\}$ und somit gilt:

$$t'/v = e = \hat{l}[t_1/x_1, \ldots, t_k[u' \leftarrow r\sigma]/x_k, \ldots, t_n]$$

Also ist auch t'/v ein Redex und es gilt $v \setminus t \xrightarrow{u} t' = \{v\} \subseteq \mathrm{RedOcc}_R(t')$.

$u < v$: Nach Definition ist $v \setminus t \xrightarrow{u} t' = \{u.w'.v' \mid v = u.w.v', r/w' = l/w \in X\}$. Sei $\hat{v} \in v \setminus t \xrightarrow{u} t'$. Somit ist $\hat{v} = u.\hat{w}'.\hat{v}'$ mit $r/\hat{w}' = l/\hat{w} = \hat{x} \in X$ und $v = u.\hat{w}.\hat{v}'$.

$$t'/\hat{v} = (t'/u)/\hat{w}'.\hat{v}' = r\sigma/\hat{w}'.\hat{v}' = \hat{x}\sigma/\hat{v}' = l\sigma/\hat{w}.\hat{v}' = (t/u)/\hat{w}.\hat{v}' = t/v$$

Also ist $t'/\hat{v} = t/v$ ein Redex und es gilt $v \setminus t \xrightarrow{u} t' \subseteq \mathrm{RedOcc}(t')$.

□

Es ergibt sich direkt aus der Definition der Residuenabbildung, daß die Residuen einer Redexstelle voneinander unabhängig sind. Dies läßt sich noch für Residuen mehrerer unabhängiger Redexstellen verallgemeinern:

Lemma 2.7 Unabhängigkeit von Residuen unabhängiger Redexstellen
Sei $t \xrightarrow{w} t'$ eine Reduktion und $u, v \in \mathrm{RedOcc}(t)$. Es gilt:

$$u \parallel v \implies (u \setminus t \xrightarrow{w} t') \parallel (v \setminus t \xrightarrow{w} t')$$

Beweis:
Gegeben ist $t \xrightarrow[l \to r]{w} t'$. Unter der Voraussetzung $u \parallel v$ können insgesamt 5 verschiedene Fälle vorliegen:

Fall 1: $w < u$ und $w < v$.

Nach Lemma 2.4 über die Redexe eines Terms und deren Stellen lassen sich u und v eindeutig zerlegen in

$$u = w.w_u.u' \text{ und } v = w.w_v.v'$$

mit $l/w_u \in X$ und $l/w_v \in X$.

$w_u = w_v$: Somit ist $\{w'_u \mid r/w'_u = l/w_u\} = \{w'_v \mid r/w'_v = l/w_v\}$, aber aufgrund von $v \parallel u$ ist $v' \parallel u'$. Es folgt damit

$$u \setminus t \xrightarrow{w} t' = \{w.w'_u.u' \mid r/w'_u = l/w_u\} \parallel \{w.w'_v.v' \mid r/w'_v = l/w_v\} = v \setminus t \xrightarrow{w} t'$$

$w_u \neq w_v$: Da l linear ist, ist $l/w_u \neq l/w_v$. Da Variablenstellen grundsätzlich voneinander unabhängig sind, gilt somit auch $w'_u \parallel w'_v$ für alle w'_u und w'_v mit $r/w'_u = l/w_u$ und $r/w'_v = l/w_v$. Also ist auch

$$u \setminus t \xrightarrow{w} t' = \{w.w'_u.u' \mid r/w'_u = l/w_u\} \parallel \{w.w'_v.v' \mid r/w'_v = l/w_v\} = v \setminus t \xrightarrow{w} t'$$

Fall 2: $w < u$ und $w \parallel v$.

Gemäß Definition gilt für alle $u'' \in u \setminus t \xrightarrow{w} t'$, daß $w < u''$. Andererseits ist $v \setminus t \xrightarrow{w} t' = \{v\}$. Da $w \parallel v$, ist somit auch $u \setminus t \xrightarrow{w} t' \parallel v \setminus t \xrightarrow{w} t'$.

Fall 3: $w < v$ und $w \parallel u$.

Analog zu Fall 2.

Fall 4: $u \leq w$.

Da $u \parallel v$, muß $w \parallel v$ sein. Somit ist $v \setminus t \xrightarrow{w} t' = \{v\}$ und $u \setminus t \xrightarrow{w} t' = \{u\}$ oder $= \emptyset$. Also ist $u \setminus t \xrightarrow{w} t' \parallel v \setminus t \xrightarrow{w} t'$.

Fall 5: $v \leq w$.

Analog zu Fall 4.

□

In [O'Do77] ist eine Residuenabbildung schlicht eine Abbildung r mit den folgenden 3 Eigenschaften:

- $v \in \mathrm{RedOcc}_{R,I}(t),\ \hat{v} \in r(v, t \xrightarrow{u} t') \implies \hat{v} \in \mathrm{RedOcc}_{R,I}(t')$
- $u, v \in \mathrm{RedOcc}_{R,I}(t),\ u \parallel v \implies r(u, t \xrightarrow{w} t') \parallel r(v, t \xrightarrow{w} t')$
- $r(u, t \xrightarrow{u} t') = \emptyset$

Somit ist für residual abgeschlossene Redexmengen $\mathrm{Red}_{R,I}$ die hier definierte Residuenabbildung auch eine Residuenabbildung im Sinne von [O'Do77], und die dort bewiesenen Lemmata können übernommen werden. Freilich wird dort die Residuenabbildung für alle Stellen $\mathbb{N}_+^*$ definiert, hier nur für die Redexstellen Red_R, um die Unabhängigkeit der Residuen von der bei der Reduktion verwendeten Regel, wie in Lemma 2.5 bewiesen, sicherzustellen. Da jedoch auch in [O'Do77] die Residuenabbildung nur auf Redexstellen angewendet wird, ist dieser Unterschied bedeutungslos.

Die Residuenabbildung ist **kanonisch** auf Mengen von Redexstellen **erweiterbar**:

$$U \setminus t \longrightarrow t' := \bigcup \{u \setminus t \longrightarrow t' \mid u \in U\}$$

Auch für parallele Reduktion definieren wir die Residuenabbildung:

$$u \setminus t \xrightarrow{V} t' := \begin{cases} \{u\} & \text{, falls für alle } v \in V\ v \parallel u \text{ oder } u < v \\ u \setminus t \xrightarrow{v} t'' & \text{, falls } v \in V \text{ mit } v \leq u \text{ existiert} \end{cases}$$

Da V eine Menge voneinander unabhängiger Redexstellen ist, ist im zweiten Fall v eindeutig bestimmt. Die Erweiterung auf Mengen von Redexstellen erfolgt wie oben.

Ist die Redexmenge $\mathrm{Red}_{R,I}$ residual abgeschlossen, so lassen sich Reduktionen vertauschen.

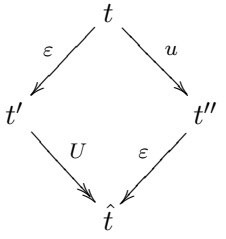
Lemma 2.8 **Einfaches Vertauschen von Reduktionen**
Sei $\mathrm{Red}_{R,I}$ residual abgeschlossen. Wenn

$$t \xrightarrow[R,I]{\varepsilon} t' \text{ und } t \xrightarrow[R,I]{u} t''$$

Reduktionen mit $u \neq \varepsilon$ sind, dann existiert $\hat{t} \in \mathrm{T}_\Sigma$, so daß

$$t' \xrightarrow[R,I]{U} \hat{t} \text{ und } t'' \xrightarrow[R,I]{\varepsilon} \hat{t}$$

mit $U := u \setminus t \xrightarrow[R,I]{\varepsilon} t'$ Reduktionen sind. Kurz:



Beweis:
Gegeben sind

$$t \xrightarrow[l \to r]{\varepsilon} t' \tag{1}$$

$$t \xrightarrow[\hat{l} \to \hat{r}]{u} t'' \tag{2}$$

(1) impliziert

$$t = l[t_1/x_1, \ldots, t_n/x_n] \tag{3}$$

$$t' = r[t_1/x_1, \ldots, t_n/x_n] \tag{4}$$

mit $\{x_1, \ldots, x_n\} = \mathrm{Var}(l)$, $t_1, \ldots t_n \in \mathrm{T}_\Sigma$.
Da $\varepsilon < u$, folgt nach Lemma 2.4 über Redexe eines Terms und deren Stellen, daß $w, u' \in \mathbb{N}_+^*$ mit

$$u = w.u' \text{ und } l/w = x_k \in X \tag{5}$$

für ein $k \in [n]$ existieren.
Aus (2) und (5) folgt

$$t_k = t/w \xrightarrow[\hat{r} \to \hat{l}]{u'} t''/w = t_k[u' \leftarrow \hat{r}\hat{\sigma}]$$

für eine Substitution $\hat{\sigma}$ und

$$t'' = t[u \leftarrow \hat{r}\hat{\sigma}] \overset{(3)}{=} (l[t_1/x_1, \ldots, t_n/x_n])[w.u' \leftarrow \hat{r}\hat{\sigma}] =: e$$

Da l linear ist, ist $\{w\} = \mathrm{Occ}(x_k, l)$ und es folgt

$$t'' = e = l[t_1/x_1, \ldots, t_k[u' \leftarrow \hat{r}\hat{\sigma}]/x_k, \ldots, t_n/x_n] \tag{6}$$

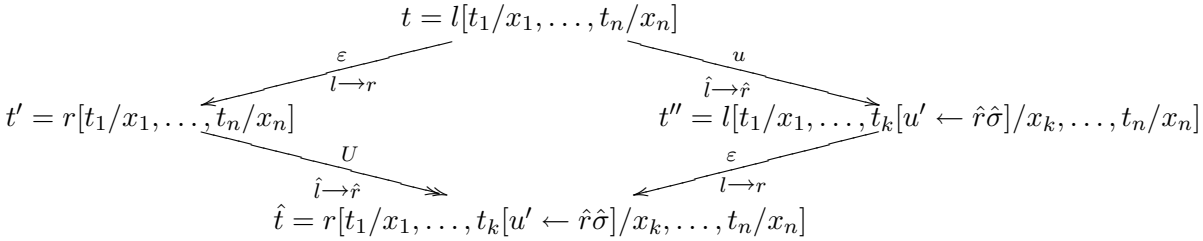
Aufgrund der residualen Abgeschlossenheit von $\mathrm{Red}_{R,I}$ ist $\{\varepsilon\} = \varepsilon \setminus t \xrightarrow{u} t'' \subseteq \mathrm{RedOcc}_{R,I}(t'')$ und somit $t'' \in \mathrm{Red}_{R,I}$. Hieraus folgt

$$t'' \overset{(6)}{=} l[t_1/x_1, \ldots, t_k[u' \leftarrow \hat{r}\hat{\sigma}]/x_k, \ldots, t_n/x_n] \xrightarrow[l \to r]{\varepsilon} r[t_1/x_1, \ldots, t_k[u' \leftarrow \hat{r}\hat{\sigma}]/x_k, \ldots, t_n/x_n] =: \hat{t} \tag{7}$$

Aus (5) folgt $U = u \setminus t \xrightarrow{\varepsilon} t' = \{w'.u' \mid r/w' = x_k\}$, und es gilt

$$\begin{array}{lll} t' & \overset{(4)}{=} & r[t_1/x_1, \ldots, t_n/x_n] \\ & \xrightarrow[\hat{l} \to \hat{r}]{U} & r[t_1/x_1, \ldots, t_n/x_n][v \leftarrow \hat{r}\hat{\sigma} \mid v \in U] \\ & = & r[t_1/x_1, \ldots, t_n/x_n][w'.u' \leftarrow \hat{r}\hat{\sigma} \mid r/w' = x_k] \\ & = & r[t_1/x_1, \ldots, t_k[u' \leftarrow \hat{r}\hat{\sigma}]/x_k, \ldots, t_n/x_n] \\ & \overset{(7)}{=} & \hat{t} \end{array}$$

Der Beweis läßt sich folgendermaßen kurz skizzieren:



□

Das einfache Vertauschen von Reduktionen läßt sich verallgemeinern zum zentralen allgemeinen Residuenlemma (Parallel Moves Lemma in [Huet&Lévy79]).

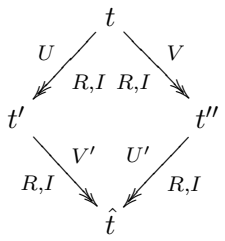
Lemma 2.9 Allgemeines Residuenlemma
Sei $\mathrm{Red}_{R,I}$ residual abgeschlossen. Wenn

$$t \xrightarrow[R,I]{U}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t' \text{ und } t \xrightarrow[R,I]{V}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t''$$

Reduktionen sind, dann existiert $\hat{t} \in \mathrm{T}_\Sigma$, so daß

$$t' \xrightarrow[R,I]{V'}\!\!\!\!\rightarrow\!\!\!\!\rightarrow \hat{t} \text{ und } t'' \xrightarrow[R,I]{U'}\!\!\!\!\rightarrow\!\!\!\!\rightarrow \hat{t}$$

mit $U' = U \setminus t \xrightarrow[R,I]{V}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t''$ und $V' = V \setminus t \xrightarrow[R,I]{U}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t'$ Reduktionen sind.
Kurz:



Beweis:
Die Reduktionsrelation $\xrightarrow[R,I]{}$ ist geschlossen (closed) nach [O'Do77], d. h. es gilt:

> Wenn $t \xrightarrow[R,I]{\varepsilon} t'$ und $t \xrightarrow[R,I]{u} t''$ Reduktionen mit $u \neq \varepsilon$ sind, dann existiert $\hat{t} \in \mathrm{T}_\Sigma$, so daß
>
> - $t'' \xrightarrow[R,I]{\varepsilon} \hat{t}$ und $t' \xrightarrow[R,I]{U}\!\!\!\!\rightarrow\!\!\!\!\rightarrow \hat{t}$ mit $U := u \setminus t \xrightarrow[R,I]{\varepsilon} t'$ Reduktionen sind, und
> - $v \in \mathrm{RedOcc}_{R,I}(t),\ v \parallel u \implies v \in \mathrm{RedOcc}_{R,I}(t''),\ v \setminus t'' \xrightarrow[R,I]{\varepsilon} \hat{t} = v \setminus t \xrightarrow[R,I]{\varepsilon} t'$ gilt.

Die erste Teileigenschaft ist im letzten Lemma bewiesen worden. Bezüglich der zweiten läßt sich feststellen, daß $v \setminus t \xrightarrow[R,I]{u} t'' = \{v\}$ und somit $v \in \mathrm{RedOcc}_{R,I}(t'')$, und

$$v \setminus t'' \xrightarrow[R,I]{\varepsilon} \hat{t} = v \setminus t'' \xrightarrow[l\longrightarrow r]{\varepsilon} \hat{t} = \{w'.v' \mid v = w.v', l/w = r/w' \in X\} = v \setminus t \xrightarrow[l\longrightarrow r]{\varepsilon} \hat{t'} = v \setminus t \xrightarrow[R,I]{\varepsilon} \hat{t'}.$$

Damit folgt die Aussage vermittels mehrfacher Induktion, siehe Lemmata 9 – 12 in [O'Do77]. □

Das allgemeine Residuenlemma impliziert, daß für jede residual abgeschlossene Redexmenge $\mathrm{Red}_{R,I}$ $\xrightarrow[R,I]{}\!\!\!\!\rightarrow\!\!\!\!\rightarrow$ stark konfluent und somit $\xrightarrow[R,I]{}\!\!\!\!\rightarrow\!\!\!\!\rightarrow$ und $\xrightarrow[R,I]{}$ konfluent sind ($\xrightarrow[R,I]{*} = \xrightarrow[R,I]{*}\!\!\!\!\rightarrow\!\!\!\!\rightarrow$). Mit Lemma 2.6 über die residuale Abgeschlossenheit der Redexmenge Red_R folgt die Konfluenz von $\xrightarrow[R]{}\!\!\!\!\rightarrow\!\!\!\!\rightarrow$ und $\xrightarrow[R]{}$.

Das Residuenlemma läßt sich auch so interpretieren, daß die Reduktion $t' \xrightarrow[R,I]{V'}\!\!\!\!\rightarrow\!\!\!\!\rightarrow \hat{t}$ durch die Reduktion von $t \xrightarrow[R,I]{V} t''$ nach $t \xrightarrow[R,I]{U} t'$ entsteht. Diese Reduktion einer Reduktion kann auf eine Reduktionsfolge induktiv fortgesetzt werden ([O'Do77], Def. 27; [Huet&Lévy79]). Auf diese Weise erhält man das Residuum A' einer Reduktionsfolge A. A' besitzt sehr ähnliche Eigenschaften wie A. Dieses werden wir später in Induktionsbeweisen ausnutzen.

Definition 2.3 Reduktion einer Reduktionsfolge
Sei $A = t_1 \xrightarrow[R,I]{U_1}\!\!\!\!\rightarrow\, t_2 \xrightarrow[R,I]{U_2}\!\!\!\!\rightarrow \ldots$ eine unendliche Reduktionsfolge und $B = t_1 \xrightarrow[R,I]{V_1}\!\!\!\!\rightarrow\, t'_1$ eine Reduktion. Seien $V_i \subseteq \mathrm{RedOcc}_{R,I}(t_i), t'_i \in \mathrm{T}_\Sigma, W_i \subseteq \mathrm{RedOcc}_{R,I}(t'_i)$ gegeben durch

$$V_{i+1} := V_i \setminus t_i \xrightarrow[R,I]{U_i}\!\!\!\!\rightarrow\, t_{i+1}$$

$$t_i \xrightarrow[R,I]{V_i}\!\!\!\!\rightarrow\, t'_i$$

$$W_i := U_i \setminus t_i \xrightarrow[R,I]{V_i}\!\!\!\!\rightarrow\, t'_i$$

für alle $i \in \mathbb{N}_+$.
Dann heißt $A \setminus B := t'_1 \xrightarrow[R,I]{W_1}\!\!\!\!\rightarrow\, t'_2 \xrightarrow[R,I]{W_2}\!\!\!\!\rightarrow \ldots$ **Residuum der Reduktion** A nach B und

$$\langle (t_i)_{i\in\mathbb{N}_+}, (t'_i)_{i\in\mathbb{N}_+}, (U_i)_{i\in\mathbb{N}_+}, (V_i)_{i\in\mathbb{N}_+}, (W_i)_{i\in\mathbb{N}_+} \rangle$$

Reduktionskonstruktion zu A nach B.
Die Reduktionskonstruktion läßt sich folgendermaßen darstellen:

$$\begin{array}{ccccccc}
t_1 & \xrightarrow[R,I]{U_1} & t_2 & \xrightarrow[R,I]{U_2} & t_3 & \xrightarrow[R,I]{U_3} & \cdots \\
\Big\downarrow{\scriptstyle V_1} & R,I & \Big\downarrow{\scriptstyle V_2} & R,I & \Big\downarrow{\scriptstyle V_3} & R,I & \\
t'_1 & \xrightarrow[R,I]{W_1} & t'_2 & \xrightarrow[R,I]{W_2} & t'_3 & \xrightarrow[R,I]{W_3} & \cdots
\end{array}$$

□

Neben der residualen Abgeschlossenheit existiert noch eine weitere Eigenschaft von Redexmengen, die in Kapitel 5 von Bedeutung sein wird.

Definition 2.4 Outer-Eigenschaft
Eine Redexmenge $\mathrm{Red}_{R,I}$ heißt genau dann **outer**, wenn

$$\begin{array}{l} t \xrightarrow[R,I]{w} t' \text{ ist eine Reduktion,} \\ u < v < w,\ v \in \mathrm{RedOcc}_{R,I}(t),\ u \in \mathrm{RedOcc}_{R,I}(t') \end{array} \implies u \in \mathrm{RedOcc}_{R,I}(t)$$

gilt (vgl. [O'Do77]). □

Diese Eigenschaft besagt, daß eine Reduktion an einer Stelle w, oberhalb derer sich an einer Stelle v ein weiterer Redex befindet, nicht oberhalb dieser Redexstelle v an einer Stelle u einen neuen Redex erzeugen kann.
Für die Redexmenge Red_R ist diese Eigenschaft gegeben.

Lemma 2.10 Outer-Eigenschaft der Redexe Red_R
Die Redexmenge Red_R ist outer.

Beweis:
Seien $t \xrightarrow[l \to r]{w} t'$ eine Reduktion, $u < v < w$, $v \in \mathrm{RedOcc}_R(t)$, $u \in \mathrm{RedOcc}_R(t')$. Also gibt es $v', w' \in \mathbb{N}_+^*$ mit $v = u.v'$ und $w = v.w' = u.v'.w'$. Aus der Reduktion folgt die Existenz einer Substitution σ mit

$$\begin{aligned} t/w &= t/u.v'.w' = l\sigma \\ t' &= t[w \leftarrow r\sigma] \end{aligned}$$

und somit

$$t'/u = t/u[v'.w' \leftarrow r\sigma]$$

und schließlich

$$t/u = t'/u[v'.w' \leftarrow l\sigma] \tag{1}$$

Da $u \in \mathrm{RedOcc}_R(t')$, existiert ein Redexschema $\hat{l} \in \mathrm{RedS}_R$ und eine Substitution $[t_1/x_1, \ldots, t_n/x_n] : \mathrm{Var}(\hat{l}) \to \mathrm{T}_\Sigma$ mit

$$t'/u = \hat{l}[t_1/x_1, \ldots, t_n/x_n] \tag{2}$$

Weil Red_R residual abgeschlossen ist und $v \setminus t \xrightarrow[l \to r]{w} t' = \{v\}$, ist $v \in \mathrm{RedOcc}_R(t')$. Nach Lemma 2.4 über Redexe und deren Stellen existieren $v_1, v_2 \in \mathbb{N}_+^*$ mit $v = u.v_1.v_2$ und $\hat{l}/v_1 \in X$, d. h. $\hat{l}/v_1 = x_k$ für ein $k \in [n]$.
Mit (1) und (2) folgt dann insgesamt

$$\begin{aligned} t/u &\overset{(1)}{=} t'/u[v'.w' \leftarrow l\sigma] \\ &\overset{(2)}{=} (\hat{l}[t_1/x_1, \ldots, t_n/x_n])[v'.w' \leftarrow l\sigma] \\ &= \hat{l}[t_1/x_1, \ldots, t_k[v_2.w' \leftarrow l\sigma]/x_k, \ldots, t_n/x_n] \end{aligned}$$

Somit ist $t/u \in \mathrm{Red}_R$ und also $u \in \mathrm{RedOcc}_R(t)$. □

Kapitel 3

Die Programme

Wir geben in 3.1 zwei Syntaxdefinitionen für konstruktorbasierte funktionale Programme erster Ordnung an. Hierbei setzen die ersteren Patternmatching zur Funktionsdefinition ein, während die letzteren explizite Test- und Dekompositionsfunktionen verwenden.

Diese Programme sind spezielle Arten von beinahe orthogonalen Termersetzungssystemen, und daher können wir in 3.2 deren Theorie auf Programme übertragen. Insbesondere ist die Reduktionsrelation eines Programms konfluent.

Somit sind Normalformen in Programmen eindeutig. Dies legt die Definition einer darauf basierenden operationellen Normalformsemantik nahe. Diese besitzt jedoch eine unschöne Eigenschaft: sie ist nicht invariant. Die Invarianz und weitere grundlegende, für formale Semantiken unserer Programme benötigte Begriffe werden in 3.3 definiert.

3.1 Die abstrakte Syntax

Anfangs haben wir festgestellt, daß eine Programmiersprache aus den zwei Komponenten Syntax und Semantik besteht. Wir definieren hier die Syntax zweier verschiedener konstruktorbasierter funktionaler Programmarten erster Ordnung. Später werden wir zu jeder Syntax mehrere verschiedene Semantiken angeben. Somit werden wir auch insgesamt nicht nur zwei, sondern weitaus mehr verschiedene Programmiersprachen erhalten.

Wir geben die Syntax nicht durch kontextfreie Grammatiken — etwa in Backus-Naur-Form — an. Derartige Grammatiken enthalten viele irrelevante Details und beschreiben hauptsächlich das äußere Erscheinen eines Programms, wie es vom Programmierer gesehen wird. Beispielsweise legen sie fest, ob Funktionsdefinitionen durch Kommata, Semikola oder Punkte getrennt oder abgeschlossen werden, und welche arithmetischen Operatoren stärker binden (`*`) als andere (`+`). Dieser „syntaktische Zucker“ ist für die Lesbarkeit eines Programms von Bedeutung, für die formale Manipulation von Programmen sind diese Details jedoch störend. Schon in 2.4 haben wir Terme rein durch ihre algebraischen Eigenschaften definiert. Entsprechend geben wir auch hier durch **abstrakte Syntax** die syntaktische Struktur von Programmen an (vgl. Kapitel 3 in [Meyer90], auch [ADJ77]).

Wir wollen uns nun zuerst den Programmen zuwenden, die Patternmatching zur Funktionsspezifikation verwenden. Bereits in der Einleitung haben wir ein Beispiel für ein Programm mit Pattern gesehen:

Beispiel 3.1 **Addition und Multiplikation mit Pattern**

```
add(x,Zero)      →  x
add(x,Succ(y))   →  Succ(add(x,y))

mult(x,Zero)     →  Zero
mult(x,Succ(y))  →  add(x,mult(x,y))
```

□

Programme bestehen also hauptsächlich aus Termen über einer Signatur. Die klare Unterscheidung zwischen den die Datenelemente aufbauenden Konstruktoren (`Zero`, `Succ`) und den eigentlichen Funktionen (`add`, `mult`) bildet die Grundlage konstruktorbasierter funktionaler Programmiersprachen und steht damit übrigens im Gegensatz zu der bei algebraischen Spezifikationen ([Wir90]) üblichen Vorgehensweise. Die Unterscheidung legen wir schon in den speziellen Signaturen für Programme, den Programmsignaturen, fest.

Definition 3.1 **Programmsignatur**
Sei $\mathcal{C}$ eine Signatur von **Konstruktorsymbolen**, die mindestens eine Konstante enthält, d. h. $\mathcal{C}_0 \neq \emptyset$. Sei $\mathcal{F}$ eine Signatur von **Funktionssymbolen**. Es gelte $\mathcal{C} \cap \mathcal{F} = \emptyset$. Dann ist $\Sigma = (\mathcal{C}, \mathcal{F})$ eine **Programmsignatur**.
Wir fassen die Programmsignatur oft auch als Signatur $\Sigma = \mathcal{C} \dot{\cup} \mathcal{F}$ auf wobei jedoch noch jedes Operationssymbol $f \in \Sigma$ eindeutig als Konstruktor- oder Funktionssymbol identifizierbar sein soll.
□

Zur einfacheren Unterscheidung verwenden wir grundsätzlich Großbuchstaben für die Bezeichnung von Konstruktorsymbolen. Funktionssymbole und auch Symbole der Programmsignatur allgemein werden durch Kleinbuchstaben gekennzeichnet. Auch in konkreten Beispielen verwenden wir diese Konvention, wobei Bezeichner von Konstruktoren dann mit einem Großbuchstaben beginnen. Die funktionalen Programmiersprachen Miranda und Haskell erzwingen dies durch ihre Syntax.

Definition 3.2 **Programm mit Pattern**
Sei Σ eine Programmsignatur und X eine Variablenmenge. Ein geordnetes Paar von Termen

$$f(p_1, \ldots, p_n) \quad \rightarrow \quad r$$

mit $f^{(n)} \in \Sigma$, $p_1, \ldots, p_n \in \mathrm{T}_{\mathcal{C}}(X)$, $l := f(p_1, \ldots, p_n)$ linear und $r \in \mathrm{T}_{\Sigma}(\mathrm{Var}(\vec{p}))$ heißt **Programmregel mit Pattern**. Eine endliche Menge P von Programmregeln mit Pattern, die die **Eindeutigkeitsbedingung**

$$l_1 \rightarrow r_1, l_2 \rightarrow r_2 \in P,\ l_1\sigma_1 = l_2\sigma_2 \implies r_1\sigma_1 = r_2\sigma_2$$

erfüllt, heißt **Programm mit Pattern**. □

Auf den offensichtlichen Zusammenhang mit beinahe orthogonalen Termersetzungssystemen werden wir erst in 3.2 eingehen.
Die Eindeutigkeitsbedingung besagt, daß die Überlappung linker Programmregelseiten nur zugelassen wird, wenn unter den unifizierenden Substitutionen auch die entsprechenden rechten Regelseiten syntaktisch gleich sind.

In üblichen funktionalen Programmiersprachen wie Miranda oder Haskell ist jedoch auch beispielsweise[1]

```
f(A)  =  A
f(A)  =  B
```

ein syntaktisch korrektes Programm. Die Mehrdeutigkeit der Funktionsspezifikation wird bei Berechnungen durch die Wahl der jeweils ersten passenden Gleichung aufgelöst. `f(A)` hat also den Wert `A` und nicht `B`. Da unsere Programme jedoch Mengen von Programmregeln sind, existiert in ihnen gar keine Reihenfolge von Programmregeln. Dieses Nichtvorhandensein einer Reihenfolge wird übrigens erst durch die abstrakte Syntax deutlich; jeder Programmtext ist linear angeordnet. Die funktionale Programmiersprache Hope setzt unsere Eindeutigkeitsbedingung auch nicht voraus und verwendet trotzdem keine Regelreihenfolge. Bei Berechnungen wird die „speziellste“ Regel verwendet, diejenige, die zu den gegebenen Argumenten am besten paßt. Diese Vorgehensweise stellt jedoch auch gewisse (Ordnungs-)Bedingungen an die linken Seiten der Regeln eines Funktionssymbols.
Wir wollen vollkommen auf derartige künstliche Prioritäten verzichten. Es ist intuitiv einsichtig, daß aufgrund der Eindeutigkeitsbedingung durch unsere Programme wirklich Funktionen spezifiziert werden, die jedem Argument(tupel) (höchstens) einen Funktionswert zuordnen. Freilich kann die Forderung der Eindeutigkeitsbedingung und auch der Linkslinearität erst bei der Betrachtung der Semantik vollständig begründet werden. Auf die Eindeutigkeitsbedingung und Prioritäten in üblichen funktionalen Programmiersprachen werden wir in Kapitel 7 auch nochmals ausführlicher eingehen.
Das obige Beispiel 3.1 besitzt überhaupt keine überlappenden linken Programmregelseiten. Das folgende Programm nutzt dagegen die bei der Eindeutigkeitsbedingung noch gegebenen Freiheiten wirklich aus.

Beispiel 3.2 (Striktes) And

```
and(True,x)        →  x
and(y,True)        →  y
and(False,False)   →  False
```

Dieses Programm mit Pattern spezifiziert die (strikte; siehe auch später) logische $\wedge$-Verknüpfung kürzer als durch die näherliegenden 4 Programmregeln.
Es ist

$$\texttt{and}(\texttt{True},\texttt{x})[\texttt{True}/\texttt{x}] = \texttt{and}(\texttt{True},\texttt{True}) = \texttt{and}(\texttt{y},\texttt{True})[\texttt{True}/\texttt{y}]$$

aber auch

$$\texttt{x}[\texttt{True}/\texttt{x}] = \texttt{True} = \texttt{y}[\texttt{True}/\texttt{y}].$$

Die Grundtermersetzungsregel $\texttt{and}(\texttt{True},\texttt{True}) \rightarrow \texttt{True}$ ist also eine Instanz beider Programmregeln. □

[1] Aus Gründen der Analogie zu unseren Programmen erster Ordnung verwenden wir für alle Miranda-Beispiele in dieser Arbeit eine für Miranda ungewöhnliche Schreibweise. Miranda (und Haskell) besitzen keine echt mehrstelligen Funktionen, sondern realisieren diese üblicherweise durch Currying. Der hier erfolgte Einsatz von teilweise redundanten Klammerungen und Tupeln ist jedoch auch korrekt.

Die linken Programmregelseiten für ein Funktionssymbol $f^{(n)} \in \mathcal{F}$ müssen keineswegs in dem Sinne vollständig sein, daß zu jedem Argumenttupel $\vec{c} \in (\mathrm{T}_\mathcal{C})^n$ eine linke Regelseite $f(\vec{p})$ im Programm und eine Substitution $\sigma : \mathrm{Var}(\vec{p}) \to \mathrm{T}_\mathcal{C}$ mit $f(\vec{c}) = f(\vec{p})\sigma$ existiert. Es ist sogar möglich, daß zu einem Funktionssymbol überhaupt keine Programmregel existiert. Das folgende Beispiel zeigt ein sinnvolles Programm mit Pattern mit unvollständigen linken Programmregelseiten.

Beispiel 3.3 Listen mit `head` und `tail`
Der Datentyp Liste wird von den Konstruktoren $\texttt{Nil}^{(0)}$ (leere Liste) und $\texttt{Cons}^{(2)}$ (Konstruktion einer Liste aus einem Element und einer Restliste) aufgebaut. Wie allgemein üblich schreiben wir `Nil` als `[]` und `Cons` infix als `:`. Für eine Liste `[]:[]:[]:[]` schreiben wir auch `[[],[],[]]`.

```
head(x:xs)  →  x
tail(x:xs)  →  xs
```

`head`- und `tail`-Funktion sind natürlich partielle Abbildungen, die nicht für leere Listen definiert sind. □

Obiges Beispiel verdeutlicht noch einen weiteren Punkt. Unsere Listen können als Listenelemente nur wiederum Listen besitzen. Die Definition von Listen natürlicher Zahlen ist unmöglich, da wir dann auch sinnlose Datenelemente wie `Succ([])` erhalten würden. Wir besitzen nur eine einheitliche Menge von Datenelementen. Durch die Verwendung von **Sorten** schon bei der Definition der Signatur und der alleinigen Zulassung der Bildung von sortenkonformen Termen läßt sich dieses Problem lösen. [Wir90], [Cou90], [ADJ77] verwenden beispielsweise Sorten, die im Zusammenhang mit funktionalen Programmiersprachen Typen genannt werden, während andere wie [Nivat75], [Gue81] und [Der&Jou90] auf sie verzichten. Da unsere Untersuchungen vom Vorhandensein von Sorten unabhängig sind, und die Verwendung von Sorten einen größeren Aufwand mit zusätzlichen Bedingungen für die Sortenkonformität impliziert, und damit insbesondere das eigentlich Wesentliche verborgen wird, verzichten wir auf sie. Die Erweiterung aller Definitionen und Aussagen um Sorten ist einfach. Einzig und allein die Auswahl sinnvoller Beispiele ist ohne Sorten stark eingeschränkt.

Patternmatching ist in Implementationen schwer direkt realisierbar. Deshalb werden üblicherweise in funktionalen Programmiersprachen die Pattern der Regeln eines jeden Funktionssymbols in einen sogenannten Matching-Tree mit expliziten Test- und Dekompositionsfunktionen übersetzt (Kapitel 8 in [Fie&Har88]). Das folgende Programm mit Hilfsfunktionen ist äquivalent zu der Spezifikation der Additon und Multiplikation über natürlichen Zahlen in Beispiel 3.1, S. 38.

Beispiel 3.4 Addition und Multiplikation mit Hilfsfunktionen

$$\begin{array}{lcl} \texttt{add(x,y)} & \to & \texttt{cond}_{\texttt{Succ}}\texttt{(y, Succ(add(x, sel}_{\texttt{Succ},1}\texttt{(y))), x)} \\ \texttt{mult(x,y)} & \to & \texttt{cond}_{\texttt{Succ}}\texttt{(y, add(x, mult(x,sel}_{\texttt{Succ},1}\texttt{(y))), x)} \end{array}$$

□

Die Bedeutung der Hilfsfunktionen sei hier kurz informal erklärt: cond_G ist eine Kombination von Test und Verzweigung. Besitzt das erste Argument t_1 des Ausdrucks $\mathrm{cond}_G(t_1, t_2, t_3)$ an der Stelle ε das Konstruktorsymbol G, so ist $\mathrm{cond}_G(t_1, t_2, t_3) = t_2$; befindet sich dort jedoch ein anderes Konstruktorsymbol, so ist $\mathrm{cond}_G(t_1, t_2, t_3) = t_3$. Die Selektion $\mathrm{sel}_{G,i}$ dient zur Dekomposition von Termen. Besitzt t an der Stelle ε das Konstruktorsymbol G, ist also $t = G(t_1, \ldots, t_n)$, so ist $\mathrm{sel}_{G,i}(t) = t_i$.
Zu einem Konstruktorsymbol gehört also ein Verzweigungs- und im allgemeinen mehrere Selektionssymbole.

Definition 3.3 Kanonische Hilfssymbole zu einer Signatur
Sei $\mathcal{C}$ eine Signatur. Die dazugehörige **Signatur kanonischer Hilfssymbole** $\mathcal{H}$ ist definiert durch:

$$\begin{aligned}\mathcal{H} \quad := \quad & \{\mathrm{cond}_G^{(3)} \mid G \in \mathcal{C}\} \text{ (\textbf{Verzweigungssymbole})} \\ & \dot\cup\ \{\mathrm{sel}_{G,i}^{(1)} \mid G^{(n)} \in \mathcal{C}, i \in [n]\} \text{ (\textbf{Selektionssymbole})}\end{aligned}$$

Dabei soll $\mathcal{C} \cap \mathcal{H} = \emptyset$ sein. □

Die Hilfssymbole fügen wir jetzt als dritte Teilsignatur zu unserer Programmsignatur hinzu.

Definition 3.4 Programmsignatur mit Hilfssymbolen
Sei $\Sigma = (\mathcal{C}, \mathcal{F})$ eine Programmsignatur. Sei $\mathcal{H}$ die Signatur der kanonischen Hilfssymbole zur Signatur der Konstruktorsymbole $\mathcal{C}$ mit $\mathcal{H} \cap \Sigma = \emptyset$. Dann heißt $\Sigma' := (\mathcal{C}, \mathcal{H}, \mathcal{F})$ **Programmsignatur mit Hilfssymbolen**. Auch hier verwenden wir oft schlicht $\Sigma' = \mathcal{C} \dot\cup \mathcal{H} \dot\cup \mathcal{F}$. □

Aus folgendem Grund haben wir kombinierte Test- und Verzweigungssymbole cond_G anstelle eines einzigen Verzweigungssymbols (`if-then-else-fi`) und von Testsymbolen für jedes Konstruktorsymbol ($\mathtt{test}_G^{(1)}$) definiert: Bei einer Trennung müßten wir die Existenz zweier verschiedener Konstruktorterme $true, false \in \mathrm{T}_{\mathcal{C}}$ voraussetzen, die durch ihre Verwendung als logische Werte ausgezeichnet wären. Bei der Verwendung von Sorten würde dies praktisch die Existenz einer Sorte Boolean voraussetzen. Außerdem ist der Einsatz des Tests im ersten Argument der Verzweigung üblicherweise dessen einzige Anwendung, so daß wir die beiden auch gleich verschmelzen können.

Definition 3.5 Programm mit Hilfsfunktionen
Sei Σ eine Programmsignatur mit Hilfssymbolen und X eine Variablenmenge. Ein geordnetes Paar von Termen

$$f(x_1, \ldots, x_n) \quad \to \quad r$$

mit $f^{(n)} \in \mathcal{F}$, $x_1, \ldots, x_n \in X$ paarweise verschieden und $r \in \mathrm{T}_\Sigma(\{x_1, \ldots, x_n\})$ heißt **Programmregel mit Hilfsfunktionen**. Eine endliche Menge P von Programmregeln mit Hilfsfunktionen, die die **Eindeutigkeitsbedingung**

$$f(x_1, \ldots, x_n) \to r_1, f(y_1, \ldots, y_n) \to r_2 \in P \implies f(x_1, \ldots, x_n) \to r_1 = f(y_1, \ldots, y_n) \to r_2$$

erfüllt, heißt **Programm mit Hilfsfunktionen**. □

Die Einhaltung der **Konsistenzbedingung**

> Wenn eine rechte Regelseite r einen Teilterm $t' = \mathrm{sel}_{G,i}(t)$ besitzt, dann hat sie auch einen Teilterm $t'' = \mathrm{cond}_G(t, t_1, t_2)$, und t' ist ein Teilterm von t_1, d. h. vor jeder Anwendung der Selektionsoperation wird sichergestellt, daß $t = G(\ldots)$.
> Formaler: Zu $u \in \mathrm{Occ}(r)$ mit $r/u = t'$ existieren v, w mit $u = v.2.w$ und $r/v = t''$.

ist sicherlich vernünftig, wird jedoch nicht verlangt. Auch für den Fall eines Verstoßes gegen diese Konsistenzbedingung sind die später betrachteten Semantiken definiert.
Die Eindeutigkeitsbedingung besagt, daß zu jedem Funktionssymbol höchstens eine Programmregel existieren darf. Diese Forderung erfolgt mit der gleichen Intention wie bei der Eindeutigkeitsbedingung der Programme mit Pattern. Es handelt sich schlicht um eine angepaßte Form dieser Eindeutigkeitsbedingung, abgesehen davon, daß hier auch direkt der triviale Fall, daß zwei Programmregeln

sich nur durch Variablennamen unterscheiden (z. B. {`f(x)` → `x`, `f(y)` → `y`}), ausgeschlossen ist. Aus Symmetrie zu den Programmen mit Pattern fordern wir im Gegensatz zu Funktionsschemata ([Cou90], [Ind93], [Ind94], [Nivat75]) nicht, daß zu jedem Funktionssymbol eine Programmregel existiert.

Wir haben in der Einleitung gefordert, daß die Syntax einer Programmiersprache entscheidbar sein muß. Es ist offensichtlich, daß diese Bedingung sowohl für unsere Programme mit Pattern als auch die Programme mit Hilfsfunktionen erfüllt ist.

3.2 Die Reduktionsrelationen

Programme sind offensichtlich spezielle Termersetzungssysteme. Somit sind deren Reduktionsrelationen auch für Programme definiert und können als Grundlage operationeller Semantiken dienen. Allerdings existieren nur für Funktionssymbole Regeln. Da Konstruktoren die Datenelemente aufbauen, besitzen sie natürlich keine Regeln. Für vollständige operationelle Semantiken benötigen wir jedoch noch Regeln für die Hilfssymbole.

Definition 3.6 **Assoziiertes Termersetzungssystem eines Programms**
Sei P ein Programm mit Hilfsfunktionen über einer Programmsignatur mit Hilfssymbolen, Σ, und einer Variablenmenge X. Dann besteht das damit **assoziierte Termersetzungsystem** $\hat{P}$ genau aus den folgenden Regeln:

- $\hat{P}$ enthält alle Regeln aus P, d. h. $P \subset \hat{P}$.
- Für jedes Verzweigungssymbol $\text{cond}_G \in \mathcal{H}$ enthält $\hat{P}$ die Regel

$$\text{cond}_G(G(x_1, \ldots, x_n), x, y) \quad \rightarrow \quad x$$

  und für alle $H \in \mathcal{C}$ mit $H \neq G$ die Regel

$$\text{cond}_G(H(x_1, \ldots, x_{n_H}), x, y) \quad \rightarrow \quad y$$

- Für jedes Selektionssymbol $\text{sel}_{G,i} \in \mathcal{H}$ enthält $\hat{P}$ die Regel

$$\text{sel}_{G,i}(G(x_1, \ldots, x_i, \ldots, x_n)) \quad \rightarrow \quad x_i$$

Ist P ein Programm mit Pattern, so ist $\hat{P} := P$ das **assoziierte Termersetzungssystem des Programms P**.
Die Regeln eines assoziierten Termersetzungssystems $\hat{P}$ eines Programms P heißen **Reduktionsregeln des Programms P**. □

Die Bezeichung eines Programms mit Pattern als sein eigenes assoziiertes Termersetzungssystem erfolgt aus Symmetriegründen. Für ein Programm mit Pattern sind somit auch die Reduktionsregeln gleich den Programmregeln.
Die Reduktionsregeln der Hilfssymbole besitzen Pattern wie Programmregeln. Assoziierte Termersetzungssysteme von Programmen mit Hilfsfunktionen können daher als spezielle Programme mit Pattern bzw. deren assoziierte Termersetzungssysteme aufgefaßt werden. Allerdings sind die Pattern der Reduktionsregeln der Hilfssymbole von besonders einfacher Struktur. Vor allem sind diese

Reduktionsregeln fest definiert und nicht durch den Programmierer veränderbar. Somit können diese in Implementationen direkt effizient, ohne ein explizites Patternmatching, codiert werden (vgl. mit Stackcode für die Verzweigung in rekursiven Funktionsdefinitionen in [Ind93]).
Das assoziierte Termersetzungssystem eines Programms dient nun als Grundlage operationeller Semantiken. Es hat die folgende entscheidende Eigenschaft:

Lemma 3.1 Beinahe Orthogonalität assoziierter Termersetzungssysteme
Das assoziierte Termersetzungssystem jedes Programms (mit Pattern oder mit Hilfsfunktionen) ist beinahe orthogonal.

Beweis:
Assoziierte Termersetzungssysteme sind linkslinear, und die Erfüllung der Eindeutigkeitsbedingung beinahe orthogonaler Termersetzungssysteme folgt direkt aus der eingeschränkten Form linker Regelseiten und der Eindeutigkeitsbedingung der Programme mit Pattern bzw. mit Hilfsfunktionen. Auch die Reduktionsregeln der Hilfssymbole sind linkslinear und erfüllen die Eindeutigkeitsbedingung der Programme mit Pattern. □

Somit gelten alle Eigenschaften beinahe orthogonaler Termersetzungssysteme auch für Programme bzw. deren assoziierte Termersetzungssysteme. Insbesondere ist die Reduktionsrelation $\xrightarrow[\hat{P}]{}$ eines assoziierten Termersetzungssystems $\hat{P}$ konfluent; eine entscheidende Voraussetzung für die Wohldefiniertheit operationeller Semantiken, wie wir noch feststellen werden.
In Anbetracht der Tatsache, daß wir die Reduktionsrelationen eines Termersetzungssystems nur auf Grundtermen definiert haben, wird hier nicht deutlich, daß die Reduktionsrelation $\xrightarrow[R]{}$ eines beinahe orthogonalen Termersetzungssystems R auch über der Menge aller Terme $\mathrm{T}_\Sigma(X)$ konfluent ist. Wir benötigen jedoch nur Grundkonfluenz. Die Menge der grundkonfluenten Termersetzungssysteme ist echt größer als die Menge der konfluenten Termersetzungssysteme. Da jedoch Grundkonfluenz eine weitaus schwieriger zu verifizierende Eigenschaft als Konfluenz ist ([KNO90]), verzichten wir auf die Untersuchung diesbezüglich eventuell möglicher größerer Freiheitsgrade der Syntax unserer Programme.
Die Bedeutung der Eindeutigkeitsbedingung der Programme und der Forderung der Linkslinearität ist somit klar. Freilich mag es scheinen, daß in Anbetracht der übrigen syntaktischen Einschränkungen die Linkslinearität überflüssig wäre. Das folgende Beispiel demonstriert jedoch, daß dann die Konfluenz verloren ginge.

Beispiel 3.5 Fehlende Konfluenz bei fehlender Linkslinearität (S.174 in [Klop87])

```
d(x,x)  →  E
c(x)    →  d(x,c(x))
a       →  c(a)
```

Für den Term `c(a)` gibt es die zwei Reduktionsketten

```
c(a)  ⟶  d(a,c(a))  ⟶  d(c(a),c(a))  ⟶        E
c(a)  ⟶   c(c(a))   ⟶  c(d(a,c(a)))  ⟶  c(d(c(a),c(a)))  ⟶  c(E)
```

Die beiden Terme `E` und `c(E)` lassen sich jedoch nicht zu einem gemeinsamen Term reduzieren. `E` ist bereits in Normalform und `c(E)` läßt sich nicht zu `E` reduzieren. `c(E)` besitzt überhaupt keine Normalform. □

Es gibt auch einen schlichten, pragmatischen Grund für die Forderung der Linkslinearität: Ein Test auf Anwendbarkeit einer nicht-linkslinearen Regel bei einer Reduktion erfordert einen Test auf syntaktische Gleichheit der für die mehrfach auftretenden Variablen eingesetzten Argumente. Da Terme eventuell sehr groß sind, kann dies sehr zeitaufwendig sein. Bei linkslinearen Regeln wird dagegen nur ein Vergleich mit dem Nicht-Variablen-Teil eines Redexschemas benötigt; für Programmregeln mit Hilfsfunktionen ist der Test auf Anwendbarkeit sogar trivial.
Wir müssen zwischen den Programmregeln und den Reduktionsregeln eines Programms unterscheiden. Trotzdem werden wir das assoziierte Termersetzungssystem $\hat{P}$ eines Programms P selten explizit erwähnen, und wir schreiben Red_P für $\mathrm{Red}_{\hat{P}}$, RedOcc_P für $\mathrm{RedOcc}_{\hat{P}}$, $\xrightarrow[P]{}$ für $\xrightarrow[\hat{P}]{}$ und $\xrightarrow[P]{}\!\!\!\!\!\rightarrow$ für $\xrightarrow[\hat{P}]{}\!\!\!\!\!\rightarrow$.

Unsere Programme ermöglichen die einfache und intuitive Spezifikation von Funktionen bzw. Datentypen. Da die assoziierten Termersetzungssysteme spezielle Vertreter der wohluntersuchten und gut handhabbaren beinahe orthogonalen Termersetzungssysteme sind, können wir auf deren bekannte Eigenschaften und mächtige Beweismethoden (allgemeines Residuenlemma) zurückgreifen.

Im weiteren sei immer ein Programm P über einer Programmsignatur Σ und einer Variablenmenge X gegeben. Soweit nicht anders angegeben, kann dieses Programm P sowohl ein Programm mit Pattern sein, und es ist dann $\Sigma = (\mathcal{C}, \mathcal{F})$, als auch ein Programm mit Hilfsfunktionen mit $\Sigma = (\mathcal{C}, \mathcal{H}, \mathcal{F})$.

3.3 Formale Semantiken

Gemäß unserer Einleitung bestimmt eine operationelle Semantik eines Programms eine Eingabe-Ausgabe-Abbildung. Eine Eingabe für unsere Programme ist ein beliebiger Grundterm. Eine operationelle Semantik soll nun die Auswertung dieses Eingabeterms durch Übergänge zwischen teilweise ausgewerteten Termen beschreiben. Die Reduktionsrelation $\xrightarrow[P]{}$ eines Programms P ist eine derartige Übergangsrelation zwischen Grundtermen. Die Reduktion kann als Auswertung oder Berechnung aufgefaßt werden. Eine derartige operationelle Semantik wird daher auch **Reduktionssemantik** genannt.
Da die Reduktionsrelation $\xrightarrow[P]{}$ konfluent ist, ist die Normalform eines Terms eindeutig. Es liegt also nahe, die Normalform eines Terms als dessen semantischen Wert zu bezeichnen, und eine entsprechende Eingabe-Ausgabe-Abbildung zu definieren. Allerdings besitzt im allgemeinen nicht jeder Term eine Normalform. Wir wollen jedoch keine partielle Eingabe-Ausgabe-Abbildung definieren, da partielle Abbildungen mathematisch aufwendiger sind, und es sich vor allem in Hinblick auf die weiteren Betrachtungen als sinnvoll erweisen wird, einen zusätzlichen semantischen Wert $\perp$ für diesen Fall einzuführen. Da das Programm den semantischen Wert eines Terms ohne Normalform gewissermaßen nicht spezifiziert, wird $\perp$ als Undefiniertheit oder fehlende Information interpretiert.
Unsere konstruktorbasierten Programme sollen Funktionen über Konstruktortermen spezifizieren. Somit definieren wir als semantische Wertemenge oder **Rechenbereich** die Menge $\mathrm{T}_{\mathcal{C}}^{\perp} := \mathrm{T}_{\mathcal{C}} \dot{\cup} \{\perp\}$.
Da die linken Regelseiten für ein Funktions- oder Hilfssymbol im allgemeinen nicht vollständig sind (z. B. für sel_G), und sogar Funktionssymbole ohne jegliche zugehörige Programmregel existieren können, sind Normalformen nicht immer Konstruktorterme. Konsequent wird auch diesen Termen der semantische Wert $\perp$ für fehlende Information bzw. Undefiniertheit zugewiesen.
Insgesamt führen diese Überlegungen zu der folgenden Normalformsemantik.

Definition 3.7 Normalformsemantik
Die **Normalformsemantik**

$$[\![\cdot]\!]_P^{\mathrm{nf}} : \mathrm{T}_\Sigma \to \mathrm{T}_\mathcal{C}^\perp$$

ist für einen Term $t \in \mathrm{T}_\Sigma$ definiert durch

$$[\![t]\!]_P^{\mathrm{nf}} := \begin{cases} t{\downarrow_P} & \text{, falls } t{\downarrow_P} \text{ existiert und } t{\downarrow_P} \in \mathrm{T}_\mathcal{C} \text{ ist} \\ \perp & \text{, andernfalls} \end{cases}$$

□

Die Normalformsemantik hat leider eine äußerst unangenehme Eigenschaft: sie ist nicht invariant. Verschiedene Terme mit gleicher Semantik verhalten sich im gleichen Kontext unterschiedlich, wie folgendes Beispiel demonstriert:

Beispiel 3.6 Fehlende Invarianz der Normalformsemantik

```
liste1      →  []:liste1
liste2      →  [[]]:liste2
head(x:xs)  →  x
```

(Dieses Programm mit Pattern läßt sich auch in ein äquivalentes Programm mit Hilfsfunktionen übersetzen.)
Wie man leicht sieht

$$\texttt{liste1} \xrightarrow[P]{} [] : \texttt{liste1} \xrightarrow[P]{} [] : [] : \texttt{liste1} \xrightarrow[P]{} \ldots$$

besitzen weder `liste1` noch `liste2` eine Normalform und es ist daher

$$[\![\texttt{liste1}]\!]_P^{\mathrm{nf}} = [\![\texttt{liste2}]\!]_P^{\mathrm{nf}} = \perp.$$

Aber es gilt

$$\begin{array}{llll} \texttt{head(}\underline{\texttt{liste1}}\texttt{)} & \xrightarrow[P]{} & \underline{\texttt{head}}\texttt{([]:liste1)} & \xrightarrow[P]{} \quad \texttt{[]} \\ \texttt{head(}\underline{\texttt{liste2}}\texttt{)} & \xrightarrow[P]{} & \underline{\texttt{head}}\texttt{([[]]:liste2)} & \xrightarrow[P]{} \quad \texttt{[[]]}, \end{array}$$

und somit erhalten wir

$$[\![\texttt{head(liste1)}]\!]_P^{\mathrm{nf}} = [] \neq [[]] = [\![\texttt{head(liste2)}]\!]_P^{\mathrm{nf}}.$$

Auch Terme, deren Teilterme nicht alle eine Normalform besitzen, können selber eine Normalform haben. □

Da Terme zusammengesetzte, strukturierte Gebilde sind, erwarten wir von einer Semantik, daß zwei Terme, die sich nur in Teiltermen unterscheiden, die den gleichen semantischen Wert besitzen, ebenfalls semantisch gleich sind. Die Eingabe-Ausgabe-Abbildung soll daher eine Grundtermsemantik in folgendem Sinne sein.

Definition 3.8 Grundtermsemantik
Sei A eine beliebige nicht-leere Menge. Dann heißt eine Abbildung $[\![\cdot]\!] : \mathrm{T}_\Sigma \to A$ **Grundtermsemantik**, falls sie **invariant** ist, d. h.

$$[\![t']\!] = [\![t'']\!] \implies [\![t[t'/x]]\!] = [\![t[t''/x]]\!]$$

für alle $t', t'' \in \mathrm{T}_\Sigma$ und $t \in \mathrm{T}_\Sigma(\{x\})$ mit $x \in X$. □

Basierend auf einer solchen Grundtermsemantik kann dann eine semantische Gleichheit definiert werden.

Definition 3.9 Semantische Gleichheit zu einer Grundtermsemantik
Sei $[\![\cdot]\!] : \mathrm{T}_\Sigma \to A$ eine Grundtermsemantik. Dann ist die **semantische Gleichheit zu der Grundtermsemantik $[\![\cdot]\!]$** die durch

$$t \sim_{[\![\cdot]\!]} t' \iff [\![t]\!] = [\![t']\!]$$

für alle $t, t' \in \mathrm{T}_\Sigma$ definierte Relation $\sim_{[\![\cdot]\!]} \subseteq \mathrm{T}_\Sigma \times \mathrm{T}_\Sigma$. □

Natürlich kann eine derartige Relation $\sim$ zu jeder beliebigen Abbildung $\varphi : \mathrm{T}_\Sigma \to A$ definiert werden. Aber aufgrund der Invarianz der Grundtermsemantik ist die semantische Gleichheit eine Kongruenz und Leibnitz' Gesetz des Ersetzens von Gleichem durch Gleiches gilt, d. h.

$$t' \sim_{[\![\cdot]\!]} t'' \implies t[t'/x] \sim_{[\![\cdot]\!]} t[t''/x]$$

für alle $t', t'' \in \mathrm{T}_\Sigma$ und $t \in \mathrm{T}_\Sigma(\{x\})$, wie wir es von einer „Gleichheit" erwarten.

Lemma 3.2 Kongruenzeigenschaft der semantischen Gleichheit
Die Semantische Gleichheit $\sim_{[\![\cdot]\!]}$ zu einer Grundtermsemantik $[\![\cdot]\!]$ ist eine Kongruenz.

Beweis:
Reflexivität, Symmetrie und Transitivität von $\sim_{[\![\cdot]\!]}$ sind trivialerweise gegeben. Natürlich ist $\sim_{[\![\cdot]\!]}$ genau dann invariant (abgeschlossen unter Kontextbildung), wenn die Abbildung $[\![\cdot]\!]$ invariant ist. □

Um im weiteren die Invarianz einer gegebenen Abbildung $[\![\cdot]\!] : \mathrm{T}_\Sigma \to A$ leichter überprüfen zu können, sei hier die folgende einfachere, äquivalente Definition der Invarianz angegeben.

Lemma 3.3 Charakterisierung der Invarianz
Eine Abbildung $[\![\cdot]\!] : \mathrm{T}_\Sigma \to A$ ist genau dann invariant, wenn

$$[\![t']\!] = [\![t'']\!] \implies [\![g(s_1, \ldots, s_{i-1}, t', s_{i+1}, \ldots, s_n)]\!] = [\![g(s_1, \ldots, s_{i-1}, t'', s_{i+1}, \ldots, s_n)]\!]$$

für alle $t', t'', s_1, \ldots, s_n \in T_\Sigma$, $g^{(n)} \in \Sigma$, $i \in [n]$.

Beweis:

$\Rightarrow$: Es ist klar, daß die obige Eigenschaft erfüllt ist, wenn $[\![\cdot]\!]$ invariant ist.

$\Leftarrow$: Seien $t', t'' \in T_\Sigma$ mit $[\![t']\!] = [\![t'']\!]$.

$t = x$: $(x \in X)$.
$[\![t[t'/x]]\!] = [\![t']\!] = [\![t'']\!] = [\![t[t''/x]]\!]$.

$\underline{t = g(s_1, \ldots, s_n)\text{:}}$ $(g^{(n)} \in \Sigma)$.

Nach Induktionsvoraussetzung gilt für alle s_i:

$$[\![s_i[t'/x]]\!] = [\![s_i[t''/x]]\!]$$

Somit folgt:

$$\begin{array}{rl} & [\![t[t'/x]]\!] \\ = & [\![g(s_1[t'/x], s_2[t'/x], \ldots, s_n[t'/x])]\!] \\ \overset{\text{I.V.}}{=} & [\![g(s_1[t''/x], s_2[t''/x], \ldots, s_n[t''/x])]\!] \\ = & [\![t[t''/x]]\!] \end{array}$$

□

Operationelle Semantiken unserer Programme sollen also letztendlich immer Grundtermsemantiken sein. Dagegen sollen denotationelle Semantiken einem Programm einen Datentyp, eine Algebra zuordnen. Wie wir schon in der Einleitung angedeutet haben, kann aus einem solchen Datentyp leicht eine Eingabe-Ausgabe-Abbildung, d. h. eine Grundtermsemantik gewonnen werden.

Definition 3.10 Algebraische Termsemantik
Sei $\mathfrak{A} = \langle A, \alpha \rangle$ eine Σ-Algebra, $Y \subseteq X$ und $\beta : Y \to A$ eine Variablenbelegung. Die **algebraische Termsemantik**

$$[\![\cdot]\!]^{\text{alg}}_{\mathfrak{A},\beta} : \mathrm{T}_\Sigma(Y) \to A$$

ist für einen Term $t \in \mathrm{T}_\Sigma(Y)$ induktiv definiert durch

$$\begin{array}{rcl} [\![x]\!]^{\text{alg}}_{\mathfrak{A},\beta} & := & \beta(x) \\ [\![g(\hat{t}_1, \ldots, \hat{t}_n)]\!]^{\text{alg}}_{\mathfrak{A},\beta} & := & g^{\mathfrak{A}}([\![\hat{t}_1]\!]^{\text{alg}}_{\mathfrak{A},\beta}, \ldots, [\![\hat{t}_n]\!]^{\text{alg}}_{\mathfrak{A},\beta}) \end{array}$$

für alle $x \in Y$,$g^{(n)} \in \Sigma$ und $\hat{t}_1, \ldots, \hat{t}_n \in \mathrm{T}_\Sigma(Y)$.
Ist t ein Grundterm, so ist β überflüssig und wir schreiben $[\![t]\!]^{\text{alg}}_{\mathfrak{A}}$. □

Die algebraische Termsemantik $[\![\cdot]\!]^{\text{alg}}_{\mathfrak{A},\beta}$ ist somit die eindeutige Fortsetzung von β zu einem Homomorphismus von der frei über X erzeugten Termalgebra $\mathcal{T}_\Sigma(X)$ in die Algebra $\mathfrak{A}$. Daraus folgt auch direkt die Invarianz der auf Grundterme beschränkten algebraischen Termsemantik $[\![\cdot]\!]^{\text{alg}}_{\mathfrak{A}} : \mathrm{T}_\Sigma \to A$. Somit ist $[\![\cdot]\!]^{\text{alg}}_{\mathfrak{A}}$ eine Grundtermsemantik, die **algebraische Grundtermsemantik**.
Umgekehrt können wir zu einer Grundtermsemantik auch Datentypen, d. h. Algebren definieren.

Definition 3.11 Datentyp einer Grundtermsemantik
Sei $[\![\cdot]\!] : \mathrm{T}_\Sigma \to A$ eine Grundtermsemantik. Dann heißt $\mathfrak{A}_{[\![\cdot]\!]} = \langle A, \alpha \rangle$ genau dann ein **Datentyp der Grundtermsemantik $[\![\cdot]\!]$**, wenn

$$g^{\mathfrak{A}}(a_1, \ldots, a_n) = [\![g(t_1, \ldots, t_n)]\!]$$

für alle $g^{(n)} \in \Sigma$, $t_1, \ldots, t_n \in \mathrm{T}_\Sigma$ und $a_1, \ldots, a_n \in A$ mit $[\![t_1]\!] = a_1, \ldots, [\![t_n]\!] = a_n$. □

Da aufgrund der Invarianz der Grundtermsemantik $[\![g(t_1,\ldots,t_n)]\!] = [\![g(s_1,\ldots,s_n)]\!]$, wenn $[\![t_1]\!] = [\![s_1]\!],\ldots,[\![t_n]\!] = [\![s_n]\!]$, sind die Datentypen einer Grundtermsemantik wohldefiniert. Hiermit wird jedoch auch deutlich, daß nur zu einer operationellen Semantik, die eine invariante Eingabe-Ausgabe-Abbildung definiert, überhaupt ein entsprechender denotationell definierter Datentyp existieren kann.
Wenn für jedes $a \in A$ ein $t \in \mathrm{T}_\Sigma$ mit $[\![t]\!] = a$ existiert, so gibt es genau einen Datentypen zu $[\![\cdot]\!]$. Meistens gibt es jedoch mehrere Datentypen zu einer Grundtermsemantik. Somit ist es, wie schon in der Einleitung erwähnt, im allgemeinen nicht möglich, aus der Eingabe-Ausgabe-Abbildung, d. h. der Grundtermsemantik, den Datentyp eines Programms zu gewinnen.
Für die Datentypen einer Grundtermsemantik gilt nun:

Korollar 3.4 Datentypen einer Grundtermsemantik
Sei $[\![\cdot]\!] : \mathrm{T}_\Sigma \to A$ eine Grundtermsemantik und $\mathfrak{A}_{[\![\cdot]\!]}$ ein Datentyp dieser Grundtermsemantik. Dann gilt

$$[\![t]\!] = [\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_{[\![\cdot]\!]}}$$

für alle $t \in \mathrm{T}_\Sigma$. □

Der Beweis der Übereinstimmung einer operationellen und einer denotationellen Semantik besteht somit darin, zu zeigen, daß der denotationell definierte Datentyp ein Datentyp der operationell definierten Grundtermsemantik, bzw. die algebraische Grundtermsemantik bezüglich des denotationell definierten Datentyps gleich der operationell definierten Grundtermsemantik ist.

Die Normalformsemantik hat neben der fehlenden Invarianz noch einen weiteren Nachteil: Die Reduktionsrelation $\xrightarrow[P]{}$ ist nicht deterministisch. Zu einem Term $t \in \mathrm{T}_\Sigma$ existieren im allgemeinen mehrere — um nicht zu sagen viele — Nachfolgerterme t' mit $t \xrightarrow[P]{} t'$. Somit ist noch kein einfaches Verfahren für die Berechung der Semantik eines Terms gegeben. Es kann auch nicht ein beliebiger Nachfolgerterm ausgewählt werden, da von einem Term, der eine Normalform besitzt, auch unendliche Reduktionsfolgen ausgehen können. Die naive Lösung der parallelen Weiterverfolgung aller Nachfolgerterme ist für eine Implementierung aufgrund der Ineffizienz nicht akzeptabel. Wünschenswert ist daher eine Reduktionsstrategie, die eine einfache, deterministische Berechnung ermöglicht.

Für operationelle Semantiken unserer Programme suchen wir somit Reduktionsstrategien, die (invariante) Grundtermsemantiken definieren.

Kapitel 4

Die Standardsemantiken

Nachdem wir nun die Syntax unserer Programme und die Grundlagen ihrer Semantik festgelegt haben, zeigen wir in 4.1 einige Beziehungen zwischen unseren Programmen und den wohlbekannten Kalkülen der Funktionsschemata, der Termersetzungssysteme und algebraischer Spezifikationen auf. Daraufhin betrachten wir die in funktionalen Programmiersprachen eingesetzten und insbesondere im Rahmen der Funktionsschemata untersuchten zwei Auswertungsmechanismen call-by-value (cbv) und call-by-name (cbn). Diese wollen wir als Grundlage (invarianter) Grundtermsemantiken verwenden.
Zuerst stellen wir in 4.2 die cbv-Semantik vor. In 4.2.1 definieren wir diese operationell durch eine Reduktionssemantik und in 4.2.2 auf denotationelle Weise. Die denotationelle Definition erfolgt mit einer auf ω-vollständigen Halbordnungen und dem Fixpunktsatz von Tarski beruhenden Fixpunktsemantik. In 4.2.3 zeigen wir dann die Übereinstimmung dieser beiden Definitionen.
Entsprechend definieren wir in 4.3 die cbn-Semantik. In 4.3.1 geben wir hierzu eine Fixpunktsemantik an und in 4.3.2 eine Reduktionssemantik. Den Beweis der Übereinstimmung der beiden Definitionen werden wir erst in Kapitel 5 führen.

4.1 Call-by-value und call-by-name

Funktionsschemata bilden die theoretische Grundlage funktionaler Programmiersprachen erster Ordnung. [Cou90] stellt hierzu eine Übersicht dar. Sie ähneln sehr unseren Programmen mit Hilfsfunktionen; insbesondere, da sie ebenfalls kein Patternmatching verwenden. Allerdings werden Funktionsschemata stets unabhängig von einem konkreten Basisdatentyp betrachtet. Dagegen sollen unsere Programme einen festen, aber doch sehr allgemeinen, auf Konstruktoren beruhenden Basisdatentyp besitzen. Hierzu gehören auch die unabhängig von einem konkreten Programm definierten Operationen der Hilfssymbole. Somit sind unsere Programm mit Hilfsfunktionen Funktionsschemata mit speziellen Basisdatentypen (Interpretationen in der Terminologie der Funktionsschemata). Wir können nun bekannte semantische Konzepte der Funktionsschemata für unsere Programme verwenden. Die insbesondere in [Gue81] betrachteten algebraischen Semantiken — nicht zu verwechseln mit unserer algebraischen Termsemantik — werden wir dabei nicht verwenden, da diese nur bei Betrachtung beliebiger Basisdatentypen vorteilhaft sind. Jedoch werden wir sowohl die bekannten, auf dem call-by-value und dem call-by-name Auswertungsmechanismus beruhenden Reduktionssemantiken, auf die wir gleich wieder zurückkommen werden, als auch die entsprechenden Fixpunktsemantiken auf Programme übertragen. Leider betrachten die Fixpunktsemantiken von Funktionsschemata meistens nur Basisdatentypen über flachen Halbordnungen, während wir schon in 4.3.1 andere ω-vollständige Halbordnungen verwenden müssen. Diese Semantiken und ihre

Übereinstimmung werden in [Manna74], [Vui74], [Vui74b], [Nivat75], [Dow&Se76], [Ber&Lévy77], [ADJ77], [Gue81], [Loe&Sie87], [Cou90], [Ind93] und [Ind94] untersucht.
Da unsere Programme mit Pattern aufgrund der Pattern auf den linken Regelseiten keine Funktionsschemata sind, greifen wir bei den Reduktionssemantiken, wir wir schon gesehen haben, hauptsächlich auf den Kalkül der Termersetzungssysteme zurück. Schon [O'Do77] verwendet Termersetzungssysteme als Grundlage operationeller Semantiken funktionaler Programmiersprachen.
Aus den beiden Blickwinkeln — der Funktionsschemata und der Termersetzungssysteme — ergeben sich widersprüchliche Einordnungen der Hilfsfunktionen der Programme mit Hilfsfunktionen. Aus Sicht der Funktionsschemata sind sie — da unabhängig vom konkreten Programm — Teil des Basisdatentyps und damit den Konstruktoren gleichgestellt. Andererseits existieren für sie Termersetzungsregeln genauso wie für die übrigen Funktionen, und beide werden bei der Reduktion gleich behandelt. Da wir der Einfachheit halber meistens Programme mit Hilfsfunktionen als spezielle Programme mit Pattern auffassen, neigen wir mehr dem letzteren Standpunkt zu. Wir wollen trotzdem nicht vergessen, daß Hilfsoperationen eine Sonderstellung zwischen den Konstruktoroperationen und den Funktionsoperationen einnehmen.
Schließlich wollen wir noch auf die Beziehung zu algebraischen Spezifikationen hinweisen, für die [Wir90] eine ausführliche Übersicht darstellt, und die in [Broy&Wir82], [Broy&Wir83], [Gut77], [Gut&Hor78] und [Thiel84] behandelt werden. Unsere Programme sind ausführbare Spezifikationen. Daher übernehmen wir von algebraischen Spezifikationen den Standpunkt, ein Programm als die Spezifikation eines Datentyps, einer Algebra aufzufassen. Der Kalkül der Funktionsschemata verwendet nämlich nur die Operation eines ausgezeichneten Funktionssymbols als Semantik, wie wir schon in der Einleitung erwähnten. Die hierarchische Spezifikation eines Datentyps auf der Grundlage eines anderen Basisdatentyps ist eines der Hauptanliegen der Theorie der algebraischen Spezifikationen. In Kapitel 6 werden wir unsere Semantiken noch einmal vom Standpunkt der algebraischen Spezifikationen betrachten und untersuchen.

Nach unseren Überlegungen zur Semantik unserer Programme in 3.3 sind wir jetzt auf der Suche nach (invarianten) Grundtermsemantiken für unsere Programme, insbesondere solchen, die auf (deterministischen) Reduktionsstrategien beruhen. Im Zusammenhang mit höheren Programmiersprachen fällt oft der Begriff des Auswertungsmechanismus. Dieser beschreibt, wie in einer höheren Programmiersprache die Parameterübergabe an Funktionen oder Prozeduren erfolgt. Für funktionale Programmiersprachen sind die beiden unterschiedlichen **Auswertungsmechanismen call-by-value** (cbv) und **call-by-name** (cbn) bekannt, und sie stehen in engem Zusammenhang mit bestimmten Reduktionsstrategien.
Der cbv-Auswertungsmechanismus wertet erst die Argumente einer Funktion vollständig aus, bevor die Funktion selber ausgeführt wird. Dies ist der Auswertungsmechanismus der meisten imperativen Programmiersprachen. Dagegen erfolgt bei dem cbn-Auswertungsmechanismus der Funktionsaufruf direkt; die unausgewerteten Argumente werden in den Funktionskörper hineinkopiert. Die Argumente werden erst ausgewertet, wenn sie wirklich benötigt werden. Das folgende Beispiel verdeutlicht die beiden Auswertungsmechanismen.

Beispiel 4.1 Call-by-value und call-by-name

$$\begin{array}{lcl} \texttt{liste1} & \rightarrow & \texttt{[]:liste1} \\ \texttt{head(x:xs)} & \rightarrow & \texttt{x} \end{array}$$

Call-by-value Auswertung:

$$\texttt{head}(\underline{\texttt{liste1}}) \xrightarrow[\texttt{P}]{} \texttt{head}([] : \underline{\texttt{liste1}}) \xrightarrow[\texttt{P}]{} \texttt{head}([] : [] : \underline{\texttt{liste1}}) \xrightarrow[\texttt{P}]{} \ldots$$

Call-by-name Auswertung:

$$\texttt{head}(\underline{\texttt{liste1}}) \xrightarrow[\texttt{P}]{} \underline{\texttt{head}}([] : \texttt{liste1}) \xrightarrow[\texttt{P}]{} []$$

□

Sowohl im Rahmen der Funktionsschemata ([Ind93], [Ind94]) als auch in dem des λ-Kalküls[1] ([Fie&Har88], Kapitel 6) werden die beiden Auswertungsmechanismen jeweils mit den Reduktionsstrategien leftmost-innermost*[2] und leftmost-outermost gleichgesetzt. Hierbei heißt ein Redex eines Terms genau dann **innermost***, wenn er keinen Redex als echten Teilterm enthält, und **outermost**, wenn er selbst kein echter Teilterm eines Redexes des Terms ist. Der **leftmost** Redex einer Menge von Redexen eines Terms ist dann der Redex, dessen Stelle die bezüglich der lexikographischen Ordnung kleinste Stelle all der Redexe der Menge ist. Bei der **leftmost-innermost* Reduktionsstrategie** wird in jedem Reduktionsschritt der leftmost-innermost* Redex reduziert, und bei der **leftmost-outermost Reduktionsstrategie** entsprechend der leftmost-outermost Redex. In obigem Beispiel erfolgt die call-by-value Auswertung mit leftmost-innermost Reduktion und die call-by-name Auswertung mit leftmost-outermost Reduktion.
Wir wollen nun auf Basis dieser beiden Auswertungsmechanismen zwei Semantiken für unsere Programme definieren.

4.2 Die cbv-Semantik

4.2.1 Die li-Reduktionssemantik

Da die cbv-Semantik im allgemeinen mit der leftmost-innermost Reduktionsstrategie assoziiert wird, definieren wir diese für unsere Programme.

Definition 4.1 **Leftmost-innermost (li-) Reduktion**
Sei $t \in \mathrm{T}_\Sigma$.

