Johann Wolfgang Goethe-Universität
Frankfurt am Main

Fachbereich Informatik und Mathematik
Institut für Informatik

Diplomarbeit

Automatische Prüfung und Berechnung von Überlappungen zum Vollständigkeitsnachweis von Diagrammen in Programmiersprachen höherer Ordnung

Conrad Rau

06. März 2009

eingereicht bei
Prof. Dr. Manfred Schmidt-Schauß
Künstliche Intelligenz und Softwaretechnologie

Danksagung

Ich möchte mich hiermit bei allen bedanken, die mich während der Entstehung dieser Arbeit begleitet und unterstützt haben.

Mein besonderer Dank gilt meinem Betreuer, Herrn Prof. Dr. Manfred Schmidt-Schauß und seinem Mitarbeiter Dr. David Sabel für Ihre hervorragende Betreuung und Ihre zahlreichen hilfreichen Anregungen.

Auch danken möchte ich auf diesem Wege ganz besonders meinen Eltern, die mir dieses Studium ermöglicht haben, meiner Freundin Rebecca, die immer ein offenes Ohr für meine Sorgen im Zusammenhang mit der Diplomarbeit hatte, und mir geholfen hat, diese zu meistern, meinem Bruder Philip, der meine Arbeit korrigiert und mir hilfreiche Tipps bezüglich verschiedener Formulierungen gegeben hat, Sieglinde Braun, die meine Arbeit Korrektur gelesen hat, sowie Florian Dörfler, der mich bei der Erstellung einer Webseite, zur Veröffentlichung des im Rahmen dieser Arbeit entstandenen Programms, unterstützt hat.

Conrad Rau

Erklärung gemäß DPO §11 Abs. 11

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbständig verfasst habe und keine anderen Quellen oder Hilfsmittel als die in dieser Arbeit angegebenen verwendet habe.

Conrad Rau, Oberursel den 06. März 2009

Contents

1 Einleitung

1.1 Motivation und Zielsetzung

Funktionale Programmiersprachen weisen eine Reihe von Eigenschaften auf, die zur modernen Softwareentwicklung benötigt werden: Zuverlässigkeit, Modularisierung, Wiederverwendbarkeit und Verifizierbarkeit.

Als Basis der Semantik von funktionalen Programmiersprachen dient der λ-Kalkül (Barendregt, 1984). Er besteht aus einer Sprache, den so genannten λ-Ausdrücken und aus Regeln, die angeben, wie λ-Ausdrücke transformiert werden können. Die Folge einer Anwendung von Reduktionsregeln auf einen Ausdruck wird als Reduktion oder Auswertung des Ausdrucks bezeichnet. Endet eine Folge von Reduktionen mit einem Ausdruck, auf den keine Reduktionsregel mehr anwendbar ist, und besitzt dieser Ausdruck eine bestimmte festgelegte Form, dann sagt man die Auswertung terminiert. Der resultierende Ausdruck wird als Wert bezeichnet.

Funktionale Programme, deren Semantik auf dem λ-Kalkül basiert, können als λ-Ausdrücke betrachtet werden. Die Art und Weise wie der Wert eines funktionalen Programmes berechnet wird, ist dann durch die Reduktionsregeln des λ-Kalküls bestimmt. Man bezeichnet den λ-Kalkül in diesem Fall als die Kernsprache der funktionalen Programmiersprache.

Es existieren eine Vielzahl verschiedener Varianten von λ-Kalkülen, die sich in ihrem Sprachumfang, ihren Reduktionsregeln und ihren Auswertungsstrategien unterscheiden. Eine Klasse von Varianten des λ-Kalküls, die eine besondere Relevanz als semantische Grundlage für funktionale Programmiersprachen besitzt, sind die so genannten Lazy-λ-Kalküle, die auch λ-Kalküle mit Sharing oder λ-Kalküle mit `letrec`-Konstrukt genannt werden (Abramsky, 1990; Ariola & Felleisen, 1997; Schmidt-Schauß, 2003). Sie verfügen über Sprachkonstrukte und entsprechende Reduktionsregeln, die eine effiziente Auswertung von Ausdrücken durch die geteilte Nutzung gemeinsamer Unterausdrücke erlaubt.

Die Methode, die Semantik einer funktionalen Programmiersprache auf einer Variante des λ-Kalküls aufzubauen, hat den Vorteil, dass der zugrundeliegende Kalkül verwendet werden kann, um Aussagen darüber zu treffen, wann ein Berechnungsschritt bei der Auswertung eines funktionalen Programms als korrekt zu bezeichnen ist. Als theoretische Basis für solche Korrektheitsaussagen dient das Konzept der

Gleichheit von Ausdrücken. Der Begriff der Gleichheit von Ausdrücken ist als Verhaltensgleichheit von Ausdrücken definiert: Zwei Ausdrücke s und t sind gleich, wenn in einem beliebigen Programm der Ausdruck s durch den Ausdruck t ersetzt werden kann, ohne dass sich etwas am Terminierungsverhalten des Programms ändert. In λ-Kalkülen bezeichnet man das umschließende Programm als Kontext und die Gleichheitsrelation auf Ausdrücken als kontextuelle Äquivalenz. Die Gleichheitsrelation kann man auch folgenderma"sen interpretieren:

> Ausdrücke sind gleich, wenn man sie mit keinem Experiment unterscheiden kann (Schmidt-Schauß, 2006).

Das Experiment besteht aus der Einsetzung in einen Kontext (für alle möglichen Kontexte) und der anschließenden Beobachtung des Terminierungsverhaltens, des durch die Einsetzung entstandenen Programms.

Die so definierte Gleichheitsrelation ist eine Äquivalenzrelation auf Ausdrücken, die verwendet wird um die Korrektheit von Programmtransformationen zu definieren. Durch Programmtransformationen können Ausdrücke zu anderen Ausdrücken umgeformt werden. Für eine korrekte Programmtransformation gilt, dass der Ausdruck, auf den die Transformation angewendet wird, kontextuell äquivalent ist zu dem Ausdruck, der aus der Anwendung der Transformation resultiert. D.h. ein Ausdruck, der durch eine korrekte Programmtransformation umgeformt wird, weist vor und nach der Transformation das gleiche Terminierungsverhalten unter Einsetzung in einen beliebigen Kontext auf.

Die Reduktionsregeln eines λ-Kalküls sind Programmtransformationen. Durch den Begriff der Korrektheit von Programmtransformationen hat man eine Methode, um für eine funktionale Programmiersprache, deren Semantik auf einem λ-Kalkül basiert, Aussagen über die Korrektheit von Berechungsschritten in der funktionalen Programmiersprache zu machen. Eine andere wichtige Anwendung des Konzepts der Korrektheit von Programmtransformationen liegt im Bereich der Optimierung. Ein ineffizientes Unterprogramm wird durch eine Transformation zu einem effizienten Unterprogramm umgeformt. Wenn diese Transformation korrekt ist, dann wird durch die kontextuelle Äquivalenz der beiden Unterprogramme sichergestellt, dass in einem Programm das ineffiziente Unterprogramm an jeder Stelle durch seine transformierte, effiziente Variante ersetzt werden kann.

Um die Korrektheit einer Programmtransformation, die einen Ausdruck s zu einem Ausdruck t umformt, zu beweisen, muss gezeigt werden, dass s und t kontextuell äquivalent sind. Dazu muss das Terminierungsverhalten von s eingesetzt in einen beliebigen Kontext C (geschrieben als $C[s]$) verglichen werden mit dem Terminierungsverhalten von t eingesetzt in den gleichen Kontext C. Dies kann man zeigen, indem man annimmt, dass $C[s]$ ein Programm repräsentiert, das terminiert. D.h. es gibt für $C[s]$ eine Folge von Reduktionen, so dass die Auswertung mit einem Wert anhält. Nun versucht man, induktiv über die Anzahl der Reduktionsschritte eine Folge von Re-

duktionen zu konstruieren, so dass das Programm $C[t]$ unter dieser Folge terminiert. Für den vollständigen Beweis der kontextuellen Äquivalenz von s und t, muss noch analog gezeigt werden: Wenn $C[t]$ ein terminierendes Programm darstellt, dann ist auch $C[s]$ ein terminierendes Programm. Die Konstruktion einer terminierenden Folge von Reduktionen während der Induktion geschieht unter der Verwendung von Diagrammen. Für die beiden Schritte im Korrektheitsbeweis[1] werden verschiedene Arten von Diagrammen benötigt. Damit die Diagramme im Beweis verwendbar sind, muss außerdem ein bestimmtes Vollständigkeits-Kriterium erfüllt sein: Es wird eine Menge von Diagrammen benötigt, die alle Möglichkeiten der Konstruktion von terminierenden Reduktionsfolgen abdeckt. Die Erstellung vollständiger Mengen von Diagrammen erfolgt in den Arbeiten, die Korrektheit von Programmtransformationen in λ-Kalkuli mit Sharing untersuchen von Hand (Kutzner, 2000; Schmidt-Schauß, 2003; Sabel, 2003; Schmidt-Schauß, Sabel, & Schuetz, 2007; Sabel, 2008). Dieses Vorgehen ist umständlich, fehleranfällig und variiert stark im Grade der Ausführlichkeit zwischen den einzelnen Arbeiten.

Huber (2000) stellt in seiner Diplomarbeit ein Programm vor, das für eine gegebene Menge von Diagrammen anhand von Beispielausdrücken testet, ob die Menge der Diagramme die geforderte Vollständigkeits-Bedingung erfüllt. Dieses Vorgehen kann durch die geschickte Erzeugung von Ausdrücken feststellen, ob in einer Menge von Diagrammen, eines enthalten ist, das für die zu testenden Ausdrücke anwendbar ist. Die Vollständigkeit einer Menge von Diagrammen kann mit dieser Methode nicht gezeigt werden.

In dieser Arbeit wird ein einfacher λ-Kalkül mit einem `letrec`-Konstrukt betrachtet, der so genannte Λ^{let}-Kalkül. Das Ziel, ist für diesen Kalkül eine Methode zu entwickeln, mit der eine vollständige Menge von Diagrammen berechnet werden kann, die bei den Korrektheitsbeweisen von Programmtransformationen im Λ^{let}-Kalkül verwendet werden.

1.2 Überblick

Im Folgenden geben wir einen Überblick über die weiteren Kapitel.

In Kapitel 2 wird der Λ^{let}-Kalkül definiert. Der Kalkül stellt eine Teilmenge des LR-Kalküls aus Schmidt-Schauß et al. (2007) dar. Es werden die zentralen Konzepte der Reduktion, eine standardisierte Form der Auswertung und der Begriff der kontextuellen Äquivalenz aus Schmidt-Schauß et al. (2007) angegeben. Anschließend wird beschrieben, wann Programmtransformationen als korrekt bezeichnet werden, und welche Hilfsmittel für den Beweis der Korrektheit von Programmtransformationen verwendet werden. Diese sind das Kontextlemma und vollständige Sätze von

[1] Die Terminierung von $C[s]$ impliziert die Terminierung von $C[t]$ und die Terminierung von $C[t]$ impliziert die Terminierung von $C[s]$.

Gabel- und Vertauschungsdiagrammen. Das Kapitel endet mit einer Skizzierung der Methode zur Berechnung eines vollständigen Satzes von Gabeldiagrammen. Die Methode basiert auf der Berechnung von Überlappungen von Reduktionsregeln. Dazu müssen Reduktionsregeln des Λ^{let}-Kalküls unifiziert werden.

Die folgenden vier Kapitel beschäftigen sich mit den Unifikationsmethoden, die zur Unifikation von verschiedenen Konstrukten, die in Λ^{let}-Reduktionsregeln enthalten sind, benötigt werden. Die Objekte auf denen Λ^{let}-Reduktionsregeln operieren, werden als Ausdrücke bezeichnet. In der Literatur wird die Theorie der Unifikation (Baader & Snyder, 2001) allerdings im Rahmen von Termen und nicht bezüglich Ausdrücken behandelt. In den Kapiteln 3, 4, 5 und 6 wird deshalb eine Signatur Σ^{let} definiert, mit deren Hilfe Konstrukte, die in Λ^{let}-Reduktionsregeln vorkommen, als Terme über der Signatur Σ^{let} dargestellt und unifiziert werden können.

Die Reduktionsregeln in Λ^{let} enthalten Variablen verschiedener Sorten. In Kapitel 3 werden Terme mit Sorten definiert und das Verfahren zur Unifikation von Termen mit Sorten beschrieben.

Kapitel 4 befasst sich mit der Unifikation von Funktionssymbolen, deren Argumente vertauschbar sind. Diese Unifikationsmethode wird benötigt, da die Elemente von `letrec`-Umgebungen in Λ^{let} vertauschbar sind und diese Vertauschbarkeit bei der Unifikation berücksichtigt werden muss.

`letrec`-Umgebungen in Λ^{let}-Reduktionsregeln enthalten in manchen Fällen ein Konstrukt, um Variablenketten beliebiger Länge darzustellen. Die Methoden, die zur Unifikation von Variablenketten verwendet werden, sind in Kapitel 5 beschrieben.

Als letztes Konstrukt, das es bei der Unifikation zu berücksichtigen gilt, enthalten Λ^{let}-Reduktionsregeln Kontextvariablen mit Sorten. Deren Unifikation erfolgt wie in Kapitel 6 dargelegt.

In Kapitel 7 wird der Begriff der Überlappung von Reduktionsregeln formal definiert. Dazu geben wir zuerst die Definition der Signatur Σ^{let}, die zur Darstellung und Unifikation von Λ^{let}-Reduktionsregeln verwendet wird, zusammenfassend an. Dann werden Reduktionsregeln für Terme über dieser Signatur und eine standardisierte Form der Auswertung, analog zu den entsprechenden Definitionen des Λ^{let}-Kalküls, definiert. Die in Σ^{let} definierte Normalordnungsreduktion stellt eine Einschränkung der Normalordnungsreduktion aus Λ^{let} dar. Abschließend beschreiben wir, wie alle Überlappungen für einen vollständigen Satz von Gabeldiagrammen berechnet werden können.

Das Kapitel 8 befasst sich mit der Implementierung, die im Rahmen dieser Arbeit entstanden ist. Sie berechnet vollständige Sätze von Gabeldiagrammen. Wir skizzieren einige Datenstrukturen des Programms und geben Beispiele für die Arbeitsweise der wichtigsten Funktionen. Anschließend geben wir die vollständigen Sätze von Gabeldiagrammen an, die das Programm berechnet.

Eine Zusammenfassung der Arbeit sowie ein Ausblick auf mögliche weitere Untersuchungen bezüglich des in der Arbeit behandelten Themas wird in Kapitel 9 gegeben.

2 Der Λ^{let}-Kalkül

Der Λ^{let}-Kalkül ist ein Lambda-Kalkül, erweitert um ein `letrec`-Konstrukt. Der Kalkül ist eine Teilmenge des LR-Kalküls aus (Schmidt-Schauß et al., 2007) und besteht aus der Sprache $L_{\Lambda^{let}}$, *Reduktionsregeln* zur Transformation von Ausdrücken der Sprache und einer standardisierten Form der Auswertung, der so genannten *Normalordnung*. Im vorliegenden Kapitel werden diese Begriffe analog zu (Schmidt-Schauß et al., 2007) definiert. Ebenfalls auf diesem Artikel basierend, wird das zentrale Konzept der *kontextuellen Äquivalenz* von Ausdrücken definiert und davon ausgehend wird festgelegt, wann Transformationen von Ausdrücken (so genannte *Programmtransformationen*) als *korrekt* bezeichnet werden. Als wesentliche Hilfsmittel, um die Korrektheit von Programmtransformationen zu zeigen, dienen das *Kontextlemma* und vollständige Sätze von *Gabel- und Vertauschungsdiagrammen*. Abschließend wird informell besprochen, wie die Berechnung aller Überlappungen für Gabeldiagramme möglich ist.

2.1 Syntax

Definition 2.1.1. Sei X eine abzählbar unendliche Menge von Variablen. Die Syntax der Sprache $L_{\Lambda^{let}}$ wird durch die Grammatik in Abbildung 2.1 definiert.

E ::=	V	(Variable)
|	$(\lambda V.E)$	(Abstraktion)
|	$(E_1\ E_2)$	(Applikation)
|	$(\texttt{letrec}\ \{V_1 = E_1, \ldots, V_n = E_n\}\ \texttt{in}\ E)$	(`letrec`-Ausdruck)

Figure 2.1. Syntax der Sprache $L_{\Lambda^{let}}$

Die Nichtterminale V, V_i erzeugen Variablen $x, y, z, \ldots$ aus der Menge der Variablen X. Die Nichtterminale E, E_i erzeugen Ausdrücke. Für den `letrec`-Ausdruck gelten folgende Bedingungen:

- Die Variablen V_i sind paarweise verschiedene Variablen.
- Die Bindungen in `letrec`-Ausdrücken sind kommutativ (d.h. vertauschbar). D.h. zwei `letrec`-Ausdrücke, die sich lediglich in der Reihenfolge der Bindun-

gen unterscheiden, sind syntaktisch äquivalent.

- Das `letrec`-Konstrukt ist rekursiv, d.h. im Ausdruck (`letrec` $\{x_1 = E_1, \ldots, x_n = E_n\}$ `in` E) sind die Variablen $x_1, \ldots, x_n$ in $E_1 \ldots, E_n, E$ gebunden.

Mit $x, x_1, \ldots, y, z$ bezeichnen wir Variablen aus der Menge X. Beliebige Ausdrücke werden durch $s, s_1, \ldots, t, u$ bezeichnet. Die einzelnen Elemente $x_i = s_i$ eines `letrec`-Ausdrucks nennen wir `letrec`-*Bindungen* und die Menge aller `letrec`-Bindungen eines `letrec`-Ausdruckes `letrec`-*Umgebung*. Für einen `letrec`-Ausdruck (`letrec` $\{x_1 = s_1, \ldots, x_n = t_n\}$ `in` t) schreiben wir auch (`letrec` $\{Env\}$ `in` t), wobei Env syntaktisch äquivalent ist zu $x_1 = t_1, \ldots, x_n = t_n$. Die Notation $\{x_{g(i)} = s_{h(i)}\}_{i=m}^{n}$ wird verwendet für die Folge von `letrec`-Bindungen $x_{g(m)} = s_{h(m)}, x_{g(m+1)} = s_{h(m+1)}, \ldots, x_{g(n)} = s_{h(n)}$, mit $g, h : \mathbb{N} \to \mathbb{N}$. Eine solche Folge von Bindungen wird Kette genannt. Die Bindung $x_{g(m)} = s_{h(m)}$ wird als Anfangsbindung, die Bindung $x_{g(n)} = s_{h(n)}$ als Endbindung der Kette bezeichnet. D.h. $\{x_i = s_{i-1}\}_{i=m}^{n}$ steht für die Kette $x_m = s_{m-1}, x_{m+1} = s_m, \ldots, x_n = s_{n-1}$ mit Anfangsbindung $x_m = s_{m-1}$ und Endbindung $x_n = s_{n-1}$.

Freie und gebundene Variablen von Ausdrücken werden auf die übliche Art und Weise definiert (siehe beispielsweise Sabel (2003)). Abstraktionen und `letrec`-Ausdrücke binden Variablen. Um bei der Reduktion von Ausdrücken das ungewollte Einführen von freien Variablen in Bindungsbereiche zu vermeiden, wird für Ausdrücke angenommen, dass alle gebundenen Variablen verschiedene Namen besitzen, die sich von den Namen der freien Variablen eines Ausdrucks unterscheiden. Reduktionen führen bei Bedarf eine Umbenennung gebundener Variablen durch (was nur notwendig ist für die Kopier-Regel (*cp*), siehe Definition 2.3.1).

2.2 Kontexte

Kontexte sind Ausdrücke, die an einer bestimmten Stelle ein Loch $[\cdot]$ enthalten. Es werden verschiedene Klassen von Kontexten definiert, die sich dadurch unterscheiden, an welchen Positionen ein Loch zulässig ist.

Definition 2.2.1. Die Klasse $\mathcal{C}$ der *allgemeinen Kontexte* ist durch folgende Grammatik definiert:

$$
\begin{aligned}
\mathcal{C} ::= {} & [\cdot] \\
\mid {} & (\lambda x.\mathcal{C}) \\
\mid {} & (\mathcal{C}\ E) \mid (E\ \mathcal{C}) \\
\mid {} & (\texttt{letrec}\ \{x_1 = E_1, \ldots, x_n = E_n\}\ \texttt{in}\ \mathcal{C}) \\
\mid {} & (\texttt{letrec}\ \{x_1 = E_1, \ldots, x_{i-1} = E_{i-1}, x_i = \mathcal{C}, x_{i+1} = E_{i+1}, \ldots, x_n = E_n\}\ \texttt{in}\ E)
\end{aligned}
$$

Für Kontexte aus der Klasse der allgemeinen Kontexte verwenden wir die Buchstaben C, D. In einem allgemeinen Kontext darf das Loch an einer beliebigen Position in einem Ausdruck stehen, abgesehen von Positionen, an denen nur Variablen erlaubt sind (beispielsweise ist $\lambda[\cdot].t$ kein allgemeiner Kontext). Sei C ein Kontext und t ein Ausdruck, dann ist $C[t]$ der Ausdruck, der sich ergibt, wenn das Loch in C durch t ersetzt wird.

Wir definieren die Klasse der Oberflächenkontexte, die eine Einschränkung der Klasse der allgemeinen Kontexte darstellt: In einem Oberflächenkontext darf das Loch nicht im Rumpf einer Abstraktion vorkommen.

Definition 2.2.2. Die Klasse $\mathcal{S}$ der *Oberflächenkontexte* ist durch folgende Grammatik definiert:

$$
\begin{aligned}
\mathcal{S} ::= {} & [\cdot] \\
| \quad & (\mathcal{S}\ E) \mid (E\ \mathcal{S}) \\
| \quad & (\texttt{letrec}\ \{x_1 = E_1, \ldots, x_n = E_n\}\ \texttt{in}\ \mathcal{S}) \\
| \quad & (\texttt{letrec}\ \{x_1 = E_1, \ldots, x_{i-1} = E_{i-1}, x_i = \mathcal{S}, x_{i+1} = E_{i+1}, \ldots, x_n = E_n\}\ \texttt{in}\ E)
\end{aligned}
$$

Wir schreiben S für einen Kontext aus der Klasse der Oberflächenkontexte.

Eine weitere Kontextklasse ist die Klasse der Reduktionskontexte, die mit Hilfe einer Unterklasse, der Klasse der schwachen Reduktionskontexte, definiert wird.

Definition 2.2.3. Die Klasse $\mathcal{R}^-$ der *schwachen Reduktionskontexte* ist durch folgende Grammatik definiert:

$$
\begin{aligned}
\mathcal{R}^- ::= {} & [\cdot] \\
| \quad & (\mathcal{R}^-\ E)
\end{aligned}
$$

Die Klasse $\mathcal{R}$ der *Reduktionskontexte* ist definiert durch:

$$
\begin{aligned}
\mathcal{R} ::= {} & \mathcal{R}^- \\
| \quad & (\texttt{letrec}\ \{Env\}\ \texttt{in}\ \mathcal{R}^-) \\
| \quad & (\texttt{letrec}\ \{x_1 = \mathcal{R}_1^-, x_2 = \mathcal{R}_2^-[x_1], \ldots, x_j = \mathcal{R}_j^-[x_{j-1}], Env\}\ \texttt{in}\ \mathcal{R}^-[x_j]), \\
& \text{wobei } \mathcal{R}^-, \mathcal{R}_i^- \text{ Kontexte der Klasse der schwachen Reduktionskontexte sind.}
\end{aligned}
$$

Für Kontexte aus der Klasse der schwachen Reduktionskontexte schreiben wir R^- und für Kontexte aus der Klasse der Reduktionskontexte R.

Sei t ein Ausdruck mit $t = R^-[t_1]$, dann nennen wir R^- *maximal für* t, wenn es keinen größeren schwachen Reduktionskontext R_0^- gibt, so dass $t = R_0^-[t_2]$.

Sei $t = R[t_1]$ ein Ausdruck, so nennen wir R einen *maximalen Reduktionskontext für* t, wenn R

- ein maximaler schwacher Reduktionskontext R^- ist, oder

- von der Form $(\texttt{letrec}\ \{x_1 = s_1, \ldots, x_n = s_n\}\ \texttt{in}\ R^-)$ ist, wobei R^- ein maximaler schwacher Reduktionskontext ist und für alle j gilt $t_1 \neq x_j$, oder
- von der Form $(\texttt{letrec}\ \{x_1 = R_1^-, x_2 = R_2^-[x_1], \ldots, x_j = R_j^-[x_{j-1}], Env\}\ \texttt{in}\ R^-[x_j])$ ist, wobei R_1^- ein maximaler schwacher Reduktionskontext für $R_1^-[t_1]$ und die Anzahl j der Bindungen maximal ist.

Beispiel 2.2.4. Für den Ausdruck $t = (\texttt{letrec}\ \{x_1 = (\lambda y.y), x_2 = (x_1\ x_2)\}\ \texttt{in}\ x_2)$ ist der maximale Reduktionskontext $(\texttt{letrec}\ \{x_1 = [\cdot], x_2 = (x_1\ x_2)\}\ \texttt{in}\ x_2)$. Reduktionskontexte für t, die nicht maximal sind, haben die Form $(\texttt{letrec}\ \{x_1 = (\lambda y.y), x_2 = (x_1\ x_2)\}\ \texttt{in}\ [\cdot])$ oder $(\texttt{letrec}\ \{x_1 = (\lambda y.y), x_2 = ([\cdot]\ x_2)\}\ \texttt{in}\ x_2)$.

Eine Möglichkeit für einen Ausdruck $t = R[t_1]$ den maximalen Reduktionskontext zu bestimmen, ist durch den so genannten *Unwind* Algorithmus gegeben:

Definition 2.2.5 (Unwind)**.** Es werden vier Markierungen verwendet, die über die Struktur eines Ausdrucks geschoben werden: T, S, V, W. Die Markierung T kennzeichnet den am weitesten oben stehenden Ausdruck. Soll für einen Ausdruck t der maximale Reduktionskontext bestimmt werden, dann startet der Algorithmus mit t^T. Unterausdrücke von t werden mit S markiert. Unterausdrücke, die schon einmal besucht wurden, werden durch V und W markiert. Die Markierung W wird für Variablen an bestimmten Positionen verwendet und signalisiert, dass die markierte Variable nicht durch eine *cp*-Reduktion (die in Definition 2.3.1 zu sehen ist) ersetzt wird. Die Markierung $S \vee T$ steht für einen entweder mit S oder T gekennzeichneten Ausdruck. Soll ein bereits mit V oder W markierter Unterausdruck eine weitere Markierung erhalten, stoppt Unwind mit einem Fehler, um Schleifen zu vermeiden. Wenn keine Regel zum Verschieben von Markierungen mehr anwendbar ist, stoppt Unwind, mit dem Resultat $t = R[t_1^{S \vee T}]$, wobei R ein maximaler Reduktionskontext für t ist. Die Markierungen werden nach folgenden Regeln verschoben:

1. $(\texttt{letrec}\ \{Env\}\ \texttt{in}\ s)^T \rightarrow (\texttt{letrec}\ \{Env\}\ \texttt{in}\ s^S)^V$
2. $(s\ t)^{S \vee T} \rightarrow (s^S\ t)^V$
3. $(\texttt{letrec}\ \{x = s, Env\}\ \texttt{in}\ C[x^S]) \rightarrow (\texttt{letrec}\ \{x = s^S, Env\}\ \texttt{in}\ C[x^V])$
4. $(\texttt{letrec}\ \{x = s, y = C[x^S], Env\}\ \texttt{in}\ t) \rightarrow (\texttt{letrec}\ \{x = s^S, y = C[x^V], Env\}\ \texttt{in}\ t)$ wenn $C[x] \neq x$ gilt
5. $(\texttt{letrec}\ \{x = s, y = x^S, Env\}\ \texttt{in}\ t) \rightarrow (\texttt{letrec}\ \{x = s^S, y = x^W, Env\}\ \texttt{in}\ t)$

Der Markierungsalgorithmus steigt nicht in `letrec`-Ausdrücke hinab, die mit S markiert sind.

Beispiel 2.2.6. Sei $t = (\texttt{letrec}\ \{x_1 = (\lambda y.y), x_2 = (x_1\ x_2)\}\ \texttt{in}\ x_2)$ ein Ausdruck, für den der maximale Reduktionskontext mit Unwind bestimmt werden soll. Der Pfeil $\xrightarrow{i}$ gibt an, dass die Regel i des Algorithmus zum Verschieben einer Markierung

verwendet wurde.

$$\begin{aligned}
&(\texttt{letrec}\ \{x_1 = (\lambda y.y), x_2 = (x_1\ x_2)\}\ \texttt{in}\ x_2)^T \\
\xrightarrow{1}\ &(\texttt{letrec}\ \{x_1 = (\lambda y.y), x_2 = (x_1\ x_2)\}\ \texttt{in}\ x_2^S)^V \\
\xrightarrow{3}\ &(\texttt{letrec}\ \{x_1 = (\lambda y.y), x_2 = (x_1\ x_2)^S\}\ \texttt{in}\ x_2^V)^V \\
\xrightarrow{2}\ &(\texttt{letrec}\ \{x_1 = (\lambda y.y), x_2 = (x_1^S\ x_2)^V\}\ \texttt{in}\ x_2^V)^V \\
\xrightarrow{4}\ &(\texttt{letrec}\ \{x_1 = (\lambda y.y)^S, x_2 = (x_1^V\ x_2)^V\}\ \texttt{in}\ x_2^V)^V
\end{aligned}$$

Auf den markierten Ausdruck in der letzten Zeile ist keine Regel mehr anwendbar. Der maximale Reduktionskontext von t ist somit $(\texttt{letrec}\ \{x_1 = [\cdot], x_2 = (x_1\ x_2)\}\ \texttt{in}\ x_2)$, entsprechend dem Beispiel 2.2.4.

2.3 Reduktionen

Definition 2.3.1 (Reduktionsregeln)**.** Die *Reduktionsregeln* des Λ^{let}-Kalküls sind in Abbildung 2.2 zu sehen. Eine Reduktionsregel der Form

$$(name)\ \ a \to b$$

ist zu lesen als: Ein Ausdruck der Form a kann durch einen Ausdruck der Form b ersetzt werden durch Anwendung der Reduktionsregel mit der Bezeichnung $(name)$.

Die Vereinigung der Regeln $(llet\text{-}in)$ und $(llet\text{-}e)$ wird als $(llet)$, die Vereinigung der Regeln $(cp\text{-}in)$ und $(cp\text{-}e)$ als (cp) und die Vereinigung der Regeln $(lapp)$ und $(llet)$ als (lll) bezeichnet.

(llet-in)	$(\texttt{letrec}\ \{Env_1\}\ \texttt{in}\ (\texttt{letrec}\ \{Env_2\}\ \texttt{in}\ s)^S) \longrightarrow (\texttt{letrec}\ \{Env_1, Env_2\}\ \texttt{in}\ s)$
(llet-e)	$(\texttt{letrec}\ \{Env_1, x = (\texttt{letrec}\ \{Env_2\}\ \texttt{in}\ s)^S\}\ \texttt{in}\ t) \longrightarrow (\texttt{letrec}\ \{Env_1, Env_2, x = s\}\ \texttt{in}\ t)$
(cp-in)	$(\texttt{letrec}\ \{x_1 = v^S, \{x_i = x_{i-1}\}_{i=2}^n, Env\}\ \texttt{in}\ C[x_n^V]) \longrightarrow (\texttt{letrec}\ \{x_1 = v, \{x_i = x_{i-1}\}_{i=2}^n, Env\}\ \texttt{in}\ C[v])$ wenn v eine Abstraktion ist.
(cp-e)	$(\texttt{letrec}\ \{x_1 = v^S, \{x_i = x_{i-1}\}_{i=2}^n, Env, y = C[x_n^V]\}\ \texttt{in}\ t) \longrightarrow (\texttt{letrec}\ \{x_1 = v, \{x_i = x_{i-1}\}_{i=2}^n, Env, y = C[v]\}\ \texttt{in}\ t)$ wenn v eine Abstraktion ist.
(lapp)	$C[((\texttt{letrec}\ \{Env\}\ \texttt{in}\ s)^S\ t)] \longrightarrow C[(\texttt{letrec}\ \{Env\}\ \texttt{in}\ (s\ t))]$
(lbeta)	$C[((\lambda x.s)^S\ t)] \longrightarrow C[(\texttt{letrec}\ \{x = t\}\ \texttt{in}\ s)]$

Figure 2.2. Reduktionsregeln des Λ^{let}-Kalküls.

Wenn notwendig, notieren wir die verwendete Reduktionsregel oder auch den verwendeten Kontext über dem Reduktionspfeil. So ist z.B. $\xrightarrow{R,llet\text{-}in}$ eine $(llet\text{-}in)$-Reduktion in einem Reduktionskontext. Die transitive Hülle von Reduktionen bezeichnen wir mit $+$, die reflexiv-transitive Hülle mit $*$. So ist beispielsweise $\xrightarrow{let^+}$ die transitive Hülle von $\xrightarrow{let}$.