- Die Stelle $u \in \mathrm{Occ}(t)$ eines Redexes $\hat{l} = t/u \in \mathrm{Red}_P$ von t heißt genau dann **innermost Redexstelle des Terms t**, wenn
  - $\hat{l} = f(c_1, \ldots, c_n)$ mit $f^{(n)} \in \mathcal{F}$, $c_1, \ldots, c_n \in \mathrm{T}_\mathcal{C}$, oder
  - $\hat{l} = \mathrm{sel}_{G,i}(c)$ mit $\mathrm{sel}_{G,i} \in \mathcal{H}$, $c \in \mathrm{T}_\mathcal{C}$, oder
  - $\hat{l} = \mathrm{cond}_G(c, t_1, t_2)$ mit $\mathrm{cond}_G \in \mathcal{H}$, $c \in \mathrm{T}_\mathcal{C}$, $t_1, t_2 \in \mathrm{T}_\Sigma$ ist.
- Die bezüglich der lexikographischen Ordnung kleinste innermost Redexstelle von t heißt **leftmost-innermost Redexstelle des Terms t**.
- Eine Reduktion $A = t \xrightarrow[l \to r]{u} t'$ heißt genau dann **leftmost-innermostReduktion**, wenn u die li-Redexstelle von t ist. Hierdurch ist auch die **leftmost-innermost Reduktionsstrategie** $\xrightarrow[P,\mathbf{li}]{}$ definiert.

□

[1] Im λ-Kalkül wird allerdings normalerweise nur die Reduktion bis zur sogenannten weak head-normal form betrachtet.

[2] Wir markieren hier den Begriff innermost mit einem $*$, da wir im folgenden innermost etwas von dem in der Literatur verwendeten Begriff abweichend definieren werden. Wir gehen an entsprechender Stelle noch hierauf ein.

Da ein Term nur höchstens eine li-Redexstelle besitzt, ist $\xrightarrow[P,\mathrm{li}]{}$ wirklich eine (deterministische) Reduktionsstrategie. Wir definieren nun die li-Reduktionssemantik genauso wie die Normalformsemantik im letzten Kapitel, nur daß wir die li-Reduktionsstrategie anstelle von beliebigen Reduktionen mit $\xrightarrow[P]{}$ verwenden.

Definition 4.2 li-Reduktionssemantik
Die **li-Reduktionssemantik**

$$[\![\cdot]\!]_P^{\mathrm{li}} : \mathrm{T}_\Sigma \to \mathrm{T}_\mathcal{C}^\perp$$

ist für einen Term $t \in \mathrm{T}_\Sigma$ definiert durch

$$[\![t]\!]_P^{\mathrm{li}} := \begin{cases} t{\downarrow_{P,\mathrm{li}}} & \text{, falls } t{\downarrow_{P,\mathrm{li}}} \text{ existiert und } t{\downarrow_{P,\mathrm{li}}} \in \mathrm{T}_\mathcal{C} \text{ ist,} \\ \perp & \text{, andernfalls.} \end{cases}$$

□

Unsere Definition von (leftmost-)innermost Redexstellen unterscheidet sich von der üblichen, in 4.1 gegebenen Definition von (leftmost-)innermost* Redexen in zwei Punkten:

Nicht jede Stelle eines innermost* Redexes ist eine innermost Redexstelle.

Beispiel 4.2 Undefinierte Funktion
Das Programm P mit den Funktionssymbolen $\mathtt{f}^{(1)}$ und $\mathtt{a}^{(0)}$ bestehe allein aus der Programmregel

$$\mathtt{f(x)} \quad \to \quad \mathtt{A}$$

In dem Term `f(a)` ist `f(a)` selbst der (leftmost-)innermost* Redex, da `a` kein Redex ist. Mit der in 4.1 beschriebenen leftmost-innermost* Reduktionsstrategie ergibt sich

$$\mathtt{f(a)} \xrightarrow[P,\mathrm{li}^*]{} \mathtt{A}$$

Die Stelle ε in `f(a)` ist jedoch keine (leftmost-)innermost Redexstelle. `f(a)` besitzt überhaupt keine leftmost-innermost Redexstelle und ist somit schon in li-Normalform. Es gilt also

$$[\![\mathtt{f(a)}]\!]_\mathtt{P}^{\mathrm{li}} = \perp$$

□

Sobald ein Term ein Funktions- oder Hilfssymbol enthält, zu dem keine jemals passende Reduktionsregel existiert, ist der Term vermittels li-Reduktion nicht mehr zu einem Konstruktorterm reduzierbar. Die Funktions- und Hilfssymboloperationen werden in diesem Fall als undefiniert betrachtet, da schließlich das Programm keinerlei Information über sie enthält. Außerdem ist nur auf diese Weise die Invarianz der li-Reduktionssemantik erreichbar.

Umgekehrt befindet sich auch nicht an jeder innermost Redexstelle ein innermost* Redex. Die Verzweigungssymbole cond_G können im 2. und 3. Argument „geschachtelt“ sein, so daß die Bezeichnung „innermost“ eigentlich auch nicht mehr ganz passend ist. Dies ist jedoch nötig, um die Verzweigung überhaupt sinnvoll einsetzen zu können, wie das folgende Beispiel zeigt.

Beispiel 4.3 Verzweigung und innermost Redexstellen

$$\texttt{add(x,y)} \quad \rightarrow \quad \text{cond}_{\texttt{Succ}}\texttt{(y, Succ(add(x, sel}_{\texttt{Succ},1}\texttt{(y))), x)}$$

Bei Verwendung der leftmost-innermost* Reduktion ergibt sich

$$\begin{aligned}
\underline{\texttt{add}}\texttt{(Zero,Zero)} \quad & \xrightarrow[P,\text{li}^*]{} \quad \text{cond}_{\texttt{Succ}}\texttt{(Zero, Succ(}\underline{\texttt{add}}\texttt{(Zero, sel}_{\texttt{Succ},1}\texttt{(Zero))), Zero)} \\
& \xrightarrow[P,\text{li}^*]{} \quad \text{cond}_{\texttt{Succ}}\texttt{(Zero, Succ(} \\
& \qquad\quad \text{cond}_{\texttt{Succ}}\texttt{(sel}_{\texttt{Succ},1}\texttt{(Zero),} \\
& \qquad\quad \texttt{Succ(}\underline{\texttt{add}}\texttt{(Zero, sel}_{\texttt{Succ},1}\texttt{(sel}_{\texttt{Succ},1}\texttt{(Zero)))), Zero)} \\
& \qquad\quad \texttt{,Zero)} \\
& \xrightarrow[P,\text{li}^*]{} \quad \ldots
\end{aligned}$$

eine unendliche Reduktionsfolge. Demnach wäre

$$[\![\texttt{add(Zero, Zero)}]\!]_{\texttt{P}}^{\text{li}^*} = \bot.$$

Erst bei Verwendung der li-Reduktion erhält man mit

$$\underline{\texttt{add}}\texttt{(Zero, Zero)} \xrightarrow[P,\text{li}]{} \underline{\text{cond}}_{\texttt{Succ}}\texttt{(Zero, Succ(add(Zero, sel}_{\texttt{Succ},1}\texttt{(Zero))), Zero)} \xrightarrow[P,\text{li}]{} \texttt{Zero}$$

das gewünschte Ergebnis. □

Da in Funktionsschemata Funktionen immer vollständig spezifiziert sind — zu jedem Funktionssymbol existiert genau eine Regel — , kommt unsere erste Abweichung dort gar nicht zum Tragen. Weil in Funktionsschemata unsere Verzweigungen überlicherweise zum Basisdatentyp gehören würden, für den keine „normalen“ Reduktionsregeln existieren, fällt auch der zweite Punkt erst bei Betrachtung unserer Programme mit Hilfsfunktionen auf. Nur [Cou90] definiert auch eine Verzweigung `if-then-else-fi` mit fester Bedeutung und verwendet daher eine ähnliche Definition wie wir.

Bemerkung 4.1: Innermost Redexe und Redexstellen
Wir haben (leftmost-)innermost Redexstellen, jedoch keine (leftmost-)innermost Redexe eines Terms definiert. Wir haben auf die Definition der letzteren verzichtet, da sich der intendierte leftmost-innermost Redex gar nicht korrekt definieren läßt. Bezeichnen wir nämlich den an der leftmost-innermost Redexstelle in einem Term t befindlichen Redex $\hat{l}$ als leftmost-innermost Redex, so entsteht folgendes Problem: genau der gleiche Redex kann noch an anderen Stellen in t auftauchen. Dort soll er aber kein li-Redex sein.
In einem Term $t = \texttt{f}(\texttt{g}(\texttt{A}), \texttt{g}(\texttt{A}))$ soll beispielsweise der Teilterm $t/1 = \texttt{g}(\texttt{A})$ ein li-Redex sein, nicht jedoch der Teilterm $t/2$.
Das gleiche Problem taucht auch beim outermost Redex auf, der an einer anderen Stelle auch Teilterm eines Redexes sein kann; ebenso beim lo-Redex. Daher werden wir in 4.3.2 auch nur (leftmost-)outermost Redexstellen definieren.
Nur ein innermost Redex ließe sich tatsächlich sinnvoll definieren, da diese Eigenschaft nur von dem Redex selber und nicht von seiner Stellung in einem Term abhängt. □

Wir wollen uns nun noch einmal das Programm aus Beispiel 3.6, S. 45, anschauen, dessentwegen wir die Normalformsemantik zurückgewiesen haben.

Beispiel 4.4 Invarianz der li-Reduktionssemantik

$$\begin{array}{lcl} \texttt{liste1} & \to & \texttt{[]:liste1} \\ \texttt{liste2} & \to & \texttt{[[]]:liste2} \\ \texttt{head(x:xs)} & \to & \texttt{x} \end{array}$$

`liste1` und `liste2` besitzen keine li-Normalform und somit ist

$$[\![\texttt{liste1}]\!]^{\text{li}}_{\text{P}} = \bot = [\![\texttt{liste2}]\!]^{\text{li}}_{\text{P}}$$

Wegen

$$\begin{array}{lllllll} \texttt{head(}\underline{\texttt{liste1}}\texttt{)} & \xrightarrow[P,\text{li}]{} & \texttt{head([]:}\underline{\texttt{liste1}}\texttt{)} & \xrightarrow[P,\text{li}]{} & \texttt{head([]:[]:}\underline{\texttt{liste1}}\texttt{)} & \xrightarrow[P,\text{li}]{} & \ldots \\ \texttt{head(}\underline{\texttt{liste2}}\texttt{)} & \xrightarrow[P,\text{li}]{} & \texttt{head([[]]:}\underline{\texttt{liste2}}\texttt{)} & \xrightarrow[P,\text{li}]{} & \texttt{head([[]]:[[]]:}\underline{\texttt{liste2}}\texttt{)} & \xrightarrow[P,\text{li}]{} & \ldots \end{array}$$

ist aber auch

$$[\![\texttt{head(liste1)}]\!]^{\text{li}}_P = \bot = [\![\texttt{head(liste2)}]\!]^{\text{li}}_P.$$

□

Hier ist das Problem also verschwunden, und in der Tat ist die li-Reduktionssemantik invariant.

Lemma 4.1 Invarianz der li-Reduktionssemantik
Die li-Reduktionssemantik $[\![\cdot]\!]^{\text{li}}_P$ ist invariant. Sie ist also eine Grundtermsemantik.

Beweis:
Wir zeigen die Invarianz anhand ihrer in Lemma 3.3, S. 46, angegebenen Charakterisierung.
Sei $f^{(n)} \in \Sigma$ und $t_1, \ldots, t_n, t'_1, \ldots, t'_n \in \mathrm{T}_\Sigma$ mit $[\![t_1]\!]^{\text{li}}_P = [\![t'_1]\!]^{\text{li}}_P, \ldots, [\![t_n]\!]^{\text{li}}_P = [\![t'_n]\!]^{\text{li}}_P$.

Fall 1: $f \neq \text{cond}_G$ oder $[\![t_1]\!]^{\text{li}}_P = [\![t'_1]\!]^{\text{li}}_P = \bot$.

Fall 1.1: Es existiert $i \in [n]$ mit $[\![t_1]\!]^{\text{li}}_P \neq \bot, \ldots, [\![t_{i-1}]\!]^{\text{li}}_P \neq \bot, [\![t_i]\!]^{\text{li}}_P = \bot$.
Somit besitzen weder t_i noch t'_i eine li-Konstruktornormalform (d. h. eine li-Normalform in $\mathrm{T}_\mathcal{C}$).

$$\begin{array}{lll} f(t_1, \ldots, t_i, \ldots, t_n) & \xrightarrow[P,\text{li}]{*} & f(t_1{\downarrow_{P,\text{li}}}, \ldots, t_{i-1}{\downarrow_{P,\text{li}}}, t_i, \ldots, t_n) \\ f(t'_1, \ldots, t'_i, \ldots, t'_n) & \xrightarrow[P,\text{li}]{*} & f(t'_1{\downarrow_{P,\text{li}}}, \ldots, t'_{i-1}{\downarrow_{P,\text{li}}}, t'_i, \ldots, t'_n) \end{array}$$

Auch $f(t_1{\downarrow_{P,\text{li}}}, \ldots, t_{i-1}{\downarrow_{P,\text{li}}}, t_i, \ldots, t_n)$ und $f(t'_1{\downarrow_{P,\text{li}}}, \ldots, t'_{i-1}{\downarrow_{P,\text{li}}}, t'_i, \ldots, t'_n)$ besitzen keine li-Konstruktornormalform. Damit ist $[\![f(t_1, \ldots, t_i, \ldots, t_n)]\!]^{\text{li}}_P = \bot = [\![f(t'_1, \ldots, t'_i, \ldots, t'_n)]\!]^{\text{li}}_P$.

Fall 1.2: $[\![t_1]\!]^{\text{li}}_P \neq \bot, \ldots, [\![t_n]\!]^{\text{li}}_P \neq \bot$.

$$\begin{array}{lll} f(t_1, \ldots, t_n) & \xrightarrow[P,\text{li}]{*} & f(t_1{\downarrow_{P,\text{li}}}, \ldots, t_n{\downarrow_{P,\text{li}}}) \\ f(t'_1, \ldots, t'_n) & \xrightarrow[P,\text{li}]{*} & f(t'_1{\downarrow_{P,\text{li}}}, \ldots, t'_n{\downarrow_{P,\text{li}}}) \end{array}$$

Also ist

$$[\![f(t_1, \ldots, t_n)]\!]^{\text{li}}_P = [\![f(t_1{\downarrow_{P,\text{li}}}, \ldots, t_n{\downarrow_{P,\text{li}}})]\!]^{\text{li}}_P = [\![f(t'_1{\downarrow_{P,\text{li}}}, \ldots, t'_n{\downarrow_{P,\text{li}}})]\!]^{\text{li}}_P = [\![f(t'_1, \ldots, t'_n)]\!]^{\text{li}}_P.$$

Fall 2: $f = \mathrm{cond}_G \in \mathcal{H}$ und $[\![t_1]\!]^{\mathrm{li}}_P = [\![t'_1]\!]^{\mathrm{li}}_P \neq \bot$.

O. B. d. A. sei $(t_1 \downarrow_{P,\mathrm{li}})(\varepsilon) = G$.

$$\begin{array}{lllll} \mathrm{cond}_G(t_1,t_2,t_3) & \xrightarrow[P,\mathrm{li}]{*} & \mathrm{cond}_G(t_1\downarrow_{P,\mathrm{li}},t_2,t_3) & \xrightarrow[P,\mathrm{li}]{*} & t_2 \\ \mathrm{cond}_G(t'_1,t'_2,t'_3) & \xrightarrow[P,\mathrm{li}]{*} & \mathrm{cond}_G(t'_1\downarrow_{P,\mathrm{li}},t'_2,t'_3) & \xrightarrow[P,\mathrm{li}]{*} & t'_2 \end{array}$$

Somit ist $[\![\mathrm{cond}_G(t_1,t_2,t_3)]\!]^{\mathrm{li}}_P = [\![t_2]\!]^{\mathrm{li}}_P = [\![t'_2]\!]^{\mathrm{li}}_P = [\![\mathrm{cond}_G(t'_1,t'_2,t'_3)]\!]^{\mathrm{li}}_P$.

□

Die li-Reduktionsstrategie ist nicht normalisierend. Für die li*-Reduktion gilt sogar, daß die mit einem Term t beginnende li*-Reduktionsfolge nur genau dann terminiert, wenn von diesem Term überhaupt keine unendliche Reduktionsfolge ausgeht (Theorem 11 und 16 in [O'Do77]). Daher wurde sie in der Literatur oft als unvollständig bezeichnet ([Vui74], [Dow&Se76], [Ber&Lévy77]). Zum ersten Mal gibt [de Bakker76] im Rahmen des λ-Kalküls eine denotationelle Fixpunktsemantik für sie an.

4.2.2 Die Fixpunktsemantik

Wir geben nun eine denotationelle Definition der cbv-Semantik an. Dafür ordnen wir einem Programm einen Datentyp, eine Algebra als Semantik zu.
Zuerst betrachten wir den Basisdatentyp, da dieser unabhängig von einem konkreten Programm ist, und wir diesen daher ein für allemal festlegen können. Die Datenelemente sollen aus Konstruktoren aufgebaut sein. Der Träger besteht somit aus den Konstruktortermen $\mathrm{T}_{\mathcal{C}}$ und dem Element $\bot$ für die Undefiniertheit, ist also der schon bei der li-Reduktionssemantik verwendete Rechenbereich. Dies ist natürlich auch wesentlich, da der Datentyp ein Datentyp der li-Reduktionssemantik sein soll. Die Konstruktoren werden wie in einer Herbrandalgebra durch sich selbst interpretiert. Schließlich gehören noch die Operationen der Funktionssymbole, die zu deren Reduktionsregeln passen müssen, zum Basisdatentyp.
Wir verzichten hier jedoch auf die formale Definition des Basisdatentyps und nähern uns stattdessen dem festzulegenden Datentyp des Programms auf etwas andere Weise. Wir definieren die Menge der Interpretationen: die Menge der Σ-Algebren, die, ohne daß wir Informationen über das konkrete Programm benutzen, potentielle Datentypen des Programms sind. Damit besteht eine Interpretation im Prinzip aus dem Basisdatentyp plus den Operationen der Funktionssymbole, wobei letztere beliebig sein können. Wie wir schon erwähnten, weichen wir bezüglich der Hilfsoperationen vom Konzept des Basisdatentyps etwas ab. Die Operationen der Hilfssymbole lassen sich aus den Reduktionsregeln auf genau die gleiche Art gewinnen wie die Operationen der Funktionssymbole, weshalb wir die Hilfsoperationen in den Interpretationen ebenfalls noch nicht festlegen.

Definition 4.3 cbv-Interpretation
Sei $\langle \mathrm{T}^{\bot}_{\mathcal{C}}, \trianglelefteq \rangle$ die flache Halbordnung des Rechenbereichs mit $\bot$ als kleinstem Element. Sei

$$\alpha : \bigcup_{n\in\mathbb{N}} \Sigma_n \to [(\mathrm{T}^{\bot}_{\mathcal{C}})^n \to \mathrm{T}^{\bot}_{\mathcal{C}}]$$

eine Zuordnung von ω-stetigen Operationen an die Operationssymbole mit

$$\begin{array}{rcl} \alpha(G)(\underline{t}_1,\ldots,\underline{t}_n) & = & \begin{cases} G(\underline{t}_1,\ldots,\underline{t}_n) & \text{, falls } \underline{t}_1,\ldots,\underline{t}_n \in \mathrm{T}_{\mathcal{C}} \\ \bot & \text{, andernfalls} \end{cases} \\ \alpha(f)(\underline{t}_1,\ldots,\underline{t}_n) & = & \bot \text{ , wenn } \bot \in \{\underline{t}_1,\ldots,\underline{t}_n\} \\ \alpha(\mathrm{cond}_G)(\bot,\underline{t}_1,\underline{t}_2) & = & \alpha(\mathrm{sel}_{G,i})(\bot) \;=\; \bot \end{array}$$

für alle $G^{(n)} \in \mathcal{C}, f^{(n)} \in \mathcal{F}, \mathrm{cond}_G, \mathrm{sel}_{G,i} \in \mathcal{H}$ und $\underline{t}_1, \underline{t}_2, \ldots \in \mathrm{T}_{\mathcal{C}}^{\perp}$.
Die ω-vollständige Σ-Algebra $\mathfrak{A} = \langle \mathrm{T}_{\mathcal{C}}^{\perp}, \trianglelefteq, \alpha \rangle$ heißt **cbv-Interpretation**. Die Menge aller cbv-Interpretationen zu einer festen Programmsignatur Σ wird mit $\mathrm{Int}_{\Sigma,\mathrm{cbv}}$ bezeichnet.
Mit der kanonischen Halbordnungsrelation $\sqsubseteq$ der Σ-Algebren ist $\langle \mathrm{Int}_{\Sigma,\mathrm{cbv}}, \sqsubseteq \rangle$ die kanonische Halbordnung der cbv-Interpretationen. In der kleinsten cbv-Interpretation $\perp_{\mathrm{cbv}} := \perp_{\mathrm{Int}_{\Sigma,\mathrm{cbv}}} \in \mathrm{Int}_{\Sigma,\mathrm{cbv}}$ sind die Ergebniswerte aller Funktions- und Hilfsoperationen konstant $\perp$. □

In der Definition der cbv-Interpretationen fällt auf, daß alle Operationen — abgesehen von den Verzweigungsoperationen — als strikt definiert werden. Genau an diesem Punkt geht der cbv-Auswertungsmechanismus ein, und dies ist der entscheidende Unterschied zur cbn-Interpretation, die wir in 4.3.1 definieren werden. Da die Argumente einer Funktion vor der Ausführung der Funktion vollständig ausgerechnet werden, kann der gesamte Funktionsausdruck nicht terminieren, wenn die Berechnung nur eines Arguments nicht terminiert. Alle Funktionen sind somit strikt; nur die Verzweigungsoperationen stellen analog zu und aus den gleichen Gründen wie bei der Reduktionssemantik eine Ausnahme dar.
Die Festlegung der Striktheit der Funktions- und Hilfsoperationen schon in der cbv-Interpretation ist eigentlich nicht nötig, da sie sich im folgenden auch automatisch ergibt. Wir haben sie nur der Symmetrie zu den Konstruktoroperationen halber und, da sich dies schon aus dem cbv-Auswertungsmechanismus unabhängig von einem Programm ergibt, hier festgelegt.
Wir definieren eine cbv-Interpretation als geordnete, ω-vollständige Σ-Algebra und verwenden dabei die flache Halbordnung mit $\perp$ als kleinstem Element. Die Verwendung ω-vollständiger Halbordnungen und darauf ω-stetiger Abbildungen dient letztendlich allein dem Zweck, in Definition 4.5 einem Programm genau eine ausgezeichnete cbv-Interpretation als Datentyp zuordnen zu können. Trotzdem kann der Halbordnung eine intuitive Bedeutung zugemessen werden. Die Halbordnung $\langle \mathrm{T}_{\mathcal{C}}^{\perp}, \trianglelefteq \rangle$ ist eine Ordnung bezüglich des Informationsgehalts. $\perp$ als kleinstes Element steht für das völlige Fehlen von Information. Alle anderen Elemente $t \in \mathrm{T}_{\mathcal{C}}$ sind größer als $\perp$, da sie Information repräsentieren. Sie stellen alle verschiedene Information dar und sind daher nicht miteinander vergleichbar. Aufgrund dieser Nicht-Vergleichbarkeit stellt die geforderte ω-Stetigkeit der Operationen auch keine Einschränkung der Operationen über dieser Menge $\mathrm{T}_{\mathcal{C}}$ dar. Die Menge der durch Programme beschreibaren Operationen ist also hierdurch diesbezüglich nicht beschränkt. Andererseits ist die geforderte Monotonie der Operationen über $\mathrm{T}_{\mathcal{C}}^{\perp}$ auch einleuchtend: Besitzt ein Element b mehr Information als ein Element a, so erwarten wir von einer Operation φ, daß $\varphi(b)$ mehr Information besitzt als $\varphi(a)$. In nicht-flachen Halbordnungen, wie wir sie ab 4.3.1 betrachten werden, wird die Abstufung bezüglich des Informationsgehalts noch deutlicher.
Aus der ω-Vollständigkeit der cbv-Interpretationen folgt die ω-Vollständigkeit der Halbordnung $\langle \mathrm{Int}_{\Sigma,\mathrm{cbv}}, \sqsubseteq \rangle$ der cbv-Interpretationen. Wir wollen dies hier nicht beweisen, wie wir überhaupt in diesem Kapitel auf viele Beweise, insbesondere der Wohldefiniertheit der Fixpunktsemantiken, verzichten. Hier sollen hauptsächlich die Prinzipien und Konzepte vorgestellt werden. Alle Beweise werden in Kapitel 5 in einem abstrakteren, allgemeineren Rahmen durchgeführt.
Wir haben hier zum ersten Mal folgende Konvention verwendet: Elemente des Rechenbereichs ($\mathrm{T}_{\mathcal{C}}^{\perp}$) werden mit unterstrichenen Kleinbuchstaben t bezeichnet. Die dadurch auch erfolgte Trennung von syntaktischen Termen (T_{Σ}) und semantischen Rechentermen ($\mathrm{T}_{\mathcal{C}}^{\perp}$) ist jedoch mit Vorsicht zu genießen. Die beiden Mengen besitzen gemeinsame Elemente ($\mathrm{T}_{\Sigma} \cap \mathrm{T}_{\mathcal{C}}^{\perp} = \mathrm{T}_{\mathcal{C}}$), und in einer Reduktionssemantik ist der Übergang zwischen beiden praktisch fließend. Es erweist sich außerdem meistens als unpraktisch, für das gleiche Element $t = \underline{t}$ sowohl einen syntaktischen (t) als auch einen semantischen ($\underline{t}$) Bezeichner zu verwenden.

Nun definieren wir durch die Reduktionsregeln eines Programms dessen Datentyp. Die Operatio-

nen der auf den linken Reduktionsregelseiten befindlichen Funktions- und Hilfssymbole werden durch ihre jeweiligen rechten Regelseiten spezifiziert. Zwar ergibt sich daraus noch keine direkte Definition der Operationen, da die Funktions- und Hilfssymbole im allgemeinen auch auf den rechten Regelseiten auftauchen, aber das gesamte Programm ist als eine Abbildung auffaßbar: Ist eine cbv-Interpretation zur Interpretation der Symbole der rechten Regelseiten gegeben, so werden durch das Programm die Operationen der Symbole der linken Regelseiten und somit eine neue cbv-Interpretation bestimmt.

Definition 4.4 cbv-Transformation
Die **cbv-Transformation** des Programms P über Σ,

$$\Phi_{P,\mathrm{cbv}} : [\mathrm{Int}_{\Sigma,\mathrm{cbv}} \to \mathrm{Int}_{\Sigma,\mathrm{cbv}}],$$

ist definiert durch

$$f^{\Phi_{P,\mathrm{cbv}}(\mathfrak{A})}(\vec{\underline{t}}) := \begin{cases} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} & , \text{ falls eine Programmregel } f(\vec{p}) \to r \in P \\ & \quad \text{und eine Variablenbelegung } \beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C}} \\ & \quad \text{mit } [\![p_1]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}},\beta} = \underline{t}_1 \ (\neq \perp), \ldots, [\![p_n]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}},\beta} = \underline{t}_n \ (\neq \perp) \text{ existiert} \\ \perp & , \text{ andernfalls} \end{cases}$$

für alle $f^{(n)} \in \mathcal{F}$, $\vec{\underline{t}} \in (\mathrm{T}^{\perp}_{\mathcal{C}})^n$ und $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\mathrm{cbv}}$,
und außerdem, wenn $\Sigma = (\mathcal{C}, \mathcal{H}, \mathcal{F})$,

$$\mathrm{cond}_G^{\Phi_{P,\mathrm{cbv}}(\mathfrak{A})}(\underline{t}_1, \underline{t}_2, \underline{t}_3) := \begin{cases} \perp & , \text{ falls } \underline{t}_1 = \perp \\ \underline{t}_2 & , \text{ falls eine Variablenbelegung } \beta : \{x_1, \ldots, x_n\} \to \mathrm{T}_{\mathcal{C}} \\ & \quad \text{mit } [\![G(x_1, \ldots, x_n)]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}},\beta} = \underline{t}_1 \ (\neq \perp) \text{ existiert} \\ \underline{t}_3 & , \text{ andernfalls} \end{cases}$$

$$\mathrm{sel}_{G,i}^{\Phi_{P,\mathrm{cbv}}(\mathfrak{A})}(\underline{t}) := \begin{cases} \beta(x_i) & , \text{ falls eine Variablenbelegung } \beta : \{x_1, \ldots, x_n\} \to \mathrm{T}_{\mathcal{C}} \\ & \quad \text{mit } [\![G(x_1, \ldots, x_n)]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}},\beta} = \underline{t} \ (\neq \perp) \text{ existiert} \\ \perp & , \text{ andernfalls} \end{cases}$$

für alle $\mathrm{cond}_G, \mathrm{sel}_{G,i} \in \mathcal{H}$, $\underline{t}, \underline{t}_1, \underline{t}_2, \underline{t}_3 \in \mathrm{T}^{\perp}_{\mathcal{C}}$ und $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\mathrm{cbv}}$. □

Die zu einem konkreten Operationsargument gehörende rechte Regelseite wird durch ein semantisches Patternmatching ausgewählt. Hierbei ist zu beachten, daß die Definition von der Wahl der in der algebraischen Termsemantik beim Patternmatching ($[\![p_i]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}},\beta} = \underline{t}_i$) verwendeten cbv-Interpretation ($\perp_{\mathrm{cbv}}$) unabhängig ist. Da $p_i \in \mathrm{T}_{\mathcal{C}}$ ist, und die Konstruktoroperationen in allen cbv-Interpretationen gleich sind, kann dort eine beliebige cbv-Interpretation verwendet werden. Es wird hier $\perp_{\mathrm{cbv}}$ gewählt, da die Verwendung der einzigen anderen ausgezeichneten cbv-Interpretation $\mathfrak{A}$ den falschen Eindruck einer (nicht vorhandenen) Abhängigkeit schaffen würde.
Wie angekündigt, werden alle Operationen als strikt definiert; nur die Verzweigungsoperationen können an der 2. und 3. Argumentstelle nicht-strikt sein. Deshalb wird die Definition der Hilfsoperationen getrennt aufgeführt. Die Selektionsoperationen werden allerdings genauso wie die Funktionsoperationen behandelt.
Die Eindeutigkeit der Definition, die ω-Stetigkeit der Funktions- und Hilfsoperationen und die ω-Stetigkeit der cbv-Transformation überhaupt werden wie erwähnt erst in Kapitel 5 bewiesen.

BEMERKUNG 4.2: Algebraische Termsemantik und Substitution
Die cbv-Transformation ließe sich auch äquivalent mit Applikationen einer Substitution anstelle der algebraischen Termsemantiken mit Variablenbelegung definieren:

$$f^{\Phi_{P,\mathrm{cbv}}(\mathfrak{A})}(\vec{\underline{t}}) := \begin{cases} [\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}} & , \text{ falls eine Programmregel } f(\vec{p})\to r \in P \\ & \quad \text{und eine Substitution } \sigma : \mathrm{Var}(\vec{p})\to \mathrm{T}_{\mathcal{C}} \\ & \quad \text{mit } p_1\sigma = \underline{t}_1, \ldots, p_n\sigma = \underline{t}_n \text{ existiert} \\ \bot & , \text{ andernfalls} \end{cases}$$

(entsprechend für die Hilfsoperationen).
Für eine Variablenbelegung $\beta : \mathrm{Var}(\vec{p})\to\mathrm{T}_{\mathcal{C}}$ und eine Substitution $\sigma : \mathrm{Var}(\vec{p})\to\mathrm{T}_{\mathcal{C}}$ mit $\beta(x) = x\sigma$ für alle $x \in \mathrm{Var}(\vec{p})$ gilt gemäß des nachfolgenden Lemmas 4.2 über Substitutionsapplikation und algebraische Termsemantik $[\![c]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![c\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}$ für alle $c \in \mathrm{T}_{\mathcal{C}}$ und $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\mathrm{cbv}}$. Demnach ist dann insbesondere

$$[\![p_i]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbv}},\beta} = [\![p_i\sigma]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbv}}} = p_i\sigma$$

für alle $i \in [n]$ und

$$[\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}.$$

Somit sind die beiden Definitionen wirklich äquivalent.
Die Verwendung der Substitution hat den Vorteil der größeren Nähe zur Reduktionssemantik. Daher werden wir diese Variante auch in 4.2.3 beim Beweis der Übereinstimmung von Fixpunkt- und Reduktionssemantik verwenden.
Eine Applikation einer Substitution ist im Prinzip eine spezielle algebraische Termsemantik, weshalb auch die obige Äquivalenz gilt. Für die cbn-Transformation und für die in Kapitel 5 definierte ς-Transformation können wir leider keine Substitution mehr verwenden. Wir müssen die allgemeineren algebraischen Termsemantiken einsetzen. Daher erfolgt auch hier schon die Definition der cbv-Transformation mit algebraischen Termsemantiken. □

Lemma 4.2 Substitutionsapplikation und algebraische Termsemantik
Sei Σ eine Signatur, $\mathfrak{A} = \langle A, \alpha\rangle \in \mathrm{Alg}_{\Sigma}$, $\sigma : X\to\mathrm{T}_{\Sigma}(X)$ eine Substitution und $\beta, \beta' : X\to A$ Variablenbelegungen.
Es gelte

$$\beta(x) = [\![x\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta'}$$

für alle $x \in X$. Dann ist

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![t\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta'}$$

für alle $t \in \mathrm{T}_{\Sigma}(X)$.

Beweis:

$t = x \in X$:

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = \beta(x) \overset{\mathrm{Vor.}}{=} [\![x\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta'} = [\![t\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta'}.$$

$t = g(t_1, \ldots, t_n)$: $(g^{(n)} \in \Sigma)$.

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = g^{\mathfrak{A}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}) \overset{\mathrm{I.V.}}{=} g^{\mathfrak{A}}([\![t_1\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta'}, \ldots, [\![t_n\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta'}) = [\![t\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta'}.$$

□

Für den durch ein Programm spezifizierten Datentyp $\mathcal{D} \in \mathrm{Int}_{\Sigma,\mathrm{cbv}}$ muß natürlich $\mathcal{D} = \Phi_{P,\mathrm{cbv}}(\mathcal{D})$ gelten, d. h. $\mathcal{D}$ muß ein Fixpunkt der Transformation sein. Die Transformationen der meisten Programme besitzen allerdings viele Fixpunkte:

```
undef  →  undef
```

Da dieses Programm über den semantischen Wert von `undef` überhaupt nichts aussagt, ist es sinnvoll, `undef` den semantischen Wert für fehlende Information oder Undefiniertheit, $\perp$, zuzuordnen. Entsprechend wählen wir von allen Fixpunkten der Transformation den kleinsten als Datentyp des Programms. Dieser kleinste Fixpunkt ist praktisch der kleinste gemeinsame Nenner aller Fixpunkte bezüglich des Informationsgehalts und besitzt somit nur die Eigenschaften, die durch das Programm eindeutig spezifiziert werden.

Definition 4.5 cbv-Fixpunktdatentyp und cbv-Fixpunktsemantik
Der **cbv-Fixpunktdatentyp** $\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbv}}$ des Programms P ist definiert als der kleinste Fixpunkt der cbv-Transformation $\Phi_{P,\mathrm{cbv}}$:

$$\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbv}} := \mathrm{Fix}(\Phi_{P,\mathrm{cbv}}) = \bigsqcup_{i\in\mathbb{N}} (\Phi_{P,\mathrm{cbv}})^i(\perp_{\mathrm{cbv}}).$$

Die **cbv-Fixpunktsemantik** $[\![t]\!]^{\mathrm{fix}}_{P,\mathrm{cbv}}$ des Grundterms $t \in \mathrm{T}_\Sigma$ bezüglich P ist definiert als die algebraische Grundtermsemantik von t bezüglich des cbv-Fixpunktdatentyps:

$$[\![t]\!]^{\mathrm{fix}}_{P,\mathrm{cbv}} := [\![t]\!]^{\mathrm{alg}}_{\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbv}}}.$$

□

Hier wird die ω-Vollständigkeit der cbv-Interpretation benötigt. Aus ihr folgt die ω-Stetigkeit der cbv-Transformation (siehe Kapitel 5) und mit dem Fixpunktsatz von Tarski wissen wir, daß jede cbv-Transformation einen kleinsten Fixpunkt besitzt. Somit ordnet unsere Fixpunktsemantik wirklich jedem syntaktisch korrekten Programm genau eine Bedeutung, einen Datentyp zu.
Der Fixpunktsatz gibt uns sogar gewissermaßen eine Methode zur Bestimmung des cbv-Fixpunktdatentyps. Beginnend mit der kleinsten Interpretation $\perp_{\mathrm{cbv}}$ können wir durch fortwährende Anwendung der Transformation unseren Datentyp erreichen. Freilich stellt dies kein effektives Verfahren zur Berechnung des Datentyps dar, da erstens die Iteration im allgemeinen bis ∞ geht, und wir zweitens Interpretationen $\mathfrak{A}$ nicht direkt endlich darstellen können. Interpretationen lassen sich durch Programme darstellen, womit wir uns aber im Kreis drehen. Somit ist die Fixpunktsemantik wirklich keine operationelle Semantik, sondern eine denotationelle. Ein Programm denotiert einen Datentyp.
Diese Methode der iterativen Anwendung der Transformation bildet übrigens die Grundlage der **abstrakten Interpretation** (Kapitel 20 in [Fie&Har88]). Damit lassen sich gewisse semantische Eigenschaften von Programmen — insbesondere für Optimierungszwecke — feststellen.

Beispiel 4.5 Bestimmung eines cbv-Fixpunktdatentyps (siehe Bsp. 3.6, S. 45)

```
liste1      →  []:liste1
liste2      →  [[]]:liste2
head(x:xs)  →  x
```

Sei $\mathfrak{A}_i := (\Phi_{P,\mathrm{cbv}})^i(\bot_{\mathrm{cbv}})$ für alle $i \in \mathbb{N}$.

	$i=0$	$i=1$	$i=2$	$\ldots$	$i=\infty$ $(\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbv}} = \mathfrak{A}_i)$
$\mathtt{liste1}^{\mathfrak{A}_i}()$	$\bot$	$[\![[] : \mathtt{liste1}]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbv}},(\mathtt{liste1}\mapsto\bot)} = \bot$	wie $i=1$	$\ldots$	wie $i=1$
$\mathtt{liste2}^{\mathfrak{A}_i}()$	$\bot$	$[\![[[]] : \mathtt{liste2}]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbv}},(\mathtt{liste2}\mapsto\bot)} = \bot$	wie $i=1$	$\ldots$	wie $i=1$
$\mathtt{head}^{\mathfrak{A}_i}$	$\underline{t} \mapsto \bot$	$\begin{pmatrix} \bot & \mapsto & \bot \\ [] & \mapsto & \bot \\ \underline{t}_1{:}\underline{t}_2 & \mapsto & \underline{t}_1 \end{pmatrix}$	wie $i=1$	$\ldots$	wie $i=1$

mit $\underline{t} \in \mathrm{T}^{\bot}_{\mathcal{C}}$, $\underline{t}_1, \underline{t}_2 \in \mathrm{T}_{\mathcal{C}}$. □

Wir geben hier auch noch kurz die Hilfsoperationen an, da diese wie gesagt unabhängig von einem konkreten Programm sind und zum Basisdatentyp gehören.

Korollar 4.3 Hilfsoperationen des cbv-Fixpunktdatentyps

$$\mathrm{cond}_G^{\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbv}}}(\underline{t}_1,\underline{t}_2,\underline{t}_3) = \begin{cases} \bot & , \text{ falls } \underline{t}_1 = \bot \\ \underline{t}_2 & , \text{ falls eine Variablenbelegung } \beta : \{x_1,\ldots,x_n\} \to \mathrm{T}_{\mathcal{C}} \\ & \quad \text{mit } [\![G(x_1,\ldots,x_n)]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbv}},\beta} = \underline{t}_1 \ (\neq \bot) \text{ existiert} \\ \underline{t}_3 & , \text{ andernfalls} \end{cases}$$

$$\mathrm{sel}_{G,i}^{\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbv}}}(\vec{\underline{t}}) = \begin{cases} \beta(x_i) & , \text{ falls eine Variablenbelegung } \beta : \{x_1,\ldots,x_n\} \to \mathrm{T}_{\mathcal{C}} \\ & \quad \text{mit } [\![G(x_1,\ldots,x_n)]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbv}},\beta} = \underline{t} \ (\neq \bot) \text{ existiert} \\ \bot & , \text{ andernfalls} \end{cases}$$

für alle $\mathrm{cond}_G, \mathrm{sel}_{G,i} \in \mathcal{H}$ und $\underline{t}, \underline{t}_1, \underline{t}_2, \underline{t}_3 \in \mathrm{T}^{\bot}_{\mathcal{C}}$.

Beweis:
Ergibt sich direkt aus der cbv-Transformation. □

Es stellt sich nun die Frage, ob diese cbv-Fixpunktsemantik gleich der li-Reduktionssemantik ist, oder ob wir wohlmöglich zwei verschiedene statt einer cbv-Semantik definiert haben.

4.2.3 Übereinstimmung der Reduktions- und der Fixpunktsemantik

Obwohl die Definitionen der li-Reduktions- und der cbv-Fixpunktsemantik viele Ähnlichkeiten aufweisen, ist ein direkter Beweis der Gleichheit der beiden Grundtermsemantiken $[\![\cdot]\!]^{\mathrm{li}}_P$ und $[\![\cdot]\!]^{\mathrm{fix}}_{P,\mathrm{cbv}}$ ziemlich schwierig. Eine strukturelle Induktion über den Termaufbau ist nicht möglich. Eine Induktion über die Anzahl der li-Reduktionsschritte bis zur Normalform läßt uns noch das nicht-triviale Problem übrig, zu zeigen, daß für alle Terme $t \in \mathrm{T}_\Sigma$, die keine li-(Konstruktor)normalform besitzen $[\![t]\!]^{\mathrm{fix}}_{P,\mathrm{cbv}} = \bot$ gilt. Auch zu zeigen, daß $\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbv}}$ ein Datentyp der Grundtermsemantik $[\![\cdot]\!]^{\mathrm{li}}_P$ ist, erweist sich als ähnlich schwierig.
Wir verwenden daher hier eine ganz andere Beweismethode. Wir definieren zu der li-Reduktionssemantik $[\![\cdot]\!]^{\mathrm{li}}_P$ einen Datentyp $\mathcal{D}^{\mathrm{li}}_P$, und zeigen dann, daß dieser gleich dem cbv-Fixpunktdatentyp $\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbv}}$ ist.
Wir haben allerdings schon festgestellt, daß es im allgemeinen mehrere Datentypen einer Grundtermsemantik gibt. Das Problem liegt darin, daß zu einem (semantischen) Rechenterm, d. h. einem Element des Trägers, eventuell kein syntaktischer Term existiert, der den Rechenterm denotiert. Diese Möglichkeit existiert auch bei der li-Reduktionssemantik, nämlich, wenn es keinen „undefinierten“ Term gibt, d. h. kein $t_\bot \in \mathrm{T}_\Sigma$ mit $[\![t_\bot]\!]^{\mathrm{li}}_P = \bot$. In diesem Fall müssen wir, um genau einen Datentyp zu erhalten, die Definition des Datentyps der li-Reduktionssemantik „ergänzen“.

Definition 4.6 li-Datentyp
Sei $\langle \mathrm{T}_{\mathcal{C}}^{\perp}, \alpha \rangle$ der Datentyp der li-Reduktionssemantik, dessen Operationen abgesehen von den Verzweigungsoperationen alle strikt sind, und für den

$$\alpha(\mathrm{cond}_G)(\underline{t}_1, \underline{t}_2, \underline{t}_3) := \begin{cases} \perp & , \quad \text{falls } \underline{t}_1 = \perp \\ \underline{t}_2 & , \quad \text{falls eine Variablenbelegung } \beta : \{x_1, \ldots, x_n\} \to \mathrm{T}_{\mathcal{C}} \\ & \quad \text{mit } [\![G(x_1, \ldots, x_n)]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}}, \beta} = \underline{t}_1 \ (\neq \perp) \text{ existiert} \\ \underline{t}_3 & , \quad \text{andernfalls} \end{cases}$$

für alle $\mathrm{cond}_G \in \mathcal{H}$ und $\underline{t}_1, \underline{t}_2, \underline{t}_3 \in \mathrm{T}_{\mathcal{C}}^{\perp}$ gilt.
$\langle \mathrm{T}_{\mathcal{C}}^{\perp}, \trianglelefteq \rangle$ sei die flache Halbordnung des Rechenbereichs.
Dann heißt $\mathcal{D}_P^{\mathrm{li}} := \langle \mathrm{T}_{\mathcal{C}}^{\perp}, \trianglelefteq, \alpha \rangle$ **li-Datentyp** □

Die Ergänzung um die flache Halbordnung wurde einzig und allein zur Vergleichbarkeit mit dem cbv-Fixpunktdatentyp $\mathcal{D}_{P,\mathrm{cbv}}^{\mathrm{fix}}$ vorgenommen. Der li-Datentyp ist damit eine geordnete Algebra. Es ist noch zu zeigen, daß $\mathcal{D}_P^{\mathrm{li}}$ wohldefiniert ist, d. h. daß die geforderte Striktheit auch gegeben ist, wenn ein $t_{\perp} \in \mathrm{T}_{\Sigma}$ mit $[\![t_{\perp}]\!]_P^{\mathrm{li}} = \perp$ existiert. Außerdem haben wir für den li-Datentyp zur Vereinfachung des Übereinstimmungsbeweises die Verzweigungsoperationen vollständig neu angegeben. Wir müssen zeigen, daß diese mit der li-Reduktionssemantik übereinstimmen. Andernfalls gäbe es gar keinen li-Datentyp.

Lemma 4.4 Wohldefiniertheit des li-Datentyps
Der li-Datentyp, $\mathcal{D}_P^{\mathrm{li}}$, ist wohldefiniert, d. h. es gilt

$$[\![g(\vec{t})]\!]_P^{\mathrm{li}} = \perp$$

für alle $g^{(n)} \in \Sigma \setminus \{\mathrm{cond}_G \in \mathcal{H}\}$ und $\vec{t} \in (\mathrm{T}_{\Sigma})^n$ mit $\perp \in \{[\![t_1]\!]_P^{\mathrm{li}}, \ldots, [\![t_n]\!]_P^{\mathrm{li}}\}$
und

$$[\![\mathrm{cond}_G(t_1, t_2, t_3)]\!]_P^{\mathrm{li}} = \begin{cases} \perp & , \quad \text{falls } \underline{t}_1 = \perp \\ \underline{t}_2 & , \quad \text{falls eine Variablenbelegung } \beta : \{x_1, \ldots, x_n\} \to \mathrm{T}_{\mathcal{C}} \\ & \quad \text{mit } [\![G(x_1, \ldots, x_n)]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}}, \beta} = \underline{t}_1 \ (\neq \perp) \text{ existiert} \\ \underline{t}_3 & , \quad \text{andernfalls} \end{cases}$$

für alle $\mathrm{cond}_G \in \mathcal{H}$ und $t_1, t_2, t_3 \in \mathrm{T}_{\Sigma}$, $\underline{t}_1, \underline{t}_2, \underline{t}_3 \in \mathrm{T}_{\mathcal{C}}^{\perp}$ mit $[\![t_1]\!]_P^{\mathrm{li}} = \underline{t}_1, [\![t_2]\!]_P^{\mathrm{li}} = \underline{t}_2, [\![t_3]\!]_P^{\mathrm{li}} = \underline{t}_3$.
Beweis:

Teil 1: Ist für ein $i \in [n]$ $[\![t_i]\!]_P^{\mathrm{li}} = \perp$, so besitzt t_i keine li-Konstruktornormalform. Dann besitzt auch $g(t_1, \ldots, t_n)$ keine li-Konstruktornormalform und es gilt $[\![g(\vec{t})]\!]_P^{\mathrm{li}} = \perp$.

Teil 2: Fall 1: $\underline{t}_1 = \perp$.

Somit besitzt t_1 keine li-Konstruktornormalform. Also besitzt auch $\mathrm{cond}_G(t_1, t_2, t_3)$ keine li-Konstruktornormalform und es gilt $[\![\mathrm{cond}_G(t_1, t_2, t_3)]\!]_P^{\mathrm{li}} = \perp$.

Fall 2: Es existiert eine Variablenbelegung $\beta : \{x_1, \ldots, x_n\} \to \mathrm{T}_{\mathcal{C}}$ mit $[\![G(x_1, \ldots, x_n)]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}}, \beta} = \underline{t}_1 \ (\neq \perp)$.

Es ist

$$t_1 \downarrow_{P,\mathrm{li}} = \underline{t}_1 = [\![G(x_1, \ldots, x_n)]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}}, \beta} = G(\ldots),$$

und daher

$$\mathrm{cond}_G(t_1, t_2, t_3) \xrightarrow[P,\mathrm{li}]{*} \mathrm{cond}_G(t_1 \downarrow_{P,\mathrm{li}}, t_2, t_3) \xrightarrow[P,\mathrm{li}]{} t_2.$$

Somit gilt

$$[\![\mathrm{cond}_G(t_1, t_2, t_3)]\!]^{\mathrm{li}}_P = [\![t_2]\!]^{\mathrm{li}}_P = \underline{t}_2.$$

Fall 3: Andernfalls.

Analog zu Fall 2 folgt

$$[\![\mathrm{cond}_G(t_1, t_2, t_3)]\!]^{\mathrm{li}}_P = [\![t_3]\!]^{\mathrm{li}}_P = \underline{t}_3.$$

□

Existiert ein Term $t_\perp \in \mathrm{T}_\Sigma$ mit $[\![t_\perp]\!]^{\mathrm{li}}_P = \perp$, so existiert auch nur ein einziger Datentyp der li-Reduktionssemantik — unser gerade definierter li-Datentyp $\mathcal{D}^{\mathrm{li}}_P$. Andernfalls existieren jedoch mehrere. Wir haben einen von diesen als li-Datentyp ausgewählt. Diese Auswahl erfolgte natürlich im Hinblick auf die im folgenden Satz zu beweisende Übereinstimmung von $\mathcal{D}^{\mathrm{li}}_P$ und $\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbv}}$. Der Datentyp $\mathcal{D}^{\mathrm{li}}_P$ ist jedoch noch auf eine andere Weise ausgezeichnet: Ergänzen wir unser Programm P um ein neues Funktionssymbol `undef` und eine Programmregel

$$\texttt{undef} \quad \rightarrow \quad \texttt{undef}$$

zu einem Programm P', so ist $[\![\texttt{undef}]\!]^{\mathrm{li}}_{P'} = \perp$. Die meisten Operationen des nun wieder eindeutigen Datentyps der li-Reduktionssemantik, $\mathcal{D}^{\mathrm{li}}_{P'}$, unterscheiden sich jetzt von den entsprechenden Operationen der Datentypen der li-Reduktionssemantik des Programms P; abgesehen von dem einen Datentyp $\mathcal{D}^{\mathrm{li}}_P$. $\mathcal{D}^{\mathrm{li}}_{P'}$ ist schlicht die Ergänzung von $\mathcal{D}^{\mathrm{li}}_P$ um die eine Operation des Funktionssymbols `undef`. Der li-Datentyp $\mathcal{D}^{\mathrm{li}}_P$ ist der einzige **kompositionelle** Datentyp aller Datentypen der li-Reduktionssemantik. Da die schrittweise Erweiterung eines Programms um zusätzliche Funktionen eine übliche Vorgehensweise ist, ist der Erhalt des bisher definierten Datentyps, also die Kompositionalität der Semantik, wünschenswert. Wir werden in Kapitel 6 noch einmal darauf eingehen.

Satz 4.5 Übereinstimmung der Datentypen $\mathcal{D}^{\mathrm{li}}_P$ und $\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbv}}$
Der li-Datentyp ist gleich dem cbv-Fixpunktdatentyp:

$$\mathcal{D}^{\mathrm{li}}_P = \mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbv}}.$$

Beweis:
Wir zeigen in 3 Schritten, daß $\mathcal{D}^{\mathrm{li}}_P = \langle \mathrm{T}^{\perp}_{\mathcal{C}}, \trianglelefteq, \alpha \rangle$ ebenfalls der kleinste Fixpunkt der cbv-Transformation $\Phi_{P,\mathrm{cbv}}$ und somit gleich $\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbv}}$ ist.

Schritt 1: $\mathcal{D}^{\mathrm{li}}_P$ ist eine cbv-Interpretation.

$\langle \mathrm{T}^{\perp}_{\mathcal{C}}, \trianglelefteq \rangle$ ist die flache Halbordnung des Rechenbereichs.

Für alle $G^{(n)} \in \mathcal{C}$ und $\underline{t}_1, \ldots, \underline{t}_n \in \mathrm{T}_{\mathcal{C}}$ gilt

$$G(\underline{t}_1, \ldots, \underline{t}_n) = G(\underline{t}_1, \ldots, \underline{t}_n) \downarrow_{P,\mathrm{li}},$$

also

$$G^{\mathcal{D}^{\mathrm{li}}_P}(\underline{t}_1, \ldots, \underline{t}_n) = [\![G(\underline{t}_1, \ldots, \underline{t}_n)]\!]^{\mathrm{li}}_P = G(\underline{t}_1, \ldots, \underline{t}_n).$$

Auch sind alle Konstruktor-, Funktions- und Selektionsoperationen strikt. Ebenso sind die Verzweigungsoperationen gemäß Definition im ersten Argument strikt. Aus dieser Striktheit

folgt auch gleich die ω-Stetigkeit der Operationen (über der flachen Halbordnung). Die Verzweigungsoperationen des li-Datentyps sind gemäß dessen Definition und dem Korollar 4.3 sowieso gleich den Verzweigungsoperationen des (ω-vollständigen) cbv-Fixpunktdatentyps.

Somit ist $\mathcal{D}_P^{\mathrm{li}}$ ω-vollständig und eine cbv-Interpretation.

Schritt 2: $\mathcal{D}_P^{\mathrm{li}}$ ist ein Fixpunkt der cbv-Transformation.

Träger, Ordnung und Konstruktoroperationen von $\Phi_{P,\mathrm{cbv}}(\mathcal{D}_P^{\mathrm{li}})$ und $\mathcal{D}_P^{\mathrm{li}}$ stimmen überein. Wir zeigen noch $f^{\Phi_{P,\mathrm{cbv}}(\mathcal{D}_P^{\mathrm{li}})} = f^{\mathcal{D}_P^{\mathrm{li}}}$ für alle $f \in \mathcal{F}(\dot\cup\, \mathcal{H})$.

Für die Verzweigungssymbole $\mathrm{cond}_G \in \mathcal{H}$ ist $\mathrm{cond}_G^{\Phi_{P,\mathrm{cbv}}(\mathcal{D}_P^{\mathrm{li}})} = \mathrm{cond}_G^{\mathcal{D}_P^{\mathrm{li}}}$ gegeben.

Sei $f^{(n)} \in \mathcal{F}(\dot\cup\, \{\mathrm{sel}_{G,i} \in \mathcal{H}\})$ und $t_1, \ldots, t_n \in \mathrm{T}_\Sigma$, $\underline{t}_1, \ldots, \underline{t}_n \in \mathrm{T}_\mathcal{C}^\perp$ mit $[\![t_1]\!]_P^{\mathrm{li}} = \underline{t}_1, \ldots, [\![t_n]\!]_P^{\mathrm{li}} = \underline{t}_n$.

Fall 1: Es existiert eine Reduktionsregel $f(\vec{p}) \to r \in \hat{P}$ und eine Substitution $\sigma : \mathrm{Var}(\vec{p}) \to \mathrm{T}_\mathcal{C}$ mit $p_1\sigma = \underline{t}_1, \ldots, p_n\sigma = \underline{t}_n$.
Da $t_1{\downarrow_{P,\mathrm{li}}} = \underline{t}_1 \neq \perp, \ldots, t_n{\downarrow_{P,\mathrm{li}}} = \underline{t}_n \neq \perp$, gilt:

$$f(t_1, \ldots, t_n) \xrightarrow[P,\mathrm{li}]{*} f(t_1{\downarrow_{P,\mathrm{li}}}, \ldots, t_n{\downarrow_{P,\mathrm{li}}}) \xrightarrow[P,\mathrm{li}]{*} r\sigma$$

und somit

$$[\![f(t_1, \ldots, t_n)]\!]_P^{\mathrm{li}} = [\![r\sigma]\!]_P^{\mathrm{li}}.$$

Gemäß der Definition der cbv-Transformation ist

$$f^{\Phi_{P,\mathrm{cbv}}(\mathcal{D}_P^{\mathrm{li}})}(\underline{t}_1, \ldots, \underline{t}_n) = [\![r\sigma]\!]_{\mathcal{D}_P^{\mathrm{li}}}^{\mathrm{alg}}.$$

Nach Korollar 3.4, S. 48, über Datentypen einer Grundtermsemantik ist

$$[\![r\sigma]\!]_{\mathcal{D}_P^{\mathrm{li}}}^{\mathrm{alg}} = [\![r\sigma]\!]_P^{\mathrm{li}}.$$

Insgesamt gilt also

$$f^{\Phi_{P,\mathrm{cbv}}(\mathcal{D}_P^{\mathrm{li}})}(\underline{t}_1, \ldots, \underline{t}_n) = [\![r\sigma]\!]_{\mathcal{D}_P^{\mathrm{li}}}^{\mathrm{alg}} = [\![r\sigma]\!]_P^{\mathrm{li}} = [\![f(t_1, \ldots, t_n)]\!]_P^{\mathrm{li}} = f^{\mathcal{D}_P^{\mathrm{li}}}(\underline{t}_1, \ldots, \underline{t}_n).$$

Fall 2: $\perp \notin \{\underline{t}_1, \ldots, \underline{t}_n\}$ und es existiert keine passende Regel.

$$f(t_1, \ldots, t_n) \xrightarrow[P,\mathrm{li}]{*} f(t_1{\downarrow_{P,\mathrm{li}}}, \ldots, t_n{\downarrow_{P,\mathrm{li}}}) = f(t_1, \ldots, t_n){\downarrow_{P,\mathrm{li}}} \notin \mathrm{T}_\mathcal{C},$$

$$f^{\Phi_{P,\mathrm{cbv}}(\mathcal{D}_P^{\mathrm{li}})}(\underline{t}_1, \ldots, \underline{t}_n) = \perp = [\![f(t_1, \ldots, t_n){\downarrow_{P,\mathrm{li}}}]\!]_P^{\mathrm{li}} = [\![f(t_1, \ldots, t_n)]\!]_P^{\mathrm{li}} = f^{\mathcal{D}_P^{\mathrm{li}}}(\underline{t}_1, \ldots, \underline{t}_n).$$

Fall 3: $\perp \in \{\underline{t}_1, \ldots, \underline{t}_n\}$.
$f(t_1, \ldots, t_n)$ besitzt keine li-Konstruktornormalform. Somit ist

$$f^{\Phi_{P,\mathrm{cbv}}(\mathcal{D}_P^{\mathrm{li}})}(\underline{t}_1, \ldots, \underline{t}_n) = \perp = [\![f(t_1, \ldots, t_n)]\!]_P^{\mathrm{li}} = f^{\mathcal{D}_P^{\mathrm{li}}}(\underline{t}_1, \ldots, \underline{t}_n).$$

Schritt 3: $\mathcal{D}_P^{\mathrm{li}}$ ist der kleinste Fixpunkt der cbv-Transformation.

Angenommen, es gibt einen Fixpunkt $\mathfrak{A} = \mathrm{Int}_{\Sigma,\mathrm{cbv}}$ der Transformation $\Phi_{P,\mathrm{cbv}}$ mit $\mathfrak{A} \sqsubset \mathcal{D}_P^{\mathrm{li}}$. Demnach existiert ein $f^{(n)} \in \mathcal{F}(\dot\cup\,\mathcal{H})$, $\underline{t}_1, \ldots, \underline{t}_n \in \mathrm{T}_{\mathcal{C}}^{\downarrow}$ mit

$$f^{\mathfrak{A}}(\underline{t}_1, \ldots, \underline{t}_n) \lhd f^{\mathcal{D}_P^{\mathrm{li}}}(\underline{t}_1, \ldots, \underline{t}_n)$$

Somit ist $f^{\mathfrak{A}}(\underline{t}_1, \ldots, \underline{t}_n) = \bot$ und $f^{\mathcal{D}_P^{\mathrm{li}}}(\underline{t}_1, \ldots, \underline{t}_n) \neq \bot$.

Es existiert also ein Term $t \in \mathrm{T}_\Sigma$ mit

$$[\![t]\!]_{\mathfrak{A}}^{\mathrm{alg}} = \bot \text{ und } [\![t]\!]_{\mathcal{D}_P^{\mathrm{li}}}^{\mathrm{alg}} \neq \bot. \tag{$*$}$$

Aus $[\![t]\!]_{\mathcal{D}_P^{\mathrm{li}}}^{\mathrm{alg}} \neq \bot$ folgt die Existenz einer terminierenden li-Reduktionsfolge $t \xrightarrow[P,\mathrm{li}]{*} t{\downarrow_{P,\mathrm{li}}} \in \mathrm{T}_{\mathcal{C}}$.

Sei t ein bezüglich $\xrightarrow[P,\mathrm{li}]{*}$ minimaler Term mit obiger Eigenschaft $(*)$ (dabei ist t' „kleiner-gleich“ t gdw. $t \xrightarrow[P,\mathrm{li}]{*} t'$; „minimal“ ist wohldefiniert, da für alle Terme mit obiger Eigenschaft $(*)$ die li-Reduktion terminiert).

Die Existenz der li-Reduktionsfolge impliziert die Existenz der li-Reduktion

$$t \xrightarrow[P,\mathrm{li}]{u} t',$$

d. h. es gibt eine Reduktionsregel $l \to r \in \hat{P}$ und eine Grundsubstitution σ mit

$$t = t[u \leftarrow l\sigma] \tag{1}$$

$$t' = t[u \leftarrow r\sigma] \tag{2}$$

und außerdem ist selbstverständlich

$$[\![t]\!]_{\mathcal{D}_P^{\mathrm{li}}}^{\mathrm{alg}} = [\![t']\!]_{\mathcal{D}_P^{\mathrm{li}}}^{\mathrm{alg}}. \tag{3}$$

Da $\mathfrak{A}$ ein Fixpunkt der Transformation $\Phi_{P,\mathrm{cbv}}$ und $l\sigma = f(t_1, \ldots, t_n)$ mit $f^{(n)} \in \mathcal{F}(\dot\cup\,\mathcal{H})$ und $t_1, \ldots, t_n \in \mathrm{T}_\Sigma$ ist, gilt

$$[\![l\sigma]\!]_{\mathfrak{A}}^{\mathrm{alg}} = f^{\mathfrak{A}}([\![t_1]\!]_{\mathfrak{A}}^{\mathrm{alg}}, \ldots, [\![t_n]\!]_{\mathfrak{A}}^{\mathrm{alg}}) = [\![r\sigma]\!]_{\mathfrak{A}}^{\mathrm{alg}}.$$

Mit der Invarianz der algebraischen Termsemantik folgt hieraus

$$[\![t[u \leftarrow l\sigma]]\!]_{\mathfrak{A}}^{\mathrm{alg}} = [\![t[u \leftarrow r\sigma]]\!]_{\mathfrak{A}}^{\mathrm{alg}}, \tag{4}$$

und insgesamt gilt

$$\bot = [\![t]\!]_{\mathfrak{A}}^{\mathrm{alg}} \overset{(1)}{=} [\![t[u \leftarrow l\sigma]]\!]_{\mathfrak{A}}^{\mathrm{alg}} \overset{(4)}{=} [\![t[u \leftarrow r\sigma]]\!]_{\mathfrak{A}}^{\mathrm{alg}} \overset{(2)}{=} [\![t']\!]_{\mathfrak{A}}^{\mathrm{alg}}.$$

Somit ist $[\![t']\!]_{\mathfrak{A}}^{\mathrm{alg}} = \bot$ und nach (3) $[\![t']\!]_{\mathcal{D}_P^{\mathrm{li}}}^{\mathrm{alg}} \neq \bot$. Dies steht im Widerspruch zu der Annahme, daß t ein bezüglich $\xrightarrow[P,\mathrm{li}]{*}$ minimaler Term mit der Eigenschaft $(*)$ ist.

□

Aus der Gleichheit der Datentypen folgt auch die Gleichheit der Grundtermsemantiken.

Korollar 4.6 Gleichheit der li-Reduktionssemantik und der cbv-Fixpunktsemantik

$$[\![\cdot]\!]^{\mathrm{li}}_{P} = [\![\cdot]\!]^{\mathrm{fix}}_{P,\mathrm{cbv}}.$$

Beweis:
$[\![\cdot]\!]^{\mathrm{li}}_{P} = [\![\cdot]\!]^{\mathrm{alg}}_{\mathcal{D}^{\mathrm{li}}_{P}} = [\![\cdot]\!]^{\mathrm{alg}}_{\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbv}}} = [\![\cdot]\!]^{\mathrm{fix}}_{P,\mathrm{cbv}}$. □

4.3 Die cbn-Semantik

Der call-by-name Auswertungsmechanismus übergibt die Argumente einer Funktion an diese in einer unausgewerteten Form und wertet diese erst aus, wenn sie im Funktionskörper wirklich für das Funktionsergebnis benötigt werden. Während der cbv-Auswertungsmechanismus die Striktheit der Funktionen erzwingt, können in mit dem cbn-Auswertungsmechanismus arbeitenden funktionalen Programmiersprachen Funktionen auch nicht-strikt sein. Da insbesondere die die Datenelemente aufbauenden Konstruktoren nicht-strikt sind, ist die Spezifikation sogenannter „unendlicher" Datenstrukturen möglich. Diese „unendlichen" Datenstrukturen können zwar niemals selbst das Ergebnis einer (terminierenden) Berechnung sein, aber mit ihnen lassen sich viele Programmierprobleme sehr elegant lösen (siehe Kapitel 4 in [Fie&Har88]).

4.3.1 Die Fixpunktsemantik

Die Definition der cbn-Fixpunktsemantik erfolgt analog zur cbv-Fixpunktsemantik. Unterschiede ergeben sich nur aus der möglichen Nicht-Striktheit der Operationen. Funktions- und Hilfsoperationen sind nur noch strikt, wenn sie durch das Programm bzw. das assoziierte Termersetzungssystem derart spezifiziert sind, und Konstruktoroperationen sind generell nicht-strikt.
Da nun für die Applikation von Konstruktoroperationen auf $\perp$ neue Datenelemente benötigt werden, verwenden wir anstelle von $\mathrm{T}^{\perp}_{\mathcal{C}}$ die Menge der unendlichen, partiellen Konstruktorterme $\mathrm{T}^{\infty}_{\mathcal{C},\perp}$ als **Rechenbereich**. Die echt partiellen Konstruktorterme stellen Approximationen von totalen Konstruktortermen dar, und die kanonische Halbordnung $\langle \mathrm{T}^{\infty}_{\mathcal{C},\perp}, \trianglelefteq \rangle$ repräsentiert diese Approximation. Die Halbordnung $\langle \mathrm{T}^{\infty}_{\mathcal{C},\perp}, \trianglelefteq \rangle$ ist ω-vollständig, wie es für die Fixpunktsemantik benötigt wird.

Definition 4.7 cbn-Interpretation
Sei $\langle \mathrm{T}^{\infty}_{\mathcal{C},\perp}, \trianglelefteq \rangle$ die kanonische Halbordnung der unendlichen, partiellen Konstruktorterme. Sei

$$\alpha : \bigcup_{i \in \mathbb{N}} \Sigma_n \rightarrow [(\mathrm{T}^{\infty}_{\mathcal{C},\perp})^n \rightarrow \mathrm{T}^{\infty}_{\mathcal{C},\perp}]$$

eine Zuordnung von ω-stetigen Operationen an die Operationssymbole mit

$$\alpha(G)(\underline{t}_1, \ldots, \underline{t}_n) = G(\underline{t}_1, \ldots, \underline{t}_n)$$

für alle $G^{(n)} \in \mathcal{C}$ und $\underline{t}_1, \ldots, \underline{t}_n \in \mathrm{T}^{\infty}_{\mathcal{C},\perp}$.
Eine ω-vollständige Σ-Algebra $\mathfrak{A} = \langle \mathrm{T}^{\infty}_{\mathcal{C},\perp}, \trianglelefteq, \alpha \rangle$ heißt **cbn-Interpretation**. Die Menge aller cbn-Interpretationen zu einer festen Programmsignatur Σ wird mit $\mathrm{Int}_{\Sigma,\mathrm{cbn}}$ bezeichnet. Wir schreiben $\perp_{\mathrm{cbn}} := \perp_{\mathrm{Int}_{\Sigma,\mathrm{cbn}}}$ für das kleinste Element der kanonischen Halbordnung $\langle \mathrm{Int}_{\Sigma,\mathrm{cbn}}, \sqsubseteq \rangle$. □

Die cbn-Transformation unterscheidet sich von der cbv-Transformation im wesentlichen nur durch die fehlende Erzwingung der Striktheit der Operationen. Da die Nicht-Striktheit der Verzweigungsoperationen an der 2. und 3. Argumentstelle somit kein Problem mehr darstellt, ist eine getrennte Definition der Hilfsoperationen hier auch nicht mehr nötig.

Definition 4.8 cbn-Transformation
Die **cbn-Transformation** zu dem Programm P über Σ,

$$\Phi_{P,\mathrm{cbn}} : [\mathrm{Int}_{\Sigma,\mathrm{cbn}} \to \mathrm{Int}_{\Sigma,\mathrm{cbn}}],$$

ist definiert durch

$$f^{\Phi_{P,\mathrm{cbn}}(\mathfrak{A})}(\vec{\underline{t}}) := \begin{cases} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} & , \quad \text{falls eine Reduktionsregel } f(\vec{p}) \to r \in \hat{P} \\ & \quad \text{und eine Variablenbelegung } \beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}^{\infty}_{\mathcal{C},\perp} \\ & \quad \text{mit } [\![p_1]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}},\beta} = \underline{t}_1, \ldots, [\![p_n]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}},\beta} = \underline{t}_n \text{ existiert} \\ \perp & , \quad \text{andernfalls} \end{cases}$$

für alle $f^{(n)} \in \mathcal{F}, \vec{\underline{t}} \in (\mathrm{T}^{\infty}_{\mathcal{C},\perp})^n$ und $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\mathrm{cbn}}$. □

Es ist zu beachten, daß bei der Definition der cbn-Transformation der Einsatz algebraischer Termsemantiken wirklich nötig ist. Aufgrund der partiellen und der unendlichen Konstruktorterme ist eine äquivalente Definition mit Substitutionen (wie bei der cbv-Transformation in Bemerkung 4.2, S. 58, erwähnt) nicht mehr möglich.

Definition 4.9 cbn-Fixpunktdatentyp und cbn-Fixpunktsemantik
Der **cbn-Fixpunktdatentyp** $\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbn}}$ des Programms P ist definiert als der kleinste Fixpunkt der cbn-Transformation $\Phi_{P,\mathrm{cbn}}$:

$$\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbn}} := \mathrm{Fix}(\Phi_{P,\mathrm{cbn}}) = \bigsqcup_{i \in \mathbb{N}} (\Phi_{P,\mathrm{cbn}})^i(\perp_{\mathrm{cbn}}).$$

Die **cbn-Fixpunktsemantik** $[\![t]\!]^{\mathrm{fix}}_{P,\mathrm{cbn}}$ des Grundterms $t \in \mathrm{T}_{\Sigma}$ bezüglich P ist definiert als die algebraische Grundtermsemantik von t bezüglich des cbn-Fixpunktdatentyps:

$$[\![t]\!]^{\mathrm{fix}}_{P,\mathrm{cbn}} := [\![t]\!]^{\mathrm{alg}}_{\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbn}}}.$$

□

Zum Vergleich mit der cbv-Fixpunktsemantik (Bsp. 4.5, S. 59) betrachten wir die cbn-Fixpunktsemantik des für die Normalformsemantik so problematischen Programms aus Beispiel 3.6, S. 45. Als algebraische Termsemantik ist die cbn-Fixpunktsemantik natürlich invariant.

Beispiel 4.6 Bestimmung eines cbn-Fixpunktdatentyps

```
liste1      →  []:liste1
liste2      →  [[]]:liste2
head(x:xs)  →  x
```

Sei $\mathfrak{A}_i := (\Phi_{P,\mathrm{cbn}})^i(\bot_{\mathrm{cbn}})$ für alle $i \in \mathbb{N}$.

	$i=0$	$i=1$	$i=2$	$\ldots$	$i=\infty$ $(\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbv}} = \mathfrak{A}_i)$
$\texttt{liste1}^{\mathfrak{A}_i}()$	$\bot$	$[] : \bot$	$[] : [] : \bot$	$\ldots$	$[[], [], [], \ldots]$
$\texttt{liste2}^{\mathfrak{A}_i}()$	$\bot$	$[[]] : \bot$	$[[]] : [[]] : \bot$	$\ldots$	$[[[]], [[]], [[]], \ldots]$
$\texttt{head}^{\mathfrak{A}_i}$	$\underline{t} \mapsto \bot$	$\begin{pmatrix} \bot & \mapsto & \bot \\ [] & \mapsto & \bot \\ \underline{t}_1{:}\underline{t}_2 & \mapsto & \underline{t}_1 \end{pmatrix}$	wie $i=1$	$\ldots$	wie $i=1$

mit $\underline{t}, \underline{t}_1, \underline{t}_2 \in \mathrm{T}^{\infty}_{\mathcal{C},\bot}$.
Somit ist

$$[\![\texttt{liste1}]\!]^{\mathrm{fix}}_{\mathrm{P,cbn}} = [[], [], [], \ldots] \text{ und } [\![\texttt{liste2}]\!]^{\mathrm{fix}}_{\mathrm{P,cbn}} = [[[]], [[]], [[]], \ldots],$$

und

$$[\![\texttt{head(liste1)}]\!]^{\mathrm{fix}}_{P,\mathrm{cbn}} = [] \text{ und } [\![\texttt{head(liste2)}]\!]^{\mathrm{fix}}_{P,\mathrm{cbn}} = [[]]$$

stellt keinen Widerspruch zur Invarianz dar. □

Ein anderes Beispiel verdeutlicht die Ausdrucksmächtigkeit der Programme mit Pattern zusammen mit der cbn-Semantik. Wir werden in Kapitel 7 noch einmal auf dieses Beispiel eingehen.

Beispiel 4.7 Paralleles And

```
and(False,x)     →  False
and(x,False)     →  False
and(True,True)   →  True
```

Die Operation $\texttt{and}^{\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbn}}}$ wird durch die folgende Wertetabelle beschrieben.

$\texttt{and}^{\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbn}}}$	$\bot$	False	True
$\bot$	$\bot$	False	$\bot$
False	False	False	False
True	$\bot$	False	True

□

BEMERKUNG 4.3: Überabzählbarkeit des Rechenbereichs
Die Menge aller unendlichen, partiellen Konstruktorterme ist leider überabzählbar, wir wie in 2.4.2 festgestellt haben. Da jedoch nur eine abzählbare Menge von syntaktischen Termen T_Σ und eine abzählbare Menge von Programmen existiert, benötigen wir für deren Semantik eigentlich auch nur einen abzählbaren Rechenbereich (wir können „abzählbar“ sogar durch „aufzählbar“ ersetzen). Für viele Elemente $\underline{t}$ des Rechenbereichs $\mathrm{T}^{\infty}_{\mathcal{C},\bot}$ existiert kein $t \in \mathrm{T}_\Sigma$ und kein Programm P mit $[\![t]\!]^{\mathrm{fix}}_{P,\mathrm{cbn}} = \underline{t}$. Es ist jedoch keine sinnvolle Charakterisierung der wirklich benötigten Teilmenge von $\mathrm{T}^{\infty}_{\mathcal{C},\bot}$ bekannt. Die schlichte Definition der benötigten Teilmenge als die Menge aller durch einen beliebigen Term bei einem beliebigen Programm denotierten unendlichen, partiellen Konstruktorterme stellt keine nützliche Lösung dar. Außerdem sind alle endlichen, partiellen Konstruktorterme $\mathrm{T}_{\mathcal{C},\bot}$ denotierbar, und die echt unendlichen werden zumindest für die Fixpunktsemantik (und auch für die Reduktionssemantik wie wir sehen werden) aufgrund der geforderten ω-Vollständigkeit benötigt. □

4.3.2 Die Reduktionssemantik

Der cbn-Auswertungsmechanismus wird üblicherweise mit der leftmost-outermost Reduktionsstrategie gleichgesetzt. Wir definieren diese nun formal, wobei wir aus dem in Bemerkung 4.1, S. 53, genannten Grund nur (leftmost-)outermost Redexstellen und keine (leftmost-)outermost Redexe definieren.

Definition 4.10 Leftmost-outermost (lo-) Reduktion
Sei $t \in \mathrm{T}_\Sigma$.

- Die Stelle $u \in \mathrm{RedOcc}_P(t)$ heißt genau dann **outermost Redexstelle des Terms t**, wenn keine Redexstelle $v \in \mathrm{RedOcc}_P(t)$ mit $v < u$ existiert.
- Die bezüglich der lexikographischen Ordnung kleinste outermost Redexstelle von t heißt **leftmost-outermost Redexstelle des Terms t**.
- Eine Reduktion $A = t \xrightarrow[l \to r]{u} t'$ heißt genau dann **leftmost-outermost Reduktion**, wenn u die lo-Redexstelle von t ist. Hierdurch ist auch die **leftmost-outermost Reduktionsstrategie** $\xrightarrow[P,\mathrm{lo}]{}$ definiert.

□

Da ein Term nur höchstens eine lo-Redexstelle besitzt, ist die lo-Reduktionsrelation $\xrightarrow[P,\mathrm{lo}]{}$ wirklich eine sequentielle Reduktionsstrategie.
Wir haben in Beispiel 4.1, S. 50, die lo-Reduktionsstrategie erfolgreich für die cbn-Auswertung eingesetzt. Dennoch ist die lo-Reduktionsstrategie im allgemeinen leider unvollständig; sie stimmt nicht mit der cbn-Fixpunktsemantik überein.

Beispiel 4.8 Unvollständigkeit der lo-Reduktionsstrategie

$$\begin{array}{lcl} \texttt{and(x,False)} & \to & \texttt{False} \\ \texttt{a} & \to & \texttt{False} \\ \texttt{undef} & \to & \texttt{undef} \end{array}$$

$$[\![\texttt{and}(\texttt{undef}, \texttt{a})]\!]^{\mathrm{fix}}_{P,\mathrm{cbn}} = \texttt{and}^{\mathcal{D}^{\mathrm{fix}}_{P,\mathrm{cbn}}}(\bot, \texttt{False}) = \texttt{False},$$

aber:

$$\texttt{and}(\underline{\texttt{undef}}, \texttt{a}) \xrightarrow[P,\mathrm{lo}]{} \texttt{and}(\underline{\texttt{undef}}, \texttt{a}) \xrightarrow[P,\mathrm{lo}]{} \ldots.$$

Das 2. Argument `a` wird niemals zu `False` reduziert, so daß die Programmregel für `and` niemals angewendet werden kann. □

Daher verwenden wir für die cbn-Reduktionssemantik die parallel-outermost Reduktion.

Definition 4.11 Parallel-outermost (po-) Reduktion
Eine Reduktion $A = t \xrightarrow[P]{U} t'$ heißt genau dann **parallel-outermost Reduktion**, wenn U die Menge aller outermost Redexe des Terms t ist. Hierdurch ist auch die **parallel-outermost Reduktionsstrategie** $\xrightarrow[P,po]{}$ definiert. □

Trivialerweise sind die outermost Redexe eines Terms voneinander unabhängig. Die (deterministische) po-Reduktionsstrategie ist eine parallele Reduktionsstrategie.
Mit der po-Reduktionsstrategie bekommen wir zu Beispiel 4.8 nun den gewünschten Konstruktorterm als po-Normalform:

$$\texttt{and}(\underline{\texttt{undef}}, \underline{\texttt{a}}) \xrightarrow[P,\mathrm{po}]{}\!\!\!\!\!\rightarrow \underline{\texttt{and}}(\texttt{undef}, \texttt{False}) \xrightarrow[P,\mathrm{po}]{}\!\!\!\!\!\rightarrow \texttt{False}.$$

Die einfache Verwendung der po-Normalform als Semantik eines Terms, wenn diese existiert und ein Konstruktorterm ist, und $\bot$ andernfalls — analog zur li-Reduktionssemantik —, ist hier natürlich nicht mehr möglich, da wir als semantischen Wertebereich den Rechenbereich $\mathrm{T}^{\infty}_{\mathcal{C},\bot}$ mit partiellen und unendlichen Konstruktortermen verwenden.

Beispiel 4.9 Unendliche Liste

$$\texttt{liste1} \quad \rightarrow \quad \texttt{[]:liste1}$$

Gemäß Beispiel 4.6 ist

$$[\![\texttt{liste1}]\!]^{\mathrm{fix}}_{\mathrm{P,cbn}} = [[], [], [], \ldots].$$

Wir berechnen

$$\texttt{liste1} \xrightarrow[P,\mathrm{po}]{}\!\!\!\!\!\rightarrow [] : \texttt{liste1} \xrightarrow[P,\mathrm{po}]{}\!\!\!\!\!\rightarrow [] : [] : \texttt{liste1} \xrightarrow[P,\mathrm{po}]{}\!\!\!\!\!\rightarrow [] : [] : [] : \texttt{liste1} \xrightarrow[P,\mathrm{po}]{}\!\!\!\!\!\rightarrow \ldots.$$

□

Natürlich können sich echt unendliche Konstruktorterme niemals als Ergebnis einer Berechnung ergeben. Auch für syntaktische Terme, denen ein echt partieller Konstruktorterm als Semantik zugeordnet ist, kann dieser oft nicht berechnet werden, da insbesondere die ergebnislose Nichttermination mit $\bot$ gleichgesetzt wird. Aber derartige Berechnungsterme können beliebig genau approximiert werden.
In obiger Reduktionsfolge von `liste1` ist die Approximation der unendlichen Ergebnisliste gut sichtbar. Offensichtlich können Konstruktorsymbole, die sich oben bzw. außen in einem Term befinden, auch niemals durch Reduktion wieder verschwinden. Dieser „obere Konstruktorteil“ eines Terms ist somit mit Sicherheit eine Approximation des Endergebnisses. Wir bekommen diesen „oberen Konstruktorteil“ durch die algebraische Termsemantik des Terms bezüglich der kleinsten cbn-Interpretation $\bot_{\mathrm{cbn}}$, in der alle Funktions- und Hilfsoperationen stets $\bot$ als Ergebnis liefern.

Definition 4.12 Semantische cbn-Approximation
Die algebraische Semantik bezüglich der kleinsten cbn-Interpretation $\bot_{\mathrm{cbn}}$,

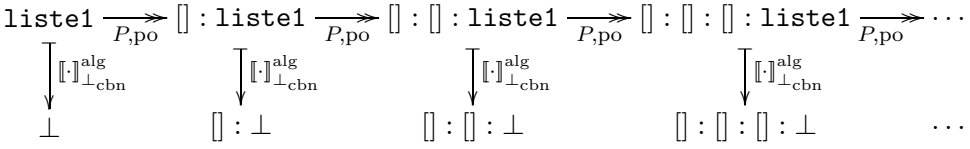
$$[\![\cdot]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbn}}} : \mathrm{T}_{\Sigma} \rightarrow \mathrm{T}^{\infty}_{\mathcal{C},\bot},$$

heißt **semantische cbn-Approximation**.

$$\begin{aligned} [\![G(t_1,\ldots,t_n)]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbn}}} &= G([\![t_1]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbn}}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbn}}}) \\ [\![f(t_1,\ldots,t_n)]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbn}}} &= \bot \end{aligned}$$

für alle $G^{(n)} \in \mathcal{C}$, $f^{(n)} \in \mathcal{F}(\,\dot\cup\,\mathcal{H})$ (gemäß der Definitionen der algebraischen Termsemantik und der cbn-Interpretation). □

Die semantischen Approximationen der Terme der po-Reduktionsfolge bilden eine ω-Kette:



Die aufgrund der ω-Vollständigkeit der kanonischen Halbordnung $\langle \mathrm{T}^{\infty}_{\mathcal{C},\perp}, \trianglelefteq \rangle$ existierende kleinste obere Schranke dieser ω-Kette ist dann die po-Reduktionssemantik.

Definition 4.13 **po-Reduktionssemantik**
Die **po-Reduktionssemantik**

$$[\![\cdot]\!]^{\mathrm{po}}_P : \mathrm{T}_\Sigma \to \mathrm{T}^{\infty}_{\mathcal{C},\perp}$$

ist für einen Term $t \in \mathrm{T}_\Sigma$ definiert durch

$$[\![t]\!]^{\mathrm{po}}_P := \bigsqcup \{ [\![t']\!]^{\mathrm{alg}}_{\perp\mathrm{cbn}} \mid t \xrightarrow[P,\mathrm{po}]{}{}^{*} t' \}.$$

□

Für Beispiel 4.9 erhalten wir also wie gewünscht

$$[\![\texttt{liste1}]\!]^{\mathrm{po}}_P = \bigsqcup \{\perp, [] : \perp, [] : [] : \perp, [] : [] : [] : \perp, \ldots\} = [[], [], [], \ldots].$$

Hiermit haben wir die Definition der po-Reduktionssemantik motiviert. Einen Beweis ihrer Wohldefiniertheit, d. h. daß die Approximationen wirklich eine ω-Kette bilden, führen wir jedoch erst in Kapitel 5.

In Anbetracht der Tatsache, daß die po-Reduktionssemantik eines Terms immer die kleinste obere Schranke einer ω-Kette von Approximationen ist, stellt sich nun die Frage, ob eventuell auch bei endlichen, totalen Konstruktortermen als semantischer Wert dieser nur beliebig genau approximiert aber nie erreicht wird. Dies ist jedoch nicht der Fall.

Lemma 4.7 **Berechenbarkeit der po-Reduktionssemantik**
Sei $t \in \mathrm{T}_\Sigma$. Wenn

$$[\![t]\!]^{\mathrm{po}}_P = \underline{t} \in \mathrm{T}_{\mathcal{C}},$$

dann existiert die po-Normalform $t{\downarrow_{P,\mathrm{po}}}$, und es gilt

$$[\![t]\!]^{\mathrm{po}}_P = \underline{t} = t{\downarrow_{P,\mathrm{po}}}.$$

Beweis:
Spezialfall des Lemmas 5.21, S. 102, über die Berechenbarkeit der ς-Reduktionsemantiken. □

Es ist bekannt, daß die po-Reduktionsstrategie normalisierend ist ([O'Do77], Kapitel V, 5. – 7.). Die Invarianz, die die po-Reduktionssemantik im Gegensatz zur Normalformsemantik besitzt, wird durch eine stärkere Differenzierung der Terme ohne Normalform vermittels der echt partiellen und echt unendlichen Konstruktorterme erreicht. Leider können wir die Invarianz nicht auf ähnlich einfache Weise wie bei der li-Reduktionssemantik zeigen (Lemma 4.1, S. 54). Ein direkter Beweis der Invarianz der po-Reduktionsemantik erweist sich als praktisch unmöglich. Stattdessen werden wir in Kapitel 5 erst die Übereinstimmung der Fixpunktsemantik und der po-Reduktionssemantik

zeigen, und dann daraus mit der Invarianz der algebraischen Termsemantik die Invarianz der po-Reduktionssemantik folgern.
Somit können wir aber auch den Beweis der Übereinstimmung der Fixpunkt- und der po-Reduktionssemantik nicht analog zu dem entsprechenden Beweis für die cbv-Semantik führen. Dort haben wir einen speziellen Datentyp der li-Reduktionssemantik, $\mathcal{D}_P^{\text{li}}$, definiert. Für dessen Wohldefiniertheit ist die Invarianz der li-Reduktionssemantik jedoch Voraussetzung. Außerdem wäre die analoge Definition eines po-Datentyps weitaus schwieriger, da nicht nur zu $\bot$, sondern auch zu vielen anderen echt partiellen und echt unendlichen Berechnungstermen im allgemeinen kein sie denotierender syntaktischer Term existiert. Dies folgt schon allein aus der Überabzählbarkeit von $\mathrm{T}_{\mathcal{C},\bot}^{\infty}$. Immerhin läßt sich leicht einsehen, daß für alle endlichen, partiellen Berechnungsterme $\mathrm{T}_{\mathcal{C},\bot}$ jeweils ein sie denotierender syntaktischer Term aus T_Σ existiert, wenn dies nur für einen einzigen echt partiellen $(\mathrm{T}_{\mathcal{C},\bot} \setminus \mathrm{T}_{\mathcal{C}})$ der Fall ist. Aufgrund der geforderten ω-Stetigkeit der Operationen wäre die Fortsetzung der Operationen auf die echt unendlichen Konstruktorterme eindeutig. Die Wohldefiniertheit dieses po-Datentyps — bezüglich der Übereinstimmung mit der po-Reduktionssemantik — wäre jedoch auch noch zu zeigen. Schließlich wäre die Übereinstimmung des po-Datentyps und des cbn-Fixpunktdatentyps auch nicht analog zur cbv-Semantik beweisbar, da wir bei dieser die speziellen Eigenschaften der flachen Halbordnung ausgenutzt haben (Satz 4.5, S. 62). Wir werden daher in Kapitel 5 ein anderes Beweisprinzip verwenden.

Zum Schluß wollen wir noch einmal einen kurzen Blick auf die verworfene lo-Reduktionsstrategie werfen. Wir haben in Beispiel 4.8 die Unvollständigkeit der lo-Reduktionsstrategie anhand eines Programms mit Pattern aufgezeigt. Folgendes Beispiel zeigt, daß die Unvollständigkeit auch bei Programmen mit Hilfsfunktionen gegeben ist.

Beispiel 4.10 **Unvollständigkeit der lo-Reduktionsstrategie, II**

$$\begin{array}{lcl} \texttt{liste} & \to & \texttt{undef:liste} \\ \texttt{undef} & \to & \texttt{undef} \end{array}$$

$$[\![\texttt{liste}]\!]_{P,\text{cbn}}^{\text{fix}} = [\bot, \bot, \ldots]$$

und aus

$$\underline{\texttt{liste}} \xrightarrow[P,\text{po}]{}\!\!\!\!\!\rightarrow \underline{\texttt{undef}} : \underline{\texttt{liste}} \xrightarrow[P,\text{po}]{}\!\!\!\!\!\rightarrow \underline{\texttt{undef}} : \underline{\texttt{undef}} : \underline{\texttt{liste}} \xrightarrow[P,\text{po}]{}\!\!\!\!\!\rightarrow \ldots$$

folgt auch

$$[\![\texttt{liste}]\!]_P^{\text{po}} = \bigsqcup\{\bot, \bot : \bot, \bot : \bot : \bot, \ldots\} = [\bot, \bot, \bot, \ldots],$$

aber

$$\underline{\texttt{liste}} \xrightarrow[P,\text{lo}]{} \underline{\texttt{undef}} : \texttt{liste} \xrightarrow[P,\text{lo}]{} \underline{\texttt{undef}} : \texttt{liste} \xrightarrow[P,\text{lo}]{} \ldots,$$

und somit wäre

$$[\![\texttt{liste}]\!]_P^{\text{lo}} = \bot : \bot.$$

□

Diese unvollständige Approximation ergibt sich jedoch nur bei Termen mit echt partiellen oder echt unendlichen Konstruktortermen als Semantik. Für Terme, die eine (po-)Konstruktornormalform besitzten, sind diese bei Programmen mit Hilfsfunktionen wirklich mit der lo-Reduktionsstrategie berechenbar. Auf den Beweis dieser Aussage wollen wir jedoch verzichten. Dennoch erklärt dies

die allgemeine Verwendung der lo-Reduktionsstrategie für den cbn-Auswertungsmechanismus. Programme (erster Ordnung) funktionaler Programmiersprachen wie Miranda, Haskell und Hope sind in gewisser Weise auch nur Programme mit Hilfsfunktionen und keine mit Pattern, wie wir in Kapitel 7 sehen werden.
Allerdings geben die mit dem cbn-Auswertungsmechanismus arbeitenden funktionalen Programmiersprachen für interaktive Ein- und Ausgabe auch Approximationen potentiell unendlicher Datenstrukturen aus. Jedoch handelt es sich bei diesen ausgegebenen Datenstrukturen immer um Listen und um keinen anderen algebraischen Datentyp (alle Ausgaben werden in Zeichenketten (Strings), also Listen konvertiert). Diese „passen“ aufgrund ihres speziellen Aufbaus genau zur lo-Reduktionsstrategie.

Beispiel 4.11 lo-Reduktion und unendliche Listen

$$\begin{array}{lcl} \texttt{liste1'} & \to & \texttt{a:liste1'} \\ \texttt{a} & \to & \texttt{[]} \end{array}$$

$$[\![\texttt{liste1}']\!]^{\mathrm{fix}}_{\mathrm{P,cbn}} = [[], [], [], \ldots].$$

Dieser unendliche Konstruktorterm wird auch durch die lo-Reduktionsstrategie approximiert:

$$\underline{\texttt{liste1}'} \xrightarrow[\mathrm{P,lo}]{} \underline{\texttt{a}} : \texttt{liste1}' \xrightarrow[\mathrm{P,lo}]{} [] : \underline{\texttt{liste1}'} \xrightarrow[\mathrm{P,lo}]{} [] : \underline{\texttt{a}} : \texttt{liste1}' \xrightarrow[\mathrm{P,lo}]{} \ldots.$$

Die zur lo-Reduktionsstrategie symmetrische rightmost-outermost Reduktionsstrategie produziert dagegen

$$\begin{array}{ccccccc} \underline{\texttt{liste1}'} & \xrightarrow[P,\mathrm{ro}]{} & \texttt{a} : \underline{\texttt{liste1}'} & \xrightarrow[P,\mathrm{ro}]{} & \texttt{a} : \texttt{a} : \underline{\texttt{liste1}'} & \xrightarrow[P,\mathrm{ro}]{} & \cdots \\ \downarrow [\![\cdot]\!]^{\mathrm{alg}}_{\perp\mathrm{cbn}} & & \downarrow [\![\cdot]\!]^{\mathrm{alg}}_{\perp\mathrm{cbn}} & & \downarrow [\![\cdot]\!]^{\mathrm{alg}}_{\perp\mathrm{cbn}} & & \\ \perp & & \perp : \perp & & \perp : \perp : \perp & & \cdots \end{array}$$

□

Wären die Argumente des Listenkonstruktorsymbols `Cons` bzw. `:` in der umgekehrten Reihenfolge, dann wäre somit die rightmost-outermost Reduktionsstrategie „passend“. Freilich hat die leftmost-outermost Reduktionsstrategie auch bei Listen ihre Grenzen:
Das Miranda-Programm

```
liste3  =  1:2:undef:3:[]
```

(`undef = undef`) produziert zu der Eingabe `liste3` nur die Ausgabe:

$$[1, 2,$$

Bei Betrachtung realer funktionaler Programmiersprachen gehören in diesen Zusammenhang jedoch auch noch Begriffe wie head-normal form und weak head-normal form, auf die wir jedoch nicht eingehen wollen (siehe [Fie&Har88]).

Kapitel 5

Die ς-Semantiken

Die Betrachtung der Striktheit der Operationen in unseren zwei Standardsemantiken cbv und cbn führt uns in 5.1 zum Konzept der sogenannten erzwungenen Striktheit ς. Hieraus ergibt sich die Definition einer bezüglich dieser erzwungenen Striktheit parametrisierten Menge von Semantiken für unsere Programme, den ς-Semantiken. Die cbv- und die cbn-Semantik sind nur zwei, allerdings ausgezeichnete Vertreter dieser Menge.
In 5.2 definieren wir die ς-Semantik durch eine Fixpunktsemantik. Wir zeigen auch, daß die cbv- und die cbn-Fixpunktsemantik wirklich spezielle ς-Fixpunktsemantiken sind (5.2.2). In 5.2.4 wird auch die Wohldefiniertheit der ς-Fixpunktsemantik und damit insbesondere auch die der cbv- und der cbn-Fixpunktsemantik bewiesen.
In 5.3 wenden wir uns dann Reduktionssemantiken für unsere ς-Semantik zu. Nach der Untersuchung des Begriffs der in der ς-Fixpunktsemantik korrekten (ς-)Reduktion definieren wir in 5.3.1 zwei ς-Reduktionssemantiken. Von diesen ist die erste mehr von theoretischem Interesse, während die zweite implementierungssnäher ist. Anschließend zeigen wir in 5.3.2 noch die Beziehung dieser ς-Reduktionssemantiken zu der in Kapitel 4 definierten li- und der po-Reduktionssemantik der cbv- respektive cbn-Semantik auf. Die Wohldefiniertheit dieser zwei Reduktionssemantiken wird in 5.3.3 bewiesen.
Daraufhin zeigen wir in 5.4 einige grundlegende Beziehungen zwischen der ς-Fixpunkt- und den ς-Reduktionssemantiken auf. Insbesondere beweisen wir in 5.4.3 die schon erwähnte Korrektheit der ς-Reduktion.
Damit können wir schließlich in 5.5 die Übereinstimmung der ς-Fixpunkt- und der zwei ς-Reduktionssemantiken beweisen. Sich hieraus ergebende Anregungen für effizientere ς-Reduktionssemantiken greifen wir in 5.5.4 kurz auf.

5.1 Verallgemeinerung der zwei Standardsemantiken

Bei Betrachtung der Definitionen der cbv- und der cbn-Fixpunktsemantik fällt auf, daß diese jeweils im wesentlichen aus zwei Teilen bestehen: den Interpretationen und der Transformation. Die Interpretationen stehen in enger Beziehung zum jeweiligen Basisdatentyp: die cbv-Semantik besitzt mit der flachen Halbordnung $\langle \mathrm{T}_{\mathcal{C}}^{\perp}, \trianglelefteq \rangle$ des Rechenbereichs nur endliche Datenstrukturen, während die cbn-Semantik mit $\langle \mathrm{T}_{\mathcal{C},\perp}^{\infty}, \trianglelefteq \rangle$ auch unendliche Datenstrukturen ermöglicht. Die Transformation legt dagegen die Semantik und damit insbesondere die Strikt- oder Nicht-Striktheit der Funktions- und Hilfsoperationen fest.
Da die Interpretationen und die Transformation jeweils ziemlich unabhängig voneinander sind, liegt der Gedanke eines Austauschs dieser zwischen cbv- und cbn-Semantik nahe. Bei nur geringfügigen

Änderungen läßt sich die cbn-Transformation auch auf die cbv-Interpretationen und umgekehrt die cbv-Transformation auf die cbn-Interpretationen anwenden.
Die auf erstere Art entstehende Semantik ist gar nicht so ungewöhnlich. In der Literatur über Fixpunktsemantiken von Funktionsschemata ([Vui74], [Manna74], [Nivat75], [Dow&Se76], [Loe&Sie87], [Ind93]) werden „cbn-Semantiken“ mit nicht-strikten Funktionen über flach geordneten Basisdatentypen (discrete domains) betrachtet (nur [Ber&Lévy77] und [Cou90] betrachten auch nicht-flach geordnete Basisdatentypen). Somit wird die „unnatürliche“, „erzwungene“ Striktheit der Funktionen in der Transformation vermieden; aufgrund der flachen Halbordnung des Basisdatentyps bleibt die gesamte Theorie der Fixpunktsemantik jedoch so einfach wie unsere cbv-Fixpunktsemantik.
Die Semantik, die auf die zweite Art entsteht, ist sogar in der funktionalen Programmiersprache Hope[1] realisiert worden (Kapitel 4 in [Fie&Har88]). Diese kombiniert nicht-strikte Konstruktoren mit darüber definierten strikten Funktionen. Auf diese Weise besitzt Hope die mächtigen Ausdrucksmöglichkeiten unendlicher Datenstrukturen, kann aber den in Implementationen effizienteren cbv-Auswertungsmechanismus teilweise verwenden.
Durch die möglichen Kombinationen von Interpretationen und Transformationen erhalten wir insgesamt vier verschiedene Semantiken. Dies hat den unerfreulichen Aspekt, daß wir auch vier verschiedene Reduktionssemantiken (noch zwei) definieren und viermal die Übereinstimmung von Fixpunkt- und Reduktionssemantik beweisen müssen. Zwischen den vier Semantiken bestehen zwar enge Beziehungen und daher Ähnlichkeiten, aber allgemeine Aussagen über genau diese vier lassen sich schwer beweisen.
Da außerdem die Hilfsoperationen zum Basisdatentyp gehören, sollte sich ihre Striktheit bzw. Nicht-Striktheit nach den Konstruktor- und nicht nach den Funktionsoperationen richten. Allerdings sind ihre Striktheiten aufgrund ihrer speziellen Reduktionsregeln sowieso in allen vier Semantiken gleich.
Die Lösung des Problems besteht in der schlichten Idee einer weiteren Verallgemeinerung. Wir führen einen neuen Parameter ς ein, der für jede Argumentstelle jedes Operationssymbols angibt, ob die Operation an dieser Argumentstelle strikt sein soll. Auf diese Weise erhalten wir zwar eine beliebig große Anzahl von Semantiken (in Abhängigkeit von der Programmsignatur), aber wir können diese auf ziemlich einfache Weise einheitlich behandeln. Wir werden sehen, daß gerade diese Verallgemeinerung zu einem tieferen Verständnis — auch der schon untersuchten cbv- und cbn-Semantiken — führt.
Die Striktheit bzw. Nicht-Striktheit der Konstruktoroperationen wird direkt durch den neuen Parameter ς bestimmt. Dagegen kann die Striktheit von Funktions- und Hilfsoperationen schon allein durch das Programm festgelegt sein (siehe dazu Beispiel 3.2, S. 39; die Operation von `and` ist auch in der cbn-Semantik strikt). Bei prinzipiell möglicher Nicht-Striktheit von Funktions- oder Hilfsoperationen soll der Parameter ς die Striktheit jedoch erzwingen können. Dies führt uns zu der Bezeichung „erzwungene Striktheit“.

Definition 5.1 **Erzwungene Striktheit**
Eine Abbildung

$$\varsigma : \bigcup_{n\in\mathbb{N}} \Sigma_n \rightarrow \mathbb{B}^n$$

heißt **erzwungene Striktheit** zur Signatur Σ. Sie ordnet jedem Signatursymbol $g \in \Sigma$ seine **erzwungene Striktheit** $\varsigma(g)$ zu.
Sei ς eine erzwungene Striktheit, $g^{(n)} \in \Sigma_n$ und $\underline{t}_1, \ldots, \underline{t}_n \in \mathrm{T}^{\infty}_{\Sigma,\perp}$. g heißt genau dann **erzwungen**

[1] Es existieren neben der Standardimplementierung allerdings auch Implementierungen von Hope, die mit dem reinen cbn-Auswertungsmechanismus arbeiten.

strikt für $\vec{\underline{t}}$, wenn

$$\exists i \in [n].\ \varsigma(g)(i) = \mathsf{tt} \wedge t_i = \bot$$

gilt. □

Im weiteren bezeichne ς stets eine beliebige aber feste erzwungene Striktheit.

Aus der durch ς festgelegten Striktheit der Konstruktoroperationen ergeben sich direkt die Datenelemente, d. h. der Rechenbereich der ς-Semantik.

Definition 5.2 Rechenbereich
Sei $\langle \mathrm{T}^{\infty}_{\mathcal{C},\bot}, \trianglelefteq \rangle$ die ω-vollständige Halbordnung der unendlichen, partiellen Konstruktorterme. Der **Rechenbereich zu C und ς**, $\mathrm{T}_{\mathcal{C},\varsigma}$, ist die kleinste Teilmenge von $\mathrm{T}^{\infty}_{\mathcal{C},\bot}$, die die folgenden zwei Eigenschaften besitzt:

1. $G^{(n)} \in \mathcal{C},\ \underline{t}_1, \ldots, \underline{t}_n \in \mathrm{T}_{\mathcal{C},\varsigma}, G$ nicht erzwungen strikt für $\vec{\underline{t}} \implies G(\underline{t}_1, \ldots, \underline{t}_n) \in \mathrm{T}_{\mathcal{C},\varsigma}$.
2. $T \subseteq \mathrm{T}_{\mathcal{C},\varsigma}$ abzählbare Kette $\implies \bigsqcup T \in \mathrm{T}_{\mathcal{C},\varsigma}$.

Mit der Relation $\trianglelefteq$ der ω-vollständigen Halbordnung der unendlichen, partiellen Konstruktorterme ist $\langle \mathrm{T}_{\mathcal{C},\varsigma}, \trianglelefteq \rangle$ die ω-vollständige Halbordnung des Rechenbereichs. □

Aufgrund der zweiten Eigenschaft ist die ω-Vollständigkeit von $\langle \mathrm{T}_{\mathcal{C},\varsigma}, \trianglelefteq \rangle$ trivialerweise gegeben. Insbesondere folgt mit $T = \emptyset$ aus der zweiten Eigenschaft, daß $\bot \in \mathrm{T}_{\mathcal{C},\varsigma}$ ist. Die endlichen Terme $\mathrm{T}_{\mathcal{C},\varsigma} \cap \mathrm{T}_{\mathcal{C},\bot}$ sind genau die ω-kompakten Terme und $\langle \mathrm{T}_{\mathcal{C},\varsigma}, \trianglelefteq \rangle$ ist ω-induktiv. Aus der ersten Eigenschaft folgt $\mathrm{T}_{\mathcal{C}} \subseteq \mathrm{T}_{\mathcal{C},\varsigma}$. Es gilt somit für beliebige Striktheitsfunktionen ς:

$$\mathrm{T}^{\bot}_{\mathcal{C}} \subseteq \mathrm{T}_{\mathcal{C},\varsigma} \subseteq \mathrm{T}^{\infty}_{\mathcal{C},\bot}.$$

Prinzipiell wäre die alleinige Verwendung aller unendlichen, partiellen Konstruktorterme $\mathrm{T}^{\infty}_{\mathcal{C},\bot}$ als Rechenbereich für alle ς-Semantiken denkbar, doch dann wären in den meisten ς-Semantiken viele Rechenterme überhaupt nicht denotierbar.
Außerdem sollen die schon definierten cbv- und cbn-Semantiken Spezialfälle der ς-Semantiken sein. Wir können dies schlicht dadurch zum Ausdruck bringen, daß wir die Bezeichnungen „cbv“ und „cbn“ als spezielle erzwungene Striktheiten auffassen.

Definition 5.3 Erzwungene Striktheiten cbv und cbn
Die erzwungenen Striktheiten

$$\mathrm{cbv}, \mathrm{cbn} : \bigcup_{n \in \mathbb{N}} \Sigma_n \to \mathbb{B}^n$$

sind definiert durch

$$\mathrm{cbv}(g)(i) := \mathsf{tt}$$

für alle $g^{(n)} \in \mathcal{C} \,\dot\cup\, \mathcal{F}, i \in [n]$,

$$\begin{aligned} \mathrm{cbv}(\mathrm{cond}_G) &:= (\mathsf{tt}, \mathsf{ff}, \mathsf{ff}) \\ \mathrm{cbv}(\mathrm{sel}_{G,i}) &:= (\mathsf{tt}) \end{aligned}$$

für alle $\mathrm{cond}_G, \mathrm{sel}_{G,i} \in \mathcal{H}$,

$$\mathrm{cbn}(g)(i) := \mathsf{ff}$$

für alle $g^{(n)} \in \Sigma, i \in [n]$. □

Die erzwungene Striktheit für die erste Argumentstelle der Verzweigungs- und die Argumentstelle der Selektionssymbole kann eigentlich beliebig sein, da diese Operationen schon aufgrund der zughörigen Reduktionsregeln an diesen Stellen strikt sind.
In Übereinstimmung mit den bisherigen Definitionen des Rechenbereichs ist $\mathrm{T}_{\mathcal{C},\mathrm{cbv}} = \mathrm{T}_{\mathcal{C}}^{\perp}$ und $\mathrm{T}_{\mathcal{C},\mathrm{cbn}} = \mathrm{T}_{\mathcal{C},\perp}^{\infty}$. Wie wir gesehen haben sind dies die beiden extremen möglichen Rechenbereiche. Ganz generell stellt cbv allerdings kein Extrem der erzwungenen Striktheit dar. Die Verzweigungssymbole sind an der 2. und 3. Argumentstelle immer noch nicht-strikt. Aber wir haben schon in 4.2.1 festgestellt, daß völlig strikte Verzweigungen nicht mehr sinnvoll einsetzbar wären.

In der ς-Fixpunktsemantik zeigt sich die Bedeutung der erzwungenen Striktheit ς.

5.2 Die ς-Fixpunktsemantik

5.2.1 Definition der ς-Fixpunktsemantik

Die Definition der ς-Fixpunktsemantik erfolgt völlig analog zur cbv- und cbn-Fixpunktsemantik. In der ς-Interpretation werden die möglichen Striktheiten aller Operationen entsprechend der erzwungenen Striktheit eingeschränkt.

Definition 5.4 **ς-Interpretation**
Sei $\langle \mathrm{T}_{\mathcal{C},\varsigma}, \trianglelefteq \rangle$ die ω-vollständige Halbordnung des Rechenbereichs. Sei

$$\alpha : \bigcup_{n\in\mathbb{N}} \Sigma_n \to [(\mathrm{T}_{\mathcal{C},\varsigma})^n \to \mathrm{T}_{\mathcal{C},\varsigma}]$$

eine Zuordnungsfunktion von ω-stetigen Operationen an die Operationssymbole mit

$$\alpha(G)(\underline{t}_1,\ldots,\underline{t}_n) = G(\underline{t}_1,\ldots,\underline{t}_n) \text{ , wenn } G \text{ nicht erzwungen strikt für } \vec{\underline{t}} \text{ ist}$$

für alle $G^{(n)} \in \mathcal{C}$, $\underline{t}_1,\ldots,\underline{t}_n \in \mathrm{T}_{\mathcal{C},\varsigma}$, und

$$\alpha(g)(\underline{t}_1,\ldots,\underline{t}_n) = \perp \text{ , wenn } g \text{ erzwungen strikt für } \vec{\underline{t}} \text{ ist,}$$

für alle $g^{(n)} \in \Sigma$ und $\underline{t}_1,\ldots,\underline{t}_n \in \mathrm{T}_{\mathcal{C},\varsigma}$.
Eine ω-vollständige Σ-Algebra $\langle \mathrm{T}_{\mathcal{C},\varsigma}, \trianglelefteq, \alpha \rangle$ heißt **ς-Interpretation**.
Die Menge aller ς-Interpretationen zu festem ς und Σ wird mit $\mathrm{Int}_{\Sigma,\varsigma}$ bezeichnet. $\langle \mathrm{Int}_{\Sigma,\varsigma}, \sqsubseteq \rangle$ ist die kanonische Halbordnung aller ς-Interpretationen. □

Man versichere sich, daß die Spezialfälle der cbv- und der cbn-Interpretation genau mit unseren früher gegebenen Definitionen 4.3, S. 55, respektive 4.7, S. 65, übereinstimmen.

Auch die ς-Transformation wird analog zu den cbv- und cbn-Semantiken definiert. Da das semantische Patternmatching jetzt jedoch komplizierter ist, definieren wir zuerst den Begriff des semantischen ς-Matchens, den wir dann direkt bei der ς-Transformation einsetzen.

Definition 5.5 **Semantisches ς-Matchen**
Ein Termtupel $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ heißt genau dann mit einem Redexschema $f^{(n)}(\vec{p}) \in \mathrm{RedS}_P$ vermittels einer Variablenbelegung $\beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C},\varsigma}$ **semantische ς-matchbar**, wenn

$$\begin{aligned} \forall i \in [n]. \qquad & (\varsigma(f)(i) = \mathsf{tt} \Rightarrow \underline{t}_i \neq \perp) \\ \wedge \quad & p_i[\perp/\mathrm{Var}(p_i)] \trianglelefteq \underline{t}_i \\ \wedge \quad & [\![p_i]\!]^{\mathrm{alg}}_{\perp_\varsigma,\beta} = \underline{t}_i \end{aligned}$$

□

Definition 5.6 ς-Transformation
Die **ς-Transformation** zum Programm P,

$$\Phi_{P,\varsigma} : [\mathrm{Int}_{\Sigma,\varsigma} \to \mathrm{Int}_{\Sigma,\varsigma}],$$

ist definiert durch

$$f^{\Phi_{P,\varsigma}(\mathfrak{A})}(\underline{\vec{t}}) := \begin{cases} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} & , \text{ wenn } \underline{\vec{t}} \text{ mit der linken Seite einer Reduktionsregel } f(\vec{p}) \to r \text{ von } P \\ & \quad \text{vermittels einer Variablenbelegung } \beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C},\varsigma} \\ & \quad \text{semantisch } \varsigma\text{-gematcht wird.} \\ \bot & , \text{ andernfalls} \end{cases}$$

für alle $\underline{\vec{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$, $f^{(n)} \in \mathcal{F}(\dot\cup\, \mathcal{H})$. □

Die Definition des semantischen ς-Matchens enthält neben der sich aus dem allgemeinen Konzept der erzwungenen Striktheit ergebenden Striktheitsbedingung und der eigentlichen Matchingbedingung ($[\![p_i]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta} = \underline{t}_i$ für alle $i \in [n]$) außerdem noch eine Ordnungsbedingung:

$$p_i[\bot/\mathrm{Var}(p_i)] \trianglelefteq \underline{t}_i$$

für alle $i \in [n]$. Das folgende Beispiel verdeutlicht die Notwendigkeit dieser Bedingung.

Beispiel 5.1 Die Ordnungsbedingung des semantischen ς-Matchens

```
f(G(x))  →  A
f(H(x))  →  B
```

Es sei $\varsigma(\mathtt{f}) := (\mathtt{ff}), \varsigma(\mathtt{G}) := (\mathtt{tt}), \varsigma(\mathtt{H}) := (\mathtt{tt})$ und $\underline{t} = \bot$.
Betrachten wir $\mathtt{f}^{\Phi_{P,\varsigma}(\mathfrak{A})}(\underline{t})$ für ein $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$.
f ist nicht erzwungen strikt für $\underline{t}$ und mit $\beta(\mathtt{x}) := \bot$ ist

$$[\![\mathtt{G(x)}]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta} = \mathtt{G}^{\bot_\varsigma}(\bot) = \bot = \underline{t}.$$

Somit wäre ohne die Ordnungsbedingung

$$\mathtt{f}^{\Phi_{P,\varsigma}(\mathfrak{A})}(\underline{t}) = [\![\mathtt{A}]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = \mathtt{A}.$$

Dies ist jedoch erstens nicht erwünscht und führt zweitens mit

$$[\![\mathtt{H(x)}]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta} = \mathtt{H}^{\bot_\varsigma}(\bot) = \bot = \underline{t}.$$

$$\mathtt{f}^{\Phi_{P,\varsigma}(\mathfrak{A})}(\underline{t}) = [\![\mathtt{B}]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = \mathtt{B}.$$

zu einem Widerspruch.
Aufgrund der Ordnungsbedingung ist jedoch $(\underline{t})$ weder mit `f(G(x))` noch mit `f(H(x))` semantisch ς-matchbar, und es gilt wie beabsichtigt

$$\mathtt{f}^{\Phi_{P,\varsigma}(\mathfrak{A})}(\underline{t}) = \bot.$$

□

Unsere früheren Definitionen 4.4 und 4.8 der cbv- respektive cbn-Transformation beinhalten diese Bedingung nicht; sie ist in beiden Fällen schon automatisch durch die Gültigkeit der restlichen Bedingung erfüllt, wie wir gleich in Abschnitt 5.2.2 zeigen werden.

Die Definition des ς-Fixpunktdatentyps und der ς-Fixpunktsemantik ist erwartungsgemäß.

Definition 5.7 ς-Fixpunktdatentyp und ς-Fixpunktsemantik
Der **ς-Fixpunktdatentyp** $\mathcal{D}^{\text{fix}}_{P,\varsigma}$ des Programms P ist definiert als der kleinste Fixpunkt der ς-Transformation $\Phi_{P,\varsigma}$:

$$\mathcal{D}^{\text{fix}}_{P,\varsigma} := \text{Fix}(\Phi_{P,\varsigma}) = \bigsqcup_{i\in\mathbb{N}} (\Phi_{P,\varsigma})^i(\bot_\varsigma).$$

Die **ς-Fixpunktsemantik** $[\![t]\!]^{\text{fix}}_{P,\varsigma}$ des Grundterms $t \in \text{T}_\Sigma$ bezüglich P ist definiert als die algebraische Grundtermsemantik von t bezüglich des ς-Fixpunktdatentyps:

$$[\![t]\!]^{\text{fix}}_{P,\varsigma} := [\![t]\!]^{\text{alg}}_{\mathcal{D}^{\text{fix}}_{P,\varsigma}}.$$

□

Wie wir schon bei der cbv-Fixpunktsemantik erwähnten, definieren wir die Hilfsoperationen genauso wie die Funktionsoperationen durch die ς-Transformation und nicht schon in der ς-Interpretation, weil diese Betrachtung der Programme mit Hilfsfunktionen als spezielle Programme mit Pattern die Definition der (ς-)Fixpunktsemantik vereinfacht. Die Hilfsoperationen gehören jedoch zum Basisdatentyp und sind von dem konkreten Programm unabhängig. Daher geben wir diese Operationen hier noch einmal direkt an.

Lemma 5.1 Hilfsoperationen in der ς-Fixpunktsemantik
Ist P ein Programm mit Hilfsfunktionen, so ergeben sich die Hilfsoperationen wie folgt:

$$\text{cond}_G^{\mathcal{D}^{\text{fix}}_{P,\varsigma}}(\underline{t}_1,\underline{t}_2,\underline{t}_3) = \begin{cases} \bot & , \text{ falls } \underline{t}_1 = \bot \text{ oder für ein } i \in \{2,3\} \\ & \quad (\varsigma(\text{cond}_G(i) = \mathsf{tt} \text{ und } \underline{t}_i = \bot) \\ \underline{t}_2 & , \text{ falls eine Variablenbelegung } \beta : \{x_1,\ldots,x_n\} \to \text{T}_{\mathcal{C},\varsigma} \\ & \quad \text{mit } [\![G(x_1,\ldots,x_n)]\!]^{\text{alg}}_{\bot_\varsigma,\beta} = \underline{t}_1 \neq \bot \text{ existiert} \\ & \quad \text{und } (\varsigma(\text{cond}_G(3) = \mathsf{ff} \text{ oder } \underline{t}_3 \neq \bot) \\ \underline{t}_3 & , \text{ andernfalls} \end{cases}$$

$$\text{sel}_{G,i}^{\mathcal{D}^{\text{fix}}_{P,\varsigma}}(\underline{t}) = \begin{cases} \beta(x_i) & , \text{ falls eine Variablenbelegung } \beta : \{x_1,\ldots,x_n\} \to \text{T}_{\mathcal{C},\varsigma} \\ & \quad \text{mit } [\![G(x_1,\ldots,x_n)]\!]^{\text{alg}}_{\bot_\varsigma,\beta} = \underline{t}_1 \neq \bot \text{ existiert} \\ \bot & , \text{ andernfalls} \end{cases}$$

für alle $\text{cond}_G, \text{sel}_{G,i} \in \mathcal{H}$ und $\underline{t}, \underline{t}_1, \underline{t}_2, \underline{t}_3 \in \text{T}_{\mathcal{C},\varsigma}$.
Unabhängig von der erzwungenen Striktheit ς sind die Operationen $\text{sel}_{G,i}^{\mathcal{D}^{\text{fix}}_{P,\varsigma}}$ immer und $\text{cond}_G^{\mathcal{D}^{\text{fix}}_{P,\varsigma}}$ an ihren ersten Argumentstellen strikt.

Beweis:
Dies folgt direkt aus der ς-Transformation und den Reduktionsregeln der Hilfssymbole unter Vereinfachung der Bedingungen des semantischen ς-Matchens bei Ausnutzung der Lemmata in 5.2.2.
□

5.2.2 Das semantische cbv- und cbn-Matchen

Da wir in Kapitel 4 den Begriff des semantischen ς-Matchens noch nicht definiert hatten, wollen wir nun zeigen, daß die bei den Definitionen 4.4, S. 57, und 4.8, S. 66, der cbv- respektive cbn-Transformation angegebenen Matchingbedingungen äquivalent zum semantischen cbv- respektive cbn-Matchen sind. Damit ist dann vollständig bewiesen, daß wir in Kapitel 4 zwei Spezialfälle der ς-Fixpunktsemantiken definiert haben.
Außerdem werden wir einige der hier bewiesenen Eigenschaften noch in Kapitel 6 benötigen.

Für das semantische cbn-Matchen läßt sich leicht die Äquivalenz mit der in Definition 4.8 der cbn-Transformation verwendeten Bedingung zeigen.

Lemma 5.2 Charakterisierung des semantischen cbn-Matchens
$\underline{\vec{t}} \in (\mathrm{T}^{\infty}_{\mathcal{C},\perp})^n$ ist genau dann mit einem Redexschema $f(\vec{p}) \in \mathrm{RedS}_P$ vermittels $\beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}^{\infty}_{\mathcal{C},\perp}$ semantisch cbn-matchbar, wenn für alle $i \in [n]$

$$[\![p_i]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}},\beta} = t_i$$

Beweis:

$\Rightarrow$: Folgt direkt aus der Definition des semantischen ς-Matchens.

$\Leftarrow$: Durch strukturelle Induktion wird gezeigt, daß für alle $c \in \mathrm{T}_{\mathcal{C}}(X)$ und $\beta : \mathrm{Var}(c) \to \mathrm{T}^{\infty}_{\mathcal{C},\perp}$

$$c[\perp/\mathrm{Var}(c)] \trianglelefteq [\![c]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}},\beta}$$

gilt.

$\underline{c = x}$: $(x \in X)$.
$x[\perp/x] = \perp \trianglelefteq \beta(x) = [\![x]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}},\beta}$.
$\underline{c = G(c_1, \ldots, c_n)}$: $(G^{(n)} \in \mathcal{C})$.

$$\begin{aligned} c[\perp/\mathrm{Var}(c)] &= G(c_1[\perp/\mathrm{Var}(c_1)], \ldots, c_n[\perp/\mathrm{Var}(c_n)]) \\ &\overset{\text{I.V.}}{\trianglelefteq} G([\![c_1]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}},\beta|_{\mathrm{Var}(c_1)}}, \ldots, [\![c_n]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}},\beta|_{\mathrm{Var}(c_n)}}) \\ &= G^{\perp_{\mathrm{cbn}}}(\ldots) \\ &= [\![c]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}},\beta} \end{aligned}$$

Somit ist für alle $i \in [n]$

$$p_i[\perp/\mathrm{Var}(p_i)] \trianglelefteq [\![p_i]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}},\beta}$$

Da trivialerweise auch f nicht erzwungen strikt für $\underline{\vec{t}}$ ist, ist $\underline{\vec{t}}$ mit $f(\vec{p})$ vermittels β semantisch cbn-matchbar.

□

Der entsprechende Beweis für das semantische cbv-Matchen benötigt erst einige Hilfsaussagen.

Lemma 5.3 Über Variablenbelegungen mit ⊥ für cbv
Sei $c \in \mathrm{T}_{\mathcal{C}}(X)$ und $\beta : \mathrm{Var}(c) \to \mathrm{T}_{\mathcal{C}}^{\perp}$ mit $\beta(x) = \perp$ für ein $x \in \mathrm{Var}(c)$. Dann ist

$$[\![c]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}},\beta} = \perp$$

Beweis:

$c = x$: $(x \in X)$.

$[\![x]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}},\beta} = \beta(x) = \perp$.

$c = G(c_1, \ldots, c_n)$: $(G^{(n)} \in \mathcal{C})$.

Nach Voraussetzung existiert $i \in [n]$ mit $x \in \mathrm{Var}(c_i)$, so daß $\beta(x) = \perp$. Nach Induktionsvoraussetzung ist somit

$$[\![c_i]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}},\beta|_{\mathrm{Var}(c_i)}} = \perp$$

Also gilt

$$[\![c]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}},\beta} = G^{\perp_{\mathrm{cbv}}}([\![c_1]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}},\beta|_{\mathrm{Var}(c_1)}}, \ldots, [\![c_n]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}},\beta|_{\mathrm{Var}(c_n)}}) = G^{\perp_{\mathrm{cbv}}}(\ldots, \perp, \ldots) = \perp$$

□

Lemma 5.4 Über semantische cbv-Matchbarkeit
Werde $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C}}^{\perp})^n$ mit dem Redexschema $f^{(n)}(\vec{p}) \in \mathrm{RedS}_P$, wobei f kein Verzweigungssymbol cond_G sei, vermittels einer Variablenbelegung $\beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C}}^{\perp}$ semantisch cbv-gematcht.
Dann ist $\beta(x) \neq \perp$ für alle $x \in \mathrm{Var}(\vec{p})$, d. h. $\beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C}}^{\perp} \setminus \{\perp\}$.

Beweis:
Angenommen, es existiert ein $x \in \mathrm{Var}(\vec{p})$ mit $\beta(x) = \perp$. Sei $i \in [n]$, so daß dieses $x \in \mathrm{Var}(p_i)$. Mit Lemma 5.3 über Variablenbelegungen mit ⊥ für cbv folgt $[\![p_i]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbv}},\beta} = \perp$, d. h. $t_i = \perp$. Dann kann jedoch im Widerspruch zur Voraussetzung aufgrund der erzwungenen Striktheit $\vec{\underline{t}}$ nicht mit $f(\vec{p})$ vermittels β semantisch cbv-matchbar sein. □

Die folgende Aussage gilt für beliebige erzwungene Striktheiten ς.

Lemma 5.5 Über Variablenbelegungen ohne ⊥
Sei $c \in \mathrm{T}_{\mathcal{C}}(X)$ und $\beta : \mathrm{Var}(c) \to \mathrm{T}_{\mathcal{C},\varsigma} \setminus \{\perp\}$. Dann ist

$$c[\perp/\mathrm{Var}(c)] \lhd [\![c]\!]^{\mathrm{alg}}_{\perp_\varsigma,\beta}.$$

Insbesondere ist also $[\![c]\!]^{\mathrm{alg}}_{\perp_\varsigma,\beta} \neq \perp$.

Beweis:

$c = x$: $(x \in X)$.

$c[\perp/\mathrm{Var}(c)] = \perp \lhd \beta(x) = [\![c]\!]^{\mathrm{alg}}_{\perp_\varsigma,\beta}$.

$\underline{c = G(c_1, \ldots, c_n)}$: $(G^{(n)} \in \mathcal{C})$.

$$\begin{aligned} c[\bot/\mathrm{Var}(c)] &= G(c_1[\bot/\mathrm{Var}(c_1), \ldots, c_n[\bot/\mathrm{Var}(c_n)]) \\ &\overset{\text{I.V.}}{\lhd} G([\![c_1]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta|_{\mathrm{Var}(c_1)}}, \ldots, [\![c_n]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta|_{\mathrm{Var}(c_n)}}) \\ &= G^{\bot_\varsigma}(\ldots) \\ &= [\![c]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta} \end{aligned}$$

□

Nun folgt das Lemma 5.2 entsprechende Lemma für semantisches cbv-Matchen. Es macht keine Aussage über die Verzweigungssymbole cond_G, da diese aufgrund der Nicht-Striktheit im 2. und 3. Argument eine Sonderrolle spielen. Die Äquivalenz des semantischen cbv-Matchens zu den in der Definition 4.4 der cbv-Transformation für die Verzweigungs- (und Selektions-)symbole separat aufgeführten Matchingbedingungen ist offensichtlich.

Lemma 5.6 Charakterisierung des semantischen cbv-Matchens
Sei $f^{(n)} \in \Sigma$ kein Verzweigungssymbol cond_G.
$\vec{t} \in (\mathrm{T}^{\bot}_{\mathcal{C}})^n$ ist genau dann mit einem Redexschema $f(\vec{p}) \in \mathrm{RedS}_P$ vermittels einer Variablenbelegung $\beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}^{\bot}_{\mathcal{C}}$ semantisch cbv-matchbar, wenn für alle $i \in [n]$

$$[\![p_i]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbv}},\beta} = t_i \neq \bot$$

Beweis:

$\Rightarrow$: Folgt direkt aus der Definition des semantischen ς-Matchens.

$\Leftarrow$: Mit Lemma 5.3 über Variablenbelegungen mit $\bot$ für cbv folgt aus der Voraussetzung $[\![p_i]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbv}},\beta} \neq \bot$ für alle $i \in [n]$, daß der Wertebereich von β $\mathrm{T}^{\bot}_{\mathcal{C}} \setminus \{\bot\}$ ist. Mit Lemma 5.5 über Variablenbelegungen ohne $\bot$ folgt dann für alle $i \in [n]$

$$p_i[\bot/\mathrm{Var}(p_i)] \trianglelefteq [\![p_i]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbv}},\beta}.$$

Somit ist $\vec{t}$ mit dem Redexschema $f(\vec{p})$ vermittels β semantisch cbv-matchbar.

□

5.2.3 Eigenschaften des semantischen ς-Matchens

Sowohl für den Beweis der Wohldefiniertheit der ς-Transformation in 5.2.4 als auch für den Beweis der Übereinstimmung der ς-Fixpunktsemantik mit den in 5.3 definierten ς-Reduktionssemantiken in 5.4 und 5.5 benötigen wir einige Aussagen über das semantische ς-Matchen, die wir hier beweisen.

Das erste Lemma stellt nur ein Hilfslemma für die folgenden, wichtigen Lemmata dar.

Lemma 5.7 Semantisches ς-Matchen mit linearen Pattern
Sei $\vec{\underline{t}} \in \mathrm{T}_{\mathcal{C},\varsigma}$, $p \in \mathrm{T}_{\mathcal{C}}(X)$ ein lineares Pattern und $\beta : X \to \mathrm{T}_{\mathcal{C},\varsigma}$. Es gelte

$$p[\bot/\mathrm{Var}(p)] \trianglelefteq \underline{t}. \tag{1}$$

Dann ist

$$[\![p]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta} = \underline{t} \tag{2}$$

genau dann, wenn

$$\beta(x) = \underline{t}/u \text{ , wobei } \{u\} = \mathrm{Occ}(x,p), \tag{3}$$

für alle $x \in \mathrm{Var}(p)$ ist.

Beweis:
Strukturelle Induktion über p.

$p = x$: $(x \in X)$.

Trivialerweise gilt (2):

$$[\![p]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta} = \beta(x) = \underline{t}$$

genau dann, wenn

$$\beta(x) = \underline{t}/\varepsilon \text{ , wobei } \{\varepsilon\} = \mathrm{Occ}(x,p),$$

gilt.

$p = G(p_1,\ldots,p_n)$: $(p_1,\ldots,p_n \in \mathrm{T}_{\mathcal{C}}(X)$ linear, $G^{(n)} \in \mathcal{C})$.

Aus Voraussetzung (1) folgt, daß

$$\underline{t} = G(\underline{t}_1,\ldots,\underline{t}_n) \tag{4}$$

mit $\underline{t}_1,\ldots,\underline{t}_n \in \mathrm{T}_{\mathcal{C},\varsigma}$ ist, und

$$\forall i \in [n].\ p_i[\bot/\mathrm{Var}(p)] \trianglelefteq \underline{t}_i \tag{5}$$

gilt.

Weil $\underline{t} \in \mathrm{T}_{\mathcal{C},\varsigma}$, folgt aus (4):

$$\forall i \in [n].\ (\varsigma(G)(i) = \mathsf{tt} \ \Rightarrow\ \underline{t}_i \neq \bot) \tag{6}$$

Es gilt nun

$$\underline{t} = [\![p]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta} \stackrel{\mathbf{def}}{=} G^{\bot_\varsigma}([\![p_1]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta},\ldots,[\![p_n]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta}) \tag{7}$$

genau dann, wenn

$$G^{\bot_\varsigma}([\![p_1]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta},\ldots,[\![p_n]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta}) = G([\![p_1]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta},\ldots,[\![p_n]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta}) \tag{8}$$

und

$$\forall i \in [n].\ [\![p_i]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta} = \underline{t}_i \tag{9}$$

gilt.

(8) läßt sich umformulieren zu der äquivalenten Bedingung

$$\forall i \in [n].\ (\varsigma(G)(i) = \mathsf{tt} \Rightarrow [\![p_i]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta} \neq \bot).$$

Dies wird jedoch schon von (9) zusammen mit (6) impliziert. Daher gelten (8) und (9), bzw. (7), genau dann, wenn (9) gilt.

Aufgrund von (5) kann die Induktionsvoraussetzung angewendet werden, und (9) gilt somit genau dann, wenn

$$\forall i \in [n].\ \forall x \in \mathrm{Var}(p_i).\ \beta(x) = \underline{t}_i/u \text{ , wobei } \{u\} = \mathrm{Occ}(x, p_i) \text{ ist.}$$

Dies gilt wiederum genau dann, wenn

$$\beta(x) = \underline{t}/u \text{ , wobei } \{u\} = \mathrm{Occ}(x, p),$$

für alle $x \in \mathrm{Var}(p)$ gilt.

□

Es ist zu beachten, daß aus $[\![p]\!]^{\mathrm{alg}}_{\perp_\varsigma,\beta} = \underline{t}$ nicht $p[\perp/\mathrm{Var}(p)] \trianglelefteq \underline{t}$ folgt, wie wir in Beispiel 5.1, S. 77, gesehen haben.

Lemma 5.8 Semantisches ς-Matchen
Sei $n \in \mathbb{N}$, $\underline{t}_1, \ldots, \underline{t}_n \in \mathrm{T}_{\mathcal{C},\varsigma}$, $f(p_1, \ldots, p_n) \in \mathrm{RedS}_P$ und $\beta : X \to \mathrm{T}_{\mathcal{C},\varsigma}$.
$\vec{\underline{t}}$ wird mit $f(\vec{p})$ von der Variablenbelegung β genau dann semantisch ς-gematcht, wenn

$$\forall i \in [n].\ (\varsigma(f)(i) = \mathtt{tt} \ \Rightarrow\ \underline{t}_i \neq \perp) \wedge p_i[\perp/\mathrm{Var}(p_i)] \trianglelefteq \underline{t}_i \tag{1}$$

und

$$\beta(x) = \vec{\underline{t}}/u \text{ , wobei } \{u\} = \mathrm{Occ}(x, \vec{p}), \tag{2}$$

für alle $x \in \mathrm{Var}(\vec{p})$ gilt.

Beweis:

$\Leftarrow$: Sei $i \in [n]$. Gemäß (1) gilt

$$p_i[\perp/\mathrm{Var}(p_i)] \trianglelefteq \underline{t}_i.$$

Aus (2) folgt

$$\beta(x) = \underline{t}_i/u' \text{ , wobei } \{u'\} = \mathrm{Occ}(x, p_i),$$

für alle $x \in \mathrm{Var}(p_i)$.

Mit Lemma 5.7 über das semantische ς-Matchen mit linearen Pattern folgt, daß

$$[\![p_i]\!]^{\mathrm{alg}}_{\perp_\varsigma,\beta} = \underline{t}_i.$$

Da dies für alle $i \in [n]$ gilt, folgt zusammen mit (1) gemäß der Definition des semantischen ς-Matchens, daß $\vec{\underline{t}}$ mit $f(\vec{p})$ von β semantisch ς-gematcht wird.

$\Rightarrow$: Da $\vec{\underline{t}}$ mit $f(\vec{p})$ von β semantisch ς-gematcht wird, gilt nach Definition des semantischen ς-Matchens (1) und es ist

$$[\![p_i]\!]^{\mathrm{alg}}_{\perp_\varsigma,\beta} = \underline{t}_i$$

für alle $i \in [n]$.

Daraus folgt mit Lemma 5.7 über semantisches ς-Matchen mit linearen Pattern, daß

$$\beta(x) = \underline{t}_i/u' \text{ , wobei } \{u'\} = \mathrm{Occ}(x, p_i),$$

für alle $x \in \mathrm{Var}(p_i)$ und $i \in [n]$ gilt. Somit ist auch (2) erfüllt.

□

Dies impliziert das folgende Korollar.

Korollar 5.9 Semantische ς-Matchbarkeit
$\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ ist genau dann mit $f(p_1,\ldots,p_n) \in \mathrm{RedS}_P$ semantisch ς-matchbar, wenn f nicht erzwungen strikt in $\vec{\underline{t}}$ ist und $\vec{\underline{t}}$ mit dem Pattern $\vec{p}$ matchbar ist.

Beweis:
Im vorhergehenden Lemma 5.8 über semantisches ς-Matchen stellt nur (1) Anforderungen an $\vec{\underline{t}}$ und $f(\vec{p})$, während Bedingung (2) nur die Variablenbelegung β festlegt. □

Für den Beweis der ω-Stetigkeit der ς-Transformation, 5.20, benötigen wir:

Lemma 5.10 ω-Stetigkeit des semantischen ς-Matchens
Sei $n \in \mathbb{N}$, $T = (\vec{\underline{t}}_j)_{j\in\mathbb{N}}$ eine ω-Kette von Termtupeln mit $\underline{t}_{i,j} \in \mathrm{T}_{\mathcal{C},\varsigma}$ und $f(p_1,\ldots,p_n) \in \mathrm{RedS}_P$. Sei $\vec{\hat{t}} := \bigsqcup T$.

1. Es existieren genau dann $\vec{\underline{t}}_k \in T$, das mit $f(\vec{p})$ semantisch ς-matchbar ist, wenn $\vec{\hat{t}}$ mit $f(\vec{p})$ semantisch ς-matchbar ist.
2. Wenn für alle $j \in \mathbb{N}$ $\vec{\underline{t}}_j \in T$ mit $f(\vec{p})$ von einer Substitution $\beta_j : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C},\varsigma}$ semantisch ς-gematcht wird, dann wird $\vec{\hat{\underline{t}}}$ mit $f(\vec{p})$ von der Substitution $\beta = \bigsqcup_{j\in\mathbb{N}} \beta_j : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C},\varsigma}$ semantisch ς-gematcht.

Beweis:

1., $\Rightarrow$: $\vec{\underline{t}}_k \in T$ sei mit $f(\vec{p})$ semantisch ς-matchbar. Nach Korollar 5.9 über semantische ς-Matchbarkeit gilt dann

$$\forall i \in [n].\ (\varsigma(f)(i) = \mathsf{tt} \ \Rightarrow\ \underline{t}_{i,k} \neq \bot) \wedge p_i[\bot/\mathrm{Var}(p_i)] \trianglelefteq \underline{t}_{i,k}$$

Es ist jedoch

$$\vec{\underline{t}}_k \trianglelefteq \bigsqcup T = \vec{\hat{t}}$$

und somit gilt

$$\forall i \in [n].\ (\varsigma(f)(i) = \mathsf{tt} \ \Rightarrow\ \hat{\underline{t}}_i \neq \bot) \wedge p_i[\bot/\mathrm{Var}(p_i)] \trianglelefteq \hat{\underline{t}}_i$$

Mit Korollar 5.9 folgt dann wiederum, daß $\vec{\hat{t}}$ mit $f(\vec{p})$ semantisch ς-matchbar ist.

1., $\Leftarrow$: $\vec{\hat{t}}$ sei mit $f(\vec{p})$ semantisch ς-matchbar. Nach Korollar 5.9 gilt dann

$$\forall i \in [n].\ (\varsigma(f)(i) = \mathsf{tt} \ \Rightarrow\ \hat{\underline{t}}_i \neq \bot) \wedge p_i[\bot/\mathrm{Var}(p_i)] \trianglelefteq \hat{\underline{t}}_i$$

Sei $i' \in [n]$ mit $\varsigma(f)(i') = \mathsf{tt}$. Somit ist $\hat{\underline{t}}_{i'} \neq \bot$. Da $\hat{\underline{t}}_{i'} = \bigsqcup_{j\in\mathbb{N}} \underline{t}_{i',j}$, existiert $k_{i'} \in \mathbb{N}$ mit $\underline{t}_{i',j} \neq \bot$ für alle $j \geq k_{i'}$. Also gilt für alle $j \geq k := \max\{k_{i'} \mid i' \in [n], \varsigma(f)(i') = \mathsf{tt}\}$:

$$\forall i \in [n].\ (\varsigma(f)(i) = \mathsf{tt} \ \Rightarrow \underline{t}_{i,j} \neq \bot).$$

Sei $i' \in [n]$ beliebig. Der Term $p_i[\bot/\mathrm{Var}(p_i)]$ ist endlich und somit ω-kompakt. Aus $p_i[\bot/\mathrm{Var}(p_i)] \trianglelefteq \hat{\underline{t}}_i = \bigsqcup_{j\in\mathbb{N}} t_{i,j}$ folgt nach der Definition der ω-Kompaktheit die Existenz eines $l_i \in \mathbb{N}$ mit $p_i[\bot/\mathrm{Var}(p_i)] \leq \underline{t}_{i,j}$ für alle $j \geq l_i$. Somit gilt für alle $j \geq l := \max\{l_i \mid i' \in [n]\}$:

$$\forall i \in [n].\ p_i[\bot/\mathrm{Var}(p_i)] \trianglelefteq \underline{t}_{i,j}.$$

Insgesamt ist also für alle $j \geq \max\{k,l\}$ f nicht erzwungen strikt für $\vec{\underline{t}}_j$ und $\vec{\underline{t}}_j$ mit dem Pattern $\vec{p}$ matchbar.

Also sind nach Korollar 5.9 alle $\vec{\underline{t}}_j$ mit $j \geq \max\{k,l\}$ mit $f(\vec{p})$ semantisch ς-matchbar.

2.: Für alle $j \in \mathbb{N}$ seien β_j die Variablenbelegungen, die $\vec{\underline{t}}_j$ mit $f(\vec{p})$ semantisch ς-matchen. Nach Lemma 5.8 über semantisches ς-Matchen ist

$$\beta_j(x) = \vec{\underline{t}}_j/u \text{ , wobei } \{u\} = \mathrm{Occ}(x,\vec{p}), \tag{1}$$

für alle $x \in \mathrm{Var}(\vec{p})$ und $j \in \mathbb{N}$.

Somit existiert $\bigsqcup_{j\in\mathbb{N}} \beta_j$, denn es gilt:

$$(\bigsqcup_{j\in\mathbb{N}} \beta_j)(x) = \bigsqcup_{j\in\mathbb{N}} (\beta_j(x)) \overset{(1)}{=} \bigsqcup_{j\in\mathbb{N}} (\vec{\underline{t}}_j/u) = (\bigsqcup_{j\in\mathbb{N}} \vec{\underline{t}}_j)/u = \vec{\hat{t}}/u \tag{2}$$

für alle $x \in \mathrm{Var}(\vec{p})$.

Aufgrund von Teil 1 dieses Lemmas wird $\vec{\hat{t}}$ mit $f(\vec{p})$ von einer Variablenbelegung β semantisch ς-gematcht. Nach Lemma 5.8 über semantisches ς-Matchen ist

$$\beta(x) = \vec{\hat{t}}/u \text{ , wobei } \{u\} = \mathrm{Occ}(x,\vec{p}), \tag{3}$$

für alle $x \in \mathrm{Var}(\vec{p})$.

Aus (2) und (3) folgt schließlich

$$\beta = \bigsqcup_{j\in\mathbb{N}} \beta_j.$$

□

Bemerkung 5.1:
Das semantische ς-Matchen von $\vec{\underline{t}}$ mit $f(\vec{p})$ vermittels β kann als partielle Abbildung

$$\begin{aligned} \mathbf{sem-\varsigma-match} &: (\mathrm{T}_{\mathcal{C},\varsigma})^n \times \mathrm{RedS}_P \nrightarrow (X \to \mathrm{T}_{\mathcal{C},\varsigma}) \\ &: (\vec{\underline{t}}, f(\vec{p})) \mapsto \beta \end{aligned}$$

aufgefaßt werden. Die Halbordnung derartiger Funktionen setzt sich aus der Halbordnung der partiellen Funktionen und der Halbordnung der punktweise geordneten Funktionen $X \to \mathrm{T}_{\mathcal{C},\varsigma}$ zusammen. □

5.2.4 Die Wohldefiniertheit

Hier zeigen wir endlich, daß die ς-Fixpunktsemantik — und somit auch insbesondere die in Kapitel 4 definierte cbv- und cbn-Fixpunktsemantik — wohldefiniert ist. Die Beweise der Wohldefiniertheit bestehen im einzelnen aus denen, die die ς-Interpretationen, die ω-Vollständigkeit der kanonischen Halbordnung der ς-Interpretationen, $\langle \mathrm{Int}_{\Sigma,\varsigma}, \sqsubseteq \rangle$, und die die ς-Transformation betreffen.

Für die Wohldefiniertheit der ς-Interpretation müssen wir zeigen, daß die festgelegten Konstruktoroperationen auch ω-stetig sind. Andernfalls würde überhaupt keine ς-Interpretation existieren.

Dafür benötigen wir jedoch zuerst folgendes Hilfslemma.

Lemma 5.11 **ω-Stetigkeit der erzwungenen Striktheit**
Sei $g^{(n)} \in \Sigma$, $T = (\vec{\underline{t}}_j)_{j\in\mathbb{N}}$ eine ω-Kette mit $\underline{t}_{i,j} \in \mathrm{T}_{\mathcal{C},\varsigma}$ und $\vec{\underline{t}} = \bigsqcup T$.
Das Operationssymbol g ist genau dann erzwungen strikt für $\vec{\underline{t}}$, wenn g für alle $\vec{\underline{t}}_j \in T$ erzwungen strikt ist.

Beweis:

Fall 1: g ist erzwungen strikt für $\vec{\underline{t}}$.

Demnach existiert ein $i \in [n]$ mit $\varsigma(g)(i) = \mathtt{tt}$ und $\underline{t}_i = \bot$. Da $\underline{t}_i = \bigsqcup_{j\in\mathbb{N}} \underline{t}_{i,j}$, ist auch für alle $j \in \mathbb{N}$ $\underline{t}_{i,j} = \bot$. Somit ist g auch für alle $\vec{\underline{t}}_j \in T$ erzwungen strikt.

Fall 2: g ist nicht erzwungen strikt für $\vec{\underline{t}}$.

Also ist für alle $i \in [n]$ mit $\varsigma(g)(i) = \mathtt{tt}$ $\underline{t}_i \neq \bot$. Sei $i' \in [n]$ mit $\varsigma(g)(i') = \mathtt{tt}$. Da $\underline{t}_{i'} = \bigsqcup_{j\in\mathbb{N}} \underline{t}_{i',j}$, existiert $k_{i'} \in \mathbb{N}$ mit $\underline{t}_{i',j} \neq \bot$ für alle $j \geq k_{i'}$. Somit gilt für alle $j \geq k := \max\{k_{i'} \mid i' \in [n],\ \varsigma(g)(i') = \mathtt{tt}\}$:

$$\forall i \in [n].\ (\varsigma(g)(i) = \mathtt{tt} \ \Rightarrow\ \underline{t}_{i,j} \neq \bot),$$

d. h. g ist nicht erzwungen strikt für alle $\vec{\underline{t}}_j \in T$ mit $j \geq k$.

□

Lemma 5.12 **ω-Stetigkeit der Konstruktoroperationen der ς-Interpretationen**
Die in den ς-Interpretationen definierten Operationen der Konstruktorsymbole sind ω-stetig.

Beweis:
Sei $G^{(n)} \in \mathcal{C}$. Sei $T = (\vec{\underline{t}}_j)_{j\in\mathbb{N}}$ eine ω-Kette mit $\underline{t}_{i,j} \in \mathrm{T}_{\mathcal{C},\varsigma}$ und $\vec{\underline{t}} = \bigsqcup T$. Sei $\mathfrak{A}$ eine ς-Interpretation.

Fall 1: G ist erzwungen strikt für $\vec{\underline{t}}$

Nach Lemma 5.11 über die ω-Stetigkeit der erzwungenen Striktheit ist G dann auch für alle $\vec{\underline{t}}_j \in T$ erzwungen strikt. Somit gilt

$$\bigsqcup G^{\mathfrak{A}}(T) = \bigsqcup\{\bot\} = \bot = G^{\mathfrak{A}}(\bigsqcup T)$$

Fall 2: G ist nicht erzwungen strikt für $\vec{t}$

Nach Lemma 5.11 über die ω-Stetigkeit (und somit auch Monotonie) der erzwungenen Striktheit existiert $k \in \mathbb{N}$, so daß G für alle $\underline{t}_j$ mit $j \geq k$ nicht erzwungen strikt ist. Also gilt

$$\begin{aligned}
\bigsqcup G^{\mathfrak{A}}(T) &= \bigsqcup\{\bigsqcup_{j<k} G^{\mathfrak{A}}(\vec{\underline{t}}_j), \bigsqcup_{j\geq k} G^{\mathfrak{A}}(\vec{\underline{t}}_j)\} \\
&= \bigsqcup_{j\geq k} G^{\mathfrak{A}}(\vec{\underline{t}}_j) \\
&= \bigsqcup_{j\geq k} G(\vec{\underline{t}}_j) = G(\bigsqcup_{j\geq k} \vec{\underline{t}}_j) \\
&= G^{\mathfrak{A}}(\bigsqcup_{j\geq k} \vec{\underline{t}}_j) = G^{\mathfrak{A}}(\vec{\underline{t}})
\end{aligned}$$

□

Lemma 5.13 ω-Vollständigkeit von $\langle \mathbf{Int}_{\Sigma,\varsigma}, \sqsubseteq \rangle$
Die kanonische Halbordnung aller ς-Interpretationen $\langle \mathrm{Int}_{\Sigma,\varsigma}, \sqsubseteq \rangle$ ist ω-vollständig.

Beweis:
Sei $T \subseteq \mathrm{Int}_{\Sigma,\varsigma}$ ein abzählbare gerichtete Menge. Die aufgrund der ω-Vollständigkeit der Halbordnung der ω-stetigen Operationen der Interpretationen durch

$$g^{\mathfrak{A}} := \bigsqcup\{g^{\mathfrak{B}} \mid \mathfrak{B} \in T\}$$

für alle $g \in \Sigma$ wohldefinierte Algebra $\mathfrak{A} \in \mathrm{Alg}^{\infty}_{\Sigma,\perp}$ ist die kleinste obere Schranke von T.
Die Konstruktoroperationen sind für alle ς-Interpretationen gleich, und daher ist

$$G^{\mathfrak{A}} = \bigsqcup\{G^{\mathfrak{B}} \mid \mathfrak{B} \in T\} = G^{\mathfrak{B}}$$

für alle $G \in \mathcal{C}$ und $\mathfrak{B} \in T$.
Sei $g^{(n)} \in \Sigma, \vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ und g erzwungen strikt für $\vec{\underline{t}}$. Dann ist

$$g^{\mathfrak{B}}(\vec{\underline{t}}) = \perp$$

für alle $\mathfrak{B} \in T$, und somit auch

$$g^{\mathfrak{A}}(\vec{\underline{t}}) = \bigsqcup\{g^{\mathfrak{B}}(\vec{\underline{t}}) \mid \mathfrak{B} \in T\} = \bigsqcup\{\perp\} = \perp.$$

Also ist $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$. □

Der Beweis der Wohldefiniertheit der ς-Transformation ist am aufwendigsten. Er erfordert das Beweisen vieler Einzelaussagen.
Zuerst zeigen wir, daß die ς-Transformation die Funktions- und Hilfsoperationen trotz eventuell überlappender linker Regelseiten und somit mehrerer zu einem Argumenttupel semantisch ς-matchbarer Redexschemata eindeutig definiert. Diese semantische Aussage steht in Beziehung zur syntaktischen bzw. operationellen Eigenschaft der Konfluenz.