Die Reduktionsregeln $(cp\text{-}in)$ und $(cp\text{-}e)$ kann man als Regelschemata verstehen, die eine Menge von Regeln beschreiben: Für alle m die entsprechende Regel mit einer Variablenkette $\{x_i = x_{i-1}\}_{i=2}^{m}$ der Länge m.

Die Reduktionsregeln des Λ^{let}-Kalküls verwenden eine Meta-Notation: In den Reduktionsregeln symbolisieren die Buchstaben s, t beliebige Ausdrücke, Env, Env_i steht für beliebige `letrec`-Umgebungen und v repräsentiert eine beliebige Abstraktion. Außerdem enthalten die Reduktionsregeln die Kontextvariable C, die einen allgemeinen Kontext symbolisiert. Die jeweiligen Symbole können wie *Metavariablen* verstanden werden, für die Ausdrücke eines entsprechenden Typs eingesetzt werden können. In späteren Kapiteln werden wir schrittweise eine Abbildung definieren, die Ausdrücke in Terme eines anderen Kalküls übersetzt. Dabei sind wir vor allem an der Übersetzung von linken Seiten von Reduktionsregeln interessiert. Diese enthalten Metavariablen, die kein direkter Bestandteil der Sprache $L_{\Lambda^{let}}$ sind. Deswegen legen wir als Bezeichnung fest: Wenn wir in Zukunft von Λ^{let}-Ausdrücken sprechen, dann meinen wir Ausdrücke der Sprache $L_{\Lambda^{let}}$ erweitert um die Meta-Notation der Reduktionsregeln. Ist von der Übersetzung von Λ^{let} die Rede, dann ist die Übersetzung von Ausdrücken der Sprache $L_{\Lambda^{let}}$ erweitert um die Meta-Notation der Reduktionsregeln gemeint.

2.4 Normalordnungsreduktion

Sei R ein maximaler Reduktionskontext für einen Ausdruck t und $t = R[t']$. Die Normalordnungsreduktion wendet eine der Reduktionsregeln aus Definition 2.3.1 auf t' oder den Ausdruck direkt über t' an. Bevor wir die Normalordnungsreduktion formal definieren, betrachten wir, wie sie mit Hilfe des Unwind Algorithmus formuliert werden kann.

Sei t ein Ausdruck. Eine Ein-Schritt *Normalordnungsreduktion* $\xrightarrow{n}$ ist so definiert, dass zuerst der Unwind Algorithmus (aus Definition 2.2.5) auf t angewendet wird. Terminiert dieser erfolgreich, dann wird, wenn möglich, eine der Reduktionen aus Definition 2.3.1 angewendet, wobei die Markierungen S und V der Regeln den Markierungen von t entsprechen müssen.

Beispiel 2.4.1. Wir reduzieren einen Ausdruck t in Normalordnung. In jedem

Ausdruck sind die Markierungen, die durch *Unwind* gesetzt werden, mit angegeben.

$$
\begin{aligned}
t &= (\texttt{letrec}\ \{x_1 = (\lambda y.y)^S, x_2 = x_1^W, x_3 = (x_2^V\ (\lambda z.z))^V\}\ \texttt{in}\ x_3^V) \\
&\xrightarrow{n,cp\text{-}e} (\texttt{letrec}\ \{x_1 = (\lambda y.y), x_2 = x_1, x_3 = ((\lambda y.y)^S\ (\lambda z.z))^V\}\ \texttt{in}\ x_3^V) \\
&\xrightarrow{n,lbeta} (\texttt{letrec}\ \{x_1 = (\lambda y.y), x_2 = x_1, x_3 = (\texttt{letrec}\ \{y = (\lambda z.z)\}\ \texttt{in}\ y)^S\}\ \texttt{in}\ x_3^V) \\
&\xrightarrow{n,llet\text{-}in} (\texttt{letrec}\ \{x_1 = (\lambda y.y), x_2 = x_1, x_3 = y^W, y = (\lambda z.z)^S\}\ \texttt{in}\ x_3^V) \\
&\xrightarrow{n,cp\text{-}in} (\texttt{letrec}\ \{x_1 = (\lambda y.y), x_2 = x_1, x_3 = y, y = (\lambda z.z)\}\ \texttt{in}\ (\lambda z.z)^S)^V
\end{aligned}
$$

Auf den letzten Ausdruck ist keine Normalordnungsreduktion mehr anwendbar.

Definition 2.4.2 (Normalordnungsreduktion)**.** Sei t ein Ausdruck und R ein maximaler Reduktionskontext, so dass $t = R[t']$ für einen Ausdruck t'. Die Normalordnungsreduktion $\xrightarrow{n}$ ist durch einen der folgenden Fälle definiert:

1. t' ist ein `letrec`-Ausdruck und R ist nicht trivial (d.h. nicht gleich $[\cdot]$). Sei R_0 ein Reduktionskontext, so können folgende Fälle auftreten:
   a) $R = R_0[([\cdot]\ s)]$. Dann reduziere $(t'\ s)$ mit der Regel (*lapp*).
   b) $R = (\texttt{letrec}\ Env\ \texttt{in}\ [\cdot])$. Dann reduziere t mit der Regel (*llet-in*).
   c) $R = (\texttt{letrec}\ \{x = [\cdot],\ Env\}\ \texttt{in}\ t'')$. Dann reduziere t mit der Regel (*llet-e*).
2. t' ist eine Abstraktion. R_0 ist Reduktionskontext und R_0^- ist schwacher Reduktionskontext. Es können folgende Fälle auftreten:
   a) $R = R_0[([\cdot]\ s)]$. Dann reduziere $(t\ s)$ mit der Regel (*lbeta*).
   b) $R = (\texttt{letrec}\ \{x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^n, Env\}\ \texttt{in}\ R_0^-[x_n])$. Dann reduziere t mit der Regel (*cp-in*), so dass $R_0^-[x_n]$ durch $R_0^-[t']$ ersetzt wird.
   d) $R = (\texttt{letrec}\ \{x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^n, Env, y = R_0^-[(x_n\ s)]\}\ \texttt{in}\ t'')$, wobei y in einem Reduktionskontext ist. Dann reduziere t mit der Regel (*cp-e*), so dass $R_0^-[(x_n\ s)]$ durch $R_0^-[(t'\ s)]$ ersetzt wird.

Der *Normalordnungsredex* ist der gesamte Unterausdruck, auf den die entsprechende Reduktionsregel angewendet wird.

Der Begriff Normalordnungsreduktion wird abgekürzt als *no-Reduktion*. Eine Reduktionsfolge $t \to \cdots \to t_n$ ist eine Folge von Reduktionen des Λ^{let}-Kalküls. Besteht die Reduktionsfolge ausgehend von t ausschließlich aus Normalordnungsreduktionen, wird sie als *no-Reduktionsfolge von* t bezeichnet.

Korollar 2.4.3. Für alle Ausdrücke t gilt: Wenn t einen Normalordnungsredex besitzt, dann ist dieser eindeutig.

Wir sind hauptsächlich an Folgen von Normalordnungsreduktionen interessiert $\xrightarrow{n,*}$, die mit Ausdrücken enden, die nicht mehr weiter reduzierbar sind.

Definition 2.4.4 (WHNF)**.** Ein Ausdruck t ist eine *schwache Kopfnormalform* (WHNF[1]), wenn

- t eine Abstraktion ist, oder
- t von der Form $t = (\texttt{letrec}\ \{Env\}\ \texttt{in}\ v)$ und v eine Abstraktion ist.

Der letzte Ausdruck in Beispiel 2.4.1 ist eine WHNF, da er der zweiten Form aus obiger Definition entspricht.

Definition 2.4.5. Man sagt ein Ausdruck t *terminiert*, geschrieben als $t{\Downarrow}$, gdw. eine no-Reduktionsfolge ausgehend von t zu einer WHNF existiert. Sonst sagt man t *divergiert* und schreibt $t{\Uparrow}$.

2.5 Kontextuelle Äquivalenz

Die Grundlage der Semantik des Λ^{let}-Kalküls ist die Gleichheit von Ausdrücken, die durch den Begriff der *kontextuellen Äquivalenz* definiert wird.

Definition 2.5.1 (Kontextuelle Äquivalenz)**.** Die *Kontextuelle Quasiordnung* $\leq_C$ auf Ausdrücken s, t ist definiert durch

$$s \leq_C t,\ gdw.\ \forall C[\cdot] : C[s]{\Downarrow} \Rightarrow C[t]{\Downarrow},$$

und die *Kontextuelle Äquivalenz* $\sim_C$ für Ausdrücke s, t ist definiert durch

$$s \sim_C t,\ gdw.\ s \leq_C t \wedge t \leq_c s.$$

Unter der Kontextuellen Äquivalenz werden zwei Ausdrücke als gleich angesehen, wenn ihr Terminierungsverhalten unter Einsetzung in beliebige Kontexte gleich ist.

Die Relation $\leq_C$ ist eine Quasiordnung auf Ausdrücken (d.h. sie ist reflexiv und transitiv) und die Relation $\sim_C$ ist eine Äquivalenzrelation auf Ausdrücken (d.h. sie ist reflexiv, transitiv und symmetrisch). Außerdem sind beide Relationen stabil unter Einsetzung in Kontexte.

Proposition 2.5.2. Die Relation $\leq_C$ ist eine Quasikongruenz, d.h. für alle Kontexte C gilt $s \leq_c t \Rightarrow C[s] \leq_c C[t]$. Die Relation $\sim_C$ ist eine Kongruenz, d.h. für alle Kontexte C gilt $s \sim_c t \Rightarrow C[s] \sim_c C[t]$.

Beweis. Siehe (Schmidt-Schauß, 2003), Proposition 6.6. □

Um für zwei Ausdrücke s und t eine Aussage über ihre Kontextuelle Äquivalenz zu treffen, muss ihr Terminierungsverhalten unter Einsetzung in (alle) Kontexte

[1] WHNF ist die Abkürzung des englischen Begriffs weak head normalform.

C betrachtet werden. Die Kontexte sind dabei aus der Klasse der allgemeinen Kontexte. Folgendes Lemma zeigt, dass eine Betrachtung aller Kontexte R aus der kleineren Klasse der Reduktionskontexte ausreichend ist, um die Kontextuelle Äquivalenz von zwei Ausdrücken zu zeigen.

Lemma 2.5.3 (Kontextlemma)**.** Seien s und t Ausdrücke. Wenn für alle Reduktionskontexte R gilt: $R[s]\Downarrow \Rightarrow R[t]\Downarrow$, dann gilt auch für alle Kontexte C: $C[s]\Downarrow \Rightarrow C[t]\Downarrow$.

Beweis. Siehe (Schmidt-Schauß et al., 2007), Lemma A.1. □

Definition 2.5.4 (Korrekte Programmtransformation)**.** Eine *Programmtransformation* T ist eine binäre Relation auf Ausdrücken.

Eine Programmtransformation T ist korrekt, wenn für zwei Ausdrücke s, t gilt:

$$s\ T\ t \Rightarrow s \sim_C t.$$

Die Reduktionsregeln des Λ^{let}-Kalküls sind Programmtransformationen im Sinne der Definition 2.5.4.

Für manche Reduktionsregeln ist die Korrektheit einfach zu zeigen.

Proposition 2.5.5. Die Reduktionsregeln *lbeta* und *lapp* sind korrekte Programmtransformationen. D.h. es gilt $s \xrightarrow{a} t \Rightarrow s \sim_C t$ für $a \in \{lbeta, lapp\}$.

Beweis. Es sei $s \xrightarrow{a} t$, $a \in \{lbeta, lapp\}$ gegeben. Nach dem Kontextlemma reicht es zu zeigen, dass $R[s]\Downarrow \Leftrightarrow R[t]\Downarrow$ für alle Reduktionskontexte R gilt. Nach der Struktur von Reduktionskontexten kann eine $a \in \{lbeta, lapp\}$ Reduktion in einem Reduktionskontext nur eine no-Reduktion sein, d.h. $R[s] \xrightarrow{n,a} R[t]$. Da die Normalordnungsreduktion nach Korollar 2.4.3 eindeutig ist, folgt $R[s]\Downarrow \Leftrightarrow R[t]\Downarrow$. Durch Anwendung des Kontextlemmas folgt die Aussage des Lemmas. □

Um generell die Korrektheit einer Programmtransformation $s \xrightarrow{a} t$ zu zeigen, muss $s \xrightarrow{a} t \Rightarrow s \sim_C t$ gezeigt werden, was nach Definition 2.5.1 und dem Kontextlemma gleichbedeutend ist mit $s \xrightarrow{a} t \Rightarrow R[s]\Downarrow \Leftrightarrow R[t]\Downarrow$. Zum Beweis der Korrektheit der Programmtransformation a wird angenommen, dass $s \xrightarrow{a} t$ und $R[s]\Downarrow$ (bzw. $R[t]\Downarrow$) gelte. D.h. für $R[s]$ ($R[t]$) existiert eine no-Reduktionsfolge, die mit einem Ausdruck s' in WHNF terminiert. Ausgehend von dieser no-Reduktionsfolge und von $s \xrightarrow{a} t$ wird induktiv eine terminierende Reduktionsfolge für $R[t]$ ($R[s]$) konstruiert. Das wesentliche Hilfsmittel zu diesem Vorgehen sind Diagramme, die die Konstruktion von terminierenden Reduktionsfolgen während der Induktion ermöglichen.

2.6 Vertauschungs- und Gabeldiagramme

Notation 2.6.1. Reduktionen, die keine no-Reduktionen sind, werden *interne Reduktionen* genannt und mit $\xrightarrow{i}$ bezeichnet. Findet eine interne Reduktion in einem bestimmten Kontext statt, schreiben wir den Kontext mit an den Pfeil, beispielsweise $\xrightarrow{iC}$ für eine interne Reduktion in einem allgemeinen Kontext.

Interne Reduktionen sind von besonderem Interesse, weil für eine no-Reduktion $s \xrightarrow{n,a} t$ gilt $R[s] \xrightarrow{n,a} R[t]$ für alle Reduktionskontexte R und, da die no-Reduktion eindeutig ist, (Korollar 2.4.3) haben wir $R[s]{\Downarrow} \Leftrightarrow R[t]{\Downarrow}$. Dies gilt für interne Reduktionen i.A. nicht (außer für $(iR, lbeta)$ und $(iR, lapp)$ wie wir in Proposition 2.5.5 gesehen haben).

Definition 2.6.2. Eine Transformation auf Reduktionsfolgen hat die Form

$$\xrightarrow{iX,red} \cdot \xrightarrow{n,a_1} \cdot \ldots \cdot \xrightarrow{n,a_k} \quad \rightsquigarrow \quad \xrightarrow{n,b_1} \cdot \ldots \cdot \xrightarrow{n,b_m} \cdot \xrightarrow{iX,red_1} \cdot \ldots \cdot \xrightarrow{iX,red_h},$$

wobei red eine Λ^{let}-Reduktion aus Definition 2.3.1 ist und $iX \in \{iC, iS, iR\}$.

Eine Transformation

$$\xrightarrow{iX,red} \cdot \xrightarrow{n,a_1} \cdot \ldots \cdot \xrightarrow{n,a_k} \quad \rightsquigarrow \quad \xrightarrow{n,b_1} \cdot \ldots \cdot \xrightarrow{n,b_m} \cdot \xrightarrow{iX,red_1} \cdot \ldots \cdot \xrightarrow{iX,red_h}$$

ist *anwendbar* auf den Präfix einer Reduktionsfolge RED

$$s \xrightarrow{iX,red} t_1 \xrightarrow{n,a_1} t_2 \ldots t_k \xrightarrow{n,a_k} t$$

wenn Ausdrücke $y_1, \ldots y_m, z_1, \ldots z_{h-1}$ existieren, so dass

$$s \xrightarrow{n,b_1} y_1 \ldots y_{m-1} \xrightarrow{n,b_m} y_m \xrightarrow{iX,red_1} z_1 \ldots z_{h-1} \xrightarrow{iX,red_h} t$$

gilt. Die Transformation besteht aus dem Ersetzen des Präfixes mit dem Resultat:

$$s \xrightarrow{n,b_1} t'_1 \ldots t'_{m-1} \xrightarrow{n,b_m} t'_m \xrightarrow{iX,red_1} t''_1 \ldots t''_{h-1} \xrightarrow{iX,red_h} t,$$

wobei die Ausdrücke t'_i, t''_j durch die entsprechenden b_i-, red_j-Reduktionen entstehen.

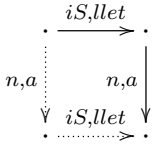
Definition 2.6.3 (Vollständiger Satz von Vertauschungsdiagrammen)**.** Zur Reduktion $\xrightarrow{iX,red}$ ist ein vollständiger Satz von Vertauschungsdiagrammen gegeben durch eine Menge von Transformationen auf Reduktionsfolgen der Form

$$\xrightarrow{iX,red} \cdot \xrightarrow{n,a_1} \cdot \ldots \cdot \xrightarrow{n,a_k} \quad \rightsquigarrow \quad \xrightarrow{n,b_1} \cdot \ldots \cdot \xrightarrow{n,b_m} \cdot \xrightarrow{iX,red_1} \cdot \ldots \cdot \xrightarrow{iX,red_{k'}}$$

mit $k, k' \geq 0, m \geq 1$, so dass für jede Reduktionsfolge $t_0 \xrightarrow{iX,red} t_1 \xrightarrow{n} \ldots \xrightarrow{n} t_l$, wobei t_l eine WHNF ist, mindestens eine der Transformationsregeln auf einen Präfix der Reduktionsfolge anwendbar ist.

Für den Fall $l = 1$ muss $t_0 \xrightarrow{iX,red} t_1$ so aussehen, dass t_0 keine WHNF und t_1 eine WHNF ist.

Zur Vereinfachung der Notation werden Vertauschungsdiagramme in Diagrammform dargestellt. Beispielsweise wird das Vertauschungsdiagramm $\xrightarrow{iS,llet} \cdot \xrightarrow{n,a} \rightsquigarrow \xrightarrow{n,a} \cdot \xrightarrow{iS,llet}$ repräsentiert durch



wobei die durchgezogenen Pfeile die gegebenen Reduktionen der linken Seite der Transformationsregeln und die gestrichelten Pfeile die existierenden Reduktionen der rechten Seite der Transformationsregel darstellen. Eine Variable a für den Namen einer Reduktionsregel, die mehrmals in einem Diagramm vorkommt, symbolisiert überall die gleiche Reduktionsregel.

Anstatt einer Reduktion $\xrightarrow{a}$ kann in einem Diagramm auch die transitive Hülle $\xrightarrow{a^+}$ (bzw. die reflexiv-transitive Hülle) einer Reduktion stehen.

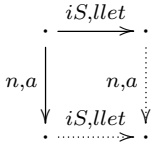
Definition 2.6.4 (Vollständiger Satz von Gabeldiagrammen)**.** Zur Reduktion $\xrightarrow{iX,red}$ ist ein vollständiger Satz von Gabeldiagrammen gegeben durch eine Menge von Transformationen auf Reduktionsfolgen der Form

$$\xleftarrow{n,a_1} \cdot \ldots \cdot \xleftarrow{n,a_k} \cdot \xrightarrow{iX,red} \quad \rightsquigarrow \quad \xrightarrow{iX,red_1} \cdot \ldots \cdot \xrightarrow{iX,red_{k'}} \cdot \xleftarrow{n,b_1} \cdot \ldots \cdot \xleftarrow{n,b_m}$$

mit $k, k' \geq 0, m \geq 1$, so dass für jede Reduktionsfolge $t_l \overset{n}{\leftarrow} \ldots t_2 \overset{n}{\leftarrow} t_1 \xrightarrow{iX,red} t_0$, $l > 1$, wobei t_l eine WHNF ist, mindestens eine der Transformationsregeln auf einen Suffix der Reduktionsfolge anwendbar ist.

Für den Fall $l = 1$ muss $t_1 \xrightarrow{iX,red} t_0$ so aussehen, dass t_1 eine WHNF und t_0 keine WHNF ist.

Für Gabeldiagramme verwenden wir ebenfalls eine vereinfachte Schreibweise. Beispielsweise wird das Gabeldiagramm $\xleftarrow{n,a} \cdot \xrightarrow{iS,llet} \rightsquigarrow \xrightarrow{iS,llet} \cdot \xleftarrow{n,a}$ repräsentiert durch



In den meisten Fällen können dieselben Diagramme für eine vollständige Menge von Gabel- und eine vollständige Menge von Vertauschungsdiagrammen verwendet werden. Mit dem Unterschied, an welcher Seite sich die existensquantifizierten Reduktionen befinden (siehe Sabel (2008), S. 113.) Aus diesem Grund ist es meistens ausreichend, sich bei der Erstellung eines vollständigen Diagrammsatzes entweder auf Gabel- oder Vertauschungsdiagramme zu konzentrieren. Der jeweils andere Satz kann dann durch Rückgriff auf den bereits vorhanden Satz gewonnen werden. In dieser Arbeit konzentrieren wir uns auf Gabeldiagramme.

2.6.1 Berechnung von Überlappungen für Gabeldiagramme

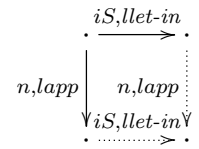
Um die Vollständigkeit eines Satzes von Gabeldiagrammen für eine interne Reduktion *red* zu zeigen, müssen alle Gabelungen der Form $\xleftarrow{n,a} \cdot \xrightarrow{iS,red}$,[2] wobei a eine beliebige no-Reduktion ist, betrachtet werden. Dazu müssen alle Überlappungen zwischen der no-Reduktion a und *red* bestimmt werden. Informell definieren wir Überlappungen folgendermaßen: Die Ausdrücke s, t *überlappen* in einem Ausdruck u, wenn s und t beides Unterausdrücke von u sind (d.h. sie kommen in u vor). Zwei Reduktionen a, b überlappen in u, wenn s ein a-Redex ist und t ein b-Redex und beide Redexe im Ausdruck u vorkommen.[3] Wenn alle möglichen Überlappungen zwischen einer no-Reduktion a und einer internen Reduktion *red* bestimmt werden sollen, kann man sich aufgrund der Struktur von Reduktionskontexten und der Normalordnungsreduktion darauf beschränken, alle Positionen in a zu bestimmen, an denen ein *red*-Redex vorkommen kann. (Da a eine no-Reduktion ist, ist die Struktur des (n, a)-Redexes s festgelegt auf $R[s]$, wobei R ein Reduktionskontext ist. Der (iS, red)-Redex t kann dann in R oder in s vorkommen.) Sind alle Überlappungen zwischen den (n, a)-Reduktionen und der internen Reduktion *red* bestimmt, dann müssen die Gabeln noch geschlossen werden. Dabei muss unter anderem überlegt werden, ob die Reduktion des internen Redexes den betrachteten no-Redex in einem Reduktionskontext belässt. Wir geben ein Beispiel, welche Überlegungen bezüglich Gabeldiagrammen angestellt werden müssen (für eine bestimmte no-Reduktion a, nicht für einen vollständigen Satz).

Beispiel 2.6.5. Wir betrachten als interne Reduktion eine *llet-in*-Reduktion und als no-Reduktion eine *lapp*-Reduktion, deren Redex die Form $R[((\texttt{letrec}\ \{Env\}\ \texttt{in}\ s)\ t)]$ hat. D.h. die Gabelung hat die Form $\xleftarrow{n,lapp} \cdot \xrightarrow{iS,llet\text{-}in}$. Die Positionen, an denen der interne *llet-in*-Redex im no-Redex vorkommen kann, sind: In R, Env, s oder t oder der interne Redex kann mit dem `letrec`-Ausdruck des no-Redexes überlappen: $R[((\texttt{letrec}\ \{Env\}\ \texttt{in}\ (\texttt{letrec}\ \{Env'\}\ \texttt{in}\ s'))\ t)]$. Für die ersten vier Fälle können die zugehörigen Gabeldiagramme jeweils durch $\xrightarrow{iS,llet\text{-}in} \cdot \xleftarrow{n,lapp}$ geschlossen werden, da die Reduktion des internen Redexes keinen Einfluss auf den no-Redex hat, d.h. eine Reduktion des internen Redexes belässt den no-Redex in einem Reduktionskontext und somit bleibt dieser no-reduzierbar. Ebenso wenig wird der interne Redex durch die Reduktion des no-Redex beeinflusst.

[2] Nach dem Kontextlemma ist es eigentlich ausreichend, interne Reduktionen in Reduktionskontexten zu betrachten. Zum Schließen von Diagrammen ist es aber in manchen Fällen notwendig, interne Reduktionen in den etwas allgemeineren Oberflächenkontexten zu betrachten.

[3] Eine formale Definition ist an dieser Stelle schwierig, weil Begriffe wie Substitution und Positionen eines Ausdrucks nicht zur Verfügung stehen. Eine formale Definition des Überlappungsbegriffs wird in Kapitel 7 gegeben.

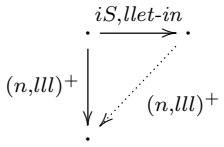
Für diese Fälle ergibt sich folgendes Gabeldiagramm:



Für den letzten Fall kann das Diagramm folgendermaßen geschlossen werden:

$$
\begin{array}{rl}
 & R[((\texttt{letrec}\ \{Env\}\ \texttt{in}\ (\texttt{letrec}\ \{Env'\}\ \texttt{in}\ s'))\ t)] \\
\xrightarrow{iS,llet\text{-}in} & R[((\texttt{letrec}\ \{Env, Env'\}\ \texttt{in}\ s')\ t)] \\
\xrightarrow{n,lapp} & R[((\texttt{letrec}\ \{Env, Env'\}\ \texttt{in}\ (s'\ t))] \\
\xrightarrow{(n,lll)^*} & (\texttt{letrec}\ \{Env, Env'\}\ \texttt{in}\ R[(s'\ t)]) \\
\hline
 & R[((\texttt{letrec}\ \{Env\}\ \texttt{in}\ (\texttt{letrec}\ \{Env'\}\ \texttt{in}\ s'))\ t)] \\
\xrightarrow{n,lapp} & R[(\texttt{letrec}\ \{Env\}\ \texttt{in}\ ((\texttt{letrec}\ \{Env'\}\ \texttt{in}\ s')\ t))] \\
\xrightarrow{(n,lll)^*} & (\texttt{letrec}\ \{Env\}\ \texttt{in}\ R[((\texttt{letrec}\ \{Env'\}\ \texttt{in}\ s')\ t)]) \\
\xrightarrow{n,lapp} & (\texttt{letrec}\ \{Env\}\ \texttt{in}\ R[(\texttt{letrec}\ \{Env'\}\ \texttt{in}\ (s'\ t))]) \\
\xrightarrow{(n,lll)^*} & (\texttt{letrec}\ \{Env\}\ \texttt{in}\ (\texttt{letrec}\ \{Env'\}\ \texttt{in}\ R[(s'\ t)])) \\
\xrightarrow{n,let\text{-}in} & (\texttt{letrec}\ \{Env, Env'\}\ \texttt{in}\ R[(s'\ t)])]
\end{array}
$$

Das zugehörige Gabeldiagramm ist



(2.1)

Zum Schließen des Diagramms wird hier die Tatsache verwendet, dass `letrec`-Ausdrücke in einem Reduktionskontext durch wiederholte Anwendung von (n, lll)-Reduktionen an die oberste Position gebracht werden können.

Lemma 2.6.6. Sei $t = (\texttt{letrec}\ \{Env\}\ \texttt{in}\ t')$ ein Ausdruck und R ein Reduktionskontext.

1. Ist R von der Form $R = (\texttt{letrec}\ \{Env'\}\ \texttt{in}\ R')$, wobei R' ein schwacher Reduktionskontext ist, dann gilt $R[(\texttt{letrec}\ \{Env\}\ \texttt{in}\ t')] \xrightarrow{(n,lll)^+} (\texttt{letrec}\ \{Env', Env\}\ \texttt{in}\ R'[t])$.
2. Ist R von der Form $R = (\texttt{letrec}\ \{Env', x = R'\}\ \texttt{in}\ r)$, wobei R' ein schwacher Reduktionskontext ist, dann gilt $R[(\texttt{letrec}\ \{Env\}\ \texttt{in}\ t')] \xrightarrow{(n,lll)^+} (\texttt{letrec}\ \{Env', Env, x = R'[t']\}\ \texttt{in}\ r)$, wobei $(\texttt{letrec}\ \{Env', Env, x = R'[\cdot]\}\ \texttt{in}\ r)$ ein Reduktionskontext ist.

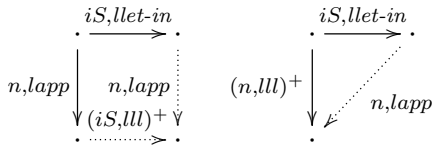
3. Ist R kein `letrec`-Ausdruck, d.h. ein schwacher Reduktionskontext, dann gilt $R[(\texttt{letrec}\ \{Env\}\ \texttt{in}\ t')] \xrightarrow{(n,lll)^*} (\texttt{letrec}\ \{Env\}\ \texttt{in}\ R[t'])$, wobei $(\texttt{letrec}\ \{Env\}\ \texttt{in}\ R[\cdot])$ ein Reduktionskontext ist.

Beweis. Siehe (Schmidt-Schauß et al., 2007), Lemma 2.5. □

Das Diagramm für die Überlappung $R[((\texttt{letrec}\ \{Env\}\ \texttt{in}\ (\texttt{letrec}\ \{Env'\}\ \texttt{in}\ s'))\ t)]$ kann aber noch auf eine andere Weise geschlossen werden:

$$\begin{array}{rl}
 & R[((\texttt{letrec}\ \{Env\}\ \texttt{in}\ (\texttt{letrec}\ \{Env'\}\ \texttt{in}\ s'))\ t)] \\
\xrightarrow{iS,llet\text{-}in} & R[((\texttt{letrec}\ \{Env, Env'\}\ \texttt{in}\ s')\ t)] \\
\xrightarrow{n,lapp} & R[((\texttt{letrec}\ \{Env, Env'\}\ \texttt{in}\ (s'\ t))] \\
\hline
 & R[((\texttt{letrec}\ \{Env\}\ \texttt{in}\ (\texttt{letrec}\ \{Env'\}\ \texttt{in}\ s'))\ t)] \\
\xrightarrow{n,lapp} & R[(\texttt{letrec}\ \{Env\}\ \texttt{in}\ ((\texttt{letrec}\ \{Env'\}\ \texttt{in}\ s')\ t))] \\
\xrightarrow{iS\wedge n,lapp} & R[(\texttt{letrec}\ \{Env\}\ \texttt{in}\ (\texttt{letrec}\ \{Env'\}\ \texttt{in}\ (s'\ t)))] \\
\xrightarrow{iS\wedge n,llet\text{-}in} & R[(\texttt{letrec}\ \{Env, Env'\}\ \texttt{in}\ (s'\ t))]
\end{array}$$

Die beiden letzten Reduktionen sind no-Reduktionen falls $R = [\cdot]$, sonst sind es interne Reduktionen. Die entsprechenden Gabeldiagramme sind



(2.2)

Man hat die Wahl, das Diagramm (2.1) oder die beiden Diagramme (2.2) in einen Satz vollständiger Gabeldiagramme aufzunehmen. Beide Diagramme sind auf den Suffix aller Transformationsfolgen der Form $\ldots, \xleftarrow{n,a} \cdot \xleftarrow{n,lapp} \cdot \xrightarrow{iS,llet\text{-}in}$ anwendbar.

Um einen vollständigen Satz von Gabeldiagrammen für die interne *llet-in*-Reduktion zu erhalten, muss das Vorgehen aus Beispiel 2.6.5 für alle weiteren no-Reduktionen wiederholt werden, was zu einer komplizierten Fallunterscheidung führt.

Sätze vollständiger Diagramme werden in einer Vielzahl von Arbeiten zum Beweis der Korrektheit von Programmtransformationen verwendet. (Kutzner, 2000; Schmidt-Schauß, 2003; Sabel, 2003; Schmidt-Schauß et al., 2007; Sabel, 2008). Die Argumentation, dass es sich bei einer Menge von Gabel- oder Vertauschungsdiagrammen um einen vollständigen Satz von Diagrammen handelt, basiert in diesen Arbeiten auf Fallanalysen und wird durch typische Beispiele (wie in Beispiel 2.6.5 zu sehen) illustriert. Die Vollständigkeitsbeweise variieren stark in ihrer Ausführlichkeit und Abstraktion: Von sehr ausführlich, wie beispielsweise in Sabel (2003), bis sehr knapp, bzw. ohne Beweis der Vollständigkeit etwa in Schmidt-Schauß et al. (2007).

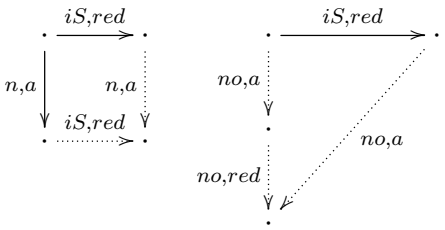
Das generelle Problem der Validierung eines vollständigen Satzes von Gabeldiagrammen von Hand ist, dass im Prinzip alle Überlappungen zwischen einer internen Reduktion und allen möglichen no-Reduktionen berücksichtigt werden müssen. Dies funktioniert nicht, da es unendlich viele Überlappungen gibt, weil der interne Redex beliebig tief in einem Oberflächenkontext stehen kann. Aus diesem Grund beschränkt man sich bei der Fallunterscheidung auf typische Fälle, was dazu führen kann, dass andere vorkommende Fälle übersehen werden.