Lemma 5.14 Semantische Unifikation impliziert syntaktische Unifikation
Seien $p, p' \in \mathrm{T}_{\mathcal{C}}(X)$ zwei lineare Pattern, die zueinander variablendisjunkt sind, d. h. es gilt $\mathrm{Var}(p) \cap \mathrm{Var}(p') = \emptyset$.
Sei $\underline{t} \in \mathrm{T}_{\mathcal{C},\varsigma}$ und seien $\beta : \mathrm{Var}(p) \to \mathrm{T}_{\mathcal{C},\varsigma}$ und $\beta' : \mathrm{Var}(p') \to \mathrm{T}_{\mathcal{C},\varsigma}$ zwei Variablenbelegungen. Es gelte

$$p[\bot/\mathrm{Var}(p)] \trianglelefteq \underline{t} \qquad\qquad p'[\bot/\mathrm{Var}(p')] \trianglelefteq \underline{t} \tag{1}$$

$$[\![p]\!]^{\mathrm{alg}}_{\bot\varsigma,\beta} = \underline{t} = [\![p']\!]^{\mathrm{alg}}_{\bot\varsigma,\beta'} \tag{2}$$

Dann existieren zwei Substitutionen

$$\begin{array}{rcl} \sigma & : & \mathrm{Var}(p) \to \mathrm{T}_{\mathcal{C}}(\mathrm{Var}(p) \dot\cup \mathrm{Var}(p')) \\ \sigma' & : & \mathrm{Var}(p') \to \mathrm{T}_{\mathcal{C}}(\mathrm{Var}(p) \dot\cup \mathrm{Var}(p')) \end{array}$$

und eine Variablenbelegung

$$\hat\beta : (\mathrm{Var}(p) \dot\cup \mathrm{Var}(p')) \to \mathrm{T}_{\mathcal{C},\varsigma},$$

so daß gilt:

$$\begin{array}{rcll} p\sigma & = & p'\sigma' & \\ \beta(x) & = & [\![x\sigma]\!]^{\mathrm{alg}}_{\bot\varsigma,\hat\beta} & \text{für alle } x \in \mathrm{Var}(p) \\ \beta'(x) & = & [\![x\sigma']\!]^{\mathrm{alg}}_{\bot\varsigma,\hat\beta} & \text{für alle } x \in \mathrm{Var}(p') \end{array}$$

Beweis:
Parallele strukturelle Induktion über p und p'.

Fall 1: $p \in X$ oder $p' \in X$, o. B. d. A. sei $p = x \in X$.

Man definiert

$$\begin{array}{rcll} \sigma & := & [p'/x] & \\ \sigma' & := & [] & \\ \hat\beta(x) & := & \beta'(x) & \text{für alle } x \in \mathrm{Var}(p) \dot\cup \mathrm{Var}(p') \end{array}$$

und es gilt

$$p\sigma = x[p'/x] = p' = p'[] = p'\sigma'$$

$$\beta(x) = [\![p]\!]^{\mathrm{alg}}_{\bot\varsigma,\hat\beta} = [\![p']\!]^{\mathrm{alg}}_{\bot\varsigma,\hat\beta'} = [\![x\sigma]\!]^{\mathrm{alg}}_{\bot\varsigma,\hat\beta'} = [\![x\sigma]\!]^{\mathrm{alg}}_{\bot\varsigma,\hat{\hat\beta}} \text{ für alle } x \in \mathrm{Var}(p)$$

$$\beta'(x) = \beta'(x\sigma') = [\![x\sigma']\!]^{\mathrm{alg}}_{\bot\varsigma,\hat\beta'} = [\![x\sigma']\!]^{\mathrm{alg}}_{\bot\varsigma,\hat{\hat\beta}} \text{ für alle } x \in \mathrm{Var}(p')$$

Fall 2: $p = G(p_1, \ldots, p_n)$ und $p' \notin X$.

Mit (1) folgt

$$\underline{t} = G(\underline{t}_1, \ldots, \underline{t}_n) \tag{3}$$

und

$$\forall i \in [n].\ p_i[\bot/\mathrm{Var}(p_i)] \trianglelefteq \underline{t}_i \tag{4}$$

Aus

$$G^{\mathfrak{A}}([\![p_1]\!]^{\mathrm{alg}}_{\bot\varsigma,\hat\beta}, \ldots, [\![p_n]\!]^{\mathrm{alg}}_{\bot\varsigma,\hat\beta}) = [\![p]\!]^{\mathrm{alg}}_{\bot\varsigma,\hat\beta} \overset{(2)}{=} \underline{t} \overset{(3)}{=} G(\underline{t}_1, \ldots, \underline{t}_n) \tag{5}$$

folgt

$$\forall i \in [n].\ [\![p_i]\!]^{\mathrm{alg}}_{\bot\varsigma,\hat\beta} = \underline{t}_i \tag{6}$$

Aus

$$[\![p']\!]^{\mathrm{alg}}_{\perp_\varsigma,\hat{\beta}'} = [\![p]\!]^{\mathrm{alg}}_{\perp_\varsigma,\hat{\beta}} \overset{(5)}{=} G(\underline{t}_1,\ldots,\underline{t}_n)$$

folgt

$$p' = G(p'_1,\ldots,p'_n)$$

und

$$\forall i \in [n].\ [\![p'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma,\hat{\beta}} = \underline{t}_i \tag{7}$$

Außerdem folgt mit (1):

$$\forall i \in [n].\ p_i[\perp/\mathrm{Var}(p_i)] \trianglelefteq \underline{t}_i \tag{8}$$

Aufgrund von (4), (8), (6) und (7) läßt sich die Induktionsvoraussetzung anwenden. Für alle $i \in [n]$ existieren

$$\begin{array}{rcl} \sigma_i & : & \mathrm{Var}(p_i)\to \mathrm{T}_{\mathcal{C}}(\mathrm{Var}(p_i) \dot\cup \mathrm{Var}(p'_i)) \\ \sigma'_i & : & \mathrm{Var}(p'_i)\to \mathrm{T}_{\mathcal{C}}(\mathrm{Var}(p) \dot\cup \mathrm{Var}(p'_i)) \\ \hat\beta_i & : & (\mathrm{Var}(p_i) \dot\cup \mathrm{Var}(p'_i))\to \mathrm{T}_{\mathcal{C},\varsigma} \end{array}$$

mit

$$\begin{array}{rcll} p_i\sigma_i & = & p'_i\sigma'_i & \\ \beta_i(x) & = & [\![x\sigma_i]\!]^{\mathrm{alg}}_{\perp_\varsigma,\hat\beta_i} & \text{für alle } x \in \mathrm{Var}(p_i) \\ \beta'_i(x) & = & [\![x\sigma'_i]\!]^{\mathrm{alg}}_{\perp_\varsigma,\hat\beta_i} & \text{für alle } x \in \mathrm{Var}(p'_i) \end{array}$$

Aufgrund der Linearität der Pattern und der Variablendisjunktheit von p und p' können wir nun definieren:

$$\begin{array}{rcl} \sigma & := & \dot\bigcup_{i\in[n]} \sigma_i : \mathrm{Var}(p)\to \mathrm{T}_{\mathcal{C}}(\mathrm{Var}(p) \dot\cup \mathrm{Var}(p')) \\ \sigma' & := & \dot\bigcup_{i\in[n]} \sigma'_i : \mathrm{Var}(p')\to \mathrm{T}_{\mathcal{C}}(\mathrm{Var}(p) \dot\cup \mathrm{Var}(p')) \\ \hat\beta & := & \dot\bigcup_{i\in[n]} (\mathrm{Var}(p) \dot\cup \mathrm{Var}(p'))\to \mathrm{T}_{\mathcal{C},\varsigma} \end{array}$$

und es gilt

$$p\sigma = G(p_1\sigma_1,\ldots,p_n\sigma_n) = G(p'_1\sigma'_1,\ldots,p'_n\sigma'_n) = p'\sigma'$$

$$\begin{array}{llllllll} \forall i \in [n]. & \forall x \in \mathrm{Var}(p_i). & \beta(x) & = & \beta_i(x) & = & [\![x\sigma]\!]^{\mathrm{alg}}_{\perp_\varsigma,\hat\beta_i} & = & [\![x\sigma]\!]^{\mathrm{alg}}_{\perp_\varsigma,\hat\beta} \\ \forall i \in [n]. & \forall x \in \mathrm{Var}(p'_i). & \beta'(x) & = & \beta'_i(x) & = & [\![x\sigma']\!]^{\mathrm{alg}}_{\perp_\varsigma,\hat\beta_i} & = & [\![x\sigma']\!]^{\mathrm{alg}}_{\perp_\varsigma,\hat\beta} \end{array}$$

□

Lemma 5.15 **Eindeutige Definiertheit der Funktions- und Hilfsoperationen in der ς-Transformation**

Sei $n \in \mathbb{N}$. Wird das Termtupel $\vec{t} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ mit der linken Seite einer Reduktionsregel $f(p_1,\ldots,p_n)\to r$ von einer Variablenbelegung $\beta : \mathrm{Var}(\vec{p})\to \mathrm{T}_{\mathcal{C},\varsigma}$ und mit der linken Seite einer Reduktionsregel $f(p'_1,\ldots,p'_n)\to r'$ von einer Variablenbelegung $\beta' : \mathrm{Var}(\vec{p'})\to \mathrm{T}_{\mathcal{C},\varsigma}$ semantisch ς-gematcht, dann ist für beliebige Interpretationen $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$:

$$[\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![r']\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}$$

Beweis:
O. B. d. A. seien $\vec{p}$ und $\vec{p}'$ variablendisjunkt. Aus der semantischen ς-Matchbarkeit folgt gemäß dessen Definition für alle $i \in [n]$:

$$p_i[\bot/\mathrm{Var}(p_i)] \trianglelefteq \underline{t}_i \qquad\qquad p'_i[\bot/\mathrm{Var}(p'_i)] \trianglelefteq \underline{t}_i$$

$$[\![p_i]\!]^{\mathrm{alg}}_{\bot\varsigma,\beta} = \underline{t} = [\![p'_i]\!]^{\mathrm{alg}}_{\bot\varsigma,\beta'}$$

Da nach Lemma 5.14 semantische Unifikation syntaktische Unifikation impliziert, existieren für alle $i \in [n]$

$$\begin{array}{rcl} \sigma_i & : & \mathrm{Var}(p_i) \to \mathrm{T}_{\mathcal{C}}(\mathrm{Var}(p_i) \dot\cup \mathrm{Var}(p'_i)) \\ \sigma'_i & : & \mathrm{Var}(p'_i) \to \mathrm{T}_{\mathcal{C}}(\mathrm{Var}(p) \dot\cup \mathrm{Var}(p'_i)) \\ \hat\beta_i & : & (\mathrm{Var}(p_i) \dot\cup \mathrm{Var}(p'_i)) \to \mathrm{T}_{\mathcal{C},\varsigma} \end{array}$$

mit

$$\begin{array}{rcll} p_i\sigma_i & = & p'_i\sigma'_i & \\ \beta_i(x) & = & [\![x\sigma_i]\!]^{\mathrm{alg}}_{\bot_\varsigma,\hat\beta_i} & \text{für alle } x \in \mathrm{Var}(p_i) \\ \beta'_i(x) & = & [\![x\sigma'_i]\!]^{\mathrm{alg}}_{\bot_\varsigma,\hat\beta_i} & \text{für alle } x \in \mathrm{Var}(p'_i) \end{array}$$

Wie im Beweis von Lemma 5.14 können wir nun

$$\sigma := \dot{\bigcup_{i\in[n]}} \sigma_i, \sigma' := \dot{\bigcup_{i\in[n]}} \sigma'_i \text{ und } \hat\beta := \dot{\bigcup_{i\in[n]}} \hat\beta_i$$

definieren, und es gilt:

$$f(p_1,\ldots,p_n)\sigma = f(p_1\sigma_1,\ldots,p_n\sigma_n) = f(p'_1\sigma'_1,\ldots,p'_n\sigma'_n) = f(p'_1,\ldots,p'_n)\sigma' \tag{1}$$

$$\begin{array}{rcll} \beta(x) & = & [\![x\sigma]\!]^{\mathrm{alg}}_{\bot_\varsigma,\hat\beta} & \text{für alle } x \in \mathrm{Var}(\vec{p}) \\ \beta'(x) & = & [\![x\sigma']\!]^{\mathrm{alg}}_{\bot_\varsigma,\hat\beta} & \text{für alle } x \in \mathrm{Var}(\vec{p}') \end{array} \tag{2}$$

Aus (1) folgt mit der Eindeutigkeitsbedingung beinahe orthogonaler Termersetzungssysteme

$$r\sigma = r\sigma' \tag{3}$$

Sei $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ eine beliebige ς-Interpretation. Da für alle $x \in \mathrm{Var}(\vec{p})$ $x\sigma \in \mathrm{T}_{\mathcal{C}}(\mathrm{Var}(\vec{p}) \cup \mathrm{Var}(\vec{p}'))$ und für alle $x \in \mathrm{Var}(\vec{p}')$ $x\sigma \in \mathrm{T}_{\mathcal{C}}(\mathrm{Var}(\vec{p}) \cup \mathrm{Var}(\vec{p}'))$, gilt offensichtlich

$$\begin{array}{rcl} \forall x \in \mathrm{Var}(\vec{p}). \quad [\![x\sigma]\!]^{\mathrm{alg}}_{\bot_\varsigma,\hat\beta} & = & [\![x\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A},\hat\beta} \\ \forall x \in \mathrm{Var}(\vec{p}'). \quad [\![x\sigma']\!]^{\mathrm{alg}}_{\bot_\varsigma,\hat\beta} & = & [\![x\sigma']\!]^{\mathrm{alg}}_{\mathfrak{A},\hat\beta} \end{array} \tag{4}$$

Weil $r, r' \in \mathrm{T}_\Sigma(\mathrm{Var}(\vec{p}) \dot\cup \mathrm{Var}(\vec{p'}))$, folgt aus (2) und (4) mit Lemma 4.2 über Substitution und algebraische Semantik

$$[\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A},\hat\beta} \text{ und } [\![r']\!]^{\mathrm{alg}}_{\mathfrak{A},\beta'} = [\![r'\sigma']\!]^{\mathrm{alg}}_{\mathfrak{A},\hat\beta} \tag{5}$$

Insgesamt gilt somit

$$[\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \overset{(5)}{=} [\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A},\hat\beta} \overset{(3)}{=} [\![r'\sigma']\!]^{\mathrm{alg}}_{\mathfrak{A},\hat\beta} \overset{(5)}{=} [\![r']\!]^{\mathrm{alg}}_{\mathfrak{A},\beta'}$$

□

Für die nächsten, aber auch für spätere Beweise benötigen wir die ω-Stetigkeit der algebraischen Semantik sowohl bezüglich der Algebra als auch der Variablenbelegung als Parameter.

Lemma 5.16 ω-Stetigkeit der algebraischen Termsemantik bzgl. der Algebra
Sei Σ eine Signatur und $\langle A, \leq\rangle$ eine ω-vollständige Halbordnung. Sei $I \subseteq \mathrm{Alg}^{\infty}_{\Sigma,\perp}(\langle A, \leq\rangle)$, so daß die kanonische Halbordnung $\langle I, \sqsubseteq\rangle$ ω-vollständig ist. Seien $t \in \mathrm{T}_\Sigma(X)$ und $\beta : X \to A$.
Dann ist die algebraische Termsemantik $[\![t]\!]^{\mathrm{alg}}_{\cdot,\beta} : I \to A$ eine ω-vollständige Abbildung bezüglich der Algebra aus I.

Beweis:
Sei $K \subseteq I$ eine ω-Kette. Wir zeigen durch strukturelle Induktion über $t \in \mathrm{T}_\Sigma(X)$, daß

$$e := \bigsqcup \{[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \mid \mathfrak{A} \in K\}$$

existiert, und

$$e = [\![t]\!]^{\mathrm{alg}}_{\sqcup K,\beta}$$

ist.

$t = x$: $(x \in X)$.

Es ist

$$\begin{aligned} T &:= \{[\![x]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \mid \mathfrak{A} \in K\} \\ &= \{\beta(x) \mid \mathfrak{A} \in K\} \\ &= \{\beta(x)\} \end{aligned}$$

Somit existiert $\bigsqcup T$ und es ist

$$\bigsqcup T = \beta(x) = [\![x]\!]^{\mathrm{alg}}_{\sqcup K,\beta}$$

$t = f(t_1, \ldots, t_n)$: $(f^{(n)} \in \Sigma)$.

Es gilt:

$$\begin{aligned} & & & [\![f(t_1, \ldots, t_n)]\!]^{\mathrm{alg}}_{\sqcup K,\beta} \\ & \text{(Def. der alg. Semantik)} & = & f^{\sqcup K}([\![t_1]\!]^{\mathrm{alg}}_{\sqcup K,\beta}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\sqcup K,\beta}) \\ & & = & ((\bigsqcup\{\alpha \mid \langle A, \leq, \alpha\rangle \in K\})(f))(\ldots) \\ & (\omega\text{-Vollst. von } \langle \Sigma_n \to \mathrm{Ops}_n, \preceq\rangle) & = & (\bigsqcup\{f^{\mathfrak{A}} \mid \mathfrak{A} \in K\})(\ldots) \\ & (\omega\text{-Vollst. von } \langle [A^n \to A], \preceq\rangle) & = & \bigsqcup\{f^{\mathfrak{A}}(\ldots) \mid \mathfrak{A} \in K\} \\ & \text{(I.V.)} & = & \bigsqcup\{f^{\mathfrak{A}}(\bigsqcup\{[\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_1,\beta} \mid \mathfrak{A}_1 \in K\}, \ldots) \mid \mathfrak{A} \in K\} \\ & (\omega\text{-Stet. der } f^{\mathfrak{A}} \in \mathrm{Ops}_n) & = & \bigsqcup\{f^{\mathfrak{A}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_1,\beta}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}_n,\beta}) \mid \mathfrak{A}, \mathfrak{A}_1, \ldots, \mathfrak{A}_n \in K\} \\ & & = & \bigsqcup T_1 \end{aligned}$$

wobei $T_1 := \{f^{\mathfrak{A}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_1,\beta}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}_n,\beta}) \mid \mathfrak{A}, \mathfrak{A}_1, \ldots, \mathfrak{A}_n \in K\}$.

Außerdem ist

$$T_2 := \{[\![f(t_1, \ldots, t_n)]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = \{f^{\mathfrak{A}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}) \mid \mathfrak{A} \in K\}$$

Sei nun $e_1 = f^{\mathfrak{A}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_1,\beta}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}_n,\beta}) \mid \mathfrak{A} \in K\} \in T_1$ beliebig. Da K eine ω-Kette ist, existiert $\mathfrak{A}' \in I$ mit $\{\mathfrak{A}, \mathfrak{A}_1, \ldots, \mathfrak{A}_n\} \leq \mathfrak{A}'$. Nach Induktionsvoraussetzung ist die algebraische Semantik

der Teilterme $t_1, \ldots, t_n$ ω-stetig und somit auch monoton bezüglich der Variablenbelegung, d. h. es gilt:

$$[\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_1,\beta} \leq [\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}',\beta}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}_n,\beta} \leq [\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}',\beta}$$

Auch folgt aus $\mathfrak{A} \leq \mathfrak{A}'$, daß $f^{\mathfrak{A}} \preceq f^{\mathfrak{A}'}$, und somit gilt insgesamt:

$$e_1 \leq f^{\mathfrak{A}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}',\beta}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}',\beta}) \leq f^{\mathfrak{A}'}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}',\beta}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}',\beta}) =: e_2$$

Es ist $e_2 \in T_2$.

Somit ist T_1 kofinal in T_2. Außerdem ist T_2 kofinal in T_1, da $T_2 \subseteq T_1$. Nach Lemma 2.1, S. 12, über Kofinalität und kleinste obere Schranken existiert also $\bigsqcup T_2$ und es ist

$$\bigsqcup T_2 = \bigsqcup T_1 = [\![f(t_1, \ldots, t_n)]\!]^{\mathrm{alg}}_{\sqcup K,\beta}$$

□

Fast analog verläuft der Beweis für das folgende Lemma.

Lemma 5.17 **ω-Stetigkeit der algebraischen Termsemantik bezüglich der Variablenbelegung**

Sei Σ eine Signatur und $\langle A, \leq\rangle$ eine ω-vollständige Halbordnung. Seien $Y \subseteq X$, $t \in \mathrm{T}_\Sigma(Y)$ und $\mathfrak{A} \in \mathrm{Alg}^{\infty}_{\Sigma,\bot}(\langle A, \leq\rangle)$.

Dann ist die algebraische Termsemantik $[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\cdot} : (Y{\to}A){\to}A$ eine ω-stetige Abbildung bezüglich der Variablenbelegung aus $Y{\to}A$.

Beweis:
Sei $K \subseteq (Y{\to}A)$ eine ω-Kette der kanonischen Halbordnung $\langle (Y{\to}A), \preceq\rangle$. Wir zeigen durch strukturelle Induktion über $t \in \mathrm{T}_\Sigma(Y)$, daß

$$e := \bigsqcup \{[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \mid \beta \in K\}$$

existiert, und

$$e = [\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\sqcup K}$$

ist.

$\underline{t = x}$: $(x \in Y)$.

Es ist

$$T := \{[\![x]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \mid \beta \in K\} = \{\beta(x) \mid \beta \in K\}$$

Da K eine ω-Kette ist, ist auch $T \subseteq A$ eine ω-Kette. Somit besitzt T eine kleinste obere Schranke, und aufgrund der ω-Vollständigkeit von $\langle (Y{\to}A), \leq\rangle$ gilt:

$$\bigsqcup T = (\bigsqcup \{\beta \mid \beta \in K\})(x) = [\![x]\!]^{\mathrm{alg}}_{\mathfrak{A},\sqcup K}$$

$\underline{t = f(t_1, \ldots, t_n)}$: $(f^{(n)} \in \Sigma)$.

Es gilt:

$$\begin{aligned}
& [\![f(t_1,\ldots,t_n)]\!]^{\mathrm{alg}}_{\mathfrak{A},\bigsqcup K} \\
= \;& f^{\mathfrak{A}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\bigsqcup K},\ldots,[\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A},\bigsqcup K} \\
\text{(I.V.)} \;= \;& f^{\mathfrak{A}}(\bigsqcup\{[\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_1} \mid \beta_1 \in K\},\ldots) \\
(\omega\text{-Stetigkeit von } f^{\mathfrak{A}} \in \mathrm{Ops}_n) \;= \;& \bigsqcup\{f^{\mathfrak{A}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_1},\ldots,[\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_n}) \mid \beta_1,\ldots,\beta_n \in K\} \\
= \;& \bigsqcup T_1
\end{aligned}$$

wobei $T_1 := \{f^{\mathfrak{A}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_1},\ldots,[\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_n}) \mid \beta_1,\ldots,\beta_n \in K\}$.

Außerdem ist

$$T_2 := \{[\![f(t_1,\ldots,t_n)]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \mid \beta \in K\} = \{f^{\mathfrak{A}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta},\ldots,[\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}) \mid \beta \in K\}$$

Sei nun $e_1 := f^{\mathfrak{A}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_1},\ldots,[\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_n}) \in T_1$ beliebig. Da K eine ω-Kette ist, existiert $\beta' \in (Y{\to}A)$ mit $\{\beta_1,\ldots,\beta_n\} \leq \beta'$.

Nach Induktionsvoraussetzung ist die algebraische Semantik der Teilterme $t_1,\ldots,t_n$ ω-stetig und somit auch monoton bezüglich der Variablenbelegung, d. h. es gilt:

$$[\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_1} \leq [\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta'},\ldots,[\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_n} \leq [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta'}$$

Zusammen mit der Monotonie der Operationen $f^{\mathfrak{A}}$ folgt:

$$e_1 \leq f^{\mathfrak{A}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta'},\ldots,[\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta'}) =: e_2$$

Es ist $e_2 \in T_2$.

Somit ist T_1 kofinal in T_2. Außerdem ist T_2 kofinal in T_1, da $T_2 \subseteq T_1$. Nach Lemma 2.1, S. 12, über Kofinalität und kleinste obere Schranken existiert damit $\bigsqcup T_2$ und es ist

$$\bigsqcup T_2 = \bigsqcup T_1 = [\![f(t_1,\ldots,t_n)]\!]^{\mathrm{alg}}_{\mathfrak{A},\bigsqcup K}$$

□

Eine Abbildung $\varphi : A^n{\to}A$ zu $t \in \mathrm{T}_\Sigma(\{x_1,\ldots,x_n\})$ und $\mathfrak{A} = \langle A,\leq,\alpha\rangle \in \mathrm{Alg}^{\infty}_{\Sigma,\bot}(\langle A,\leq\rangle)$, definiert durch

$$\varphi(a_1,\ldots,a_n) := [\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \text{ mit } \beta(x_i) := a_i \text{ für alle } i \in [n]$$

wird auch **abgeleitete Operation** (derived operation) genannt. Wir haben gerade bewiesen, daß abgeleitete Operationen genauso wie Operationen ω-stetig sind.

Wir zeigen nun, daß die durch die ς-Transformation definierten Funktions- und Hilfsoperationen ω-stetig sind. Daraus folgt direkt, daß das Ergebnis einer ς-Transformation immer eine ς-Interpretation ist. Zum Schluß beweisen wir die ω-Stetigkeit der ς-Transformation.
Auf diese Weise ist dann insgesamt bewiesen, daß $\Phi_{P,\varsigma}$ wirklich von dem angegebenen Typ $[\mathrm{Int}_{\Sigma,\varsigma}{\to}\mathrm{Int}_{\Sigma,\varsigma}]$ ist.

Lemma 5.18 **ω-Stetigkeit der Funktions- und Hilfssymboloperationen nach einer ς-Transformation**

Sei $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ und $\mathfrak{A}' := \Phi_{P,\varsigma}(\mathfrak{A})$. Sei $f \in \mathcal{F}(\dot\cup\,\mathcal{H})$.
Dann ist $f^{\mathfrak{A}'}$ ω-stetig.

Beweis:
Sei $f^{(n)} \in \mathcal{F}(\dot\cup\, \mathcal{H})$. Sei $T = (\vec{\underline{t}}_j)_{j\in\mathbb{N}}$ eine ω-Kette mit $\underline{t}_{i,j} \in \mathrm{T}_{\mathcal{C},\varsigma}$. Sei $\vec{\underline{t}} := \bigsqcup T$.
Es ist zu zeigen, daß

$$\bigsqcup(f^{\mathfrak{A}}(T)) =: e$$

existiert und

$$e = f^{\mathfrak{A}'}(\vec{\underline{t}})$$

ist.
Sei k die kleinste natürliche Zahl, so daß sich $\vec{\underline{t}}_k \in T$ mit der linken Seite einer Reduktionsregel $f(p_1,\ldots,p_n)\to r \in P$ semantisch ς-matchen läßt.

Fall 1: k existiert nicht.

Es läßt sich also kein $\vec{\underline{t}}_j \in T$ mit irgendeinem Redexschema des Programms semantisch ς-matchen. Mit Lemma 5.10, S. 84, über die ω-Stetigkeit des semantischen ς-Matchens folgt, daß sich auch $\vec{\underline{t}}$ mit keinem Redexschema $f(p_1,\ldots,p_n) \in \mathrm{RedS}_P$ semantisch ς-matchen läßt.

Aus der Definition der ς-Transformation folgt dann

$$\bigsqcup(f^{\mathfrak{A}}(T)) = \bigsqcup \bot = \bot = f^{\mathfrak{A}'}(\vec{\underline{t}})$$

Fall 2: $k \in \mathbb{N}$ existiert.

Mit Lemma 5.10 über die ω-Stetigkeit und somit auch Monotonie des semantischen ς-Matchens folgt, daß sich für alle $j \geq k$ $\vec{\underline{t}}_j \in T$ mit $f(p_1,\ldots,p_n)$ von einer Variablenbelegung β_j semantisch ς-matchen lassen, und daß sich $\vec{\underline{t}}$ mit $f(p_1,\ldots,p_n)$ von der Variablenbelegung $\beta = \bigsqcup_{j\geq k} \beta_j$ semantisch ς-matchen läßt.

Aus der Definition der ς-Transformation folgt damit

$$\begin{aligned}
\bigsqcup(f^{\mathfrak{A}'}(T)) &= \bigsqcup\{\bigsqcup_{j<k}(f^{\mathfrak{A}'}(\vec{\underline{t}}_j)), \bigsqcup_{j\geq k}(f^{\mathfrak{A}'}(\vec{\underline{t}}_j))\} \\
&= \bigsqcup\{\bigsqcup \bot, \bigsqcup_{j\geq k} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_j}\} \\
&= \bigsqcup_{j\geq k} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_j} \\
&\overset{(*)}{=} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\bigsqcup_{j\geq k}\beta_j} = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = f^{\mathfrak{A}'}(\vec{\underline{t}})
\end{aligned}$$

Der Schritt (*) ist aufgrund der ω-Stetigkeit der algebraischen Semantik bezüglich der Variablenbelegung gemäß Lemma 5.17, S. 92, gültig.

□

Korollar 5.19 $\mathrm{Int}_{\Sigma,\varsigma}$ ist die Wertemenge der ς-Transformation
Sei $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$. Dann ist $\Phi_{P,\varsigma}(\mathfrak{A}) \in \mathrm{Int}_{\Sigma,\varsigma}$.

Beweis:
Sei $\mathfrak{A}' := \Phi_{P,\varsigma}(\mathfrak{A})$. Für alle $f^{(n)} \in \mathcal{F}(\dot\cup\, \mathcal{H})$ erfüllt $f^{\mathfrak{A}'}$ aufgrund der Definition der ς-Transformation die Striktheitsbedingung

$$f^{\mathfrak{A}'}(\vec{\underline{t}}) = \bot \text{ , wenn } \exists i \in [n].\ t_i = \bot \wedge \varsigma(f)(i) = \mathtt{tt}$$

für alle $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ und nach Lemma 5.18 ist die Operation auch ω-stetig. □

Lemma 5.20 ω-Stetigkeit der ς-Transformation
Die ς-Transformation $\Phi_{P,\varsigma}$ ist ω-stetig.

Beweis:
Im folgenden wird $\Phi_{P,\varsigma}$ durch Φ abgekürzt.
Es ist zu zeigen, daß für alle ω-Ketten $T = (\mathfrak{A}_j)_{j\in\mathbb{N}}$ mit $\mathfrak{A}_j \in \mathrm{Int}_{\Sigma,\varsigma}$ die kleinste obere Schranke $\bigsqcup\Phi(T)$ existiert und $\bigsqcup\Phi(T) = \Phi(\bigsqcup T)$ ist. Dafür ist zu zeigen, daß für alle $f^{(n)} \in \mathcal{F}(\dot\cup\,\mathcal{H})$ und alle $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$

$$f^{\sqcup\Phi(T)}(\vec{\underline{t}}) =: e$$

existiert und

$$e = f^{\Phi(\sqcup T)}(\vec{\underline{t}})$$

ist.
Es gilt

$$e = f^{\sqcup\Phi(T)}(\vec{\underline{t}}) = (\bigsqcup f^{\Phi(T)})(\vec{\underline{t}}) =: e_2,$$

da $f^{\mathfrak{A}}$ für alle $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ gemäß der Definition der ς-Interpretationen ω-stetig ist. Weiterhin ist nach der Definition der kanonischen Halbordnung von Funktionen

$$e_2 = \bigsqcup(f^{\Phi(t)}(\vec{\underline{t}})) =: e_3.$$

Da die $f^{\Phi(t)}(\vec{\underline{t}})$ schlicht Elemente des ω-vollständigen Rechenbereichs $\mathrm{T}_{\mathcal{C},\varsigma}$ sind und aufgrund der ω-Ketteneigenschaft von T eine ω-Kette bilden, existiert e_3 und somit auch e.

Fall 1: $\vec{\underline{t}}$ wird mit der linken Seite einer Reduktionsregel $f(\vec{p})\to r$ von einer Variablenbelegung β : $\mathrm{Var}(\vec{p})\to\mathrm{T}_{\mathcal{C},\varsigma}$ semantisch ς-gematcht.

Nach Definition der ς-Transformation gilt

$$e_3 = \bigsqcup[\![r]\!]^{\mathrm{alg}}_{T,\beta} =: e_4.$$

Mit der ω-Stetigkeit der algebraischen Semantik bezüglich der Algebra nach Lemma 5.16, folgt

$$e_4 = [\![r]\!]^{\mathrm{alg}}_{\sqcup T,\beta} =: e_5.$$

Erneute Anwendung der Definition der ς-Transformtion ergibt schließlich

$$e = e_5 = f^{\Phi(\sqcup T)}(\vec{\underline{t}}).$$

Fall 2: $\vec{\underline{t}}$ ist mit keinem Redexschema semantisch ς-matchbar.

Aus der Definition der ς-Transformation folgt direkt

$$e = e_3 = \bigsqcup\bot = \bot = f^{\Phi(\sqcup T)}(\vec{\underline{t}}).$$

□

5.2.5 Eine alternative ς-Transformation

Betrachten wir noch einmal die Definition der ς-Transformation $\Phi_{P,\varsigma}$ auf Seite 77. Ist $\vec{\underline{t}}$ mit der linken Seite einer Regel $f(\vec{p})\to r$ vermittels einer Variablenbelegung β semantisch ς-matchbar, so erhält bei der Transformation von $\mathfrak{A}$ $f^{\Phi_{P,\varsigma}(\mathfrak{A})}(\vec{\underline{t}})$ den Wert $[\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}$, d. h. den semantischen Wert der rechten Regelseite unter $\mathfrak{A}$ und β. Prinzipiell werden so auch Transformationen für rekursive Funktionsschemata definiert (vgl. [Ind93]). Ist f jedoch erzwungen strikt für $\vec{\underline{t}}$, so ist $f^{\Phi_{P,\varsigma}(\mathfrak{A})}(\vec{\underline{t}}) = \bot$. Dies entspricht genau der Bedeutung der Striktheit und ist gemäß der Definition der ς-Interpretationen auch notwendig, da die Operationen für alle erzwungen strikten Argumente auch strikt sein müssen. Ist f allerdings nicht erzwungen strikt für $\vec{\underline{t}}$, aber existiert auch keine passende linke Regelseite, so enthält das Programm keinerlei Information über den Wert $f^{\Phi_{P,\varsigma}(\mathfrak{A})}(\vec{\underline{t}})$. Deshalb wird in der ς-Transformation dieser Wert gleich $\bot$ definiert. Da $\bot$ Undefiniertheit oder fehlende Information repräsentiert, ist dies sinnvoll. Gibt es überhaupt eine Alternative? Da $\bot$ als kleinstes Element das einzige irgendwie ausgezeichnete Element des Rechenbereichs ist, ist die feste Wahl eines anderen Elements des Rechenbereichs unsinnig. Es existiert aber noch eine andere Möglichkeit: Der Wert $f^{\Phi_{P,\varsigma}(\mathfrak{A})}(\vec{\underline{t}})$ kann schlicht als dem „alten“ Wert $f^{\mathfrak{A}}(\vec{\underline{t}})$ gleich definiert werden.
Damit ergibt sich folgende alternative ς-Transformation $\Phi^*_{P,\varsigma}$:

$$f^{\Phi^*_{P,\varsigma}(\mathfrak{A})}(\vec{\underline{t}}) := \begin{cases} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} & , \quad \text{wenn } \vec{\underline{t}} \text{ mit der linken Seite einer Reduktionsregel } f(\vec{p})\to r \text{ von } P \\ & \quad \text{vermittels einer Variablenbelegung } \beta : \mathrm{Var}(\vec{p})\to \mathrm{T}_{\mathcal{C},\varsigma} \\ & \quad \text{semantisch ς-gematcht wird.} \\ f^{\mathfrak{A}}(\vec{\underline{t}}) & , \quad \text{sonst} \end{cases}$$

für alle $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$, $f^{(n)} \in \mathcal{F}(\,\dot\cup\, \mathcal{H})$ und $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$.
Die Definition von $f^{\Phi^*_{P,\varsigma}(\mathfrak{A})}(\vec{\underline{t}}) := f^{\mathfrak{A}}(\vec{\underline{t}})$, wenn f erzwungen strikt für $\vec{\underline{t}}$ ist, ändert nichts gegenüber der ς-Transformation $\Phi_{P,\varsigma}$, da dieser Wert in allen ς-Interpretationen gleich $\bot$ ist.
Auch sehen wir direkt, daß $\Phi^*_{P,\varsigma}(\bot_\varsigma) = \Phi_{P,\varsigma}(\bot_\varsigma)$ und allgemein $\mathfrak{A}_n := (\Phi^*_{P,\varsigma})^n(\bot_\varsigma) = (\Phi_{P,\varsigma})^n(\bot_\varsigma)$ für alle $n \in \mathbb{N}$ ist. Für diejenigen f und $\vec{\underline{t}}$, für die keine passende linke Regelseite besteht, ist $f^{\mathfrak{A}_n}(\vec{\underline{t}}) = \bot$ für alle $n \in \mathbb{N}$. Somit ist auch

$$\bigsqcup_{n\in\mathbb{N}} (\Phi^*_{P,\varsigma})^n(\bot_\varsigma) = \bigsqcup_{n\in\mathbb{N}} (\Phi_{P,\varsigma})^n(\bot_\varsigma),$$

und die ς-Datentypen ließen sich also mit der alternativen ς-Transformation $\Phi^*_{P,\varsigma}$ genauso gut definieren wie mit $\Phi_{P,\varsigma}$.
Warum verwenden wir dann $\Phi^*_{P,\varsigma}$ nicht?
Es gibt einen guten Grund, $\Phi_{P,\varsigma}$ vorzuziehen: Ein Ergebnis von $\Phi^*_{P,\varsigma}$ ist im allgemeinen keine ς-Interpretation mehr. Die Operationen der sich ergebenden Algebra sind im allgemeinen nicht mehr ω-stetig. Freilich läßt sich durch eine Änderung der Definition der ς-Interpretation, derart, daß beliebige, auch unstetige Operationen zugelassen werden, dieses Problem beseitigen. Es ergibt sich jedoch sofort ein neues: $\Phi^*_{P,\varsigma}$ ist auf diesem erweiterten Definitionsbereich nicht ω-stetig. Die Übereinstimmung von Definitions- und Wertebereich und die ω-Stetigkeit sind jedoch zentrale Voraussetzungen für die Anwendung des Fixpunktsatzes von Tarski bei der Definition des ς-Datentyps.

Gehen wir jedoch Schritt für Schritt vor.

Beispiel 5.2 $\boldsymbol{\Phi^*_{P,\varsigma}(\mathfrak{A}) \notin \mathbf{Int}_{\Sigma,\varsigma}}$

$$\mathtt{f(G(A))} \quad \to \quad \mathtt{A}$$

Sei $\varsigma(\mathtt{G}) = (\mathtt{ff})$, $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ mit $\mathtt{f}^{\mathfrak{A}}(\bot) = \bot$ und $\mathtt{f}^{\mathfrak{A}}(\underline{t}) = \mathtt{G}(\mathtt{A})$ für alle $\underline{t} \in \mathrm{T}_{\mathcal{C},\varsigma} \setminus \{\bot\}$.
Sei nun $\mathfrak{A}' := \Phi^*_{P,\varsigma}(\mathfrak{A})$. Dann ist

$$\mathtt{f}^{\mathfrak{A}'}(\mathtt{G}(\bot)) = \mathtt{G}(\mathtt{A}) \not\sqsubseteq \mathtt{A} = \mathtt{f}^{\mathfrak{A}'}(\mathtt{G}(\mathtt{A})).$$

Also ist $f^{\mathfrak{A}'}$ nicht monoton und somit $\mathfrak{A}' \notin \mathrm{Int}_{\Sigma,\varsigma}$. □

Der Spezialfall $\Phi^*_{P,\mathrm{cbv}}(\mathfrak{A}) \in \mathrm{Int}_{\Sigma,\mathrm{cbv}}$ gilt freilich, da in der flachen Halbordnung alle strikten Funktionen ω-stetig sind, und sich auch die nicht-strikten Verzweigungsoperationen cond_G als unproblematisch erweisen. Aber wir wollen eine allgemeine Transformation für beliebige erzwungene Striktheiten haben.
Definieren wir nun eine größere Menge von Algebren, die ς-Interpretationen* $\mathrm{Int}^*_{\Sigma,\varsigma}$, als Definitions- und Wertebereich der Transformation $\Phi^*_{P,\varsigma}$. Eine Algebra ist genau dann eine ς-Interpretation*, wenn sie eine ς-Interpretation ist, abgesehen davon, daß die Operationen nicht ω-stetig sein müssen. Es ist leicht zu sehen, daß

$$\Phi^*_{P,\varsigma} : \mathrm{Int}^*_{\Sigma,\varsigma} \to \mathrm{Int}^*_{\Sigma,\varsigma}$$

ist. $\Phi^*_{P,\varsigma}$ ist jedoch nicht ω-stetig.

Beispiel 5.3 **Unstetigkeit von** $\boldsymbol{\Phi^*_{P,\varsigma}}$

$$\mathtt{a} \quad \to \quad \mathtt{f(a)}$$

$\mathfrak{A}, \mathfrak{A}' \in \mathrm{Int}^*_{\Sigma,\varsigma}$ seien definiert durch

$$\begin{array}{cccccc} \mathtt{a}^{\mathfrak{A}}() := \bot & & & & \mathtt{a}^{\mathfrak{A}'}() := \mathtt{A} & \\ \mathtt{f}^{\mathfrak{A}}(\bot) & := & \mathtt{f}^{\mathfrak{A}'}(\bot) & := & \mathtt{A} \\ \mathtt{f}^{\mathfrak{A}}(\mathtt{A}) & := & \mathtt{f}^{\mathfrak{A}'}(\mathtt{A}) & := & \bot \end{array}$$

Dann ist $\mathfrak{A} \sqsubseteq \mathfrak{A}'$, jedoch

$$\mathtt{a}^{\Phi^*_{P,\varsigma}(\mathfrak{A})}() = \mathtt{A} \not\sqsubseteq \bot = \mathtt{a}^{\Phi^*_{P,\varsigma}(\mathfrak{A}')}()$$

und somit

$$\Phi^*_{P,\varsigma}(\mathfrak{A}) \not\sqsubseteq \Phi^*_{P,\varsigma}(\mathfrak{A}').$$

$\Phi^*_{P,\varsigma}$ ist also nicht monoton. □

Allerdings haben wir festgestellt, daß

$$\bigsqcup_{n\in\mathbb{N}} (\Phi^*_{P,\varsigma})^n(\bot_\varsigma) = \bigsqcup_{n\in\mathbb{N}} (\Phi_{P,\varsigma})^n(\bot_\varsigma) = \mathcal{D}_{P,\varsigma}.$$

Nur können wir jetzt nicht mit Hilfe des Fixpunktsatzes von Tarski schließen, daß $\mathcal{D}_{P,\varsigma}$ ein Fixpunkt von $\Phi^*_{P,\varsigma}$ und gar der kleinste ist. Ersteres folgt jedoch direkt daraus, daß $\mathcal{D}_{P,\varsigma}$ ein Fixpunkt von $\Phi_{P,\varsigma}$ ist. Letzteres läßt sich analog zu den Beweisen der Lemmata 6.6, S. 140, und 6.7, S. 141, zeigen.

Wir haben damit aber $\Phi_{P,\varsigma}$ verwendet, um zu zeigen, daß $\mathcal{D}_{P,\varsigma} = \bigsqcup_{n\in\mathbb{N}}(\Phi^*_{P,\varsigma})^n(\bot_\varsigma)$ der kleinste Fixpunkt von $\Phi^*_{P,\varsigma}$ ist. Außerdem haben wir in Kapitel 3 die Ordnung $\langle \mathrm{T}_{\mathcal{C},\varsigma}, \trianglelefteq\rangle$ als Ordnung bezüglich des Informationsgehalts gedeutet und auf diese Weise eine intuitive Begründung für die Beschränkung auf ω-stetige Operationen gegeben. In Abschnitt 6.2 werden wir zwar eine Eigenschaft betrachten, die $\Phi^*_{P,\varsigma}$ im Gegensatz zu $\Phi_{P,\varsigma}$ besitzt, die jedoch von keiner nützlichen Bedeutung ist.
Da sich $\Phi_{P,\varsigma}$ insgesamt als einfacher und intuitiv einleuchtender als $\Phi^*_{P,\varsigma}$ herausstellt, haben wir es als ς-Transformation gewählt.

5.3 Die ς-Reduktionssemantiken

5.3.1 Definition der ς-Reduktionssemantiken

Nach der Definition der ς-Fixpunktsemantik suchen wir nun nach einer passenden ς-Reduktionssemantik. Im Falle der cbv- und der cbn-Semantik war dies einfach, da wir auf allgemein bekannte Reduktionsstrategien zurückgreifen konnten. Auch ist die Verwendung einer innermost Reduktionsstrategie für den cbv-Auswertungsmechanismus und einer outermost Reduktionsstrategie für den cbn-Auswertungsmechanismus naheliegend. Beliebige erzwungene Striktheiten ς führen nun zu einer Vermischung der beiden Auswertungsmechanismen. Die Suche nach einer „middlemost“ Reduktionsstrategie dürfte aber kaum aussichtsreich sein. Außerdem sind die unterschiedlichen Striktheiten der Konstruktoroperationen verwirrend, da zu den Konstruktorsymbolen doch überhaupt gar keine Reduktionsregeln existieren.

Wir wollen daher erst einmal Abstand von dem Problem nehmen, und nicht direkt eine Reduktionsstategie definieren. Stattdessen untersuchen wir, welche Art von Reduktionen in der ς-Fixpunktsemantik korrekt ist. Eine Reduktion $t \xrightarrow[l\to r]{u} t'$ heißt genau dann **korrekt in der ς-Fixpunktsemantik**, wenn $[\![t]\!]^{\mathrm{fix}}_{P,\varsigma} = [\![t']\!]^{\mathrm{fix}}_{P,\varsigma}$ gilt.
Beispielsweise ist in der cbv-Fixpunktsemantik die Reduktion an einer innermost Redexstelle immer korrekt.

Beispiel 5.4 Reduktion an einer innermost Redexstelle und die cbv-Semantik

$$\texttt{inf} \quad \to \quad \texttt{Succ(inf)}$$

Betrachten wir die Reduktion

$$\mathrm{cond}_{\texttt{Succ}}\texttt{(Succ(inf),Zero,}\underline{\texttt{inf}}\texttt{)} \quad \xrightarrow{3} \quad \mathrm{cond}_{\texttt{Succ}}\texttt{(Succ(inf),Zero,Succ(inf))}$$

Wegen $[\![\texttt{inf}]\!]^{\mathrm{fix}}_{P,\mathrm{cbv}} = \bot$ gilt auch

$$[\![\mathrm{cond}_{\texttt{Succ}}(\texttt{Succ(inf)},\texttt{Zero},\texttt{inf})]\!]^{\mathrm{fix}}_{P,\mathrm{cbv}} = \bot = [\![\mathrm{cond}_{\texttt{Succ}}(\texttt{Succ(inf)},\texttt{Zero},\texttt{Succ(inf)})]\!]^{\mathrm{fix}}_{P,\mathrm{cbv}}.$$

3 und und die li-Redexstelle 1.1 sind die einzigen innermost Redexstellen des Ausgangsterms. □

Andererseits sind in der cbv-Semantik offensichtlich nicht alle Reduktionen korrekt.

Beispiel 5.5 Reduktion an einer outermost Redexstelle und die cbv-Semantik

$$\texttt{inf} \quad \rightarrow \quad \texttt{Succ(inf)}$$

Betrachten wir jetzt

$$\underline{\text{cond}_{\texttt{Succ}}}\texttt{(Succ(inf),Zero,inf)} \quad \xrightarrow{\varepsilon} \quad \texttt{Zero}$$

Es ist

$$[\![\text{cond}_{\texttt{Succ}}(\texttt{Succ}(\texttt{inf}), \texttt{Zero}, \texttt{inf})]\!]^{\text{fix}}_{P,\text{cbv}} = \bot \neq \texttt{Zero} = [\![\texttt{Zero}]\!]^{\text{fix}}_{P,\text{cbv}}.$$

Die Reduktion an der outermost Redexstelle ε ist nicht korrekt in der cbv-Fixpunktsemantik, da $[\![\texttt{Succ}(\texttt{inf})]\!]^{\text{fix}}_{P,\text{cbv}} = \bot$, und die Verzweigungsoperation $\text{cond}^{\mathcal{D}^{\text{fix}}_{P,\text{cbv}}}_{\texttt{Succ}}$ an der ersten Stelle strikt ist. □

Das allgemeine Problem bei der Reduktion mit einer Reduktionsregel $f(\vec{p})\rightarrow r$ besteht darin, daß die Argumente von f in einem Redex $f(\vec{t})$ meistens noch Funktions- oder Hilfssymbole enthalten. Die Reduktion ist jedoch nur dann mit Sicherheit korrekt, wenn f nicht erzwungen strikt für die unbekannten semantischen Werte der Argumente ist, und die semantischen Werte der Argumente auch zum Pattern der Reduktionsregel passen. (Die Reduktion kann natürlich auch andernfalls korrekt sein; dies hängt auch von der rechten Regelseite ab; diese wollen wir jedoch nicht betrachten.)

Die semantische Approximation ermöglicht es sicherzustellen, daß die genannten Bedingungen erfüllt sind.

Definition 5.8 Semantische Approximation
Die algebraische Grundtermsemantik bezüglich der kleinsten ς-Interpretation $\bot_\varsigma$, $[\![\cdot]\!]^{\text{alg}}_{\bot_\varsigma}$, heißt **semantische Approximation** bezüglich der erzwungenen Striktheit ς.

$$[\![G(\vec{t})]\!]^{\text{alg}}_{\bot_\varsigma} = \begin{cases} G([\![t_1]\!]^{\text{alg}}_{\bot_\varsigma}, \ldots, [\![t_n]\!]^{\text{alg}}_{\bot_\varsigma}) & , \quad \text{falls } G \text{ nicht erzwungen strikt} \\ & \quad\ \text{für } ([\![t_1]\!]^{\text{alg}}_{\bot_\varsigma}, \ldots, [\![t_n]\!]^{\text{alg}}_{\bot_\varsigma}) \\ \bot & , \quad \text{andernfalls} \end{cases}$$

$$[\![f(\vec{t})]\!]^{\text{alg}}_{\bot_\varsigma} = \bot$$

für alle $G^{(n)} \in \mathcal{C}$, $f^{(n)} \in \mathcal{F}(\dot\cup\, \mathcal{H})$. □

Aus der Monotonie der algebraischen Termsemantik bezüglich der Algebra gemäß Lemma 5.16, S. 91, folgt, daß für alle $\mathfrak{A} \in \text{Int}_{\Sigma,\varsigma}$ und $t \in \text{T}_\Sigma$ $[\![t]\!]^{\text{alg}}_{\bot_\varsigma} \trianglelefteq [\![t]\!]^{\text{alg}}_{\mathfrak{A}}$ ist. Für ein beliebiges $\underline{t} \in \text{T}^\infty_{\mathcal{C},\bot}$ garantiert $\underline{t} \trianglelefteq [\![t]\!]^{\text{alg}}_{\bot_\varsigma}$ also auch $\underline{t} \trianglelefteq [\![t]\!]^{\text{alg}}_{\mathfrak{A}}$.

Wir definieren den Begriff des syntaktischen ς-Matchens in Analogie zum semantischen ς-Matchen (Def. 5.5, S. 76). Damit definieren wir die Menge der ς-Redexe, deren Reduktion in der ς-Fixpunktsemantik korrekt ist.

Definition 5.9 Syntaktisches ς-Matchen
Ein Termtupel $\vec{t} \in (\text{T}_\Sigma)^n$ heißt genau dann mit einem Redexschema $f^{(n)}(\vec{p}) \in \text{RedS}_P$ unter einer Substitution $\sigma : \text{Var}(\vec{p})\rightarrow\text{T}_\Sigma$ **syntaktisch ς-matchbar**, wenn

1. f nicht erzwungen strikt für $([\![t_1]\!]^{\text{alg}}_{\bot_\varsigma}, \ldots, [\![t_n]\!]^{\text{alg}}_{\bot_\varsigma})$ ist, und

2. die Ordnungsbedingung
$$p_i[\bot/\mathrm{Var}(p_i)] \trianglelefteq [\![t_i]\!]^{\mathrm{alg}}_{\bot_\varsigma}$$
für alle $i \in [n]$ erfüllt ist, und

3.
$$f(\vec{t}) = f(\vec{p})\sigma$$
ist.

□

Definition 5.10 ς-Redex
$f(\vec{t})$ mit $f^{(n)} \in \mathcal{F}(\dot\cup\ \mathcal{H})$ und $\vec{t} \in (\mathrm{T}_\Sigma)^n$ heißt genau dann **ς-Redex** zu einem Redexschema $f(\vec{p}) \in \mathrm{RedS}_P$, wenn $\vec{t}$ vermittels einer Substitution $\sigma : \mathrm{Var}(\vec{p}) \to \mathrm{T}_\Sigma$ mit $f(\vec{p})$ syntaktisch ς-matchbar ist.
Wir bezeichnen die Menge aller ς-Redexe des Programms P mit $\mathrm{Red}_{P,\varsigma}$. □

Auch [Cou90] definiert für seine cbv- und cbn-Semantik zwei verschiedene Mengen von Redexen.

Durch die Menge der ς-Redexe ist eine Instanz eines Programms, präziser: des damit assoziierten Termersetzungssystems, definiert. Wir verwenden ς auch zur Bezeichnung dieser Instanz. Durch die Definition der Menge der ς-Redexe erhalten wir somit auch die Begriffe der Menge aller ς-Reduktionsstellen, $\mathrm{RedOcc}_{P,\varsigma}$, der ς-Reduktion und der ς-Reduktionsrelationen $\xrightarrow[P,\varsigma]{}$ und $\xrightarrow[P,\varsigma]{}\!\!\!\!\!\rightarrow$.
Beweisen werden wir die Korrektheit der ς-Reduktion in der ς-Fixpunktsemantik erst in 5.4.3 mit Hilfe mehrerer noch anzugebender Lemmata.

Mit der ς-Reduktionsrelation definieren wir nun eine ς-Reduktionssemantik. Wir definieren sie ähnlich wie die Normalformsemantik (Def. 3.7, S. 45), jedoch unter Verwendung der semantischen Approximation, da der Rechenbereich $\mathrm{T}_{\mathcal{C},\varsigma}$ im allgemeinen auch unendliche und partielle Terme enthält, die wir durch Berechnungen nur approximieren können.

Definition 5.11 Allgemeine ς-Reduktionssemantik
Die **allgemeine ς-Reduktionssemantik**
$$[\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} : \mathrm{T}_\Sigma \to \mathrm{T}_{\mathcal{C},\varsigma}$$
ist definiert durch
$$[\![t]\!]^{\mathrm{red}}_{P,\varsigma} := \bigsqcup\{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t'\}.$$

□

Die Existenz der kleinsten oberen Schranke in der Definition ist nicht offensichtlich. Wir beweisen die Wohldefiniertheit der allgemeinen ς-Reduktionssemantik und der gleich folgenden parallel-outermost ς-Reduktionssemantik in 5.3.3.
Die Definition der allgemeinen ς-Reduktionssemantik ist sehr schlicht. Es ist intuitiv einleuchtend, daß die Berechnung aller von einem Term ausgehender korrekter Reduktionen (von denen die ς-Reduktionen freilich auch nur eine Teilmenge sind) zur Semantik des Terms führt. Beim Beweis der Übereinstimmung der ς-Fixpunkt- und der ς-Reduktionssemantiken in 5.5 erweist sich die allgemeine ς-Reduktionssemantik als sehr nützlich. Für eine Implementierung stellt sie jedoch keine

vernünftige Basis dar, da wir dafür — wie schon in Bezug auf die Normalformsemantik gesagt — eine (deterministische) Reduktionsstrategie brauchen.

Orientierend an der po-Reduktionsstrategie der cbn-Semantik definieren wir eine parallel-outermost ς-Reduktionsstrategie. Hierbei werden in einer parallelen Reduktion alle outermost Redexstellen der ς-Redexe eines Terms reduziert. Aus 5. – 7., Kapitel V in [O'Do77] wissen wir, daß die parallel-outermost Reduktionsstrategie für die Terme, die eine Normalform besitzen, immer terminiert, und zwar generell in Instanzen von beinahe orthogonalen Termersetzungssystemen, deren Redexmengen residual abgeschlossen und outer sind; Eigenschaften, die wir in 5.3.3 respektive 5.5.3 für die Menge der ς-Redexe nachweisen werden. Somit erscheint es wahrscheinlich, daß auch unsere semantischen Rechenterme richtig approximiert werden.

Definition 5.12 **po-*I*-Reduktion**
Sei I eine Instanz eines beinahe orthogonalen Termersetzungssystems R über einer Signatur Σ mit der Menge der I-Redexe $\mathrm{Red}_{R,I}$.
Die **Menge der outermost *I*-Redexstellen**, $\mathrm{Outer}_{R,I}(t)$, und die **Menge der non-outermost *I*-Redexstellen** , $\mathrm{NOuter}_{R,I}(t)$, eines Terms $t \in \mathrm{T}_\Sigma$ sind definiert durch:

$$\begin{aligned} \mathrm{Outer}_{R,I}(t) &:= \{u \in \mathrm{RedOcc}_{R,I}(t) \mid \nexists v \in \mathrm{RedOcc}_{R,I}(t).\ v < u\} \\ \mathrm{NOuter}_{R,I}(t) &:= \mathrm{RedOcc}_{R,I}(t) \setminus \mathrm{Outer}_{R,I}(t). \end{aligned}$$

Eine parallele I-Reduktion $t \xrightarrow[R,I]{U}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t'$ heißt genau dann

- **non-outermost *I*-Reduktion**, geschrieben $t \xrightarrow[R,\mathrm{no},I]{}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t'$, wenn $U \subseteq \mathrm{NOuter}_{R,I}(t)$;
- **outermost *I*-Reduktion**, geschrieben $t \xrightarrow[R,\mathrm{o},I]{}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t'$, wenn $U \subseteq \mathrm{Outer}_{R,I}(t)$;
- **parallel-outermost (po-) *I*-Reduktion**, geschrieben $t \xrightarrow[R,\mathrm{po},I]{}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t'$, wenn $U = \mathrm{Outer}_{R,I}(t)$.

□

In dieser Arbeit benötigen wir die obige, allgemeine Definition freilich nur für den Fall $I = \varsigma$. Trotzdem zeigen wir im weiteren viele Aussagen in diesem allgemeineren Rahmen, um die Unabhängigkeit dieser Aussagen von den speziellen Programmen bzw. deren assoziierten Termersetzungssystemen zu demonstrieren.
Obwohl die definierten Begriffe sich auf die durch die erzwungene Striktheit ς eindeutig definierte Instanz des zum Programm P assoziierten Termersetzungssystems $\hat{P}$ beziehen, verwenden wir in allen Bezeichneren P anstelle von $\hat{P}$: $\mathrm{Outer}_{P,\varsigma}(t)$, $\mathrm{NOuter}_{P,\varsigma}(t)$, $\xrightarrow[P,\mathrm{no},\varsigma]{}\!\!\!\!\rightarrow\!\!\!\!\rightarrow$, $\xrightarrow[P,\mathrm{o},\varsigma]{}\!\!\!\!\rightarrow\!\!\!\!\rightarrow$, $\xrightarrow[P,\mathrm{po},\varsigma]{}\!\!\!\!\rightarrow\!\!\!\!\rightarrow$.

Definition 5.13 **Parallel-outermost ς-Reduktionssemantik**
Die **parallel-outermost (po-) ς-Reduktionssemantik**

$$[\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma} : \mathrm{T}_\Sigma \to \mathrm{T}_{\mathcal{C},\varsigma}$$

ist definiert durch

$$[\![t]\!]^{\mathrm{po}}_{P,\varsigma} := \bigsqcup \{ [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \xrightarrow[P,\mathrm{po},\varsigma]{*}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t' \}.$$

□

Wie schon die po-Reduktionssemantik der cbn-Semantik werden sowohl die allgemeine als auch die po-ς-Reduktionssemantik durch die kleinste obere Schranke einer im allgemeinen unendlichen Menge von Approximationen definiert. Wie dort stellt sich daher die Frage, ob es möglich ist, daß ein semantischer Wert eines Terms, der ein endlicher, totaler Konstruktorterm ist, nur beliebig genau approximiert, aber nie erreicht werden kann. Dies ist nicht der Fall.

Lemma 5.21 Berechenbarkeit der ς-Reduktionssemantiken
Sei $t \in \mathrm{T}_\Sigma$. Wenn

$$[\![t]\!]^{\mathrm{red}}_{P,\varsigma} = \underline{t} \in \mathrm{T}_{\mathcal{C}} \text{ bzw. } [\![t]\!]^{\mathrm{po}}_{P,\varsigma} = \underline{t} \in \mathrm{T}_{\mathcal{C}},$$

dann existiert die ς-Normalform $t{\downarrow_{P,\varsigma}}$ bzw. die po-ς-Normalform $t{\downarrow_{P,\mathrm{po},\varsigma}}$, und es gilt

$$[\![t]\!]^{\mathrm{red}}_{P,\varsigma} = \underline{t} = t{\downarrow_{P,\varsigma}} \text{ bzw. } [\![t]\!]^{\mathrm{po}}_{P,\varsigma} = \underline{t} = t{\downarrow_{P,\mathrm{po},\varsigma}}.$$

Beweis:
Sei $\underline{t} = [\![t]\!]^{\mathrm{red}}_{P,\varsigma} = \bigsqcup\{[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t'\} \in \mathrm{T}_{\mathcal{C}}$. $\langle \mathrm{T}_{\mathcal{C},\varsigma}, \trianglelefteq \rangle$ ist ω-induktiv und $\underline{t}$ ω-kompakt. Somit existiert $[\![\hat{t}]\!]^{\mathrm{alg}}_{\perp_\varsigma} \in \{[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t'\}$ mit $\underline{t} = [\![\hat{t}]\!]^{\mathrm{alg}}_{\perp_\varsigma}$. Wegen $\underline{t} \in \mathrm{T}_{\mathcal{C}}$ folgt $\hat{t} = \underline{t}$. Also gilt $t \xrightarrow[P,\varsigma]{*} \hat{t} = \underline{t}$, und $\underline{t}$ ist die ς-Normalform von t.
Die Aussage über die po-ς-Reduktionssemantik folgt analog. □

Zusammen mit der Übereinstimmung der allgemeinen und der po-ς-Reduktionssemantik, die wir in 5.5 zeigen, läßt sich leicht folgern, daß die ς-Normalform und die po-ς-Normalform eines Terms übereinstimmen.

Aufgrund der ω-Induktivität der kanonischen Halbordnung des Rechenbereichs sind alle semantischen Werte schon durch ihre endlichen Approximationen eindeutig bestimmt. Daher können zwei syntaktische Terme $t, t' \in \mathrm{T}_\Sigma$ unterschiedlicher denotationeller Semantik ($[\![t]\!]^{\mathrm{fix}}_{P,\varsigma} \neq [\![t']\!]^{\mathrm{fix}}_{P,\varsigma}$) immer operationell unterschieden werden, d. h. bei Hinzunahme von zusätzlichen Funktionssymbolen und dazugehörigen Programmregeln (die die kompositionelle Semantik nur erweitern) existiert $f^{(n)} \in \mathcal{F}$ mit[2] $f(t) \xrightarrow[P,\varsigma]{*} \underline{t} \in \mathrm{T}_{\mathcal{C}}$ und <u>nicht</u> $f(t') \xrightarrow[P,\varsigma]{*} \underline{t}$.
Für unseren Programmiersprachen ähnliche Programmiersprachen wird eine derartige Aussage in [Ra&Vui80] bewiesen. In funktionalen Programmiersprachen höherer Ordnung ist diese erwünschte **fully abstract Eigenschaft** von Semantiken jedoch aufgrund der Sequentialität aller Operationen (siehe Kapitel 7) ein Problem ([Ber&Cur82], [Ca&Fell92]).

5.3.2 ς-Reduktionssemantiken und die li- und die po-Reduktionssemantik

Wir haben die ς-Semantiken als Verallgemeinerung der cbv- und der cbn-Semantik eingeführt. Bei der Definition der ς-Reduktionssemantiken haben wir jedoch kaum auf die Definitionen der li- und der po-Reduktionssemantik zurückgegriffen. Wir wollen hier die bestehenden Beziehungen aufzeigen.

Betrachten wir die Definition der ς-Redexe, so sehen wir, daß die cbv-Redexe genau die innermost Redexe sind, und alle Redexe cbn-Redexe sind ($\mathrm{Red}_{P,\mathrm{cbn}} = \mathrm{Red}_P$).

[2] Da die allgemeine und die po-ς-Reduktionssemantik übereinstimmen, wie wir in 5.5 zeigen, gilt diese Aussage selbstverständlich auch für po-ς-Reduktion.

Hieraus folgt direkt, daß die po-Reduktionssemantik gleich der po-cbn-Reduktionssemantik ist. In diesem Fall ist die in Kapitel 4 angegebene Reduktionssemantik also wirklich ein Spezialfall einer der zwei ς-Reduktionssemantiken.
Für die cbv-Semantik ist dies jedoch nicht gegeben. Die hierfür in Kapitel 4 definierte li-Reduktionssemantik basiert zwar auf der Reduktion an innermost Redexstellen, d. h. cbv-Redexstellen, aber die sequentielle li-Reduktionsstrategie reduziert nur immer genau an einer cbv-Redexstelle, der leftmost-innermost Redexstelle. Dagegen reduziert die parallele po-cbv-Reduktionsstrategie im allgemeinen an mehreren innermost Redexstellen, wie das folgende Beispiel demonstriert.

Beispiel 5.6 li- und po-cbv-Reduktionsstrategie

$$\texttt{inf} \;\to\; \texttt{Succ(inf)}$$

Betrachten wir die Reduktion eines Terms

$$\begin{aligned}
\text{cond}_{\texttt{Succ}}(\texttt{Succ}(\underline{\texttt{inf}}),\texttt{Zero},\underline{\texttt{inf}}) &\xrightarrow[P,\text{po,cbv}]{}\!\!\!\!\!\!\rightarrow\; \text{cond}_{\texttt{Succ}}(\texttt{Succ}(\texttt{Succ}(\underline{\texttt{inf}})),\texttt{Zero},\texttt{Succ}(\underline{\texttt{inf}})) \\
&\xrightarrow[P,\text{po,cbv}]{}\!\!\!\!\!\!\rightarrow\; \text{cond}_{\texttt{Succ}}(\texttt{Succ}^3(\underline{\texttt{inf}}),\texttt{Zero},\texttt{Succ}^2(\underline{\texttt{inf}})) \\
&\xrightarrow[P,\text{po,cbv}]{}\!\!\!\!\!\!\rightarrow\; \ldots
\end{aligned}$$

$$\begin{aligned}
\text{cond}_{\texttt{Succ}}(\texttt{Succ}(\underline{\texttt{inf}}),\texttt{Zero},\texttt{inf}) &\xrightarrow[P,\text{li}]{}\; \text{cond}_{\texttt{Succ}}(\texttt{Succ}(\texttt{Succ}(\underline{\texttt{inf}})),\texttt{Zero},\texttt{inf}) \\
&\xrightarrow[P,\text{li}]{}\; \text{cond}_{\texttt{Succ}}(\texttt{Succ}^3(\underline{\texttt{inf}}),\texttt{Zero},\texttt{inf}) \\
&\xrightarrow[P,\text{li}]{}\; \ldots
\end{aligned}$$

□

Die po-cbv-Reduktionsstrategie reduziert in einem Schritt parallel alle outermost Redexstellen der innermost Redexstellen. Innermost Redexstellen können schließlich aufgrund der Verzweigungssymbole sehr wohl übereinander liegen. Wie wir schon bei ihrer Definition anmerkten, ist die Bezeichnung „innermost“ Redexstellen irreführend.
Die li-Reduktion ist sozusagen eine lo-cbv-Reduktion. Wir haben in 4.3 gesehen, daß mit der lo-Reduktionsstrategie, die wir entsprechend auch lo-cbn-Reduktionsstrategie nennen, keine cbn-Semantik definierbar ist. Somit realisiert eine lo-ς-Reduktionsstrategie für beliebige ς keine ς-Semantik. Daher definieren wir auch keine lo-ς-Reduktionssemantik. Im Falle der cbv-Semantik führt die lo-cbv-Reduktionsstrategie jedoch aufgrund der speziellen Semantik der Verzweigungsoperationen und der Striktheit aller übrigen Operationen zum Ziel. Da die cbv-Semantik ein solcher Ausnahmefall ist, und die li-Reduktionssemantik also kein Spezialfall einer unserer zwei ς-Reduktionssemantiken ist, haben wir in Kapitel 4 die Invarianz der li-Reduktionssemantik und ihre Übereinstimmung mit der cbv-Fixpunktsemantik schon bewiesen.

Es sei noch erwähnt, daß die andersartige Formulierung der li-Reduktionssemantik, d. h. der Verzicht auf die Verwendung semantischer Approximationen, keine Rolle bei dieser Abweichung spielt. Es ist leicht einzusehen, daß für alle $t \in \mathrm{T}_\Sigma$

$$\begin{aligned}
[\![t]\!]^{\text{li}}_P &= \begin{cases} t{\downarrow_{P,\text{li}}} & \text{, falls } t{\downarrow_{P,\text{li}}} \text{ existiert und } t{\downarrow_{P,\text{li}}} \in \mathrm{T}_{\mathcal{C}} \text{ ist} \\ \bot & \text{, andernfalls} \end{cases} \\
&= \bigsqcup\{[\![t']\!]^{\text{alg}}_{\bot_{\text{cbv}}} \mid t \xrightarrow[P,\text{li}]{*} t'\}
\end{aligned}$$

gilt. Die einfachere Definitionsform vermittels der Konstruktornormalform ist für jede ς-Reduktionssemantik möglich, deren kanonische Ordnung des Rechenbereichs flach, also gleich $\langle \mathrm{T}_{\mathcal{C}}^{\perp}, \trianglelefteq \rangle$ ist.

Aus dem neugewonnenen Blickwinkel der ς-Semantiken sehen wir somit, daß der Unterschied zwischen Reduktionssemantiken der cbv- und der cbn-Semantik nur scheinbar auf der Position im Term (inner- oder outermost) der reduzierten Redexe beruht. Stattdessen liegt der wesentliche Unterschied in den verschiedenen Arten von Redexen (cbv- oder cbn-Redexe), die reduziert werden.

Die ς-Semantiken sind, auch wenn sie abgesehen von der cbv- und der cbn-Semantik und vielleicht noch den zwei anfangs erwähnten Mischformen nie in realen Programmiersprachen eingesetzt werden, in einem weiteren Punkt nützlich: Die cbv-Auswertung erweist sich in der Praxis als effizienter implementierbar als die cbn-Auswertung. Insbesondere aufgrund der Möglichkeiten der unendlichen Datenstrukturen besitzen jedoch viele moderne funktionale Programmiersprachen wie Miranda und Haskell cbn-Semantik. Das Effizienzproblem wird dann mit einer Striktheitsanalyse angegangen, die beispielsweise mit der schon erwähnten abstrakten Interpretation durchgeführt werden kann. Diese bestimmt — soweit möglich — welche Operationen an welchen Argumentstellen strikt sind. An diesen Stellen kann dann der effizientere cbv-Auswertungsmechanismus eingesetzt werden. Insgesamt entsteht so eine Mischung von cbv- und cbn-Auswertung.
Die ς-Semantiken bieten hierfür eine theoretische Grundlage. Man führt die Striktheitsanalyse im Rahmen einer cbn-Semantik durch. Aus der Definition der ς-Fixpunktsemantik ist leicht ersichtlich, daß die erzwungene Striktheit einer Argumentstelle eines Operationssymbols von nicht-strikt zu strikt geändert werden kann, wenn die Operation an dieser Argumentstelle strikt ist, ohne dadurch die Semantik (den Datentyp) des Programms zu verändern. Aus der Striktheitsanalyse gewinnt man somit eine „möglichst strikte“ erzwungene Striktheit ς mit $\mathcal{D}_{P,\varsigma}^{\mathrm{fix}} = \mathcal{D}_{P,\mathrm{cbn}}^{\mathrm{fix}}$. Berechnungen können nun mit einer ς-Reduktionssemantik erfolgen.
Die Korrektheit eines derart optimierenden Interpreters oder Compilers ließe sich also durch den Beweis der Übereinstimmung mit einer ς-Reduktionssemantik nachweisen.

5.3.3 Die Wohldefiniertheit

Wir beweisen nun noch die Wohldefiniertheit der allgemeinen ς-Reduktionssemantik und der po-ς-Reduktionssemantik. Die hierfür bewiesenen zentralen Lemmata 5.23 und 5.24 sind auch für den Beweis der Übereinstimmung der ς-Fixpunkt- und der ς-Reduktionssemantiken in 5.5 von großer Bedeutung.

Zuerst zeigen wir eine Hilfsaussage für das darauf folgende Lemma.

Lemma 5.22 **Vertauschung von semantischer Approximation und Teiltermersetzung**
Seien $t, t' \in \mathrm{T}_{\Sigma}$ Grundterme und $u \in \mathrm{Occ}(t)$. Dann ist

$$[\![t[u \leftarrow t']]\!]_{\perp_\varsigma}^{\mathrm{alg}} = [\![t]\!]_{\perp_\varsigma}^{\mathrm{alg}}[u \leftarrow [\![t']\!]_{\perp_\varsigma}^{\mathrm{alg}}].$$

Beweis:

$u = \varepsilon$: $[\![t[\varepsilon \leftarrow t']]\!]_{\perp_\varsigma}^{\mathrm{alg}} = [\![t']\!]_{\perp_\varsigma}^{\mathrm{alg}} = [\![t]\!]_{\perp_\varsigma}^{\mathrm{alg}}[\varepsilon \leftarrow [\![t']\!]_{\perp_\varsigma}^{\mathrm{alg}}]$.

$u = i.u'$: $(i \in \mathbb{N}_+)$

$\underline{t = G(t_1, \ldots, t_n)\text{:}}$ $(G^{(n)} \in \mathcal{C})$

$$\begin{aligned}
[\![t[u \leftarrow t']]\!]^{\mathrm{alg}}_{\perp_\varsigma} &= G^{\perp_\varsigma}([\![t_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t_i[u' \leftarrow t']]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\perp_\varsigma}) \\
&\overset{\text{I.V.}}{=} G^{\perp_\varsigma}([\![t_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t_i]\!]^{\mathrm{alg}}_{\perp_\varsigma}[u' \leftarrow [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}], \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\perp_\varsigma}) \\
(\text{entweder} = \perp \text{ oder} = G(\ldots)) \quad &= G^{\perp_\varsigma}([\![t_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\perp_\varsigma})[u \leftarrow [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}] \\
&= [\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma}[u \leftarrow [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}].
\end{aligned}$$

$\underline{t = f(t_1, \ldots, t_n)\text{:}}$ $(f^{(n)} \in \mathcal{F})$

$$[\![t[u \leftarrow t']]\!]^{\mathrm{alg}}_{\perp_\varsigma} = \perp = \perp[i.u' \leftarrow [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}] = [\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma}[u \leftarrow [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}].$$

□

Im Beispiel 4.9, S. 69, haben wir für die cbn-Semantik gesehen, wie ein semantischer Wert eines Terms durch fortwährende Reduktion approximiert wird. Im folgenden Lemma beweisen wir, daß Reduktion nur zu einem Informationsgewinn (bzgl. der Halbordnung $\langle \mathrm{T}_{\mathcal{C},\varsigma}, \trianglelefteq \rangle$) und niemals zu einem Informationsverlust führen kann.

Lemma 5.23 Informationsgewinn durch Reduktion

$$\begin{aligned}
t \xrightarrow[P]{} t' &\implies [\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma} \trianglelefteq [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \\
t \xrightarrow[P,\mathrm{no}]{} t' &\implies [\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma} = [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}
\end{aligned}$$

Beweis:
Gegeben sei die Reduktion $t \xrightarrow[f(\vec{p}) \to r]{u} t'$. Somit ist

$$\begin{aligned}
t/u &= f(\vec{p})\sigma \\
t'/u &= r\sigma
\end{aligned}$$

für eine Substitution σ, und es gilt

$$[\![f(\vec{p})\sigma]\!]^{\mathrm{alg}}_{\perp_\varsigma} = \perp \trianglelefteq [\![r\sigma]\!]^{\mathrm{alg}}_{\perp_\varsigma}.$$

Mit Lemma 5.22 über die Vertauschung von semantischer Approximation und Teiltermersetzung und der Invarianz der kanonischen Halbordnung der partiellen Terme folgt:

$$\begin{aligned}
[\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma} &= [\![t[u \leftarrow f(\vec{p})\sigma]]\!]^{\mathrm{alg}}_{\perp_\varsigma} \\
&= [\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma}[u \leftarrow f(\vec{p})\sigma] \\
&\trianglelefteq [\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma}[u \leftarrow r\sigma] \\
&= [\![t[u \leftarrow r\sigma]]\!]^{\mathrm{alg}}_{\perp_\varsigma} \\
&= [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}.
\end{aligned}$$

Ist $u \in \mathrm{NOuter}_P(t)$, so existiert eine Stelle $v < u$ mit $t(v) \in \mathcal{F}(\,\dot\cup\,\mathcal{H})$, d. h. $[\![t/v]\!]^{\mathrm{alg}}_{\perp_\varsigma} = \perp$. Somit ist $u \in \mathrm{Occ}([\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma})$, und es gilt:

$$[\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma} = [\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma}[u \leftarrow f(\vec{p})\sigma] = [\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma} = [\![t]\!]^{\mathrm{alg}}_{\perp_\varsigma}[u \leftarrow r\sigma] = [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}.$$

□

Diese Aussage gilt natürlich insbesondere auch für die einfachen und parallelen ς-Reduktionen ($t \xrightarrow[P,\mathrm{no},\varsigma]{} t'$ impliziert $t \xrightarrow[P,\mathrm{no}]{} t'$).

Lemma 5.24 Residuale Abgeschlossenheit der ς-Redexe
Die Menge der ς-Redexe $\mathrm{Red}_{P,\varsigma}$ ist residual abgeschlossen.

Beweis:
Sei $t \xrightarrow{u} t'$ eine Reduktion und $v \in \mathrm{RedOcc}_{P,\varsigma}(t)$.
Nach Lemma 2.6, S. 29, ist die Menge aller Redexe eines Programms P, Red_P, residual abgeschlossen. Somit ist $v \setminus t \xrightarrow{u} t' \subseteq \mathrm{RedOcc}_P(t')$. Außerdem besagt Lemma 2.6, daß in den Fällen $v = u$, $v \parallel u$ und $u < v$ für alle $\hat{v} \in v \setminus t \xrightarrow{u} t'$ $t/v = t'/\hat{v}$. Also ist in diesen Fällen auch $v \setminus t \xrightarrow{u} t' \subseteq \mathrm{RedOcc}_{P,\varsigma}(t')$.

Sei nun $v < u$. Es ist u = v.k.u' für ein $k \in \mathbb{N}_+$ und $u' \in \mathbb{N}_+^*$. Da $v \in \mathrm{RedOcc}_{P,\varsigma}(t)$, ist

$$\begin{aligned} t/v &= f(t_1, \ldots, t_n) \\ t'/v &= f(t'_1, \ldots, t'_n) \end{aligned}$$

für $f^{(n)} \in \mathcal{F}(\dot\cup\, \mathcal{H})$ und $t_1, \ldots, t_n, t'_1, \ldots, t'_n \in \mathrm{T}_\Sigma$ mit

$$t_k \xrightarrow{u'} t'_k \tag{1}$$

$$\forall k \neq i \in [n].\ t_i = t'_i. \tag{2}$$

Nach Lemma 5.23 über Informationsgewinn bei Reduktion folgt aus (1)

$$[\![t_k]\!]^{\mathrm{alg}}_{\perp_\varsigma} \trianglelefteq [\![t'_k]\!]^{\mathrm{alg}}_{\perp_\varsigma} \tag{3}$$

und aus (2) folgt trivialerweise

$$\forall k \neq i \in [n].\ [\![t_i]\!]^{\mathrm{alg}}_{\perp_\varsigma} = [\![t'_i]\!]^{\mathrm{alg}}_{\perp_\varsigma}. \tag{4}$$

Da $v \in \mathrm{RedOcc}_{P,\varsigma}(t)$, ist f nicht erzwungen strikt für $([\![t_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\perp_\varsigma})$ und dieses ist syntaktisch ς-matchbar mit dem Pattern $(p_1, \ldots, p_n)$ eines Redexschemas $f(p_1, \ldots, p_n) \in \mathrm{RedS}_P$. Aufgrund von (3) und (4) trifft dies auch auf $([\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t'_n]\!]^{\mathrm{alg}}_{\perp_\varsigma})$ zu. Somit ist v eine ς-Redexstelle in t' und es gilt $v \setminus t \xrightarrow{u} t' = \{v\} \subseteq \mathrm{RedOcc}_{P,\varsigma}(t')$. □

Aus der residualen Abgeschlossenheit der ς-Redexe folgt gemäß Kapitel 2 die Gültigkeit des allgemeinen Residuenlemmas für die ς-Reduktion und die Konfluenz der ς-Reduktionsrelation $\xrightarrow[P,\varsigma]{}$.

Lemma 5.25 Wohldefiniertheit der ς-Reduktionssemantiken
Die allgemeine ς-Reduktionssemantik $[\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma}$ und die po-ς-Reduktionssemantik $[\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma}$ sind wohldefiniert, d. h. für alle $t \in \mathrm{T}_\Sigma$ existieren die kleinsten oberen Schranken von

$$T_{\mathrm{red}} := \{[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t'\}$$

und

$$T_{\mathrm{po}} := \{[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \xrightarrow[P,\mathrm{po},\varsigma]{*} t'\} = \{[\![t_i]\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid i \in \mathbb{N}_+, t = t_1 \xrightarrow[P,\mathrm{po},\varsigma]{} t_2 \xrightarrow[P,\mathrm{po},\varsigma]{} \ldots\}.$$

Beweis:
Wir zeigen, daß T_{red} eine abzählbare gerichtete Teilmenge und T_{po} sogar eine ω-Kette von $\mathrm{T}_{\mathcal{C},\varsigma}$ ist. Da T_Σ abzählbar ist, ist auch $T_{\text{red}} \subseteq [\![\mathrm{T}_\Sigma]\!]^{\text{alg}}_{\perp_\varsigma} = \{[\![t']\!]^{\text{alg}}_{\perp_\varsigma} \mid t \in \mathrm{T}_\Sigma\}$ abzählbar. Seien $[\![t']\!]^{\text{alg}}_{\perp_\varsigma}, [\![t'']\!]^{\text{alg}}_{\perp_\varsigma} \in T_{\text{red}}$. Somit ist $t \xrightarrow[P,\varsigma]{*} t'$ und $t \xrightarrow[P,\varsigma]{*} t''$. Aufgrund der Konfluenz der ς-Reduktionsrelation existiert ein $\hat{t} \in \mathrm{T}_\Sigma$ mit $t' \xrightarrow[P,\varsigma]{*} \hat{t}$ und $t'' \xrightarrow[P,\varsigma]{*} \hat{t}$. Mit Lemma 5.23 über Informationsgewinn bei Reduktion folgt $[\![t']\!]^{\text{alg}}_{\perp_\varsigma} \trianglelefteq [\![\hat{t}]\!]^{\text{alg}}_{\perp_\varsigma}$ und $[\![t'']\!]^{\text{alg}}_{\perp_\varsigma} \trianglelefteq [\![\hat{t}]\!]^{\text{alg}}_{\perp_\varsigma}$. Also ist T_{red} gerichtet.
Aus Lemma über Informationsgewinn bei Reduktion folgt auch direkt, daß T_{po} eine ω-Kette ist.
T_{red} und T_{po} besitzen als abzählbare gerichtete Teilmenge respektive ω-Kette aufgrund der ω-Vollständigkeit von $\langle \mathrm{T}_{\mathcal{C},\varsigma}, \trianglelefteq \rangle$ eine kleinste obere Schranke. □

5.4 Syntaktisches und semantisches ς-Matchen

Hier beweisen wir einige grundlegende Beziehungen zwischen dem syntaktischen und dem sematischen ς-Matchen.
In 5.4.1 zeigen wir zuerst Eigenschaften des syntaktischen ς-Matchens, analog zu den Eigenschaften des semantischen ς-Matchens, die wir in 5.2.3 bewiesen haben. Hieraus folgt in 5.4.2 der zentrale Satz über syntaktisches und semantisches ς-Matchen. Dieser bildet die Grundlage des Beweises der Übereinstimmung der ς-Fixpunkt- und der ς-Reduktionssemantiken. In 5.4.3 beweisen wir mit Hilfe dieses Satzes schon die Korrektheit der ς-Reduktion in der ς-Fixpunktsemantik.

5.4.1 Eigenschaften des syntaktischen ς-Matchens

Lemma 5.26 Syntaktisches ς-Matchen mit linearen Pattern
Sei $t \in \mathrm{T}_\Sigma$, $p \in \mathrm{T}_\mathcal{C}(X)$ ein lineares Pattern, $\sigma : X \to \mathrm{T}_\Sigma$.
Es gilt

$$p\sigma = t \tag{1}$$

genau dann, wenn

$$p[\perp/\mathrm{Var}(p)] \trianglelefteq t \tag{2}$$

und

$$x\sigma = t/u \text{ , wobei } \{u\} = \mathrm{Occ}(x,p) \text{ ist,} \tag{3}$$

für alle $x \in \mathrm{Var}(p)$ gilt.

Beweis:
Strukturelle Induktion über p.

$p = x$: $(x \in X)$.

(2) ist trivialerweise erfüllt, und es ist klar, daß (1):

$$p\sigma = x\sigma = t$$

genau dann gilt, wenn (3):

$$x\sigma = t/\varepsilon \text{ , wobei } \{\varepsilon\} = \mathrm{Occ}(x,p) \text{ ist,}$$

gilt.

$\underline{p = G(p_1, \ldots, p_n)}$: ($G^{(n)} \in \Sigma$, $p_1, \ldots, p_n \in \mathrm{T}_{\mathcal{C}}(X)$ linear).

$$t = p\sigma = G(p_1, \ldots, p_n)\sigma = G(p_1\sigma, \ldots, p_n\sigma)$$

gilt genau dann, wenn

$$t = G(t_1, \ldots, t_n)$$

und

$$\forall i \in [n].\ p_i\sigma = t_i$$

gilt. Dies wiederum gilt nach Induktionsvoraussetzung genau dann, wenn

$$t = G(t_1, \ldots, t_n),$$

$$\forall i \in [n].\ p_i[\bot/\mathrm{Var}(p_i)] \leq t_i$$

$$\forall i \in [n].\ \forall x \in \mathrm{Var}(p_i).\ x\sigma = t_i/u \text{ , wobei } \{u\} = \mathrm{Occ}(x, p_i) \text{ ist,}$$

gilt. Unter Zusammenfassung der ersten beiden Bedingungen gilt dies genau dann, wenn

$$p[\bot/\mathrm{Var}(p)] = G(p_1, \ldots, p_n)[\bot/\mathrm{Var}(p)] \trianglelefteq t$$

und

$$x\sigma = t/u \text{ , wobei } \{u\} = \mathrm{Occ}(x, p) \text{ ist,}$$

für alle $x \in \mathrm{Var}(p)$ gilt.

□

Die Linearität des Patterns p ist in obigem Lemma von entscheidender Bedeutung, da nur deshalb $|\mathrm{Occ}(x,p)| = |\{u\}| = 1$ gelten kann. Außerdem ist zu beachten, daß in (2) der Ausdruck t/u wohldefiniert ist: Aus dem Beweis geht implizit hervor, daß aus (1) $u \in \mathrm{Occ}(t)$ folgt.

Lemma 5.27 Syntaktisches ς-Matchen
Sei $f^{(n)}(\vec{p}) \in \mathrm{RedS}_P$, $\vec{t} \in (\mathrm{T}_\Sigma)^n$, $\sigma : X \to \mathrm{T}_\Sigma$.
$\vec{t}$ wird mit $f(\vec{p})$ von der Substitution σ genau dann syntaktisch ς-gematcht, wenn

$$\forall i \in [n].\ (\varsigma(f)(i) = \mathtt{tt} \ \Rightarrow\ [\![t_i]\!]^{\mathrm{alg}}_{\bot_\varsigma} \neq \bot) \wedge (p_i[\bot/\mathrm{Var}(p_i)] \trianglelefteq [\![t_i]\!]^{\mathrm{alg}}_{\bot_\varsigma}) \tag{1}$$

und

$$x\sigma = \vec{t}/u \text{ , wobei } \{u\} = \mathrm{Occ}(x, \vec{p}), \tag{2}$$

für alle $x \in \mathrm{Var}(\vec{p})$ gilt.
Beweis:

$\underline{\Leftarrow}$: Sei $i \in [n]$. Nach (2) ist

$$x\sigma = t_i/u \text{ , wobei } \{u\} = \mathrm{Occ}(x, p_i),$$

für alle $x \in \mathrm{Var}(p_i)$.

Zusammen mit (1) folgt nach Lemma 5.26 über syntaktisches ς-Matchen mit linearen Pattern

$$p_i\sigma = t_i.$$

Da dies für alle $i \in [n]$ gilt, ist $f(\vec{t})$ ein Redex zu $f(\vec{p})$. Zusammen mit (1) folgt gemäß der Definition des ς-Redexes und der Definition des syntaktischen ς-Matchens, daß $\vec{t}$ mit $f(\vec{p})$ unter σ syntaktisch ς-matchbar ist.

$\Rightarrow$: Nach Definition des ς-Redexes bzw. des semantischen ς-Matchens gilt (1). Da $f(\vec{t})$ ein Redex von $f(\vec{p})$ unter σ ist, ist

$$p_i\sigma = t_i$$

für alle $i \in [n]$. Daraus folgt mit Lemma 5.26 über syntaktisches ς-Matchen mit linearen Pattern, daß

$$x\sigma = t_i/u \text{ , wobei } \{u\} = \mathrm{Occ}(x, p_i),$$

für alle $x \in \mathrm{Var}(p_i)$ und $i \in [n]$ ist. Somit ist auch (2) gegeben.

□

Die Wohldefiniertheit von (2) ist aus den gleichen Gründen wie beim vorhergehenden Lemma 5.26 gegeben.

Korollar 5.28 Syntaktische ς-Matchbarkeit
Sei $f^{(n)}(\vec{p}) \in \mathrm{RedS}_P$, $\vec{t} \in (\mathrm{T}_\Sigma)^n$, $\sigma : X \to \mathrm{T}_\Sigma$.
$\vec{t}$ ist mit $f(\vec{p})$ genau dann syntaktisch ς-matchbar, wenn

$$\forall i \in [n].\ (\varsigma(f)(i) = \mathtt{tt} \ \Rightarrow\ [\![t_i]\!]^{\mathrm{alg}}_{\perp_\varsigma} \neq \perp) \wedge (p_i[\perp/\mathrm{Var}(p_i)] \trianglelefteq [\![t_i]\!]^{\mathrm{alg}}_{\perp_\varsigma})$$

gilt.

Beweis:
Im vorhergehenden Lemma 5.27 über syntaktisches ς-Matchen stellt nur (1) Anforderungen an $\vec{t}$ und $f(\vec{p})$, während (2) die Substitution σ festlegt. □

5.4.2 Der Satz über syntaktisches und semantisches ς-Matchen

Das folgende Vertauschungslemma dient allein als Hilfslemma für den darauf folgenden Satz.

Lemma 5.29 Vertauschung von Semantik und Teiltermbildung
Sei $t \in \mathrm{T}_{\mathcal{C}}(X)$ ein Konstruktorterm, $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ und $\beta : \mathrm{Var}(t) \to \mathrm{T}_{\mathcal{C},\varsigma}$ eine Variablenbelegung. Sei $u \in \mathrm{Occ}(t) \cap \mathrm{Occ}([\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta})$. Dann ist

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}/u = [\![t/u]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

Beweis:

$u = \varepsilon$: $[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}/\varepsilon = [\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![t/\varepsilon]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}$.

$u = u'.i$: $(i \in \mathbb{N}_+)$.

Da $u' \in \mathrm{Occ}(t) \cap \mathrm{Occ}([\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta})$, gilt nach Induktionsvoraussetzung

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}/u' = [\![t/u']\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}. \tag{1}$$

Da nach Voraussetzung $u'.i \in \mathrm{Occ}(t)$, kann nicht

$$t/u' = x \in X$$

sein, sondern es muß

$$t/u' = G(t_1, \ldots, t_n)$$

für ein $G^{(n)} \in \mathcal{C}$, $t_1, \ldots, t_n \in \mathrm{T}_{\mathcal{C}}(X)$, $n \geq i$ sein. Dies impliziert

$$t/u'.i = t_i. \tag{2}$$

Definitionsgemäß gilt

$$[\![t/u']\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = G^{\mathfrak{A}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}). \tag{3}$$

Da nach Voraussetzung $u'.i \in \mathrm{Occ}([\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta})$, kann aus (3) nicht

$$[\![t/u']\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = \bot$$

folgen, sondern es muß

$$[\![t/u']\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = G([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta})$$

sein. Daraus folgt

$$[\![t/u']\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}/i = [\![t_i]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}. \tag{4}$$

Insgesamt gilt nun:

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}/u = ([\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}/u')/i \overset{(1)}{=} [\![t/u']\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}/i \overset{(4)}{=} [\![t_i]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \overset{(3)}{=} [\![t/u'.i]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![t/u]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

□

Der folgende Satz ist von grundlegender Bedeutung für die Übereinstimmung der ς-Fixpunkt- und der ς-Reduktionssemantiken.

Satz 5.30 Syntaktisches und semantisches ς-Matchen
Sei $f^{(n)}(\vec{p}) \in \mathrm{RedS}_P$ und $\vec{t} \in (\mathrm{T}_\Sigma)^n$.

$\vec{t}$ ist mit $f(\vec{p})$ unter einer Substitution $\sigma : X \to \mathrm{T}_\Sigma$ syntaktisch ς-matchbar

$$\Longleftrightarrow$$

$([\![t_1]\!]^{\mathrm{alg}}_{\bot_\varsigma}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\bot_\varsigma})$ mit $f(\vec{p})$ semantisch ς-matchbar ist

$$\Longleftrightarrow$$

für alle Interpretationen $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ $([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}})$ mit $f(\vec{p})$ unter einer jeweiligen Variablenbelegung $\beta_{\mathfrak{A}} : X \to \mathrm{T}_{\mathcal{C},\varsigma}$ semantisch ς-matchbar ist.

Ist die syntaktische und semantische ς-Matchbarkeit gegeben, so besteht folgender Zusammenhang zwischen der Substitution σ und der Variablenbelegung $\beta_{\mathfrak{A}}$:

$$\beta_{\mathfrak{A}}(x) = [\![x\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}$$

für alle $x \in \mathrm{Var}(\vec{p})$, bzw. allgemeiner

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_{\mathfrak{A}}} = [\![t\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}$$

für alle $t \in \mathrm{T}_\Sigma(\mathrm{Var}(\vec{p}))$.

Beweis:
Nach Korollar 5.28 über syntaktische ς-Matchbarkeit ist $\vec{t}$ mit $f(\vec{p})$ genau dann syntaktisch ς-matchbar, wenn

$$\forall i \in [n].\ (\varsigma(f)(i) = \mathtt{tt}\ \Rightarrow\ [\![t_i]\!]^{\mathrm{alg}}_{\perp_\varsigma} \neq \perp) \wedge (p_i[\perp/\mathrm{Var}(p_i)] \trianglelefteq [\![t_i]\!]^{\mathrm{alg}}_{\perp_\varsigma})$$

gilt. Nach Korollar 5.9 über semantische ς-Matchbarkeit gilt dies wiederum genau dann, wenn $([\![t_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\perp_\varsigma})$ mit $f(\vec{p})$ semantisch ς-matchbar ist.
Mit der Monotonie der semantischen ς-Matchbarkeit gemäß Lemma 5.10, S. 84, folgt aus der semantischen ς-Matchbarkeit von $([\![t_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\perp_\varsigma})$ mit $f(\vec{p})$ auch die semantische ς-Matchbarkeit von $([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}})$ mit $f(\vec{p})$ für alle Interpretationen $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$. Da $\perp_\varsigma \in \mathrm{Int}_{\Sigma,\varsigma}$, ist natürlich auch die umgekehrte Schlußrichtung korrekt.

Wird $\vec{t}$ mit $f(\vec{p})$ unter σ syntaktisch ς-gematcht, so ist nach Lemma 5.27 über syntaktisches ς-Matchen

$$x\sigma = \vec{t}/u\ ,\ \text{wobei}\ \{u\} = \mathrm{Occ}(x, \vec{p}), \tag{1}$$

für alle $x \in \mathrm{Var}(\vec{p})$.
Wird $([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}})$ mit $f(\vec{p})$ unter $\beta_{\mathfrak{A}}$ semantisch ς-gematcht, so ist nach Lemma 5.8 über semantisches ς-Matchen

$$\beta_{\mathfrak{A}}(x) = ([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}})/u\ ,\ \text{wobei}\ \{u\} = \mathrm{Occ}(x, \vec{p}), \tag{2}$$

für alle $x \in \mathrm{Var}(\vec{p})$.
Aus letzterem folgt

$$\beta_{\mathfrak{A}}(x) = [\![t_i]\!]^{\mathrm{alg}}_{\mathfrak{A}}/u'\ ,\ \text{wobei}\ \{i.u'\} = \mathrm{Occ}(x, \vec{p}), \tag{3}$$

für alle $x \in \mathrm{Var}(\vec{p})$.
Aufgrund des Lemmas 5.29 über die Vertauschung von Semantik und Teiltermbildung ist

$$[\![t_i]\!]^{\mathrm{alg}}_{\mathfrak{A}}/u' = [\![t_i/u']\!]^{\mathrm{alg}}_{\mathfrak{A}} \tag{4}$$

mit $\{i.u'\} = \mathrm{Occ}(x, \vec{p})$ für alle $x \in \mathrm{Var}(\vec{p})$.
Insgesamt gilt somit:

$$\beta_{\mathfrak{A}}(x) \overset{(3)}{=} [\![t_i]\!]^{\mathrm{alg}}_{\mathfrak{A}}/u' \overset{(4)}{=} [\![t_i/u']\!]^{\mathrm{alg}}_{\mathfrak{A}} = [\![\vec{t}/u]\!]^{\mathrm{alg}}_{\mathfrak{A}} \overset{(1)}{=} [\![x\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}} \tag{5}$$

für alle $x \in \mathrm{Var}(\vec{p})$.
Für $t \in \mathrm{T}_\Sigma(\mathrm{Var}(\vec{p}))$ enthält $t\sigma$ keine Variablen mehr. Damit folgt aus (5) nach Lemma 4.2 über Substitutionsapplikation und algebraische Termsemantik, daß

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_{\mathfrak{A}}} = [\![t\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}$$

für alle $t \in \mathrm{T}_\Sigma(\mathrm{Var}(\vec{p}))$ gilt. □

Wenn nur für einige aber nicht für alle $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ $([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}})$ mit $f(\vec{p})$ semantisch ς-matchbar ist, muß $\vec{t}$ nicht mit $f(\vec{p})$ syntaktisch ς-matchbar sein:

Beispiel 5.7 Zum syntaktischen und semantischen ς-Matchen
Sei $f(p_1) := \mathtt{f(A)}$, $t_1 := \mathtt{a}$ und $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ mit $\mathtt{a}^{\mathfrak{A}} := \mathtt{A}$.
Somit ist $([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}}) = ([\![\mathtt{a}]\!]^{\mathrm{alg}}_{\mathfrak{A}}) = (\mathtt{A})$ mit `f(A)` semantisch ς-matchbar, aber nicht $(t_1) = (\mathtt{a})$ syntaktisch mit `f(A)`. □

5.4.3 Korrektheit der ς-Reduktion

Wir zeigen die Korrektheit von ς-Reduktionsfolgen in der ς-Fixpunktsemantik in mehreren Schritten.

Lemma 5.31 **Korrektheit der ς-Reduktion in Fixpunkten der ς-Transformation, I**
Sei $f(\vec{p})\to r$ eine Reduktionsregel, $\sigma : X\to \mathrm{T}_\Sigma$ eine Substitution und $\mathfrak{A} = \Phi_{P,\varsigma}(\mathfrak{A}) \in \mathrm{Int}_{\Sigma,\varsigma}$. Sei $f(\vec{p})\sigma$ ein ς-Redex. Dann gilt:

$$[\![f(\vec{p})\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}} = [\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}.$$

Beweis:
Da $f(\vec{p})\sigma$ ein ς-Redex ist, ist nach Definition des syntaktischen ς-Matchens $(p_1\sigma, \ldots, p_n\sigma)$ mit $f(\vec{p})$ unter σ syntaktisch ς-matchbar.
Mit Lemma 5.30 über syntaktisches und semantisches ς-Matchen folgt dann:
$([\![p_1\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![p_n\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}})$ ist mit $f(\vec{p})$ unter $\beta_{\mathfrak{A}} : X\to \mathrm{T}_{\mathcal{C},\varsigma}$, definiert durch $\beta_{\mathfrak{A}}(x) := [\![x\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}$ für alle $x \in X$, semantisch ς-matchbar; es gilt also

$$f^{\Phi_{P,\varsigma}(\mathfrak{A})}([\![p_1\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![p_n\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}) = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_{\mathfrak{A}}} \tag{1}$$

und außerdem ist

$$[\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_{\mathfrak{A}}} = [\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}. \tag{2}$$

Da $\mathfrak{A}$ ein Fixpunkt von $\Phi_{P,\varsigma}$ ist, gilt:

$$[\![f(\vec{p})\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}} = f^{\mathfrak{A}}([\![p_1\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![p_n\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}) = f^{\Phi_{P,\varsigma}(\mathfrak{A})}([\![p_1\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![p_n\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}). \tag{3}$$

Insgesamt folgt:

$$[\![f(\vec{p})\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}} \overset{(3)}{=} f^{\Phi_{P,\varsigma}(\mathfrak{A})}([\![p_1\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![p_n\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}) \overset{(1)}{=} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta_{\mathfrak{A}}} \overset{(2)}{=} [\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}.$$

□

Lemma 5.32 **Korrektheit der ς-Reduktion in Fixpunkten der ς-Transformation, II**
Sei $\mathfrak{A} = \Phi_{P,\varsigma}(\mathfrak{A}) \in \mathrm{Int}_{\Sigma,\varsigma}$. Für alle $t, t' \in \mathrm{T}_\Sigma$ gilt:

$$t \xrightarrow[\varsigma]{} t' \implies [\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}} = [\![t']\!]^{\mathrm{alg}}_{\mathfrak{A}}.$$

Beweis:
Sei $t \xrightarrow[\varsigma]{u} t'$ eine Reduktion. Nach Definition der Reduktion ist

$$\begin{array}{rcll} t/u & = & l\sigma & , \text{d. h.} \quad t = t[u \leftarrow l\sigma] \\ t' & = & t[u \leftarrow r\sigma] & \end{array}$$

für eine Reduktionsregel $l\to r \in \hat{P}$ und eine Substitution σ. Da eine ς-Reduktion vorliegt, ist $l\sigma$ ein ς-Redex. Mit dem vorhergehenden Lemma 5.31 folgt

$$[\![l\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}} = [\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}}.$$

Aus der Invarianz der algebraischen Termsemantik folgt dann direkt

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}} = [\![t[u \leftarrow l\sigma]]\!]^{\mathrm{alg}}_{\mathfrak{A}} = [\![t[u \leftarrow r\sigma]]\!]^{\mathrm{alg}}_{\mathfrak{A}} = [\![t']\!]^{\mathrm{alg}}_{\mathfrak{A}}.$$

□

Satz 5.33 **Korrektheit von ς-Reduktionsfolgen in der ς-Fixpunktsemantik**
Für alle $t, t' \in \mathrm{T}_\Sigma$ gilt:

$$t \xrightarrow[\varsigma]{*} t' \implies [\![t]\!]^{\mathrm{fix}}_{P,\varsigma} = [\![t']\!]^{\mathrm{fix}}_{P,\varsigma}$$

Beweis:
Folgt direkt aus dem vorhergehenden Lemma 5.32. □

5.5 Übereinstimmung der drei ς-Semantiken

Wie wir schon in Kapitel 4 bei der cbn-Semantik erwähnten, verwenden wir für den Beweis der Übereinstimmung der ς-Fixpunkt und der ς-Reduktionssemantik eine andere Methode als für den Beweis der Übereinstimmung der cbv-Fixpunkt- und der li-Reduktionssemantik. Die für die cbn-Semantik genannten Gründe treffen erst recht auf die allgemeineren ς-Semantiken zu. Anstelle der Verwendung von Datentypen von Grundtermsemantiken beweisen wir direkt die Übereinstimmung der drei Grundtermsemantiken.
Aus der bekannten Invarianz der algebraischen ς-Fixpunktsemantik folgt so auch direkt die Invarianz aller drei Grundtermsemantiken. Ein direkter Beweis der Invarianz der allgemeinen ς-Reduktionssemantik ist zwar möglich, aber der Beweis verläuft analog und ist fast ebenso komplex wie der in 5.5.2 angegebene Beweis für die Übereinstimmung der ς-Fixpunkt- und der allgemeinen ς-Reduktionssemantik.

Wir nennen eine Grundtermsemantik $[\![\cdot]\!] : \mathrm{T}_\Sigma \to \mathrm{T}_{\mathcal{C},\varsigma}$ genau dann **korrekt** bezüglich einer zweiten Grundtermsemantik $[\![\cdot]\!]' : \mathrm{T}_\Sigma \to \mathrm{T}_{\mathcal{C},\varsigma}$, wenn $[\![\cdot]\!] \preceq [\![\cdot]\!]'$ gilt, und **vollständig** , wenn $[\![\cdot]\!] \succeq [\![\cdot]\!]'$ ist.

Natürlich existieren in der schon genannten Literatur über Funktionsschemata viele Beweise der Übereinstimmung von Fixpunktsemantiken und verschiedenster Reduktionssemantiken. Diese lassen sich jedoch nur sehr begrenzt übertragen. [Vui74], [Manna74] und [Dow&Se76] betrachten nur syntaktisch stark eingeschränkte Funktionsschemata. Fast die gesamte Literatur über Funktionsschemta betrachtet auch, wie schon erwähnt, nur Basisdatentypen mit einer flachen Halbordnung und nutzt dies in Beweisen aus, wie wir es bei der cbv-Semantik getan haben.
Außerdem besitzen Funktionsschemata keine Pattern. Nicht jede Stelle eines Terms, an der sich ein Funktions- oder Hilfssymbol befindet, ist auch eine (ς-)Redexstelle. Für das Patternmatching haben wir sehr viele Aussagen über syntaktisches und semantisches ς-Matchen beweisen müssen. Dagegen sind Pattern im Rahmen beinahe orthogonaler Termersetzungssysteme, wie sie [O'Do77] behandelt, selbstverständlich. Leider werden dort nur terminierende Reduktionsfolgen und keine unendlichen Approximationsketten betrachtet. Das in [O'Do77] für den Beweis der Termination sogenannter eventually outermost Reduktionsfolgen benutzte Prinzip ist jedoch äußerst mächtig, und wir beweisen damit in 5.5.3 die Vollständigkeit der po-ς-Reduktionssemantik bezüglich der allgemeinen ς-Reduktionssemantik.

5.5.1 Der Übereinstimmungssatz

Einige Korrektheitsaussagen zwischen den drei Grundtermsemantiken lassen sich relativ leicht beweisen.

Lemma 5.34 **Korrektheit der ς-Reduktions- bezüglich der ς-Fixpunktsemantik**

$$[\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma},$$

$$[\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma}.$$

Beweis:
Sei $t \in \mathrm{T}_\Sigma$ und $t' \in \mathrm{T}_\Sigma$ mit $t \xrightarrow[\varsigma]{*} t'$.
Nach Satz 5.33, S. 113, über die Korrektheit einer ς-Reduktionsfolge in der ς-Fixpunktsemantik ist

$$[\![t]\!]^{\mathrm{fix}}_{P,\varsigma} = [\![t']\!]^{\mathrm{fix}}_{P,\varsigma}.$$

Es gilt $\bot_\varsigma \sqsubseteq \mathcal{D}^{\mathrm{fix}}_{P,\varsigma}$, und mit der Monotonie der algebraischen Termsemantik bezüglich der Algebra gemäß Lemma 5.16, S. 91, folgt

$$[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \trianglelefteq [\![t']\!]^{\mathrm{alg}}_{\mathcal{D}^{\mathrm{fix}}_{P,\varsigma}} = [\![t']\!]^{\mathrm{fix}}_{P,\varsigma}.$$

Insgesamt gilt

$$[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \trianglelefteq [\![t]\!]^{\mathrm{fix}}_{P,\varsigma},$$

und somit ist $[\![t]\!]^{\mathrm{fix}}_{P,\varsigma}$ eine obere Schranke für

$$\begin{aligned} T_{\mathrm{red}} &:= \{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[\varsigma]{*} t'\} \text{ und} \\ T_{\mathrm{po}} &:= \{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[\mathrm{po},\varsigma]{*} t'\} \end{aligned}$$

und es folgt

$$\begin{aligned} [\![t]\!]^{\mathrm{red}}_{P,\varsigma} &= \bigsqcup T_{\mathrm{red}} \preceq [\![t]\!]^{\mathrm{fix}}_{P,\varsigma}, \\ [\![t]\!]^{\mathrm{po}}_{P,\varsigma} &= \bigsqcup T_{\mathrm{po}} \preceq [\![t]\!]^{\mathrm{fix}}_{P,\varsigma}. \end{aligned}$$

□

Lemma 5.35 Korrektheit der po- bezüglich der allgemeinen ς-Reduktionssemantik

$$[\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma}$$

Beweis:
Sei $t \in \mathrm{T}_\Sigma$. Es ist

$$\{t' \mid t \xrightarrow[\mathrm{po},\varsigma]{*} t'\} \subseteq \{t' \mid t \xrightarrow[\varsigma]{*} t'\},$$

und somit

$$\{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[\mathrm{po},\varsigma]{*} t'\} \subseteq \{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[\varsigma]{*} t'\}.$$

Mit Lemma 2.1, S. 12, über Kofinalität und kleinste obere Schranken folgt

$$[\![t]\!]^{\mathrm{po}}_{P,\varsigma} = \bigsqcup\{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[\mathrm{po},\varsigma]{*} t'\} \trianglelefteq \bigsqcup\{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[\varsigma]{*} t'\} = [\![t]\!]^{\mathrm{red}}_{P,\varsigma}.$$

□

Wir wissen nun

$$[\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma}.$$

Mit einem Beweis von $[\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma}$ ließe sich der Kreis schließen, und die Übereinstimmung aller Grundtermsemantiken wäre bewiesen. Leider erweist sich ein solcher Beweis als äußerst problematisch. Daher beweisen wir stattdessen $[\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma}$ und $[\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma}$. Da auch diese Beweise aufwendig sind, führen wir sie in 5.5.2 respektive 5.5.3 separat auf, und halten hier schon fest:

Satz 5.36 Übereinstimmung der ς-Fixpunkt und der ς-Reduktionssemantiken

$$[\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma} = [\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} = [\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma}.$$

Beweis:
$[\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma}$ gemäß Lemma 5.34.
$[\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma}$ gemäß Lemma 5.34.
$[\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma}$ gemäß Lemma 5.35.
$[\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma}$ gemäß Lemma 5.39, S. 119.
$[\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma}$ gemäß Lemma 5.50, S. 130. □

Aufgrund der Übereinstimmung können wir eine Bezeichnung anstelle der drei verschiedenen verwenden.

Definition 5.14 ς-Datentyp, ς-Grundtermsemantik
Die ς-Interpretation über Σ

$$\mathcal{D}_{P,\varsigma} := \mathcal{D}^{\mathrm{fix}}_{P,\varsigma}$$

heißt **ς-Datentyp** und die Abbildung

$$[\![\cdot]\!]_{P,\varsigma} := [\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma} = [\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} = [\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma}$$

heißt **ς-Grundtermsemantik** der **ς-Semantik**. □

Wie angekündigt gilt:

Lemma 5.37 Invarianz der ς-Grundtermsemantik
Die ς-Grundtermsemantik ist invariant.

Beweis:
$[\![\cdot]\!]_{P,\varsigma} = [\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma} = [\![\cdot]\!]^{\mathrm{alg}}_{\mathcal{D}^{\mathrm{fix}}_{P,\varsigma}}$, und die algebraische Grundtermsemantik ist grundsätzlich invariant. □

5.5.2 Vollständigkeit der allgemeinen ς-Reduktions- bezüglich der ς-Fixpunktsemantik

Das von uns verwendete Beweisprinzip ist das folgende:
Wir zeigen, daß sich die ς-Fixpunktsemantik eines Terms $t \in \mathrm{T}_\Sigma$ schreiben läßt als

$$[\![t]\!]^{\mathrm{fix}}_{P,\varsigma} = \bigsqcup\{[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} \mid \mathfrak{A}_i = (\Phi_{P,\varsigma})^i(\bot_\varsigma),\ i \in \mathbb{N}\}.$$

Die allgemeine ς-Reduktionssemantik ist bekanntlich definiert durch

$$[\![t]\!]^{\mathrm{red}}_{P,\varsigma} = \bigsqcup\{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t'\}.$$

Beide Grundtermsemantiken sind also kleinste obere Schranken von Mengen. Wir beweisen, daß die kleinste obere Schranke der ς-Fixpunktsemantik kleiner-gleich der kleinsten oberen Schranke der allgemeinen ς-Reduktionssemantik ist, indem wir zeigen, daß die Menge, von der in der ς-Fixpunktsemantik die kleinste obere Schranke gebildet wird, kofinal in die Menge ist, von der in der allgemeinen ς-Reduktionssemantik die kleinste obere Schranke gebildet wird.
Da der folgende Beweis sehr komplex ist, beachte man auch das darauf folgende Beispiel.

Lemma 5.38 **Kofinalität der Approximationen der allgemeinen ς-Reduktionssemantik in die der ς-Fixpunktsemantik**

Sei $t \in \mathrm{T}_\Sigma$. Sei $\mathfrak{A}_i := (\Phi_{P,\varsigma})^i(\bot_\varsigma) \in \mathrm{Int}_{\Sigma,\varsigma}$ für alle $i \in \mathbb{N}$.
Dann existiert zu jedem $i \in \mathbb{N}$ ein $t' \in \mathrm{T}_\Sigma$ mit

$$t \xrightarrow[\varsigma]{*} t'$$

und

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} \trianglelefteq [\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma}.$$

Beweis:

$i = 0$: Setze $t' := t$ und es gilt trivialerweise

$$t \xrightarrow[\varsigma]{*} t' \text{ und } [\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} = [\![t]\!]^{\mathrm{alg}}_{\bot_\varsigma} = [\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma}.$$

$i \Rightarrow i+1$:

$t = f(t_1, \ldots, t_n)$: $(f^{(n)} \in \mathcal{F}(\dot\cup\, \mathcal{H}))$.

Fall 1: $([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}})$ ist mit der linken Seite einer Reduktionsregel $f(\vec{p}) \to r \in \hat{P}$ unter einer Variablenbelegung $\beta : X \to \mathrm{T}_{\mathcal{C},\varsigma}$ semantisch ς-matchbar.
Es gilt also

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}} = f^{\mathfrak{A}_{i+1}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}) = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A}_i,\beta}. \tag{1}$$

Nach Induktionsvoraussetzung der strukturellen Induktion existieren $t'_1, \ldots, t'_n \in \mathrm{T}_\Sigma$ mit

$$t_l \xrightarrow[\varsigma]{*} t'_l \tag{2}$$

und

$$[\![t_l]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}} \trianglelefteq [\![t'_l]\!]^{\mathrm{alg}}_{\bot_\varsigma} \tag{3}$$

für alle $l \in [n]$.
(2) impliziert

$$t = f(t_1, \ldots, t_n) \xrightarrow[\varsigma]{*} f(t'_1, \ldots, t'_n) \tag{4}$$

Aufgrund der Monotonie der algebraischen Termsemantik bezüglich der Algebra gemäß Lemma 5.16, S. 91, ist

$$[\![t'_l]\!]^{\mathrm{alg}}_{\bot_\varsigma} \trianglelefteq [\![t'_l]\!]^{\mathrm{alg}}_{\mathfrak{A}} \tag{5}$$

für alle $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$, $l \in [n]$.
Aus (3) und (5) folgt mit der nach Lemma 5.10, S. 84, gegebenen Monotonie des semantischen ς-Matchens, daß für alle Interpretationen $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$, insbesondere $\mathfrak{A}_i$, $([\![t'_1]\!]^{\mathrm{alg}}_{\mathfrak{A}}, \ldots, [\![t'_n]\!]^{\mathrm{alg}}_{\mathfrak{A}})$ mit $f(\vec{p})$ unter einer jeweiligen Variablenbelegung $\beta_{\mathfrak{A}}$ semantisch ς-matchbar ist, und

$$\beta \preceq \beta_{\mathfrak{A}} \tag{6}$$

ist.
Dann ist nach Satz 5.30, S. 110, über syntaktisches und semantisches ς-Matchen auch $(t'_1, \ldots, t'_n)$ mit $f(\vec{p})$ unter einer Substitution σ syntaktisch ς-matchbar, und es gilt

$$[\![\hat{t}\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} = [\![\hat{t}]\!]^{\mathrm{alg}}_{\mathfrak{A}_i,\beta_{\mathfrak{A}_i}} \tag{7}$$

für alle $\hat{t} \in \mathrm{T}_\Sigma(\mathrm{Var}(\vec{p}))$.
Die syntaktische ς-Matchbarkeit impliziert

$$f(t'_1, \ldots, t'_n) \xrightarrow[\varsigma]{} r\sigma. \tag{8}$$

(7) ist insbesondere für $\hat{t} = r$ gültig:

$$[\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A}_i, \beta_{\mathfrak{A}_i}} \tag{9}$$

Aus (6) folgt mit der Monotonie der algebraischen Termsemantik bezüglich der Variablenbelegung gemäß Lemma 5.17, S. 92:

$$[\![r]\!]^{\mathrm{alg}}_{\mathfrak{A}_i, \beta} \trianglelefteq [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A}_i, \beta_{\mathfrak{A}_i}}. \tag{10}$$

Schließlich existiert aufgrund der Induktionsvoraussetzung der Induktion über i ein $t' \in \mathrm{T}_\Sigma$ mit

$$r\sigma \xrightarrow[\varsigma]{*} t' \tag{11}$$

und

$$[\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} \trianglelefteq [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}. \tag{12}$$

Insgesamt ergeben nun (4), (8) und (11) zusammen

$$t = f(t_1, \ldots, t_n) \xrightarrow[\varsigma]{*} f(t'_1, \ldots, t'_n) \xrightarrow[\varsigma]{} r\sigma \xrightarrow[\varsigma]{*} t' \tag{13}$$

und es gilt

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}} \overset{(1)}{=} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A}_i, \beta} \overset{(10)}{\trianglelefteq} [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A}_i, \beta_{\mathfrak{A}_i}} \overset{(9)}{=} [\![r\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} \overset{(12)}{\trianglelefteq} [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}. \tag{14}$$

Fall 2: Sonst, d. h. es existiert keine passende Reduktionsregel.
Setze $t' := t$ und es gilt

$$t \xrightarrow[\varsigma]{*} t'$$

und

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}} = f^{\mathfrak{A}_i}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}) = \perp \trianglelefteq [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}.$$

$t = G(t_1, \ldots, t_n)$: $(G^{(n)} \in \mathcal{C})$.
Nach Induktionsvoraussetzung der strukturellen Induktion existieren $t'_1, \ldots, t'_n \in \mathrm{T}_\Sigma$ mit

$$t_l \xrightarrow[\varsigma]{*} t'_l \tag{1}$$

und

$$[\![t_l]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}} \trianglelefteq [\![t'_l]\!]^{\mathrm{alg}}_{\perp_\varsigma} \tag{2}$$

für alle $l \in [n]$.
Mit der Monotonie aller Operationen einer ς-Interpretation folgt aus (2)

$$G^{\mathfrak{A}_{i+1}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}) \trianglelefteq G^{\mathfrak{A}_{i+1}}([\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t'_n]\!]^{\mathrm{alg}}_{\perp_\varsigma}). \tag{3}$$

Außerdem sind die Operationen einer Konstruktorsymbols in allen ς-Interpretationen gleich:

$$G^{\mathfrak{A}_{i+1}} = G^{\mathfrak{A}_0} = G^{\perp_\varsigma}. \tag{4}$$

Setze nun $t' := G(t'_1, \ldots, t'_n)$. Aus (1) folgt

$$t = G(t_1, \ldots, t_n) \xrightarrow[\varsigma]{*} G(t'_1, \ldots, t'_n) = t'$$

und (3) und (4) ergeben zusammen

$$\begin{aligned} [\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}} &= G^{\mathfrak{A}_{i+1}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A}_{i+1}}) \\ &\overset{(3)}{\trianglelefteq} G^{\mathfrak{A}_{i+1}}([\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t'_n]\!]^{\mathrm{alg}}_{\perp_\varsigma}) \\ &\overset{(4)}{=} G^{\perp_\varsigma}([\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t'_n]\!]^{\mathrm{alg}}_{\perp_\varsigma}) \\ &= [\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma}. \end{aligned}$$

□

Beispiel 5.8 Zur Veranschaulichung des Beweises
Sei P das folgende Programm mit Pattern:

$$\begin{array}{lcl} \mathtt{a} & \to & \mathtt{A} \\ \mathtt{g(x)} & \to & \mathtt{G(a)} \\ \mathtt{h(x)} & \to & \mathtt{g(x)} \\ \mathtt{f(x)} & \to & \mathtt{x} \end{array}$$

Es sei ς = cbn. Es ergeben sich folgende Operationen für die ersten Approximationen des ς-Fixpunktdatentyps:

	$i=0$	$i=1$	$i=2$
$\mathtt{a}^{\mathfrak{A}_i}()$	$\perp$	$\mathtt{A}$	$\mathtt{A}$
$\mathtt{g}^{\mathfrak{A}_i}(\underline{t})$	$\perp$	$\mathtt{G}(\perp)$	$\mathtt{G}(\mathtt{A})$
$\mathtt{h}^{\mathfrak{A}_i}(\underline{t})$	$\perp$	$\perp$	$\mathtt{G}(\perp)$
$\mathtt{f}^{\mathfrak{A}_i}(\underline{t})$	$\perp$	$\underline{t}$	$\underline{t}$

Wir betrachten nun den Term $t = \mathtt{f}(\mathtt{h}(\mathtt{A})) \in \mathrm{T}_\Sigma$ im Schritt von $i = 1$ zu $i + 1 = 2$, wie im Fall $t = f(t_1, \ldots, t_n)$ und Fall 1 des Beweises beschrieben.
$([\![\mathtt{h}(\mathtt{A})]\!]^{\mathrm{alg}}_{\mathfrak{A}_2}) = (\mathtt{G}(\perp))$ ist mit der linken Seite der Reduktionsregel $\mathtt{f(x)}\to\mathtt{x}$ unter der Variablenbelegung $\beta = (\mathtt{x} \mapsto \mathtt{G}(\perp))$ semantisch ς-matchbar.
Es gilt also

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_2} = \mathtt{f}^{\mathfrak{A}_2}(\mathtt{G}(\perp)) = [\![\mathtt{x}]\!]^{\mathrm{alg}}_{\mathfrak{A}_1,\beta} = \mathtt{G}(\perp). \tag{1}$$

Es existiert $t'_1 = \mathtt{G}(\mathtt{a}) \in \mathrm{T}_\Sigma$ mit

$$t_1 := \mathtt{h}(\mathtt{A}) \xrightarrow[\varsigma]{} \mathtt{g}(\mathtt{A}) \xrightarrow[\varsigma]{} \mathtt{G}(\mathtt{a}) = t'_1 \tag{2}$$

und

$$[\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_2} = \mathtt{G}(\perp) \trianglelefteq \mathtt{G}(\perp) = [\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}. \tag{3}$$

(2) impliziert

$$t = \mathtt{f}(\mathtt{h}(\mathtt{A})) \xrightarrow[\varsigma]{*} \mathtt{f}(\mathtt{G}(\mathtt{a})). \tag{4}$$

Es gilt speziell

$$[\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma} = \mathtt{G}(\perp) \trianglelefteq \mathtt{G}(\mathtt{A}) = [\![t'_1]\!]^{\mathrm{alg}}_{\mathfrak{A}_1} \tag{5'}$$

$([\![t_1']\!]^{\mathrm{alg}}_{\mathfrak{A}_1}) = (\mathtt{G(A)})$ ist mit $\mathtt{f(x)}$ unter der Variablenbelegung $\beta_{\mathfrak{A}_1} = (\mathtt{x} \mapsto \mathtt{G(A)})$ semantisch ς-matchbar, und es ist

$$\beta = (\mathtt{x} \mapsto \mathtt{G}(\bot)) \preceq (\mathtt{x} \mapsto \mathtt{G(A)}) = \beta_{\mathfrak{A}_1}. \tag{6'}$$

Auch ist $(\mathtt{G(a)})$ mit $\mathtt{f(x)}$ unter der Substitution $\sigma = [\mathtt{G(a)/x}]$ syntaktisch ς-matchbar.
Es gilt somit

$$\mathtt{f(G(a))} \xrightarrow[\varsigma]{} \mathtt{x}\sigma = \mathtt{G(a)}, \tag{8}$$

und außerdem

$$[\![\mathtt{x}]\!]^{\mathrm{alg}}_{\mathfrak{A}_1,\beta_{\mathfrak{A}_1}} = \beta_{\mathfrak{A}_1}(\mathtt{x}) = \mathtt{G(A)} = [\![\mathtt{G(a)}]\!]^{\mathrm{alg}}_{\mathfrak{A}_1} = [\![\mathtt{x}\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}_1}. \tag{9}$$

Aus (6') folgt auch

$$[\![\mathtt{x}]\!]^{\mathrm{alg}}_{\mathfrak{A}_1,\beta} = \beta(\mathtt{x}) = \mathtt{G}(\bot) \trianglelefteq \mathtt{G(A)} = [\![\mathtt{x}]\!]^{\mathrm{alg}}_{\mathfrak{A}_1,\beta_{\mathfrak{A}_1}}. \tag{10}$$

Schließlich existiert $t' := \mathtt{G(A)} \in \mathrm{T}_\Sigma$ mit

$$\mathtt{x}\sigma = \mathtt{G(a)} \xrightarrow[\varsigma]{} \mathtt{G(A)} = t' \tag{11}$$

und

$$[\![\mathtt{x}\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}_1} = [\![\mathtt{G(a)}]\!]^{\mathrm{alg}}_{\mathfrak{A}_1} = \mathtt{G(A)} \trianglelefteq \mathtt{G(A)} = [\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma}. \tag{12}$$

Insgesamt ergeben nun (4), (8) und (11) zusammen

$$t = \mathtt{f(h(A))} \xrightarrow[\varsigma]{*} \mathtt{f(G(a))} \xrightarrow[\varsigma]{} \mathtt{G(a)} \xrightarrow[\varsigma]{} \mathtt{G(A)} = t'$$

und

$$[\![\mathtt{f(h(A))}]\!]^{\mathrm{alg}}_{\mathfrak{A}_2} \overset{(1)}{=} [\![\mathtt{x}]\!]^{\mathrm{alg}}_{\mathfrak{A}_1,\beta} = \mathtt{G}(\bot) \overset{(10)}{\trianglelefteq} \mathtt{G(A)} = [\![\mathtt{x}]\!]^{\mathrm{alg}}_{\mathfrak{A}_1,\beta_{\mathfrak{A}_1}} \overset{(9)}{=} [\![\mathtt{x}\sigma]\!]^{\mathrm{alg}}_{\mathfrak{A}_1} \overset{(12)}{\trianglelefteq} \mathtt{G(A)} = [\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma}.$$

□

Lemma 5.39 **Vollständigkeit der allgemeinen ς-Reduktions- bezüglich der ς-Fixpunktsemantik**

$$[\![\cdot]\!]^{\mathrm{fix}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma}.$$

Beweis:
Sei $t \in \mathrm{T}_\Sigma$. Sei $\mathfrak{A}_i := (\Phi_{P,\varsigma})^i(\bot_\varsigma) \in \mathrm{Int}_{\Sigma,\varsigma}$ für alle $i \in \mathbb{N}$.
Nach dem vorhergehenden Lemma 5.38 ist $\{[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} \mid i \in \mathbb{N}\}$ kofinal in $\{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[\varsigma]{*} t'\}$, und mit Lemma 2.1, S. 12, über Kofinalität und kleinste obere Schranken folgt

$$\bigsqcup_{i\in\mathbb{N}} [\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} \trianglelefteq \bigsqcup\{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[\varsigma]{*} t'\}.$$

Es gilt:

$$\begin{aligned}
& & [\![t]\!]^{\mathrm{fix}}_{P,\varsigma} & \\
& \text{(Definition)} & = [\![t]\!]^{\mathrm{alg}}_{\mathcal{D}^{\mathrm{fix}}_{P,\varsigma}} & \\
& \text{(Definition)} & = [\![t]\!]^{\mathrm{alg}}_{\bigsqcup_{i\in\mathbb{N}}\mathfrak{A}_i} & \\
& (\omega\text{-Stet. der alg. Termsemantik bzgl. der Algebra gemäß Lemma 5.16}) & = \bigsqcup_{i\in\mathbb{N}} [\![t]\!]^{\mathrm{alg}}_{\mathfrak{A}_i} & \\
& \text{(s.o.)} & \trianglelefteq \bigsqcup\{[\![t']\!]^{\mathrm{alg}}_{\bot_\varsigma} \mid t \xrightarrow[\varsigma]{*} t'\} & \\
& \text{(Definition)} & = [\![t]\!]^{\mathrm{red}}_{P,\varsigma}. &
\end{aligned}$$

□

BEMERKUNG 5.2: Zur po-ς-Reduktionssemantik
Leider läßt sich das auf der Kofinalität beruhende Beweisprinzip nicht für die po-ς-Reduktionssemantik verwenden. Das Problem der Übertragung des Lemmas 5.38 liegt im Schritt von (2) nach (4). Gemäß der dann verwendeten Induktionsvoraussetzung würden natürlich $t'_1, \ldots, t'_n \in \mathrm{T}_\Sigma$ mit

$$t_l \xrightarrow[\mathrm{po},\varsigma]{*} t'_l \tag{2*}$$

für alle $l \in [n]$ existieren. Es gilt jedoch dann im allgemeinen nicht:

$$t = f(t_1, \ldots, t_n) \xrightarrow[\mathrm{po},\varsigma]{*} f(t'_1, \ldots, t'_n) \tag{2*}$$

Das Problem liegt weniger darin, daß in allen n Teiltermen an den Stellen $1, \ldots, n$ jeweils gleichviele po-ς-Reduktionsschritte zu erfolgen haben, sondern vielmehr darin, daß die po-ς-Reduktion im allgemeinen schon vor Erreichen der $t'_1, \ldots, t'_n$ die Reduktion an der Stelle ε mit $f(\vec{p}) \to r$ fordert.
Man betrachte hierzu das vorhergehende Beispiel 5.8:
Es wird dort festgestellt, daß

$$t_1 := \mathtt{h(A)} \xrightarrow[\varsigma]{} \mathtt{g(A)} \xrightarrow[\varsigma]{} \mathtt{G(a)} = t'_1, \tag{2}$$

und daher

$$t = \mathtt{f(h(A))} \xrightarrow[\varsigma]{*} \mathtt{f(G(a))}. \tag{4}$$

gilt.
Es gilt auch

$$t_1 := \mathtt{h(A)} \xrightarrow[\mathrm{po},\varsigma]{} \mathtt{g(A)} \xrightarrow[\mathrm{po},\varsigma]{} \mathtt{G(a)} = t'_1, \tag{2*}$$

jedoch nicht

$$t = \mathtt{f(h(A))} \xrightarrow[\mathrm{po},\varsigma]{*} \mathtt{f(G(a))} =: \hat{t}. \tag{4*}$$

Mit $\hat{t}$ ist der Induktionsschritt nicht durchführbar. Es läßt sich zwar in (13) anstelle von $r\sigma$ verwenden, aber in (14) ist $r\sigma$ nicht durch $\hat{t}$ ersetzbar, da

$$[\![\mathtt{r}]\!]^{\mathrm{alg}}_{\mathfrak{A}_1} = \beta(\mathtt{x}) = \mathtt{G}(\bot) \rhd \bot = [\![\mathtt{h(A)}]\!]^{\mathrm{alg}}_{\mathfrak{A}_1} = [\![\hat{\mathtt{t}}]\!]^{\mathrm{alg}}_{\mathfrak{A}_1}.$$

Deshalb wird die Übereinstimmung der po-ς-Reduktionssemantik und der allgemeinen ς-Reduktionssemantik im folgenden rein operationell gezeigt. Daraus folgt die Übereinstimmung der po-ς-Reduktionssemantik mit der ς-Fixpunktsemantik. Somit gilt die Aussage des Lemmas 5.38 über die Kofinalität der semantischen Approximationen sehr wohl auch für die po-ς-Reduktionssemantik; denn die kleinsten oberen Schranken der Approximationen der po-ς-Reduktions- und der ς-Fixpunktsemantik sind gleich, die Approximationen sind ω-Ketten ω-kompakter Elemente der ω-induktiven Halbordnung der unendlichen Terme, und damit sind nach einer Anmerkung in 2.2 die beiden Approximationen gegenseitig kofinal. □

5.5.3 Vollständigkeit der po-ς- bezüglich der allgemeinen ς-Reduktionssemantik

Wir führen nun den letzten und aufwendigsten Teil des Beweises der Übereinstimmung der drei Grundtermsemantiken der ς-Semantik.

Die wesentliche Beweisidee besteht darin, mit der allgemeinen Menge der eventually outermost Reduktionsfolgen zu arbeiten, statt nur po-ς-Reduktionsfolgen zu betrachten. Grob gesprochen

heißt eine ς-Reduktionsfolge eventually outermost, wenn jede outermost ς-Redexstelle, die in einem der Terme der Folge vorkommt, im weiteren Verlauf der ς-Reduktionsfolge wieder eliminiert wird, d. h. an der Stelle selbst wird reduziert oder durch eine andere Reduktion entsteht oberhalb der Stelle eine neue ς-Redexstelle, so daß die betrachtete Stelle non-outermost wird.
Das verwendete Beweisprinzip stammt, wie schon erwähnt, aus [O'Do77]. Das dortige Theorem 10 besagt, daß eine eventually outermost Reduktionsfolge immer terminiert, wenn der Ausgangsterm überhaupt eine Normalform besitzt. Dieses Theorem beruht auf dem davor aufgeführten Lemma 17, welches mit Hilfe einer komplexen Konstruktion von Reduktionen beweist, daß das Residuum einer eventually outermost Reduktionsfolge wiederum eine eventually outermost Reduktionsfolge ist (das Residuum einer Reduktionsfolge ist in Def. 2.3, S. 34, definiert worden).
Wir definieren diese Hilfskonstruktion zuerst in Definition 5.17 und geben dann den Beweis zerlegt in vier Lemmata (5.44 – 5.47) an. Einerseits wird auf diese Weise der Beweis übersichtlicher; andererseits verwenden wir die eventually outermost Hilfskonstruktion bzw. einige dieser Lemmata, um noch weiterführende Aussagen über die mit eventually outermost ς-Redukionsfolgen erreichbaren semantischen Approximationen zu zeigen (Lemmata 5.48 und 5.49). Daraus können wir dann insgesamt die gewünschte Vollständigkeit der po-ς- bezüglich der allgemeinen ς-Reduktionssemantik schließen.
Eine ausführliche Darstellung des Beweises des Lemmas 17 und Theorems 10 aus [O'Do77], der dort außerdem für sogenannte kombinatorische Reduktionssysteme erweitert wird, befindet sich im Anhang von [Ber&Klop86].

Zuerst zeigen wir, daß die Menge der ς-Redexe outer ist. Diese Eigenschaft ist Voraussetzung für die meisten der folgenden Lemmata.

Lemma 5.40 Outer-Eigenschaft der ς-Redexe
Die Redexmenge $\mathrm{Red}_{P,\varsigma}$ ist outer.

Beweis:
Sei $t \xrightarrow[l\longrightarrow r,\varsigma]{w} t'$ eine ς-Reduktion, $u < v < w$ Stellen, $v \in \mathrm{RedOcc}_{P,\varsigma}(t)$, $u \in \mathrm{RedOcc}_{P,\varsigma}(t')$.
Es gibt somit $v', w' \in \mathbb{N}_+^*$ und $k \in \mathbb{N}_+$ mit $v = u.k.v'$ und $w = v.w' = u.k.v'.w'$.
Da $\mathrm{Red}_{P,\varsigma} \subseteq \mathrm{Red}_P$, folgt nach Lemma 2.10 über die Outer-Eigenschaft der Redexe, daß $u \in \mathrm{RedOcc}_P(t)$.
Somit ist

$$\begin{aligned} t/u &= f(t_1, \ldots, t_n) \\ t'/u &= f(t'_1, \ldots, t'_n) \end{aligned}$$

für $f^{(n)} \in \mathcal{F}(\dot\cup\, \mathcal{H})$ und $t_1, \ldots, t_n, t'_1, \ldots, t'_n \in \mathrm{T}_\Sigma$ mit

$$t_i = t'_i \tag{1}$$

für alle $k \neq i \in [n]$ und

$$t_k \xrightarrow[l\longrightarrow r,\varsigma]{v'.w'} t'_k.$$

Da $v' \in \mathrm{RedOcc}_{P,\varsigma}(t)$, gilt $t_k \xrightarrow[\mathrm{no},\varsigma]{v'.w'} t'_k$. Nach Lemma 5.23, S. 105, über Informationsgewinn bei Reduktion ist somit

$$[\![t_k]\!]^{\mathrm{alg}}_{\perp_\varsigma} = [\![t'_k]\!]^{\mathrm{alg}}_{\perp_\varsigma}. \tag{2}$$

Da $u \in \mathrm{RedOcc}_{P,\varsigma}(t')$, ist f nicht erzwungen strikt für $([\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t'_n]\!]^{\mathrm{alg}}_{\perp_\varsigma})$, und $([\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t'_n]\!]^{\mathrm{alg}}_{\perp_\varsigma})$ ist semantisch ς-matchbar mit dem Pattern $\vec{p}$ eines Redexschemas $f(\vec{p}) \in \mathrm{RedS}_P$. Aufgrund von (1) und (2) trifft dies auch auf $([\![t_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\perp_\varsigma})$ zu. Somit ist $u \in \mathrm{RedOcc}_{P,\varsigma}(t)$. □

Die folgenden, auf [O'Do77] zurückgehenden Definitionen und Lemmata gelten nicht nur für unsere Programme mit einer erzwungenen Striktheit. Im folgenden sei daher I eine Instanz eines beinahe orthogonalen Termersetzungssystems R mit der I-Redexmenge $\mathrm{Red}_{R,I}$.

Wir definieren nun formal die Elimination einer outermost Redexstelle und eventually outermost Reduktionsfolgen.

Definition 5.15 Elimination einer outermost Redexstelle (vgl. Def. 30 in [O'Do77])

Sei $t_1 \xrightarrow[I]{U_1}\!\!\!\!\rightarrow\ t_2 \xrightarrow[I]{U_2}\!\!\!\!\rightarrow\ t_3 \xrightarrow[I]{U_3}\!\!\!\!\rightarrow \ldots$ eine Reduktionsfolge mit $u \in \mathrm{Outer}_{R,I}(t_k)$, $k \in \mathbb{N}_+$. Die outermost Redexstelle u wird genau dann in t_l **eliminiert**, wenn l der kleinste Index größer k mit

$$u \in U_{l-1} \text{ oder } u \notin \mathrm{Outer}_{R,I}(t_l)$$

ist. □

Man beachte, daß für alle $i \in \{k, k+1, \ldots, l-2\}$

$$u \setminus t_i \xrightarrow[I]{U_i}\!\!\!\!\rightarrow\ t_{i+1} = \{u\}.$$

Wenn $u \in U_{l-1}$, so ist

$$u \setminus t_{l-1} \xrightarrow[I]{U_{l-1}}\!\!\!\!\rightarrow\ t_l = \emptyset,$$

was nicht $u \notin \mathrm{RedOcc}_{R,I}(t_l)$ impliziert; andernfalls ist

$$u \setminus t_{l-1} \xrightarrow[I]{U_{l-1}}\!\!\!\!\rightarrow\ t_l = \{u\} \subseteq \mathrm{NOuter}_{R,I}(t_l).$$

Definition 5.16 Eventually outermost ς-Reduktionsfolge
(vgl. Def. 31 in [O'Do77]; Def. 7.5 von outermost-fair in [Ber&Klop86]).
Eine I-Reduktionsfolge $t_1 \xrightarrow[I]{}\!\!\!\!\rightarrow\ t_2 \xrightarrow[I]{}\!\!\!\!\rightarrow\ t_3 \xrightarrow[I]{}\!\!\!\!\rightarrow \ldots$ heißt genau dann **eventually outermost**, wenn

$$\forall k \in \mathbb{N}_+.\ \forall u \in \mathrm{Outer}_{R,I}(t_k).\ \exists l > k.\ u \text{ wird in } t_l \text{ eliminiert.}$$

□

Man beachte, daß jede endliche eventually outermost Reduktionsfolge mit der I-Normalform des Startterms t_1 endet.

Die folgenden Eigenschaften von Reduktionen (5.41 – 5.43) werden wir im weiteren häufig brauchen.

Lemma 5.41 Erhalt von unabhängigen non-outermost Redexstellen
Sei $\mathrm{Red}_{R,I}$ residual abgeschlossen. Dann gilt:

$$t \xrightarrow[I]{U}\!\!\!\!\rightarrow\ t' \text{ ist eine } I\text{-Reduktion},\ U \parallel V,\ V \subseteq \mathrm{NOuter}_{R,I}(t) \implies V \subseteq \mathrm{NOuter}_{R,I}(t').$$

Beweis:

Sei $v \in V$. Wegen $v \parallel U$ ist $v \setminus t \xrightarrow[I]{U}\!\!\!\!\rightarrow\ t' = \{v\}$, also aufgrund der residualen Abgeschlossenheit $v \in \mathrm{RedOcc}_{R,I}(t')$. Da $v \in \mathrm{NOuter}_{R,I}(t)$, existiert $w \in \mathrm{RedOcc}_{R,I}(t)$ mit $w < v$. Für alle $u \in U$ ist $u \not\leq w$, weil $u \parallel v$. Somit ist $w \setminus t \xrightarrow[I]{U}\!\!\!\!\rightarrow\ t' = \{w\}$, also $w \in \mathrm{RedOcc}_{R,I}(t')$. Insgesamt folgt $v \in \mathrm{NOuter}_{R,I}(t')$. □

Korollar 5.42 Zerlegung einer Reduktion
Sei $\mathrm{Red}_{R,I}$ residual abgeschlossen. Wenn $t \xrightarrow[I]{U}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t'$ eine I-Reduktion ist, so ist auch

$$t \xrightarrow[\mathrm{o},I]{U_1}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t'' \xrightarrow[\mathrm{no},I]{U_2}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t'$$

mit $U_1 := U \cap \mathrm{Outer}_{R,I}(t)$ und $U_2 := U \setminus \mathrm{Outer}_{R,I}(t)$ eine I-Reduktionsfolge.

Beweis:
Die Möglichkeit der Zerlegung einer Reduktion folgt direkt aus der Definition der parallen Reduktion unabhängiger Redexe. Mit Lemma 5.41 über den Erhalt unabängiger non-outermost Redexstellen folgt auch $U_2 \subseteq \mathrm{NOuter}_{R,I}(t'')$. □

Lemma 5.43 Non-outermost Reduktion erzeugt keine outermost Redexstellen
Sei $\mathrm{Red}_{R,I}$ residual abgeschlossen und outer. Dann gilt:

$$t \xrightarrow[\mathrm{no},I]{}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t' \implies \mathrm{Outer}_{R,I}(t') \subseteq \mathrm{Outer}_{R,I}(t).$$

Beweis:
Sei $t \xrightarrow[\mathrm{no},I]{u}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t'$ eine I-Reduktion und $v \in \mathrm{Outer}_{R,I}(t')$.

1. Z.z.: Es existiert kein $w < v$ mit $w \in \mathrm{RedOcc}_{R,I}(t)$.

   Angenommen, ein solches w existiert. Dann existiert insbesondere ein $w < v$ mit $w \in \mathrm{Outer}_{R,I}(t)$. Da $u \in \mathrm{NOuter}_{R,I}(t)$, ist $w \setminus t \xrightarrow[\mathrm{no},I]{u}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t' = \{w\} \subseteq \mathrm{RedOcc}_{R,I}(t')$. Dies steht im Widerspruch zu der Annahme $v \in \mathrm{Outer}_{R,I}(t')$.

2. Z.z.: $v \in \mathrm{RedOcc}_{R,I}(t)$.

   Nach 1. kann nicht $u < v$ sein, und es bleiben die folgenden 2 Fälle.

   $u \parallel v$: Dann ist $t/v = t'/v$ und somit $v \in \mathrm{RedOcc}_{R,I}(t)$.

   $v \leq u$: Da $u \in \mathrm{NOuter}_{R,I}(t)$, existiert $w \in \mathrm{RedOcc}_{R,I}(t)$ mit $w < u$ und wegen 1. gilt $v \leq w < u$.

      $v = w$: Also ist $v \in \mathrm{RedOcc}_{R,I}(t)$.

      $v < w$: Aus der Outer-Eigenschaft von $\mathrm{Red}_{R,I}$ folgt direkt $v \in \mathrm{RedOcc}(t)$.

3. Z.z.: $v \in \mathrm{Outer}_{R,I}(t)$.

   Dies folgt direkt aus 1. und 2.

Wegen $t \xrightarrow[\mathrm{no},I]{U}\!\!\!\!\rightarrow\!\!\!\!\rightarrow t'$ gdw. $t \xrightarrow[\mathrm{no},I]{u_1} t_2 \xrightarrow[\mathrm{no},I]{u_2} \ldots \xrightarrow[\mathrm{no},I]{u_n} t'$ mit $U = \{u_1, \ldots, u_n\}$ (Erhalt von unabhängigen non-outermost Redexstellen, Lemma 5.41) folgt schließlich induktiv $\mathrm{Outer}_{R,I}(t') \subseteq \mathrm{Outer}_{R,I}(t)$. □

Die umgekehrte Inklusion ($\mathrm{Outer}_{R,I}(t) \subseteq \mathrm{Outer}_{R,I}(t')$) gilt auch, siehe Lemma 14 in [O'Do77], wird aber im folgenden nicht benötig.

Die eventually outermost Hilfskonstruktion und die darauf folgenden vier Lemmata gehen auf Lemma 17 in [O'Do77] zurück.

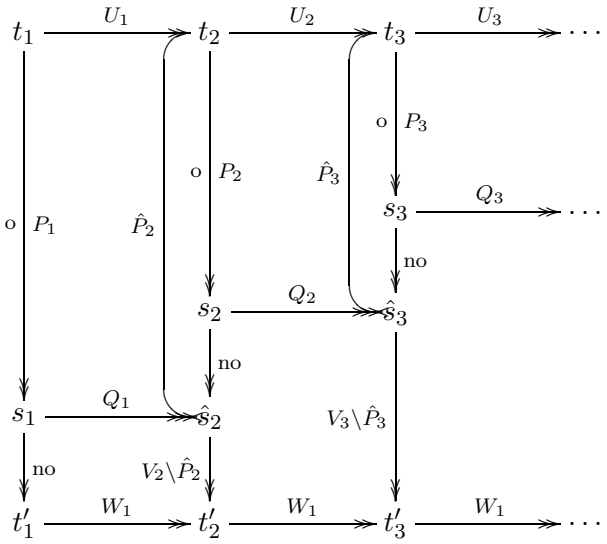


Abbildung 5.1: Die eventually outermost Hilfskonstruktion.

Definition 5.17 Eventually outermost Hilfskonstruktion
Sei $\mathrm{Red}_{R,I}$ residual abgeschlossen und outer. Sei $A = t_1 \xrightarrow[R,I]{U_1}\!\!\!\!\rightarrow\, t_2 \xrightarrow[R,I]{U_2}\!\!\!\!\rightarrow \ldots$ eine Reduktionsfolge und $B = t_1 \xrightarrow[R,I]{V_1}\!\!\!\!\rightarrow\, t'_1$ eine I-Reduktion.
Die **eventually outermost Hilfskonstruktion zu A nach B**, dargestellt in Abbildung 5.1, ist definiert als das Tupel

$$\begin{aligned}&((t_i)_{i\in\mathbb{N}_+}, (t'_i)_{i\in\mathbb{N}_+}, (s_i)_{i\in\mathbb{N}_+}, (\hat{s}_{i+1})_{i\in\mathbb{N}_+},\\ &(P_i)_{i\in\mathbb{N}_+}, (\hat{P}_{i+1})_{i\in\mathbb{N}_+}, (Q_i)_{i\in\mathbb{N}_+}, (U_i)_{i\in\mathbb{N}_+}, (V_i)_{i\in\mathbb{N}_+}, (W_i)_{i\in\mathbb{N}_+}),\end{aligned}$$

wobei die Bestandteile wie folgt definiert sind:

$$((t_i)_{i\in\mathbb{N}_+}, (t'_i)_{i\in\mathbb{N}_+}, (U_i)_{i\in\mathbb{N}_+}, (V_i)_{i\in\mathbb{N}_+}, (W_i)_{i\in\mathbb{N}_+})$$

ist die Residuenkonstruktion zu A nach B.
Für alle $i \in \mathbb{N}_+$ sind definiert:

$$\begin{aligned}P_1 &:= V_1 \cap \mathrm{Outer}_{R,I}(t_1)\\ \hat{P}_{i+1} &:= P_i \setminus t_i \xrightarrow[R,I]{U_i}\!\!\!\!\rightarrow\, t_{i+1} = P_i \setminus U_i \quad (\text{da } P_i \subseteq \mathrm{Outer}_{R,I}(t_i))\\ P_{i+1} &:= \hat{P}_{i+1} \cap \mathrm{Outer}(t_{i+1})\\ Q_i &:= U_i \setminus t_i \xrightarrow[R,I]{P_i}\!\!\!\!\rightarrow\, s_i\end{aligned}$$

s_i ist durch den I-Reduktionsschritt $t_i \xrightarrow[R,I]{P_i}\!\!\!\!\rightarrow\, s_i$ definiert.

Da $P_i \subseteq V_i$, gilt $s_i \xrightarrow[R,I]{V_i\setminus P_i}\!\!\!\!\rightarrow\, t'_i$. Dabei wird $t_1 \xrightarrow[R,I]{V_1}\!\!\!\!\rightarrow\, t'_1$ gemäß Korollar 5.42 in eine outermost und eine non-outermost I-Reduktion $t_1 \xrightarrow[R,\mathrm{o},I]{P_1}\!\!\!\!\rightarrow\, s_1 \xrightarrow[R,\mathrm{no},I]{V_1\setminus P_1}\!\!\!\!\rightarrow\, t'_1$ zerlegt.