In dieser Arbeit wird anhand des Λ^{let}-Kalküls untersucht, ob eine Berechnung von vollständigen Sätzen von Gabeldiagrammen möglich ist. Für die Berechnung eines Gabeldiagramms zu einer internen Reduktion *red* müssen zunächst alle Gabelungen $\xleftarrow{n,a} \cdot \xrightarrow{iS,red}$ für alle no-Reduktionen a bestimmt werden. Dazu müssen alle Überlappungen zwischen der no-Reduktion a und der internen Reduktion *red* berechnet werden. Wie wir in Beispiel 2.6.5 gesehen haben, sind manche Überlappungen einfacher zu schließen als andere: Die Fälle, in denen die interne *llet-in*-Reduktion innerhalb einer Meta- oder Kontextvariablen der no-Reduktion auftaucht, sind alle durch das gleiche einfache Gabeldiagramm schließbar. Der Grund hierfür ist, dass in solchen Fällen die Reduktion eines Redex nicht mit der Reduktion des anderen Redex interferiert. Außerdem verbleibt der no-Redex durch die Reduktion des internen Redex in einem Reduktionskontext. Solche Fälle, bei denen der interne Redex innerhalb einer Metavariablen oder einer Kontextvariablen des no-Redex auftaucht, werden als *Schachtelung* (und nicht als Überlappung) von Reduktionen bezeichnet.

Lemma 2.6.7. Für alle no-Reduktionen a und alle Reduktionen *red* gilt: Wenn sich die Gabelung $\xleftarrow{n,a} \cdot \xrightarrow{iS,red}$ durch eine Schachtelung der no-Reduktionen a mit der internen Reduktion *red* ergibt, dann kann das Diagramm geschlossen werden durch $\xrightarrow{iS,red} \cdot \xleftarrow{n,a}$ oder $\xrightarrow{no,red} \cdot \xleftarrow{n,a}$.

Beweis. Die Reduktionen der beiden Redexe interferieren nicht miteinander, was durch Fallunterscheidung über die möglichen no-Reduktionen a gezeigt werden kann. Der zweite Fall $\xrightarrow{no,red} \cdot \xleftarrow{n,a}$ tritt auf, wenn sich der iS-Redex nach der no-Reduktion in einem Reduktionkontext befindet und dadurch zu einem no-Redex wird. □

Bei der Berechnung aller Überlappungen müssen also solche Schachtelungen nicht mit berechnet werden, da sie immer durch Standarddiagramme der Form



geschlossen werden können. Wenn sich die no-Reduktion a und die interne Reduktion red ein Symbol teilen, das keine Meta- oder Kontextvariable ist[4], dann ist nicht direkt offensichtlich, wie das Gabeldiagramm geschlossen werden kann. Bei der Berechnung von Überlappungen konzentrieren wir uns auf solche Fälle.

Um alle Überlappungen einer internen Reduktion red mit no-Reduktionen a zu berechnen, gehe folgendermaßen vor: Die Form der internen Reduktion ist durch die Definition der Reduktionsregeln gegeben. Sei $l(red)$ die linke Seite der entsprechenden Reduktionsregel. Der interne Redex ist eine Instanz dieser linken Seite. Der (n, a)-Redex besitzt keine direkte Entsprechung in der linken Seite einer Reduktionsregel. Um die genaue Form des no-Redex zu bestimmen, muss auf die Definition der Normalordnung und die Struktur von Reduktionskontexten zurückgegriffen werden. Es ist möglich, dass ein (n, a)-Redex mehrere verschiedene Formen hat. Beispielsweise sind für eine $(n, lapp)$-Reduktion die Formen

- $R^-[((\texttt{letrec}\ \{Env\}\ \texttt{in}\ s)\ t)]$, oder
- $(\texttt{letrec}\ \{E'\}\ \texttt{in}\ R^-[((\texttt{letrec}\ \{Env\}\ \texttt{in}\ s)\ t))]$, oder
- $(\texttt{letrec}\ x_1 = R_1^-[((\texttt{letrec}\ \{Env\}\ \texttt{in}\ s)\ t), x_2 = R_2^-[x_1], \ldots, x_j = R_j^-[x_{j-1}]$
  $\texttt{in}\ R^-[x_j])$

für den $(n, lapp)$-Redex möglich, (wobei R^-, R_i^- jeweils schwache Reduktionskontexte sind). Sei ar eine mögliche Variante des (no, a)-Redex. Um alle Überlappungen mit dem internen Redex zu bestimmen, müssen alle Positionen in ar bestimmt werden, an denen der interne Redex vorkommen kann. Dies kann berechnet werden, indem alle Unterausdrücke von ar, die keine Meta- oder Kontextvariablen sind, mit $l(red)$ unifiziert werden. Dabei beschränken wir uns auf Überlappungen in Oberflächenkontexten, d.h. $l(red)$ wird nicht mit einem Unterausdruck von ar unifiziert, der im Rumpf einer Abstraktion vorkommt. Die zu überlappenden Ausdrücke werden vor der Unifikation so umbenannt, dass sie variablendisjunkt sind. Um alle Überlappungen mit der internen Reduktion red zu berechnen, muss dieses Vorgehen für alle no-Reduktionen a und für alle spezifischen (n, a)-Redexe ar durchgeführt werden.[5]

Zur Realisierung dieses Vorgehens wird *Unifikation* für Λ^{let}-Ausdrücke mit Meta-Notation benötigt. In der Literatur (Baader & Snyder, 2001) wird Unifikation im Rahmen von Termen behandelt, und nicht bezüglich Ausdrücken. Aus diesem Grund übersetzen wir Λ^{let}-Ausdrücke mit Metavariablen in ein Kalkül mit Termen, in dessen Rahmen die Unifikation zur Berechnung der Überlappungen durchgeführt werden kann. Bei der Unifikation sind eine Reihe von Problemen zu bewältigen:

[4] Im Beispiel 2.6.5 teilen sich der $(n, lapp)$-Redex und der $(iS, llet\text{-}in)$-Redex das `letrec`-Symbol in $R[((\texttt{letrec}\ \{Env\}\ \texttt{in}\ (\texttt{letrec}\ \{Env'\}\ \texttt{in}\ s'))\ t)]$.

[5] Das Vorgehen zur Berechnung von Überlappungen orientiert sich an der Berechnung kritischer Überlappungen in der Theorie der Termersetzungssysteme (Baader & Nipkow, 1998), (Bezem, Klop, & Vrijer, 2003).

1. Die Metavariablen in Λ^{let} verfügen über Typen: In den Reduktionsregeln stehen s, t für beliebige Ausdrücke, v symbolisiert nur Abstraktionen. Wie diese bei der Unifikation berücksichtigt werden, wird in Kapitel 3 dargelegt.
2. Bindungen innerhalb von `letrec`-Umgebungen sind vertauschbar. Auf die Unifikationsmethoden dazu wird in Kapitel 4 eingegangen.
3. Die (cp)-Reduktionsregeln enthalten Variablenketten beliebiger Längen und stellen somit Regelschemata für eine (abzählbar unendliche) Menge von Regeln dar. Wie das Ketten-Konstrukt bei der Unifikation behandelt werden kann, wird in Kapitel 5 beschrieben.
4. Die zu unifizierenden Ausdrücke enthalten Kontextvariablen. Was zu deren Unifikation notwendig ist, wird in Kapitel 6 dargelegt.

3 Unifikation für Terme mit Sorten

Um kritische Überlappungen zwischen Ausdrücken des Ursprungskalküls Λ^{let}, der über die Konstrukte Abstraktion, Applikation und `letrec` verfügt, zu berechnen, müssen Ausdrücke des Kalküls syntaktisch gleich gemacht werden. Die Methode, die man verwendet, um Ausdrücke miteinander zu identifizieren, heißt *syntaktische Unifikation*. Man betrachtet *Terme*, die rekursiv konstruiert werden aus Variablen und der Anwendung von Funktionssymbolen auf Terme. Ein *Unifikationsproblem* für zwei Terme $s = f(a, x)$ und $t = f(y, b)$ stellt die Frage, ob es möglich ist, die Variablen x, y in t und s durch Terme zu ersetzen, so dass die resultierenden Terme syntaktisch gleich sind. Im Beispiel $f(a, x) =^? f(y, b)$ ist $\{x \mapsto b, y \mapsto a\}$ eine *Lösung* – eine *Substitution*, die s mit t identifiziert, auch *Unifikator* genannt – für das Unifikationsproblem. Der *unifizierte* Term ist $f(a, b)$. Für ein Unifikationsproblem kann es mehrere Lösungen geben. Betrachtet man beispielsweise $f(x, y) =^? f(y, x)$, dann repräsentiert $\tau = \{x \mapsto a, y \mapsto a\}$ eine Lösung. Eine *allgemeinere Lösung* ist allerdings $\sigma = \{x \mapsto y\}$, da $\{x \mapsto a, y \mapsto a\} = \{y \mapsto a\}\sigma$. Man sagt, τ kann durch *Instantiierung* von σ gewonnen werden; σ ist in diesem Fall ein *allgemeinster Unifikator* (mgu). Allgemeinste Unifikatoren vereinfachen die Berechnung von Lösungen eines Unifikationsproblems, da alle Lösungen eines Problems als Instanzen einer allgemeinsten Lösung gewonnen werden können. Aus diesem Grund muss nicht die Menge aller Lösungen eines Unifikationsproblems berechnet werden, sondern die Berechnung der wesentlich kleineren Menge der allgemeinsten Lösungen ist ausreichend.

Ausdrücke des Λ^{let}-Kalküls, die keine Kontextvariablen enthalten (da diese Variablen höherer Ordnung sind und in einem späteren Kapitel behandelt werden), können als Terme aufgefasst werden. Beispielsweise kann der Ausdruck (`letrec` $\{E, x = v\}$ `in` t) geschrieben werden als $letrec(umg(E, bind(x, v)), t)$, wobei $letrec, umg$ und $bind$ jeweils zweistellige Funktionssymbole, E, x, v und t Variablen sind. Der Beispielterm unterliegt allerdings stärkeren Restriktionen als normale Terme erster Ordnung, da die Funktionssymbole und Variablen Typen besitzen. Das *letrec*-Funktionssymbol erwartet als erstes Argument einen Term, der eine Umgebung repräsentiert. Für einen beliebigen anderen Term als erstes Argument, der keine Umgebung darstellt, ist das *letrec*-Funktionssymbol nicht definiert. Auf syntaktischer Ebene bezeichnet man Typen als *Sorten*. Mit Hilfe von Sortensymbolen lässt sich der Definitions- und Wertebereich von Funktionssymbolen syntaktisch beschreiben. Dem *letrec*-Funktionssymbol ist beispielsweise die Sorte

$letrec : Umgebung \to Term \to Term$ zugeordnet. Die Sortensymbole stehen in Relation zueinander, der sogenannten *Subsortrelation*. Ist z.B. *Abstraktion* (verkürzt geschrieben als A) eine Subsorte von $Term$ (T), geschrieben als $A \sqsubset T$, so ist die intuitive Bedeutung, dass alle Terme der Sorte A auch Term der Sorte T sind. Diese Sorten- und Subsorteninformation gilt es bei der Unifikation zu berücksichtigen. Für ein Unifikationsproblem zwischen einer Variablen der Sorte A und einer Variablen der Sorte T, geschrieben als $x_A =^? t_T$, ist $\{x_A \mapsto t_T\}$ keine zu akzeptierende Lösung, da für eine Abstraktionsvariable nur Terme der Sorte R substituiert werden dürfen, wobei R entweder die Sorte A oder eine Subsorte desselben ist. Die Substitution $\{t_T \mapsto x_A\}$ ist eine Lösung, für die diese Bedingung erfüllt ist, da $A \sqsubset T$ gilt. Eine Substitution, die diese Bedingung erfüllt, wird als *wohlsortiert* bezeichnet.

Bei der Unifikation von Termen mit Sorten sollen allgemeinste, wohlsortierte Unifikatoren berechnet werden. Ein Unifikationsalgorithmus wird gewöhnlich als eine Menge von *Transformationsregeln* präsentiert, die ein Unifikationsproblem schrittweise in ein Problem in *gelöster Form* transformieren, aus dem eine allgemeinste, wohlsortierte Lösung direkt abgelesen werden kann. Dabei sollen die Transformationsschritte die Menge der Lösungen des Unifikationsproblems nicht verändern, was als *Vollständigkeit* bezeichnet wird. Eine wichtige Frage, die man sich im Bezug auf den Transformationsprozess stellt ist, ob er für alle eingegebenen Unifikationsprobleme terminiert und wie effizient sich auf diese Art Lösungen berechnen lassen. Außerdem ist von Interesse, ob alle Unifikationsprobleme immer eine eindeutige Lösung (d.h. einen einzelnen mgu), oder eine endliche Menge von nicht vergleichbaren Lösungen bzw. unendlich viele Lösungen besitzen.

Das vorliegende Kapitel beschäftigt sich mit den oben skizzierten Themen, um einen Teil des Problems der Berechnung von Überlappungen zu lösen. Dabei ist es folgenderma"sen aufgebaut:

Abschnitt 3.1 enthält einige wichtige Definitionen für im Text häufig verwendete Begriffe aus dem Bereich der Ordnungsrelationen.

In Abschnitt 3.2 werden die Begriffe Term und Substitution für den Fall, dass keine Sorten vorliegen, eingeführt.

Im daran anschließenden Teil 3.3 wird zuerst beschrieben, wie Signaturen um Sorten bereichert werden (3.3.1). Dabei wird auch angegeben, wie eine Signatur Σ^{let} aussieht, die Ausdrücke aus dem Kalkül Λ^{let} als Terme mit Sorten darstellen kann. Darauf folgt in Abschnitt 3.3.2 die Definition von Termen, die Sorten aus einer gegebenen Signatur mit Sorten berücksichtigen. Außerdem wird beschrieben, wie man Ausdrücke des Λ^{let}-Kalküls in entsprechende Terme mit Sorten übersetzen kann. Abschließend wird in Teil 3.3.3 darauf eingegangen, was es für Substitutionen bedeutet, einer gegebenen Struktur von Sorten zu entsprechen.

Die Unifikation von Termen mit Sorten ist das Thema von Abschnitt 3.4. Zuerst werden grundlegende Begriffe wie *Instantiierungs-Quasiordnung* und *Unifikations-*

problem eingeführt. Anschließend wird beschrieben, wie Unifikationsprobleme durch wiederholte Anwendung von Transformationen gelöst werden können (3.4.1). Ein wichtiger Teil dieses Abschnittes beschäftigt sich mit den Eigenschaften des Transformationsprozesses: Es wird gezeigt, dass er vollständig ist und terminiert. Dass sich die Menge aller Lösungen für eine bestimmt interessante Klasse von Unifikationsproblemen (nämlich die Unifikationsprobleme über der Signatur Σ^{let}) als Instanz eines einzelnen mgu repräsentieren lässt, wird in Teil 3.4.2 gezeigt. Im abschließenden Abschnitt 3.4.3 wird gezeigt, dass für diese Klasse von Unifikationsproblemen ein mgu effizient berechnet werden kann (in Quasi-Linear-Zeit).

3.1 Ordnungsrelationen

In den folgenden Abschnitten werden an verschiedenen Stellen *Ordnungsrelationen* verwendet. Die grundlegenden Definitionen werden hier kurz angegeben.

Definition 3.1.1. Sei $\triangleright \subseteq A \times A$ eine binäre Relation auf einer Menge A. Die Relation $\triangleright$ ist

reflexiv , gdw. $\forall x \in A : x \triangleright x$,

irreflexiv , gdw. $\forall x \in A : \neg(x \triangleright x)$,

transitiv , gdw. $\forall x, y, z \in A : x \triangleright y \ \wedge \ y \triangleright z \Rightarrow x \triangleright z$,

symmetrisch , gdw. $\forall x, y \in A : x \triangleright y \Rightarrow y \triangleright x$,

antisymmetrisch , gdw. $\forall x, y \in A : x \triangleright y \ \wedge \ y \triangleright x \Rightarrow x = y$.

Definition 3.1.2 (Äquivalenzrelation)**.** Eine *Äquivalenzrelation* ist eine reflexive, transitive und symmetrische Relation: $\sim \subseteq A \times A$. Eine Äquivalenzrelation erzeugt Äquivalenzklassen auf der Menge A: $[a]_\sim := \{a' \in A \mid a \sim a'\}$ für alle $a \in A$ und eine Faktormenge $A/_\sim := \{[a]_\sim \mid a \in A\}$. Äquivalenzklassen partitionieren eine Menge A: Zwei Äquivalenzklassen $[a]_\sim$ und $[b]_\sim$ sind entweder identisch (wenn $a \sim b$ gilt) oder disjunkt (wenn $a \sim b$ nicht gilt).

Definition 3.1.3 (Quasiordnung)**.** Eine reflexive, transitive Relation $\lesssim$ auf einer Menge A wird als *Quasiordnung* bezeichnet. Das Paar $(A, \lesssim)$ wird quasi-geordnete Menge genannt.

Definition 3.1.4 (Partialordnung)**.** Eine antisymmetrische Quasiordnung $\leq$ wird als *Partialordnung* bezeichnet. Das Paar $(A, \leq)$ wird partiell geordnete Menge genannt.

Definition 3.1.5 (Strikte Ordnung)**.** Eine *strikte Ordnung* $<$ ist eine transitive und irreflexive Relation.

Die Schreibweise $x \gtrsim y$ wird verwendet für $y \lesssim x$ und $x \not\lesssim y$ ist definiert als $\neg(x \lesssim y)$ (analoge Definitionen werden für $\leq$ und $<$ getroffen).

Es gelten folgende Beziehungen:

- Jede Quasiordnung $\lesssim$ induziert eine strikte Ordnung
$$x < y :\Leftrightarrow x \lesssim y \ \wedge \ \neg(y \lesssim x),$$
der sogenannte *strikte Anteil* von $\lesssim$.
- Jede Quasiordnung $\lesssim$ induziert eine Äquivalenzrelation
$$x \sim y :\Leftrightarrow x \lesssim y \ \wedge \ y \lesssim x.$$
- Jede Quasiordnung $\lesssim$ induziert eine Partialordnung auf $A/_\sim$
$$[x]_\sim \leq [y]_\sim :\Leftrightarrow x \lesssim y.$$
- Jede Partialordnung $\leq$ induziert eine strikte Ordnung
$$x < y :\Leftrightarrow x \leq y \ \wedge \ y \neq x.$$
- Jede strikte Ordnung $<$ induziert eine Partialordnung
$$x \leq y :\Leftrightarrow x < y \ \vee \ y = x.$$

Definition 3.1.6. Sei $(A, \leq)$ eine partiell geordnete Menge und $M \subseteq A$. Es wird definiert:

- $min(M) := \{m \in M \mid n \not< m \ \forall n \in M\}$, die Menge der *minimalen* Elemente von M.
- $max(M) := \{m \in M \mid m \not< n \ \forall n \in M\}$, die Menge der *maximalen* Elemente von M.
- $ls(M) := m \in M$, so dass $m \leq n \ \forall n \in M$, das *kleinste* Element von M.
- $lbs(M) := \{l \in A \mid l \leq n \ \forall n \in M\}$, die Menge der *unteren Schranken* von M.
- $glb(M) := max(lbs(M))$, die Menge der *größten unteren Schranken* von M.

Die Begriffe *größtes* Element, *obere Schranken* und *kleinste obere Schranken* (kurz *lub*) werden dual definiert (durch Umkehren der Ordnungsrelation). Für die Menge der unteren Schranken zweier Elemente $m, n \in A$ schreiben wir $lbs(m, n)$ anstatt $lbs(\{m, n\})$ (ebenso für $glb(m, n)$).

Wenn $glb(M)$ und $lub(M)$ für alle endlichen Teilmengen M von A existieren und genau ein Element enthalten, dann ist $(A, \leq)$ ein *Verband*. Wenn $lub(M)$ (bzw. $glb(M)$) für alle endlichen Teilmengen M von A existiert und immer einelementig ist, aber $glb(M)$ (bzw. $lub(M)$) nicht immer, dann ist $(A, \leq)$ ein *oberer* (bzw. *unterer*) *Halbverband*. Eine äquivalente Charakterisierung eines oberen Halbverbandes für eine endliche Mengen A ist:

1. $\forall a, b \in A: \ |lbs(a,b)| \geq 1 \Rightarrow |glb(a,b)| = 1$. D.h. alle $a, b \in A$, die gemeinsame untere Schranken besitzen, haben eine eindeutige größte untere Schranke, und
2. A besitzt ein größtes Element.

Definition 3.1.7. Sei $(A, \leq)$ eine partiell geordnete Menge. Die Relation $\leq$ ist

linear , gdw. $\forall x, y \in A : x \neq y \Rightarrow x \leq y \vee y \leq x$.

fundiert , gdw. es keine unendlichen, echt absteigenden Ketten $x_1 \geq x_2 \geq x_2 \geq \ldots$ gibt (d.h. jede nichtleere Teilmenge von A ein minimales Element besitzt).

Für eine lineare Partialordnung stimmen minimales und kleinstes Element überein. Sei $\leq$ eine lineare, fundierte Partialordnung auf der Menge A, dann besitzt A ein kleinstes Element $ls(A)$.

3.2 Terme und Substitutionen ohne Sorten

Zunächst werden die Begriffe Signatur, Term und Substitution für den Fall ohne Sorten eingeführt. Die Notation $\overline{\ }$(Überstrich) wird im weiteren Verlauf verwendet, um zu kennzeichnen, dass es sich um Objekte handelt, denen keine Sorten zugeordnet sind, speziell im Fall von Signaturen ohne Sorten ($\overline{\Sigma}$), die später zur Definition von Signaturen mit Sorten herangezogen werden. Die Darstellung orientiert sich an Baader und Nipkow (1998) sowie Baader und Snyder (2001).

Um zu verdeutlichen, welche Funktionssymbole in einem bestimmten Kontext zur Verfügung stehen und welche Stelligkeiten sie besitzen, wird eine Menge von Funktionssymbolen definiert.

Definition 3.2.1 (Signatur ohne Sorten)**.** Eine *Signatur* $\overline{\Sigma}$ ist eine Menge von *Funktionssymbolen*. Jedem Funktionssymbol $f \in \overline{\Sigma}$ ist eine Zahl $n \in \mathbb{N}$ zugeordnet, die *Stelligkeit* von f. Für $n \geq 0$ wird die Menge der n-stelligen Elemente von $\overline{\Sigma}$ mit $\overline{\Sigma}^n$ bezeichnet. Die Elemente aus $\overline{\Sigma}^0$ werden als *Konstanten* bezeichnet.

Terme werden gebildet aus Variablen oder durch die Anwendung eines Funktionssymbols auf Terme.

Definition 3.2.2 ($\overline{\Sigma}$-Term)**.** Sei $\overline{\Sigma}$ eine Signatur und X eine (abzählbar unendliche) Menge von *Variablen*, so dass $\overline{\Sigma} \cap X = \emptyset$. Die Menge $T(\overline{\Sigma}, X)$ aller $\overline{\Sigma}$-*Terme* über Variablen X ist induktiv definiert durch

1. $X \subseteq T(\overline{\Sigma}, X)$ (jede Variable ist ein Term),
2. für alle $n \geq 0$, alle $f \in \overline{\Sigma}^n$ und alle $t_1, \ldots, t_n \in T(\overline{\Sigma}, X)$ ist $f(t_1, \ldots, t_n) \in T(\overline{\Sigma}, X)$ (die Anwendungen von Funktionssymbolen auf Terme ergibt Terme).

Die Struktur eines Terms kann in natürlicher Weise als Baum dargestellt werden, wobei Funktionssymbole als Knoten und Argumente einer Funktion als Kinder des Funktionsknotens repräsentiert werden. Abbildung 3.1 zeigt einen Beispielterm t in seiner Baumdarstellung. Dort ist zu sehen, wie die Knoten des Baumes in einer Links-Rechts-Ordnung durchnummeriert werden können. Über diese Nummerierung ist es möglich, sich auf einzelne Symbole an bestimmten Positionen eines Terms oder auf Subterme zu beziehen. Im Beispiel steht an Position ϵ das Funktionssymbol f und an Position 1 das Konstantensymbol e, das erste Argument von f. Der Subterm von t an Position 21 ist $g(y)$. Formal können Begriffe wie die Positionen oder die Größe eines Terms durch Induktion über die Struktur von Termen definiert werden.

Figure 3.1. Baumdarstellung von $t = f(e, f(g(y), e))$.

Definition 3.2.3. Sei $\overline{\Sigma}$ eine Signatur, X eine Menge von Variablen, die disjunkt ist zu $\overline{\Sigma}$ und $s, t \in T(\overline{\Sigma}, X)$.

1. Die Menge der *Positionen* eines Term s ist die Menge $Pos(s)$ von Worten über dem Alphabet der natürlichen Zahlen induktiv definiert durch:
   - Wenn $s = x \in X$, dann ist $Pos(s) := \{\epsilon\}$, wobei ϵ das leere Wort bezeichnet.
   - Wenn $s = f(t_1, \ldots, t_n)$, dann
   $$Pos(s) := \{\epsilon\} \cup \bigcup_{i=1}^{n} \{ip \mid p \in Pos(s_i)\}.$$

   Die Position ϵ wird *Wurzelposition* des Terms s genannt und das Funktionssymbol oder die Variable an dieser Position heißt *Wurzelsymbol*. Die *Präfix-Ordnung*, definiert durch
   $$p \leq q \text{ gdw. es gibt } p' \text{ so dass } pp' = q,$$
   ist eine partielle Ordnung auf Positionen. Positionen p, q werden *parallel* genannt $(p||q)$,, gdw. p und q nicht vergleichbar bezüglich $\leq$ sind. Die Position p befindet sich *oberhalb* der Position q, wenn $p \leq q$ und p ist *strikt oberhalb* von q, wenn gilt $p < q$ (*unterhalb* wird analog definiert).

2. Die *Größe* $|s|$ eines Term s ist die Kardinalität von $Pos(s)$.
3. Sei $p \in Pos(s)$, dann wird der *Subterm* von s *an Position* p bezeichnet durch $s|_p$, definiert durch Induktion über die Länge von p:
   - $s|_\epsilon := s$,
   - $f(s_1, \ldots, s_n)|_{iq} := s_i|_q$.
4. Sei $p \in Pos(s)$, dann wird mit $s[t]_p$ der Term bezeichnet, der aus s entsteht, durch die *Ersetzung des Subterms an Position* p durch t:
   - $s[t]_\epsilon := t$
   - $f(s_1, \ldots, s_n)[t]_{iq} := f(s_1, \ldots, s_i[t]_q, \ldots, s_n)$
5. Mit $Var(s)$ wird die Menge der in Term s *vorkommenden Variablen* bezeichnet:
   $$Var(s) := \{x \in X \mid es\ gibt\ p \in Pos(s),\ so\ dass\ s|_p = x\}$$
   Eine Position $p \in Pos(s)$ wird *Variablen-Position* genannt, wenn $t|_p$ eine Variable ist.

Für den Term t aus obigem Beispiel haben wir $Pos(t) = \{\epsilon, 1, 2, 21, 211, 22\}$, $t|_{21} = g(y)$, $t[e]_2 = f(e, e)$, $Var(t) = \{y\}$ und $|t| = 6$. Die Größe von t ist gleich der Anzahl der Knoten in der Baumdarstellung von t.

Der Hauptunterschied zwischen Konstantensymbolen und Variablen besteht darin, dass Variablen durch Substitutionen ersetzt werden können.

Definition 3.2.4 (Substitution)**.** Sei $\overline{\Sigma}$ eine Signatur und X eine abzählbar unendliche Menge von Variablen. Eine $T(\overline{\Sigma}, X)$-*Substitution* (oder einfach nur Substitution, wenn die Menge der Terme irrelevant ist oder aus dem Kontext hervorgeht) ist eine Funktion
$$\sigma : X \to T(\overline{\Sigma}, X),$$
so dass die Menge $\{x \in X \mid \sigma(x) \neq x\}$ endlich ist. Diese endliche Menge von Variablen, die unter σ nicht auf sich selbst abgebildet werden, bezeichnet man als *Domain* von σ:
$$Dom(\sigma) := \{x \in X \mid \sigma(x) \neq x\}.$$
Die *Range* von σ ist
$$Ran(\sigma) := \{\sigma(x) \mid x \in Dom(\sigma)\}$$
und die *Variablen-Range* von σ enthält die in $Ran(\sigma)$ vorkommenden Variablen:
$$VRan(\sigma) := \bigcup_{x \in Dom(\sigma)} Var(\sigma(x)).$$
Ist $Dom(\sigma) = \{x_1, \ldots, x_n\}$, kann man σ schreiben, indem man die Menge von Variablen-Termbindungen angibt, die σ definieren (da $Dom(\sigma)$ endlich ist).
$$\sigma = \{x_1 \mapsto \sigma(x_1), \ldots, x_n \mapsto \sigma(x_n)\}.$$

Man sagt σ *instantiiert* x, wenn $x \in Dom(\sigma)$. Die Menge aller $T(\overline{\Sigma}, X)$-Substitutionen wird bezeichnet durch $Sub(T(\overline{\Sigma}, X))$ oder einfach $Sub_{\overline{\Sigma}}$.

Jede $T(\overline{\Sigma}, X)$-Substitution kann folgendermaßen zu einer Abbildung $\widehat{\sigma} : T(\overline{\Sigma}, X) \to T(\overline{\Sigma}, X)$ *erweitert* werden:

- Für $x \in X$, $\widehat{\sigma}(x) := \sigma(x)$ und
- für jeden nicht Variablenterm $s = f(s_1, \ldots, s_n)$ ist $\widehat{\sigma}(s) := f(\widehat{\sigma}(s_1), \ldots, \widehat{\sigma}(s_n))$.

Die Anwendung einer Substitution σ auf einen Term ersetzt gleichzeitig alle Vorkommen von Variablen durch die jeweiligen σ-Bilder. Sei beispielsweise $s = f(e, x)$ und $t = f(y, f(x, y))$ sowie $\sigma = \{x \mapsto g(y), y \mapsto e\}$, dann ist $\widehat{\sigma}(s) = f(e, g(y))$ und $\widehat{\sigma}(t) = f(e, f(g(y), e))$.

Die *Identitätssubstitution*, die alle Variablen auf sich selbst abbildet (d.h. $\sigma(x) = x$ für alle $x \in X$), wird mit Id bezeichnet.

Zwei Substitutionen σ und τ sind *äquivalent* bezüglich einer Menge von Variablen $W \subseteq X$, geschrieben als $\sigma = \tau[W]$, wenn gilt $\sigma(x) = \tau(x)$ für alle $x \in W$. Ist $W = X$ dann schreibt man $\sigma = \tau$.

Die *Einschränkung* einer Substitution σ auf eine Menge von Variablen X, geschrieben als $\sigma|_X$, ist definiert als $\sigma|_X(x) := \sigma(x)$, wenn $x \in X$, sonst $\sigma|_X x := x$.

Die *Komposition* $\sigma\tau$ zweier Substitutionen σ und τ ist definiert als

$$\sigma\tau(x) := \widehat{\sigma}(\tau(x)).$$

Sind zwei Substitutionen $\sigma = \{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}$ und $\tau = \{y_1 \mapsto t_1, \ldots, y_m \mapsto t_m\}$ als Mengen von Variablen-Termbindungen gegeben, kann ihre Komposition $\sigma\tau$ folgendermaßen konstruiert werden: $\sigma\tau = \{y_1 \mapsto \widehat{\sigma}t_1, \ldots, y_m \mapsto \widehat{\sigma}t_m\} \cup \{x_i \mapsto s_i \mid x_i \in Dom(\sigma) - Dom(\tau)\}$. Die Komposition von Substitutionen ergibt wieder eine Substitution und ist assoziativ, d.h. es gilt $\sigma(\tau\delta) = (\sigma\tau)\delta$.

Zur Vereinfachung der Notation wird üblicherweise nicht unterschieden zwischen Substitutionen $\sigma : X \to T(\overline{\Sigma}, X)$ und ihrer Erweiterung $\widehat{\sigma} : T(\overline{\Sigma}, X) \to T(\overline{\Sigma}, X)$. Im weiteren Verlauf wird σ verwendet, um die Substitution und ihre Erweiterung auf Terme zu bezeichnen. Außerdem wird die Anwendung einer Substitution σ auf einen Term t häufig ohne Klammern geschrieben, σt anstatt $\sigma(t)$.

Ein Term t wird *Instanz* eines Terms s genannt, wenn es eine Substitution σ gibt, so dass $t = \sigma s$. Man schreibt $s \lesssim t$.