$\hat{s}_{i+1}$ ist durch $t_{i+1} \xrightarrow[R,I]{\hat{P}_{i+1}}\!\!\!\!\rightarrow\, \hat{s}_{i+1}$ und $s_i \xrightarrow[R,I]{Q_i}\!\!\!\!\rightarrow\, \hat{s}_{i+1}$ aufgrund des allgemeinen Residuenlemmas wohldefiniert.

$t_{i+1} \xrightarrow[R,I]{\hat{P}_{i+1}} \hat{s}_{i+1}$ wird gemäß Korollar 5.42 in eine outermost und eine non-outermost I-Reduktion $t_{i+1} \xrightarrow[R,\mathrm{o},I]{P_{i+1}} s_{i+1} \xrightarrow[R,\mathrm{no},I]{\hat{P}_{i+1}\setminus P_i} \hat{s}_{i+1}$ zerlegt.

Man beachte, daß $\hat{s}_{i+1} \xrightarrow[R,I]{V_{i+1}\setminus\hat{P}_{i+1}} t'_{i+1}$, weil $\hat{P}_{i+1} \subseteq V_{i+1}$. □

Lemma 5.44 Konvergenz in der eventually outermost Hilfskonstruktion
Sei $\mathrm{Red}_{R,I}$ residual abgeschlossen und outer. Sei $A = t_1 \xrightarrow[R,I]{U_1} t_2 \xrightarrow[R,I]{U_2} \ldots$ eine I-Reduktionsfolge und $B = t_1 \xrightarrow[R,I]{V_1} t'_1$ eine I-Reduktion. Sei die eventually outermost Hilfskonstruktion zu A nach B mit den in Definition 5.17 verwendeten Bezeichnern gegeben.
Dann existiert ein $l > 1$ mit $s_i = t_i$ und $U_i = Q_i$ für alle $i \geq l$.

Beweis:
Nach Definition der eventually outermost Hilfskonstruktion ist $P_1 = \{p_1, \ldots, p_n\} \subseteq \mathrm{Outer}_{R,I}(t_1)$. Da A eventually outermost ist, existiert zu jedem p_j ein $l_j > 1$, so daß p_j in t_{l_j} eliminiert wird.
Wir zeigen nun für alle $i \in \mathbb{N}_+$:

$$P_i = \{p_j \in P_1 \mid l_j > i\}.$$

$i = 1$: Da $l_j > 1$, ist $P_1 = \{p_j \in P_1 \mid l_j > 1\}$.

$i \Rightarrow i+1$: Nach Definition ist

$$P_{i+1} = (P_i \setminus U_i) \cap \mathrm{Outer}_{R,I}(t_{i+1}).$$

Es folgt zusammen mit der Induktionsvoraussetzung

$$P_{i+1} \subseteq P_i = \{p_j \in P_1 \mid l_j > i\}.$$

Wir betrachten alle $p_j \in P_i$:

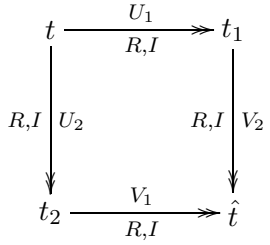
$l_j = i+1$: Nach Definition der Elimination einer outermost Redexstelle ist $p_j \in U_i$ oder $p_j \notin \mathrm{Outer}_{R,I}(t_{i+1})$. Somit $p_j \notin P_{i+1}$.

$l_j > i+1$: Nach Definition der Elimination einer outermost Redexstelle ist $p_j \notin U_i$ und $p_j \in \mathrm{Outer}_{R,I}(t_{i+1})$. Also $p_j \in P_{i+1}$.

Insgesamt ist also $P_{i+1} = \{p_j \in P_1 \mid l_j > i+1\}$.

Somit ist $P_i = \emptyset$ für alle $i \geq l := \max\{l_j \mid j \in [n]\}$. Wegen $t_i \xrightarrow[R,I]{P_i} s_i$ folgt $s_i = t_i$ und $U_i = Q_i$ für alle $i \geq l$. □

Lemma 5.45 Erhalt einer outermost *I*-Redexstelle
Sei $\mathrm{Red}_{R,I}$ residual abgeschlossen und outer. Gegeben seien die folgenden I-Reduktionsschritte



mit

$$\begin{aligned} V_1 &:= U_1 \setminus t \xrightarrow[R,I]{U_2}\!\!\!\!\!\rightarrow t_2 \\ V_2 &:= U_2 \setminus t \xrightarrow[R,I]{U_1}\!\!\!\!\!\rightarrow t_1 \end{aligned}$$

und außerdem sei

$$\begin{aligned} u &\in \mathrm{Outer}_{R,I}(t) \cap \mathrm{Outer}_{R,I}(\hat{t}) \\ u &\notin U_2 \cup V_1 \end{aligned}$$

Dann ist auch

$$\begin{aligned} u &\in \mathrm{Outer}_{R,I}(t_1) \\ u &\notin U_1 \cup V_2 \end{aligned}$$

Beweis:
Sei $(v] := \{v' \in \mathbb{N}_+^* \mid v' \leq v\}$ für beliebige $v \in \mathbb{N}_+^*$.

1. Z.z.: $u \notin U_1$.

Angenommen, $u \in U_1$. Da $u \in \mathrm{Outer}_{R,I}(t)$ und $u \notin U_2$, ist $(u] \cap U_2 = \emptyset$. Nach Definition der Residuenabbildung ist dann $u \in U_1 \setminus t \xrightarrow[R,I]{U_2}\!\!\!\!\!\rightarrow t_2 = V_1$. Dies steht im Widerspruch zu den Voraussetzungen.

2. Z.z.: $(u] \cap V_2 = \emptyset$.

Angenommen, $u' \in (u] \cap V_2 \subseteq U_2 \setminus t \xrightarrow[R,I]{U_1}\!\!\!\!\!\rightarrow t_1$. Nach Definition der Residuenabbildung existiert dann ein $u'' \in U_1 \cup U_2$ mit $u'' \leq u'$, d. h. $u'' \in (u]$. Dies steht im Widerspruch dazu, daß $u \in \mathrm{Outer}(t)$ und $u \notin U_1 \cup U_2 \subseteq \mathrm{RedOcc}_{R,I}(t)$, und somit $(u] \cap (U_1 \cup U_2) = \emptyset$.

3. Z.z.: $u \notin V_2$.

Dies folgt direkt aus 2.

4. Z.z.: $u \in \mathrm{Outer}_{R,I}(t_1)$.

Da $u \in \mathrm{Outer}_{R,I}(t)$ und $u \notin U_1$, ist $(u] \cap U_1 = \emptyset$. Somit ist $u \setminus t \xrightarrow[R,I]{U_1}\!\!\!\!\!\rightarrow t_1 = \{u\}$ und also $u \in \mathrm{RedOcc}_{R,I}(t_1)$.

Angenommen, $u \notin \mathrm{Outer}_{R,I}(t_1)$. Dann existiert $u' \in \mathrm{Outer}_{R,I}(t_1)$ mit $u' < u$. Da $(u'] \subseteq (u]$, ist nach 2. $(u'] \cap V_2 = \emptyset$. Gemäß der Definition der Residuenabbildung ist dann $u' \setminus t_1 \xrightarrow[R,I]{V_2}\!\!\!\!\!\rightarrow \hat{t} = \{u'\}$. Somit ist $u' \in \mathrm{RedOcc}_{R,I}(\hat{t})$. Da $u' < u$, ist $u \notin \mathrm{Outer}_{R,I}(\hat{t})$. Dies steht im Widerspruch zu den Voraussetzungen.

□

Lemma 5.46 Erhalt einer outermost Redexstelle in der eventually outermost Hilfskonstruktion

Sei die eventually outermost Hilfskonstruktion mit den in Definition 5.17 verwendeten Bezeichnern gegeben.
Sei $u \in \mathrm{Outer}_{R,I}(t'_i)$ und $u \notin W_i$ für alle $i \in \mathbb{N}_+$. Dann ist $u \in \mathrm{Outer}_{R,I}(s_i)$ und $u \notin Q_i$ für alle $i \in \mathbb{N}_+$.

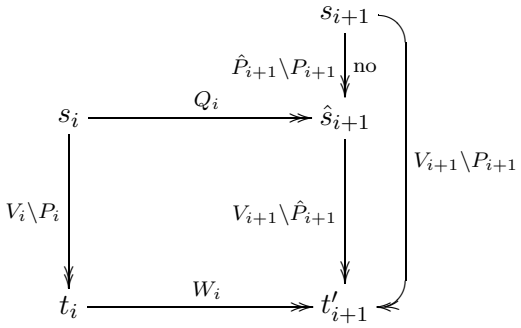
Beweis:
Wir zeigen durch vollständige Induktion für alle $i \in \mathbb{N}_+$:

1. $i > 1 \implies u \notin Q_{i-1}$.
2. $u \in \mathrm{Outer}_{R,I}(s_i)$.
3. $u \notin V_i \setminus P_i$.

$i = 1$:

1. Offensichtlich.
2. Da $V_1 \setminus P_1 \subseteq \mathrm{NOuter}_{R,I}(s_1)$ und $u \in \mathrm{Outer}_{R,I}(t'_1)$ und nach Lemma 5.43 non-outermost I-Reduktion keine outermost I-Redexstellen erzeugen kann, ist $u \in \mathrm{Outer}_{R,I}(s_1)$.
3. Wegen $V_1 \setminus P_1 \subseteq \mathrm{NOuter}_{R,I}(s_1)$ und $u \in \mathrm{Outer}_{R,I}(s_1)$ nach 2. ist $u \notin V_1 \setminus P_1$.

$i \Rightarrow i+1$: Folgende Reduktionen sind jetzt von Bedeutung:



Es ist

$$W_i = Q_i \setminus s_i \xrightarrow[R,I]{V_i \setminus P_i}\!\!\!\!\!\!\rightarrow\; t_i = U_i \setminus t_i \xrightarrow[R,I]{V_i}\!\!\!\!\!\!\rightarrow\; t'_i,$$

$$V_{i+1} \setminus \hat{P}_{i+1} = (V_i \setminus P_i) \setminus s_i \xrightarrow[R,I]{Q_i}\!\!\!\!\!\!\rightarrow\; s_{i+1}.$$

Nach den allgemeinen Voraussetzungen gilt

$$\begin{array}{lcl} u & \in & \mathrm{Outer}_{R,I}(t'_{i+1}) \\ u & \notin & W_i \end{array}$$

Nach Induktionsvoraussetzung gilt

$$\begin{array}{lcl} u & \in & \mathrm{Outer}_{R,I}(s_i) \\ u & \notin & V_i \setminus P_i \end{array}$$

Mit Lemma 5.45 über den Erhalt einer outermost I-Redexstelle folgt dann

$$\begin{array}{lcl} u & \in & \mathrm{Outer}_{R,I}(\hat{s}_{i+1}) \\ u & \notin & Q_i \cup (V_{i+1} \setminus \hat{P}_{i+1}). \end{array}$$

1. Nach obigem ist $u \notin Q_i$.

2. Da $\hat{P}_{i+1} \setminus P_{i+1} \subseteq \mathrm{NOuter}_{R,I}(s_{i+1})$ und $u \in \mathrm{Outer}_{R,I}(\hat{s}_{i+1})$ und nach Lemma 5.43 non-outermost I-Reduktion keine outermost I-Redexstellen erzeugen kann, ist $u \in \mathrm{Outer}_{R,I}(s_{i+1})$.
3. Da $\hat{P}_{i+1} \setminus P_{i+1} \subseteq \mathrm{NOuter}_{R,I}(s_{i+1})$ und $u \in \mathrm{Outer}_{R,I}(s_{i+1})$, ist $u \notin \hat{P}_{i+1} \setminus P_{i+1}$. Zusammen mit $u \notin V_{i+1} \setminus \hat{P}_{i+1}$ folgt $u \notin (\hat{P}_{i+1} \setminus P_{i+1}) \cup (V_{i+1} \setminus \hat{P}_{i+1}) = V_{i+1} \setminus P_{i+1}$.

□

Lemma 5.47 Abgeschlossenheit der eventually outermost Reduktionsfolgen unter Residuenbildung

Sei $\mathrm{Red}_{R,I}$ residual abgeschlossen und outer. Sei $A = t_1 \xrightarrow[R,I]{} t_2 \xrightarrow[R,I]{} \ldots$ eine I-Reduktionsfolge und $B = t_1 \xrightarrow[R,I]{V_1} t'_1$ eine I-Reduktion.
Dann ist $A \setminus B$ eine eventually outermost I-Reduktionsfolge.

Beweis:
Sei $A = t_1 \xrightarrow[R,I]{U_1} t_2 \xrightarrow[R,I]{U_2} \ldots$ und $A \setminus B = t'_1 \xrightarrow[R,I]{W_1} t'_2 \xrightarrow[R,I]{W_2} \ldots$.
Angenommen, $A \setminus B$ ist nicht eventually outermost. O. B. d. A. existiert eine Redexstelle $u \in \mathrm{Outer}_{R,I}(t'_1)$, die in $A \setminus B$ nicht eliminiert wird, d. h. $u \in \mathrm{Outer}_{R,I}(t'_i)$ und $u \notin W_i$ für alle $i \in \mathbb{N}_+$.
Betrachten wir nun die eventually outermost Hilfskonstruktion zu A nach B mit den in Definition 5.17 verwendeten Bezeichnern.
Aus Lemma 5.46 über den Erhalt eines outermost Redexes in der eventually outermost Hilfskonstruktion folgt, daß $u \in \mathrm{Outer}_{R,I}(s_i)$ und $u \notin Q_i$ für alle $i \in \mathbb{N}_+$.
Nach Lemma 5.44 über die Konvergenz in der eventually outermost Hilfskonstruktion existiert ein $l > 1$ mit $s_i = t_i$ und $U_i = Q_i$ für alle $i \geq l$.
Beide Eigenschaften zusammen besagen, daß $u \in \mathrm{Outer}_{R,I}(t_i)$ und $u \notin U_i$ für alle $i > l$, für ein $l > 1$. Demnach wird $u \in \mathrm{Outer}_{R,I}(t_l)$ nie eliminiert, und A ist nicht eventually outermost. Dies steht im Widerspruch zu der Voraussetzung. □

Mit Hilfe der vorausgegangenen Lemmata beweisen wir nun Aussagen über die mit eventually outermost Reduktionsfolgen erreichbaren semantischen Approximationen. Damit weisen wir schließlich die Vollständigkeit der po-ς- bezüglich der allgemeinen ς-Reduktionssemantik nach. Diese Aussagen gelten natürlich nur noch für Programme P mit einer erzwungenen Striktheit ς.

Lemma 5.48 Einfache Dominanz von eventually outermost ς-Reduktionsfolgen

Sei $A = t_1 \xrightarrow[P,\varsigma]{} t_2 \xrightarrow[P,\varsigma]{} \ldots$ eine ς-Reduktionsfolge und $B = t_1 \xrightarrow[P,\varsigma]{V_1} t'_1$ eine ς-Reduktion.
Dann existiert ein $l \in \mathbb{N}_+$ mit

$$[\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma} \trianglelefteq [\![t_l]\!]^{\mathrm{alg}}_{\perp_\varsigma}.$$

Beweis:
Betrachten wir die eventually outermost Hilfskonstruktion zu A nach B mit den in Definition 5.17 verwendeten Bezeichnern.
Wir zeigen durch vollständige Induktion, daß $[\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma} \trianglelefteq [\![s_i]\!]^{\mathrm{alg}}_{\perp_\varsigma}$ für alle $i \in \mathbb{N}_+$.

$i = 1$: Es ist $t_1 \xrightarrow[P,\mathrm{o},\varsigma]{} s_1 \xrightarrow[P,\mathrm{no},\varsigma]{} t'_1$. Nach Lemma 5.23, S. 105, über Informationsgewinn bei Reduktion ist $[\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma} = [\![s_1]\!]^{\mathrm{alg}}_{\perp_\varsigma}$

$\underline{i \Rightarrow i+1:}$ Es gilt $s_i \xrightarrow[P,\varsigma]{} \hat{s}_{i+1}$ und $s_{i+1} \xrightarrow[P,\mathrm{no},\varsigma]{} \hat{s}_{i+1}$. Nach Lemma 5.23 über Informationsgewinn bei Reduktion ist $[\![s_i]\!]^{\mathrm{alg}}_{\perp_\varsigma} \trianglelefteq [\![\hat{s}_{i+1}]\!]^{\mathrm{alg}}_{\perp_\varsigma} = [\![s_{i+1}]\!]^{\mathrm{alg}}_{\perp_\varsigma}$. Zusammen mit der Induktionsvoraussetzung $[\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma} \trianglelefteq [\![s_i]\!]^{\mathrm{alg}}_{\perp_\varsigma}$ folgt $[\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma} \trianglelefteq [\![s_{i+1}]\!]^{\mathrm{alg}}_{\perp_\varsigma}$.

Nach Lemma 5.44 über die Konvergenz in der eventually outermost Hilfskonstruktion existiert ein $l \in \mathbb{N}_+$ mit $t_l = s_l$.
Insgesamt gilt somit $[\![t'_1]\!]^{\mathrm{alg}}_{\perp_\varsigma} \trianglelefteq [\![t_l]\!]^{\mathrm{alg}}_{\perp_\varsigma}$. □

Lemma 5.49 Dominanz von eventually outermost Reduktionsfolgen
Sei $A = t_1 \xrightarrow[P,\varsigma]{*} t'$ eine endliche ς-Reduktionsfolge. Sei $B = t_1 \xrightarrow[P,\varsigma]{} t_2 \xrightarrow[P,\varsigma]{} \ldots$ eine eventually outermost ς-Reduktionsfolge.
Dann existiert ein $k \in \mathbb{N}_+$ mit

$$[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \trianglelefteq [\![t_k]\!]^{\mathrm{alg}}_{\perp_\varsigma}.$$

Beweis:
Vollständige Induktion über die Länge n der ς-Reduktionsfolge $A = t_1 \xrightarrow[P,\varsigma]{n} t'$.

$\underline{n = 0:}$ Für $k := 1$ gilt $[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} = [\![t_1]\!]^{\mathrm{alg}}_{\perp_\varsigma} = [\![t_k]\!]^{\mathrm{alg}}_{\perp_\varsigma}$.

$\underline{n \Rightarrow n+1:}$ Sei $A = t_1 \xrightarrow[P,\varsigma]{U} t'_1 \xrightarrow[P,\varsigma]{n} t'$.

Da nach Lemma 5.47 eventually outermost ς-Reduktionsfolgen unter Residuenbildung abgeschlossen sind, ist

$$B' := B \setminus t_1 \xrightarrow[P,\varsigma]{U} t'_1 = t'_1 \xrightarrow[P,\varsigma]{} t'_2 \xrightarrow[P,\varsigma]{} \ldots$$

eventually outermost.

Nach Induktionsvoraussetzung existiert ein $k' \in \mathbb{N}_+$ mit

$$[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \trianglelefteq [\![t'_{k'}]\!]^{\mathrm{alg}}_{\perp_\varsigma}.$$

Man betrachte die Residuenkonstruktion von B nach $t_1 \xrightarrow[P,\varsigma]{U} t'_1$:

$$\begin{array}{ccccccc}
B = t_1 & \xrightarrow[P,\varsigma]{} & t_2 & \xrightarrow[P,\varsigma]{} & \cdots & \cdots \xrightarrow[P,\varsigma]{} & t_{k'} \xrightarrow[P,\varsigma]{} t_{k'+1} \xrightarrow[P,\varsigma]{} \cdots \\
\downarrow{\scriptstyle P,\varsigma} & & \downarrow{\scriptstyle P,\varsigma} & & & & \downarrow{\scriptstyle P,\varsigma} \quad \downarrow{\scriptstyle P,\varsigma} \\
B' = t'_1 & \xrightarrow[P,\varsigma]{} & t'_2 & \xrightarrow[P,\varsigma]{} & \cdots & \cdots \xrightarrow[P,\varsigma]{} & t'_{k'} \xrightarrow[P,\varsigma]{} t'_{k'+1} \xrightarrow[P,\varsigma]{} \cdots
\end{array}$$

Es ist

$$t'_{k'} \xrightarrow[P,\varsigma]{} t'_{k'+1} \xrightarrow[P,\varsigma]{} \ldots = (t_{k'} \xrightarrow[P,\varsigma]{} t_{k'+1} \xrightarrow[P,\varsigma]{} \ldots) \setminus (t_{k'} \xrightarrow[P,\varsigma]{} t'_{k'}),$$

und gemäß Lemma 5.48 über einfache Dominanz von eventually outermost Reduktionsfolgen existiert ein $k \geq k'$ mit

$$[\![t'_{k'}]\!]^{\mathrm{alg}}_{\perp_\varsigma} \trianglelefteq [\![t_k]\!]^{\mathrm{alg}}_{\perp_\varsigma}.$$

Insgesamt gilt also

$$[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \trianglelefteq [\![t_k]\!]^{\mathrm{alg}}_{\perp_\varsigma}.$$

Das Beweisprinzip dieses und des letzten Lemmas (5.48) ist noch einmal in Abbildung 5.2 skizziert.
□

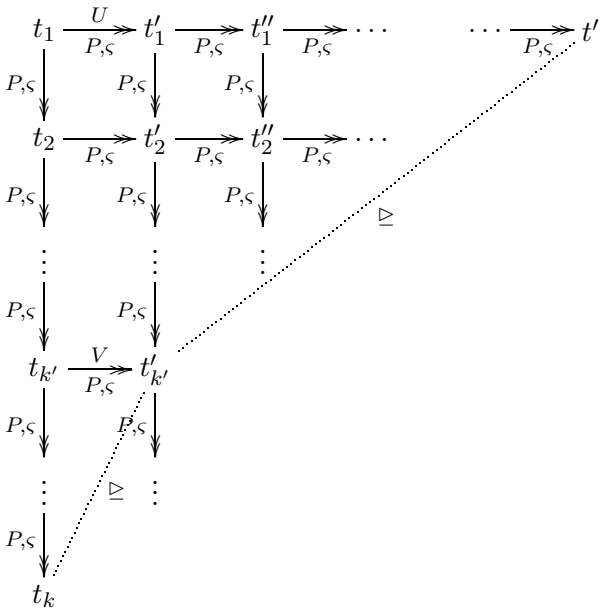


Abbildung 5.2: Beweisprinzip der Lemmata 5.48 und 5.49

Lemma 5.50 Vollständigkeit der po- bezüglich der allgemeinen ς-Reduktionssemantik

$$[\![\cdot]\!]^{\mathrm{red}}_{P,\varsigma} \preceq [\![\cdot]\!]^{\mathrm{po}}_{P,\varsigma}.$$

Beweis:
Sei $t \in \mathrm{T}_\Sigma$. Seien

$$\begin{aligned} T_{\mathrm{red}} &:= \{[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t'\} \\ T_{\mathrm{po}} &:= \{[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \xrightarrow[P,\mathrm{po},\varsigma]{*} t'\} \end{aligned}$$

Sei nun $[\![t_{\mathrm{red}}]\!]^{\mathrm{alg}}_{\perp_\varsigma} \in T_{\mathrm{red}}$, d. h. $t \xrightarrow[P,\varsigma]{*} t_{\mathrm{red}}$.
Betrachten wir dann

$$t = t_1 \xrightarrow[P,\mathrm{po},\varsigma]{} t_2 \xrightarrow[P,\mathrm{po},\varsigma]{} \ldots.$$

Die po-ς-Reduktionsfolge ist eventually outermost. Somit existiert gemäß Lemma 5.49 über die Dominanz von eventually outermost ς-Reduktionsfolgen ein $k \in \mathbb{N}_+$ mit

$$[\![t_{\mathrm{red}}]\!]^{\mathrm{alg}}_{\perp_\varsigma} \trianglelefteq [\![t_k]\!]^{\mathrm{alg}}_{\perp_\varsigma}.$$

Da $[\![t_k]\!]^{\mathrm{alg}}_{\perp_\varsigma} \in T_{\mathrm{po}}$, ist also T_{red} kofinal in T_{po}.
Mit Lemma 2.1, S. 12, über Kofinalität und kleinste obere Schranken folgt dann

$$[\![t]\!]^{\mathrm{red}}_{P,\varsigma} = \bigsqcup T_{\mathrm{red}} \trianglelefteq \bigsqcup T_{\mathrm{po}} = [\![t]\!]^{\mathrm{po}}_{P,\varsigma}.$$

□

5.5.4 Effizientere ς-Reduktionssemantiken

Bei der Definition der Normalformsemantik und später der allgemeinen ς-Reduktionssemantik haben wir darauf hingewiesen, daß diese aufgrund ihres Nicht-Determinismus bei der Reduktion kaum die Grundlage einer Implementierung bilden können. Diese Unzulänglichkeit wurde mit den daraufhin definierten, auf deterministischen Reduktionsstrategien beruhenden Reduktionssemantiken, hauptsächlich der po-ς-Reduktionssemantik beseitigt. Wir haben die Effizienz der po-ς-Reduktionssemantik jedoch nicht weiter betrachtet, da das Auffinden von effizienten ς-Reduktionsstrategien nicht Ziel dieser Arbeit ist. Aus dem letzten Abschnitt ergeben sich diesbezüglich jedoch einige Anregungen, die wir kurz diskutieren wollen.
Für die Untersuchung der Effizienz benötigen wir ein Kostenmaß. Wir verwenden dafür die Anzahl der Redexstellen, an denen reduziert wird bis die Normalform erreicht ist. Für ein gründliches Studium der Effizienz wäre sicherlich auch die Betrachtung unendlicher Reduktionsfolgen, welche partielle oder unendliche Konstruktorterme approximieren, sinnvoll.

Im vorangegangenen Abschnitt 5.5.3 haben wir zwar am Ende nur noch die po-ς-Reduktionsstrategie betrachtet, aber im Prinzip die Eignung jeder beliebigen eventually outermost ς-Reduktionsstrategie für eine ς-Reduktionssemantik nachgewiesen. Die po- ist im allgemeinen keinesfalls die effizienteste aller eventually outermost ς-Reduktionsstrategien.

Es ist bekannt, daß bei reinen outermost (ς-)Reduktionsstrategien häufig Redexe vervielfacht werden, die im weiteren Verlauf der Reduktion alle wieder reduziert werden. Dieses Problem wird in Implementierungen jedoch durch die Speicherung von Termen in azyklischen Graphen und die dadurch mögliche Verwendung gemeinsamer Teilterme gelöst. Man spricht hier auch von call-by-need anstelle von call-by-name Auswertung (siehe Kapitel VI in [O'Do77] oder [Fie&Har88]).

Außerdem dürfen wir nicht vergessen, daß outermost ς-Redexstellen nicht nur durch Reduktion an ihnen selbst eliminiert werden können, sondern auch dadurch, daß oberhalb von ihnen neue ς-Redexstellen entstehen.
Das folgende Beispiel verdeutlicht dies.

Beispiel 5.9

$$\begin{array}{lcl} \texttt{and(x,False)} & \rightarrow & \texttt{False} \\ \texttt{a} & \rightarrow & \texttt{b} \\ \texttt{b} & \rightarrow & \texttt{False} \end{array}$$

$$\texttt{and}(\texttt{a},\underline{\texttt{b}}) \xrightarrow[\text{o,cbn}]{2} \underline{\texttt{and}}(\texttt{a},\texttt{False}) \xrightarrow[\text{o,cbn}]{\varepsilon} \texttt{False}.$$

Bei po-cbn-Reduktion wäre im ersten Reduktionschritt zusätzlich und völlig unnötig an der outermost Redexstelle 1 reduziert worden. □

Somit kann es durchaus sinnvoll sein, in einem Reduktionsschritt nur an einer echten Teilmenge der outermost ς-Redexstellen zu reduzieren.
Eine Verallgemeinerung dieses Beispiels zeigt auch, daß nicht jede die Semantik approximierende ς-Reduktionsfolge eventually outermost sein muß:

Beispiel 5.10

$$\begin{array}{lcl} \texttt{and(x,False)} & \to & \ldots \\ \texttt{a} & \to & \ldots \\ \texttt{b} & \to & \ldots \end{array}$$

$$\texttt{and}(\texttt{a}, \underline{\texttt{b}}) \xrightarrow[\text{o,cbn}]{2} \texttt{and}(\texttt{a}, \ldots) \xrightarrow[\text{o,cbn}]{} \ldots$$

In dem Term `and(a,b)` kann auf eine Reduktion an der outermost cbn-Redexstelle 1 wie schon vorhin verzichtet werden. Nur an der anderen outermost cbn-Redexstelle 2 muß reduziert werden. Wird dort nach eventuell wiederholter Reduktion schließlich `False` erreicht, so wird durch die Reduktion an der Stelle ε auch die ursprüngliche outermost cbn-Redexstelle 1 eliminiert.
Ist jedoch `False` dort nicht erreichbar, so ist schon $[\![\texttt{and}(\texttt{a},\texttt{b})]\!]_{P,\text{cbn}} = [\![\texttt{and}(\texttt{a},\texttt{b})]\!]^{\text{alg}}_{\perp_\varsigma} = \perp$, und die Reduktionsfolge approximiert somit die Semantik ebenfalls. Diese (unendliche) cbn-Reduktionsfolge ist jedoch nicht eventually outermost, da 1 nie eliminiert wird. □

Es stellt sich nun die Frage, wie wir die Vollständigkeit derartiger ς-Redukionsstrategien beweisen können.
Wie schon erwähnt, wird auch im Anhang von [Ber&Klop86] ein Beweis für den Erhalt der eventually outermost Eigenschaft bei Residuenbildung geführt. Der dortige Beweis ist auch in sofern allgemeiner, als dort nicht eine feste Aufteilung der Redexstellen eines Terms in outermost und non-outermost Redexstellen vorausgesetzt wird. Stattdessen darf eine beliebige Aufteilung von $\text{RedOcc}_{P,\varsigma}(t)$ in zwei Mengen $\text{Gain}_{P,\varsigma}(t)$ (**gaining ς-Redexstellen**) und $\text{NGain}_{P,\varsigma}(t)$ (**non-gaining ς-Redexstellen**) mit $\text{RedOcc}_{P,\varsigma}(t) = \text{Gain}_{P,\varsigma}(t) \dot\cup \text{NGain}_{P,\varsigma}(t)$ vorliegen. Wenn dann für diese Aufteilung analog zu den Lemmata 5.41, 5.43 und 5.45 gilt, daß unabhängige non-gaining ς-Redexstellen erhalten bleiben, non-gaining ς-Redexstellen keine gaining ς-Redexstellen erzeugen und gaining ς-Redexstellen erhalten bleiben, so sind auch analog zu Lemma 5.47 die analog zu Definition 5.16 definierten eventually gaining ς-Reduktionsfolgen unter Residuenbildung abgeschlossen, denn der gegebene Beweis von Lemma 5.47 beruht nur auf diesen drei Eigenschaften der outermost ς-Redexstellen. Ist $\text{Gain}_{P,\varsigma}(t) \subseteq \text{Outer}_{P,\varsigma}(t)$, so lassen sich mit Sicherheit auch Teile der Beweise der Lemmata 5.41, 5.43 und 5.45 zum Beweisen der drei Eigenschaften wiederverwenden.

Wir haben die Bezeichnung gaining ς-Redexstellen natürlich nicht ohne Grund gewählt. Wenn noch analog zu Lemma 5.23, S. 105, über Informationsgewinn bei Reduktion

$$\begin{array}{rcl} t \xrightarrow[P]{u} t',\ u \in \text{Gain}_{P,\varsigma}(t) & \Longrightarrow & [\![t]\!]^{\text{alg}}_{\perp_\varsigma} \trianglelefteq [\![t']\!]^{\text{alg}}_{\perp_\varsigma} \\ t \xrightarrow[P]{u} t',\ u \in \text{NGain}_{P,\varsigma}(t) & \Longrightarrow & [\![t]\!]^{\text{alg}}_{\perp_\varsigma} = [\![t']\!]^{\text{alg}}_{\perp_\varsigma} \end{array}$$

gilt, so ist auch die Dominanz von eventually gaining ς-Reduktionsfolgen analog zu Lemma 5.49 gegeben.
Jede ς-Reduktionsstrategie, die auf derartigen eventually gaining ς-Reduktionsfolgen beruht, ist also vollständig bezüglich der ς-Fixpunktsemantik. Da nach Lemma 5.34, S. 113, jede ς-Reduktionssemantik korrekt ist, stimmt eine solche eventually gaining ς-Reduktionssemantik mit der ς-Grundtermsemantik überein.

In unserem obigen Beispiel sollte also in `and(a,b)` die Stelle 2 zu den gaining und 1 zu den non-gaining cbn-Redexstellen gehören. In Kapitel 7 werden wir noch eine Anregung für eine allgemeine Definition von $\text{Gain}_{P,\varsigma}(t)$ und $\text{NGain}_{P,\varsigma}(t)$ geben.

Kapitel 6

Untersuchung der ς-Semantiken

Wir haben die ς-Semantiken unserer Programme sowohl denotationell als auch operationell definiert. Wir untersuchen jetzt die Beziehungen der ς-Semantiken zu anderen häufig eingesetzten Semantiken.
Für einfache algebraische Spezifikationen und für Termersetzungssysteme wird meistens das initiale Modell bzw. die kanonische Quotientenalgebra als Semantik verwendet. In 6.1 begründen wir, warum diese Quotientenalgebra als Semantik unserer Programme ungeeignet ist. Dennoch zeigen wir immerhin einen Zusammenhang zwischen der Quotientenalgebra und unseren ς-Datentypen auf.
Die insbesondere im Bereich logischer Programmiersprachen eingesetzten deklarativen Semantiken beruhen auf dem Gültigkeits- und Modellbegriff der Logik. In 6.2 stellen wir fest, daß allein unsere cbn-Semantik deklarativ definierbar ist. Immerhin ist die cbv-Semantik für Programme mit Pattern aus deklarativer Sicht ebenfalls ausgezeichnet.
Dies führt uns in 6.3 zur Verwendung partieller Algebren, um damit die cbv-Semantik für Programme mit Pattern deklarativ zu definieren.

6.1 Das initiale Modell

Mit unserer denotationellen ς-Fixpunktsemantik betrachten wir ein Programm als Spezifikation eines Datentyps, einer speziellen Algebra. Diesen Standpunkt haben wir von den algebraischen Spezifikationen übernommen ([Wir90]). Algebraische Spezifikationen bestehen meistens aus Formeln einer Gleichungslogik erster Ordnung. Eine Algebra heißt genau dann Modell der Spezifikation, wenn die Formeln der Spezifikation in dieser Algebra gültig sind. Einfache algebraische Spezifikationen sind schlicht Mengen von Termgleichungen, d. h. Termpaaren. Somit kann der Begriff der Gültigkeit einer Gleichung leicht auf Reduktionsregeln übertragen werden, und wir können die Modelle eines Programms definieren.

Definition 6.1 **Gültigkeit, Modell**
Eine Termersetzungsregel $l \to r$ mit $l, r \in \mathrm{T}_\Sigma(X)$ ist in einer Algebra $\mathfrak{A} = \langle A, \alpha \rangle \in \mathrm{Alg}_\Sigma$ genau dann **gültig**, wenn für alle Variablenbelegungen $\beta : X \to A$

$$[\![l]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}$$

gilt. Eine Algebra $\mathfrak{A} \in \mathrm{Alg}_\Sigma$ heißt genau dann **Modell des Programms P**, wenn alle Reduktionsregeln des Programms P in $\mathfrak{A}$ gültig sind.

Die **Klasse[1] aller Modelle des Programms P** wird mit Mod_P bezeichnet. □

Zueinander isomorphe Modelle werden üblicherweise nicht unterschieden. Man spricht dann von abstrakten algebraischen Datentypen. Trotzdem besitzt eine algebraische Spezifikation im allgemeinen viele (nicht zueinander isomorphe) Modelle, von denen nur eines der durch die Spezifikation bestimmte (abstrakte algebraische) Datentyp sein soll. Bei rein aus Gleichungen bestehenden algebraischen Spezifikationen wird hierfür das initiale Modell, d. h. die initiale Algebra aller Modelle, gewählt. Es ist bekannt, daß das initiale Modell in diesem Fall immer existiert: Es ist die Quotientenalgebra der Grundtermalgebra modulo der Kongruenz der Gleichungen (bzw. ist isomorph zu dieser). Das **initiale Modell eines Programms P** ist somit die **Quotientenalgebra der Grundtermalgebra $\mathcal{T}_\Sigma$ modulo der Kongruenz $\overset{*}{\underset{P}{\leftrightarrow}}$**, $\mathcal{T}_\Sigma / \overset{*}{\underset{P}{\leftrightarrow}}$. Im Kalkül der Termersetzungssysteme wird das initiale Modell bzw. diese Quotientenalgebra ebenso verwendet (Kapitel 3 in [Der&Jou90]).

Warum definieren wir nicht analog zu den algebraischen Spezifikationen den Datentyp eines Programms als dessen initiales Modell?

Hierfür gibt es zwei gute Gründe.

Erstens sollen Programme ausführbare Spezifikationen sein. Da sich mit abstrakten Äquivalenzklassen jedoch nicht effektiv rechnen läßt, sind Berechnungen nicht in beliebigen Quotientenalgebren möglich. Programme algebraischer Spezifikationssprachen, die ausführbar sein sollen, sind daher terminierende, konfluente Termersetzungssysteme. In diesem Fall ist die initiale Algebra isomorph zum sogenannten Normalformmodell. Da ein Programm P bzw. dessen Reduktionsrelation $\underset{P}{\longrightarrow}$ grundsätzlich konfluent ist, müssen wir für die Wohldefiniertheit des Normalformmodells nur noch die Termination fordern.

Definition 6.2 **Normalformmodell eines terminierenden Programms**
Das Programm P, d. h. seine Reduktionsrelation $\underset{P}{\longrightarrow}$, sei terminierend. Das **Normalformmodell des Programms P** ist definiert als die Algebra $\mathcal{M}_P^{\mathrm{nf}} := \langle \mathrm{T}_\Sigma \downarrow_P, \alpha \rangle$, gegeben durch

$$\mathrm{T}_\Sigma \downarrow_P := \{ t\downarrow_P \mid t \in \mathrm{T}_\Sigma \}$$

und

$$g^{\mathcal{M}_P^{\mathrm{nf}}}(\vec{t}) := g(\vec{t})\downarrow_P$$

für alle $g^{(n)} \in \Sigma$, $\vec{t} \in (\mathrm{T}_\Sigma \downarrow_P)^n$. □

Im Normalformmodell ist eine Äquivalenzklasse durch eine Normalform eindeutig repräsentiert. Die Berechnung der Semantik eines Terms erfolgt schlicht durch die Berechnung von dessen Normalform. Es ist zu beachten, daß trotz der großen Ähnlichkeit zu der in Definition 3.7, S. 45, definierten Normalformsemantik das Normalformmodell kein Datentyp dieser Normalformsemantik ist. Der Träger des Normalformmodells enthält kein spezielles Element $\perp$, und es wird nicht zwischen Konstruktornormalformen und anderen Normalformen unterschieden.

Hieraus ergibt sich die zweite unerwünschte Eigenschaft des initialen Modells: Zwar wird kein spezieller Wert $\perp$ für nicht-terminierende Berechnungen benötigt, aber im allgemeinen enthält der Träger des Normalformmodells neben den Konstruktorgrundtermen, die grundsätzlich Normalformen sind, auch noch andere Terme.

[1] Entsprechend unserer Anmerkung in 2.1 ist die Kollektion der Modelle eine Klasse und keine Menge.

Beispiel 6.1 Unerwünschte Datenelemente im Normalformmodell

$$\begin{array}{lcl} \texttt{head(x:xs)} & \rightarrow & \texttt{x} \\ \texttt{tail(x:xs)} & \rightarrow & \texttt{xs} \end{array}$$

ist ein terminierendes Programm mit Pattern mit $\mathcal{C} = \{[]^{(0)}, :^{(2)}\}$ und $\mathcal{F} = \{\texttt{head}^{(1)}, \texttt{tail}^{(1)}\}$.

$$\mathrm{T}_\Sigma\downarrow = \{[], [[]], [[], []], \ldots, \texttt{head}([]), \texttt{tail}([]), \ldots\}$$

ist der Träger des Normalformmodells $\mathcal{M}_P^{\mathrm{nf}}$. □

Dies widerspricht jedoch dem geforderten Prinzip der Konstruktorbasiertheit der Programmiersprache, wonach die Datenelemente durch die Konstruktoren aufgebaut werden, und die Funktions- und Hilfsoperationen nur auf diesen Datenelementen operieren.
Daher ist das Normalformmodell auch nicht **kompositionell** (siehe S. 62); die Erweiterung eines Programms um zusätzliche Funktionssymbole mit zugehörigen Programmregeln führt im allgemeinen zu einer Erweiterung des Trägers und damit auch zu einer Änderung der „alten“ Operationen. Das Konzept des Basisdatentyps garantiert dagegen die Kompositionalität der Semantik. Die Funktionsoperationen (und Hilfsoperationen) operieren auf dem fest gegebenen Träger des Basisdatentyps. In unseren konstruktorbasierten Programmiersprachen sollen die Datenelemente dieses Trägers durch die Konstruktoren aufgebaut werden. In dem Normalformmodell ist jedoch überhaupt kein Basisdatentyp auffindbar. Dieses Problem ist auch im Bereich der algebraischen Spezifikationen bekannt. Als Reaktion wurde in [Gut77] der Begriff der ausreichenden Vollständigkeit (sufficient completeness) von Spezifikationen eingeführt (siehe auch [Gut&Hor78] und 3.2 in [Der&Jou90]). Übertragen auf unsere Programme ist diese wie folgt definiert:

Definition 6.3 Ausreichende Vollständigkeit
Ein Programm P heißt genau dann **ausreichend vollständig**, wenn für jeden Term $t \in \mathrm{T}_\Sigma$ ein Konstruktorterm $c \in \mathrm{T}_\mathcal{C}$ mit $t \xrightarrow[P]{*} c$ existiert[2]. □

Ausreichende Vollständigkeit impliziert selbstverständlich $\mathrm{T}_\Sigma\downarrow = \mathrm{T}_\mathcal{C}$, und garantiert damit die Kompositionalität des Normalformmodells.
Offensichtlich ist für die ausreichende Vollständigkeit eines Programms die Vollständigkeit der Redexschemata in folgendem Sinne notwendig:

Definition 6.4 Vollständigkeit der Redexschemata
Die Menge der Redexschemata, RedS_P, heißt genau dann **vollständig**, wenn für alle $f^{(n)} \in \mathcal{F}(\dot\cup\,\mathcal{H})$ und $\vec{c} \in (\mathrm{T}_\mathcal{C})^n \quad f(\vec{c})$ ein Redex ist. □

Lemma 6.1 Vollständigkeit der Redexschemata ausreichend vollständiger Programme
Wenn P ein ausreichend vollständiges Programm ist, dann ist die Menge der Redexschemata, RedS_P, vollständig.

[2] Die für die ausreichende Vollständigkeit beliebiger Termersetzungssysteme in [Der&Jou90] geforderte Eigenschaft $t \xleftrightarrow[P]{*} c$ ist hier aufgrund der Konfluenz von $\xrightarrow[P]{}$ äquivalent zu $t \xrightarrow[P]{*} c$.

Beweis:
Seien $f^{(n)} \in \mathcal{F}(\dot\cup\,\mathcal{H})$ und $\vec{c} \in (\mathrm{T}_\mathcal{C})^n$ beliebig. Da P ausreichend vollständig ist, existiert $f(\vec{c})\downarrow_P$ und es ist $f(\vec{c})\downarrow_P \in \mathrm{T}_\mathcal{C}$. Weil $f(\vec{c}) \notin \mathrm{T}_\mathcal{C}$, muß ein $t' \in \mathrm{T}_\Sigma$ mit $f(\vec{c}) \xrightarrow[P]{} t'$ existieren. Da das äußerste Symbol $\hat{l}(\varepsilon)$ aller Redexe $\hat{l}$ ein Funktions- oder Hilfssymbol ist, erfolgt die Reduktion an der Stelle ε in $f(\vec{c})$. Somit ist $f(\vec{c})$ ein Redex. □

Allein die Vollständigkeit der Redexschemata garantiert schon, daß alle Normalformen Konstruktorterme sind. Dies gilt sogar für die ς-Reduktion mit einer beliebigen erzwungenen Striktheit ς.

Lemma 6.2 (ς-)Normalformen im Falle vollständiger Redexschemata
Sei RedS_P vollständig. Sei $t \in \mathrm{T}_\Sigma$ ein Term, dessen ς-Normalform $t\downarrow_{P,\varsigma}$ existiert. Dann existiert auch die Normalform $t\downarrow_P$ und es gilt

$$t\downarrow_{P,\varsigma} = t\downarrow_P \in \mathrm{T}_\mathcal{C}.$$

Insbesondere gilt für eine existierende Normalform $t\downarrow_P$

$$t\downarrow_P \in \mathrm{T}_\mathcal{C}.$$

Beweis:
Sei $\hat{t} := t\downarrow_{P,\varsigma}$ für einen Term $t \in \mathrm{T}_\Sigma$.
Angenommen, $\hat{t} \notin \mathrm{T}_\mathcal{C}$. Dann existiert eine Stelle $u \in \mathrm{Occ}(\hat{t})$ mit $\hat{t}(u) = f^{(n)} \in \mathcal{F}(\dot\cup\,\mathcal{H})$ und $c_i := t/u.i \in \mathrm{T}_\mathcal{C}$ für alle $i \in [n]$. Aufgrund der Vollständigkeit der Redexschemata ist $\hat{t}/u = f(\vec{c})$ ein Redex. Da $[\![c_1]\!]^{\mathrm{alg}}_{\perp_\varsigma} = c_1 \neq \perp, \ldots, [\![c_n]\!]^{\mathrm{alg}}_{\perp_\varsigma} = c_n \neq \perp$, ist $\hat{t}/u$ auch ein ς-Redex. Dies steht im Widerspruch zu der Voraussetzung, daß $\hat{t}$ eine ς-Normalform ist.
Also ist $\hat{t} \in \mathrm{T}_\mathcal{C}$ und somit auch $\hat{t} = t\downarrow_P$.
Die Aussage über $t\downarrow_P$ gilt, da $t\downarrow_{P,\mathrm{cbn}} = t\downarrow_P$. □

Natürlich impliziert die Vollständigkeit der Redexschemata alleine noch nicht die ausreichende Vollständigkeit eines Programms.

$$\mathtt{a} \;\rightarrow \mathtt{a}$$

ist ein triviales Gegenbeispiel. Aber es gilt:

Lemma 6.3 Ausreichende Vollständigkeit terminierender Programme mit vollständigen Redexschemata
Das Programm P bzw. seine Reduktionsrelation $\xrightarrow[P]{}$ sei terminierend und die Menge seiner Redexschemata vollständig. Dann ist P ausreichend vollständig.

Beweis:
Sei $t \in \mathrm{T}_\Sigma$ beliebig. Da P terminierend ist, existiert die Normalform $t\downarrow_P$. Nach Lemma 6.2 folgt aus der Vollständigkeit der Redexschemata, daß $t\downarrow_P \in \mathrm{T}_\mathcal{C}$, d. h. $t \xrightarrow[P]{*} t\downarrow_P \in \mathrm{T}_\mathcal{C}$. Also ist P ausreichend vollständig. □

Die Termination des Programms wird sowieso schon für die Wohldefiniertheit des Normalformmodells benötigt. Für die ausreichende Vollständigkeit eines Programms brauchen wir somit nur noch die Vollständigkeit der Redexschemata zu fordern.

Interessant ist übrigens, daß Programme zwar aufgrund ihrer Eindeutigkeitsbedingung in einem gewissen Sinne deterministisch sind — zu jedem Redex existiert nur genau ein Term, durch den der Redex in einem Reduktionsschritt ersetzt werden kann —, aber die naheliegende Vermutung, daß jedes ausreichend vollständige Programm terminiert, ist falsch, wie das folgende Beispiel demonstriert.

Beispiel 6.2 Ausreichende Vollständigkeit $\not\Rightarrow$ Termination

$$\begin{array}{lcl} \mathtt{a} & \to & \mathtt{f(a)} \\ \mathtt{f(x)} & \to & \mathtt{A} \end{array}$$

Alle Terme haben die Normalform `A`. Trotzdem existiert die unendliche Reduktionsfolge

$$\mathtt{a} \xrightarrow[P]{} \mathtt{f(a)} \xrightarrow[P]{} \mathtt{f(f(a))} \xrightarrow[P]{} \mathtt{f(f(f(A)))} \xrightarrow[P]{} \ldots.$$

□

Allerdings können ausreichend vollständige Programme (sogar beliebige erweitert orthogonale Termersetzungssysteme bezüglich derer alle Terme eine Normalform besitzen) keine unendlichen Reduktionsfolgen $t_1 \longrightarrow t_2 \longrightarrow t_3 \longrightarrow \ldots$ mit einem Zyklus besitzen, d. h. es existieren keine $i \neq j \in \mathbb{N}_+$ mit $t_i = t_j$. Da diese Eigenschaft hier jedoch nicht relevant ist, verzichten wir auf den Beweis.
Immerhin demonstriert dies die Subtibilität der ausreichenden Vollständigkeit. In allgemeinen (nicht-terminierenden) Termersetzungssystemen ist diese Eigenschaft unentscheidbar ([Gut&Hor78]).

Betrachten wir nun aufgrund unserer Forderung nach einer berechenbaren, kompositionellen, auf dem Basisdatentyp der Konstruktorterme beruhenden Semantik nur noch terminierende Programme mit vollständigen Redexschemata, so stellen wir fest:

Lemma 6.4 Übereinstimmung aller ς-Semantiken und der initialen Semantik für terminierende, ausreichend vollständige Programme
Sei P ein terminierendes, ausreichend vollständiges Programm. Dann gilt für alle erzwungenen Striktheiten ς und alle Terme $t \in \mathrm{T}_\Sigma$:

$$[\![t]\!]_{P,\varsigma} = t{\downarrow_P}.$$

Beweis:
Aufgund der Termination und Konfluenz der Reduktionsrelation $\xrightarrow[P]{}$ und des Informationsgewinns durch Reduktion gemäß Lemma 5.23, S. 105, gilt

$$\bigsqcup\{[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t'\} = [\![t{\downarrow_{P,\varsigma}}]\!]^{\mathrm{alg}}_{\perp_\varsigma}. \tag{1}$$

Nach Lemma 6.1 ist die Menge der Redexschemata RedS_P vollständig, und somit ist nach Lemma 6.2 über (ς-)Normalformen im Falle vollständiger Redexschemata

$$t{\downarrow_{P,\varsigma}} = t{\downarrow_P} \in \mathrm{T}_{\mathcal{C}}. \tag{2}$$

Insgesamt gilt:

$$[\![t]\!]_{P,\varsigma} = [\![t]\!]^{\mathrm{red}}_{P,\varsigma} = \bigsqcup\{[\![t']\!]^{\mathrm{alg}}_{\perp_\varsigma} \mid t \xrightarrow[P,\varsigma]{*} t'\} \overset{(1)}{=} [\![t{\downarrow_{P,\varsigma}}]\!]^{\mathrm{alg}}_{\perp_\varsigma} \overset{(2)}{=} t{\downarrow_P}.$$

□

Anders formuliert ist

$$\mathcal{T}_\Sigma/\sim_{[\![\cdot]\!]_{P,\varsigma}} = \mathcal{M}_P^{\mathrm{nf}}$$

für alle erzwungenen Striktheiten ς. Aufgrund des Vorhandenseins von $\perp$ und anderen partiellen Konstruktortermen im Träger $\mathrm{T}_{\mathcal{C},\varsigma}$ der ς-Datentypen ist allerdings $\mathcal{D}_{P,\varsigma} \neq \mathcal{M}_P^{\mathrm{nf}}$. Diese partiellen Konstruktorterme sind jedoch unter den gegebenen Bedingungen gar nicht denotierbar.

Bei diesem Ergebnis ist zu bedenken, daß die Forderung der Termination und vollständiger Redexschemata für eine Programmiersprache unzumutbar ist. Unter dieser Bedingung sind partielle Operationen wie head und tail von Listen nicht definierbar[3]. Die Termination ist überhaupt nicht entscheidbar. Sprachen für ausführbare algebraische Spezifikationen wie OBJ (9.2 in [Wir90]) überlassen es dem Programmierer, die Termination und weitere unentscheidbare hinreichende Bedingungen für die Übereinstimmung von operationeller und initialer Semantik sicherzustellen. Dies widerspricht jedoch unserem schon in der Einleitung dargestellten Standpunkt, daß eine Semantik jedem (entscheidbar) syntaktisch korrektem Programm eine Bedeutung zuordnen soll.
Somit ist das initiale Modell als Datentyp eines funktionalen Programms ungeeignet.

6.2 Deklarative Eigenschaften der ς-Semantiken

Semantiken, die auf dem Modellbegriff der Logik basieren, werden deklarativ genannt. Im letzten Abschnitt haben wir die Modelle eines Programms definiert. Offensichtlich ist kein ς-Datentyp das initiale Modell eines Programms. Es wäre aber immerhin zu erwarten, daß die Reduktionsregeln eines Programms in jedem zugehörigen ς-Datentyp gültig wären, dieser also ein Modell des Programms wäre. Wenn ein ς-Datentyp außerdem ein in einer gewissen Weise ausgezeichnetes Modell ist, dann können wir ihn auch auf diese Weise deklarativ definieren.

Der cbn-Datentyp $\mathcal{D}_{P,\mathrm{cbn}}$ ist tatsächlich ein Modell des Programms P. Wir zeigen sogar noch allgemeiner, daß jeder Fixpunkt der cbn-Transformation ein Modell ist:

Lemma 6.5 **cbn-Fixpunkte $\subseteq$ Mod$_P$**
Sei $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\mathrm{cbn}}$ ein Fixpunkt der cbn-Transformation $\Phi_{P,\mathrm{cbn}}$. Dann ist $\mathfrak{A}$ ein Modell des Programms P.

Beweis:

$\mathfrak{A}$ ist ein Fixpunkt von $\Phi_{P,\mathrm{cbn}}$.

$\Longrightarrow$ (Definition)
Für alle Reduktionsregeln $f^{(n)}(\vec{p})\to r \in \hat{P}$, alle $\underline{\vec{t}} \in (\mathrm{T}_{\mathcal{C},\mathrm{cbn}})^n$ und alle $\beta : \mathrm{Var}(\vec{p})\to\mathrm{T}_{\mathcal{C},\mathrm{cbn}}$, so daß $\underline{\vec{t}}$ mit $f(\vec{p})$ vermittels β semantisch cbn-gematcht wird, gilt

$$f^{\mathfrak{A}}(\underline{\vec{t}}) = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

$\Longleftrightarrow$ (Lemma 5.2, S. 79, über die Charakterisierung des semantischen cbn-Matchens)
Für alle $f^{(n)}(\vec{p})\to r \in \hat{P}$, alle $\underline{\vec{t}} \in (\mathrm{T}_{\mathcal{C},\mathrm{cbn}})^n$ und alle $\beta : \mathrm{Var}(\vec{p})\to\mathrm{T}_{\mathcal{C},\mathrm{cbn}}$, so daß für alle $i \in [n]$ $[\![p_i]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}}} = \underline{t}_i$, gilt

$$f^{\mathfrak{A}}(\underline{\vec{t}}) = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

[3] Mit Hilfe sogenannter order-sorted Signaturen und Algebren ist dies dennoch möglich. Andererseits führen diese zu einer größeren Komplexität und neuen Problemen; siehe 3.3.4 in [Wir90].

$\Longleftrightarrow$ Für alle $f^{(n)}(\vec{p}) \to r \in \hat{P}$ und alle $\beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C},\mathrm{cbn}}$ gilt

$$f^{\mathfrak{A}}([\![p_1]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}}}, \ldots, [\![p_n]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}}}) = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

$\Longleftrightarrow$ (da $[\![p_i]\!]^{\mathrm{alg}}_{\perp_{\mathrm{cbn}}} = [\![p_i]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}$ wegen $p_i \in \mathrm{T}_{\mathcal{C}}(X)$)
Für alle $f^{(n)}(\vec{p}) \to r \in \hat{P}$ und alle $\beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C},\mathrm{cbn}}$ gilt

$$[\![f(\vec{p})]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

$\Longleftrightarrow$ $\mathfrak{A}$ ist ein Modell des Programms P.

□

Diese Aussage ist jedoch nicht verallgemeinerbar. Für erzwungene Striktheiten $\varsigma \neq \mathrm{cbn}$ gilt nicht allgemein, daß die Fixpunkte der ς-Transformation Modelle sind; es gilt noch nicht einmal $\mathcal{D}_{P,\varsigma} \in \mathrm{Mod}_P$.

Beispiel 6.3 ς-Fixpunkte $\not\subseteq$ Mod$_P$

```
f(x)  →  A
```

`f` sei erzwungen strikt an seiner einzigen Argumentstelle, d. h. $\varsigma(\mathtt{f}) = (\mathtt{tt})$.
Dann gilt für jeden Fixpunkt $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ von $\Phi_{P,\varsigma}$ $\mathtt{f}^{\mathfrak{A}}(\perp) = \perp$. Somit ist für die Variablenbelegung β mit $\beta(x) = \perp$

$$[\![\mathtt{f(x)}]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = \mathtt{f}^{\mathfrak{A}}(\perp) = \perp \neq \mathtt{A} = [\![\mathtt{A}]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

Also ist $\mathfrak{A}$ kein Modell des Programms. □

Die fehlende Modelleigenschaft der ς-Datentypen für $\varsigma \neq \mathrm{cbn}$ ist enttäuschend, aber auch leicht verständlich. Die erzwungene Striktheit ς ist ein Parameter unserer ς-Semantiken, der völlig unabhängig von einem Programm die Striktheit gewisser Operationen an bestimmten Argumentstellen erzwingt. Dagegen hängt die Gültigkeit und Modelleigenschaft alleine von dem gegebenen Programm ab.

Wir wollen auch die Umkehrung der Aussage des obigen Lemmas überprüfen. Selbstverständlich ist nicht jedes Modell ein Fixpunkt der cbn- oder gar aller ς-Transformationen. Darum betrachten wir nur die Modelle, die auch ς-Interpretationen sind.

Definition 6.5 ς-Interpretationsmodell
Eine ς-Interpretation, welche ein Modell des Programms P ist, heißt **ς-Interpretationsmodell des Programms P**. $\mathrm{IntMod}_{P,\varsigma} := \mathrm{Int}_{\Sigma,\varsigma} \cap \mathrm{Mod}_P$ ist die Menge der ς-Interpretationsmodelle des Programms P. □

Tatsächlich ist jedes cbv-Interpretationsmodell ein Fixpunkt der cbv-Transformation. Auf einen Beweis wollen wir allerdings verzichten. Dagegen sind für beliebige erzwungene Striktheiten ς oder auch nur $\varsigma = \mathrm{cbn}$ ς-Interpretationsmodelle nicht immer Fixpunkte der ς-Transformation:

Beispiel 6.4 **$\mathbf{IntMod}_{P,\varsigma} \not\subseteq$ ς-Fixpunkte**

$$\mathtt{f(A)} \quad \rightarrow \quad \mathtt{A}$$

$\mathtt{f}$ sei nicht erzwungen strikt an seiner einzigen Argumentstelle, d. h. $\varsigma(\mathtt{f}) = (\mathtt{ff})$.
$\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ mit $\mathtt{f}^{\mathfrak{A}}(\bot) := \mathtt{f}^{\mathfrak{A}}(\mathtt{A}) := \mathtt{A}$ ist ein Modell des Programms, aber wegen $\mathtt{f}^{\mathfrak{A}}(\bot) \neq \bot$ kein Fixpunkt von $\Phi_{P,\varsigma}$. □

Der Grund für das Fehlen dieser Eigenschaft liegt darin, daß gemäß der Definition der ς-Transformation in einem Transformationsschritt $f^{\Phi_{P,\varsigma}(\mathfrak{A})}(\underline{\vec{t}}) := \bot$ gesetzt wird, wenn sich $\underline{\vec{t}}$ mit keiner zu f gehörenden linken Reduktionsregelseite semantisch ς-matchen läßt. In Modellen $\mathfrak{A}'$ des Programms können diese Werte $f^{\mathfrak{A}'}(\underline{\vec{t}})$ dagegen beliebig gesetzt sein, zumindestens solange solche Symbole f nicht auf irgendwelchen rechten Reduktionsregelseiten auftauchen.
In der in Abschnitt 5.2.5 betrachteten alternativen Transformation $\Phi^*_{P,\varsigma}$ erfolgt dieses „Setzen auf $\bot$“ nicht. Daher ist tatsächlich jede ς-Interpretation*, die ein Modell ist, ein Fixpunkt von $\Phi^*_{P,\varsigma}$. Da diese Eigenschaft jedoch von keinem großen Interesse ist, verfolgen wir sie nicht weiter.

In einem anderen Zusammenhang ist der Begriff des ς-Interpretationsmodells sehr nützlich. Die ς-Interpretationen verkörpern das Konzept des Basisdatentyps. Die ς-Interpretationen sind jene Algebren, die unabhängig von einem konkreten Programm als ς-Datentypen in Frage kommen (wir vernachlässigen hierbei die Hilfsoperationen). Aus Lemma 6.5 wissen wir, daß der cbn-Datentyp ein Modell ist. Somit ist der cbn-Datentyp ein cbn-Interpretationsmodell. Es zeigt sich, daß wir den cbn-Datentyp mit Hilfe der cbn-Interpretationsmodelle sogar sehr schön charakterisieren können: Der cbn-Datentyp $\mathcal{D}_{P,\mathrm{cbn}}$ ist zwar nicht die initiale, aber die kleinste Algebra der cbn-Interpretationsmodelle $\mathrm{IntMod}_{P,\mathrm{cbn}}$.
Für den Beweis, der übrigens ähnlich dem des Fixpunktsatzes von Tarski ist, benötigen wir das folgende Hilfslemma.

Lemma 6.6 **Über cbn-Interpretationsmodelle**
Sei $\mathfrak{A}$ ein cbn-Interpretationsmodell des Programms P. Dann gilt

$$\Phi_{P,\mathrm{cbn}}(\mathfrak{A}) \sqsubseteq \mathfrak{A}.$$

Beweis:
Gemäß der Definition der cbn-Interpretationen ist für alle Konstruktorsymbole $G \in \mathcal{C}$

$$G^{\Phi_{P,\mathrm{cbn}}(\mathfrak{A})} = G^{\mathfrak{A}}.$$

Seien $f^{(n)} \in \mathcal{F}(\,\dot\cup\,\mathcal{H})$ und $\underline{\vec{t}} \in (\mathrm{T}_{\mathcal{C},\mathrm{cbn}})^n$ beliebig.

Fall 1: $\underline{\vec{t}}$ wird mit der linken Seite einer Reduktionsregel $f(\vec{p})\rightarrow r$ vermittels einer Variablenbelegung $\beta : \mathrm{Var}(\vec{p})\rightarrow \mathrm{T}_{\mathcal{C},\mathrm{cbn}}$ semantisch cbn-gematcht.

Dann ist

$$f^{\Phi_{P,\mathrm{cbn}}(\mathfrak{A})}(\vec{t}) = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

Da $\mathfrak{A}$ ein Modell des Programms P ist, gilt auch

$$f^{\mathfrak{A}}(\vec{t}) = [\![f(\vec{p})]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

Somit folgt

$$f^{\Phi_{P,\mathrm{cbn}}(\mathfrak{A})}(\vec{t}) = f^{\mathfrak{A}}(\vec{t}).$$

Fall 2: Andernfalls.

$$f^{\Phi_{P,\mathrm{cbn}}(\mathfrak{A})}(\vec{t}) = \bot \trianglelefteq f^{\mathfrak{A}}(\vec{t}).$$

□

Lemma 6.7 $\mathcal{D}_{P,\mathbf{cbn}} = \mathbf{Min}_{\langle \mathbf{Alg}^{\infty}_{\Sigma,\bot},\sqsubseteq\rangle}(\mathbf{IntMod}_{P,\mathbf{cbn}})$
Der cbn-Datentyp, also der kleinste Fixpunkt der cbn-Transformation, ist das kleinste cbn-Interpretationsmodell des Programms P.

Beweis:
Nach Lemma 6.5 ist jeder Fixpunkt von $\Phi_{P,\mathrm{cbn}}$ ein cbn-Interpretationsmodell von P. Es bleibt zu zeigen, daß $\mathcal{D}_{P,\mathrm{cbn}} = \bigsqcup_{n\in\mathbb{N}}(\Phi_{P,\mathrm{cbn}})^n(\bot_{\mathrm{cbn}})$ das kleinste aller cbn-Interpretationsmodelle von P ist. Sei $\mathfrak{A} \in \mathrm{IntMod}_{P,\mathrm{cbn}}$. Durch vollständige Induktion wird gezeigt, daß für alle $n \in \mathbb{N}$

$$(\Phi_{P,\mathrm{cbn}})^n(\bot_{\mathrm{cbn}}) \sqsubseteq \mathfrak{A}.$$

$n = 0$: $\bot_{\mathrm{cbn}} \sqsubseteq \mathfrak{A}$.

$n \Rightarrow n+1$: Nach Induktionsvoraussetzung ist

$$(\Phi_{P,\mathrm{cbn}})^n(\bot_{\mathrm{cbn}}) \sqsubseteq \mathfrak{A}.$$

Da die cbn-Transformation gemäß Lemma 5.20, S. 95, monoton ist, gilt

$$(\Phi_{P,\mathrm{cbn}})^{n+1}(\bot_{\mathrm{cbn}}) \sqsubseteq \Phi_{P,\mathrm{cbn}}(\mathfrak{A}).$$

Nach Lemma 6.6 über cbn-Interpretationsmodelle ist

$$\Phi_{P,\mathrm{cbn}}(\mathfrak{A}) \sqsubseteq \mathfrak{A}.$$

Aus den letzten 2 Ungleichungen folgt wie gewünscht

$$(\Phi_{P,\mathrm{cbn}})^{n+1}(\bot_{\mathrm{cbn}}) \sqsubseteq \mathfrak{A}.$$

Da $\mathcal{D}_{P,\mathrm{cbn}}$ die kleinste obere Schranke der ω-Kette $((\Phi_{P,\mathrm{cbn}})^n(\bot_{\mathrm{cbn}}))_{n\in\mathbb{N}}$ ist, folgt

$$\mathcal{D}_{P,\mathrm{cbn}} \sqsubseteq \mathfrak{A}.$$

□

Diese deklarative Charakterisierung ist eine ebenso gute Definition des cbn-Datentyps wie die cbn-Fixpunktsemantik. Dennoch ist die cbn-Fixpunktsemantik nicht nur aufgrund ihrer anderen Sichtweise und der größeren Allgemeinheit (beliebige erzwungene Striktheiten ς) wertvoll. Die Existenz eines kleinsten cbn-Interpretationsmodells und somit des cbn-Datentyps ist nämlich durch die deklarative Definition noch nicht garantiert. Ein direkter Beweis der Existenz ist zwar möglich, aber auch nicht einfacher als die vielen für die Wohldefiniertheit der ς-Fixpunktsemantik benötigten (hauptsächlich ω-Stetigkeits-) Beweise. Mit Hilfe des konstruktiven Fixpunktsatzes von Tarski haben wir die Existenz des cbn-Datentyps in der ς-Fixpunktsemantik sichergestellt.

Wir besitzen nun eine deklarative cbn-Semantik. Für die übrigen ς-Semantiken scheint jedoch keine deklarative Definition zu existieren, da, wie wir gezeigt haben, aufgrund der erzwungenen Striktheit Fixpunkte einer ς-Transformation und insbesondere der ς-Datentyp nicht immer Modelle sind. Die fehlende Modelleigenschaft ergibt sich aus der Quantifizierung der in den Reduktionsregeln vorkommenden Variablen über alle Elemente des Trägers der Algebra. Verzichtet man bei der Quantifizierung auf $\bot$, so sind alle Fixpunkte aller ς-Transformationen derartige „Modelle*“:

Definition 6.6 Gültigkeit*, ς-Interpretationsmodell*
Eine Termersetzungsregel $l \to r$ mit $l, r \in \mathrm{T}_\Sigma(X)$ ist in einer ς-Interpretation $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ genau dann **gültig***, wenn für alle Variablenbelegungen $\beta : X \to \mathrm{T}_{\mathcal{C},\varsigma} \setminus \{\bot\}$

$$[\![l]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}$$

gilt. Eine ς-Interpretation $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ heißt genau dann **ς-Interpretationsmodell* des Programms *P***, wenn alle Reduktionsregeln des Programms P in $\mathfrak{A}$ gültig* sind.
Die **Menge aller ς-Interpretationsmodelle** wird mit $\mathrm{IntMod}^*_{P,\varsigma}$ bezeichnet. □

Lemma 6.8 Charakterisierung des semantischen ς-Matchens
Sei $f^{(n)}(\vec{p}) \in \mathrm{RedS}_P$ und $\beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C},\varsigma} \setminus \{\bot\}$. $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ ist genau dann mit $f(\vec{p})$ vermittels β semantisch ς-matchbar, wenn für alle $i \in [n]$

$$[\![p_i]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta} = \underline{t}_i.$$

Beweis:

$\Rightarrow$: Folgt direkt aus der Definition 5.5, S. 76, des semantischen ς-Matchens.

$\Leftarrow$: Nach Lemma 5.5, S. 80, über Variablenbelegungen ohne $\bot$ gilt für alle $i \in [n]$

$$p_i[\bot/\mathrm{Var}(p_i)] \lhd [\![p_i]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta},$$

d. h.

$$p_i[\bot/\mathrm{Var}(p_i)] \lhd \underline{t}_i.$$

Somit ist f auch nicht erwungen strikt für $\vec{\underline{t}}$, und nach Definition 5.5, S. 76, des semantischen ς-Matchens ist $\vec{\underline{t}}$ also mit $f(\vec{p})$ vermittels β semantisch ς-matchbar.

□

Lemma 6.9 ς-Fixpunkte $\subseteq$ IntMod$^*_{P,\varsigma}$
Sei $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ ein Fixpunkt der ς-Transformation $\Phi_{P,\varsigma}$. Dann ist $\mathfrak{A}$ ein ς-Interpretationsmodell des Programms P.

Beweis:

$\mathfrak{A}$ ist ein Fixpunkt von $\Phi_{P,\varsigma}$.

$\Longrightarrow$ (Definition)

Für alle Reduktionsregeln $f^{(n)}(\vec{p}) \to r \in \hat{P}$, alle $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ und alle $\beta : \mathrm{Var}(\vec{p}) \to \mathrm{T}_{\mathcal{C},\varsigma}$, so daß $\vec{\underline{t}}$ mit $f(\vec{p})$ vermittels β semantisch ς-gematcht wird, gilt

$$f^{\mathfrak{A}}(\vec{\underline{t}}) = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

$\Longrightarrow$ Für alle $f^{(n)}(\vec{p})\to r \in \hat{P}$, alle $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ und alle $\beta : \mathrm{Var}(\vec{p})\to\mathrm{T}_{\mathcal{C},\varsigma}\underline{\setminus\{\bot\}}$, so daß $\vec{\underline{t}}$ mit $f(\vec{p})$ vermittels β semantisch ς-gematcht wird, gilt

$$f^{\mathfrak{A}}(\vec{\underline{t}}) = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

$\Longleftrightarrow$ (Lemma 6.8 über die Charakterisierung des semantischen ς-Matchens)
Für alle $f^{(n)}(\vec{p})\to r \in \hat{P}$, alle $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ und alle $\beta : \mathrm{Var}(\vec{p})\to\mathrm{T}_{\mathcal{C},\varsigma} \setminus \{\bot\}$, so daß für alle $i \in [n]$ $[\![p_i]\!]^{\mathrm{alg}}_{\bot_{\mathrm{cbn}}} = \underline{t}_i$, gilt

$$f^{\mathfrak{A}}(\vec{\underline{t}}) = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

$\Longleftrightarrow$ Für alle $f^{(n)}(\vec{p})\to r \in \hat{P}$ und alle $\beta : \mathrm{Var}(\vec{p})\to\mathrm{T}_{\mathcal{C},\varsigma} \setminus \{\bot\}$ gilt

$$f^{\mathfrak{A}}([\![p_1]\!]^{\mathrm{alg}}_{\bot_\varsigma}, \ldots, [\![p_n]\!]^{\mathrm{alg}}_{\bot_\varsigma}) = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

$\Longleftrightarrow$ (Weil $[\![p_i]\!]^{\mathrm{alg}}_{\bot_\varsigma} = [\![p_i]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}$ wegen $p_i \in \mathrm{T}_{\mathcal{C}}(X)$, und weil nach Lemma 5.5, S. 80, über Variablenbelegungen ohne $\bot$ $[\![p_i]\!]^{\mathrm{alg}}_{\bot_\varsigma} \neq \bot$)
Für alle $f^{(n)}(\vec{p})\to r \in \hat{P}$ und alle $\beta : \mathrm{Var}(\vec{p})\to\mathrm{T}_{\mathcal{C},\varsigma} \setminus \{\bot\}$ gilt

$$[\![f(\vec{p})]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

$\Longleftrightarrow$ $\mathfrak{A}$ ist ein ς-Interpretationsmodell* des Programms P.

□

Die Umkehrung, $\mathrm{IntMod}^*_{P,\varsigma} \subseteq$ ς-Fixpunkte, gilt ebenso wie bei den „normalen“ ς-Interpretationsmodellen <u>nicht</u> (siehe Beispiel 6.4).
Wenn wir jetzt untersuchen, ob analog zum cbn-Datentyp jeder ς-Datentyp $\mathcal{D}_{P,\varsigma}$ das kleinste ς-Interpretationsmodell* $\mathrm{IntMod}^*_{P,\varsigma}$ ist, so stellen wir fest, daß dies nicht der Fall ist.

<u>Beispiel 6.5</u> $\boldsymbol{\mathcal{D}_{P,\varsigma} \neq \mathbf{Min}_{\langle \mathbf{Alg}_\Sigma,\sqsubseteq\rangle}(\mathbf{IntMod}^*_{P,\varsigma})}$

```
f(x)  →  A
```

f sei nicht erzwungen strikt an seiner einzigen Argumentstelle, d. h. ς(f) = (ff).
Dann ist $\mathtt{f}^{\mathcal{D}_{P,\varsigma}}(\bot) = \mathtt{f}^{\mathcal{D}_{P,\varsigma}}(\mathtt{A}) = \mathtt{A}$. Jedoch ist auch $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ mit $\mathtt{f}^{\mathfrak{A}}(\bot) = \bot$ und $\mathtt{f}^{\mathfrak{A}}(\mathtt{A}) = \mathtt{A}$ ein ς-Interpretationsmodell* des Programms, und es ist $\mathfrak{A} \sqsubset \mathcal{D}_{P,\varsigma}$. □

Die Ursache des Fehlens dieser Eigenschaft sind die Operationen des ς-Datentyps, welche nicht-strikt sind. Daher tritt dieses Problem auch in der cbv-Semantik bei Programmen mit Hilfsfunktionen auf:

<u>Beispiel 6.6</u> $\boldsymbol{\mathcal{D}_{P,\mathbf{cbv}} \neq \mathbf{Min}_{\langle \mathbf{Alg}_\Sigma,\sqsubseteq\rangle}(\mathbf{IntMod}^*_{P,\mathbf{cbv}})}$
Sei P ein Programm mit Hilfsfunktionen mit $A \in \mathcal{C}_0$. Dann ist nach Lemma 5.1, S. 78, über die Hilfssymboloperationen

$$\mathrm{cond}^{\mathcal{D}_{P,\mathrm{cbv}}}_{\mathtt{A}}(\mathtt{A},\mathtt{A},\bot) = \mathtt{A}.$$

Wir definieren nun $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\mathrm{cbv}}$ durch

$$\begin{array}{rcl} \mathrm{cond}^{\mathfrak{A}}_{\mathtt{A}}(\mathtt{A},\mathtt{A},\bot) & := & \bot \\ f^{\mathfrak{A}}(\vec{\underline{t}}) & := & f^{\mathcal{D}_{P,\mathrm{cbv}}}(\vec{\underline{t}}) \quad \text{für alle sonstigen } \vec{\underline{t}} \in (\mathrm{T}^{\bot}_{\mathcal{C}})^n, f^{(n)} \in \mathcal{F}(\,\dot\cup\,\mathcal{H}) \end{array}$$

Da $\mathcal{D}_{P,\mathrm{cbv}}$ ein cbv-Interpretationsmodell* ist, ist auch $\mathfrak{A}$ ein solches. $\mathcal{D}_{P,\mathrm{cbv}}$ ist nicht das kleinste cbv-Interpretationsmodell*, weil $\mathfrak{A} \sqsubset \mathcal{D}_{P,\mathrm{cbv}}$. □

Der cbv-Datentyp $\mathcal{D}_{P,\mathrm{cbv}}$ eines Programms mit Pattern ist jedoch tatsächlich das kleinste cbv-Interpretationsmodell* des Programms.
Analog zu Lemma 6.6 über cbn-Interpretationsmodelle beweisen wir erst das folgende Hilfslemma.