Eine Substitution σ wird *idempotent* genannt, wenn gilt $\sigma\sigma = \sigma$, was genau dann der Fall ist, wenn $Dom(\sigma) \cap VRan(\sigma) = \emptyset$. Eine idempotente Substitution bestehend aus n Variablen-Termbindugen, kann in Kompositionen von n Substitutionen, jede aus lediglich einer Bindung bestehend, zerlegt werden:

$$\sigma = \{x_1 \mapsto t_1, x_2 \mapsto t_2, \ldots, x_n \mapsto t_n\} = \{x_1 \mapsto t_1\}\{x_2 \mapsto t_2\} \ldots \{x_n \mapsto t_n\}.$$

Eine solche Komposition wird als *trianguläre Form* bezeichnet und geschrieben als:

$$[x_1 \mapsto t_1; x_2 \mapsto t_2; \ldots; x_n \mapsto t_n].$$

Eine *Variablenumbenennung* ist eine Substitution $\rho \in Sub_{\overline{\Sigma}}$, so dass gilt

1. ρ ist injektiv auf $Dom(\rho)$, d.h. $\forall x_1, x_2 \in Dom(\rho) : \rho x_1 = \rho x_2 \Rightarrow x_1 = x_2$ und
2. $Ran(\rho)$ besteht nur aus Variablen.

Sei $\rho = \{x_1 \mapsto y_1, \ldots, x_n \mapsto y_1\}$ eine Variablenumbenennung, dann ist die *Umkehrung* von ρ definiert als $\rho^- := \{y_1 \mapsto x_1, \ldots, y_n \mapsto x_n\}$. Folgendes technisches Lemma über Variablenumbenennungen wird später benötigt.

Lemma 3.2.5. Sei ρ eine Variablenumbenennung. Es gilt

1. ρ^- ist eine Variablenumbenennung
2. $Dom(\rho) = Ran(\rho^-)$,
3. $Dom(\rho^-) = Ran(\rho)$,
4. $\rho\rho^- = \rho$,
5. $\rho^-\rho = \rho^-$,
6. $\rho^-\rho = Id[Dom(\rho)]$,
7. $(\rho^-)^- = \rho$.

Beweis. 1, 2, 3, und 7 sind offensichtlich. Betrachte 5. Sei $\rho = \{x_1 \mapsto y_1, \ldots, x_n \mapsto y_1\}$ eine Variablenumbenennung und deren Umkehrung sei $\rho^- := \{y_1 \mapsto x_1, \ldots, y_n \mapsto x_n\}$. Konstruiere $\rho^-\rho$ wie oben angegeben:

$$\begin{aligned}\rho^-\rho &= \{x_1 \mapsto \rho^- y_1, \ldots, x_n \mapsto \rho^- y_n\} \cup \{y_1 \mapsto x_1, \ldots, y_n \mapsto x_n\} \\ &= \{x_1 \mapsto x_1, \ldots, x_n \mapsto x_n\} \cup \{y_1 \mapsto x_1, \ldots, y_n \mapsto x_n\} \\ &= Id \cup \{y_1 \mapsto x_1, \ldots, y_n \mapsto x_n\} \\ &= \rho^-\end{aligned}$$

Damit ist auch 6 einsichtig und 4 wird analog gezeigt. □

3.3 Signaturen, Terme und Substitutionen mit Sorten

Es wird nun der Fall betrachtet, dass Terme über Sorten verfügen, die auf syntaktischer Ebene den Definitions- und Wertebereich von Funktionssymbolen beschreiben. Dazu wird zu Signaturen eine Menge von Sortensymbolen hinzugefügt, zusammen mit Mechanismen, um die Sorten von Termen zu beschreiben. Die Darstellung orientiert sich an Schmidt-Schauß (1989a).

3.3.1 Signaturen mit Sorten

Signaturen mit Sorten benötigen eine zusätzliche Symbolmenge:

- S_Σ ist die nicht leere Menge von *Sortensymbolen*. Elemente werden mit R, S bezeichnet.

Funktionssymbole (und beliebige andere Terme) erhalten Sorten durch Deklarationen.

Definition 3.3.1 (Deklarationen)**.** Eine *Termdeklaration* ist ein Paar (t, S), geschrieben als $t : S$, wobei t kein Variablenterm $t \in T(\Sigma, X)$ und $S \in S_\Sigma$ ein Sortensymbol ist. Ist t von der Form $f(x_1, \ldots, x_n)$ und $x_i \in X$ sind verschiedene Variablen, dann nennt man $t : S$ *Funktionsdeklaration*. Ist t eine Konstante, dann bezeichnet man $t : S$ als *Konstantendeklaration*. Sonst wird $t : S$ (ordnungsgemäße) Termdeklaration genannt. Eine *Subsortendeklaration* hat die Form $R \subseteq S$, wobei R und S Sortensymbole sind. In diesem Fall wird R als *Subsorte* von S bezeichnet.

Termdeklarationen werden verwendet, um Signaturen mit Sorten zu definieren.

Definition 3.3.2 (Signatur mit Sorten)**.** Eine *Signatur mit Sorten* Σ besteht aus:

1. einer Signatur ohne Sorten $\overline{\Sigma}$,
2. einer Menge von Sortensymbolen S_Σ,
3. einer Funktion $\mathbb{S} : X \to S_\Sigma$, so dass für alle Sortensymbole $S \in S_\Sigma$ abzählbar unendlich viele Variablen $x \in X$ existieren mit $\mathbb{S}(x) = S$,
4. einer Menge von Term- und Subsortendeklarationen.

Zur Definition einer Signatur mit Sorten ist es in der Regel ausreichend, Term- und Subsortendeklarationen zusammen mit Angaben zur Sorte der Variablen, die in Termdeklarationen vorkommen, anzugeben. Funktionsdeklarationen $f(x_1, \ldots, x_n) : S$ werden abgekürzt geschrieben als $f : S_1 \to \cdots \to S_n \to S$, wobei S_i die Sorte der Variablen x_i ist. Die Information, dass eine Variable x oder Konstante c die Sorte S hat, wird geschrieben als $x : S$, $c : S$ bzw. x_S, c_S.

Die Funktion $\mathbb{S} : X \to S_\Sigma$ partitioniert die Menge der Variablen X in Teilmengen X_S von Variablen der Sorte S.

Die Subsortendeklarationen der Form $R \subseteq S$ einer Signatur bilden die Basis für eine Quasiordnung auf der Menge der Sorten, die formal folgendermaßen definiert ist:

Definition 3.3.3 (Subsorten-Quasiordnung)**.** Sei Σ eine Signatur mit Sorten und sei $\subseteq$ die Relation, die definiert ist durch den reflexiven, transitiven Abschluss der Subsortendeklarationen aus Σ. Dann ist $\subseteq$ eine Quasiordnung auf der Menge der

Sortensymbole S_Σ. Die Quasiordnung wird als *Subsorten-Quasiordnung* bezeichnet. Das Paar $(S_\Sigma, \sqsubseteq_\Sigma)$ wird *Sortenstruktur* genannt.

Geht aus dem Kontext klar hervor, auf welche Signatur Σ sich die Subsorten Quasiordnung $\sqsubseteq_\Sigma$ bezieht, wird ein einfach $\sqsubseteq$ geschrieben. Der strikte Anteil von $\sqsubseteq_\Sigma$ wird durch $\sqsubset$ bezeichnet (siehe Abschnitt 3.1). Unter dem reflexiven, transitiven Abschluss $\sqsubseteq_\Sigma$ von $\sqsubseteq$ gilt $S \sqsubseteq_\Sigma S$ für alle $S \in S_\Sigma$.

Beispiel 3.3.4. Die Menge der Sortensymbole S^{let}_Σ, die verwendet werden, um das Kalkül Λ^{let} mit den Konstrukten Abstraktion, Applikation und `letrec` in ein Kalkül erster Ordnung mit Sorten zu transformieren, sei folgendermaßen definiert:

T repräsentiert die Sorte für Terme (Ausdrücke des Ursprungskalküls),

A repräsentiert die Sorte für Abstraktionen,

V repräsentiert die Sorte für Variablen,

U repräsentiert die Sorte für `letrec`-Umgebungen,

B repräsentiert die Sorte für `letrec`-Bindungen,

Es werden keine Sortensymbole für die Konstrukte Applikation und `letrec` des Ursprungskalküls definiert, da kein Bedarf besteht zwischen Applikationen (bzw. `letrec`-Ausdrücken) und beliebigen Ausdrücken zu unterscheiden. Alle Variablen, die in linken Seiten von Reduktionsregeln des Λ^{let}-Kalküls auftauchen, lassen sich einer dieser Sorten zuordnen.

Die Signatur Σ^{let} sei folgendermaßen definiert:

$$
\begin{aligned}
\Sigma^{let} = \{\ & \textbf{Subsortendeklarationen}: \\
& A \sqsubset T,\ V \sqsubset T, B \sqsubset U, \\
& \textbf{Funktionsdeklarationen}: \\
& abs : V \to T \to A \qquad (\mathsf{Abstraktion}), \\
& app : T \to T \to T \qquad (\mathsf{Applikation}), \\
& letrec : U \to T \to T \qquad (\texttt{letrec}), \\
& umg : B \to U \to U \qquad (\texttt{letrec} - \mathsf{Umgebung}), \\
& bind : V \to T \to B \qquad (\texttt{letrec} - \mathsf{Bindung}) \qquad \}
\end{aligned}
$$

Zum jetzigen Zeitpunkt werden Kontextvariablen und Umgebungen aus der Betrachtung ausgeklammert, da diese Konstrukte des Ursprungskalküls Λ^{let} bestimmte Unifikationsmethoden benötigen, auf die hier noch nicht eingegangen wird (E-Unifikation und Unifikation höherer Ordnung). Kontextvariablen und Umgebungen werden in späteren Kapiteln behandelt. Insbesondere das Funktionssymbol *umg* dient hier nur als Platzhalter, und wird in Kapitel 4 ausführlich behandelt.

In den Subsortendeklarationen wird die Variablen-Sorte V als Subsorte der Term-Sorte T deklariert. Diese Subsortenbeziehung soll das Verhalten des Ursprungskalküls nachbilden, in dem Variablen Ausdrücke darstellen, aber ein beliebiger Ausdruck in der Regel keine Variable ist. Ebenso handelt es sich bei Abstraktionen um Ausdrücke; ein beliebiger Ausdruck muss aber im Allgemeinen keine Abstraktion sein. Aus diesem Grund wird in der Subsortendeklaration auch der strikte Anteil $\sqsubset$ von $\tilde{\sqsubseteq}$ verwendet, da Abstraktionen (bzw. Variablen) eine echte Teilmenge der Ausdrücke darstellen.

Auf den `letrec`-Umgebungsoperator *umg* und *bind* wird in einem späteren Kapitel genauer eingegangen. Hier sei nur darauf hingewiesen, dass die Sorten B, U nicht vergleichbar sind bezüglich $\tilde{\sqsubseteq}_\Sigma$ mit den Sorten V, A, T. Dieses Verhalten ist erwünscht, da Terme der Sorte B, U Bestandteile von `letrec`-Umgebungen repräsentieren, die selbst keine eigenständigen Ausdrücke des Λ^{let}-Kalküls sind.

Eine Signatur Σ mit Sorten kann klassifiziert werden in Abhängigkeit der Kardinalität der Menge der Sortensymbole $|S_\Sigma|$ und der Art der Termdeklarationen, die sie enthält.

Definition 3.3.5. Sei Σ eine Signatur mit Sorten. Σ ist

- *endlich*, wenn ihre Beschreibung endlich ist, d.h. wenn die Menge der Sortensymbole, die Menge der Funktionssymbole, die Term- und Subsortendeklarationen endlich sind.
- *one-sorted*, wenn $|S_\Sigma| = 1$, d.h. es nur einen Sorte gibt.
- *many-sorted*, wenn $|S_\Sigma| > 1$ und es keine Subsortendeklarationen gibt.
- *order-sorted*, wenn $|S_\Sigma| > 1$ und es Subsortendeklarationen gibt.
- *elementar*, wenn alle Termdeklarationen Funktionsdeklarationen sind.
- *einfach*, wenn sie elementar ist und für alle Funktionssymbole existiert genau eine Funktionsdeklaration.

Diese Eigenschaften einer Signatur Σ haben einen erheblichen Einfluss auf die Unifikation für Σ-Terme, wie wir im Abschnitt 3.4.1 sehen werden.

Korollar 3.3.6. Die in Beispiel 3.3.4 definierte Signatur Σ^{let} ist *endlich*. Sie ist *elementar*, da alle Term-Deklarationen Funktionsdeklarationen sind. Außerdem ist sie *einfach*, da sie elementar ist und für alle Funktionssymbole gibt es genau eine Funktionsdeklaration.

Da wir vor allem daran interessiert sind Unifikationsprobleme für Terme über der speziellen Signatur Σ^{let} zu lösen und es sich bei dieser um eine endliche und einfache Signatur handelt, wird im weiteren Verlauf besonderes Augenmerk auf endliche und einfache Signaturen gerichtet.

3.3.2 Wohlsortierte Terme

Signaturen mit Sorten werden verwendet, um Terme zu definieren, die diese Sorten berücksichtigen. Für ein Sortensymbol $S \in S_\Sigma$ definiert man die Menge der Terme mit Sorte S.

Definition 3.3.7 (Σ-Term der Sorte S)**.** Sei Σ eine Signatur mit Sorten, X eine Menge von Variablen, so dass $\Sigma \cap X = \emptyset$. Die Menge $T(\Sigma, S, X)$ aller Σ-*Terme der Sorte* S über X ist induktiv definiert durch

1. $x \in T(\Sigma, S, X)$, wenn $\mathbb{S}(x) \sqsubseteq S$.
2. $t \in T(\Sigma, S, X)$, wenn $t : R \in \Sigma$ und $R \sqsubseteq S$.
3. $\{x \mapsto r\}t \in T(\Sigma, S, X)$, wenn $t \in T(\Sigma, S, X)$, $r \in T(\Sigma, R, X)$ und $R \sqsubseteq \mathbb{S}(x)$.

D.h. alle Variablen der Sorte kleiner gleich S sind Terme der Sorte S (1). Und alle Terme der Signatur mit Sorte kleiner gleich S, die den Termdeklarationen von Σ entnommen werden können, sind Terme der Sorte S (2). Ein neuer Term t' der Sorte S kann konstruiert werden aus einem Term t der Sorte S durch gleichzeitiges Ersetzen einer Variablen in t durch einen Term der Sorte kleiner gleich der Sorte der Variablen (3).

Aus der Definition folgt sofort

Korollar 3.3.8. Sei Σ eine Signatur mit Sorten.

1. Für alle Sorten $R, S \in S_\Sigma$ gilt: $R \sqsubset S$ impliziert $T(\Sigma, R, X) \subseteq T(\Sigma, S, X)$.
2. Für Variablen gilt: $x \in T(\Sigma, S, X) \Leftrightarrow \mathbb{S}(x) \sqsubseteq S$.

Beispiel 3.3.9. Aus Definition 3.3.7 ergeben sich unter der in Beispiel 3.3.4 definierten Signatur Σ^{let} und der Variablenmenge X folgende Terme:

- Terme mit Sorte V: Da es keine Funktionsdeklaration der Sorte V gibt, bestehen die Terme der Sorte V lediglich aus Variablen der Sorte V: $x_V, y_V, z_V \ldots$
- Terme der Sorte A: Da es außer dem Funktionssymbol *abs* kein Funktionssymbol der Sorte A in der Signatur gibt, und A keine Subsorts besitzt, sind alle Terme der Sorte A von folgender Form:
  - entweder Abstraktionsvariablen $v_A, w_A, \ldots,$
  - oder Terme bei denen das Funktionssymbol *abs* an der Wurzelposition steht.
- Terme der Sorte T: Aus $V, A \sqsubset T$ folgt nach Korollar 3.3.8, dass alle Terme der Sorte V und Terme der Sorte A auch Terme der Sorte T sind. D.h. alle Variablen $x_V, y_V, \ldots$ und $v_A, w_A, \ldots$ und Terme mit *abs* als Wurzelsymbol

sind Terme der Sorte T. Nach Definition 3.3.7 sind außerdem Terme der Sorte T von folgender Form:

1. Termvariablen $s_T, t_T, u_T \ldots$ sind Terme der Sorte T.
2. Terme t der Signatur mit $t : R$ und $R \sqsubseteq_{\Sigma} T$, d.h. $app(s_T, t_T)$ und $letrec(e_U, s_T)$ sind Terme der Sorte T.
3. Terme der Sorte T, deren Variablen durch Terme mit einer kleineren oder gleichen Sorte ersetzt werden. Beispiele dafür sind: $app(abs(x_V, s_t), t_T)$ und $letrec(d_U, letrec(e_U, s_T))$.

Terme der Sorte B oder U werden analog gebildet.

Die Abbildung $[\![\ ]\!] : \Lambda^{let} \to T(\Sigma^{let}, X)$, die Ausdrücke (in Meta-Notation) des Ursprungskalküls Λ^{let} (ausgenommen `letrec`-Umgebungen und Kontextvariablen) übersetzt in Σ^{let}-Terme erster Ordnung mit Sorten, ist folgendermaßen definiert:

$$\begin{aligned}
[\![x]\!] &= x_V \\
[\![v]\!] &= v_A \\
[\![s]\!] &= s_T \\
[\![Env]\!] &= e_U \\
[\![\lambda x.s]\!] &= abs([\![x]\!], [\![s]\!]) \\
[\![(s\ t)]\!] &= app([\![s]\!], [\![t]\!]) \\
[\![(\texttt{letrec}\ \{Env\}\ \texttt{in}\ s)]\!] &= letrec([\![Env]\!], [\![s]\!]) \\
[\![x = s]\!] &= bind([\![x]\!], [\![s]\!])
\end{aligned}$$

Variablen des Kalküls Λ^{let} werden in Variablen der entsprechenden Sorte übersetzt. Z.B: $x, y, z \xrightarrow{[\![\,]\!]} x_V, y_V, z_V$, dabei geht i.A. aus dem Kontext hervor, um welche Variablensorte es sich in einem Λ^{let}-Ausdruck handelt. Alle anderen Konstrukte des Λ^{let}-Kalküls werden auf natürliche Weise in Σ^{let}-Terme übersetzt. Die Übersetzung von `letrec`-Umgebungen lassen wir zunächst offen, sie wird in Kapitel 4 beschrieben.

Durch Induktion über die Struktur von Λ^{let}-Ausdrücken unter Verwendung von Definition 3.3.7 lässt sich zeigen:

Proposition 3.3.10. Alle Λ^{let}-Ausdrücke (die keine Kontexte und Ketten enthalten) werden durch $[\![\ ]\!]$ auf wohlsortierte Σ^{let}-Terme abgebildet.

Ein Term t kann mehrere Sorten besitzen. Zum einen die direkt ablesbaren Sorten aus den Termdeklarationen (für einen Term sind ja beliebig viele Termdeklarationen erlaubt). Zum anderen besitzt t alle Sorten S, für die gilt $t \in T(\Sigma, S, X)$. Diese Sorten sind t indirekt durch Subsortendeklarationen zugeordnet (über $\sqsubseteq_{\Sigma}$). Formal wird die Menge der Sorten eines Terms folgendermaßen definiert:

Definition 3.3.11. Die Menge $T(\Sigma, X)$ aller *wohlsortierten* Σ-Terme ist definiert als

$$\bigcup_{S \in S_\Sigma} \{T(\Sigma, S, X)\}.$$

Die *Sorte eines Terms* t ist definiert als die Menge

$$S_\Sigma(t) := \{S \in S_\Sigma \mid t \in T(\Sigma, S, X)\}.$$

Die *Sorte einer Variablen* x ist definiert als

$$S_\Sigma(x) = \{S \in S_\Sigma \mid \mathbb{S}(x) \tilde{\sqsubseteq} S\}.$$

Für jede Variable $x \in X$ besitzt die Menge $S_\Sigma(x)$ ein kleinstes Element, nämlich $\mathbb{S}(x)$.

Die Menge der Sorten eines Terms ist für endliche Signaturen effizient berechenbar (siehe Schmidt-Schauß, 1989a, S. 22). Wir sind hier nicht weiter interessiert an der Berechnung der Sorten eines Terms t, sondern richten unser Augenmerk auf eine Bedingung, für die $S_\Sigma(t)$ eine spezielle Struktur aufweist: Die Existenz einer kleinsten Sorte für alle Terme t. Ist eine Signatur so strukturiert, dass alle Terme eine kleinste Sorte besitzen, wird sie als *regulär* bezeichnet. Unter dieser Bedingung besitzt die Unifikation wesentlich bessere Berechenbarkeitseigenschaften, als wenn eine kleinste Sorte nicht für alle Terme existiert.

Definition 3.3.12. Sei Σ eine Signatur mit Sorten. Σ ist *regulär*, gdw. $(S_\Sigma, \tilde{\sqsubseteq}_\Sigma)$ eine partiell geordnete Menge ist und für jeden Term t die Menge $S_\Sigma(t)$ ein kleinstes Element besitzt (d.h. $ls(S_\Sigma(t))$ existiert bezüglich $\tilde{\sqsubseteq}_\Sigma$). Für eine reguläre Signatur Σ wird diese eindeutige kleinste Sorte des Terms t mit $LS_\Sigma(t)$ bezeichnet: $LS_\Sigma(t) := ls(S_\Sigma(t))$.

Für Regulariät wird gefordert, dass die Sortenstruktur eine partiell geordnete Menge ist. Um dies zu verdeutlichen, wird ab jetzt $\sqsubseteq_\Sigma$ geschrieben, wenn die Ordnung eine partielle Ordnung ist; für die Quasiordnung wird wie bisher $\tilde{\sqsubseteq}_\Sigma$ geschrieben.

Zwei einfache Bedingungen, unter denen eine Signatur regulär ist, liefert folgende Proposition.

Proposition 3.3.13. Sei Σ eine Signatur mit Sorten.

1. Wenn $\sqsubseteq_\Sigma$ eine lineare, fundierte Partialordnung auf S_Σ ist, dann ist Σ regulär.
2. Wenn Σ endlich und einfach ist, dann ist Σ regulär.

Beweis. 1. ist klar, da jede fundierte, linear partiell geordnete Menge ein kleinstes Element besitzt.

Zu 2: Sei $\sim_\Sigma$ die von $\tilde{\sqsubseteq}_\Sigma$ induzierte Äquivalenzrelation und sei $\sqsubseteq_\Sigma$ die von $\tilde{\sqsubseteq}_\Sigma$ induzierte Partialordnung auf der Faktormenge $S_\Sigma/_{\sim_\Sigma}$ (in der Faktormenge werden äquivalente Sortensymbole zu einer Äquivalenzklasse zusammengefasst); $\sqsubseteq_\Sigma$ ist fundiert, weil Σ endlich ist. Sei $t = f(t_1, \ldots, t_n)$, dann folgt aus der Einfachheit von Σ, dass es genau eine Funktionsdeklaration gibt mit $f(x_1, \ldots, x_n) : R$. Für die Menge der Sorten von t haben wir $S_\Sigma(t) = \{S \in S_\Sigma \mid t \in T(\Sigma, S, X)\} = \{S \in S_\Sigma/_\sim \mid R \sqsubseteq_\Sigma S\}$ (nach Definition 3.3.7 und Einfachheit von Σ) und auf dieser Menge $S_\Sigma(t)$ ist $\sqsubseteq$ eine lineare, fundierte Partialordnung mit $LS_\Sigma(t) = R$. Sei $t = x$, dann gilt nach Definition 3.3.11 $LS_\Sigma(x) = \mathbb{S}(x)$. □

Der Beweis belegt, dass jede linear, fundierte Partialordnung ein kleinstes Element besitzt, was allerdings für eine lineare, fundierte Quasiordnung nicht gelten muss, da sie nicht antisymmetrisch ist (für eine Quasiordnung ist das kleinste Element sozusagen nur modulo der induzierten Äquivalenzrelation eindeutig bestimmt). Dieser Umstand wird dadurch behoben, dass äquivalente Sortensymbole zu Äquivalenzklassen zusammengefasst werden können. Auf der Faktormenge der Äquivalenzklassen induziert $\tilde{\sqsubseteq}_\Sigma$ eine Partialordnung $\sqsubseteq_\Sigma$, bezüglich der das kleinste Element der Sortenmenge eines Term eindeutig bestimmt ist, da $\sqsubseteq_\Sigma$ auf dieser Menge linear und fundiert ist.

Aus Proposition 3.3.13 folgt, dass man in einer einfachen Signatur, die kleinsten Sorten von Termen, die durch Anwendung von Funktionssymbolen gebildet werden, direkt aus den Funktionsdeklarationen der Signatur ablesen kann.

Korollar 3.3.14. Sei Σ eine einfache Signatur mit Funktionsdeklarationen

$$f_1 : S_{1_1} \to \cdots \to S_{n_1} \to S_1, \ldots, f_k : S_{1_k} \to \cdots \to S_{n_k} \to S_k,$$

wobei alle f_i, f_j paarweise verschieden sind. Dann gilt $LS_\Sigma(f_i(t_1, \ldots, t_{n_i})) = S_i$ für alle $i = 1, \ldots, k$ und beliebige Terme $t_1, \ldots t_n$.

In einfachen Signaturen haben wir die angenehme Situation, dass die kleinste Sorte eines Terms t nicht von dessen Subtermen abhängig ist, sondern direkt abgelesen werden kann. Entweder aus den Funktionsdeklarationen für $t = f(\ldots)$, oder durch $\mathbb{S}(x)$ für $t = x$.

Wie wir gesehen haben, ist die Signatur Σ^{let} eine einfache Signatur (Korollar 3.3.6). Folglich ist Σ^{let} auch eine reguläre Signatur (nach Proposition 3.3.13), was wir in folgendem Korollar festhalten.

Korollar 3.3.15. Die Signatur Σ^{let} (aus Beispiel 3.3.4) ist *regulär*, d.h. alle Terme $t \in T(\Sigma, S, X)$ haben eine eindeutige kleinste Sorte $LS_\Sigma(t)$.

Der Umgang mit wohlsortierten Substitutionen im nächsten Abschnitt wird durch folgende Beobachtung vereinfacht.

Proposition 3.3.16. Für alle regulären Signaturen Σ gilt:

1. $S_\Sigma(t) \subseteq S_\Sigma(s) \Leftrightarrow LS_\Sigma(t) \sqsupseteq_\Sigma LS_\Sigma(s)$.
2. $LS_\Sigma(x) = \mathbb{S}(x)$.

3.3.3 Wohlsortierte Substitutionen

Wir erweitern den Begriff der Substitutionen zu Substitutionen, die Sorten berücksichtigen.

Definition 3.3.17 (Wohlsortierte Substitution)**.** Die Menge der *wohlsortierten Substitutionen* (auch Σ-Substitutionen genannt) Sub_Σ ist folgendermaßen definiert:

$$Sub_\Sigma := \{\sigma \in Sub_{\overline{\Sigma}} \mid S_\Sigma(\sigma x) \supseteq S_\Sigma(x)\}.$$

Für reguläre Signaturen ist dies nach Proposition 3.3.16 gleichbedeutend mit

$$Sub_\Sigma := \{\sigma \in Sub_{\overline{\Sigma}} \mid LS_\Sigma(\sigma x) \sqsupseteq LS_\Sigma(x)\}.$$

Wohlsortierte Substitutionen schwächen die Sorten, d.h. σx hat eine kleinerer oder gleiche Sorte als x für alle Variablen x. Aus obiger Definition folgt, dass $Sub_\Sigma \subseteq Sub_{\overline{\Sigma}}$ und dass die Identitätssubstitution Id_Σ eine wohlsortierte Substitution ist: $Id_\Sigma \in Sub_\Sigma$.

Eine Σ-*Variablenumbenennung* $\rho \in Sub_\Sigma$ ist eine Abbildung, die injektiv auf $Dom(\rho)$ ist, und Variablen auf Variablen abbildet, so dass Sorten erhalten bleiben, d.h. es gilt $S_\Sigma(\rho x) = S_\Sigma(x)$ für alle $x \in X$. Eine Substitution $\rho \in Sub_\Sigma$ wird als Σ-*Permutation* bezeichnet, wenn ρ eine Σ-Variablenumbenennung ist, so dass $Dom(\rho) = Ran(\rho)$. Dann ist ρ eine Bijektion (auf $Dom(\rho)$) mit Inversem ρ^- und es gilt $\rho\rho^- = \rho^-\rho = Id_\Sigma$.

Wohlsortierte Substitutionen sind kompatibel mit der Sortenstruktur auf $T(\Sigma, X)$, d.h. wohlsortierte Substitutionen bilden $T(\Sigma, S, X)$ nach $T(\Sigma, S, X)$ ab für alle Sorten S.

Proposition 3.3.18. Für alle wohlsortierten Terme $t \in T(\Sigma, S, X)$ und alle wohlsortierten Substitutionen $\sigma \in Sub_\Sigma$ gilt $\sigma t \in T(\Sigma, S, X)$.

Beweis. Für $t = x \in X$ folgt die Behauptung direkt aus der Wohlsortiertheit von σ.

Sei $t = f(t_1, \ldots, t_n) \in T(\Sigma, S, X)$ und sei $\sigma = \{x_1 \mapsto s_1, \ldots, x_m \mapsto s_m\} \in Sub_\Sigma$. Wenn σ eine idempotente Σ-Substitution ist, dann besitzt sie die trianguläre Form

$[x_1 \mapsto s_1; \ldots; x_m \mapsto s_m]$ und die wiederholte Anwendung einzelner Komponenten von σ auf t impliziert $t \in T(\Sigma, S, X)$ nach Definition 3.3.7.

Ist σ nicht idempotent, dann sei $\rho \in Sub_\Sigma$ eine idempotente Σ-Variablenumbenennung mit $Dom(\rho) = VRan(\sigma)$ und $Ran(\rho) \cap (\cup_{i=1}^m Var(s_i) \cup Var(t)) = \emptyset$, d.h. $Ran(\rho)$ besteht aus Variablen, die nicht in s_i oder t vorkommen. Sei ρ^- die Umkehrung von ρ. Nach Lemma 3.2.5 gilt $\rho^-\rho = \rho^-$ und, weil $Dom(\rho^-) \cap (\cup_{i=1}^m Var(s_i) \cup Var(t)) = \emptyset$ nach Konstruktion von ρ gilt, folgt $\sigma t = \rho^-\rho\sigma t$. Die Substitution $\rho\sigma$ ist idempotent (wegen Konstruktion von ρ und Komposition von Substitutionen), und kann folgendermaßen zerlegt werden: $\{x_1 \mapsto \rho s_1, x_2 \mapsto \rho s_1, \ldots, x_m \mapsto \rho s_m\} = \{x_1 \mapsto \rho s_1\}\{x_2 \mapsto \rho s_2\} \ldots \{x_2 \mapsto \rho s_2\}$. Jede Komponente $\{x_i \mapsto \rho s_i\}$ ist wohlsortiert nach folgender Überlegung: ρ ist eine Σ-Variablenumbenennung, die Sorten erhält, d.h aus $s_i \in T(\Sigma, R, X)$ folgt $\rho s_i \in T(\Sigma, R, X)$ für alle Sorten R. Schrittweise Anwendung aller Komponenten $\{x_i \mapsto \rho s_i\}$ auf t nach Definition 3.3.7 resultiert in $\rho\sigma t \in T(\Sigma, S, X)$ und aus $\sigma t = \rho^-\rho\sigma t$ folgt $\sigma t \in T(\Sigma, S, X)$. □

Korollar 3.3.19. Seien σ und τ wohlsortierte Substitutionen. Dann gilt: deren Komposition $\sigma\tau$ ist eine wohlsortierte Substitution.

Beweis. Sei $\tau = \{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\} \in Sub_\Sigma$ und $\sigma \in Sub_\Sigma$. Zu zeigen ist $\sigma\tau x \in T(\Sigma, \mathbb{S}(x), X)$ für alle $x \in X$. Nach Proposition 3.3.18 gilt $\sigma\tau x_i = \sigma s_i \in T(\Sigma, \mathbb{S}(x), X)$ für alle $x_i \in Dom(\tau)$. Für alle anderen Variablen x gilt entweder $\sigma\tau x = \sigma x \in T(\Sigma, \mathbb{S}(x), X)$ oder $\sigma\tau x = x \in T(\Sigma, \mathbb{S}(x), X)$. □

Die nächste Proposition zeigt, wie wohlsortierte Terme erzeugt werden können durch die Anwendung von wohlsortierten Substitutionen auf Terme aus Termdeklarationen.

Proposition 3.3.20. Für alle Sorten $S \in S_\Sigma$ und alle Terme $s \in T(\Sigma, S, X)$, die keine Variablen-Terme sind, gibt es eine Termdeklaration $t : R \in \Sigma$ mit $R \sqsubseteq S$ und eine Substitution $\sigma \in Sub_\Sigma$, so dass gilt $\sigma t = s$.

Beweis. Durch strukturelle Induktion unter Verwendung von Definition 3.3.7 und Korollar 3.3.19. □

3.4 Syntaktische Unifikation von wohlsortierten Termen

Für zwei wohlsortierte Terme s, t ist man daran interessiert, eine wohlsortiert Substitution σ zu finden, so dass die beiden Terme σs und σt syntaktisch gleich sind: $\sigma s = \sigma t$. Die Berechnung einer solchen Substitution wird *Unifikation* genannt, σ

wird als *Unifikator* von s und t bzw. als *Lösung* der Gleichung $s =^? t$ bezeichnet. Sei Σ eine Signatur mit nur einem Sortensymbol und alle Funktionssymbole (f, g), Konstantensymbole (a) und Variablen (x, y) seien von dieser Sorte, dann sind einige Beispiele für Unifikationsprobleme und deren Lösungen:

$f(x) =^? f(a)$ hat genau einen Unifikator $\{x \mapsto a\}$.
$x =^? f(y)$ hat viele Unifikatoren $\{x \mapsto f(y)\}, \{x \mapsto f(a), y \mapsto a\}, \ldots$
$f(x) =^? g(y)$ hat keinen Unifikator.

Wie man sieht, kann ein Unifikationsproblem keine, eine oder mehrere Lösungen besitzen. Im Falle der Existenz von mehreren Unifikatoren unterscheiden diese sich jedoch im Grade ihrer Allgemeinheit. Beispielsweise ist für das Problem $x =^? f(y)$ die Substitution $\{x \mapsto f(y)\}$ ein *allgemeinerer* Unifikator als $\{x \mapsto f(a), y \mapsto a\}$.

Definition 3.4.1 (Instantiierungs-Quasiordnung)**.** Seien $\sigma, \tau \in Sub_\Sigma$ wohlsortierte Substitutionen und $W \subseteq X$ eine Menge von Variablen. Die Substitution σ ist *allgemeiner auf* W als die Substitution τ, geschrieben als $\sigma \lesssim_\Sigma \tau[W]$, wenn es eine Substitution $\delta \in Sub_\Sigma$ gibt, so dass gilt $\tau = \delta\sigma[W]$. Die Substitution τ wird als *Instanz* von σ und die Relation $\lesssim_\Sigma [W]$ als *Instantiierungs-Quasiordnung* bezeichnet.

Für $W = X$ schreiben wir verkürzt $\lesssim_\Sigma$.

Für die Substitutionen $\sigma = \{x \mapsto f(y)\}$ und $\tau = \{x \mapsto f(a), y \mapsto a\}$ aus obigem Beispiel gilt $\sigma \lesssim_\Sigma \tau$ nach folgender Überlegung: Sei $\delta = \{y \mapsto a\}$, dann folgt $\tau = \delta\sigma$, weil $\tau(x) = f(a) = \delta(\sigma(x))$ und $\tau(y) = a = \delta(\sigma(y))$ und $\tau(z) = z = \delta(\sigma(z))$ für alle anderen Variablen z.

Lemma 3.4.2. Die Relation $\lesssim_\Sigma$ ist eine Quasiordnung auf der Menge der wohlsortierten Substitutionen Sub_Σ

Beweis. Reflexivität ist offensichtlich für $\delta = Id$. Zur Transitivität sei $\sigma_2 = \delta_1\sigma_1$ und $\sigma_3 = \delta_2\sigma_2$. Folglich $\sigma_3 = \delta_2\sigma_2 = \delta_2(\delta_1\sigma_1) = (\delta_2\delta_1)\sigma_1$, weil die Komposition von (wohlsortierten) Substitutionen assoziativ ist. □

Die Relation $\lesssim_\Sigma$ ist antisymmetrisch, d.h. beispielsweise für $\sigma = \{x_S \mapsto y_S\}$ und $\tau = \{y_S \mapsto x_S\}$ gilt $\sigma \lesssim_\Sigma \tau$ wegen $\tau = \tau\sigma$ und $\tau \lesssim_\Sigma \sigma$ wegen $\sigma = \sigma\tau$. Allerdings gilt nicht $\sigma = \tau$; trotzdem sind die beiden Substitutionen äquivalent bezüglich der von $\lesssim_\Sigma$ induzierten Äquivalenzrelation $\sim_\Sigma := \lesssim_\Sigma \cap \geq_\Sigma$. Bzw. die allgemeine Definition bezüglich einer Menge von Variablen: $\sigma \sim_\Sigma \tau[W] := \sigma \lesssim_\Sigma \tau[W] \wedge \tau \lesssim_\Sigma \sigma[W]$. Man sagt, dass die beiden Substitutionen äquivalent modulo Komposition mit einer Σ-Variablenpermutation sind.

Lemma 3.4.3. Seien $\sigma, \tau \in Sub_\Sigma$ und $W \subseteq X$ eine Menge von Variablen. Dann gilt: $\sigma \sim_\Sigma \tau[W]$, gdw. es eine Σ-Permutation ρ gibt, so dass $\sigma = \rho\tau[W]$.

Beweis. $\Leftarrow$: Sei ρ eine Σ-Permutation. $\sigma = \rho\tau[W]$ impliziert $\tau \lesssim_\Sigma \sigma[W]$. Sei ρ^- die inverse Σ-Permutation zu ρ, dann gilt $\rho^-\sigma = \rho^-\rho\tau[W]$ und wegen $\rho^-\rho = Id_\Sigma$ folgt $\rho^-\sigma = \tau[W]$ und damit auch $\sigma \lesssim_\Sigma \tau[W]$ und letztlich $\sigma \sim_\Sigma \tau[W]$.

$\Rightarrow$: Es gelte $\sigma \sim_\Sigma \tau[W]$, d.h. es gibt Substitutionen $\delta_1, \delta_2 \in Sub_\Sigma$, so dass $\tau = \delta_1\sigma[W]$ und $\sigma = \delta_2\tau[W]$. Einsetzen ergibt $\sigma = \delta_2\delta_1\sigma[W]$, woraus $\delta_2\delta_1 = Id[Dom(\sigma)]$ folgt. Andererseits gilt aber auch $\tau = \delta_1\delta_2\tau[W]$, d.h. wir haben analog $\delta_1\delta_2 = Id[Dom(\tau)]$, woraus für $\rho = \delta_2\delta_1[W]$ folgt $Dom(\rho) = Ran(\rho)$, also ist ρ eine Σ-Permutation mit $\sigma = \rho\tau[W]$. $\square$

Die Instantiierungs-Quasiordnung wird verwendet, um zu begründen, warum für ein gegebenes Unifikationsproblem nicht alle Unifikatoren berechnet werden müssen: Die Berechnung eines kleinsten Unifikators σ bezüglich $\lesssim_\Sigma$ ist ausreichend, da alle anderen Unifikatoren durch Instantiierung aus σ gewonnen werden können. Allerdings reicht es für Terme mit Sorten im Allgemeinen nicht aus, nur einen kleinsten Unifikator bezüglich $\lesssim_\Sigma$ zu berechnen. Zur Begründung betrachte folgendes Beispiel:

Beispiel 3.4.4. Sei $\{A, B \sqsupset C, D\}$ eine Signatur mit Sorten. Für das Unifikationsproblem $x_A =^? y_B$ existieren zwei allgemeinste Lösungen $\sigma_1 = \{x_A \mapsto w_C, y_B \mapsto w_C\}$ und $\sigma_2 = \{x_A \mapsto z_D, y_B \mapsto z_D\}$, die bezüglich $\lesssim_\Sigma$ nicht miteinander vergleichbar sind. Die Ursache hierfür liegt darin, dass die größte untere Schranke von A und B nicht eindeutig ist: $glb(A, B) = \{C, D\}$, d.h. die Sortenstruktur ist kein unterer Halbverband.

Folglich muss für ein Unifikationsproblem, das Terme mit Sorten enthält, eine Menge von allgemeinsten Unifikatoren betrachtet werden.

Definition 3.4.5 (Unifikationsproblem)**.** Ein Σ-*Unifikationsproblem* (oder einfach Unifikationsproblem, wenn der Bezug zu Σ klar ist) ist eine endliche Menge von Gleichungen $P = \{s_1 =^? t_1, \ldots, s_n =^? t_n\}$ mit $s_i, t_i \in T(\Sigma, X)$. Eine *Lösung* von P (auch *Unifikator* von P genannt) ist eine Substitution $\sigma \in Sub_\Sigma$, so dass $\sigma t_1 = \sigma s_i$ für $i = 1, \ldots, n$. Die Menge aller (wohlsortierten) Lösungen von P wird als $U_\Sigma(P)$ bezeichnet

$$U_\Sigma(P) := \{\sigma \in Sub_\Sigma \mid \sigma s_i = \sigma t_i \text{ für } i = 1, \ldots, n\}.$$

P ist *lösbar* wenn $U_\Sigma(P) \neq \emptyset$. Das spezielle Unifikationsproblem $\bot$ besitzt keine Lösung. Mit $Var(P)$ wird die Menge der in P auftauchenden Variablen bezeichnet: $Var(P) := \cup_{i=1}^n (Var(s_i) \cup Var(t_i))$.

Definition 3.4.6. Eine *vollständige Menge von Unifikatoren* (oder *Lösungen*) für ein Unifikationsproblem P ist eine Menge von wohlsortierten Substitutionen $CU_\Sigma(P)$, so dass

- $CU_\Sigma(P) \subseteq U_\Sigma(P)$ und
- für alle $\tau \in U_\Sigma(P)$ gibt es $\sigma \in CU_\Sigma(P)$, so dass $\sigma \lesssim_\Sigma \tau[Var(P)]$.

Eine *minimale vollständige Menge von Unifikatoren* (bzw. *Lösungen*) für ein Unifikationsproblem P ist eine vollständige Menge von Unifikatoren $MU_\Sigma(P)$, so dass die Substitutionen in $MU_\Sigma(P)$ bezüglich $\lesssim_\Sigma [Var(P)]$ nicht vergleichbar sind:

- Für alle $\sigma, \tau \in MU_\Sigma(P)$ gilt: $\sigma \lesssim_\Sigma \tau[Var(P)]$ impliziert $\sigma = \tau[Var(P)]$

Eine Substitution σ ist ein *mgu* (most general unifier oder *allgemeinste Lösung*) von P gdw. $\{\sigma\}$ eine minimale vollständige Menge von Lösungen für P ist.

Die Einschränkung von $\lesssim_\Sigma$ auf die in P vorkommenden Variablen wird in Definition 3.4.6 getroffen, weil durch $\lesssim_\Sigma [Var(P)]$ mehr Substitutionen miteinander vergleichbar sind als durch die restriktivere Relation $\lesssim_\Sigma$, die Gleichheit für alle Variablen verlangt. Dadurch wird in manchen Fällen vermieden, dass die Menge der minimalen, vollständigen Unifikatoren unnötig groß wird (für ein Beispiel siehe Baader (1991)). Außerdem ist diese Beschränkung sinnvoll, da zur Berechnung aller Überlappungen die komplette Menge der vollständigen, minimalen Unifikatoren bestimmt werden muss. Hier ist es von Vorteil, eine Quasiordnung zu verwenden, die alternative Lösungen nur in Bezug auf die kleinere Variablenmenge vergleicht. Wie wir weiter unten sehen, werden (allgemeinste) Lösungen für ein Unifikationsproblem P berechnet, indem es schrittweise zu einem Problem S transformiert wird, aus dem eine Lösung direkt ablesbar ist. Bei der Transformation können neue Variablen in das Problem eingeführt werden. Eine Lösung für das Ausgangsproblem P erhält man durch Einschränkung der Lösung des Endproblems S auf die in P vorkommenden Variablen.

Eine vollständige, minimale Menge von Unifikatoren muss nicht immer existieren, da sich Vollständigkeit und Minimalität widersprechen können. Wenn eine vollständige und minimale Menge $MU_\Sigma(P)$ von Lösungen für ein Problem P existiert, ist diese eindeutig, bis auf die von $\lesssim_\Sigma [Var(P)]$ induzierte Äquivalenzrelation $\sim_\Sigma [Var(P)]$. D.h. seien M_1 und M_2 minimale, vollständige Mengen von Unifikatoren eines Unifikationsproblems P, dann gibt es eine Bijektion $B : M_1 \to M_2$, so dass $\sigma_1 \sim_\Sigma B(\sigma_1)[Var(P)]$ für alle $\sigma_1 \in M_1$ gilt (Fages & Huet, 1986). Folglich haben alle vollständigen, minimalen Mengen von Unifikatoren eines gegebenen Problems P dieselbe Kardinalität. Ein Umstand, der zur Definition des Begriffs des *Unifikationstyps*[1] herangezogen wird.

Definition 3.4.7 (Unifikationstyp)**.** Eine Signatur Σ mit Sorten besitzt den *Unifikationstyp*

eindeutig, gdw. $MU_\Sigma(P)$ existiert und $|MU_\Sigma(P)| \leq 1$ für alle Σ-Unifikationsprobleme P gilt.

[1] In der Literatur (z.B. Baader und Snyder (2001)) wird der Begriff Unifikationstyp bezüglich einer Gleichungstheorie E definiert. Da Gleichungstheorien erst in Kapitel 4 eingeführt werden, wird der Begriff hier (leicht missbräuchlich) verwendet, um die Kardinalität von minimalen, vollständigen Menge von Unifikatoren bezüglich einer gegebenen Signatur zu charakterisieren. Streng genommen, wird hier $|MU_{\Sigma,E}(P)|$ für die leere Theorie $E = \emptyset$ betrachtet.

endlich, gdw. $MU_\Sigma(P)$ existiert und $|MU_\Sigma(P)| \leq \infty$ für alle Σ-Unifikationsprobleme P gilt.

unendlich, gdw. $MU_\Sigma(P)$ existiert für alle Σ-Unifikationsprobleme P und es ein Σ-Unifikationsproblem P mit $|MU_\Sigma(P)| = \infty$ gibt.

null, gdw. es ein Σ-Unifikationsproblem P gibt, für das $MU_\Sigma(P)$ nicht existiert.

Eine Signatur Σ ist vom Unifikationstyp *eindeutig*, wenn alle lösbaren Σ-Unifikationsprobleme einen mgu besitzen. Ist eine Signatur Σ vom Unifikationstyp *endlich*, dann lassen sich alle Lösungen für ein gegebenes Σ-Unifikationsproblem repräsentieren, als Instanzen einer endlichen Menge von Lösungen.

3.4.1 Unifikation durch Transformation

Ein Unifikationsproblem kann gelöst werden, indem wiederholt Transformationen auf die Menge der Unifikationsgleichungen des Problems angewendet werden, bis die Lösung direkt abgelesen werden kann.

Definition 3.4.8. Ein Unifikationsproblem $S = \{x_1 \stackrel{?}{=} t_1, \ldots, x_n \stackrel{?}{=} t_n\}$ ist in *gelöster Form*, wenn alle Variablen x_i paarweise verschieden sind und nicht in t_i vorkommen und $LS_\Sigma(x_i) \in S_\Sigma(t_i)$[2] für alle $i = 1, \ldots, n$. Für diesen Fall definiert man die Lösung von S als

$$\sigma_S := \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}.$$

Ein Unifikationsproblem in gelöster Form repräsentiert einen mgu für dieses Problem, wie folgende Lemmata zeigen.

Lemma 3.4.9. Sei S ein Unifikationsproblem in gelöster Form. Dann gilt $\tau = \tau\sigma_S$ für alle $\tau \in U_\Sigma(S)$.

Beweis. Sei $S = \{x_1 \stackrel{?}{=} t_1, \ldots x_n \stackrel{?}{=} t_n\}$. Es wird durch Fallunterscheidung gezeigt, dass τ und $\tau\sigma_S$ für alle Variablen denselben Wert ergeben, d.h. $\tau x = \tau\sigma_S x$ für alle $x \in X$.

- $x_i \in \{x_1, \ldots, x_n\}$: Wegen $\tau \in U_\Sigma(S)$ gilt $\tau x_i = t_i$ und damit $\tau\sigma_S x_i = \tau t_i = \tau x_i$.
- $x \notin \{x_1, \ldots, x_n\}$: $\tau x = \tau\sigma_S x$ (weil $\sigma_S x = x$). □

Lemma 3.4.10. Sei S in gelöster Form und σ_S die zugehörige Lösung. Dann gilt: σ_S ist ein idempotenter mgu von S.

[2] D.h. $LS_\Sigma(x_i) \sqsubseteq LS_\Sigma(t_i)$ für reguläre Signaturen.

Beweis. Nach Definition 3.4.8 ist $\sigma_S \in U_\Sigma(S)$ eine Lösung von S (wegen $\sigma_S x_i = t_i = \sigma_S t_i$). Nach Lemma 3.4.9 gilt $\sigma_S \lesssim_\Sigma \tau$ für alle $\tau \in U_\Sigma(S)$, mit Definition 3.4.6, folgt, dass σ_S ein mgu von S ist. Gelöste Formen sind so definiert (Def. 3.4.8), dass keine Variable x_i in einem t_i auftaucht, also gilt $Dom(\sigma_S) \cap VRan(\sigma_S) = \emptyset$, d.h. σ_S ist idempotent. □

Ein Unifikationsproblem in gelöster Form repräsentiert also eine allgemeinste, idempotente Lösung für das Unifikationsproblem. Die Unifikation geht so vor, dass ein Ausgangsproblem P_1 durch eine Folge von Transformationen in ein gelöstes Unifikationsproblem P_n transformiert wird

$$P_1 \Rightarrow P_2 \Rightarrow \cdots \Rightarrow P_n.$$

Dabei soll gelten, dass die Transformationsschritte die Menge der Lösungen nicht verändert, so dass die Lösung für das Endproblem auch eine Lösung für alle vorhergehenden, insbesondere für das Ausgangsproblem repräsentiert. D.h. für $P_1 \Rightarrow P_2$ soll gelten $U_\Sigma(P_1) = U_\Sigma(P_2)|_{Var(P_1)}$. Da ein Transformationsschritt $P_1 \Rightarrow P_2$ neue Variablen (Variablen die nicht in P_1 vorkommen) in das Unifikationsproblem P_2 einführen kann, wird die Menge der Lösungen des Endproblems auf die im Ausgangsproblem vorkommenden Variablen $Var(P_1)$ beschränkt. Formal definiert man:

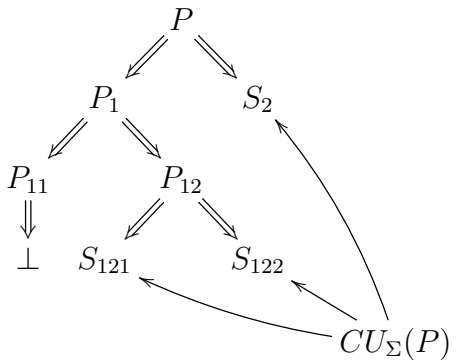
Definition 3.4.11. Sei $P_1 \Rightarrow P_2$ die Transformation eines Unifikationsproblems.

1. $P_1 \Rightarrow P_2$ ist *korrekt*, gdw. $U_\Sigma(P_1) \supseteq U_\Sigma(P_2)$.
2. $P_1 \Rightarrow P_2$ ist *vollständig*, gdw. $P_1 \Rightarrow P_2$ korrekt ist und $U_\Sigma(P_1) \subseteq U_\Sigma(P_2)|_{Var(P_1)}$, d.h. $U_\Sigma(P_1) = U_\Sigma(P_2)|_{Var(P_1)}$.
3. Eine Menge korrekter Transformationen $P \Rightarrow P_1, \ldots, P \Rightarrow P_n$ ist eine *vollständige Menge von Alternativen*, gdw. $U_\Sigma(P) = U_\Sigma(P_1)|_{Var(P)} \cup \cdots \cup U_\Sigma(P_n)|_{Var(P)}$.

Für eine Signatur mit endlichem (oder unendlichem) Unifikationstyp ist Folgendes im Bezug auf den Begriff der Vollständigkeit zu bedenken. Bei der Transformation eines Problems, das eine vollständige Menge von Lösungen besitzt, gehen zwangsläufig Lösungen verloren, da die gelöste Form jeweils nur eine Lösung repräsentiert. D.h. $U_\Sigma(P_1) \subseteq U_\Sigma(P_2)|_{Var(P_1)}$ wird nicht von allen Transformationsschritten erfüllt. Betrachte zur Erläuterung das Beispiel 3.4.4. Dort wird das Unifikationsproblem $P := \{x_A =^? y_B\}$ transformiert zu
$\Rightarrow \{x_A =^? w_C, y_B =^? w_C\} := P_1$ oder zu
$\Rightarrow \{x_A =^? z_D, y_B =^? z_D\} := P_2$
Da P_1, P_2 beides Lösungen von P sind, P_2 aber keine Lösung von P_1 (und umgekehrt), ist der Transformationsschritt $P \Rightarrow P_i, i \in \{1, 2\}$ nicht vollständig. Man schafft Abhilfe, indem man definiert:

Definition 3.4.12. Eine Menge von Transformationsregeln für Unifikationsprobleme[3] stellt eine *vollständige Unifikationsprozedur* dar, wenn für jedes Unifikationsproblem P und jede Substitution $\tau \in U_\Sigma(P)$ ein Unifikationsproblem S in gelöster Form existiert, so dass $P \Rightarrow^* S$ und $\sigma_S \lesssim_\Sigma \tau[Var(P)]$.

In der Definition wird die Tatsache verwendet, dass die durch Transformationsregeln beschriebenen Transformationsrelation $\Rightarrow$ auf der Menge der Unifikationsprobleme *nichtdeterministisch* ist. D.h. wenn auf ein Unifikationsproblem mehrere Transformationen angewendet werden können, wird eine beliebige ausgewählt. Außerdem spielt die Reihenfolge der Anwendung von Transformationsschritten keine Rolle. Die Vollständigkeit einer Unifikationsprozedur kann dann so verstanden werden: Wenn für ein Unifikationsproblem P der Baum aller möglichen Transformationsfolgen (mit den nichtdeterministischen Verzweigungen) aufgespannt wird, dann bilden die Blätter die vollständige Menge der Lösungen $CU_\Sigma(P)$, wie in folgender Abbildung beispielhaft zu sehen ist:



Durch die Entfernung redundanter Substitutionen (d.h. durch die Betrachtung der Faktormenge $CU_\Sigma(P)/_{\sim_\Sigma}$) kann $CU_\Sigma(P)$ zu einer minimalen, vollständigen Menge von Lösungen $MU_\Sigma(P)$ gemacht werden, wenn $\lesssim_\Sigma$ entscheidbar ist.

Wir betrachten nun die Transformationsregeln (auch *Unifikationsregeln* genannt), die verwendet werden, um ein Σ-Unifikationsproblem schrittweise in gelöste Form (oder $\bot$, wenn das Problem keine Lösung besitzt) zu überführen. Es werden die Unifikationsregeln aus Schmidt-Schauß (1989a) für endliche, reguläre Signaturen vorgestellt. Die Beschränkung auf Unifikationsregeln für reguläre Signaturen erfolgt, weil es sich nach Korollar 3.3.6 und 3.3.15 bei der Signatur Σ^{let} um eine endliche und reguläre Signatur handelt und wir vorrangig daran interessiert sind, Unifikationsprobleme mit Σ^{let}-Termen zu lösen, um kritische Überlappungen zu berechnen. Für Unifikationsregeln in beliebigen Signaturen siehe Schmidt-Schauß (1989a), S. 98.

[3] Diese Regeln definieren die Transformationsrelation $\Rightarrow$ und deren reflexiven, transitiven Abschluss $\Rightarrow^*$ auf der Menge der Unifikationsprobleme.

Definition 3.4.13 (Unifikationsregeln)**.** Sei Σ eine endliche, reguläre Signatur mit Sorten. Die Regeln zur Transformation von Σ-Unifikationsproblemen sind in Abbildung 3.2 definiert.

Zur Vereinfachung der Regeln und des Vollständigkeitsbeweises wird die Annahme getroffen, dass die größte untere Schranke (glb) zweier Sortensymbole (wenn sie existiert) eindeutig ist, d.h. die Sortenstruktur ist ein oberer Halbverband. Andernfalls muss die Regel *Common Subsort* auf eine Menge von glbs anstatt auf eine eindeutige glb angepasst werden.

Wird eine Variable (*Common Subsort*) oder eine Termdeklaration (*Weakening*) durch eine Transformation in ein Unifikationsproblem eingeführt, darf der eingeführte Term nur Variablen enthalten, die bisher nicht im Unifikationsproblem vorkommen, sogenannte neue Variablen.

Die Anwendung einer Substitution σ auf ein Unifikationsproblem $P = \{s_i =^? t_i \mid i = 1, \ldots n\}$ ist definiert als $\sigma P := \{\sigma s_i =^? \sigma t_i \mid i = 1, \ldots n\}$. Das Symbol $\uplus$ bezeichnet die disjunkte Vereinigung. Dadurch wird die von einer Unifikationsregel ausgewählte Gleichung aus dem Unifikationsproblem entfernt (die Regel *Variable Elimination* fügt die Gleichung wieder ein, da sie in gelöster Form ist und ein Teil der Lösung des Problems darstellt).

Bemerkung 3.4.14. Wenn Σ eine einfache Signatur ist, wird die Regel *Weakening* nicht zur Unifikation benötigt. Zur Begründung sei ein Unifikationsproblem $x =^? f(t_1, \ldots, t_n)$ gegeben mit $x \notin Var(f(t_1, \ldots, t_n))$ und $LS_\Sigma(f(t_1, \ldots, t_n)) = R \not\sqsubseteq LS_\Sigma(x)$. Dann existiert wegen der Einfachheit von Σ keine weitere Termdeklaration $f(s_1, \ldots, s_n) : S$. D.h. es kommt in diesem Fall immer zur Anwendung der Regel *Sorted Fail Fun*: Die Gleichung besitzt keine Lösung in einer einfachen Signatur.

Aus diesem Grund wird die *Weakening* Regel hier beim Beweis der Vollständigkeit nicht berücksichtigt. Der Beweis wird dadurch vereinfacht. Für beliebige Sortenstrukturen ist *Weakening* im Allgemeinen keine vollständige Unifikationsregel und deshalb muss bei deren Vollständigkeitsbeweis auf Definition 3.4.12 zurückgegriffen werden. Aus dem gleichen Grund wird die Vereinfachung der Regel *Common Subsort* in Definition 3.4.13 festgelegt. Für den kompletten Beweis siehe Schmidt-Schauß (1989a), S. 102.

Lemma 3.4.15. Die Unifikationsregeln aus Definition 3.4.13, ausgenommen die Regel *Weakening*, sind vollständige Transformationsregeln für Σ-Unifikationsprobleme.

Beweis. Für *Tautology*, *Orientation*, *Decomposition* und *Subsort* ist offensichtlich, dass ihre Anwendung die Menge der Lösungen nicht verändert. Wir betrachten die verbleibenden Regeln.

Tautology
$\{s =^? s\} \uplus P \Rightarrow P$
Orientation
$\{t =^? x\} \uplus P \Rightarrow \{x =^? t\} \cup P$
wenn t keine Variable ist.
Decomposition
$\{f(s_1, \ldots, s_n) =^? f(t_1, \ldots, t_n)\} \uplus P \Rightarrow \{s_1 =^? t_1, \ldots s_n =^? t_n\} \cup P$
Variable Elimination (**sorted**)
$\{x =^? t\} \uplus P \Rightarrow \{x =^? t\} \cup \{x \mapsto t\}P$
wenn $x \notin \mathit{Var}(t)$ und $LS_\Sigma(t) \sqsubseteq LS_\Sigma(x)$.
Symbol Clash
$\{f(s_1, \ldots, s_n) =^? g(t_1, \ldots, t_m)\} \uplus P \Rightarrow \bot$
Occurs Check
$\{x =^? t\} \uplus P \Rightarrow \bot$
wenn $x \in \mathit{Var}(t)$ und $x \neq t$.
Subsort
$\{x =^? y\} \uplus P \Rightarrow \{y =^? x\} \cup P$
wenn $LS_\Sigma(x) \sqsubseteq LS_\Sigma(y)$
Common Subsort
$\{x =^? y\} \uplus P \Rightarrow \{x =^? z, y =^? z\} \cup P$
wenn $LS_\Sigma(x) \not\sqsubseteq LS_\Sigma(y), LS_\Sigma(y) \not\sqsubseteq LS_\Sigma(x)$ und
$z : S$ ist neue Variable der Sorte $S = glb(LS_\Sigma(x), LS_\Sigma(y))$.
Sorted Fail Var
$\{x =^? y\} \uplus P \Rightarrow \bot$
wenn $LS_\Sigma(x) \not\sqsubseteq LS_\Sigma(y)$ und $LS_\Sigma(y) \not\sqsubseteq LS_\Sigma(x)$
und $glb(LS_\Sigma(x), LS_\Sigma(y))$ existiert nicht.
Weakening
$\{x =^? f(t_1, \ldots, t_n)\} \uplus P \Rightarrow \{x =^? f(s_1, \ldots, s_n), t_1 =^? s_1, \ldots, t_n =^? s_n\} \cup P$
wenn $x \notin \mathit{Var}(f(t_1, \ldots, t_n))$ und $LS_\Sigma(f(t_1, \ldots, t_n)) \not\sqsubseteq LS_\Sigma(x)$ und
$f(s_1, \ldots, s_n) : S$ ist Termdeklaration mit $S = glb(LS_\Sigma(x), LS_\Sigma(f(s_1, \ldots, s_n)))$.
Sorted Fail Fun
$\{x =^? f(t_1, \ldots, t_n)\} \uplus P \Rightarrow \bot$
wenn $x \notin \mathit{Var}(f(t_1, \ldots, t_n))$ und $LS_\Sigma(f(t_1, \ldots, t_n)) \not\sqsubseteq LS_\Sigma(x)$ und
es gibt keine Termdeklaration $f(s_1, \ldots, s_n) : S$
mit $S = glb(LS_\Sigma(x), LS_\Sigma(f(s_1, \ldots, s_n)))$.

Figure 3.2. Unifikationsregeln für Terme mit Sorten nach Schmidt-Schauß (1989).

Variable Elimination: Zur Korrektheit sei $\sigma \in U_\Sigma(\{x =^? t\} \uplus P)$, d.h. σ unifiziert x mit t, also haben wir $\sigma x = \sigma t$ und damit auch $\sigma = \sigma\{x \mapsto t\}$ (vgl. Lemma 3.4.9), woraus $\sigma\{x \mapsto t\} \in U_\Sigma(\{x =^? t\} \cup P)$ folgt, was äquivalent zu $\sigma \in U_\Sigma(\{x =^? t\} \cup \{x \mapsto t\}P)$ ist. Für den noch fehlenden Teil der Vollständigkeit sei $\sigma \in U_\Sigma(\{x =^? t\} \cup \{x \mapsto t\}P)$. Wieder wird x mit t unifiziert: $\sigma x = \sigma t$ und deswegen $\sigma = \sigma\{x \mapsto t\}$, womit $\sigma \in U_\Sigma(\{x =^? t\} \uplus P)$ folgt. An dieser Stelle macht es keinen Unterschied, ob $\{x \mapsto t\}$ wohlsortiert ist oder nicht.

Symbol Clash: Eine Gleichung $f(s_1, \ldots, s_n) =^? g(t_1, \ldots, t_m)$ besitzt keine Lösung: $\sigma f(s_1, \ldots, s_n) = f(\sigma s_1, \ldots, \sigma s_n) \neq g(\sigma t_1, \ldots, \sigma t_m) = \sigma g(t_1, \ldots, t_m)$.

Occurs Check: Eine Gleichung der Form $x =^? t$ mit $x \in \mathit{Var}(t)$ und $x \neq t$ besitzt keine Lösung: Wegen $x \neq t$ hat t die Form $f(t_1, \ldots, t_n)$ mit $x \in \mathit{Var}(t_i)$ für ein i. Dann kann es keine Substitution geben, so dass $\sigma(x) = \sigma(t)$, weil sich die Größen der Lösungsterme unterscheiden: $|\sigma(x)| \leq |\sigma(t_i)| < |\sigma(t)|$.

Common Subsort: Sei $\sigma \in U_\Sigma(\{x =^? y\} \uplus P)$, d.h. es gilt $\sigma x = \sigma y$; da σ wohlsortiert ist haben wir $LS_\Sigma(\sigma x) \sqsubseteq \mathit{glb}(LS_\Sigma(x), LS_\Sigma(y))$. Sei au"serdem $z : S$ eine neue Variable mit $S = \mathit{glb}(LS_\Sigma(x), LS_\Sigma(y))$[4] und $X = \mathit{Var}(\{x =^? y\} \cup P)$. Betrachte folgende Äquivalenzumformungen:

$$\begin{aligned} \sigma \in U_\Sigma(\{x =^? y\} \uplus P) &\Leftrightarrow \sigma x = \sigma y \wedge \sigma \in U_\Sigma(P) \\ &\Leftrightarrow \sigma x = \sigma z \wedge \sigma y = \sigma z \wedge \sigma \in U_\Sigma(P)|_X \\ &\Leftrightarrow \sigma \in U_\Sigma(\{x =^? z, y =^? z\} \cup P)|_X \end{aligned}$$

Sorted Fail: Existiert $\mathit{glb}(LS_\Sigma(x), LS_\Sigma(y))$ nicht, hat die Gleichung $x =^? y$ für $LS_\Sigma(x) \not\sqsubseteq LS_\Sigma(y)$ und $LS_\Sigma(y) \not\sqsubseteq LS_\Sigma(x)$ keine wohlsortierte Lösung.