Lemma 6.10 Über cbv-Interpretationsmodelle* von Programmen mit Pattern
Sei P ein Programm mit Pattern. Sei $\mathfrak{A}$ ein cbv-Interpretationsmodell* des Programms P. Dann gilt

$$\Phi_{P,\mathrm{cbv}}(\mathfrak{A}) \sqsubseteq \mathfrak{A}.$$

Beweis:
Gemäß der Definition der cbv-Interpretationen ist für alle Konstruktorsymbole $G \in \mathcal{C}$

$$G^{\Phi_{P,\mathrm{cbv}}(\mathfrak{A})} = G^{\mathfrak{A}}.$$

Seien $f^{(n)} \in \mathcal{F}(\dot\cup\, \mathcal{H})$ und $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\mathrm{cbv}})^n$ beliebig.

Fall 1: $\vec{\underline{t}}$ wird mit der linken Seite einer Reduktionsregel $f(\vec{p})\to r$ vermittels einer Variablenbelegung $\beta : \mathrm{Var}(\vec{p})\to \mathrm{T}_{\mathcal{C},\mathrm{cbv}}$ semantisch cbv-gematcht.

Dann ist

$$f^{\Phi_{P,\mathrm{cbv}}(\mathfrak{A})}(\vec{t}) = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

Da P ein Programm mit Pattern ist, ist f kein Verzweigungssymbol und nach Lemma 5.4, S. 80, über semantische cbv-Matchbarkeit ist $\beta(x) \neq \bot$ für alle $x \in \mathrm{Var}(\vec{p})$. Da $\mathfrak{A}$ ein cbv-Interpretationsmodell* des Programms P ist, folgt somit

$$f^{\mathfrak{A}}(\vec{t}) = [\![f(\vec{p})]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}.$$

Insgesamt gilt also

$$f^{\Phi_{P,\mathrm{cbv}}(\mathfrak{A})}(\vec{t}) = f^{\mathfrak{A}}(\vec{t}).$$

Fall 2: Sonst.

$$f^{\Phi_{P,\mathrm{cbv}}(\mathfrak{A})}(\vec{t}) = \bot \trianglelefteq f^{\mathfrak{A}}(\vec{t}).$$

□

Lemma 6.11 $\mathcal{D}_{P,\mathbf{cbv}} = \mathbf{Min}_{\langle \mathbf{Alg}_{\Sigma},\sqsubseteq\rangle}(\mathbf{IntMod}^{*}_{P,\mathbf{cbv}})$ für ein Programm mit Pattern
Sei P ein Programm mit Pattern. Der cbv-Datentyp, also der kleinste Fixpunkt der cbv-Transformation, ist das kleinste cbv-Interpretationsmodell* des Programms P.

Beweis:
Dies folgt unter Verwendung des Lemmas 6.10 über cbv-Interpretationsmodelle* von Programmen mit Pattern völlig analog zum Beweis des Lemmas 6.7, wonach der cbn-Datentyp das kleinste cbn-Interpretationsmodell des Programms ist. □

Im wesentlichen haben wir in diesem Abschnitt gezeigt, daß Fixpunkte der cbn-Transformation Modelle sind, und daß die cbn-Semantik als einzige deklarativ definierbar ist, nämlich durch $\mathcal{D}_{P,\mathrm{cbn}} = \mathrm{Min}_{\langle \mathrm{Alg}^{\infty}_{\Sigma,\perp},\sqsubseteq\rangle}(\mathrm{IntMod}_{P,\mathrm{cbn}})$. Außerdem ist für Programme mit Pattern der cbv-Datentyp auch durch $\mathcal{D}_{P,\mathrm{cbv}} = \mathrm{Min}_{\langle \mathrm{Alg}_{\Sigma},\sqsubseteq\rangle}(\mathrm{IntMod}^{*}_{P,\mathrm{cbv}})$ charakterisierbar. Die Definitionen der Gültigkeit* und des ς-Interpretationsmodells* erscheinen jedoch trotz der Ausgezeichnetheit von $\perp$ als ziemlich willkürlich und ausgefallen. Daher kann $\mathcal{D}_{P,\mathrm{cbv}} = \mathrm{Min}_{\langle \mathrm{Alg}_{\Sigma},\sqsubseteq\rangle}(\mathrm{IntMod}^{*}_{P,\mathrm{cbv}})$ kaum als deklarative Definition der cbv-Semantik verstanden werden. Diese Charakterisierung des cbv-Datentyps zusammen mit der Tatsache, daß der Rechenbereich der cbv-Semantik eine einfache flache Halbordnung ist ($\langle \mathrm{T}^{\perp}_{\mathcal{C}}, \trianglelefteq\rangle$), und die Quantifizierung der Variablen der Reduktionsregeln beim Gültigkeits*-Test genau über $\mathrm{T}_{\mathcal{C}} = \mathrm{T}^{\perp}_{\mathcal{C}} \setminus \{\perp\}$ erfolgt, führt jedoch zu der Idee, für die cbv-Semantik von Programmen mit Pattern anstelle der geordneten Algebren mit totalen Operationen sogenannte partielle Algebren zu verwenden.

6.3 Der partielle cbv-Datentyp von Programmen mit Pattern

Wir haben von Beginn an das spezielle Symbol $\perp$ für Undefiniertheit (oder fehlende Information) verwendet, obwohl der Einsatz partieller Abbildungen eigentlich naheliegender und natürlicher ist. Allerdings ist das Ergebnis einer partiellen Abbildung grundsätzlich undefiniert, wenn nur eines der Argumente undefiniert ist; d. h. partielle Abbildungen sind gemäß unserer für Abbildungen über Halbordnungen definierten Bezeichnungsweise prinzipiell strikt. Daher sind Semantiken, die den Operationssymbolen partielle Operationen zuordnen, nur für die cbv-Semantik geeignet. Außerdem sind die Programme mit Hilfsfunktionen ausgeschlossen, weil die Operationen der Verzweigungssymbole cond_G an den letzten zwei Argumentstellen prinzipiell nicht-strikt sind.
Im diesem gesamten Abschnitt betrachten wir also nur Programme mit Pattern.

Auf partiellen Algebren beruhende Semantiken algebraischer Spezifikationen werden in [Broy&Wir82] und in 3.3.2 in [Wir90] betrachtet. Eine umfassende Darstellung der mathematischen Theorie partieller Algebren ist [Bur86].
Diese Theorie erweist sich leider aufgrund der nötigen Unterscheidung von definierten und undefinierten Ausdrücken als komplizierter als die allgemein bekannte und bisher verwendete der totalen Algebren. Die meisten bekannten Begriffe lassen sich zwar leicht auf partielle Algebren übertragen, erscheinen dort jedoch häufig in mehreren verschiedenen Varianten. So werden beispielsweise in der Literatur drei verschiedene Arten von Homomorphismen und dem entsprechend auch drei verschiedene Arten von initialen partiellen Algebren aufgeführt.
Wir führen jedoch nur wenige, für den partiellen cbv-Datentyp wirklich benötigte Begriffe ein.

Definition 6.7 **Partielle Operation** (vgl. mit Def. der Operation in 2.3)

Ist A eine Menge und $n \in \mathbb{N}$, so heißt eine partielle Abbildung $f : A^n \nrightarrow A$ n-stellige **partielle Operation** auf A. Wir verwenden die Bezeichnungen:

$$\begin{aligned} \mathrm{Ops}^{\mathrm{part}}_n(A) &:= \{f \mid f : A^n \nrightarrow A\} \\ \mathrm{Ops}^{\mathrm{part}}(A) &:= \bigcup_{n\in\mathbb{N}} \mathrm{Ops}^{\mathrm{part}}_n(A) \end{aligned}$$

□

Definition 6.8 **Partielle Algebra** (vgl. mit Def. der Algebra in 2.3)

Sei Σ eine beliebige Signatur. Eine **partielle (Σ-)Algebra** $\mathfrak{A} = \langle A, \alpha \rangle$ besteht aus einer nicht-leeren Menge A, dem **Träger**, und einer **Zuweisung** $\alpha : \Sigma \rightarrow \mathrm{Ops}^{\mathrm{part}}(A)$, die jedem n-stelligen Operationssymbol $f \in \Sigma_n$ eine n-stellige partielle Operation $\alpha(f) \in \mathrm{Ops}^{\mathrm{part}}_n(A)$ zuweist. Statt $\alpha(f)$ schreiben wir meistens $f^{\mathfrak{A}}$. $\mathrm{Alg}^{\mathrm{part}}_\Sigma$ bezeichnet die **Klasse aller partiellen Σ-Algebren**. □

Wir übertragen einige der bisher für totale Algebren definierten Begriffe auf partielle Algebren.

Definition 6.9 **Kanonische Halbordnung partieller Algebren**
(vgl. mit Def. der kanonischen Halbordnung von Algebren in 2.3)
Sei Σ eine beliebige Signatur. Sei A eine nicht-leere Menge.

$$\mathrm{Alg}^{\mathrm{part}}_\Sigma(A) := \{\mathfrak{A} \in \mathrm{Alg}^{\mathrm{part}}_\Sigma \mid \mathfrak{A} = \langle A, \alpha \rangle,\ \alpha \text{ beliebig}\}$$

ist die **Menge aller partiellen Σ-Algebren des Trägers *A***.
Die **kanonische Halbordnung partieller Σ-Algebren des Trägers *A***,

$$\langle \mathrm{Alg}^{\mathrm{part}}_\Sigma(A), \sqsubseteq \rangle,$$

ist definiert durch

$$\mathfrak{A} \sqsubseteq \mathfrak{B} \iff \forall g \in \Sigma.\ g^{\mathfrak{A}} \preceq g^{\mathfrak{B}}$$

für alle $\mathfrak{A}, \mathfrak{B} \in \mathrm{Alg}^{\mathrm{part}}_\Sigma(A)$. □

Definition 6.10 **Algebraische Termsemantik bezüglich einer partiellen Algebra**
(vgl. mit Def. 3.10, S. 47, der algebraischen Termsemantik)
Sei $\mathfrak{A} = \langle A, \alpha \rangle$ eine partielle Σ-Algebra, $Y \subseteq X$ und $\beta : Y \rightarrow A$ eine Variablenbelegung. Die **algebraische Termsemantik bezüglich der partiellen Algebra** $\mathfrak{A}$

$$[\![\cdot]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} : \mathrm{T}_\Sigma(Y) \rightharpoonup A$$

ist für einen Term $t \in \mathrm{T}_\Sigma(Y)$ induktiv definiert durch

$$[\![x]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} := \beta(x)$$

$$[\![g(\hat{t}_1, \ldots, \hat{t}_n)]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} := g^{\mathfrak{A}}([\![\hat{t}_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}, \ldots, [\![\hat{t}_n]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}) \text{ , falls } [\![\hat{t}_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}, \ldots, [\![\hat{t}_n]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \text{ definiert sind.}$$

$$\text{Andernfalls ist } [\![g(\hat{t}_1, \ldots, \hat{t}_n)]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \text{ undefiniert.}$$

für alle $x \in Y$ und $g^{(n)} \in \Sigma$.
Bei Termsemantiken von Grundtermen wird β meist weggelassen. □

Da wir keinen Homomorphismus für partielle Algebren definiert haben und es deren wie erwähnt drei verschiedene gibt, können wir die algebraische Termsemantik bezüglich partieller Algebren auch nicht als den eindeutig bestimmten Einsetzungshomomorphismus bezeichnen.
Es ist zu beachten, daß die algebraische Termsemantik bezüglich einer partiellen Algebra eine partielle Abbildung ist. Eine Variablenbelegung ist jedoch auch hier immer total.

Definition 6.11 Gültigkeit, Partielles Modell
(vgl. mit Def. 6.1, S. 133, und auch mit Def. 6.6, S. 142)
Eine Termersetzungsregel $l \to r$ mit $l, r \in \mathrm{T}_\Sigma(X)$ ist in einer partiellen Algebra $\mathfrak{A} = \langle A, \alpha \rangle \in \mathrm{Alg}_\Sigma^{\mathrm{part}}$ genau dann **gültig**, wenn für alle Variablenbelegungen $\beta : X \to A$

$$\begin{array}{rl} \text{entweder} & [\![l]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \text{ und } [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \quad \text{undefiniert sind} \\ \text{oder} & [\![l]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \text{ gilt.} \end{array}$$

Eine Algebra $\mathfrak{A} \in \mathrm{Alg}_\Sigma^{\mathrm{part}}$ heißt genau dann **partielles Modell des Programms P**, wenn alle Reduktionsregeln des Programms P in $\mathfrak{A}$ gültig sind.
Die **Klasse aller partiellen Modelle des Programms P** wird mit $\mathrm{Mod}_P^{\mathrm{part}}$ bezeichnet. □

Die für den Begriff der Gültigkeit einer Termersetzungsregel definierte Art von Gleichheit heißt **starke Gleichheit** (strong equality). In der Literatur findet man noch eine zweite, andere Interpretation der Gleichheit, die **existenzielle Gleichheit** (existential equality). Hierbei heißt eine Termersetzungsregel $l \to r$ genau dann gültig in einer partiellen Algebra $\mathfrak{A} = \langle A, \alpha \rangle \in \mathrm{Alg}_\Sigma^{\mathrm{part}}$, wenn für alle Variablenbelegungen $\beta : X \to A$ $[\![l]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}$ und $[\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}$ definiert sind und $[\![l]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}$ ist.
Unsere Wahl der starken Gleichheit läßt sich sehr einfach anhand des schlichten Programms

$$\mathtt{a} \quad \to \quad \mathtt{a}$$

begründen. Bei Verwendung der existenziellen Gleichheit müßte $\mathtt{a}^{\mathfrak{A}}$ in jedem Modell $\mathfrak{A}$ des Programms definiert sein. Wir wollen jedoch, daß in dem im weiteren definierten partiellen cbv-Datentyp die Operation von `a` undefiniert ist, da das Programm keine Information über den semantischen Wert von `a` liefert. Bei existenzieller Gleichheit würde schon das Erscheinen eines Funktionssymbols auf einer rechten Programmregelseite zur Definiertheit seiner Operation führen. Ein kleinstes partielles cbv-Interpretationsmodell — wie wir es im weiteren definieren — würde bei existenzieller Gleichheit im allgemeinen nicht existieren.

Spätestens jetzt ist auch eine Bemerkung zur Bedeutung der Gleichheit in unserer Metasprache angebracht. Wir vergleichen in diesem Abschnitt häufig die Ergebnisse partieller Abbildungen. $a = b$ soll implizieren, daß a und b definierte Ausdrücke sind. Wir verwenden also in der Metasprache die existenzielle Gleichheit. Die definierende Gleichheit $a := b$ wird nur verwendet, wenn b definiert ist, und impliziert selbstverständlich, daß auch a definiert ist. Wir geben jedoch die Undefiniertheit von Ausdrücken auch oft explizit an.

Definition 6.12 Partielle cbv-Interpretation
(vgl. mit Def. 4.3, S. 55, der cbv-Interpretation)
Sei $\mathfrak{A} \in \mathrm{Alg}_\Sigma^{\mathrm{part}}(\mathrm{T}_\mathcal{C})$ eine partielle Algebra mit dem Träger $\mathrm{T}_\mathcal{C}$. Sei

$$G^{\mathfrak{A}}(\underline{\vec{t}}) := G(\underline{\vec{t}})$$

für alle $G^{(n)} \in \mathcal{C}$, $\underline{\vec{t}} \in (\mathrm{T}_\mathcal{C})^n$. Dann heißt $\mathfrak{A}$ **partielle cbv-Interpretation** Die Menge aller cbv-Interpretationen bezeichnen wir mit $\mathrm{Int}^{\mathrm{part}}_{\Sigma,\mathrm{cbv}}$.
$\langle \mathrm{Int}^{\mathrm{part}}_{\Sigma,\mathrm{cbv}}, \sqsubseteq \rangle$ ist die kanonische Halbordnung aller partiellen cbv-Interpretationen entsprechend Definition 6.9. □

Definition 6.13 Partielles cbv-Interpretationsmodell
(vgl. mit Def. 6.5 des ς-Interpretationsmodells und Def. 6.6 des cbv-Interpretationsmodells*)
$\mathfrak{A}$ heißt genau dann **partielles cbv-Interpretationsmodell eines Programms mit Pattern, P**, wenn $\mathfrak{A}$ eine partielle cbv-Interpretation und ein partielles Modell des Programms mit Pattern, P, ist. Wir schreiben $\mathrm{IntMod}^{\mathrm{part}}_{P,\mathrm{cbv}} := \mathrm{Int}^{\mathrm{part}}_{\Sigma,\mathrm{cbv}} \cap \mathrm{Mod}^{\mathrm{part}}_{P}$. □

Inspiriert durch die in 6.2 gefundene Charakterisierung des (totalen) cbv-Datentyps, $\mathcal{D}_{P,\mathrm{cbv}} := \mathrm{Min}_{\langle \mathrm{Alg}_\Sigma, \sqsubseteq\rangle}(\mathrm{IntMod}^{*}_{P,\mathrm{cbv}})$, definieren wir den partiellen cbv-Datentyp.

Definition 6.14 Partieller cbv-Datentyp
Das bezüglich der kanonischen Ordnung kleinste partielle cbv-Interpretationsmodell von P,

$$\mathcal{D}^{\mathrm{part}}_{P,\mathrm{cbv}} := \mathrm{Min}_{\langle \mathrm{Alg}^{\mathrm{part}}_\Sigma, \sqsubseteq\rangle}(\mathrm{IntMod}^{\mathrm{part}}_{P,\mathrm{cbv}}),$$

heißt **partieller cbv-Datentyp des Programms mit Pattern, P**. □

Diese Definition läßt sich zwar leicht niederschreiben, aber es bleibt die Wohldefiniertheit des partiellen cbv-Datentyps zu zeigen. Wie bei der deklarativen Definition des cbn-Datentyps müssen wir beweisen, daß ein kleinstes partielles cbv-Interpretationsmodell überhaupt existiert. Ein direkter Beweis wäre jedoch aufwendig und würde eine weitergehende Beschäftigung mit der Theorie partieller Algebren erfordern.
Wir untersuchen daher stattdessen zuerst einen anderen Punkt: Jede der gerade angegebenen Definitionen ist zwar ihrer jeweils korrespondierenden Definition für totale Algebren ähnlich, aber es stellt sich doch die Frage, ob die Bezeichnung partieller cbv-Datentyp gerechtfertigt ist.
Um die — selbstverständlich existierende — Beziehung zwischen der totalen und der partiellen cbv-Semantik eindeutig aufzuzeigen, definieren wir die folgende bijektive Abbildung zwischen totalen und partiellen cbv-Interpretationen.

Definition 6.15 Die kanonische Abbildung Π
Die **kanonische Abbildung zwischen totalen und partiellen cbv-Interpretationen**,

$$\Pi : \mathrm{Int}_{\Sigma,\mathrm{cbv}} \to \mathrm{Int}^{\mathrm{part}}_{\Sigma,\mathrm{cbv}},$$

ist definiert durch

$$\begin{array}{lcll} g^{\Pi(\mathfrak{A})}(\underline{\vec{t}}) & := & g^{\mathfrak{A}}(\underline{\vec{t}}) & , \text{ falls } g^{\mathfrak{A}}(\underline{\vec{t}}) \neq \bot \\ g^{\Pi(\mathfrak{A})}(\underline{\vec{t}}) & & \text{undefiniert} & , \text{ andernfalls} \end{array}$$

für alle $g^{(n)} \in \Sigma$, $\underline{\vec{t}} \in (\mathrm{T}_{\mathcal{C}})^n$ und $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\mathrm{cbv}}$. □

Offensichtlich ist $\Pi(\mathfrak{A}) \in \mathrm{Int}^{\mathrm{part}}_{\Sigma,\mathrm{cbv}}$ für alle $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\mathrm{cbv}}$, und die kanonische Abbildung ist somit wohldefiniert. Π ist sowohl injektiv als auch surjektiv, also eine Bijektion. Die daher existierende inverse Abbildung Π^{-1} sieht wie folgt aus:

Definition 6.16 Die Inverse der kanonischen Abbildung Π
Die **Inverse der kanonischen Abbildung Π zwischen totalen und partiellen cbv-Interpretationen**,

$$\Pi^{-1} : \mathrm{Int}^{\mathrm{part}}_{\Sigma,\mathrm{cbv}} \to \mathrm{Int}_{\Sigma,\mathrm{cbv}},$$

ist gegeben durch

$$g^{\Pi^{-1}(\mathfrak{A})}(\underline{\vec{t}}) := \begin{cases} g^{\mathfrak{A}}(\underline{\vec{t}}) & , \text{ falls } \underline{\vec{t}} \in (\mathrm{T}_{\mathcal{C}})^n \text{ und } g^{\mathfrak{A}}(\underline{\vec{t}}) \text{ definiert} \\ \bot & , \text{ andernfalls} \end{cases}$$

für alle $g^{(n)} \in \Sigma$, $\underline{\vec{t}} \in (\mathrm{T}^{\bot}_{\mathcal{C}})^n$ und $\mathfrak{A} \in \mathrm{Int}^{\mathrm{part}}_{\Sigma,\mathrm{cbv}}$. □

Es ist offensichtlich, daß Π und Π^{-1} tatsächlich zueinander invers sind.

Wir benötigen drei Eigenschaften der Abbildung Π und Π^{-1}:

Lemma 6.12 Erhalt der Ordnung unter Π und Π^{-1}
Für alle $\mathfrak{A}, \mathfrak{B} \in \mathrm{Int}_{\Sigma,\mathrm{cbv}}$ gilt

$$\mathfrak{A} \sqsubseteq \mathfrak{B} \iff \Pi(\mathfrak{A}) \sqsubseteq \Pi(\mathfrak{B}),$$

und für alle $\mathfrak{A}, \mathfrak{B} \in \mathrm{Int}^{\mathrm{part}}_{\Sigma,\mathrm{cbv}}$ gilt

$$\mathfrak{A} \sqsubseteq \mathfrak{B} \iff \Pi^{-1}(\mathfrak{A}) \sqsubseteq \Pi^{-1}(\mathfrak{B}).$$

Beweis:
Wir zeigen die Gültigkeit der ersten Äquivalenz.

$\mathfrak{A} \sqsubseteq \mathfrak{B}$

$\iff \forall g \in \Sigma.\ g^{\mathfrak{A}} \preceq g^{\mathfrak{B}}$

$\iff \forall g \in \Sigma.\ \forall \vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C}}^{\perp})^n.\ g^{\mathfrak{A}}(\vec{\underline{t}}) \trianglelefteq g^{\mathfrak{B}}(\vec{\underline{t}})$

$\iff$ (Flache Halbordnung)
$\forall g \in \Sigma.\ \forall \vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C}}^{\perp})^n.\ g^{\mathfrak{A}}(\vec{\underline{t}}) = \perp$ oder $\perp \neq g^{\mathfrak{A}}(\vec{\underline{t}}) = g^{\mathfrak{B}}(\vec{\underline{t}})$

$\iff$ (Striktheit aller Operationen)
$\forall g \in \Sigma.\ \forall \vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C}})^n.$ $g^{\Pi(\mathfrak{A})}(\vec{\underline{t}})$ undefiniert oder
$g^{\Pi(\mathfrak{A})}(\vec{\underline{t}})$ und $g^{\Pi(\mathfrak{B})}(\vec{\underline{t}})$ definiert und $g^{\Pi(\mathfrak{A})}(\vec{\underline{t}}) = g^{\Pi(\mathfrak{B})}(\vec{\underline{t}})$

$\iff$ (Definition der Halbordnung partieller Abbildungen)
$\forall g \in \Sigma.\ g^{\Pi(\mathfrak{A})} \preceq g^{\Pi(\mathfrak{B})}$

$\iff \Pi(\mathfrak{A}) \sqsubseteq \Pi(\mathfrak{B})$

Die Gültigkeit der zweiten Äquivalenz folgt analog (oder direkt aus der Surjektivtät von Π). □

Insbesondere sind Π und Π^{-1} somit auch monoton (bzw. aus der Monotonie und der Bijektivität folgt obere Eigenschaft).

Lemma 6.13 Übereinstimmung der algebraischen Semantiken
Sei $\beta : X \to \mathrm{T}_{\mathcal{C}}$ eine Variablenbelegung und $t \in \mathrm{T}_{\Sigma}$ ein Term.
Für alle cbv-Interpretationen $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\mathrm{cbv}}$ ist entweder

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = \perp \text{ und } [\![t]\!]^{\mathrm{alg}}_{\Pi(\mathfrak{A}),\beta} \text{ undefiniert}$$

oder

$$(\perp \neq)\ [\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![t]\!]^{\mathrm{alg}}_{\Pi(\mathfrak{A}),\beta}.$$

Für alle partiellen cbv-Interpretationen $\mathfrak{A} \in \mathrm{Int}^{\mathrm{part}}_{\Sigma,\mathrm{cbv}}$ ist entweder

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \text{ undefiniert und } [\![t]\!]^{\mathrm{alg}}_{\Pi^{-1}(\mathfrak{A}),\beta} = \perp$$

oder

$$[\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![t]\!]^{\mathrm{alg}}_{\Pi^{-1}(\mathfrak{A}),\beta}\ (\neq \perp).$$

Beweis:
Mit einer strukturellen Induktion über $t \in \mathrm{T}_\Sigma$ folgen unter Verwendung der Definitionen 3.10, S. 47, der algebraischen Semantik und 6.10 der algebraischen Semantik bezüglich partieller Algebren beide Aussagen direkt. □

Lemma 6.14 Erhalt der Modelleigenschaft unter Π und Π^{-1}
Es gelten

$$\mathfrak{A} \in \mathrm{IntMod}^{*}_{P,\mathrm{cbv}} \iff \Pi(\mathfrak{A}) \in \mathrm{IntMod}^{\mathrm{part}}_{P,\mathrm{cbv}}$$

und

$$\mathfrak{A} \in \mathrm{IntMod}^{\mathrm{part}}_{P,\mathrm{cbv}} \iff \Pi^{-1}(\mathfrak{A}) \in \mathrm{IntMod}^{*}_{P,\mathrm{cbv}}.$$

Beweis:
Sei $l \to r$ eine beliebige Reduktionsregel (= Programmregel) des Programms P.

$l \to r$ ist in $\mathfrak{A} \in \mathrm{IntMod}^{*}_{P,\mathrm{cbv}}$ gültig.

$\iff$ (Definition 6.6, S. 142, des cbv-Interpretationsmodells*)
$\mathfrak{A} \in \mathrm{Int}_{\Sigma,\mathrm{cbv}}$, $\forall \beta : X \to \mathrm{T}^{\perp}_{\mathcal{C}} \setminus \{\perp\}$. $[\![l]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}$

$\iff$ $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\mathrm{cbv}}$, $\forall \beta : X \to \mathrm{T}^{\perp}_{\mathcal{C}} \setminus \{\perp\}$. entweder $[\![l]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = \perp$ oder $[\![l]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \neq \perp$

$\iff$ (Lemma 6.13 der Übereinstimmung der algebraischen Termsemantiken)
$\Pi(\mathfrak{A}) \in \mathrm{Int}^{\mathrm{part}}_{\Sigma,\mathrm{cbv}}$, $\forall \beta : X \to \mathrm{T}^{\perp}_{\mathcal{C}} \setminus \{\perp\}$. entweder $[\![l]\!]^{\mathrm{alg}}_{\Pi(\mathfrak{A}),\beta}$ und $[\![r]\!]^{\mathrm{alg}}_{\Pi(\mathfrak{A}),\beta}$ undefiniert oder $[\![l]\!]^{\mathrm{alg}}_{\Pi(\mathfrak{A}),\beta} = [\![r]\!]^{\mathrm{alg}}_{\Pi(\mathfrak{A}),\beta}$

$\iff$ (Definition 6.11 der Gültigkeit in partiellen Algebren)
$l \to r$ ist in $\Pi(\mathfrak{A}) \in \mathrm{IntMod}^{\mathrm{part}}_{P,\mathrm{cbv}}$ gültig.

Die zweite Aussage folgt analog bzw. aus der Surjektivität von Π. □

Mit diesen drei Eigenschaften können wir nun die folgende fundamentale Beziehung zwischen der totalen und der partiellen cbv-Semantik beweisen:

Lemma 6.15 Übereinstimmung der cbv-Datentypen
Es ist

$$\mathcal{D}^{\mathrm{part}}_{P,\mathrm{cbv}} = \Pi(\mathcal{D}_{P,\mathrm{cbv}})$$

und

$$\mathcal{D}_{P,\mathrm{cbv}} = \Pi^{-1}(\mathcal{D}^{\mathrm{part}}_{P,\mathrm{cbv}}).$$

Beweis:
Nach Lemma 6.14 über den Erhalt der Modelleigenschaft unter Π und Π^{-1} ist $\Pi(\mathcal{D}_{P,\mathrm{cbv}}) \in \mathrm{IntMod}^{\mathrm{part}}_{P,\mathrm{cbv}}$.
Sei $\mathfrak{A} \in \mathrm{IntMod}^{\mathrm{part}}_{P,\mathrm{cbv}}$ beliebig. Nach Lemma 6.14 ist $\Pi^{-1}(\mathfrak{A}) \in \mathrm{IntMod}^{*}_{P,\mathrm{cbv}}$. Da nach Lemma 6.11, S. 144, $\mathcal{D}_{P,\mathrm{cbv}}$ das kleinste cbv-Interpretationsmodell* ist, gilt

$$\mathcal{D}_{P,\mathrm{cbv}} \sqsubseteq \Pi^{-1}(\mathfrak{A}).$$

Mit der Monotonie von Π und Π^{-1} gemäß Lemma 6.12 folgt

$$\Pi(\mathcal{D}_{P,\mathrm{cbv}}) \sqsubseteq \Pi(\Pi^{-1}(\mathfrak{A})) = \mathfrak{A}.$$

Da $\mathfrak{A}$ beliebig ist, ist $\Pi(\mathcal{D}_{P,\mathrm{cbv}})$ das kleinste partielle cbv-Interpretationsmodell von P, und somit ist gemäß Definition 6.14 des partiellen cbv-Datentyps

$$\mathcal{D}^{\mathrm{part}}_{P,\mathrm{cbv}} = \Pi(\mathcal{D}_{P,\mathrm{cbv}})$$

Die zweite Gleichung folgt analog bzw. aus der Surjektivität von Π. □

Hiermit haben wir die Berechtigung der Bezeichnung von $\mathcal{D}^{\mathrm{part}}_{P,\mathrm{cbv}}$ als partiellen cbv-Datentyp gezeigt, und außerdem dessen Wohldefiniertheit bewiesen. Die Existenz von $\mathcal{D}_{P,\mathrm{cbv}}$ ist aufgrund der Wohldefiniertheit der cbv-Fixpunktsemantik bekannt, und somit existiert auch $\mathcal{D}^{\mathrm{part}}_{P,\mathrm{cbv}} = \Pi(\mathcal{D}_{P,\mathrm{cbv}})$. Insgesamt stellen wir also fest, daß nicht nur die cbn-Semantik, sondern auch die cbv-Semantik deklarativ definierbar ist. Wie wir sehen, ist die Wahl der zugrundeliegenden mathematischen Struktur von entscheidender Bedeutung. Bei der cbn-Semantik sind es die geordneten totalen Algebren, während zu der cbv-Semantik die auch intuitiv natürlicheren partiellen Algebren am besten passen.

BEMERKUNG 6.1: Verzweigungskontrollstrukturen
Alle Aussagen dieses Abschnitts 6.3 betreffen nur Programme mit Pattern. Es ist aber durchaus möglich, für Programme, die ähnliche Verzweigungskontrollstrukturen wie die Verzweigungssymbole cond_G der Programme mit Hilfsfunktionen besitzen, eine cbv-Semantik auf der Grundlage partieller Algebren zu definieren. Hierbei dürfen die Verzweigungssymbole nur nicht mehr Teil der Signatur Σ sein, und ihre Semantik muß völlig getrennt von der Semantik der Signatursymbole betrachtet werden.
Beispielsweise könnten sogenannte „Guards“, wie sie in Miranda und Haskell existieren, definiert werden:

$$\begin{array}{lcl} f(p_1,\ldots,p_n) & \rightarrow & t_1,\ \mathrm{if}_{G_1}(s_1) \\ & & t_2,\ \mathrm{if}_{G_2}(s_2) \\ & & \vdots \\ & & t_m,\ \mathrm{otherwise} \end{array}$$

Nur $t_1,\ldots,t_m$ und $s_1,\ldots,s_{m-1}$ sind Terme, während die gesamte rechte Seite ein semantisch getrennt von diesen zu betrachtendes Konstrukt ist.
Diese „Guards“ vereinfachen die Programmierung. Sie erhöhen keinesfalls die Berechnungsstärke der Programme, da sich Programme mit „Guards“ leicht in Programme mit Pattern mit zusätzlichen Funktionssymbolen übersetzen ließen (vgl. 7.2).
In [Broy&Wir83] werden partielle Algebren sogar für nicht-strikte Semantiken verwendet. Dies geschieht aber auch erst vermittels eines Umwegs über totale Algebren, die dann auf partielle Algebren abgebildet werden. □

BEMERKUNG 6.2: Totale Homomorphismen und Definiertheitsprädikate
Wir haben partielle cbv-Interpretationen zusammen mit einer kanonischen Halbordnung definiert, um mit deren Hilfe den partiellen cbv-Datentypen $\mathcal{D}^{\mathrm{part}}_{P,\mathrm{cbv}}$ festzulegen. Dies geschah so, um eine Analogie zu unserem Vorgehen bei totalen Algebren zu schaffen. Es entspricht jedoch nicht der üblichen, in Abschnitt 3.2.2 in [Wir90] beschriebenen Methode.

- Wie schon erwähnt, kennt man mehrere verschiedene Homomorphismen für partielle Algebren. Durch die Verwendung des sogenannten **totalen Homomorphismus partieller Algebren** wird die kanonische Halbordnung partieller cbv-Interpretationen, und damit Halbordnungen generell, überflüssig. Sind $\mathfrak{A}$ und $\mathfrak{B}$ partielle Algebren, so heißt eine totale Abbildung $h : A \to B$ genau dann totaler Homomorphismus von $\mathfrak{A}$ nach $\mathfrak{B}$, wenn für alle $g^{(n)} \in \Sigma$ und $a_1, \ldots, a_n \in A$ gilt:

$$g^{\mathfrak{A}}(a_1, \ldots, a_n) \text{ definiert} \implies g^{\mathfrak{B}}(h(a_1), \ldots, h(a_n)) \text{ definiert und } h(g^{\mathfrak{A}}(a_1, \ldots, a_n)) = g^{\mathfrak{B}}(h(a_1), \ldots, h(a_n)).$$

  Der partielle cbv-Datentyp $\mathcal{D}^{\text{part}}_{P,\text{cbv}}$ des Programms ist schlicht das initiale partielle Modell aller partiellen cbv-Interpretationsmodelle des Programms. Im Prinzip wird hier die Halbordnung nur in den Homomorphismus verlagert. Es bleibt das Problem, zu beweisen, daß dieses initiale partielle Modell auch existiert.

- Statt durch die Definition von partiellen cbv-Interpretationen den gewünschten Träger $\mathrm{T}_{\mathcal{C}}$ und die Konstruktoroperationen festzulegen, wird ein Definiertheitsprädikat eingeführt. Dieses **Definiertheitsprädikat** D legt fest, daß die Semantik bestimmter Terme definiert sein muß: $\mathrm{D}(t)$ ist genau dann in einer partiellen Algebra $\mathfrak{A}$ gültig, wenn $[\![t]\!]^{\text{alg}}_{\mathfrak{A},\beta}$ für alle $\beta : X \to A$ definiert ist. Zu den Reduktionsregeln werden die Definiertheitsformeln $\mathrm{D}(G(x_1, \ldots, x_n))$ für alle Konstruktorsymbole $G^{(n)} \in \mathcal{C}$ hinzugenommen. In der initialen partiellen Algebra aller partiellen Modelle sind nun einerseits die semantischen Werte aller Konstruktorgrundterme paarweise verschieden, andererseits existieren im Träger auch keine nicht durch einen Konstruktorterm denotierten Elemente. Auch die Konstruktoroperationen sind direkt passend festgelegt. Die initiale partielle Algebra aller partiellen Modelle eines um Definiertheitsformeln erweiterten Programms ist isomorph zu der gerade genannten initialen partiellen Algebra aller cbv-Interpretationsmodelle.

Während das Prinzip des initialen cbv-Interpretationsmodells bezüglich totaler Homomorphismen unabhängig von der Verwendung eines Definiertheitsprädikats ist, ist die umgekehrte Unabhängigkeit nicht gegeben. Die Definition einer Halbordnung von partiellen Algebren setzt einen vorgegebenen, festen Träger voraus (vgl. Def. 6.9).
Da für totale Algebren kein Analogon zum totalen Homomorphismus partieller Algebren existiert, lassen sich die hier beschriebenen Methoden bei diesen nicht anwenden. □

Kapitel 7

Sequentialität

In diesem Kapitel beschäftigen wir uns mit der Ausdrucks- bzw. Berechnungsstärke unserer Programmiersprachen, d. h. der Programme mit Pattern und der Programme mit Hilfsfunktionen zusammen mit den ς-Semantiken.
Zuerst werfen wir in 7.1 einen Blick auf die verschiedenen Vergleichsmöglichkeiten.
Eine von diesen untersuchen wir dann in 7.2 genauer: In Anbetracht der Tatsache, daß wir einerseits Programme mit Pattern und andererseits Programme mit Hilfsfunktionen definiert haben, stellt sich die Frage, ob beide Programmarten in Verbindung mit den ς-Semantiken gleich ausdrucksstark sind, und ob sich Programme der einen Art in Programme der anderen Art effektiv semantikerhaltend übersetzen lassen. Dabei stellen wir fest, daß Programme mit Pattern echt ausdrucksstärker als Programme mit Hilfsfunktionen sind.
In 7.3 beweisen wir diese Aussage formal mit Hilfe der semantischen Eigenschaft der Sequentialität.
Da die Sequentialität eine fundamentale Eigenschaft im Bereich funktionaler Programmiersprachen und dem Kalkül der Termersetzungssysteme ist, betrachten wir sie in 7.4 noch näher. Sie stellt exakt das Scheidekriterium zwischen den Ausdrucksstärken der Programme mit Pattern und der mit Hilfsfunktionen dar. Auch liefert sie Anregungen für effiziente ς-Reduktionsstrategien.

7.1 Berechnungsstärke unserer Programmiersprachen

Da, wie wir schon in der Einleitung anmerkten, alle in einer funktionalen Programmiersprache spezifizierbaren Abbildungen auch berechenbar sind (mit gewissen Einschränkungen bezüglich der partiellen und der unendlichen Konstruktorterme), sind die Begriffe Ausdruckstärke und Berechnungsstärke in diesem Rahmen austauschbar.

Wollen wir die absolute Berechungsstärke unserer Programmiersprachen bestimmen, so müssen wir feststellen, daß diesbezüglich durchaus unterschiedliche Definitionen existieren.
In der klassischen Berechenbarkeitstheorie werden nur berechenbare (rekursive) Abbildungen über den natürlichen Zahlen und über der Halbgruppe der Wörter (Zeichenketten) betrachtet ([Hermes78]). Wenn wir die natürlichen Zahlen durch die Konstruktorsymbole `Zero` und `Succ` aufbauen, so können wir alle berechenbaren Abbildungen über den natürlichen Zahlen spezifizieren, denn in unseren Programmiersprachen sind Nachfolgerbildung($\texttt{Succ}^{\mathcal{D}_{P,\varsigma}}$), Projektion (Pattern oder Selektion $\text{sel}_{G,i}^{\mathcal{D}_{P,\varsigma}}$), Konstantenbildung, primitive Rekursion und Minimalisierung spezifizierbar. Ebenso lassen sich offensichtlich alle berechenbaren Abbildungen über Wörtern spezifizieren (vgl. [Dau89]).

In [Hup78] wird der Formalismus der rekursiven Abbildungen auf beliebige initiale Algebren übertragen. Auch hier ist leicht erkennbar, daß in unseren Programmiersprachen alle in diesem Sinne rekursiven Abbildungen über $T_{\mathcal{C}}$ spezifizierbar sind.
In [Ber&Tu80] und 2.4 und 4.2 in [Wir90] werden sogenannte berechenbare Datentypen, d. h. Algebren, deren Operationen alle berechenbar sind, definiert. Mit unseren Programmiersprachen sind nicht alle in diesem Sinne berechenbaren konstruktorbasierten Datentypen spezifizierbar. Immerhin existiert jedoch zu jedem berechenbaren Datentyp $\mathfrak{A}$ ein ς-Datentyp $\mathcal{D}_{P,\varsigma}$, so daß $\mathcal{D}_{P,\varsigma}$ $\mathfrak{A}$ enthält, d. h. nur mehr Operationen besitzt aber ansonsten $\mathfrak{A}$ gleicht. So ist eine Quadrierungsabbildung über den natürlichen Zahlen nicht ohne Verwendung einer anderen Abbildung, beispielsweise der Addition, definierbar (siehe Theorem 4.2.4 in [Wir90]).
Interessant wäre noch ein absoluter Berechenbarkeitsbegriff, der auch die partiellen und unendlichen Konstruktorterme umfassen und dabei möglicherweise auch die Halbordnung des Rechenbereichs mit einbeziehen würde.

Die obigen Aussagen über die absolute Berechungsstärke gelten für Programme mit Pattern und für Programme mit Hilfsfunktionen zusammen mit beliebigen ς-Semantiken.
Eine andere Untersuchungsmöglichkeit ist ein direkter Vergleich zwischen den ς-Semantiken, d. h. den verschiedenen erzwungenen Striktheiten ς. Hierbei ist insbesondere von Interesse, ob es ein effektives Verfahren zur Übersetzung eines Programms P in ein Programm P' gibt, so daß der ς'-Datentyp von P' mit dem ς-Datentyp von P übereinstimmt. Aufgrund der unterschiedlichen Rechenbereiche werden die Operationen dabei nur über $T_{\mathcal{C}}$ verglichen, und außerdem wird bei der Konstruktion von P' die Erweiterung der Signatur um zusätzliche Operationssymbole zugelassen. In [Noll95] werden in einem unseren Programmiersprachen ähnlichem Rahmen derartige Übersetzungsverfahren angegeben.

Mit einem anderen Vergleich beschäftigen wir uns eingehender. Da wir zwei unterschiedliche Arten von Programmen definiert haben, vergleichen wir deren Ausdrucksstärke und prüfen die gegenseitige semantikerhaltende Übersetzbarkeit der Programme.

7.2 Gegenseitige Übersetzbarkeit der Programme

Präzise formuliert prüfen wir, ob ein Programm mit Hilfsfunktionen (mit Pattern), P, über der Programmsignatur $\Sigma = (\mathcal{C}, \mathcal{H}, \mathcal{F})$ ($\Sigma = (\mathcal{C}, \mathcal{F})$) in ein Programm mit Pattern (mit Hilfsfunktionen), P', über $\Sigma' = (\mathcal{C}, \mathcal{F} \dot\cup \mathcal{F}')$ ($\Sigma' = (\mathcal{C}, \mathcal{H}, \mathcal{F} \dot\cup \mathcal{F}')$) effektiv übersetzbar ist, so daß mit $\varsigma'(g) = \varsigma(g)$ für alle $g \in \Sigma \cap \Sigma'$ für alle $f \in \mathcal{F}$ $f^{\mathcal{D}_{P,\varsigma}} = f^{\mathcal{D}_{P',\varsigma'}}$ gilt. Wir lassen also die Erweiterung um zusätzliche Funktionssymbole zu. Da die erzwungene Striktheit (für die gemeinsamen Operationssymbole) fest ist, können wir die Operationen über dem gesamten Rechenbereich $T_{\mathcal{C},\varsigma}$ vergleichen.

Die semantikerhaltende Übersetzung von Programmen mit Hilfsfunktionen in Programme mit Pattern ist trivial, da die assoziierten Termersetzungssysteme von Programmen mit Hilfsfunktionen praktisch Programme mit Pattern sind. Ist P ein Programm mit Hilfsfunktionen über $\Sigma = (\mathcal{C}, \mathcal{H}, \mathcal{F})$, so erfüllt $P' := \hat{P}$ über $\Sigma = (\mathcal{C}, \mathcal{F} \dot\cup \mathcal{H})$ obige Bedingung.
Eine weitere natürliche Forderung ist hierbei jedoch noch nicht erfüllt. Ist die erzwungene Striktheit $\varsigma = \mathrm{cbn}$, so soll auch $\varsigma' = \mathrm{cbn}$ sein, und wenn $\varsigma = \mathrm{cbv}$, dann soll $\varsigma' = \mathrm{cbv}$ sein. Letztere Bedingung ist problematisch, da die Verzweigungssymbole an den jeweils letzten zwei Argumentstellen auch in der cbv-Semantik nicht-strikt sind, während die Funktionssymbole dann alle erzwungen strikt sein müssen.

Es ist jedoch möglich, auf die Verzweigungssymbole zu verzichten, indem für jedes Vorkommen eines Verzweigungssymbols ein neues (an allen Stellen erzwungen striktes) Funktionssymbol eingeführt wird.

Beispiel 7.1 Programm mit Hilfsfunktionen $\rightsquigarrow$ Programm mit Pattern

$$\texttt{add(x,y)} \quad \rightarrow \quad \text{cond}_{\texttt{Succ}}\texttt{(y, Succ(add(x, sel}_{\texttt{Succ,1}}\texttt{(y))), x)}$$

wird übersetzt in

$$\begin{array}{lcl} \texttt{add(x,y)} & \rightarrow & \texttt{new(y,x,y)} \\ \texttt{new(Succ(y'),x,y)} & \rightarrow & \texttt{Succ(add(x, sel}_{\texttt{Succ,1}}\texttt{(y)))} \\ \texttt{new(Zero,x,y)} & \rightarrow & \texttt{x} \end{array}$$

□

Das mit obigem Beispiel angedeutete Verfahren ließe sich selbstverständlich noch optimieren. Wir verzichten jedoch überhaupt auf eine Definition des Übersetzungsverfahrens und einen Beweis seiner Korrektheit. Stattdessen wenden wir uns der umgekehrten, weitaus wichtigeren Übersetzungsrichtung zu.

Wie schon zur Motivation der Definition der Programme mit Hilfsfunktionen erwähnt, ist Patternmatching zwar intuitiv eingängig und ermöglicht einfachere Spezifikationen, aber es ist auch schwer direkt implementierbar. Dagegen ist die Reduktion durch eine Regel, auf deren linker Seite sich nur Variablen befinden, relativ leicht implementierbar. Die Reduktionen für die Hilfssymbole werden nicht vermittels der ihnen zugeordneten Reduktionsregeln durchgeführt, sondern direkt implementiert. Dieses Verfahren wird auch bei modernen funktionalen Programmiersprachen verwendet. Aus den zu einem Funktionssymbol gehörenden Pattern wird ein sogenannter Matching-Tree generiert (Kapitel 8 in [Fie&Har88], 5 in [Huet&Lévy79] und [Lav87]), in dem die zu einem bestimmten Funktionsaufruf passende Gleichung ausgewählt wird. Dies ist im Prinzip eine Übersetzung von Programmen mit Pattern in Programme mit Hilfsfunktionen.
Wie jedoch auch schon erwähnt, ist das Patternmatching in den verbreiteten funktionalen Programmiersprachen anders definiert als in unseren Programmiersprachen. In den meisten werden die linken Gleichungsseiten eines Funktionssymbols von oben nach unten getestet und die erste passende gewählt. Außerdem erfolgt die (teilweise) Auswertung der Argumente immer von links nach rechts.

Beispiel 7.2 Patternmatching in verbreiteten funktionalen Programmiersprachen

$$\begin{array}{lcl} \texttt{f(A,B)} & = & \texttt{A} \\ \texttt{f(x,y)} & = & \texttt{B} \\ \texttt{undef} & = & \texttt{undef} \end{array}$$

sei ein funktionales Programm in einer nicht-strikten Sprache wie Miranda oder Haskell.
Da immer die erste passende Gleichung verwendet wird, gilt

$$\texttt{f(A,B)} \xrightarrow[\text{cbn}]{*} \texttt{A}.$$

Es gilt

$$\texttt{f(B,undef)} \xrightarrow[\text{cbn}]{*} \texttt{B},$$

da das 1. Argument `B` nicht zum Pattern `A` der ersten Gleichung paßt, aber die zweite Gleichung anwendbar ist.
Der Term `f(B,undef)` demonstriert auch, von welcher Bedeutung die Richtung des Patternmatchings von links nach rechts ist. Würde das Patternmatching von rechts nach links durchgeführt werden, würde zuerst versucht werden, `undef` mit `B` zu matchen. Dafür müßte `undef` ausgewertet werden. Da diese Reduktion nicht terminiert, würde die Berechnung des Gesamtterms `f(B,undef)` nie terminieren. Genau dies geschieht hier bei dem Term `f(undef,A)`:

$$\texttt{f(}\underline{\texttt{undef}}\texttt{,A)} \xrightarrow[\text{cbn}]{} \texttt{f(}\underline{\texttt{undef}}\texttt{,A)} \xrightarrow[\text{cbn}]{} \ldots$$

□

Offensichtlich sind bei einer derartigen Semantik des Patternmatchings Programme mit Pattern leicht semantikerhaltend in Programme mit Hilfsfunktionen übersetzbar. Diese Leichtigkeit der Implementierung wird aber mit einer weitaus komplizierteren denotationellen oder gar deklarativen Semantik erkauft. In [Tho89] und [Tho94] wird ein Verfahren zur Verifikation von Miranda-Programmen durch eine Übersetzung dieser in eine Gleichungslogik und Verwendung eines allgemeinen Theorembeweisers entwickelt. Es zeigt sich dort, daß die Übersetzung für das Pattermatching äußerst komplex ist.
Unser einfaches semantisches cbn-Matchen ist dagegen weitaus intuitiver und, wie wir in Kapitel 6 gesehen haben, leicht deklarativ definierbar. Dafür ist bei unseren ς-Semantiken die Übersetzung von Programmen mit Pattern in Programme mit Hilfsfunktionen im allgemeinen nicht mehr möglich:

Beispiel 7.3 **Paralleles And** (vgl. Bsp. 4.7, S. 67)

```
and(False,x)     →  False
and(x,False)     →  False
and(True,True)   →  True
```

$\texttt{and}^{\mathcal{D}_{P,\text{cbn}}}$	$\bot$	`False`	`True`
$\bot$	$\bot$	`False`	$\bot$
`False`	`False`	`False`	`False`
`True`	$\bot$	`False`	`True`

Dieses Programm mit Pattern ist nicht in ein äquivalentes Programm mit Hilfsfunktionen übersetzbar, da mit Hilfe der Verzweigungssymbole die zwei Argumente von `and` nur nacheinander mit `True` und `False` verglichen werden können:

$$\texttt{and}_1\texttt{(x,y)} \rightarrow \text{cond}_{\texttt{False}}\texttt{(x,False,}\text{cond}_{\texttt{False}}\texttt{(y,False,True))}$$

oder

$$\texttt{and}_2\texttt{(x,y)} \rightarrow \text{cond}_{\texttt{False}}\texttt{(y,False,}\text{cond}_{\texttt{False}}\texttt{(x,False,True))}$$

Dies bedeutet aber

$$\begin{aligned} \texttt{and}_1^{\mathcal{D}_{P_1,\text{cbn}}}(\bot,\texttt{False}) &= \bot, \\ \texttt{and}_2^{\mathcal{D}_{P_2,\text{cbn}}}(\texttt{False},\bot) &= \bot \end{aligned}$$

wogegen

$$\begin{aligned} \texttt{and}^{\mathcal{D}_{P,\text{cbn}}}(\bot,\texttt{False}) &= \texttt{False}, \\ \texttt{and}^{\mathcal{D}_{P,\text{cbn}}}(\texttt{False},\bot) &= \texttt{False}. \end{aligned}$$

□

Auf den ersten Blick scheint es, als könnte dieses Problem durch das Verbieten überlappender linker Programmregelseiten beseitigt werden. Berrys Beispiel zeigt jedoch, daß dem keinesfalls so ist:

Beispiel 7.4 **Berrys Beispiel** (S. 280 in [Ber&Cur82], S. 188 in [Fie&Har88], [Lav87])

$$\begin{array}{lcl} \mathtt{f(x,A,B)} & \to & \mathtt{C} \\ \mathtt{f(B,x,A)} & \to & \mathtt{D} \\ \mathtt{f(A,B,x)} & \to & \mathtt{E} \end{array}$$

Weder

$$\mathtt{f}_1\mathtt{(x,y,z)} \quad \to \quad \mathrm{cond}_{\mathtt{A/B}}\mathtt{(x,...)}$$

noch

$$\mathtt{f}_2\mathtt{(x,y,z)} \quad \to \quad \mathrm{cond}_{\mathtt{A/B}}\mathtt{(y,...)}$$

noch

$$\mathtt{f}_3\mathtt{(x,y,z)} \quad \to \quad \mathrm{cond}_{\mathtt{A/B}}\mathtt{(z,...)}$$

stellen Ansätze für eine semantikerhaltende Übersetzung dar, da

$$\begin{array}{lcllcl} \mathtt{f}_1^{\mathcal{D}_{P_1,\mathrm{cbn}}}(\bot,\mathtt{A},\mathtt{B}) & = & \bot & \text{, aber} & \mathtt{f}^{\mathcal{D}_{P,\mathrm{cbn}}}(\bot,\mathtt{A},\mathtt{B}) & = & \mathtt{C} \\ \mathtt{f}_2^{\mathcal{D}_{P_2,\mathrm{cbn}}}(\mathtt{B},\bot,\mathtt{A}) & = & \bot & \text{, aber} & \mathtt{f}^{\mathcal{D}_{P,\mathrm{cbn}}}(\mathtt{B},\bot,\mathtt{A}) & = & \mathtt{D} \\ \mathtt{f}_3^{\mathcal{D}_{P_3,\mathrm{cbn}}}(\mathtt{A},\mathtt{B},\bot) & = & \bot & \text{, aber} & \mathtt{f}^{\mathcal{D}_{P,\mathrm{cbn}}}(\mathtt{A},\mathtt{B},\bot) & = & \mathtt{E} \end{array}$$

ist. □

Wir haben in beiden Beispielen die cbn-Semantik verwendet. Tatsächlich existiert das dargestellte Problem in der cbv-Semantik nicht. Dort werden bekanntlich — wie auch aus der li-Reduktionssemantik direkt ersichtlich — vor dem „Funktionsaufruf“ die Funktionsargumente vollständig ausgewertet. Somit können die Konstruktortests in einer beliebigen Reihenfolge erfolgen. Für die cbv-Semantik lassen sich Programme mit Pattern also offensichtlich semantikerhaltend in Programme mit Hilfsfunktionen übersetzen.
Für alle anderen erzwungenen Striktheiten ς sind jedoch leicht zu den obigen Beispielprogrammen analoge Programme konstruierbar.
Die bisherige Argumentation für die Unmöglichkeit einer Übersetzung ist zwar intuitiv einleuchtend, aber stellt noch keinen formalen Beweis dar. Diesen führen wir im folgenden Abschnitt.

7.3 Sequentialität

In Programmen mit Hilfsfunktionen können die Argumente einer Funktion nur nacheinander auf ihre Konstruktorsymbolkomponenten getestet werden. Dies führt dazu, daß alle durch Programme mit Hilfsfunktionen spezifizierten Operationen in einem gewissen Sinne sequentiell sind.

Definition 7.1 Sequentielle Abbildung
Sei $\varphi : (\mathrm{T}_{\mathcal{C},\varsigma})^n \to \mathrm{T}_{\mathcal{C},\varsigma}$ mit $n \in \mathbb{N}$ eine monotone Abbildung. φ heißt genau dann **sequentiell**, wenn

$$\begin{aligned}\forall \vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n.\quad & (\mathrm{Occ}(\bot, \vec{\underline{t}}) = \emptyset \vee \\ & \forall u' \in \mathrm{Occ}(\bot, \varphi(\vec{\underline{t}})).\ \exists u \in \mathrm{Occ}(\bot, \vec{\underline{t}}). \\ & \quad \forall \vec{\underline{t}'} \trianglerighteq \vec{\underline{t}}.\ (\varphi(\vec{\underline{t}'})/u' \neq \bot \implies \vec{\underline{t}'}/u \neq \bot) \\ &).\end{aligned}$$

Hierbei heißt u **Sequentialitätsindex** von φ für $\vec{\underline{t}}$ bezüglich u'. Ist für ein $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ die in den äußeren Klammern stehende Teilformel erfüllt, so heißt φ **sequentiell für** $\vec{\underline{t}}$. □

Diese Definition ist eine Adaption der in [Ber&Cur82] für sogenannte concrete domains angegebenen Definition 3.4.1., die dort jedoch nicht weiter verwendet wird. Sie geht auf [Kahn&Plot78] zurück. Lax formuliert ist eine Abbildung φ für ein Argumenttupel $\vec{\underline{t}}$ genau dann sequentiell, wenn zu jedem $\bot$ des Ergebnisses $\varphi(\vec{\underline{t}})$ ein $\bot$ im Argumenttupel existiert, welches erst verschwinden muß, bevor dieses $\bot$ im Ergebnis verschwinden kann. Es wird also an einer ganz bestimmten Stelle in den Argumenten zusätzliche Information benötigt, damit das Ergebnis an einer gegebenen Stelle zusätzliche Information liefern kann.
Wir veranschaulichen dies mit Hilfe mehrerer Beispiele:

Beispiel 7.5 Striktes And (vgl. Bsp. 3.2, S. 39)

```
and(True,x)        →  x
and(y,True)        →  y
and(False,False)   →  False
```

$\mathtt{and}^{\mathcal{D}_{P,\mathrm{cbn}}}$	$\bot$	False	True
$\bot$	$\bot$	$\bot$	$\bot$
False	$\bot$	False	False
True	$\bot$	False	True

1. $\vec{\underline{t}} := (\bot, \bot),\ \mathtt{and}^{\mathcal{D}_{P,\mathrm{cbn}}}(\vec{\underline{t}}) = \bot,\ \mathrm{Occ}(\bot, \mathtt{and}^{\mathcal{D}_{P,\mathrm{cbn}}}(\vec{\underline{t}})) = \{\varepsilon\}$.

   Sowohl $u := 1$ als auch $u := 2$ sind Sequentialitätsindexe von $\mathtt{and}^{\mathcal{D}_{P,\mathrm{cbn}}}$ für $(\bot, \bot)$ bezüglich $u' := \varepsilon$, denn für alle $\vec{\underline{t}'} \trianglerighteq \vec{\underline{t}}$ folgt aus $\mathtt{and}^{\mathcal{D}_{P,\mathrm{cbn}}}(\vec{\underline{t}})/\varepsilon = \mathtt{and}^{\mathcal{D}_{P,\mathrm{cbn}}}(\vec{\underline{t}}) \neq \bot$, daß auch $\vec{\underline{t}'}/u = \underline{t}_{1/2} \neq \bot$ sein muß.

   Also ist $\mathtt{and}^{\mathcal{D}_{P,\mathrm{cbn}}}$ sequentiell für $\vec{\underline{t}}$.

2. $\vec{\underline{t}} := (\mathtt{True}, \bot),\ \mathtt{and}^{\mathcal{D}_{P,\mathrm{cbn}}}(\vec{\underline{t}}) = \bot,\ \mathrm{Occ}(\bot, \mathtt{and}^{\mathcal{D}_{P,\mathrm{cbn}}}(\vec{\underline{t}})) = \{\varepsilon\}$.

   $u := 2$ ist der Sequentialitätsindex von $\mathtt{and}^{\mathcal{D}_{P,\mathrm{cbn}}}$ für $(\mathtt{True}, \bot)$ bezüglich $u' := \varepsilon$, denn für alle $\vec{\underline{t}'} \trianglerighteq \vec{\underline{t}}$ folgt aus $\mathtt{and}^{\mathcal{D}_{P,\mathrm{cbn}}}(\vec{\underline{t}})/\varepsilon = \mathtt{and}^{\mathcal{D}_{P,\mathrm{cbn}}}(\vec{\underline{t}}) \neq \bot$, daß auch $\vec{\underline{t}'}/u = \underline{t}_2 \neq \bot$ sein muß.

   Also ist $\mathtt{and}^{\mathcal{D}_{P,\mathrm{cbn}}}$ sequentiell für $\vec{\underline{t}}$.

3. $\vec{\underline{t}} := (\bot, \mathtt{True})$. Die Sequentialität von $\mathtt{and}^{\mathcal{D}_{P,\mathrm{cbn}}}$ für $\vec{\underline{t}}$ folgt analog zu 2.

Für alle anderen $\vec{\underline{t}} \in (\mathrm{T}^{\infty}_{\mathcal{C},\bot})^2$ ist $\mathrm{Occ}(\bot, \vec{\underline{t}}) = \emptyset$.
Somit ist das strikte And $\mathtt{and}^{\mathcal{D}_{P,\mathrm{cbn}}}$ sequentiell. □

Beispiel 7.6 **Paralleles And** (vgl. Bsp. 7.3)

```
and(False,x)     →  False
and(x,False)     →  False
and(True,True)   →  True
```

$\texttt{and}^{\mathcal{D}_{P,\text{cbn}}}$	$\bot$	`False`	`True`
$\bot$	$\bot$	`False`	$\bot$
`False`	`False`	`False`	`False`
`True`	$\bot$	`False`	`True`

Das parallele And $\texttt{and}^{\mathcal{D}_{P,\text{cbn}}}$ ist nicht sequentiell:
$\texttt{and}^{\mathcal{D}_{P,\text{cbn}}}$ besitzt für $\vec{t} := (\bot, \bot)$ bezüglich $u' := \varepsilon \in \text{Occ}(\bot, \texttt{and}^{\mathcal{D}_{P,\text{cbn}}}(\vec{t})) = \text{Occ}(\bot, \bot)$ keinen Sequentialitätsindex.
$u = 1 \in \text{Occ}(\bot, \vec{t})$ ist kein Sequentialitätsindex hierfür, da für $\vec{t'} := (\bot, \texttt{False}) \trianglerighteq \vec{t}$

$$\texttt{and}^{\mathcal{D}_{P,\text{cbn}}}(\vec{t'})/u' = \texttt{False}/\varepsilon = \texttt{False} \neq \bot,$$

aber trotzdem

$$\vec{t'}/u = (\bot, \texttt{False})/1 = \bot.$$

$u = 2 \in \text{Occ}(\bot, \vec{t})$ ist kein Sequentialitätsindex hierfür, da für $\vec{t'} := (\texttt{False}, \bot) \trianglerighteq \vec{t}$

$$\texttt{and}^{\mathcal{D}_{P,\text{cbn}}}(\vec{t'})/u' = \texttt{False}/\varepsilon = \texttt{False} \neq \bot,$$

aber trotzdem

$$\vec{t'}/u = (\texttt{False}, \bot)/2 = \bot.$$

□

Beispiel 7.7 **Berrys Beispiel** (vgl. Bsp. 7.4)

```
f(x,A,B)  →  C
f(B,x,A)  →  D
f(A,B,x)  →  E
```

Die Operation $f^{\mathcal{D}_{P,\text{cbn}}}$ ist nicht sequentiell:

$$\texttt{f}^{\mathcal{D}_{P,\text{cbn}}}(\bot, \bot, \bot) = \bot$$

$\texttt{f}^{\mathcal{D}_{P,\text{cbn}}}$ besitzt für $\vec{t} := (\bot, \bot, \bot)$ bezüglich $u' := \varepsilon \in \text{Occ}(\bot, \texttt{f}^{\mathcal{D}_{P,\text{cbn}}}(\bot, \bot, \bot))$ keinen Sequentialitätsindex.
$u = 1 \in \text{Occ}(\bot, \vec{t})$ ist kein Sequentialitätsindex hierfür, da für $\vec{t'} := (\bot, \texttt{A}, \texttt{B}) \trianglerighteq \vec{t}$

$$\texttt{f}^{\mathcal{D}_{P,\text{cbn}}}(\vec{t'})/u' = \texttt{C}/\varepsilon = \texttt{C} \neq \bot,$$

aber trotzdem

$$\vec{t'}/u = (\bot, \texttt{A}, \texttt{B})/1 = \bot.$$

Entsprechend sind die übrigen „⊥-Stellen“ 2 und 3 keine Sequentialitätsindexe von $\texttt{f}^{\mathcal{D}_{P,\text{cbn}}}$ für $(\bot, \bot, \bot)$ bezüglich ε. □

Offensichtlich sind durch Programme mit Pattern auch nicht-sequentielle Operationen spezifizierbar. Dagegen sind alle durch Programme mit Hilfsfunktionen spezifizierbaren Operationen sequentiell.
Dies beweisen wir im folgenden durch eine Reihe von Lemmata. Im wesentlichen zeigen wir, daß alle Operationen der kleinsten ς-Interpretation $\bot_\varsigma$ sequentiell sind, daß bei Anwendung der Transformation $\Phi_{P,\varsigma}$ eines Programms mit Hilfsfunktionen, P, wiederum nur sequentielle Operationen entstehen, und daß die Sequentialität bei Bildung der kleinsten oberen Schranke erhalten bleibt.

Lemma 7.1 Sequentialität der Projektionen
Sei $n \in \mathbb{N}$ und $i \in [n]$. Die Projektion

$$\Pi_{i,n} : (\mathrm{T}_{\mathcal{C},\varsigma})^n \to \mathrm{T}_{\mathcal{C},\varsigma},$$

die durch

$$\Pi_{i,n}(t_1, \ldots, t_n) := t_i$$

definiert ist, ist sequentiell.

Beweis:
Die Projektion $\Pi_{i,n}$ ist monoton.
Sei $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ mit $\mathrm{Occ}(\bot, \vec{\underline{t}}) \neq \emptyset$. Sei $u' \in \mathrm{Occ}(\bot, \Pi_{i,n}(\vec{\underline{t}})) = \mathrm{Occ}(\bot, \underline{t}_i)$. Dann ist $u := i.u'$ ein Sequentialitätsindex von $\Pi_{i,n}$ für $\vec{\underline{t}}$ bezüglich u'. □

Lemma 7.2 Sequentialität der Komposition sequentieller Abbildungen
Seien $m, n \in \mathbb{N}$ und $\varphi : (\mathrm{T}_{\mathcal{C},\varsigma})^m \to \mathrm{T}_{\mathcal{C},\varsigma}$ und $\varphi_1, \ldots, \varphi_m : (\mathrm{T}_{\mathcal{C},\varsigma})^n \to \mathrm{T}_{\mathcal{C},\varsigma}$ sequentielle Abbildungen. Dann ist $\psi := \varphi \circ (\varphi_1, \ldots, \varphi_m) : (\mathrm{T}_{\mathcal{C},\varsigma})^n \to \mathrm{T}_{\mathcal{C},\varsigma}$ eine sequentielle Abbildung.

Beweis:
ψ ist monoton, denn ist $\vec{\underline{t'}} \trianglerighteq \vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$, so ist $(\varphi_1(\vec{\underline{t'}}), \ldots, \varphi_m(\vec{\underline{t'}})) \trianglerighteq (\varphi_1(\vec{\underline{t}}), \ldots, \varphi_m(\vec{\underline{t}}))$, und damit $\psi(\vec{\underline{t'}}) \trianglerighteq \psi(\vec{\underline{t}})$.
Sei $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ mit $\mathrm{Occ}(\bot, \vec{\underline{t}}) \neq \emptyset$ und $u' \in \mathrm{Occ}(\bot, \psi(\vec{\underline{t}})) = \mathrm{Occ}(\bot, \varphi(\varphi_1(\vec{\underline{t}}), \ldots, \varphi_m(\vec{\underline{t}})))$.

Fall 1: $\mathrm{Occ}(\bot, (\varphi_1(\vec{\underline{t}}), \ldots, \varphi_m(\vec{\underline{t}}))) = \emptyset$.

Dann existiert kein $\vec{\underline{t''}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^m$ mit $\vec{\underline{t''}} \triangleright (\varphi_1(\vec{\underline{t}}), \ldots, \varphi_m(\vec{\underline{t}}))$. Wegen der Monotonie von φ ist damit für alle $\vec{\underline{t'}} \trianglerighteq \vec{\underline{t}}$

$$\psi(\vec{\underline{t'}}) = \varphi(\varphi_1(\vec{\underline{t'}}), \ldots, \varphi_m(\vec{\underline{t'}})) = \psi(\vec{\underline{t}}).$$

Also ist auch $\psi(\vec{\underline{t'}})/u' = \psi(\vec{\underline{t}})/u' = \bot$ für alle $\vec{\underline{t'}} \trianglerighteq \vec{\underline{t}}$. Somit ist jede Stelle $u \in \mathrm{Occ}(\bot, \vec{\underline{t}})$ ein Sequentialitätsindex von ψ für $\vec{\underline{t}}$ bezüglich u'.

Fall 2: Sonst.

Da φ sequentiell ist, existiert ein Sequentialitätsindex $v = i.v' \in \mathrm{Occ}(\bot, (\varphi_1(\vec{\underline{t}}), \ldots, \varphi_m(\vec{\underline{t}})))$ von φ für $(\varphi_1(\vec{\underline{t}}), \ldots, \varphi_m(\vec{\underline{t}}))$ bezüglich u' mit $i \in [m]$, d. h.:

$$\forall \vec{\underline{t''}} \trianglerighteq (\varphi_1(\vec{\underline{t}}), \ldots, \varphi_m(\vec{\underline{t}})).\ (\varphi(\vec{\underline{t''}})/u' \neq \bot \implies \underline{t''_i}/v' \neq \bot).$$

Aufgrund der Monotonie der $\varphi_1, \ldots, \varphi_m$ gilt damit insbesondere

$$\forall \vec{\underline{t'}} \trianglerighteq \vec{\underline{t}}.\ (\psi(\vec{\underline{t'}})/u' \neq \bot \implies \varphi_i(\vec{\underline{t'}})/v' \neq \bot).$$

Es ist $v' \in \mathrm{Occ}(\bot, \varphi_i(\vec{\underline{t}}))$. Da φ_i sequentiell ist, existiert ein Sequentialitätsindex $u \in \mathrm{Occ}(\bot, \vec{\underline{t}})$ von φ_i für $\vec{\underline{t}}$ bezüglich v', d. h.

$$\forall \vec{\underline{t}'} \trianglerighteq \vec{\underline{t}}.\ (\varphi_i(\vec{\underline{t}'})/v' \neq \bot \implies \vec{\underline{t}'}/u \neq \bot).$$

Insgesamt folgt dann

$$\forall \vec{\underline{t}'} \trianglerighteq \vec{\underline{t}}.\ (\psi(\vec{\underline{t}'})/u' \neq \bot \implies \vec{\underline{t}'}/u \neq \bot).$$

u ist somit ein Sequentialitätsindex von ψ für $\vec{\underline{t}}$ bezüglich u'.

□

Definition 7.2 Sequentielle ς-Interpretation
Eine ς-Interpretation $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$, deren Operationen $g^{\mathfrak{A}}$ für alle $g \in \Sigma$ sequentiell sind, heißt **sequentiell**. □

Lemma 7.3 Erhalt der Sequentialität bei erzwungener Striktheit
Sei $n \in \mathbb{N}$, $\varphi : (\mathrm{T}_{\mathcal{C},\varsigma})^n \to \mathrm{T}_{\mathcal{C},\varsigma}$ eine sequentielle Abbildung und $\vec{b} \in \mathbb{B}^n$ ein bool'scher Vektor. Sei $\psi : (\mathrm{T}_{\mathcal{C},\varsigma})^n \to \mathrm{T}_{\mathcal{C},\varsigma}$ definiert durch

$$\psi(\vec{\underline{t}}) := \begin{cases} \bot & \text{, falls ein } i \in [n] \text{ mit } \underline{t}_i = \bot \text{ und } b_i = \mathtt{tt} \text{ existiert} \\ \varphi(\vec{\underline{t}}) & \text{, anderfalls} \end{cases}$$

Dann ist ψ ebenfalls sequentiell.

Beweis:
Sei $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$.

Fall 1: Es existiert ein $i \in [n]$ mit $\underline{t}_i = \bot$ und $b_i = \mathtt{tt}$.

Also ist $\mathrm{Occ}(\bot, \vec{\underline{t}}) \neq \emptyset$. Es ist $\mathrm{Occ}(\bot, \varphi(\vec{\underline{t}})) = \mathrm{Occ}(\bot, \bot) = \{\varepsilon\}$. Da für alle $\vec{\underline{t}'} \trianglerighteq \vec{\underline{t}}$ mit $\vec{\underline{t}'}/i = \bot$ auch $\psi(\vec{\underline{t}'})/\varepsilon = \psi(\vec{\underline{t}'}) = \bot$ ist, ist $u := i$ ein Sequentialitätsindex von ψ für $\vec{\underline{t}}$ bezüglich u'.

Fall 2: Sonst.

Es ist $\psi(\vec{\underline{t}}) = \varphi(\vec{\underline{t}})$, und auch für alle $\vec{\underline{t}'} \trianglerighteq \vec{\underline{t}}$ gilt $\psi(\vec{\underline{t}'}) = \varphi(\vec{\underline{t}'})$. Da φ sequentiell für $\vec{\underline{t}}$ ist, ist somit auch ψ sequentiell für $\vec{\underline{t}}$.

□

Lemma 7.4 Sequentialität der Konstruktoroperationen
Sei $\mathfrak{A}$ eine ς-Interpretation. Dann sind die Konstruktoroperationen $G^{\mathfrak{A}}$ für alle $G \in \mathcal{C}$ sequentiell.