Sorted Fail Fun: $\sigma \in U_\Sigma(\{x =^? f(t_1, \ldots, t_n)\} \uplus P)$ impliziert $\sigma x = \sigma f(t_1, \ldots, t_n)$ und weil σ wohlsortiert ist, muss gelten $LS_\Sigma(\sigma x) \sqsubseteq \mathit{glb}(LS_\Sigma(x), LS_\Sigma(f(t_1, \ldots, t_n)))$ bzw. $LS_\Sigma(\sigma f(t_1, \ldots, t_n)) \sqsubseteq \mathit{glb}(LS_\Sigma(x), LS_\Sigma(f(t_1, \ldots, t_n)))$. Nach Proposition 3.3.20 existiert dann eine Termdeklaration $f(s_1, \ldots, s_n) : S$, so dass $S \sqsubseteq \mathit{glb}(LS_\Sigma(x), LS_\Sigma(f(t_1, \ldots, t_n)))$. Existiert keine solche Termdeklaration, besitzt die Gleichung keine Lösung. □

Der Algorithmus *Unify* in Abbildung 3.3 verwendet die definierten Unifikationsregeln, um rekursiv eine Lösung für ein gegebenes Unifikationsproblem P zu berechnen.

Weil alle von *Unify* zur Unifikation verwendeten Regeln nach Lemma 3.4.15 vollständig sind, folgt direkt:

Korollar 3.4.16. *Unify* ist ein vollständiger Unifikationsalgorithmus für endliche und einfache Signaturen.

[4] Hier wird die Annahme verwendet, dass die *glb* eindeutig ist. Ist sie das nicht, muss ein Element ausgewählt werden. *Common Subsort* ist für diesen Fall nicht mehr vollständig.

Unify (P) = **if** *es gibt ein* P', *so dass* $P \Rightarrow P'$
then *Unify* (P')
else if P *ist in geloester Form* **then return** σ_P
else return P *ist nicht loesbar*

Figure 3.3. Der *Unify* Algorithmus zur Unifikation von Termen mit Sorten.

Der Unifikationsalgorithmus ist nicht deterministisch. Wenn mehr als eine Unifikationsregel anwendbar ist, wählt der Algorithmus eine beliebige aus. Die Terminierung von *Unify* hängt davon ab, ob die Anwendung der Transformationsschritte $\Rightarrow$ terminiert. Schmidt-Schauß (1989a) zeigt, dass für reguläre, elementare Signaturen, deren Sortenstruktur ein oberer Halbverband ist, die Anwendung der Transformationsschritte terminiert. In Anlehnung daran soll hier kurz erläutert werden, weshalb *Unify* für endliche, einfache Signaturen terminiert. Eine Schwierigkeit des Beweises liegt darin, dass sich durch Anwendung der Regeln *Variable Elimination* und *Common Subsort* ein Unifikationsproblem vergrößern kann, wie z.B. $\{x \stackrel{?}{=} f(a,b), g(x) \stackrel{?}{=} y\} \Rightarrow \{x \stackrel{?}{=} f(a,b), g(f(a,b)) \stackrel{?}{=} y\}$. Die Lösung besteht darin, ein Komplexitätsmaß zu finden, das zeigt, dass trotz einer möglichen Vergrößerung des Problems, die Transformationen das Problem näher an eine gelöste Form bringen.

Für endliche und einfache Signaturen muss die Regel *Weakening* für die Terminierung nicht betrachtet werden, weil sie in solchen Signaturen zur Unifikation nicht benötigt wird (Bemerkung 3.4.14). Die Regeln *Subsort* und *Common Subsort* werden leicht angepasst:

Subsort*
$\{x \stackrel{?}{=} y\} \uplus P \Rightarrow \{y \stackrel{?}{=} x\} \cup \{y \mapsto x\}P$
wenn $LS_\Sigma(x) \sqsubset LS_\Sigma(y)$

Common Subsort*
$\{x \stackrel{?}{=} y\} \uplus P \Rightarrow \{x \stackrel{?}{=} z, y \stackrel{?}{=} z\} \cup \{x \mapsto z, y \mapsto z\}P$
wenn $LS_\Sigma(x) \not\sqsubseteq LS_\Sigma(y), LS_\Sigma(y) \not\sqsubseteq LS_\Sigma(x)$ und
$z : S$ ist neue Variable der Sorte $S = glb(LS_\Sigma(x), LS_\Sigma(y))$.

Lemma 3.4.17. Sei Σ eine endliche, einfache Signatur. Dann terminiert *Unify*(P) für alle Σ-Unifikationsprobleme P.

Beweis. Zuerst halten wir fest: Wird eine der Regeln *Symbol Clash*, *Occurs Check*, *Sorted Fail Var* oder *Sorted Fail Fun* auf P angewendet, dann terminiert *Unify* mit der Antwort: P besitzt keine Lösung.

Für alle anderen Regeln wird gezeigt, dass ihre Anwendung ein Unifikationsproblemen zugeordnetes Komplexitätsmaß verkleinert.

Eine Variable x wird als gelöst bezeichnet, wenn sie genau einmal in P vorkommt und zwar auf der linken Seite einer Gleichung $x =^? t$, so dass $x \notin \mathit{Var}(t)$. Einem Unifikationsproblem P wird das Komplexitätsmaß (n_1, n_2, n_3) zugeordnet, so dass

- n_1 die Anzahl der Variablen in P ist, die nicht gelöst sind,
- n_2 die Größe von P ist: $\sum_{(s=^?t)\in P}(|s| + |t|)$, und
- n_3 die Anzahl der Gleichungen der Form $t = x$ in P ist, wobei t keine Variable ist.

Folgende Tabelle zeigt, dass jede Anwendung einer Unifikationsregel die Tripel bezüglich der lexikographischen Ordnung[5] verkleinert:

	n_1	n_2	n_3
Tautology	$\geq$	$>$	
Orientation	$\geq$	$=$	$>$
Decomposition	$\geq$	$>$	
Elimination	$>$		
Subsort*	$>$		
Common Subsort*	$>$		

Tautology und *Decomposition* verkleinern n_2. *Orientation* verändert n_2 nicht, verkleinert aber n_3. *Elimination*, *Subsort** und *Common Subsort** verkleinern n_1, das durch keine andere Regel vergrößert wird. □

Der Beweis verdeutlicht noch einmal, dass die Reihenfolge, in der Unifikationsregeln auf ein Unifikationsproblem angewendet werden, keine Rolle spielen. Sind mehrere Regeln auf ein Problem anwendbar, kann eine beliebige ausgewählt werden, um das Problem zu transformieren, ohne etwas am Terminierungsverhalten des Algorithmus zu verändern.

Satz 3.4.18. Sei Σ eine endliche und einfache Signatur. Für alle Σ-Unifikationsprobleme P gilt:

- Wenn P lösbar ist, dann terminiert *Unify*(P) mit einer Lösung σ aus der vollständigen Menge der Lösungen $CU_\Sigma(P)$ von P.
- Wenn P nicht lösbar ist, dann terminiert *Unify*(P) mit der Antwort: P ist nicht lösbar.

Beweis. *Unify* ist vollständig (Korollar 3.4.16) und terminiert (Lemma 3.4.17), deswegen reicht es zu zeigen: Wenn das Unifikationsproblem P lösbar ist, dann wird

[5] Die lexikographische Ordnung ist für Tripel natürlicher Zahlen durch $(a, b, c) > (a', b', c') :\Leftrightarrow (a > a') \vee (a = a' \vee b > b') \vee (a = a', b = b', c > c')$ definiert.

es von *Unify* in eine gelöste Form S überführt. Nach Lemma 3.4.10 repräsentiert σ_S dann einen Unifikator aus der vollständigen Menge der Unifikatoren von P.

Sei $S \neq \bot$ das Unifikationsproblem mit dem *Unify*(P) terminiert. S enthält keine Gleichungen der Form $x =^? x$ (wegen *Tautology*), $t =^? x$ (wegen *Orientation*), $f(\dots) =^? f(\dots)$ (wegen *Decomposition*), $f(\dots) =^? g(\dots)$ (wegen *Symbol Clash*). Also sind alle in S enthaltenen Gleichungen von der Form $x =^? t$, mit $x \notin Var(t)$ (wegen *Occurs Check*) und $LS_\Sigma(x) \sqsupseteq LS_\Sigma(t)$, d.h. σ_S ist wohlsortiert (wegen *Subsort*, *Common Subsort*, *Sorted Fail Var* und *Sorted Fail Fun*). Wegen *Variable Elimination* kommt kein x mehr als einmal in S vor. Folglich ist S in gelöster Form. □

3.4.2 Der Unifikationstyp von endlichen und einfachen Signaturen

Zur Berechnung aller Überlappungen zwischen zwei Termen s, t muss die (minimale) vollständige Menge der Unifikatoren für alle Subterme (die keine Variablen sind) s_i und t_i von s und t bestimmt werden. Da wir vor allem an der Signatur Σ^{let} interessiert sind, stellt sich die Frage nach dem Unifikationstyp von endlichen, einfachen Signaturen. Das Vorhaben der Berechnung aller Überlappungen würde scheitern, wenn der Unifikationstyp hier *unendlich* wäre. Aus der Terminierung von *Unify* (Lemma 3.4.17) kann man allerdings direkt schließen:

Korollar 3.4.19. Endliche und einfache Signaturen sind vom Unifikationstyp *endlich*.

Die Tatsache, dass die Regel *Weakening* zur Unifikation in einfachen Signaturen nicht benötigt wird (siehe Bemerkung 3.4.14), sowie der im Vollständigkeitsbeweis der Unifikationsregeln (Lemma 3.4.15 sowie Beispiel 3.4.4) gegebenen Hinweise, dass die Regel *Common Subsort* nicht vollständig ist, wenn für zwei Sorten eine Menge größter unterer Schranken existiert, sind Indizien dafür, dass in einfachen Signaturen der Unifikationstyp komplett von der Sortenstruktur determiniert wird. Im Folgenden wird ein Resultat von Walther (1988) verwendet, der zeigt, dass für einfache Signaturen der Unifikationstyp als Funktion der Sortenstruktur charakterisiert werden kann.

Sei Σ eine endliche und einfache Signatur. Die zugehörige Sortenstruktur $(S_\Sigma, \sqsubseteq_\Sigma)$[6] wird durch eine sogenannte *maximal-sorts-condition* klassifiziert.

Definition 3.4.20. Eine partiell geordnete Menge $(S_\Sigma, \sqsubseteq_\Sigma)$ erfüllt die *maximal-sorts-condition*, gdw. für alle nichtleeren Teilmengen $M \subset S_\Sigma$ gilt

$$\forall S \in lbs(M)\ \exists S_m \in max(lbs(M)),\ \text{so dass}\ S \sqsubseteq_\Sigma S_m.$$

[6] Es wird hier verlangt, dass $\sqsubseteq_\Sigma$ eine Partialordnung auf S_Σ ist.

$(S_\Sigma, \sqsubseteq_\Sigma)$ hat den *Grad* 1, gdw. $|max(lbs(M))| \leq 1$ oder den *Grad* ω (endlich), gdw. $|max(lbs(M))| < \infty$ für alle nichtleeren Teilmengen $M \subset S_\Sigma$.

Einer Sortenstruktur, in der jede endliche Teilmenge eine eindeutige (oder gar keine) größte untere Schranke besitzt, wird der Grad 1 zugeordnet. Dieser Fall entspricht der ersten Bedingung der Definition eines oberen Halbverbandes (siehe Definition 3.1.6). Gibt es in der Sortenstruktur eine Teilmenge, die mehr als eine größte untere Schranke besitzt, erhält sie den Grad ω (für endlich). Der Satz von Walther stellt ein Verbindung zwischen dem Grad einer Sortenstruktur und dem Unifikationstyp der zugrundeliegenden Signatur her.

Satz 3.4.21 (Walther, 1988)**.** Sei Σ eine endliche und einfache Signatur, $(S_\Sigma, \sqsubseteq_\Sigma)$ die zugehörige Sortenstruktur, die die maximal-sorts-condition erfüllt. Dann gilt:

- $(S_\Sigma, \sqsubseteq_\Sigma)$ hat den Grad 1, gdw. Σ den Unifikationstyp *eindeutig* hat.
- $(S_\Sigma, \sqsubseteq_\Sigma)$ hat den Grad ω, gdw. Σ den Unifikationstyp *endlich* hat.

Es ist einfach zu zeigen, dass die durch Σ^{let} definierte Sortenstruktur die maximal-sorts-condition erfüllt und einen Grad von 1 hat.

Proposition 3.4.22. Die durch Σ^{let} (in Beispiel 3.3.4) definierte Sortenstruktur $(S_{\Sigma^{let}}, \sqsubseteq_{\Sigma^{let}})$ erfüllt die maximal-sorts-condition und hat den Grad 1.

Beweis. Die Ordnungsrelation $\sqsubseteq_{\Sigma^{let}}$ ist eine Partialordnung, weil für alle Sortensymbole $S, R \in S_{\Sigma^{let}}$ gilt $S \sqsubseteq R \wedge R \sqsubseteq S \Rightarrow S = R$ (Antisymmetrie).

Seien $S \subseteq \{V, A, T\}$ und $R \subseteq \{B, U\}$ nichtleere Teilmengen. Dann gilt $lbs(S \cup R) = \emptyset$, weil es kein Sortensymbol $T \in S_{\Sigma^{let}}$ gibt, das mit Elementen aus S und gleichzeitig mit Elementen aus R vergleichbar ist. Deswegen gilt auch $|max(lbs(S \cup R))| = 0$. Ebensowenig existieren $lbs(\{V, A, T\})$, und $lbs(V, A)$.

Betrachte die Teilmenge $\{V, T\} \subset S_{\Sigma^{let}}$, hier gilt $lbs(V, T) = \{V\}$ und $max(\{V\}) = \{V\}$ (wegen $V \not\sqsubset V$) sowie $V \sqsubseteq V$ (wegen Reflexivität) und deshalb $|max(lbs(V, T))| = 1$. Die Argumentation ist analog für die Teilmengen $\{A, T\}, \{B, U\} \subset S_{\Sigma^{let}}$.

Sei $\{R\} \subset S_{\Sigma^{let}}$ eine einelementige Teilmenge. Dann gilt $lbs(\{R\}) = max(lbs(\{R\}))$, wegen $R \sqsubseteq R$ und $R \not\sqsubset R$ für alle R und außerdem $|max(lbs(\{R\}))| = 1$.

Folglich erfüllt $(S_{\Sigma^{let}}, \sqsubseteq_{\Sigma^{let}})$ die maximal-sorts-condition und weil $|max(lbs(S))| \leq 1$ für alle nichtleeren Teilmengen $M \subset S_{\Sigma^{let}}$ gilt, hat die durch Σ^{let} definierte Sortenstruktur den Grad 1. □

Aus dieser Proposition und dem Satz von Walther folgt sofort:

Korollar 3.4.23. Die Signatur Σ^{let} hat den Unifikationstyp *eindeutig*.

Nun wird auch klar, weshalb bei der Definition der Unifikationsregeln (3.4.13) die Annahme getroffen wurde, dass in den betrachteten Sortenstrukturen die glb zweier Elemente eindeutig bestimmt ist und nicht weiter auf den Fall eingegangen wurde, dass eine Menge grö"ster unterer Schranken existiert: In der Signatur Σ^{let} haben wir es mit dem angenehmen Fall zu tun, dass eine eindeutige (oder gar keine) größte untere Schranke existiert, weshalb nach dem Satz von Walther der Unifikationstyp dieser Signatur *eindeutig* ist.

Als Endresultat können wir für die endliche und einfache Signatur Σ^{let} festhalten:

Satz 3.4.24. Für alle Σ^{let}-Unifikationsprobleme P gilt: Wenn P lösbar ist, dann besitzt P einen mgu, der durch den Unifikationsalgorithmus *Unify*(P) berechnet wird. Ist P nicht lösbar, stoppt *Unify*(P) mit der Antwort: P besitzt keine Lösung.

Beweis. Folgt direkt aus Satz 3.4.18 und Korollar 3.4.23. □

Dieses Resultat ist sehr erfreulich, vor allem wenn man bedenkt, dass es endliche und einfache Signaturen gibt, deren Unifikationstyp *endlich* ist. Betrachtet man allgemeine Signaturen (die keinen Beschränkungen wie Regularität oder Endlichkeit unterliegen), dann gilt, dass Σ-Unifikation unentscheidbar ist (siehe Schmidt-Schauß (1989a)).

3.4.3 Effizienz der Unifikation in endlichen und einfachen Signaturen

Der nichtdeterministische Algorithmus *Unify* benötigt exponentiell viel Zeit und Platz, was an folgendem Beispiel verdeutlicht werden soll.

Beispiel 3.4.25. Sei Σ eine Signatur mit nur einer Sorte und alle Symbole seien von dieser Sorte. Betrachte folgendes Σ-Unifikationsproblem

$$\{x_1 \stackrel{?}{=} f(x_0, x_0), x_2 \stackrel{?}{=} f(x_1, x_1), \ldots, x_n \stackrel{?}{=} f(x_{n-1}, x_{n-1}\}.$$

Der idempotente mgu, den man durch wiederholte Anwendung von *Variable Elimination* erhält ist

$$\{x_1 \mapsto f(x_0, x_0), x_2 \mapsto f(f(x_0, x_0), f(x_0, x_0)), \ldots\}.$$

Hier wird jeder Variable x_i ein vollständiger binärer Baum der Höhe i zugeordnet. Folglich ist die Größe des mgu exponentiell in der Größe des zu lösenden Unifikationsproblems. Das gilt auch für alle anderen mgu des Unifikationsproblems, da diese zum oben angegebenen äquivalent modulo Variablenumbenennung sind.

Das Problem in obigem Beispiel ist, dass durch *Variable Elimination* Terme kopiert werden. Unifikationsalgorithmen, die das Kopieren von Termen vermeiden, weisen

einen besseren Zeit- und Platzbedarf auf. Paterson und Wegman (1976) beschreiben eine Methode, mit der in linearer Zeit entschieden werden kann, ob Terme ohne Sorten unifizierbar sind. An dieser Stelle soll (in Anlehnung an Schmidt-Schauß (1989a)) kurz beschrieben werden, wie der Unifikationsalgorithmus von Martelli und Montanari (1982), der eine quasi-lineare Laufzeit besitzt, verwendet werden kann, um Terme über einer einfachen Signatur effizient zu unifizieren. Dieser Algorithmus verwendet Multigleichungen. Die Multigleichung $t_1 =^? t_2 =^? t_3$ entspricht den beiden Gleichungen $t_1 =^? t_2$ und $t_2 =^? t_3$. Wenn ein gegebenes Unifikationsproblem eine Lösung besitzt, dann stoppt der Algorithmus mit einer Menge von Multigleichungen in gelöster Form: $x_1 =^? x_2 \ldots =^? x_n =^? t$, wobei alle Variablen paarweise verschieden sind (über die Menge aller Multigleichungen), und maximal ein Term, der keine Variable ist, pro Multigleichung vorkommt. Diese gelöste Form stellt eine Repräsentation der Lösung dar und noch nicht die Substitution, die das Problem löst. Um die Substitution aus der Repräsentation zu gewinnen, muss diese voll instantiiert werden.

Proposition 3.4.26. Unifikation für einfache Signaturen benötigt höchstens quasi-lineare Zeit.

Beweis. Sei Σ eine einfache Signatur und P ein einfaches Σ-Unifikationsproblem. Dann gehe zur Unifikation von P folgendermaßen vor: Berechne mit dem Unifikationsalgorithmus von Martelli und Montanari eine Menge von Multigleichungen in gelöster Form $\{M_1, \ldots, M_n\}$. Ignoriere dabei alle Sorten, d.h. die Multigleichungen sind i.A. nicht wohlsortiert. Dies geht in quasi-linearer Zeit. Außerdem werden keine neuen Variablen eingeführt. Dann verfahre folgendermaßen mit allen Multigleichungen M_i:

1. M_i besteht nur aus Variablen. Sei X die Menge der in M_i vorkommenden Variablen. Berechne die Menge der (kleinsten) Sorten dieser Variablen $SX := glb(\{LS_\Sigma(x) \mid x \in X\})$, was in linearer Zeit geht, da für endliche Signaturen $LS_\Sigma(x) = \mathbb{S}(x)$ (nach Proposition 3.3.16) gilt. Folgende Fälle werden unterschieden:
   - $SX = \emptyset$, dann besitzt das Unifikationsproblem keine wohlsortierte Lösung.
   - $SX = \{S\}$, dann repräsentiert M_i eine wohlsortierte Komponente der gelösten Form.
   - $SX = \{S_1, \ldots, S_m\}$, dann sind die Elemente der Menge $\{\{x_1 =^? y_{S_i}, \ldots, x_{|X|} =^? y_{S_i}\} \mid i = 1, \ldots m\}$[7] jeweils wohlsortierte Komponenten einer gelösten Form.

[7] Dabei ist y_{S_i} jeweils eine neue Variable, die nicht im ursprünglichen Unifikationsproblem vorkommt.

2. M_i besteht aus Variablen und einem Term t. Sei X die Menge der in M_i vorkommenden Variablen. Berechne $SX := glb(\{LS_\Sigma(x) \mid x \in X\})$ und unterscheide:
   - $SX = \emptyset$, dann besitzt das Unifikationsproblem keine wohlsortierte Lösung.
   - $SX = \{S\}$, und es gilt $S \sqsupseteq LS_\Sigma(t)$, dann repräsentiert M_i eine wohlsortierte Komponente der gelösten Form. Außerdem kann $LS_\Sigma(t)$ in einfachen Signaturen in Linear-Zeit bestimmt werden (siehe Korollar 3.3.14).
   - $SX = \{S_1, \ldots, S_m\}$, dann sind die Elemente der Menge $\{\{x_1 =^? y_{S_i}, \ldots, x_{|X|} =^? y_{S_i}, y_{S_i} =^? t\} \mid i = 1, \ldots m\}$ jeweils wohlsortierte Komponenten einer gelösten Form.

Die komplette Menge der gelösten Formen, die eine vollständige Menge von wohlsortierten Lösungen repräsentiert,[8] erhält man durch die Vereinigung der jeweiligen Komponenten. □

Verdeutlichen wir uns die Arbeitsweise des Algorithmus anhand des Unifikationsproblems aus Beispiel 3.4.4: Zur Signatur $\Sigma = \{A, B \sqsupset C, D\}$ ist das Unifikationsproblem $x_A =^? y_B$ bereits in gelöster Form, wenn die Sorten nicht berücksichtigt werden. Die Gleichung besteht nur aus Variablen und $glb(A, B) = \{C, D\}$. Als Menge von wohlsortierten gelösten Formen (die Repräsentationen der Lösungen) erhält man somit $\{\{x_A =^? z_C, y_B =^? z_C\}, \{x_A =^? z_D, y_B =^? z_D\}\}$.

Der Nachteil des Algorithmus von Martelli und Montanari besteht darin, dass nur die Repräsentation eines allgemeinsten Unifikators in quasi-linearer Zeit berechnet wird. Wird die zugehörige Substitution benötigt, muss instantiiert werden, wobei wieder Terme kopiert werden, was zu exponentiellem Zeit und Platzbedarf führen kann. In Beispiel 3.4.25 repräsentiert das Unifikationsproblem bereits eine Lösung. Wird die Substitution benötigt, muss voll eingesetzt werden, was in exponentiellem Zeit- und Platzbedarf resultiert.

Unifikatoren können durch gerichtete azyklische Graphen (so genannte DAGs) in polynomieller Zeit dargestellt werden (Corbin & Bidoit, 1983).

[8] Da durch den Algorithmus neue Variablen eingeführt werden, ist die aus den Repräsentationen zu gewinnende vollständigen Menge der Lösungen einzuschränken, auf die in P vorkommenden Variablen, $CU_\Sigma(P)|_{Var(P)}$.

4 Unifikation von `letrec`-Umgebungen

Eine `letrec`-Umgebung des Λ^{let} Kalküls besteht im Allgemeinen aus einer Umgebungsvariablen (der Sorte U) und einer beliebigen endlichen Menge von Bindungen $\{E, x_1 = s_1, \ldots x_m = s_m\}$. Dabei ist in der Syntaxdefinition von Λ^{let} festgelegt, dass die Elemente der `letrec`-Umgebung *kommutativ* sind. D.h zwei `letrec`-Ausdrücke, die bis auf die Reihenfolge der Bindungen gleich sind, werden als syntaktisch äquivalent angesehen, wie beispielsweise

$(\texttt{letrec}\ \{x_1 = s_1, x_2 = s_2, x_3 = s_3\}\ \texttt{in}\ t) \equiv$ [1] $(\texttt{letrec}\ \{x_3 = s_3, x_1 = s_1, x_2 = s_2\}\ \texttt{in}\ t)$.

Diese Bedingung muss auch für `letrec`-Umgebungen in $T(\Sigma^{let}, X)$ gelten. Beliebige Vertauschbarkeit von Elementen kann in $T(\Sigma^{let}, X)$ modelliert werden durch ein Funktionssymbol, das sich wie ein Multimengenkonstruktor verhält, da in einer Menge die Reihenfolge der Elemente ebenfalls keine Rolle spielt. Dabei sind zwei verschiedene Vorgehensweisen möglich, `letrec`-Umgebungen aus Λ^{let} in Σ^{let}-Terme zu übersetzen:

1. `letrec`-Umgebungen werden dargestellt durch ein einstelliges Funktionssymbol $\{\cdot\} : B \to U$, das Umgebungen aus einer einzelnen Bindung konstruiert. Beliebige Umgebungen, mit mehr als einem Element, werden durch die Vereinigung von Umgebungen erzeugt mit dem Funktionssymbol $\cup : U \to U \to U$. Ein Beispiel für eine Übersetzung ist

   $$[\![\{x_1 = s_1, x_2 = s_2, x_3 = s_3\}]\!] = \{b(x_1, s_1)\} \cup (\{b(x_2, s_2)\} \cup \{b(x_3, s_3)\}),$$

   wobei $\{e\}$ anstelle von $\{\cdot\}(e)$, $d \cup e$ anstatt $\cup(d, e)$ geschrieben wird und *bind* als b abgekürzt wird. Um die Kommutativität der Elemente zu gewährleisten, muss für das Funktionssymbol $\cup$ gelten:

   $$\begin{aligned} d_U \cup e_U &\approx e_U \cup d_U && \text{Kommutativitaet}, \\ c_U \cup (d_U \cup e_U) &\approx (c_U \cup d_U) \cup e_U && \text{Assoziativitaet}. \end{aligned}$$

2. Oder Umgebungen werden dargestellt durch ein zweistelliges Funktionssymbol $\{\!|\cdot|\cdot|\!\} : B \to U \to U$, das, vergleichbar mit einem Listenkonstruktor, eine

[1] $\equiv$ bezeichnet hier die syntaktische Gleichheit von Λ^{let}-Ausdrücken.

Umgebung aus einem Term der Sorte B (einer Bindung) und einer Umgebung konstruiert. Wir schreiben $\{d|e\}$ anstatt $\{\cdot|\cdot\}(d,e)$. Als zusätzliches Konstantensymbol wird die leere Umgebung $\emptyset : U$ benötigt. Eine beispielhafte Übersetzung ist

$$[\![\{x_1 = s_1, x_2 = s_2, x_3 = s_3\}]\!] = \{\!|(b(x_1,s_1)|\{\!|b(x_2,s_2)|\{\!|b(x_3,s_3)|\emptyset|\!\}|\!\}|\!\}.$$

Das Axiom, das die Irrelevanz der Reihenfolge der Elemente zusichert, ist:

$$\{\!|c_B|\{\!|d_B|e_U|\!\}|\!\} \approx \{\!|d_B|\{\!|c_B|e_U|\!\}|\!\} \text{ Links-Kommutativitaet.}$$

Die beiden Darstellungsmöglichkeiten modellieren die Kommutativitätsbedingung von letrec-Umgebungen in $T(\Sigma^{let}, X)$, weil die Axiome (das sind die $\approx$ Gleichungen) verwendet werden können, um Terme äquivalent umzuformen. Betrachte etwa die Übersetzung, die Umgebungen durch das assoziative und kommutative Funktionssymbol $\cup$ darstellt:

$$\begin{array}{ccc}
\{x_1 = s_1, x_2 = s_2, x_3 = s_3\} & \equiv & \{x_3 = s_3, x_1 = s_1, x_2 = s_2\} \\
\downarrow Uebersetzung & & \downarrow Uebersetzung \\
\{b(x_1,s_1)\} \cup (\{b(x_2,s_2)\} \cup \{b(x_3,s_3)\}) & & \{b(x_3,s_3)\} \cup (\{b(x_1,s_1)\} \cup \{b(x_2,s_2)\}) \\
\downarrow Assoziativitaet & & \downarrow Kommutativitaet \\
(\{b(x_1,s_1)\} \cup \{b(x_2,s_2)\}) \cup \{b(x_3,s_3)\} & = & (\{b(x_1,s_1)\} \cup \{b(x_2,s_2)\}) \cup \{b(x_3,s_3)\}
\end{array}$$

Unter Berücksichtigung der Assoziativität und der Kommutativität des Funktionssymbols $\cup$ sind die beiden Umgebungen gleich. Man schreibt dafür

$$\{b(x_1,s_1)\} \cup (\{b(x_2,s_2)\} \cup \{b(x_3,s_3)\}) =_{AC} \{b(x_3,s_3)\} \cup (\{b(x_1,s_1)\} \cup \{b(x_2,s_2)\}).$$

Betrachtet man die Darstellung mit dem links-kommutativen Funktionssymbol $\{\!| \cdot | \cdot |\!\}$, dann kann dessen Eigenschaft zum Umformen von Termen verwendet werden:

$$\begin{array}{ccc}
\{x_1 = s_1, x_2 = s_2, x_3 = s_3\} & \equiv & \{x_3 = s_3, x_1 = s_1, x_2 = s_2\} \\
\downarrow Uebersetzung & & \downarrow Uebersetzung \\
\{\!|b(x_1,s_1)|\{\!|b(x_2,s_2)|\{\!|b(x_3,s_3)|\emptyset|\!\}|\!\}|\!\} & & \{\!|b(x_3,s_3)|\{\!|b(x_1,s_1)|\{\!|b(x_2,s_2)|\emptyset|\!\}|\!\}|\!\} \\
\downarrow Links-Kommutativitaet & & \downarrow Links-Kommutativitaet \\
\{\!|b(x_1,s_1)|\{\!|b(x_3,s_3)|\{\!|b(x_2,s_2)|\emptyset|\!\}|\!\}|\!\} & = & \{\!|b(x_1,s_1)|\{\!|b(x_3,s_3)|\{\!|b(x_2,s_2)|\emptyset|\!\}|\!\}|\!\}
\end{array}$$

Unabhängig davon, welche der beiden Übersetzungen zur Darstellung von letrec-Umgebungen verwendet wird, müssen bei der Unifikation die durch Gleichungen

($\approx$) definierten semantischen Eigenschaften von Funktionssymbolen berücksichtigt werden. Beispielsweise hat das Unifikationsproblem

$$\{\!|c_B|\{\!|b(x_V, s_T)|\emptyset|\!\}\!|\} \stackrel{?}{=} \{\!|b(y_V, t_T)|\{\!|d_B|\emptyset|\!\}\!|\},$$

für die in Kapitel 3 beschriebene Methode der syntaktischen Unifikation, einen mgu $\sigma = \{c \mapsto b(y,t), d \mapsto b(x,s)\}$. Wird berücksichtigt, dass das Funktionssymbol $\{\!|\cdot,\cdot|\!\}$ links-kommutativ ist, dann hat das Problem einen weiteren Unifikator $\tau = \{c \mapsto d, x \mapsto y, s \mapsto t\}$, für den $\sigma \not\leq_\Sigma \tau$ und $\tau \not\leq_\Sigma \sigma$ gilt.

Die Unifikationsmethode, die durch Gleichungen definierte Eigenschaften von Funktionssymbolen berücksichtigt, wird als *Gleichungsunifikation* (kurz *E-Unifikation* aus dem Englischen für *Equational Unification*) bezeichnet. Gegeben zwei Terme s, t und eine Menge E von *Gleichungen* (auch *Identitäten* genannt), versucht die E-Unifikation eine Substitution σ zu finden, so dass σs und σt gleich sind *modulo der durch E definierten Identitäten*: $\sigma s =_E \sigma t$.

Dieses Kapitel beschäftigt sich mit dem oben skizzierten Problem, wie `letrec`-Umgebungen mit vertauschbaren Elementen unifiziert werden können. Es ist folgendermaßen strukturiert:

In Abschnitt 4.1 werden *Gleichungstheorien* (informell) eingeführt, die den theoretischen Rahmen bereitstellen, um Unifikation von Funktionssymbolen mit vertauschbaren Argumenten zu behandeln. Die Definitionen verschiedener Begriffe wie Unifikationsproblem, Instantiierungs-Quasiordnung und Unifikationstyp werden auf die Situation angepasst, dass bei der Unifikation eine Gleichungstheorie zu berücksichtigen ist. Außerdem werden die beiden Gleichungstheorien $\mathcal{E}_{Cl}^{let}$ und $\mathcal{E}_{AC}^{let}$ vorgestellt, die Vertauschbarkeit von Bindungen in `letrec`-Umgebungen darstellen können.