Beweis:
Sei $\varphi_G : (\mathrm{T}_{\mathcal{C},\varsigma})^n \to \mathrm{T}_{\mathcal{C},\varsigma}$ definiert durch

$$\varphi_G(\underline{t}_1, \ldots, \underline{t}_n) := G(\underline{t}_1, \ldots, \underline{t}_n)$$

für $G^{(n)} \in \mathcal{C}$.
φ_G ist sequentiell, denn:

Sei $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ mit $\mathrm{Occ}(\bot, \vec{\underline{t}}) \neq \emptyset$ und $u' \in \mathrm{Occ}(\bot, \varphi_G(\vec{\underline{t}})) = \mathrm{Occ}(\bot, G(\underline{t}_1, \ldots, \underline{t}_n))$. Somit ist $u' \neq \varepsilon$. Es gilt $G(\underline{t}'_1, \ldots, \underline{t}'_n)/u' = \vec{\underline{t}'}/u'$ für alle $\vec{\underline{t}'} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$. Also ist $u := u'$ ein Sequentialitätsindex von φ_G für $\vec{\underline{t}}$ bezüglich u'.
Für $G^{\mathfrak{A}}$ gilt nun

$$G^{\mathfrak{A}}(\vec{\underline{t}}) = \begin{cases} \bot & \text{, falls ein } i \in [n] \text{ mit } \underline{t}_i = \bot \text{ und } \varsigma(G)(i) = \mathtt{tt} \text{ existiert} \\ \varphi(\vec{\underline{t}}) & \text{, andernfalls} \end{cases}$$

Gemäß Lemma 7.3 über den Erhalt der Sequentialität bei erzwungener Striktheit ist $G^{\mathfrak{A}}$ sequentiell. □

Lemma 7.5 Sequentialität der überall undefinierten Abbildung
Jede überall undefinierte Abbildung $\varphi : (\mathrm{T}_{\mathcal{C},\varsigma})^n \to \mathrm{T}_{\mathcal{C},\varsigma}$, d. h. $\varphi(\vec{\underline{t}}) = \bot$ für alle $\underline{t} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ bzw. $\varphi = \bot_{\langle [(\mathrm{T}_{\mathcal{C},\varsigma})^n \to \mathrm{T}_{\mathcal{C},\varsigma}], \preceq \rangle}$, ist sequentiell.
Beweis:
Sei $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ mit $\mathrm{Occ}(\bot, \vec{\underline{t}}) \neq \emptyset$. Es gibt nur $u' := \varepsilon \in \mathrm{Occ}(\bot, \varphi(\vec{\underline{t}}))$. Da $\varphi(\vec{\underline{t}'})/\varepsilon = \bot$ für alle $\vec{\underline{t}'} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$, ist jede Stelle $u \in \mathrm{Occ}(\bot, \vec{\underline{t}})$ ein Sequentialitätsindex von φ für $\vec{\underline{t}}$ bezüglich u'. □

Korollar 7.6 Sequentialität der kleinsten ς-Interpretation
Die kleinste ς-Interpretation $\bot_\varsigma$ ist sequentiell.
Beweis:
Folgt direkt aus Lemma 7.4 und Lemma 7.5. □

Lemma 7.7 Sequentialität einer abgeleiteten Operation bezüglich einer sequentiellen ς-Interpretation
Sei $\mathfrak{A} \in \mathrm{Int}_{\Sigma,\varsigma}$ eine sequentielle ς-Interpretation, $\{x_1, \ldots, x_n\} \subseteq X$ eine endliche Variablenmenge und $t \in \mathrm{T}_\Sigma(\{x_1, \ldots, x_n\})$.
Dann ist die abgeleitete Operation $\varphi : (\mathrm{T}_{\mathcal{C},\varsigma})^n \to \mathrm{T}_{\mathcal{C},\varsigma}$, definiert durch

$$\varphi(\vec{\underline{t}}) := [\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \text{ mit } \beta(x_i) := \underline{t}_i \text{ für alle } i \in [n],$$

sequentiell.
Beweis:

$\underline{t = x_i}$: Also ist $\varphi(\vec{\underline{t}}) = \underline{t}_i$ für alle $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$. Nach Lemma 7.1 sind Projektionen sequentiell.

$\underline{t = g(t_1, \ldots, t_m)}$: $(g^{(m)} \in \Sigma,\ t_1, \ldots, t_m \in \mathrm{T}_\Sigma(\{x_1, \ldots, x_n\})$.

Es ist

$$\varphi(\vec{\underline{t}}) = [\![t]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} = g^{\mathfrak{A}}([\![t_1]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}, \ldots, [\![t_n]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}).$$

Nach Induktionsvoraussetzung sind die Abbildungen φ_i mit $\varphi_i(\vec{\underline{t}}) := [\![t_i]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta}$ für alle $i \in [m]$ sequentiell. Die Operation $g^{\mathfrak{A}}$ ist sequentiell, und da nach Lemma 7.2 auch die Komposition dieser sequentiellen Abbildungen sequentiell ist, ist φ sequentiell.

□

Lemma 7.8 Erhalt der Sequentialität bei der ς-Transformation für Programme mit Hilfsfunktionen

Sei P ein Programm mit Hilfsfunktionen und $\mathfrak{A}$ eine sequentielle ς-Interpretation.
Dann ist auch $\Phi_{P,\varsigma}(\mathfrak{A})$ eine sequentielle ς-Interpretation.

Beweis:
Sei $\mathfrak{A}' := \Phi_{P,\varsigma}(\mathfrak{A})$.

Konstruktoroperationen: Die Konstruktoroperationen $G^{\mathfrak{A}'}$ sind nach Lemma 7.4 für alle $G \in \mathcal{C}$ sequentiell.

Hilfsoperationen:

Verzweigungsoperationen: Seien φ_G die Verzweigungsoperationen nach der Transformation bei Vernachlässigung der erzwungenen Striktheit (vgl. Lemma 5.1, S. 78):

$$\varphi_G(\underline{t}_1,\underline{t}_2,\underline{t}_3) := \begin{cases} \bot & , \text{ falls } \underline{t}_1 = \bot \\ \underline{t}_2 & , \text{ falls eine Variablenbelegung } \beta : \{x_1,\ldots,x_n\} \to \mathrm{T}_{\mathcal{C},\varsigma} \\ & \quad \text{mit } [\![G(x_1,\ldots,x_n)]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta} = \underline{t}_1 \neq \bot \text{ existiert} \\ \underline{t}_3 & , \text{ andernfalls} \end{cases}$$

für alle $G \in \mathcal{C}$ und $\underline{t}_1,\underline{t}_2,\underline{t}_3 \in \mathrm{T}_{\mathcal{C},\varsigma}$.
Sei $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^3$ mit $\mathrm{Occ}(\bot,\vec{\underline{t}}) \neq \emptyset$ und $u' \in \mathrm{Occ}(\bot,\varphi_G(\vec{\underline{t}}))$.

$\vec{\underline{t}} = (\bot,\underline{t}_2,\underline{t}_3)$: Da $\varphi_G(\vec{\underline{t}}) = \bot$, muß $u' = \varepsilon$ sein. $u := 1.\varepsilon$ ist ein Sequentialitätsindex von φ für $\vec{\underline{t}}$ bezüglich u'.

$\vec{\underline{t}} = (G(\underline{t}'_1,\ldots,\underline{t}'_n),\underline{t}_2,\underline{t}_3)$: Also ist $\varphi_G(\vec{\underline{t}}) = \underline{t}_2$. Somit ist $u := 2.u'$ ein Sequentialitätsindex von φ für $\vec{\underline{t}}$ bezüglich u'.

$\vec{\underline{t}} = (H(\underline{t}'_1,\ldots,\underline{t}'_m),\underline{t}_2,\underline{t}_3)$: $(G \neq H^{(m)} \in \mathcal{C})$.
Also ist $\varphi_G(\vec{\underline{t}}) = \underline{t}_3$. Somit ist $u := 3.u'$ ein Sequentialitätsindex von φ für $\vec{\underline{t}}$ bezüglich u'.

φ_G ist also sequentiell. Aus

$$\mathrm{cond}^{\mathfrak{A}'}_G(\vec{\underline{t}}) = \begin{cases} \bot & , \text{ falls ein } i \in [3] \text{ mit } \underline{t}_i = \bot \text{ und } \varphi(\mathrm{cond}_G)(i) = \mathtt{tt} \text{ existiert} \\ \varphi_G(\vec{\underline{t}}) & , \text{ anderfalls} \end{cases}$$

für alle $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\varsigma})^3$ folgt zusammen mit dem Erhalt der Sequentialität bei erzwungener Striktheit gemäß Lemma 7.3, daß die Verzweigungsoperationen $\mathrm{cond}^{\mathfrak{A}'}_G$ sequentiell sind.

Selektionsoperationen: Es gilt (vgl. Lemma 5.1, S. 78):

$$\mathrm{sel}^{\mathfrak{A}'}_{G,i}(\underline{t}) := \begin{cases} \beta(x_i) & , \text{ falls eine Variablenbelegung } \beta : \{x_1,\ldots,x_n\} \to \mathrm{T}_{\mathcal{C},\varsigma} \\ & \quad \text{mit } [\![G(x_1,\ldots,x_n)]\!]^{\mathrm{alg}}_{\bot_\varsigma,\beta} = \underline{t}_1 \neq \bot \text{ existiert} \\ \bot & , \text{ andernfalls} \end{cases}$$

für alle $\mathrm{sel}_{G,i} \in \mathcal{H}$ und $\underline{t} \in \mathrm{T}_{\mathcal{C},\varsigma}$.
Sei $(\underline{t}) \in (\mathrm{T}_{\mathcal{C},\varsigma})^1$ mit $\mathrm{Occ}(\bot,(\underline{t})) \neq \emptyset$ und $u' \in \mathrm{Occ}(\bot,\mathrm{sel}^{\mathfrak{A}'}_{G,i}(\underline{t}))$.

$\underline{t} = G(\underline{t}'_1,\ldots,\underline{t}'_n)$: Es ist $\mathrm{sel}^{\mathfrak{A}'}_{G,i}(\underline{t}) = \underline{t}'_i$. $u := i.u'$ ist ein Sequentialitätsindex bezüglich u'.

$\underline{\underline{t} = H(\underline{t}'_1, \ldots, \underline{t}'_m)}$: $(G \neq H^{(m)} \in \mathcal{C})$.

Da $\mathrm{sel}^{\mathfrak{A}'}_{G,i}(\underline{t}) = \bot$, muß $u' = \varepsilon$ sein. Da kein $\underline{t}' \trianglerighteq \underline{t}$ mit $\mathrm{sel}^{\mathfrak{A}'}_{G,i}(\underline{t}')/u' \neq \bot$ existiert, ist jede Stelle $u \in \mathrm{Occ}(\bot, \vec{\underline{t}})$ ein Sequentialitätsindex bezüglich u'.

$\underline{\underline{t} = \bot}$: Da $\mathrm{sel}^{\mathfrak{A}'}_{G,i}(\underline{t}) = \bot$, muß auch hier $u' = \varepsilon$ sein. $u := \varepsilon$ ist ein Sequentialitätsindex bezüglich u'.

Also sind die Selektionsoperationen $\mathrm{sel}^{\mathfrak{A}'}_{G,i}$ sequentiell.

Funktionsoperationen: Sei $f^{(n)} \in \mathcal{F}$.

Fall 1: P enthält die Programmregel $f(x_1, \ldots, x_n) \to r$.

Da $\mathfrak{A}$ sequentiell ist, ist nach Lemma 7.7 auch die abgeleitete Operation $\varphi : (\mathrm{T}_{\mathcal{C},\Sigma})^n \to \mathrm{T}_{\mathcal{C},\Sigma}$, die durch

$$\varphi(\vec{\underline{t}}) := [\![r]\!]^{\mathrm{alg}}_{\mathfrak{A},\beta} \text{ mit } \beta(x_i) := \underline{t}_i \text{ für alle } i \in [n],$$

definiert ist, sequentiell. Es ist

$$f^{\mathfrak{A}'}(\vec{\underline{t}}) = \begin{cases} \bot & \text{, falls ein } i \in [n] \text{ mit } \underline{t}_i = \bot \text{ und } \varsigma(f)(i) = \mathtt{tt} \text{ existiert} \\ \varphi(\vec{\underline{t}}) & \text{, andernfalls} \end{cases}$$

und somit ist $f^{\mathfrak{A}'}$ nach Lemma 7.3 über den Erhalt der Sequentialität bei erzwungener Striktheit sequentiell.

Fall 2: Sonst.

$f^{\mathfrak{A}'}$ ist überall undefiniert und somit nach Lemma 7.5 sequentiell.

□

Lemma 7.9 **ω-Stetigkeit der Sequentialität**

1. Sei $(\varphi_i)_{i \in \mathbb{N}}$ mit $\varphi_i : (\mathrm{T}_{\mathcal{C},\Sigma})^n \to \mathrm{T}_{\mathcal{C},\Sigma}$ eine ω-Kette von sequentiellen ω-stetigen Abbildungen. Dann ist $\varphi := \bigsqcup_{i \in \mathbb{N}} \varphi_i$ sequentiell.
2. Sei $(\mathfrak{A}_i)_{i \in \mathbb{N}}$ mit $\mathfrak{A}_i \in \mathrm{Int}_{\Sigma,\Sigma}$ eine ω-Kette von sequentiellen ς-Interpretationen. Dann ist $\mathfrak{A} := \bigsqcup_{i \in \mathbb{N}} \mathfrak{A}_i$ sequentiell.

Beweis:

1. Sei $\vec{\underline{t}} \in (\mathrm{T}_{\mathcal{C},\Sigma})^n$ mit $\mathrm{Occ}(\bot, \vec{\underline{t}}) \neq \emptyset$. Sei $u' \in \mathrm{Occ}(\bot, \varphi(\vec{\underline{t}})$. Für alle $i \in \mathbb{N}$ gilt entweder $u' \notin \mathrm{Occ}(\varphi_i(\vec{\underline{t}}))$ oder $\varphi_i(\vec{\underline{t}})/u' = \bot$, weil $\varphi(\vec{\underline{t}}) \trianglerighteq \varphi_i(\vec{\underline{t}})$ ist. Da $\varphi(\vec{\underline{t}}) = \bigsqcup_{i \in \mathbb{N}} \varphi_i(\vec{\underline{t}})$, und somit insbesondere $\mathrm{Occ}(\varphi(\vec{\underline{t}})) = \bigcup_{i \in \mathbb{N}} \mathrm{Occ}(\varphi_i(\vec{\underline{t}}))$, existiert auch ein $i' \in \mathbb{N}$ mit $\varphi_{i'}(\vec{\underline{t}})/u' = \bot$. Sei j die kleinste natürliche Zahl mit $\varphi_j(\vec{\underline{t}})/u' = \bot$. Aufgrund der Ordnung der Abbildungen ist dann auch $\varphi_i(\vec{\underline{t}})/u' = \bot$ für alle $i \geq j$.

   Für alle $i \geq j$ seien $U_i \subseteq \mathrm{Occ}(\bot, \vec{\underline{t}})$ die Menge der Sequentialitätsindexe von φ_i für $\vec{\underline{t}}$ bezüglich u', d. h.

$$U_i := \{u_i \in \mathrm{Occ}(\bot, \vec{\underline{t}}) \mid \forall \vec{\underline{t'}} \trianglerighteq \vec{\underline{t}}.\ (\varphi_i(\vec{\underline{t'}})/u' \neq \bot \implies \vec{\underline{t'}}/u_i \neq \bot).$$

Da die Abbildungen φ_i sequentiell sind, sind die Mengen $U_i \neq \emptyset$.

Aufgrund der Ordnung der Abbildungen gilt für alle $k \geq i \geq j$ und $\vec{\underline{t}'} \trianglerighteq \vec{\underline{t}}$

$$\varphi_i(\vec{\underline{t}'})/u' \neq \bot \implies \varphi_k(\vec{\underline{t}'})/u' \neq \bot.$$

Dies bedeutet, daß für alle $i \geq j$ ein Sequentialitätsindex u_{i+1} von φ_{i+1} für $\vec{\underline{t}}$ bezüglich u' auch ein Sequentialitätsindex von φ_i für $\vec{\underline{t}}$ bezüglich u' ist, kurz:

$$U_j \supseteq U_{j+1} \supseteq U_{j+2} \supseteq \ldots$$

Somit ist $U := \bigcap_{i \geq j} U_i \neq \emptyset$.

φ ist sequentiell, weil U die Menge der Sequentialitätsindexe von φ für $\vec{\underline{t}}$ bezüglich u' ist, denn:

Sei $u \in U$. Dann gilt für alle $i \geq j$

$$\forall \vec{\underline{t}'} \trianglerighteq \vec{\underline{t}}.\ (\varphi_i(\vec{\underline{t}'})/u' \neq \bot \implies \vec{\underline{t}'}/u \neq \bot),$$

da $u \in U_i$. Sei nun $\vec{\underline{t}'} \trianglerighteq \vec{\underline{t}}$ mit $\varphi(\vec{\underline{t}'})/u' \neq \bot$. Da $\varphi(\vec{\underline{t}'}) = \bigsqcup_{i \in \mathbb{N}} \varphi_i(\vec{\underline{t}'})$, muß ein $i \geq j$ mit $\varphi_i(\vec{\underline{t}})/u' \neq \bot$ existieren. Damit ist dann auch $\vec{\underline{t}'}/u \neq \bot$.

2. Folgt direkt aus 1.

□

Lemma 7.10 Sequentialität des ς-Datentyps eines Programms mit Hilfsfunktionen
Sei P ein Programm mit Hilfsfunktionen. Dann ist der ς-Datentyp $\mathcal{D}_{P,\varsigma}$ sequentiell.

Beweis:
Es ist

$$\mathcal{D}_{P,\varsigma} = \mathcal{D}_{P,\varsigma}^{\text{fix}} = \bigsqcup_{i \in \mathbb{N}} (\Phi_{P,\varsigma})^i(\bot_\varsigma).$$

Nach Korollar 7.6 ist die kleinste ς-Interpretation $\bot_\varsigma$ sequentiell. Mit Lemma 7.8 über den Erhalt der Sequentialität bei der ς-Transformation für Programme mit Hilfsfunktionen folgt die Sequentialität der ς-Interpretationen $\mathfrak{A}_i := (\Phi_{P,\varsigma})^i(\bot_\varsigma)$ für alle $i \in \mathbb{N}$. Aufgrund der Monotonie der ς-Transformation gemäß Lemma 5.20, S. 95, bilden die $\mathfrak{A}_i$ eine ω-Kette, und mit Lemma 7.9 folgt schließlich, daß die kleinste obere Schranke dieser ω-Kette, $\mathcal{D}_{P,\varsigma}$, sequentiell ist. □

Hiermit haben wir bewiesen, daß Programme mit Pattern echt ausdrucksstärker als Programme mit Hilfsfunktionen sind. Es ist zu beachten, daß dies kein Widerspruch zu den Aussagen in 7.1 über die Berechnungsstärke der Programme ist, da dort die spezifizierten Operationen nur über den Konstruktorgrundtermen $\mathrm{T}_\mathcal{C}$ betrachtet wurden, während die Sequentialität die gesamte Halbordnung des Rechenbereichs, $\langle \mathrm{T}_{\mathcal{C},\varsigma}, \trianglelefteq \rangle$, mit einbezieht.

7.4 Bemerkungen zur Sequentialität

Die Sequentialität ist eine zwar komplexe, aber auch sehr interessante Eigenschaft von Abbildungen. So steht sie in engem Zusammenhang mit der Striktheit und kann als Verallgemeinerung dieser aufgefaßt werden.

Lemma 7.11 Sequentialität und Striktheit
Sei $\varphi : (\mathrm{T}_{\mathcal{C},\varsigma})^n \to \mathrm{T}_{\mathcal{C},\varsigma}$ mit $n \in \mathbb{N}$ eine monotone Abbildung und $i \in [n]$.
φ ist genau dann an der i-ten Argumentstelle strikt, wenn i ein Sequentialitätsindex von φ für $(\bot, \bot, \ldots, \bot) \in (\mathrm{T}_{\mathcal{C},\varsigma})^n$ bezüglich ε ist.

Beweis:

φ ist an der i-ten Argumentstelle strikt.

$\iff \forall \vec{\underline{t}} \in \mathrm{T}_{\mathcal{C},\varsigma}.\ (\underline{t}_i = \bot \implies \varphi(\vec{\underline{t}}) = \bot)$

$\iff \forall \vec{\underline{t}} \in \mathrm{T}_{\mathcal{C},\varsigma}.\ (\varphi(\vec{\underline{t}}) \neq \bot \implies \underline{t}_i \neq \bot)$

$\iff \forall \vec{\underline{t}} \trianglerighteq (\bot, \bot, \ldots, \bot).\ (\varphi(\vec{\underline{t}})/\varepsilon \neq \bot \implies \vec{\underline{t}}/i \neq \bot)$

$\iff i$ ist Sequentialitätsindex von φ für $(\bot, \bot, \ldots, \bot)$ bezüglich ε.

□

Hieraus folgt unmittelbar die Sequentialität aller Operationen in der cbv-Semantik, auch der durch Programme mit Pattern spezifizierten. Dies ist jedoch nicht überraschend, da wir erwähnten, daß für die cbv-Semantik Programme mit Pattern sehr wohl semantikerhaltend in Programme mit Hilfsfunktionen übersetzbar sind.
Da Striktheit bekanntlich eine unentscheidbare Eigenschaft ist, weist der Zusammenhang zwischen Sequentialität und Striktheit schon darauf hin, daß es auch unentscheidbar ist, ob die durch ein Programm mit Pattern spezifizierten Operationen sequentiell sind.

Wir hätten die größere Ausdrucksstärke der Programme mit Pattern durchaus mit Hilfe einer einfacheren Eigenschaft als der Sequentialität nachweisen können. Die in [Berry78] definierte sogenannte **Stability-Eigenschaft** von Abbildungen hätte denselben Zweck erfüllt. Alle durch Programme mit Hilfsfunktionen spezifizierbaren Operationen sind stable, wogegen durch Programme mit Pattern auch Operationen spezifizierbar sind, die nicht stable sind.
Wir haben trotzdem die Sequentialität verwendet, da diese die intuitive Sequentialität der durch Programme mit Hilfsfunktionen spezifizierten Operationen vollständig verkörpert. Es existieren nämlich durch Programme mit Pattern spezifizierbare Operationen, die stable und dennoch nicht durch Programme mit Hilfsfunktionen spezifizierbar sind (vgl. 3.4 in [Ber&Cur82]). Hingegen sind wir überzeugt, daß alle sequentiellen Operationen, die durch Programme mit Pattern spezifizierbar sind, auch durch Programme mit Hilfsfunktionen spezifizierbar sind. Somit ist die Sequentialität genau das Scheidekriterium zwischen den durch beliebige Programme und den nur durch Programme mit Pattern spezifizierbaren Operationen.

Wir wollen die Richtigkeit dieser Aussage zumindest plausibel machen. Sind alle durch ein Programm mit Pattern spezifizierten Operationen sequentiell, so ließe sich das Programm semantikerhaltend in ein Programm mit Hilfsfunktionen übersetzen, indem die Teste der Argumente mit cond_G nur an Sequentialitätsindexen erfolgen würden. Die Sequentialität besagt gerade, daß die Information an den Sequentialitätsindexen benötigt wird, um den Informationsgehalt des Ergebnisses steigern zu können. Freilich wäre diese Übersetzung nur dann effektiv durchführbar, wenn nicht nur die Gegebenheit der Sequentialität, sondern auch die Sequentialitätsindexe selbst bekannt wären.

Beispiel 7.8 **Übersetzung eines sequentiellen Programms mit Pattern**

```
f(x,A,C)  →  A
f(x,A,B)  →  B
f(A,B,x)  →  C
```

2 ist Sequentialitätsindex von $\mathtt{f}^{\mathcal{D}_{P,\varsigma}}$ für $(\bot,\bot,\bot)$ bezüglich ε.
3 ist Sequentialitätsindex von $\mathtt{f}^{\mathcal{D}_{P,\varsigma}}$ für $(\bot,\mathtt{A},\bot)$ bezüglich ε.
Semantikerhaltende Übersetzung:

```
f(x,y,z)  →  cond_B(y,cond_A(x,C,f(x,y,z)),
               cond_A(y,
                 cond_B(z,B,
                     cond_C(z,A,f(x,y,z))
               ),f(x,y,z)))
```

□

Von hier aus ist es nur ein kleiner Schritt zu effizienten ς-Reduktionsstrategien für unsere Programme. Wären die Sequentialitätsindexe der Operationen bekannt, so würde eine effiziente ς-Reduktionsstrategie nur an den outermost ς-Redexstellen reduzieren, die auch Sequentialitätsindexe wären, da nur diese Reduktionen Informationsgewinn versprechen würden. Mit Hilfe der Sequentialitätsindexe ließen sich also genau die in 5.5.4 angesprochenen gaining ς-Redexstellen, d. h. $\mathrm{Gain}(t)$ und $\mathrm{NGain}(t)$ für alle Terme $t \in \mathrm{T}_\Sigma$, definieren. Die Sequentialität würde den Informationsgewinn bei Reduktion von gaining ς-Redexstellen analog zu Lemma 5.23, S. 105, sicherstellen, und mit einer eventually gaining Hilfskonstruktion ließe sich dann die Vollständigkeit der eventually gaining ς-Reduktionssemantik beweisen.
Diese Methode wäre durchaus nicht auf sequentielle Operationen beschränkt. Nicht sequentielle besitzen zwar nicht für alle Argumente Sequentialitätsindexe, aber es ließe sich immerhin eine Art Sequentialitätsmenge von Indexen definieren, d. h. eine minimale Menge von „$\bot$-Stellen“, die allerhöchstens bei Reduktionen betrachtet werden müßten.

Insbesondere in Anbetracht der Unentscheidbarkeit der Sequentialität der Programme mit Pattern bleibt freilich noch das Problem der Bestimmung der Sequenitalitätsindexe bzw. dieser Sequentialitätsmengen. Durch eine Analyse der Pattern eines Operationssymbols lassen sich jedoch schon viele Sequentialitätsindexe der Operation bestimmen. In Beispiel 7.8 ergeben sich die benötigten Sequentialitätsindexe 2 und 3 direkt aus den Pattern. Bei dieser Analyse der Pattern ließe sich auch die erzwungene Striktheit ς sinnvoll einbeziehen. Die Programme der Beispiele 7.3 und 7.4 für das parallele And und Berrys Beispiel spezifizieren durchaus sequentielle Abbildungen, wenn `and` und `f` an allen Argumentstellen erzwungen strikt sind. Mit Approximationsverfahren wie der abstrakten Interpretation (siehe S. 59) ließen sich weitere Sequentialitätsindexe bestimmen oder Sequentialitätsmengen eingrenzen.
Andererseits bestände auch die Möglichkeit, die Menge der zulässigen Pattern einzuschränken, und auf diese Weise die Sequentialität und die Berechenbarkeit der Sequentialitätsindexe zu erzwingen.

Auch in der Literatur werden häufig Sequentialitätseigenschaften in Zusammenhang mit effizienten operationellen Semantiken betrachtet.
In [Huet&Lévy79] wird bewiesen, daß bei orthogonalen Termersetzungssystemen jeder Term eine Stelle besitzt, an der reduziert werden muß (needed), damit die Normalform des Terms erreicht

werden kann (vorausgesetzt sie existiert und ist noch nicht erreicht). Somit existiert für orthogonale Termersetzungssysteme eine optimale sequentielle normalisierende Reduktionsstrategie. Leider ist dies von keinem praktischen Nutzen, da diese needed Redexstelle nicht berechenbar ist. Für eine Teilmenge der orthogonalen Termersetzungssystem, deren linke Regelseiten die sogenannte Strong-Sequentiality-Eigenschaft besitzen, ist diese Redexstelle jedoch (sogar effizient) berechenbar. [Tha87], [Klop&Midd91] und [Kes92] bauen auf dieser fundamentalen Arbeit auf.
Bemerkenswert ist, daß diese Aussage für alle orthogonalen Termersetzungssysteme und somit für alle Programme ohne überlappende linke Regelseiten gilt.
In [An&Mid94] wird sogar eine berechenbare, sequentielle, normalisierende Reduktionsstrategie für beliebige beinahe orthogonale Termersetzungssysteme angegeben. Dies mag zuerst unmöglich erscheinen, da eine schlichte Sequentialisierung der po-Reduktionsstrategie, d. h. ein po-Reduktionsschritt wird in einzelne elementare Reduktionsschritte zerlegt, im allgemeinen nicht zu einer sequentiellen Reduktionsstrategie führt: In der Folge der elementaren Reduktionen können Zyklen erscheinen, die in der po-Reduktionsfolge nicht existieren; beispielsweise dadurch, daß eine elementare Reduktion einen Term unverändert läßt. Da eine Reduktionsstrategie deterministisch ist, d. h. jedem Term (höchstens) einen Nachfolgerterm zuordnet, könnte sie niemals aus einem solchen Zyklus „ausbrechen“. Die in [An&Mid94] angewandte Technik zur Umgehung dieses Zyklusproblems ist jedoch sehr aufwendig, so daß diese sequentielle Reduktionsstrategie nicht effizient ist. Sie ist auch keinesfalls optimal, da sie im allgemeinen auch Reduktionen an Redexstellen durchführt, die sich im Nachhinein als unnötig erweisen. Da jedoch beinahe orthogonale Termersetzungssysteme auch keine needed Redexstellen wie die orthogonalen Termersetzungssyteme besitzen, ist dies unvermeidlich.
In [Lav87] wird ein Patternmatchingalgorithmus für reale funktionale Programmiersprachen für eine eingeschränkte Menge zulässiger Pattern definiert. Zwar werden hierbei nur die linken Regelseiten in einer cbn-Semantik betrachtet, aber die mögliche Ausweitung der zulässigen Pattern bei Einbeziehung von Ergebnissen einer Striktheitsanalyse wird angedeutet.

In der Literatur wird zur Gewinnung effizienter operationeller Semantiken die Sequentialität als rein syntaktische Eigenschaft betrachtet. Unsere hier angedeutete Methode beruht dagegen auf der Sequentiatität als semantische Eigenschaft von Abbildungen. Insbesondere werden die Gewinnung der Informationen über die Sequentialität (Sequentialitätsindexe bzw. -mengen) und die Nutzung dieser in einer eventually gaining ς-Reduktionssemantik sauber voneinander getrennt.

Kapitel 8

Nicht-freie Datentypen

Wir haben in den bisherigen Kapiteln die Syntax und insbesondere die Semantik einfacher konstruktorbasierter funktionaler Programmiersprachen definiert und untersucht. Mit diesen Programmiersprachen sind jedoch nur **freie (konstruktorbasierte) Datentypen** spezifizierbar, bzw. unsere ς-Datentypen sind nur Erweiterungen freier (konstruktorbasierter) Basisdatentypen. Freiheit bedeutet hierbei, daß jeder syntaktische Konstruktorgrundterm ein eindeutiger Bezeichner eines semantischen Datenelements ist[1], formal:

$$\forall t, t' \in \mathrm{T}_{\mathcal{C}}.\ t \neq t' \implies [\![t]\!]_{P,\varsigma} \neq [\![t']\!]_{P,\varsigma}.$$

Wie wir in Beispielen gesehen haben, sind damit durchaus viele wichtige Datentypen leicht und auf natürliche Weise spezifizierbar.

Beispiel 8.1 **Freie Datentypen**

Wahrheitswerte (Boolean):	$\mathcal{C} = \{\mathtt{False}^{(0)}, \mathtt{True}^{(0)}\}$
Zeichen (Character):	$\mathcal{C} = \{'\mathtt{A}'^{(0)}, '\mathtt{B}'^{(0)}, '\mathtt{C}'^{(0)}, \ldots\}$
Natürliche Zahlen:	$\mathcal{C} = \{\mathtt{Zero}^{(0)}, \mathtt{Succ}^{(1)}\}$
Listen:	$\mathcal{C} = \{\mathtt{Nil}^{(0)}, \mathtt{Cons}^{(2)}\}$
Bäume:	$\mathcal{C} = \{\mathtt{Null}^{(0)}, \mathtt{Node}^{(3)}\}$

□

Für andere Datentypen wie ganze Zahlen, rationale Zahlen, Fließkommazahlen, geordnete Listen und Mengen ist dies dagegen nicht möglich.
Natürlich lassen sich diese Datentypen durchaus durch Konstruktoren aufbauen. So sind zum Beispiel die ganzen Zahlen durch $\mathcal{C} = \{\mathtt{Zero}^{(0)}, \mathtt{Succ}^{(1)}, \mathtt{Pred}^{(1)}\}$ erzeugbar. `Pred(`t`)` bezeichnet den Vorgänger der durch t bezeichneten Zahl, und wir erhalten

$$\ldots,\ \mathtt{Pred(Zero)} \simeq -1,\ \ \mathtt{Zero} \simeq 0,\ \ \mathtt{Succ(Zero)} \simeq 1,\ \ \mathtt{Succ(Succ(Zero))} \simeq 2,\ \ \ldots$$

Aber $\mathrm{T}_{\mathcal{C}}$ enthält auch $\mathtt{Succ(Pred(Zero))}, \mathtt{Pred(Succ(Zero))}, \mathtt{Succ(Succ(Pred(Pred(Zero))))}, \ldots$.
Offensichtlich sollten all dieses letzteren Terme ebenfalls die Zahl 0 bezeichnen. In den ς-Datentypen

[1] Im algebraischen Sinne sind die Basisdatentypen, die aus dem Träger der ς-Datentypen, $\mathrm{T}_{\mathcal{C},\varsigma}$, und den Konstruktoroperationen bestehen (wir ignorieren die Hilfsoperationen), natürlich nicht absolut frei relativ zu $\{\bot\}$. Immerhin ist jedoch die Konstruktorgrundtermalgebra $\mathcal{T}_{\mathcal{C}}$ absolut initial, und $\mathcal{T}^{\infty}_{\mathcal{C},\bot}$ ist initial in $\mathrm{Alg}^{\infty}_{\mathcal{C},\bot}$.

$\mathcal{D}_{P,\varsigma}$ bzw. deren Basisdatentypen zu der Konstruktorsignatur $\mathcal{C} = \{\texttt{Zero}^{(0)}, \texttt{Succ}^{(1)}, \texttt{Pred}^{(1)}\}$ kommt dies überhaupt nicht zum Ausdruck, während **nicht-freie Datentypen** gerade dies ermöglichen (sollen).

Derartige nicht-freie Datentypen betrachten wir in diesem Kapitel. Da wir dieses komplexe Gebiet nur kurz anreißen wollen, verzichten wir ganz auf formale Definitionen und beschäftigen uns mit den neu auftretenden Phänomenen in allgemeiner und intuitiver Weise. Auf die Darstellung der speziell durch partielle und unendliche Datenstrukturen verursachten Probleme verzichten wir ganz, und wir betrachten auch keine Hilfsfunktionen, sondern nur Abwandlungen unserer Programme mit Pattern.

8.1 Datenregeln

Es liegt nahe, die semantische Gleichheit unterschiedlicher Konstruktorgrundterme durch Gleichungen von Konstruktortermen zu spezifizieren. Der Basisdatentyp ist dann die Quotientenalgebra der Konstruktorgrundtermalgebra modulo der durch diese Gleichungen bestimmten Kongruenz. Da sich, wie wir schon in 7.1 bezüglich des initialen Modells eines Programms anmerkten, nicht auf beliebigen Äquivalenzklassen rechnen läßt, verwenden wir anstelle von Gleichungen ein konfluentes, terminierendes Termersetzungssystem. Damit sind dann die Konstruktornormalformen die eindeutigen Repräsentanten der Datenelemente. Die Konstruktorsymbole zusammen mit den Regeln dieses Termersetzungssystems, den **Datenregeln**, spezifizieren den nicht-freien Basisdatentyp. Ein einen nicht-freien Basisdatentyp spezifizierendes Programm besteht damit aus Datenregeln und Programmregeln.

Beispiel 8.2 Datenregeln für ganze Zahlen
Datenregeln:

```
Succ(Pred(x))  →  x
Pred(Succ(x))  →  x
```

Programmregeln für die Addition:

```
add(x,Zero)      →  x
add(x,Succ(y))   →  Succ(add(x,y))
add(x,Pred(y))   →  Pred(add(x,y))
```

□

Diese Spezifikationsmethode bringt aber mehrere Probleme mit sich.
Erstens ist es nicht entscheidbar, ob die Datenregeln tatsächlich ein konfluentes und terminierendes Termersetzungssystem bilden. Wir haben jedoch gefordert, daß die syntaktische Korrektheit eines Programms entscheidbar sein und jedem syntaktisch korrektem Programm eine Semantik zugeordnet werden soll.
Das folgende Beispiel verdeutlicht ein zweites, fundamentales Problem.

Beispiel 8.3 Datenregeln und Test auf 0 für ganze Zahlen, I
Die Datenregeln seien die gleichen wie in Beispiel 8.2.
Programmregeln:

```
isZero(Zero)      →  Succ(Zero)
isZero(Succ(x))   →  Zero
isZero(Pred(x))   →  Zero
```

□

Das obige Programm spezifiziert keinesfalls den intendierten Test auf 0. Die Funktion `isZero` ist noch nicht einmal eine Funktion über den ganzen Zahlen! Aus den Regeln folgt beispielsweise

$$\texttt{Succ(Zero)} \sim \texttt{isZero(Zero)} \sim \texttt{isZero(Succ(Pred(Zero)))} \sim \texttt{Zero},$$

und allgemein, daß alle Konstruktorgrundterme semantisch gleich sind. Das Programm ist nicht **konsistent**, d. h. durch die Programmregeln werden Konstruktorterme semantisch gleichgesetzt, die in dem durch die Datenregeln spezifizierten Basisdatentypen nicht semantisch gleich sind. Die Forderung der Konsistenz ist auf dem Gebiet der algebraischen Spezifikationen wohlbekannt ([Gut77], [Gut&Hor78], 3.2 in [Der&Jou90], [Wir90]), aber diese Eigenschaft ist für allgemeine Termersetzungssysteme unentscheidbar (vgl. [Gut&Hor78]).
Sinnvoll, je nach Form der zugelassenen Datenregeln allerdings noch nicht hinreichend für die Konsistenz, ist mit Sicherheit eine Anpassung der Eindeutigkeitsbedingung der Programmregeln (vgl. Def. 3.2, S. 38): Wenn zwei linke Programmregelseiten bezüglich der durch die Datenregeln erzeugten Kongruenz unifizierbar sind, sollten auch die zugehörigen rechten Programmregelseiten bezüglich dieser Kongruenz unifizierbar sein (für Unifikation bezüglich einer Kongruenz siehe [Der&Jou90], [Jou&Kir91]). Das obige Beispiel 8.3 erfüllt diese Bedingung nicht. Die Unifizierbarkeit ist jedoch selbst für durch konfluente, terminierende Termersetzungssysteme gegebene Kongruenzen nur semientscheidbar (das Problem ist leicht auf Hilberts zehntes Problem der Lösbarkeit diophantischer Gleichungen reduzierbar; siehe 6.3 in [Der&Jou90], auch 6. in [Huet&Op80]).
Für unsere Programme mit Pattern und Programme mit Hilfsfunktionen folgt die Konsistenz übrigens direkt aus der Konfluenz. Diese ist bei der Verwendung von Datenregeln aber auch im allgemeinen nicht mehr gegeben, so auch in Beispiel 8.3 nicht, obwohl die Datenregeln und die Programmregeln jeweils für sich konfluent sind. Da die Konfluenz beliebiger Termersetzungssysteme unentscheidbar ist, sie aber für operationelle Semantiken im allgemeinen benötigt wird, stellt dies ein weiteres Problem dar.
Nach all diesen, mehr theoretische Aspekte betreffenden Nachteilen bringen wir schließlich noch einen ganz pragmatischen Einwand vor: Das folgende Beispiel zeigt eine korrekte, konsistente Spezifikation des Tests auf 0.

Beispiel 8.4 Datenregeln und Test auf 0 für ganz Zahlen, II
Die Datenregeln seien die gleichen wie in Beispiel 8.2.

```
isZero(x)                 →  h(Zero,Zero,x)
h(x,y,Pred(z))            →  h(x,Succ(y),z)
h(x,y,Succ(z))            →  h(Succ(x),y,z)
h(Succ(x),Succ(y),Zero)   →  h(x,y,Zero)
h(Succ(x),Zero,Zero)      →  Zero
h(Zero,Succ(x),Zero)      →  Zero
h(Zero,Zero,Zero)         →  Succ(Zero)
```

Prinzip des Algorithmus: `h` zählt die im dritten Argument auftretenden `Succ`'s und `Pred`'s im ersten respektive zweiten Argument. □

Zur Wahrung der Konsistenz ist diese äußerst umständliche und komplizierte Spezifikation nötig! Die Funktionen müssen beliebige Konstruktorterme als Argumente „korrekt verarbeiten“. Die dadurch gegebene korrekte Spezifikation des nicht-freien Datentyps ist für die Praxis jedoch nur von geringem Interesse.

Der Vorteil der Datenregeln sollte vielmehr darin bestehen, daß Funktionen nur noch mit Argumenten in Normalform „rechnen“ müßten, und spezielle Eigenschaften dieser Normalform „ausnützen“ könnten. So sollte der Test auf 0 wie in Beispiel 8.3 spezifizierbar sein, und in einer Spezifikation der rationalen Zahlen sollten Funktionen davon „ausgehen“ können, daß Zähler und Nenner der als Argumente übergebenen rationalen Zahlen keine gemeinsamen Faktoren mehr besitzen. Dagegen müßten die Funktionen selber nicht den Erhalt der Normalformeigenschaft sicherstellen; dafür wären die Datenregeln zuständig (siehe die Einleitung von [Tho90] bezüglich dieser Aufgabenteilung). Auf diese Weise würden nicht-freie Datentypen wirklich zu eleganteren und unter Umständen auch effizienteren Programmen führen.

8.2 Laws in Miranda

Genau dies ermöglichen die in früheren Versionen von Miranda vorhandenen **Laws** ([MirMan89], [Tho86], [Tho90]).

Beispiel 8.5 **Laws und Test auf 0 für ganze Zahlen in Miranda**

```
integer  ::=  Zero | Succ(integer) | Pred(integer)

              Succ(Pred(x))  =>  x
              Pred(Succ(x))  =>  x

            isZero(Zero)     =  Succ(Zero)
            isZero(Succ(x))  =  Zero
            isZero(Pred(x))  =  Zero
```

□

Abgesehen von syntaktischen Unterschieden[2] entspricht dieses Miranda-Programm genau dem Programm mit Datenregeln in Beispiel 8.3. Dennoch ist das Programm sehr wohl konsistent. Eine spezielle Reduktionsstrategie, eine Mischung aus leftmost-outermost und leftmost-innermost, die in [Tho86] lazy innermost genannt wird, garantiert, daß die Argumente einer Funktion vor dem Funktionsaufruf soweit ausgewertet sind, wie zum beabsichtigen Patternmatching erforderlich ist. Daher ist diese Reduktionsstrategie jedoch äußerst komplex und die zugehörige denotationelle Semantik ebenso. Insbesondere sind die die Funktion definierenden Gleichungen (nicht nur aufgrund des in 7.2 betrachteten, speziellen Patternmatchings) keinesfalls im Datentyp allgemein gültig, und Schlußfolgerungen wie die schon bei Beispiel 8.3 gemachte

$$\texttt{Succ(Zero)} \sim \texttt{isZero(Zero)} \sim \texttt{isZero(Succ(Pred(Zero)))} \sim \texttt{Zero},$$

sind somit falsch (dies ist für die Konsistenz selbstverständlich notwendig), obwohl sie leider auch sehr naheliegend sind. Diese kontraintuitive Semantik erschwert daher das Verständnis und auch die Verifikation von algebraischen Datentypen mit Laws (siehe [Tho86], [Tho90]). Deshalb wurde der Law-Mechanismus wieder aus Miranda entfernt.

[2] Bezüglich der Schreibweise der Miranda-Programme beachte man die Fußnote auf Seite 39.

8.3 Konstruktorfunktionen

Für die Erklärung des Law-Mechanismus und die Verifikation von algebraischen Datentypen mit Laws wird in [Tho86] und [Tho90] zwischen den Konstruktorsymbolen, die auf linken, und denen, die auf rechten Gleichungsseiten stehen, unterschieden. In [Bur&Cam93] werden anstelle dieser Unterscheidung in der Metasprache sogar zwei verschiedene Sätze von Operationssymbolen verwendet. Dies gibt die Anregung zu einer einfachen Methode, die die adäquate Darstellung nicht-freier Datentypen auch mit unseren Programmen zusammen mit den ς-Semantiken ermöglicht (vgl. auch 32.4 in [MirMan89]).

Beispiel 8.6 **Konstruktorfunktionen, Test auf 0 und Addition für ganze Zahlen**

```
        zero  →  Zero
   succ(Zero) →  Succ(Zero)
succ(Succ(x)) →  Succ(Succ(x))
succ(Pred(x)) →  x
   pred(Zero) →  Pred(Zero)
pred(Succ(x)) →  x
pred(Pred(x)) →  Pred(Pred(x))
```

```
   isZero(Zero) →  succ(zero)
isZero(Succ(x)) →  zero
isZero(Pred(x)) →  zero
    add(x,Zero) →  zero
 add(x,Succ(y)) →  succ(add(x,y))
 add(x,Pred(y)) →  pred(add(x,y))
```

□

Zu jedem Konstruktorsymbol existiert ein Funktionssymbol, das wir **Konstruktorfunktionssymbol** nennen. Die Konstruktorsymbole werden nur in den Pattern und in den rechten Seiten der Regeln der Konstruktorfunktionssymbole verwendet. Diese speziellen Regeln bestimmen den nicht-freien Datentyp. Sie sind übrigens leicht aus den Datenregeln bzw. den Laws von Miranda automatisch generierbar (vgl. 32.4 in [MirMan89]). Auf den rechten Seiten der Regeln der „normalen“ Funktionssymbole müssen anstelle der Konstruktorsymbole die Konstruktorfunktionssymbole verwendet werden. Auf diese Weise wird erreicht, daß die Funktionsoperationen immer Elemente des nicht-freien Datentyps als Ergebnis lieferen, wenn auch die übergebenen Argumente Elemente des nicht-freien Datentyps sind.
Natürlich stellt die Verwendung von Konstruktorfunktionen nur einen Programmierstil dar. Die ς-Datentypen sind und bleiben frei. Aber die Definition einer neuen, komplexen Semantik wird auf diese Weise überflüssig, und wir erhalten trotzdem die praktischen Vorteile nicht-freier Datentypen. Der nicht-freie Datentyp wäre sogar noch als die durch die Konstruktorfunktionsoperationen erzeugte Unteralgebra der ς-Datentypen definierbar.
Mit dieser Betonung der Praxis unter Vernachlässigung der semantischen Spezifizierung der nicht-freien Datentypen steht das Konzept der Konstruktorfunktionen in einem gewissen Gegensatz zu dem der Datenregeln.

8.4 Weitere Beobachtungen

Die Laws von Miranda und die Konstruktorfunktionen sind beide ausdrucksstärker als die Datenregeln oder auch beliebige Konstruktorgleichungen. Geordnete Listen sind beispielsweise nicht durch terminierende Datenregeln spezifizierbar. Hierfür werden schon bedingte Termersetzungssysteme ([Der&Jou90], [Ber&Klop86]) benötigt, wie sie die Laws im Prinzip darstellen.

Beispiel 8.7 **Laws und geordnete Listen** (S. 187 in [Tho90])

```
orderedList          ::=  ONil | OCons(num,orderedList)

OCons(x,OCons(y,z))  =>   OCons(y,OCons(x,z)), if y < x
```

□

Da wir unsere Programme mit Pattern ohne Sorten definiert haben, können wir das entsprechende Programm, welches Konstruktorfunktionen verwendet, nur skizzieren.

Beispiel 8.8 **Konstruktorfunktionen und geordnete Listen**

```
oNil                 →  ONil
oCons(x,ONil)        →  OCons(x,ONil)
oCons(x,OCons(y,z))  →  if y < x
                          then OCons(y,oCons(x,z))
                          else OCons(x,OCons(y,z))
                        fi
```

□

Für die Spezifikation der rationalen Zahlen wird eine Funktion zur Berechnung des größten gemeinsamen Teilers benötigt. Da Datenregeln und Konstruktorgleichungen jedoch Paare von Konstruktortermen sind, ist die Spezifikation rationaler Zahlen durch sie unmöglich.

Beispiel 8.9 **Laws und rationale Zahlen** (S.188 in [Tho90])

```
rational  ::=  Rat(num,num)

Rat(x,y)  =>   error("zero denominator"), if y = 0
          =>   Rat(-x,-y), if y < 0
          =>   Rat(x',y'), if g > 1
               where
               x' = x div g
               y' = y div g
               g = gcd x y
```

□

Während bei freien Datentypen die Wahl geeigneter Konstruktorsymbole meistens einfach ist, bieten sich bei nicht-freien Datentypen oft unterschiedliche Möglichkeiten an. So können die ganzen Zahlen nicht nur durch $\mathcal{C} = \{\texttt{Zero}^{(0)}, \texttt{Succ}^{(1)}, \texttt{Pred}^{(1)}\}$, sondern auch durch $\mathcal{C} = \{\texttt{Zero}^{(0)}, \texttt{Succ}^{(1)}, \texttt{Minus}^{(1)}\}$ aufgebaut werden.

Beispiel 8.10 **Datenregeln für ganze Zahlen, II**

```
Succ(Minus(Succ(x)))  →  Minus(x)
Minus(Zero)           →  Zero
Minus(Minus(x))       →  x
```

□

Die Methode der Konstruktorfunktionen ermöglicht die gleichzeitige Verwendung der zu den beiden Sätzen von Konstruktorsymbolen gehörenden Konstruktorfunktionssymbole.

Beispiel 8.11 Erweiterte Konstruktorfunktionssymbole für ganze Zahlen
Zu den Regeln des Beispiels 8.6 kommen noch

```
minus(Zero)     →  Zero
minus(Succ(x))  →  Pred(minus(x))
minus(Pred(x))  →  Succ(minus(x))
```

□

Die gleichen Konstruktorfunktionssymbole können auch bei einem Aufbau der ganzen Zahlen aus $\mathcal{C} = \{\texttt{Zero}^{(0)}, \texttt{Succ}^{(1)}, \texttt{Minus}^{(1)}\}$ verwendet werden.

Beispiel 8.12 Konstruktorfunktionen und ganze Zahlen, II

```
zero                 →  Zero
succ(Zero)           →  Succ(Zero)
succ(Succ(x))        →  Succ(Succ(x))
succ(Minus(Succ(x)))  →  Minus(x)

pred(Zero)       →  Pred(Zero)          minus(Zero)       →  Zero
pred(Succ(x))    →  x                   minus(Succ(x))    →  Minus(Succ(x))
pred(Minus(x))   →  Minus(Succ(x))      minus(Minus(x))   →  x
```

□

Auf diese Weise ist also eine vollständige Trennung der Erzeugung und der internen Darstellung von Datenelementen möglich. Allerdings werden die Konstruktoren nach wie vor beim Patternmatching verwendet. In [Bur&Cam93] sind daher mehrere Sätze von Konstruktorsymbolen zur gleichzeitigen Verwendung für einen Datentyp vorgesehen. Nur einer von diesen dient tatsächlich dem Aufbau und damit der internen Darstellung der Datenelemente. Für jeden anderen Satz von Konstruktorsymbolen, **View** genannt, wird jedoch eine Abbildung von der internen Darstellung in die View definiert, so daß alle Views zum Patternmatching verwendet werden können. Diese Abbildung ist beliebig und kann durchaus eine Projektion sein.
Durch die Views werden Patternmatching und interne Darstellung vollständig voneinander getrennt und somit das Prinzip der geheimen internen Darstellung eines abstrakten Datentyps gewahrt. Allerdings würden die Views eine Neudefinition der Semantik erfordern.

Alle hier vorgestellten Konzepte beruhen mehr oder weniger auf der Verwendung von Normalformen. Es existieren jedoch nicht-freie Datentypen, die prinzipiell keine Normalformen zulassen. Das wichtigste Beispiel hierfür sind Mengen. Elemente einer Menge sind prinzipiell ungeordnet, aber in jeder Art von Normalform wären sie geordnet. Die schlichte Festlegung einer beliebigen Ordnung ist nicht möglich, da nicht zu jeder Sorte von Mengenelementen überhaupt eine Ordnung existieren muß, und vor allem auf diese Weise nicht mehr Mengen, sondern geordnete Mengen (geordnete Listen ohne mehrfache Elemente) spezifiziert würden. Mengen sind also nicht-freie Datentypen, die mit den vorgestellten Konzepten nicht spezifizierbar sind.

Eine Erweiterung um einen Kapselungsmechanismus würde dies jedoch ändern. Innerhalb eines Moduls würden Mengen schlicht durch freie Konstruktorterme mehrdeutig repräsentiert. Außerhalb des Moduls dürfte aber nur mit bestimmten Funktionen, beispielsweise Erzeugung der leeren Menge, Hinzunahme und Entnahme eines Elements und Test auf Leerheit und Enthaltensein eines Elements, auf die Konstruktorterme zugegriffen werden. Alle mit diesen Funktionen nicht unterscheidbaren Konstruktorterme würden als semantisch gleich betrachtet. Diese Beobachtungs- oder Verhaltenssemantik (vgl. 6.5 in [Wir90]) würde das Geheimnisprinzip abstrakter Datentypen realisieren. Natürlich wäre außerhalb des Moduls Patternmatching mit den Konstruktoren der internen Darstellung nicht zulässig, aber bei Mengen wäre immerhin noch ein Patternmatching mit einer View $\{\texttt{EmptySet}^{(0)}, \texttt{NonEmptySet}^{(0)}\}$ möglich.

Views und gekapselte Module mit Beobachtungssemantik wären somit interessante Erweiterungen unserer Programmiersprachen und des pragmatischen Konzepts der Konstruktorfunktionen zur Spezifizierung nicht-freier Datentypen.

Kapitel 9

Zusammenfassung und Ausblick

Wir haben uns in dieser Arbeit mit der Definition und Untersuchung von Semantiken für konstruktorbasierte funktionale Programme erster Ordnung unter besonderer Berücksichtigung des Patternmatchings befaßt.
Am Anfang haben wir die abstrakte Syntax zweier Arten von konstruktorbasierten funktionalen Programmen erster Ordnung definiert: der Programme mit Pattern und der Programme mit Hilfsfunktionen. Für diese haben wir als Verallgemeinerung der üblichen call-by-value und call-by-name Semantiken die Menge der ς-Semantiken definiert, die bezüglich der erzwungenen Striktheit ς parametrisiert sind. Jede ς-Semantik ist durch eine denotationelle und zwei operationelle Semantiken angegeben worden. Schon die Komplexität des Beweises der Übereinstimmung dieser drei Semantiken unterstreicht ihre Verschiedenartigkeit.
Bei der Definition der denotationellen ς-Semantik stand die Forderung der Kompositionalität im Vordergrund. Ein allein durch die Konstruktorsymbole bestimmter ς-Basisdatentyp wird durch das Programm um Operationen zum ς-Datentyp des Programms, der Semantik des Programms, erweitert. Dies kann durchaus schrittweise erfolgen, da die Erweiterung eines Programms um neue Funktionssymbole mit zugehörigen Programmregeln die schon bestehenden Operationen unverändert läßt. Die denotationelle ς-Semantik ist eine Fixpunktsemantik, die auf einer Menge von ς-Interpretationen, die in engem Zusammenhang mit dem Basisdatentypen stehen, und auf einer sich aus dem Programm ergebenden ς-Transformation basiert.
Die beiden operationellen ς-Semantiken beruhen auf der Idee der in der denotationellen ς-Semantik korrekten ς-Reduktion. Diese ς-Reduktion ist durch die Menge der ς-Redexe charakterisiert. Die Semantik eines Grundterms wird durch die semantischen Approximationen der vermittels ς-Reduktion erreichbaren Grundterme bestimmt. Diese Grundtermsemantik muß invariant sein, um überhaupt die Definition einer semantischen Gleichheit und die Existenz eines zugehörigen Datentyps zu gestatten. Während die allgemeine ς-Reduktionssemantik noch auf allen möglichen ς-Reduktionen beruht, verwendet die po-ς-Reduktionssemantik eine Reduktionsstrategie. Aufgrund deren Determinismus ist sie als Basis einer Implementierung unserer Programmiersprachen geeignet.
Wir haben auch noch einen Blick auf deklarative Semantiken geworfen und dabei festgestellt, daß einzig und allein die cbn-Semantik deklarativ definierbar ist. Durch die Verwendung partieller anstelle von totalen Algebren ist jedoch auch noch die cbv-Semantik (für Programme mit Pattern) deklarativ definierbar.
Die ς-Semantiken haben uns zu einem besseren Verständnis der Semantiken konstruktorbasierter funktionaler Programmiersprachen geführt. Insbesondere ermöglichen sie die einheitliche Behandlung der zwei Standardsemantiken call-by-value und call-by-name. In der Praxis können sie dem Nachweis der Korrektheit optimierender Interpreter und Compiler mit cbn-Semantik dienen, die

unter Ausnutzung von Striktheitsinformationen einen Teil der Funktionsargumente mit dem call-by-value Mechanismus auswerten.
In weiteren Untersuchungen haben wir noch festgestellt, daß für alle ς-Semantiken — mit Ausnahme der cbv-Semantik — unsere Programme mit Pattern echt ausdrucksstärker sind als unsere Programme mit Hilfsfunktionen.
Schließlich sind wir auf die Grenzen unserer Programmiersprachen gestoßen: Sie erlauben nur die Spezifikation freier Datentypen. In der Praxis mag allerdings der Programmierstil der Konstruktorfunktionen für die Beschreibung nicht-freier Datentypen ausreichen.

Aus diesen Resultaten ergeben sich jedoch auch wiederum viele neue Fragen.

Sind unsere ς-Semantiken die einzigen sinnvollen Semantiken für unsere Programme? Oder existieren noch weitere?
Bezüglich der Beziehungen zwischen denotationellen und operationellen Semantiken wäre es lohnenswert zu untersuchen, ob zu jeder invarianten Reduktionssemantik eine Transformation existiert, so daß die Reduktionssemantik mit dem kleinsten Fixpunkt der Transformation übereinstimmt. Ebenso interessant wäre die umgekehrte Richtung, und eventuell könnten auch nicht nur kleinste Fixpunkte betrachtet werden.
Generell wäre der Beweis einer Aussage der Art, daß unsere ς-Semantiken die einzigen konstruktorbasierten, invarianten (d. h. überhaupt einen Datentyp spezifizierenden), kompositionellen (einen Basisdatentyp erweiternden) Semantiken für unsere Programm sind, sehr schön. Die einzelnen geforderten Eigenschaften wären allerdings noch zu präzisieren, um insbesondere auch triviale Semantiken auszuschließen, wie diejenige, die alle Werte gleich $\bot$ setzt.
Bei einer Einbeziehung deklarativer Semantiken erhebt sich die Frage, ob unsere cbn-Semantik dann als einzige mögliche Semantik übrig bleibt, und was wir mit der cbv-Semantik machen.
Gerade die deklarative Definition des partiellen cbv-Datentyps hat uns vor Augen geführt, wie stark die Beantwortung vieler Fragen von den allen Definitionen zugrunde gelegten mathematischen Strukturen abhängt. Dies ruft nach neuen, allgemeineren und abstrakteren mathematischen Strukturen als Grundlagen semantischer Untersuchungen, um die durch sie gegebenen Grenzen soweit wie möglich hinauszuschieben.

In den letzten Kapiteln haben wir einige konkrete Ideen vorgestellt, deren Weiterverfolgung lohnend wäre.
So wäre vor allem zu beweisen, daß die Sequentialität tatsächlich das Scheidekriterium zwischen der Ausdrucksstärke der Programme mit Pattern und der mit Hilfsfunktionen ist. Hierfür wäre eine eingehendere Beschäftigung mit Methoden zum Beweisen der semantischen Äquivalenz von Programmen nötig.
Die angedeutete Methode zur Gewinnung effizienter ς-Reduktionsstrategien auf der Basis semantischer Sequentialitätsinformationen ließe sich präzise definieren. Bei alleiniger Betrachtung der linken Regelseiten und der erzwungenen Striktheit ς ist zu erwarten, daß die so definierte eventually gaining cbv-Reduktionsstrategie gleich der li-Reduktionsstrategie ist. Durch die Untersuchung der Beziehungen zwischen Sequentialitäts- und Striktheitsanalyse ließen sich Methoden für die Gewinnung präziserer Sequentialitätsinformationen für noch weitergehendere Optimierungen gewinnen.

Wir könnten die Grenzen unserer Programmiersprachen übertreten und die angedeuteten Erweiterungen um Views und gekapselte Module mit Beobachtungssemantik vornehmen, um auch die Spezifizierung echter nicht-freier Datentypen zu ermöglichen.
Schließlich könnten wir in einem weiteren Schritt (in Richtung einer realen funktionalen Programmiersprache) noch Funktionen höherer Ordnung betrachten. Dies nicht nur, weil diese ein äußerst

mächtiges Konzept darstellen, sondern auch, weil eine Verallgemeinerung oft zu neuen Einsichten führt, wie wir es bei der erzwungenen Striktheit gesehen haben.
Auf operationeller Ebene wären diese Programmiersprachen leicht durch Termersetzungssysteme realisierbar, in denen die Funktionssymbole Konstanten wären, und die nur ein einziges mehrstelliges Operationssymbol, $\texttt{apply}^{(2)}$, besäßen, dessen Operation die Anwendung einer Operation auf eine andere Operation wäre. Diese Termersetzungssysteme wären auch beinahe orthogonal, so daß die hier vorgestellten Methoden übertragen werden könnten.
Für die denotationelle Semantik müßten Algebren mit Operationen höherer Ordnung eingeführt werden. Da hierbei Operationen unterschiedlicher Ordnung klar voneinander geschieden wären, dies bei der gerade skizzierten operationellen Semantik jedoch nicht der Fall wäre, dürfte die Übereinstimmung beider Semantiken nicht einfach zu beweisen sein.
Die Semantik des Patternmatchings könnte jedoch fast unverändert übernommen werden, da Patternmatching prinzipiell nur für Basisdaten (Operationen nullter Ordnung) entscheidbar ist.
Die Sequentialität ist dagegen nicht einfach übertragbar ([Ber&Cur82]). Sie steht auch in direktem Zusammenhang mit dem schon lange bekannten Problem der Definition einer fully abstract Semantik, welches bis heute nicht zufriedenstellend gelöst wurde.

Die Beschäftigung mit formalen Semantiken ist nicht nur die zentrale Voraussetzung für den Entwurf korrekter Interpreter und Compiler sowie die Verifikation von Programmen, sondern sie führt uns auch zu einem immer besseren Verständnis von Programmiersprachen.

Literaturverzeichnis

[ADJ77] J. A. Goguen, J. W. Thatcher, E. G. Wagner, J. B. Wright: *Initial Algebra Semantics and Continuous Algebras*; JACM, Vol. 24, No. 1, Januar 1977, S. 68 – 95.

[ADJ78] J. B. Wright, E. G. Wagner, J. W. Thatcher: *A uniform approach to inductive posets and inductive closure*; Theoretical Computer Science 7, 1978, S. 57 – 77.

[An&Mid94] S. Antoy, A. Middeldorp: *A Sequential Reduction Strategy*; Proceedings of the 4th International Conference on Algebraic and Logic Programming, Madrid, LNCS 850, September 1994, S. 168 – 185.

[de Bakker76] J. W. de Bakker: *Least fixed points revisited*; Theoretical Computer Science 2, 1976, S. 155 – 181.

[Bau&Otto84] G. Bauer, F. Otto: *Finite Complete Rewriting Systems and the Complexity of the Word Problem*; Acta Informatica 21, 1984, S. 521 – 540.

[Ber&Klop86] J. A. Bergstra, J. W. Klop: *Conditional Rewrite Rules: Confluence and Termination*; Journal of Computer and System Sciences 32, 1986, S. 323 – 362.

[Ber&Tu80] J. A. Bergstra, J. V. Tucker: *A characterisation of computable data types by means of a finite equational specification method*; 7th Colloqium on Automata, Languages and Programming, LNCS 85, 1980, S. 76 – 90.

[Ber&Lévy77] G. Berry, J.-J. Lévy: *Minimal and Optimal Computations of Recursive Programs*; Fourth ACM Symposium on Principles of Programming Languages, 1977, S. 215 – 226.

[Berry78] G. Berry: *Stable models of typed λ-calculi*; Proceedings of the 5th ICALP Conference, Udine, Italien, LNCS 62, 1978, S. 73 – 89.

[Ber&Cur82] G. Berry, P. L. Curien: *Sequential Algorithms on Concrete Data Structures*; Theoretical Computer Science 20, 1982, S. 265 – 321.

[Broy&Wir82] M. Broy, M. Wirsing: *Partial Abstract Types*; Acta Informatica 18, 1982, S. 47 – 64.

[Broy&Wir83] M. Broy, M. Wirsing: *Generalized Heterogeneous Algebras and Partial Interpretations*; in: G. Ausiello; M. Protasi (ed.): Proc. 8th CAAP, LNCS 159, Springer, 1983, S. 1 –34.

[Bur86] P. Burmeister: *A Model Theoretic Oriented Approach to Partial Algebras*; Mathematical Research, Vol. 32, Akademie-Verlag, Berlin, 1986.

[BMS80] R. M. Burstall, D. B. MacQueen, D. T. Sannella: *Hope: An experimental applicative language*; ACM Conference on Lisp and Functional Programming, Stanford, 1980, S. 136 – 143

[Bur&Cam93] F. W. Burton, R. D. Cameron: *Pattern Matching with Abstract Data Types*; Journal of Functional Programming 3 (2), 1993, S. 171 – 190.

[Ca&Fell92] R. Cartwright, M. Felleisen: *Observable Sequentiality and Full Abstraction*; Conference Record of the 19th Annual ACM Symposium on Principles of Programming Languages, POPL, 1992, S. 328 – 342.

[Chen&O'Do91] Y. Chen, M. J. O'Donnell: *Infinite Terms and Infinite Rewritings*; Conditional and typed rewriting systems – 2nd International CTRS workshop, Montreal, Canada, Juni 1990, LNCS 516, 1991, S. 115 – 126.

[Com91] H. Comon: *Disunification: A Survey*; J.-L. Lassez, G. Plotkin (ed.): Computational Logic – Essays in Honor of Alan Robinson, MIT Press, 1991, S. 322 – 359.

[Cou80] B. Courcelle: *Completions of ordered magmas*; Fundamenta Informaticae III.1, 1980, S. 105 – 116.

[Cou83] B. Courcelle: *Fundamental Properties of Infinite Trees*; Theoretical Computer Science 25, 1983, S. 95 – 169.

[Cou90] B. Courcelle: *Recursive Applicative Program Schemes*; in: J. van Leeuwen (ed.): Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics, Elsevier, Amsterdam 1990, S. 459 – 492.

[Dau89] M. Dauchet: *Simulation of Turing machines by a left-linear rewrite rule*; Proc. 3rd RTA, Springer Verlag, LNCS 355, 1989, S. 109 – 120.

[DHLT90] M. Dauchet, T. Heuillar, P. Lescanne, S. Tison: *Decidability of the confluence of finite ground term rewriting systems and of other related term rewrite systems*; Information and Computation, Vol. 88, Nr. 2, 1990, S. 187 – 201

[Der87] N. Dershowitz: *Termination of Rewriting*; Journal on Symbolic Computation 3 (1&2), 1987, S. 69 – 115, Corrigendum: 4, 1987, S. 409 – 410.

[Der&Jou90] N. Dershowitz, J.-P. Jouannaud: *Rewrite Systems*; in: J. van Leeuwen (ed.): Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics, Elsevier, Amsterdam 1990, S. 243 – 320.

[DKP91] N. Dershowitz, St. Kaplan, D. A. Plaisted: *Rewrite, rewrite, rewrite, rewrite, rewrite, ...*; Theoretical Computer Science 83, 1991, S. 71 – 96.

[DMK92] N. Dershowitz, S. Mitra, G. Sivakumar: *Decidable Matching for Convergent Systems*; CADE–11: 11th international conference on automated deduction, LNCS 607, S. 589 – 602.

[Dow&Se76] P. J. Downey, R. Sethi: *Correct Computation Rules for Recursive Languages*; SIAM J. Comput., Vol. 5, No. 3, September 1976, S. 378 – 401.

[Fehr89] E. Fehr: *Semantik von Programmiersprachen*, Kapitel 3: Mathematische Grundlagen; Springer-Verlag, 1989.

[Fie&Har88] A. F. Field, P. G. Harrison: *Functional Programming*; Addison-Wesley, 1988.

[Gue81] I. Guessarian: *Algebraic Semantics*; LNCS 99, 1981.

[Gut77] J. V. Guttag: *Abstract Data Types and the Development of Data Structures*; Comm. ACM 20 (6), Juni 1977, S. 396 – 404.

[Gut&Hor78] J. V. Guttag, J. J. Horning: *The Algebraic Specification of Abstract Data Types*; Acta Informatica 10, 1978, S. 27 – 52.

[Hermes78] H. Hermes: *Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit*; Springer-Verlag, 3. Auflage, 1978.

[Hof&Kut89] D. Hofbauer, R.-D. Kutsche: *Grundlagen des maschinellen Beweisens*; Vieweg Verlag, 2. Auflage, 1991, S. 113 – 154.

[Huet&Lank78] G. Huet, D. Lankford: *On the uniform halting problem for term rewriting systems*; Rapport de Recherche N^{o} 283, INRIA, März 1978.

[Huet&Lévy79] G. Huet, J.-J. Lévy: *Computations in Orthogonal Rewriting Systems I + II*; J.-L. Lassez, G. Plotkin (ed.): Computational Logic – Essays in Honor of Alan Robinson, MIT Press, 1991, S. 395 – 443; Orginalversion: *Call by Need Computations in Non-Ambiguous Term Rewriting Systems*, Report 359, INRIA, 1979.

[Huet80] G. Huet: *Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems*; JACM , Vol. 27, Nr. 4, 1980, S. 797 – 821.

[Huet&Op80] G. Huet, D. C. Oppen: *Equations and Rewrite Rules: A Survey*; in: R. Book (ed.): Formal Language Theory: Perspectives and Open Problems, Academic Press, 1980, S. 349 – 405.

[Hup78] U. L. Hupbach: *Rekursive Funktionen in mehrsortigen Peano-Algebren*; Elektronische Informationsverarbeitung und Kybernetik 14 (10), 1978, S. 491 – 506.

[Ind90] K. Indermark: *Universelle Algebra*; Vorlesungsskript, SS 1990.

[Ind93] K. Indermark: *Funktionale Programmierung*; Vorlesungsskript, SS 1993.

[Ind94] K. Indermark: *Programmschemata*; Vorlesungsskript, SS 1994.

[Jou&Kir91] J.-P. Jouannaud, C. Kirchner: *Solving Equations in Abstract Algebras: A Rule-Based Survey of Unification*; J.-L. Lassez, G. Plotkin (ed.): Computational Logic – Essays in Honor of Alan Robinson, MIT Press, 1991, S. 257 – 321.

[Kahn&Plot78] G. Kahn, G. Plotkin: *Concrete Domains*; Theoretical Computer Science 121 (1-2), 1993, S. 187 – 277; Orginalversion: *Structures de données conrètes*; Rapport IRIA-LABORIA 336, 1978.

[KNO90] D. Kapur, P. Narendran, F. Otto: *On Ground-Confluence of Term Rewriting Systems*; Information and Computation 86, 1990, S. 14 –31.

[KNZ87] D. Kapur, P. Narendran, H. Zhang: *On Sufficient-Completeness and Related Properties of Term Rewriting Systems*; Acta Informatica 24, 1987, S. 395 – 415.

[Kes92] D. Kesner: *Free Sequentiality in Orthogonal Order-Sorted Rewriting Systems with Constructors*; Proceedings of the 11th International Conference on Automated Deduction, Saratoga Springs, New York, LNCS 607, 1992, S. 603 – 617.

[Klop87] J. W. Klop: *Term Rewriting Systems: A Tutorial*; EATCS Bulletin 32, Juni 1987, S. 143 – 182.

[Klop&Midd91] J. W. Klop, A. Middeldorp: *Sequentiality in Orthogonal Term Rewriting Systems*; Journal of Symbolic Computation, Vol. 2, 1991, S. 161 – 195.

[Lav87] A. Laville: *Lazy pattern matching in the ML language*; Proc. 7th Conf. on Foundations of Software Technology and Theoretical Computer Science, LNCS 287, Springer, 1987, S. 400 – 419.

[Loe&Sie87] J. Loeckx, K. Sieber: *The Foundations of Program Verification*, Part B; Wiley – Teubner, 1987.

[Manna74] Z. Manna: *Mathematical Theory of Computation*, Chapter 5; McGraw Hill, 1974.

[Mar76] G. Markowsky: *Chain-complete posets and directed sets with applications*; Algebra Universalis 6, 1976, S. 53 – 68.

[Meyer90] B. Meyer: *Introduction to the Theory of Programming Languages*; Prentice Hall, 1990.

[MirMan89] *Miranda, Release 2, System Manual*; Research Software Ltd., 1989.

[Naoi&Ina89] T. Naoi, Y. Inagaki: *Algebraic Semantics and Complexity of Term Rewriting Systems*; 3rd International Conference on Rewriting Techniques and Applications, LNCS 355, 1989, S. 311 – 325.

[Nivat75] M. Nivat: *On the interpretation of recursive polyadic program schemes*; Symposia Mathematica 15 – Convegni del Febbraio e dell' Aprile del 1973, Rom, 1975, S. 255 – 281.

[Noll95] T. Noll: *Why Higher-Order Functions and Lazy Evaluation do not Matter*; wird eingereicht für: Functional Programming Languages and Computer Architecture 1995.

[O'Do77] M. J. O'Donnell: *Computing in Systems Described by Equations*; LNCS 58, Springer, 1977.

[Qya87] M. Oyamaguchi: *The Church-Rosser property for ground term rewriting systems is decidable*; Theoret. Comput. Sci., Vol. 49, Nr. 1, 1987, S. 43 – 79.

[Ra&Vui80] J.-C. Raoult, J. Vuillemin: *Operational and Semantic Equivalence Between Recursive Programs*; JACM, Vol. 27, No. 4, Oktober 1980, S. 772 – 796.

[Ros73] B. K. Rosen: *Tree-Manipulating Systems and Church-Rosser Theorems*; JACM, Vol. 20, 1973, S. 160 – 187.

[Shoen77] J. R. Shoenfield: *Axioms of Set Theory*; J. Barwise (ed.): Handbook of Mathematical Logic (Part B), Amsterdam, North-Holland, 1977, S. 321 – 344.

[Tha87] S. Thatte: *A Refinement of Strong Sequentiality for Term Rewriting Systems with Constructors*; Information and Computation 72, 1987, S. 46 – 65.

[Thiel84] J. J. Thiel: *Stop losing sleep over incomplete data type specifications*; Proceedings of the 11th Annual ACM Symposium on Principles of Programming Languages, Salt Lake City, Utah, 1984, S. 76 – 82.

[Tho86] S. J. Thompson: *Laws in Miranda*; Conference Record of the 1986 Conference on LISP and Functional Programming, 1986, S. 1 – 12.

[Tho89] S. J. Thompson: *A Logic for Miranda*; Formal Aspects of Computing 1, 1989, S. 339 – 365.

[Tho90] S. J. Thompson: *Lawful functions and program verification in Miranda*; Science of Computer Programming 13 (2-3), Mai 1990, S. 181 – 218.

[Tho94] S. J. Thompson: *A Logic for Miranda, Revisited*; Technical Report No. 1-94, University of Kent at Canterbury, 1994.

[Vui74] J. Vuillemin: *Correct and Optimal Implementations of Recursion in a Simple Programming Language*; Journal of Computer and System Sciences 9, 1974, S. 332 – 354.

[Vui74b] J. Vuillemin: *Syntax, sémantique et axiomatique d'un langage de programmation simple*; Thèse, Université Paris VI, 1974.

[We92] W. Wechler: *Universal Algebra for Computer Scientists*; Springer-Verlag, 1992.

[Wir90] M. Wirsing: *Algebraic Specification*; in: J. van Leeuwen (ed.): Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics, Elsevier, Amsterdam 1990, S. 675 – 788.

Index