Im darauf folgenden Abschnitt 4.2 wird kurz eine vollständige Unifikationsprozedur vorgestellt, die zur Unifikation in beliebigen Gleichungstheorien verwendet werden kann. Die Nachteile dieser Prozedur werden diskutiert und es wird erläutert, warum sie für die Unifikation von `letrec`-Umgebungen nicht geeignet ist.

Abschnitt 4.3 charakterisiert das Verhältnis von Sorten zu den Identitäten, die eine Gleichungstheorie definieren. Gelten bestimmte Bedingungen für dieses Verhältnis, so können Unifikatoren auf besonders einfache Weise berechnet werden. In Unterabschnitt 4.3.1 wird dargelegt, dass die beiden Gleichungstheorien, die Vertauschbarkeit von Elementen in `letrec`-Umgebungen axiomatisieren, diese Bedingungen erfüllen. Weitere Eigenschaften der beiden Theorien werden kurz diskutiert. Außerdem wird begründet, weshalb der Theorie $\mathcal{E}_{Cl}^{let}$, zur Darstellung und Unifikation von `letrec`-Umgebungen in dieser Arbeit der Vorzug vor $\mathcal{E}_{AC}^{let}$ gegeben wird.

Abschnitt 4.4 beschäftigt sich mit der Unifikation von Termen in der Theorie $\mathcal{E}_{Cl}^{let}$. Es wird ein Unifikationsalgorithmus vorgestellt, der terminiert (Abschnitt 4.4.1) und vollständig ist (Abschnitt 4.4.2). Der abschließende Teil 4.4.3 geht kurz auf den

Unifikationstyp der Theorie $\mathcal{E}_{Cl}^{let}$ und die Komplexität der Unifikation in dieser Theorie ein. Außerdem wird erklärt, wie die komplette vollständige Menge von Unifikatoren berechnet werden kann, die zur Berechnung aller Überlappungen benötigt wird.

4.1 Grundlegende Definitionen

Definition 4.1.1 (Σ-Identitäten)**.** Sei Σ eine Signatur (mit Sorten) und X eine abzählbar unendliche Menge von Variablen, die disjunkt ist zur Menge der Funktionssymbole. Eine Σ-*Identität* (oder einfach Identität) ist ein Paar $(s,t) \in T(\Sigma,X) \times T(\Sigma,X)$. Identitäten werden geschrieben als $s \approx t$, wobei s linke Seite (*lhs*) und r rechte Seite (*rhs*) heißt.

Identitäten können verwendet werden, um Terme zu anderen äquivalenten Termen zu transformieren, indem die Instanz einer linken Seite durch die entsprechende Instanz der rechten Seite ersetzt werden (und umgekehrt). Eine Identität wird auch *Axiom* genannt. Auf das Verhältnis von Identitäten zu Sorten wird im Abschnitt 4.3 näher eingegangen.

Beispiele für Identitäten, die Eigenschaften von Funktionssymbolen axiomatisieren sind:

$$
\begin{array}{lll}
A_f & = \{f(x,f(y,z)) \approx f(f(x,y),z)\} & \text{Assoziativität,} \\
C_f & = \{f(x,y) \approx f(y,x)\} & \text{Kommutativität,} \\
AC_f & = A_f \cup C_f & \text{Assoziativität und Kommutativität,} \\
U_f & = \{f(x,e) \approx x, f(e,x) \approx x\} & \text{Einheitselement,} \\
I_f & = \{f(x,x) \approx x\} & \text{Idempotenz,} \\
Cl_f & = \{f(x,f(y,z)) \approx f(y,f(x,z))\} & \text{Links-Kommutativität.}
\end{array}
$$

Ein assoziatives und kommutatives Funktionssymbol wird als AC-Funktionssymbol, ein links-kommutatives Symbol als Cl-Funktionssymbol, bezeichnet.

Eine *Gleichungstheorie* wird definiert durch eine Menge von Identitäten E. Sie ist die kleinste Kongruenzrelation auf der Termalgebra $\mathcal{T}(\Sigma,X)$, die E enthält und abgeschlossen ist unter Substitution und wird durch $=_{\Sigma,E}$ bezeichnet.[2] Gilt $s =_{\Sigma,E} t$, dann sagt man s und t sind *gleich modulo* E. Der Begriff "'Gleichungstheorie"' wird hier (leicht missbräuchlich) folgendermaßen verwendet: Wir sprechen von $\mathcal{E} = (\Sigma, E)$ als der Gleichungstheorie $\mathcal{E}$, die durch die Identitäten in E über der Signatur Σ definiert ist. Für eine gegebene Gleichungstheorie schreiben wir verkürzt

[2] Für eine ausführlichere Definition von Gleichungstheorien siehe Baader und Nipkow (1998), bzw. für den Fall mit Sorten Schmidt-Schauß (1989a).

$s =_{\mathcal{E}} t$, für die Kongruenzrelation $s =_{\Sigma,E} t$. Gleichungstheorien dienen dazu, semantische Eigenschaften von Funktionssymbolen darzustellen und bei der Unifikation zu berücksichtigen. Außerdem bilden sie die Basis für die Theorie der Termersetzung (Baader & Nipkow, 1998; Bezem et al., 2003).

Definition 4.1.2. Die beiden Gleichungstheorien, die Vertauschbarkeit von `letrec`-Umgebungen in $T(\Sigma^{let}, X)$ darstellen, sind folgenderma"sen definiert:

$\mathcal{E}_{AC}^{let}$ besteht aus der Signatur $\Sigma_{\cup}^{let}$, die folgenderma"sen definiert ist:

$$\Sigma^{let} \cup \{\ \{\cdot\} : B \to U, \quad \cup : U \to U \to U\ \}$$

zusammen mit folgender Menge von Identitäten:

$$\{\ c_U \cup (d_U \cup e_U) \approx (c_U \cup d_U) \cup e_U\ \ (A)$$
$$d_U \cup e_U \approx e_U \cup d_U\ \ (C)\ \}$$

$\mathcal{E}_{Cl}^{let}$ besteht aus der Signatur $\Sigma_{\{|\cdot|\cdot|\}}^{let}$, die folgenderma"sen definiert ist:

$$\Sigma^{let} \cup \{\ \{\!|\cdot|\cdot|\!\} : B \to U \to U, \quad \emptyset : U\ \}$$

und der Identität

$$\{\ \{\!|c_B|\{\!|d_B|e_U|\!\}|\!\} \approx \{\!|d_B|\{\!|c_B|e_U|\!\}|\!\}\ \ (Cl)\ \}$$

Bemerkung 4.1.3. Die $\mathcal{E}_{AC}^{let}$ Theorie und die $\mathcal{E}_{Cl}^{let}$ Theorie axiomatisieren beide die Vertauschbarkeit von Elementen in `letrec`-Umgebungen. Trotzdem gibt es Unterschiede zwischen den beiden Theorien, insbesondere bezüglich der Ausdruckskraft der zugrunde liegenden Signaturen. Ein Unterschied besteht darin, dass Umgebungen, die mehr als eine Variable der Sorte U enthalten, durch die Signatur $\Sigma_{\cup}^{let}$ aber nicht durch $\Sigma_{\{|\cdot|\cdot|\}}^{let}$ dargestellt werden können. Beispielsweise kann $\{d_U\} \cup \{e_U\}$ durch $\Sigma_{\cup}^{let}$ aber nicht durch $\Sigma_{\{|\cdot|\cdot|\}}^{let}$ repräsentiert werden. Der Grund hierfür ist, dass das Funktionssymbol $\{\!|\cdot|\cdot|\!\} : B \to U \to U$ eingeschränkt ist, weil es als erstes Argument einen Term der Sorte B erwartet. In $\Sigma_{\{|\cdot|\cdot|\}}^{let}$ können deshalb nur Umgebungen mit maximal einer Variablen der Sorte U dargestellt werden. Dies ist allerdings ausreichend zur Berechnung aller Überlappungen für Gabeldiagramme, da für alle linken Seiten der Reduktionsregeln des Λ^{let}-Kalküls gilt, dass sie maximal über eine Umgebungsvariable verfügen. In der Signatur $\Sigma_{\cup}^{let}$ unterliegt das Funktionssymbol $\cup : U \to U \to U$ diesen Beschränkungen nicht. D.h. die $\mathcal{E}_{Cl}^{let}$ Theorie stellt eine "'Einschränkung"' der allgemeineren Theorie $\mathcal{E}_{AC}^{let}$ dar.

Eine Entscheidung, welche der beiden Theorien in dieser Arbeit zur Darstellung von `letrec`-Umgebungen verwendet wird, wird erst später getroffen (in Abschnitt 4.3.1). Zuvor werden Themen behandelt, die für beliebige Gleichungstheorien gelten.

Wir passen die im Kapitel über syntaktische Unifikation von Termen mit Sorten definierten Begriffe auf die Situation an, dass eine Gleichungstheorie $\mathcal{E}$ gegeben ist.

Definition 4.1.4 ($\mathcal{E}$-Unifikationsproblem)**.** Sei $\mathcal{E} = (\Sigma, E)$ eine Gleichungstheorie. Ein *$\mathcal{E}$-Unifikationsproblem* ist eine endliche Menge von Gleichungen

$$P = \{s_1 =^?_{\mathcal{E}} t_1, \ldots, s_n =^?_{\mathcal{E}} t_n\}$$

zwischen Σ-Termen. Ein *$\mathcal{E}$-Unifikator* (bzw. eine *$\mathcal{E}$-Lösung*) ist eine Substitution $\sigma \in Sub_\Sigma$, so dass $\sigma s_1 =_{\mathcal{E}} \sigma t_1, \ldots, \sigma s_n =_{\mathcal{E}} \sigma t_n$. Die Menge aller $\mathcal{E}$-Unifikatoren von P wird mit $U_{\mathcal{E}}(P)$ bezeichnet; P ist *lösbar*, gdw. $U_{\mathcal{E}}(P) \neq \emptyset$.

Ist der Bezug zu $\mathcal{E}$ klar, wird verkürzt $=^?$ anstatt $=^?_{\mathcal{E}}$ geschrieben.

Syntaktische Σ-Unifikation (wie in Kapitel 3 behandelt) ist ein Spezialfall der $\mathcal{E}$-Unifikation mit $E = \emptyset$. Jeder syntaktische $\mathcal{E}$-Unifikator ist auch ein $\mathcal{E}$-Unifikator, aber für $E \neq \emptyset$ kann die Menge $U_{\mathcal{E}}(P)$ zusätzliche Elemente besitzen.

$\mathcal{E}$-Gleichheit wird auf wohlsortierte Substitutionen $\sigma, \tau \in Sub_\Sigma$ ausgeweitet:

$$\sigma =_{\mathcal{E}} \tau \text{ , gdw. } \sigma x =_{\mathcal{E}} \tau x \text{ für alle Variablen} x \in X.$$

Ist man nur am Verhalten auf einer speziellen Menge von Variablen $W \subseteq X$ interessiert, schreibt man wie gewohnt:

$$\sigma =_{\mathcal{E}} \tau[W] \text{ , gdw. } \sigma x =_{\mathcal{E}} \tau x \ \forall x \in W.$$

Die Instantiierungs-Quasiordnung auf der Menge der wohlsortierten Substitutionen wird ebenfalls angepasst.

Definition 4.1.5. Sei $\mathcal{E}$ eine Gleichungstheorie, $W \subseteq X$ eine Menge von Variablen und $\sigma, \tau \in Sub_\Sigma$ wohlsortierte Substitutionen.

Die Substitution σ ist *allgemeiner modulo E auf W* als die Substitution τ, gdw. es eine Substitution δ gibt, so dass $\tau =_{\mathcal{E}} \delta\sigma[W]$. In diesem Fall schreibt man $\sigma \lesssim_{\mathcal{E}} \tau[W]$ und bezeichnet τ als *$\mathcal{E}$-Instanz* von σ auf W.

Werden $\mathcal{E}$-Unifikatoren eines Unifikationsproblems P verglichen, dann ist die Menge der Variablen W, die Menge der in P vorkommenden Variablen $Var(P)$. Die Relation $\lesssim_{\mathcal{E}}$ ist eine Quasiordnung auf der Menge der wohlsortierten Substitutionen, die zugehörige Äquivalenzrelation ist $\sim_{\mathcal{E}} := \lesssim_{\mathcal{E}} \cap \gtrsim_{\mathcal{E}}$.

Die Situation, dass im Allgemeinen kein einzelner mgu, sondern eine vollständige Menge von $\mathcal{E}$-Unifikatoren betrachtet werden muss, bleibt bestehen.

Definition 4.1.6. Eine *vollständige Menge von E-Unifikatoren* für ein $\mathcal{E}$-Unifikationsproblem P ist eine Menge von (wohlsortierten) Substitutionen $CU_{\mathcal{E}}(P)$, so dass

- $CU_{\mathcal{E}}(P) \subseteq U_{\mathcal{E}}(P)$ und
- für alle $\tau \in U_{\mathcal{E}}(P)$ gibt es $\sigma \in CU_{\mathcal{E}}(P)$, so dass $\sigma \lesssim_{\mathcal{E}} \tau[Var(P)]$.

Eine *minimale vollständige Menge von $\mathcal{E}$-Unifikatoren* für ein $\mathcal{E}$-Unifikationsproblem P ist eine vollständige Menge von Unifikatoren $MU_{\mathcal{E}}(P)$, so dass die Substitutionen in $MU_{\mathcal{E}}(P)$ bezüglich $\lesssim_{\mathcal{E}} [Var(P)]$ nicht vergleichbar sind:

- Für alle $\sigma, \tau \in MU_{\mathcal{E}}(P)$ gilt: $\sigma \lesssim_{\mathcal{E}} \tau[Var(P)]$ impliziert $\sigma =_{\mathcal{E}} \tau[Var(P)]$

Eine Substitution σ ist ein *allgemeinster $\mathcal{E}$-Unifikator* von P (kurz *$\mathcal{E}$-mgu*), gdw. $\{\sigma\}$ eine minimale vollständige Menge von Lösungen für P ist.

Ist ein $\mathcal{E}$-Unifikationsproblem nicht unifizierbar, dann ist die leere Menge eine minimale, vollständige Menge von $\mathcal{E}$-Unifikatoren von P. Die minimale vollständige Menge von $\mathcal{E}$-Unifikatoren eines $\mathcal{E}$-Unifikationsproblems ist eindeutig bestimmt, bis auf die Äquivalenzrelation $\sim_{\mathcal{E}} [Var(P)]$. Insbesondere haben alle minimalen, vollständigen Mengen von Unifikatoren die gleiche Kardinalität. Deswegen kann der Unifikationstyp einer Gleichungstheorie definiert werden.

Definition 4.1.7. Eine Gleichungstheorie $\mathcal{E} = (\Sigma, E)$ besitzt den *Unifikationstyp*

eindeutig, gdw. $MU_{\mathcal{E}}(P)$ existiert und $|MU_{\mathcal{E}}(P)| \leq 1$ für alle $\mathcal{E}$-Unifikationsprobleme P gilt.

endlich, gdw. $MU_{\mathcal{E}}(P)$ existiert und $|MU_{\mathcal{E}}(P)| \leq \infty$ für alle $\mathcal{E}$-Unifikationsprobleme P gilt.

unendlich, gdw. $MU_{\mathcal{E}}(P)$ existiert für alle $\mathcal{E}$-Unifikationsprobleme P und es ein $\mathcal{E}$-Unifikationsproblem P mit $|MU_{\mathcal{E}}(P)| = \infty$ gibt.

null, gdw. es ein $\mathcal{E}$-Unifikationsproblem P gibt, für das $MU_{\mathcal{E}}(P)$ nicht existiert.

4.2 Syntaktische Unifikation in beliebigen Gleichungstheorien

Die $\mathcal{E}$-Unifikation, bezüglich einer beliebigen Gleichungstheorie $\mathcal{E}$, benötigt eine Methode, um die Identitäten, die $\mathcal{E}$ definieren, in die Berechnung einer vollständigen Menge von $\mathcal{E}$-Unifikatoren einzubeziehen. Eine allgemeine Methode, die sich auf keine spezifische Gleichungstheorie $\mathcal{E}$ beschränkt, erhält man durch die Erweiterung der Unifikationsregeln (für Terme über regulären Signaturen: Definition 3.4.13) durch folgende Regel (nach Schmidt-Schauß (1989a)):

Definition 4.2.1. Sei $\mathcal{E} = (\Sigma, E)$ eine Gleichungstheorie mit einer regulären Signatur Σ. Die Unifikationsregeln für endliche und reguläre Signaturen werden um folgende Regel erweitert, um beliebige $\mathcal{E}$-Unifikationsprobleme zu lösen:

Mutation

$$\{s =^? t\} \uplus P \Rightarrow \{s =^? l, t =^? r\} \cup P$$

Wobei $l \approx r$ eine Version mit neuen Variablen von $l \approx r \in E \cup E^-$ ist und

- t keine Variable ist und
- wenn l keine Variable ist, dann sind die Wurzelsymbole von s und l gleich und als nächste Transformation wird *Decomposition* auf $s =^? l$ oder $t =^? r$ angewendet.

Schmidt-Schauß (1989a) zeigt, dass die Unifikationsregeln für reguläre Signaturen zusammen mit der Regel *Mutation* eine vollständige Unifikationsprozedur für $\mathcal{E}$-Unifikationsprobleme zu einer beliebigen Gleichungstheorie $\mathcal{E}$ darstellen (zur Vollständigkeit im unsortierten Fall siehe Gallier und Snyder (1989)). An dieser Stelle wird nicht auf den Vollständigkeitsbeweis eingegangen, sondern wir wollen an einem Beispiel betrachten, wie die Unifikationsprozedur mit der *Mutation*-Regel einen $\mathcal{E}$-Unifikator für ein $\mathcal{E}$-Unifikationsproblem P berechnet.

Beispiel 4.2.2. Sei $P := \{\{\!|c, \{\!|b(x,s), \emptyset|\!\}|\!\} =^? \{\!|b(y,t), \{\!|d, \emptyset|\!\}|\!\}\}$ ein $\mathcal{E}_{Cl}^{let}$-Unifikationsproblem mit den Variablensorten

$$\begin{aligned}
&\mathbb{S}(c) = \mathbb{S}(d) = \mathbb{S}(b_i) = B,\\
&\mathbb{S}(e_i) = U,\\
&\mathbb{S}(x) = \mathbb{S}(y) = V,\\
&\mathbb{S}(s) = \mathbb{S}(t) = T.
\end{aligned}$$

Eine mögliche Folge von Transformationen des Problems ist:[3]

$$\begin{aligned}
&\{\{\!|c|\{\!|b(x,s)|\emptyset|\!\}|\!\} =^? \{\!|b(y,t)|\{\!|d|\emptyset|\!\}|\!\}\}\\
\xrightarrow{Mu\text{-}Cl}\ &\{\{\!|c|\{\!|b(x,s)|\emptyset|\!\}|\!\} =^? \{\!|b_1|\{\!|b_2|e_1|\!\}|\!\}, \{\!|b(y,t)|\{\!|d|\emptyset|\!\}|\!\} =^? \{\!|b_2|\{\!|b_1|e_1|\!\}|\!\}\}\\
\xrightarrow{Dec\times 2}\ &\{c =^? b_1, b(x,s) =^? b_2, \emptyset =^? e_1, \{\!|b(y,t)|\{\!|d|\emptyset|\!\}|\!\} =^? \{\!|b_2|\{\!|b_1|e_1|\!\}|\!\}\}\\
\xrightarrow{Orient\times 2}\ &\{c =^? b_1, b_2 =^? b(x,s), e_1 =^? \emptyset, \{\!|b(y,t)|\{\!|d|\emptyset|\!\}|\!\} =^? \{\!|b_2|\{\!|b_1|e_1|\!\}|\!\}\}\\
\xrightarrow{Elim\ b_2}\ &\{c =^? b_1, b_2 =^? b(x,s), e_1 =^? \emptyset, \{\!|b(y,t)|\{\!|d|\emptyset|\!\}|\!\} =^? \{\!|b(x,s)|\{\!|b_1|e_1|\!\}|\!\}\}\\
\xrightarrow{Dec\times 2}\ &\{c =^? b_1, b_2 =^? b(x,s), e_1 =^? \emptyset, b(y,t) =^? b(x,s), d =^? b_1, \emptyset =^? e_1\}\\
\xrightarrow{Elim\ e_1}\ &\{c =^? b_1, b_2 =^? b(x,s), e_1 =^? \emptyset, b(y,t) =^? b(x,s), d =^? b_1, \emptyset =^? \emptyset\}\\
\xrightarrow{Taut}\ &\{c =^? b_1, b_2 =^? b(x,s), e_1 =^? \emptyset, b(y,t) =^? b(x,s), d =^? b_1\}\\
\xrightarrow{Dec}\ &\{c =^? b_1, b_2 =^? b(x,s), e_1 =^? \emptyset, y =^? x, t =^? s, d =^? b_1\}
\end{aligned}$$

[3] *Mu-Cl* steht für einen *Mutation*-Schritt mit dem *Cl*-Axiom. Die Namen der anderen Unifikationsregeln sind ebenfalls abgekürzt. $\times n$ bedeutet n-fache Anwendung einer Unifikationsregel.

Das Unifikationsproblem in der letzten Zeile ist in (wohlsortierter) gelöster Form und es gilt $\sigma_S|_{Var(P)} \in CU_{\mathcal{E}_{Cl}^{let}}(P)$.

Es ist leicht zu sehen, wie die Folge der Transformationen (ohne *Mutation*) auszusehen hat, die den anderen in der Einleitung angegebenen Unifikator zu obigem Beispielproblem findet. Die Vollständigkeit der Unifikationsprozedur ist also auch neben dem Beweis intuitiv plausibel. Der Vorteil, dass diese Unifikationsprozedur mit beliebigen Gleichungstheorien verwendbar ist, wird von dem Nachteil wett gemacht, dass sich außer der Vollständigkeit der Prozedur nicht viel über ihre Eigenschaften sagen lässt. Dass sie beispielsweise für $\mathcal{E}_{Cl}^{let}$-Unifikationsprobleme nicht immer terminiert, verdeutlicht folgendes Beispiel.

Beispiel 4.2.3. Wir betrachten die Situation wie in Beispiel 4.2.2. Eine mögliche nicht terminierende Folge von Transformationsschritten ist:

$$
\begin{aligned}
& \{\{\!|c|\{\!|b(x,s)|\emptyset|\!\}|\!\} \stackrel{?}{=} \{\!|b(y,t)|\{\!|d|\emptyset|\!\}|\!\}\} \\
\xrightarrow{\textit{Mu-Cl}} \; & \{\{\!|c|\{\!|b(x,s)|\emptyset|\!\}|\!\} \stackrel{?}{=} \{\!|b_1|\{\!|b_2|e_1|\!\}|\!\}, \{\!|b(y,t)|\{\!|d|\emptyset|\!\}|\!\} \stackrel{?}{=} \{\!|b_2|\{\!|b_1|e_1|\!\}|\!\}\} \\
\xrightarrow{\textit{Dec}} \; & \{c \stackrel{?}{=} b_1, \{\!|b(x,s)|\emptyset|\!\} \stackrel{?}{=} \{\!|b_2|e_1|\!\}, \{\!|b(y,t)|\{\!|d|\emptyset|\!\}|\!\} \stackrel{?}{=} \{\!|b_2|\{\!|b_1|e_1|\!\}|\!\}\} \\
\xrightarrow{\textit{Mu-Cl}} \; & \{c \stackrel{?}{=} b_1, \{\!|b(x,s)|\emptyset|\!\} \stackrel{?}{=} \{\!|b_2|e_1|\!\}, \{\!|b(y,t)|\{\!|d|\emptyset|\!\}|\!\} \stackrel{?}{=} \{\!|b_3|\{\!|b_4|e_2|\!\}|\!\}, \\
& \quad \{\!|b_2|\{\!|b_1|e_1|\!\}|\!\} \stackrel{?}{=} \{\!|b_4|\{\!|b_3|e_2|\!\}|\!\}\} \\
\xrightarrow{\textit{Dec}} \; & \{c \stackrel{?}{=} b_1, \{\!|b(x,s)|\emptyset|\!\} \stackrel{?}{=} \{\!|b_2|e_1|\!\}, b(y,t) \stackrel{?}{=} b_3, \{\!|d|\emptyset|\!\} \stackrel{?}{=} \{\!|b_4|e_2|\!\}, \\
& \quad \{\!|b_2|\{\!|b_1|e_1|\!\}|\!\} \stackrel{?}{=} \{\!|b_4|\{\!|b_3|e_2|\!\}|\!\}\} \\
\xrightarrow{\textit{Mu-Cl}} \; & \ldots
\end{aligned}
$$

Nach jeder Anwendung des *Cl*-Axioms wird eine *Decomposition*-Transformation möglich, die direkt wieder von einem *Cl*-Schritt gefolgt werden kann. Die Anwendung des Axioms wird von der Unifikationsprozedur nicht beschränkt, abgesehen von den oben genannten Bedingungen (Def. 4.2.1), die hier erfüllt sind. Dadurch ist es möglich, die Bindungen einer Umgebung beliebig oft zu vertauschen. Dies entspricht auch der mathematischen Intuition, hat hier allerdings die Nichtterminierung der Unifikation zur Folge.

Unifikation modulo beliebiger Gleichungstheorien ist generell unentscheidbar, ebenso wie Eigenschaften allgemeiner $\mathcal{E}$-Unifikationsprobleme (siehe z.B. (Nutt, 1991) zur Unentscheidbarkeit des Unifikationstyps). Allgemeine Methoden, die $\mathcal{E}$-Unifikatoren für beliebige Gleichungstheorien berechnen, liefern aus diesem Grund nur schwache Ergebnisse. So kann die oben skizzierte Methode nur verwendet werden, um eine vollständige Menge von Unifikatoren rekursiv aufzuzählen. Sie liefert somit keine Entscheidungsprozedur für $\mathcal{E}$-Unifizierbarkeit, selbst wenn in der betrachteten Gleichungstheorie das Unifikationsproblem entscheidbar ist und der Unifikationstyp der Theorie eindeutig oder endlich ist.

Um bessere Berechenbarkeitsresultate für spezifische Gleichungstheorien zu erhalten, wendet man sich einzelnen Theorien zu. Die beiden Theorien (ohne Signaturen mit Sorten) AC und Cl besitzen eine Reihe guter Eigenschaften. Beide sind entscheidbar, allerdings ist Unifikation in beiden Theorien NP-vollständig und beide Theorien haben den Unifikationstyp endlich. Außerdem existieren für die Theorien Unifikationsalgorithmen. Zur AC-Theorie und zu Unifikationsalgorithmen siehe beispielsweise: Livesey und Siekmann (1976); Stickel (1981); Fortenbacher (1985); Büttner (1986); Lincoln und Christian (1989); Boudet, Contejean, und Devie (1990). Zur Cl-Theorie und zu Unifikationsalgorithmen siehe beispielsweise: Dovier, Policriti, und Rossi (1998); Dovier, Pontelli, und Rossi (1998); Dantsin und Voronkov (1999); Stolzenburg (1999); Dovier, Pontelli, und Rossi (2006)

Allerdings berücksichtigt keiner dieser Algorithmen Sorten. Um zu klären, wie sich das Verhältnis von Sorten zu Identitäten, die eine Gleichungstheorie definieren, auf die $\mathcal{E}$-Unifikation für spezifische Theorien auswirkt, werden zunächst Charakterisierungen für Gleichungstheorien über Signaturen mit Sorten betrachtet. Dabei werden wir Folgendes feststellen: Wenn bestimmte Bedingungen für eine Gleichungstheorie $\mathcal{E}$ über einer Signatur mit Sorten gelten, dann können $\mathcal{E}$-Unifikatoren berechnet werden, indem man mit einem speziellen $\mathcal{E}$-Unifikationsalgorithmus Unifikatoren berechnet, ohne die Sorten zu berücksichtigen, und anschließend die berechneten Unifikatoren an die entsprechende Sortenstruktur anpasst.

4.3 Unifikation in Congruence Closed und Sort Preserving Gleichungstheorien

In diesem Abschnitt werden zwei wichtige Eigenschaften von Gleichungstheorien eingeführt. Anschließend wird gezeigt, wie diese Eigenschaften verwendet werden können, um $\mathcal{E}$-Unifikatoren in zwei unabhängigen Schritten zu berechnen. Dieser Abschnitt orientiert sich an Schmidt-Schauß (1989a).

Die Identitäten, die eine Gleichungstheorie definieren, werden formuliert mit Funktionssymbolen aus einer Signatur mit Sorten Σ. Dabei sind zunächst keine Bedingungen an die Identitäten gestellt. Insbesondere lassen sich Gleichungen definieren, die nicht kompatibel mit der durch Σ definierten Sortenstruktur sind. Betrachtet man beispielsweise Σ^{let} und $E = \{abs(x,u) \approx b(y,abs(x,u))\}$, sowie die beiden Terme $s = abs(z,r)$, $t = b(y,(abs(z,r))$, dann gilt $s =_{\Sigma^{let},E} t$. Die beiden Terme haben aber unterschiedliche Sorten, d.h. $S_\Sigma(s) \neq S_\Sigma(t)$. Außerdem haben wir beispielsweise $app(s,r) =_{\Sigma^{let},E} app(t,r)$, wobei allerdings $app(t,r)$ kein wohlsortierter Term ist. Um solche Verhältnisse von Sorten zu Identitäten zu charakterisieren definiert man:

Definition 4.3.1. Eine Gleichungstheorie $\mathcal{E} = (\Sigma, E)$ ist

1. *deduction closed*, gdw. $s_1 =_{\mathcal{E}} t_1, \ldots s_n =_{\mathcal{E}} t_n$ und $f(s_1, \ldots, s_n) \in T(\Sigma, X)$ impliziert, dass $f(t_1, \ldots, t_n)$ wohlsortiert ist.
2. *sort preserving*, gdw. für alle $s =_{\mathcal{E}} t$ gilt $S_\Sigma(t) = S_\Sigma(s)$.

Eine Gleichungstheorie ist deduction closed, wenn Subterme eines wohlsortierten Terms t durch gleiche Terme ersetzt werden können, so dass der Term t wohlsortiert bleibt. Eine Theorie ist sort preserving, wenn gleiche Terme die gleichen Sorten besitzen. Ist eine Theorie sort preserving, dann ist sie auch deduction closed. Die für uns interessante Eigenschaft ist hier sort preservation, wie wir weiter unten sehen werden (auf deduction closedness wird hier nicht weiter eingegangen). Gleichungstheorien, die sort preserving sind, haben eine wichtige Eigenschaft, die die Berechnung von $\mathcal{E}$-Unifikatoren vereinfacht: Die Gleichheit von Substitutionen erhält deren Wohlsortiertheit.

Lemma 4.3.2. Sei $\mathcal{E} = (\Sigma, E)$ eine sort preserving Gleichungstheorie. Dann gilt

$$\forall \sigma \in Sub_\Sigma \; \forall \tau \in Sub_{\overline{\Sigma}} : \sigma =_{\mathcal{E}} \tau \Rightarrow \tau \in Sub_\Sigma.$$

Beweis. Sei $\sigma \in Sub_\Sigma$ und $\tau \in Sub_{\overline{\Sigma}}$. Dann gilt $\{x \mapsto \sigma x\} \in Sub_\Sigma$ für alle x. Nach Definition von wohlsortierten Substitutionen gilt $S_\Sigma(\sigma x) \supseteq S_\Sigma(x)$ und nach Voraussetzung ($\sigma =_{\mathcal{E}} \tau$) und weil $\mathcal{E}$ sort preserving ist gilt $S_\Sigma(\tau x) \supseteq S_\Sigma(x)$. Damit haben wir $\{x \mapsto \tau x\} \in Sub_\Sigma$ für alle x und letztlich $\tau \in Sub_\Sigma$. □

Jeder Gleichungstheorie $\mathcal{E} = (\Sigma, E)$ kann eine *unsortierte Gleichungstheorie* $\overline{\mathcal{E}} = (\overline{\Sigma}, E)$ zugeordnet werden, die durch dieselben Identitäten E definiert ist, aber alle Sorten ignoriert. Eine weitere, für die Berechnung von $\mathcal{E}$-Unifikatoren interessante Eigenschaft von Gleichungstheorien, wird charakterisiert durch die Beziehung zwischen einer Theorie $\mathcal{E}$ und der zugehörigen unsortierten Theorie $\overline{\mathcal{E}}$.

Definition 4.3.3. Eine Gleichungstheorie $\mathcal{E} = (\Sigma, E)$ ist *congruence closed*, gdw. für alle $s, t \in T(\Sigma, X)$ gilt $s =_{\mathcal{E}} t \Leftrightarrow s =_{\overline{\mathcal{E}}} t$.

Ist eine Gleichungstheorie $\mathcal{E}$ congruence closed, dann kann zur Berechnung von $\mathcal{E}$-Unifikatoren in zwei unabhängigen, aufeinander folgenden Schritten vorgegangen werden:

- Ignoriere die Sorten und berechne einen $\overline{\mathcal{E}}$-Unifikator für die unsortierte Theorie $\overline{\mathcal{E}}$.
- Ignoriere alle Identitäten und passe die berechneten $\overline{\mathcal{E}}$-Unifikatoren an gegebene Sorten an.

Proposition 4.3.4. Sei $\mathcal{E} = (\Sigma, E)$ eine congruence closed Gleichungstheorie. Dann gilt

$$U_{\overline{\mathcal{E}}}(P) \cap Sub_\Sigma = U_{\mathcal{E}}(P)$$

für alle $\mathcal{E}$-Unifikationsprobleme P.

Beweis. $U_{\overline{\mathcal{E}}}(P) \cap Sub_\Sigma \supseteq U_{\mathcal{E}}(P)$ folgt direkt aus $Sub_\Sigma \subseteq Sub_{\overline{\Sigma}}$.

Es wird $U_{\overline{\mathcal{E}}}(P) \cap Sub_\Sigma \subseteq U_{\mathcal{E}}(P)$ gezeigt. Sei dazu $\sigma \in U_{\overline{\mathcal{E}}}(P) \cap Sub_\Sigma$. Damit gilt $\sigma s_i =_{\overline{\mathcal{E}}} \sigma t_i$ für alle $\mathcal{E}$-Unifikationsprobleme $P = \{s_i =^? t_i\}$ und weil $\mathcal{E}$ congruence closed ist, haben wir $\sigma s_i =_{\overline{\mathcal{E}}} \sigma t_i \Leftrightarrow \sigma s_i =_{\mathcal{E}} \sigma t_i$. D.h. $\sigma \in U_{\mathcal{E}}(P)$. □

Für eine Gleichungstheorie $\mathcal{E} = (\Sigma, E)$, die nicht congruence closed ist, hat man nach dieser Proposition folgende unangenehme Situation: Es gibt Terme s, t für die $s =_{\overline{\mathcal{E}}} t$ und $s \neq_{\mathcal{E}} t$ gilt, woraus folgt $Id \in U_{\overline{\mathcal{E}}}(s =^? t) \cap Sub_\Sigma$ und $Id \notin U_{\mathcal{E}}(s =^? t)$.

Die Anpassung von $\overline{\mathcal{E}}$-Unifikatoren, die berechnet werden, ohne die Sorten zu beachten, an die gegebene Sortenstruktur, geschieht durch sogenannte *Weakenings*.

Definition 4.3.5 (Weakening)**.** Das *Weakening*-Problem zu einer Substitution $\tau \in Sub_{\overline{\Sigma}}$ ist folgendermaßen definiert: Gegeben eine (unsortierte) Substitution $\tau \in Sub_{\overline{\Sigma}}$, finde eine wohlsortierte Substitution $\omega \in Sub_\Sigma$, so dass $\omega\tau \in Sub_\Sigma$. Die wohlsortierte Substitution ω wird als *Weakening* von τ bezeichnet. Die Menge der Lösungen für das Weakening-Problem von τ wird durch $W_\Sigma(\tau)$ bezeichnet. Analog zu Definition 4.1.6 wird eine vollständige (minimale) Menge von Weakenings $CW_\Sigma(\tau)$ (bzw. $MW_\Sigma(\tau)$) zu einer Substitution τ definiert.

Man beachte, dass Weakenings nicht bezüglich Gleichungstheorien definiert sind, sondern nur bezüglich Signaturen. Für reguläre und elementare Signaturen hat die Menge der Weakenings eine besonders einfache Struktur.

Proposition 4.3.6. Sei Σ eine endliche, elementare und reguläre Signatur und $\tau \in Sub_{\overline{\Sigma}}$, so dass eine wohlsortierte Substitution ω existiert mit $\omega\tau \in Sub_\Sigma$. Dann existiert eine minimale, vollständige Menge $MU_\Sigma(\tau)$, die nur aus $\overline{\Sigma}$-Variablenumbenennungen[4] besteht.

Beweis. Sei ω eine Substitution, so dass $\omega\tau \in Sub_\Sigma$ wohlsortiert ist. Sei δ eine Substitution, mit $Dom(\delta) = VRan(\tau)$ und $Ran(\delta)$ bestehe aus neuen Variablen, so dass $LS_\Sigma(\delta x) = LS_\Sigma(\omega x)$. Dann gilt $LS_\Sigma(\delta\tau y) = LS_\Sigma(\omega\tau y)$ für alle $y \in VRan(\tau)$, weil Σ elementar ist[5] und nach Konstruktion von δ. Folglich ist δ ein Weakening von τ, außerdem ist δ nach Konstruktion eine $\overline{\Sigma}$-Variablenumbenennung. Nach Konstruktion gilt außerdem $\delta \lesssim_\Sigma \omega[VRan(\tau)]$, woraus wir folgern, dass $CU_\Sigma(\tau)$ existiert und lediglich aus $\overline{\Sigma}$-Variablenumbenennungen besteht. Die vollständige Menge von Weakenings ist endlich, weil Σ und $VRan(\tau)$ endlich sind. Also kann sie zu einer minimalen vollständigen Menge von Weakenings $MW_\Sigma(\tau)$ gemacht werden, indem jeweils nur ein Repräsentant der Äquivalenzklassen bezüglich $\sim_\Sigma$ in $MW_\Sigma(\tau)$ aufgenommen wird. □

[4] Eine $\overline{\Sigma}$-Variablenumbenennungen ρ, muss nicht sort preserving sein, d.h. $S_\Sigma(\rho x) = S_\Sigma(x)$ gilt i.A. nicht für alle $x \in Dom(\rho)$.

[5] In elementaren Signaturen ist die Sorte eines Terms $f(t_1, \ldots, t_n)$ vollständig bestimmt durch die Sorte von f und die Sorten der t_i.

Um das zentrale Resultat zu beweisen, benötigen wir folgendes technisches Detail: Alle Variablen einer unsortierten Substitution σ in $VRan(\sigma)$ können so gewählt werden, dass sie eine maximale Sorte TOP besitzen, für die $S \sqsubseteq_{\sim} TOP$ für alle $S \in S_\Sigma$ gilt. Besitzt eine Signatur keine solche TOP-Sorte, dann kann sie um eine solche erweitert werden.

Lemma 4.3.7. Sei $\mathcal{E} = (\Sigma, E)$ eine Gleichungstheorie. Dann kann zu der Menge S_Σ der Sortensymbole eine Sorte TOP hinzugefügt werden, mit $S \sqsubseteq_{\sim} TOP$ für alle $S \in S_\Sigma$, so dass sich an der Instantiierungs-Quasiordnung nichts ändert.

Beweis. Die Menge der Terme $T(\Sigma, X)$ ändert sich nicht, es kommen nur neue Variablen der Sorte TOP hinzu. Neue Komponenten in Substitutionen sind $\{x_{TOP} \mapsto t\}$, die wohlsortiert sind und die $\lesssim_{\mathcal{E}}$ auf alten Substitutionen nicht beeinflussen. □

Korollar 4.3.8. Sei $\sigma \in Sub_{\overline{\Sigma}}$. Dann können die Variablen in $VRan(\sigma)$ so gewählt werden, dass $S_\Sigma(x) = TOP$ für alle $x \in VRan(\Sigma)$ gilt.

Wir kommen zum zentralen Satz, der angibt, wie $\mathcal{E}$-Unifikatoren für eine Gleichungstheorie berechnet werden können, die congruence closed und sort preserving ist.

Satz 4.3.9 (Schmidt-Schauß (1989a))**.** Sei $\mathcal{E} = (\Sigma, E)$ eine Gleichungstheorie, die congruence closed und sort preserving ist und sei P ein beliebiges $\mathcal{E}$-Unifikationsproblem mit $W = Var(P)$. Dann gilt

$$\begin{aligned}&\text{für alle } \delta \in U_{\mathcal{E}}(P)\\ &\text{gibt es } \sigma \in \{\omega\tau \mid \tau \in CU_{\overline{\mathcal{E}}}(P), \omega \in MW_\Sigma(\tau)\},\\ &\text{so dass } \sigma \lesssim_{\mathcal{E}} \delta[W].\end{aligned}$$

Beweis. Sei $\delta \in U_{\mathcal{E}}(P)$. Nach Proposition 4.3.4 gilt $U_{\mathcal{E}}(P) = U_{\overline{\mathcal{E}}}(P) \cap Sub_\Sigma$, weil $\mathcal{E}$ congruence closed ist, d.h. $\delta \in U_{\overline{\mathcal{E}}}(P) \cap Sub_\Sigma$. Folglich gibt es für δ eine (unsortierte) Substitution $\tau \in CU_{\overline{\mathcal{E}}}(P)$ und eine Substitution $\theta \in Sub_{\overline{\Sigma}}$, so dass $Dom(\theta) = VRan(\tau)$ und $\theta\tau =_{\overline{\mathcal{E}}} \delta[W]$. Nach Korollar 4.3.8 können alle Variablen in $x \in VRan(\tau)$ so gewählt werden, dass sie die TOP-Sorte besitzen: $S_\Sigma(x) = TOP$. Anwendung der congruence closed Eigenschaft von $\mathcal{E}$ auf $\theta\tau =_{\overline{\mathcal{E}}} \delta[W]$ ergibt $\theta\tau =_{\mathcal{E}} \delta[W]$.

Jetzt ist noch zu zeigen, dass θ durch Instantiierung aus einem wohlsortierten minimalen Weakening von τ gewonnen werden kann. Dazu wird zuerst gezeigt, dass θ wohlsortiert ist. Weil $\mathcal{E}$ sort preserving ist, und $\theta\tau =_{\overline{\mathcal{E}}} \delta[W]$, $\delta \in Sub_\Sigma$ gilt, kann man mit Lemma 4.3.2 folgern, dass $\theta\tau x$ wohlsortiert ist für alle $x \in W$. Insbesondere sind alle Terme in $Ran(\theta)$ wohlsortiert. Es gilt $Dom(\theta) = VRan(\tau)$ (nach Konstruktion von θ) und da alle Variablen in $VRan(\tau)$ so gewählt wurden, dass sie die TOP-Sorte besitzen, gilt $\forall x \in Dom(\theta) : S_\Sigma(x) \supseteq S_\Sigma(\theta x)$, d.h. $\theta \in Sub_\Sigma$.

Halten wir fest, dass θ und δ wohlsortiert sind und $\theta\tau =_{\mathcal{E}} \delta[W]$ gilt, d.h. θ ist ein Weakening von τ (Def. 4.3.5). Die minimale vollständige Menge $MU_\Sigma(\tau)$ existiert[6] und wir wählen ein $\omega \in MU_\Sigma(\tau)$, so dass $\omega \lesssim_\Sigma \theta$, d.h. $\exists\eta \in Sub_\Sigma : \eta\omega = \theta[W]$. Einsetzen von $\eta\omega$ für θ ergibt $\eta\omega\tau = \delta[W]$ und folglich $\omega\tau \lesssim_{\mathcal{E}} \delta[W]$. □

Nach diesem Satz ist $\{\omega\tau \mid \tau \in CU_{\overline{\mathcal{E}}}(P), \omega \in MW_\Sigma(\tau)\}$ eine vollständige Menge von Unifikatoren für ein $\mathcal{E}$-Unifikationsproblem, wenn $\mathcal{E}$ sort preserving und congruence closed ist. Der entscheidende Punkt hierbei ist, dass $CU_{\overline{\mathcal{E}}}(P)$ berechnet werden kann, ohne die Sorten von $\mathcal{E}$ zu berücksichtigen, und $MW_\Sigma(\tau)$ berechnet werden kann, ohne die definierenden Identitäten von $\mathcal{E}$ zu berücksichtigen.

4.3.1 Eigenschaften von $\mathcal{E}^{let}_{AC}$ und $\mathcal{E}^{let}_{Cl}$

Wir untersuchen die beiden Theorien $\mathcal{E}^{let}_{AC}$ und $\mathcal{E}^{let}_{Cl}$ auf congruence closedness und sort preservation.

Bemerkung 4.3.10. Die beiden Theorien $\mathcal{E}^{let}_{AC}$ und $\mathcal{E}^{let}_{Cl}$ verfügen über endliche und einfache Signaturen. Aus der Einfachheit folgt, dass die Signaturen auch elementar und regulär sind (vgl. Abschnitt 3.3.2).

Im Allgemeinen ist es unentscheidbar, ob eine Gleichungstheorie congruence closed ist, was man durch Reduktion auf das Wortproblem[7] zeigen kann (Schmidt-Schauß, 1989a). Folgendes Lemma gibt ein einfaches Kriterium, mit dem sich für endliche, reguläre und elementare Signaturen prüfen lässt, ob eine Gleichungstheorie congruence closed ist. Die Beweise der beiden Lemmata, mit denen sich für bestimmte (endliche, reguläre und elementare) Gleichungstheorien feststellen lässt, ob sie congruence closed und sort preserving sind, sind hier ausgespart. Sie sind in Schmidt-Schauß (1989a) zu finden.

Lemma 4.3.11. Sei Σ eine endliche, elementare und reguläre Signatur. Wenn für alle Identitäten $s \approx t \in E$ und für alle $\overline{\Sigma}$-Variablenumbenennungen ρ gilt $\rho s \in T(\Sigma, X)$ impliziert $\rho \in Sub_\Sigma$, dann ist $\mathcal{E}$ congruence closed.

Proposition 4.3.12. Die Theorien $\mathcal{E}^{let}_{AC}$ und $\mathcal{E}^{let}_{Cl}$ sind congruence closed.

Beweis. Wir zeigen congruence closedness für $\mathcal{E}^{let}_{Cl}$. Sei ρ eine $\overline{\Sigma}$-Variablenumbenennung, so dass $\rho(\{\!|c_B|\{\!|d_B|e_U|\!\}|\!\}) \in T(\Sigma^{let}, X)$. Jetzt ist zu zeigen, dass $LS_{\Sigma^{let}}(x) \sqsubseteq LS_{\Sigma^{let}}(\rho x)$ für alle $x \in Dom(\rho)$ gilt, was offensichtlich der Fall ist, weil die Wohlsortiertheit von $\rho(\{\!|c_B|\{\!|d_B|e_U|\!\}|\!\})$ impliziert $\rho(c), \rho(d) : B$ und $\rho(e) : R \sqsubseteq U$.

[6] Für reguläre elementare Signaturen nach Proposition 4.3.6, im allgemeinen Fall siehe Schmidt-Schauß (1989a), S. 96.

[7] Gegeben eine Gleichungstheorie $\mathcal{E}$ und Terme s, t. Gilt $s =_{\mathcal{E}} t$?

Der Beweis geht analog für $\mathcal{E}_{AC}^{let}$. □

Die Eigenschaft sort preservation ist ebenfalls unentscheidbar für den allgemeinen Fall, allerdings ist die Eigenschaft entscheidbar, wenn die Signatur endlich, elementar und regulär ist.

Lemma 4.3.13. Sei Σ eine endliche und elementare Signatur.

1. Für alle $\sigma \in Sub_\Sigma$ und für alle $s \approx t \in E$ gilt

$$S_\Sigma(\sigma s) = S_\Sigma(\sigma t) \;\Leftrightarrow\; =_E \text{ ist sort preserving.}$$

2. Sei Σ außerdem regulär. Wenn für alle wohlsortierten $\overline{\Sigma}$-Variablenumbenennungen ρ und für alle $s \approx t \in E$ gilt $LS_\Sigma(\rho s) = LS_\Sigma(\rho t)$, dann folgt $\forall \sigma \in Sub_\Sigma \;\; \forall s \approx t \in E : LS_\Sigma(\sigma s) = LS_\Sigma(\sigma t)$.

Proposition 4.3.14. Die Theorien $\mathcal{E}_{AC}^{let}$ und $\mathcal{E}_{Cl}^{let}$ sind sort preserving.

Beweis. Es wird sort preservation für $\mathcal{E}_{Cl}^{let}$ gezeigt. Dazu muss für alle wohlsortierten $\overline{\Sigma}$-Variablenumbenennungen ρ geprüft werden, ob $LS_\Sigma(\rho(\{\!|c_B|\{\!|d_B|e_U|\!\}\!\})) = LS_\Sigma(\rho(\{\!|d_B|\{\!|c_B|e_U|\!\}\!\}))$ gilt. Als wohlsortierte $\overline{\Sigma}$-Variablenumbenennungen kommen in Frage: $\rho = \{c \mapsto c', d \mapsto d', e \mapsto e'\}$ mit $c', d' : B$, $e' : R \sqsubseteq U$, wofür die Bedingung erfüllt ist. Sort preservation für $\mathcal{E}_{Cl}^{let}$ folgt dann aus der ersten Aussage von Lemma 4.3.13.

Der Beweis für die Theorie $\mathcal{E}_{AC}^{let}$ geht analog. □

Beide Theorien sind sowohl congruence closed als auch sort preserving. Damit können in beiden Theorien Unifikatoren auf die beschriebene Art in zwei unabhängigen Schritten berechnet werden. Alle Sorten können zunächst ignoriert werden. Zur Unifikation von Termen mit einem assoziativen und kommutativen, bzw. einem links-kommutativen Funktionssymbol kann ein Unifikationsalgorithmus verwendet werden, der Terme unter Berücksichtigung der entsprechenden Axiome unifiziert. Wie wir bereits am Ende von Abschnitt 4.2 bemerkt haben, existierten für *AC*- und *Cl*-Funktionssymbole eine Reihe von Unifikationsalgorithmen, auf die zurückgegriffen werden kann. Betrachten wir zunächst die generelle Vorgehensweise zur Unifikation von *AC*-Funktionssymbolen.

Aufgrund der hohen Bedeutung von *AC*-Funktionssymbolen existiert eine Vielzahl von Arbeiten, die sich mit deren Unifikation beschäftigen (für eine Übersicht siehe die bereits zitierten Arbeiten, sowie Baader und Snyder (2001)). *AC*-Unifikation verwendet (semantische) Eigenschaften der durch die *AC*-Theorie definierten freien Algebren, um Unifikationsprobleme in Gleichungen über bestimmte algebraische Strukturen (Diophantische Gleichungen) zu übersetzen und dann mit Hilfe bekannter mathematischer Resultate zu lösen. Dabei wurden historisch zunächst nur *AC*-Unifikationsprobleme zwischen Termen betrachtet, die lediglich *AC*-Funktionssymbole, Konstanten und Variablen enthielten (Stickel (1975), Livesey und Siekmann

(1976)). Um Unifikationsprobleme mit zusätzlichen freien Funktionssymbolen (das sind Funktionssymbole, die nicht assoziativ und kommutativ sind) zu lösen, benötigt man zusätzlich die Theorie zur Kombination von Unifikationsalgorithmen (Schmidt-Schauß (1989b), Baader und Schulz (1996)). Da letrec-Umgebungen i.A. freie Funktionssymbole ($\{\cdot\}$ und $b(\cdot,\cdot)$) enthalten, muss die Kombination von Unifikationsalgorithmen für allgemeine Unifikation von AC-Funktionssymbolen berücksichtigt werden.

Unifikationsalgorithmen für Cl-Funktionssymbole folgen einem eher syntaktischen Ansatz. Unifikationsprobleme werden schrittweise in gelöste Form transformiert. Die *Decomposition*-Regel wird für das Cl-Funktionssymbol durch eine Regel ersetzt, die Vertauschbarkeit von Argumenten berücksichtigt. In der Literatur ist dieser Ansatz unter dem Stichwort "'Unifikation von (Multi-) Mengen"' zu finden (Dovier, Policriti, & Rossi, 1998; Dantsin & Voronkov, 1999; Dovier et al., 2006). Wie wir gesehen haben (Bemerkung 4.1.3), stellt die Cl-Theorie eine Einschränkung der AC-Theorie dar, da die Anzahl der Variablen der Sorte U in Cl-Umgebungstermen auf eins beschränkt ist, im Gegensatz zu AC-Umgebungstermen, die beliebig viele Variablen der Sorte U enthalten können (zum Verhältnis von AC-Theorien zu Cl-Theorien siehe auch Dovier, Pontelli, und Rossi (1998)). Allerdings ist die Cl-Theorie ausreichend, um Vertauschbarkeit von Elementen in letrec-Umgebungen zu axiomatisieren (Bemerkung 4.1.3). Sie ermöglicht einen Unifikationsalgorithmus, der sich im Rahmen der bisher behandelten (syntaktischen) Begriffe bewegt, ohne zusätzlich auf Methoden zurückzugreifen, die semantische Eigenschaften der Gleichungstheorien verwenden. Ein Vorteil dieses syntaktischen Ansatzes ist, dass er direkt mit den in Kapitel 3 definierten Unifikationsregeln für reguläre Signaturen verwendet werden kann. Ein Vorgehen in zwei Schritten wie oben beschrieben, ist nicht notwendig.[8] Aus diesem Grund, und da sich die beiden Theorien bezüglich der Berechenbarkeitseigenschaften nicht unterscheiden, fällt die Wahl hier auf die $\mathcal{E}^{let}_{Cl}$-Theorie anstelle der $\mathcal{E}^{let}_{AC}$ Theorie, um die Vertauschbarkeit von Elementen in letrec-Umgebung zu repräsentieren.

Nachdem die Entscheidung auf die $\mathcal{E}^{let}_{Cl}$-Theorie gefallen ist, betrachten wir noch einmal genau, wie letrec-Umgebungen in $\Sigma^{let}_{\{\cdot|\cdot\}}$-Terme übersetzt werden.

Definition 4.3.15. Die in Abschnitt 3.3 definierte Abbildung $[\![\ ]\!] : \Lambda^{let} \to T(\Sigma^{let}, X)$, wird folgendermaßen zu einer Abbildung $[\![\ ]\!] : \Lambda^{let} \to T(\Sigma^{let}_{\{\cdot|\cdot\}}, X)$ erweitert, um letrec-Umgebungen zu übersetzen.

$$
\begin{aligned}
[\![(\texttt{letrec}\ \{Env\}\ \texttt{in}\ s)]\!] &= letrec([\![Env]\!], [\![s]\!]) \\
[\![\{Env, x_1 = s_1, \ldots, x_n = s_n\}]\!] &= [\![\{x_1 = s_1, \ldots, x_n = s_n, Env\}]\!] \\
[\![\{x_1 = s_1, \ldots, Env, \ldots, x_n = s_n\}]\!] &= [\![\{x_1 = s_1, \ldots, x_n = s_n, Env\}]\!] \\
[\![\{x_1 = s_1, x_2 = s_2, \ldots, x_n = s_n\}]\!] &= \{|[\![x_1 = s_1]\!] | [\![\{x_2 = s_2, \ldots, x_n = s_n\}]\!]|\}
\end{aligned}
$$

[8] Schon aus dem Grunde nicht, weil die Signatur Σ^{let} über einen eindeutigen Unifikationstyp verfügt.

$$\begin{aligned}[[\{x = s, Env\}]] &= \{\!|[[x = s]]|[[Env]]|\!\} \\ [[\{x = s\}]] &= \{\!|[[x = s]]|\emptyset|\!\} \\ [[x = s]] &= bind([[x]], [[s]])\end{aligned}$$

Aufgrund der Sorte $B \to U \to U$ des Funktionssymbols $\{\!|\cdot|\cdot|\!\}$ muss die Umgebungsvariable an die letzte Position in der Umgebung getauscht werden. Ist in einer Umgebung keine Umgebungsvariable enthalten, muss das zusätzliche Konstantensymbol $\emptyset$ an letzter Position durch die Übersetzung eingefügt werden. So wird sichergestellt, dass bei der Übersetzung keine Umgebungen der Form $\{\!|b_1|b_2|\!\}$ mit b_1, b_2 Terme der Sorte B entstehen. Umgebungsterme dieser Form werden vermieden, weil für Unifikationsprobleme der Gestalt $\{\!|b_1|b_2|\!\} =^? \{\!|b_3|b_4|\!\}$ das *Cl*-Axiom nicht anwendbar ist, und deshalb nicht alle möglichen Unifikatoren berechnet werden. Für das Problem $\{\!|b_1|\{\!|b_2|\emptyset|\!\}|\!\} =^? \{\!|b_3|\{\!|b_4|\emptyset|\!\}|\!\}$ ist das *Cl*-Axiom anwendbar und alle Unifikatoren können wie erwartet berechnet werden.

Für einen $\Sigma^{let}_{\{\!|\cdot|\cdot|\!\}}$-Umgebungsterm wird verkürzt $\{\!|s_1, \ldots, s_m|S|\!\}$ geschrieben, anstatt $\{\!|s_1|\{\!| \ldots \{\!|s_m|S|\!\} \ldots |\!\}|\!\}$. Ist $S = \emptyset$, schreiben wir $\{\!|s_1, \ldots, s_m|\!\}$. Im weiteren Verlauf bezeichnen X, Y, N Variablen der Sorte U, die auch Umgebungsvariablen genannt werden.

4.4 Unifikation in $\mathcal{E}_{Cl}^{let}$

In diesem Abschnitt wird ein Algorithmus vorgestellt, um Unifikatoren für $\mathcal{E}_{Cl}^{let}$-Unifikationsprobleme zu berechnen. Da das links-kommutative Funktionssymbol $\{\!|\cdot|\cdot|\!\}$ als Multimengenkonstruktor verstanden werden kann, orientieren wir uns an Arbeiten zur Unifikation von Multimengen. Der Algorithmus wird bezüglich der Theorie $\mathcal{E}_{Cl}^{let}$ erklärt. Er kann aber zur Unifikation von beliebigen Σ-Termen verwendet werden, wenn

- Σ eine reguläre und elementare Signatur ist und
- in der Signatur ein *Cl*-Funktionssymbol enthalten ist, das eine vergleichbare Sorte wie $\{\!|\cdot|\cdot|\!\}$ besitzt.

Ein Unifikator für ein gegebenes Unifikationsproblem wird durch Transformation des Problems in gelöste Form berechnet. Dazu werden die Unifikationsregeln des vorhergehenden Kapitels verwendet, wobei die *Decomposition* Regel für das *Cl*-Funktionssymbol $\{\!|\cdot|\cdot|\!\}$ ersetzt wird durch eine spezielle Unifikationsregel, die Vertauschbarkeit von Argumenten berücksichtigt, da die Transformation von $\{\!|s_1, \ldots, s_m|S|\!\} =^? \{\!|t_1, \ldots, t_n|T|\!\}$ mit *Decomposition* i.A. in Nicht-Unifizierbarkeit resultiert und nicht alle möglichen Unifikatoren gefunden werden.

Definition 4.4.1. Sei P ein $\mathcal{E}_{Cl}^{let}$-Unifikationsproblem. Die Unifikationsregeln für

reguläre Signaturen (Definition 3.4.13) werden folgenderma"sen erweitert und modifiziert (nach Dovier, Policriti, & Rossi, 1998):

Decomposition
$\{f(s_1, \ldots s_n) =^? f(t_1, \ldots t_n)\} \uplus P \Rightarrow \{s_1 =^? t_1, \ldots s_n =^? t_n\} \cup P$
wenn $f \in \Sigma^n - \{\{\!|\cdot|\cdot|\!\}\}$.

U − Decomposition
$\{\{\!|t|s|\!\} =^? \{\!|t'|s'|\!\}\} \uplus P \Rightarrow$ *i)* $\{t =^? t', s =^? s'\} \cup P$
ii) $\{s =^? \{\!|t'|N|\!\}, \{\!|t|N|\!\} =^? s'\} \cup P$
wobei N eine neue Variable der Sorte U ist.

Die Regel *U-Decomposition* wird im weiteren Verlauf abgekürzt bezeichnet als *U-Dec*. Zur Unifikation von Gleichungen der Art $\{\!|s_1, \ldots, s_m|S|\!\} =^? \{\!|t_1, \ldots, t_n|T|\!\}$ wird nichtdeterministisch eine der Transformationsmöglichkeiten *i)* oder *ii)* der *U-Dec*-Regel gewählt.

Für Unifikationsprobleme zwischen zwei Umgebungen, die beide dieselbe Umgebungsvariable X enthalten, können die oben definierten Unifikationsregeln zu einer nicht terminierenden Folge von Transformationen führen. Beispielsweise

$$\begin{aligned} & \{\!|t|X|\!\} =^? \{\!|u|X|\!\} \\ \xrightarrow{\textit{U-Dec ii)}} & \{X =^? \{\!|u|N|\!\}, \{\!|t|N|\!\} =^? X\} \\ \xrightarrow{\textit{Elim}} & \{X =^? \{\!|u|N|\!\}, \{\!|t|N|\!\} =^? \{\!|u|N|\!\}\}. \end{aligned}$$

Das letzte Unifikationsproblem ist mindestens so kompliziert wie das erste. Diesem Problem kann man begegnen, indem man sich überlegt, dass eine Gleichung zwischen Umgebungen, in der dieselbe Umgebungsvariable in beiden Umgebungen vorkommt, nur dann eine Lösung besitzt (unter *Cl*), wenn beide Umgebungen genau die gleiche Anzahl von Elementen besitzen. Die Regel *U-Dec* wird dann aufgespalten in die Fälle:

1. Die Umgebungsvariablen zweier zu unifizierender Umgebungen sind gleich, dann ersetze beide Variablen durch die Konstante $\emptyset$ und unifiziere die resultierenden Umgebungen. Oder
2. die zu unifizierenden Umgebungen besitzen keine gemeinsamen Umgebungsvariablen, dann unifiziere die Umgebungen wie unter *U-Dec* angegeben.

Um die entsprechenden modifizierten Unifikationsregeln zu definieren, benötigen wir eine Methode, die Umgebungsvariablen zweier Umgebungen zu vergleichen. Wir definieren

$$\begin{aligned} \mathit{tail}(s) &= s, \text{ wenn } s \text{ ein Term mit Wurzelsymbol ungleich } \{\!|\cdot|\cdot|\!\} \text{ ist und} \\ \mathit{tail}(\{\!|t|s|\!\}) &= \mathit{tail}(s). \end{aligned}$$

Definition 4.4.2. Die modifizierten Unifikationsregeln, die Nichtterminierung ausschließen für den Fall, dass zwei zu unifizierende Umgebungen dieselbe Variable enthalten, sind folgendermaßen definiert.

U – Decomposition

$\{\{\!|t|s|\!\} =^? \{\!|t'|s'|\!\}\} \uplus P \Rightarrow$ *i)* $\{t =^? t', s =^? s'\} \cup P$

ii) $\{s =^? \{\!|t'|N|\!\}, \{\!|t|N|\!\} =^? s'\} \cup P$

wenn $tail(s)$ und $tail(s')$ nicht dieselbe Variable X der Sorte U bezeichnen.

N ist eine neue Variable der Sorte U.

U – Decomposition*

$\{\{\!|t|s|\!\} =^? \{\!|t'|s'|\!\}\} \uplus P \Rightarrow \{\{\!|t|\widehat{s}|\!\} =^? \{\!|u|\widehat{s'}|\!\}\} \cup P$

wenn $tail(s) = tail(s') = X : U$.

$\widehat{s}, \widehat{s'}$ sind gleich s, s' mit $tail(s), tail(s')$ durch $\emptyset$ ersetzt.

U – Fail

$\{\{\!|t|s|\!\} =^? \{\!|t'|s'|\!\}\} \uplus P \Rightarrow \bot$

wenn $tail(s) = tail(s') = \emptyset$ und

s, s' besitzen nicht die gleiche Anzahl von Elementen.

4.4.1 Terminierung

Terminierung für den Unifikationsalgorithmus mit der *U-Dec* Regel zu zeigen, wird erschwert durch die Einführung neuer Variablen bei der Anwendung von *U-Dec ii*). Die Variablen werden so eingeführt, dass sie nicht direkt eliminiert werden können, so wie es beim Beweis der Terminierung ohne *U-Dec* Transformation für die Regeln *Subsort* und *Common Subsort* der Fall war (siehe Lemma 3.4.17). Ziel dieses Abschnittes ist es zu zeigen, dass der Algorithmus mit der zusätzlichen Regel trotzdem für alle $\mathcal{E}_{Cl}^{let}$-Unifikationsprobleme terminiert. Die Überlegung dazu sieht folgendermaßen aus: Jedem $\mathcal{E}_{Cl}^{let}$-Unifikationsproblem wird ein Komplexitätsmaß zugeordnet. Bei der Unifikation von zwei Umgebungstermen vergrößert eine Anwendung von *U-Dec ii*) dieses Maß zunächst. Allerdings kann die Gleichung zwischen den Umgebungstermen durch eine Folge von Transformationen komplett zerlegt werden, so dass sich das Komplexitätsmaß am Ende dieser Zerlegung verkleinert hat. Komplette Zerlegung von Gleichungen zwischen Umgebungen definieren wir folgendermaßen:

Definition 4.4.3 (Semi-gelöste Form)**.** Ein Unifikationsproblem

$$P = \{X_1 =^? u_1, \ldots X_m =^? u_m, s_1 =^? t_1, \ldots s_n =^? t_n\}$$

ist in *semi-gelöster Form*, gdw.

- alle X_i paarweise verschiedene Variablen, der Sorte U sind, die nicht in u_i und t_j vorkommen. Und
- für alle Gleichungen $s_i =^? t_i$ gilt, dass die Wurzelsymbole von s_i, t_i ungleich $\{\!|\cdot|\cdot|\!\}$ sind.

Wenn ein Unifikationsproblem zwischen Umgebungen lösbar ist, dann kann es in eine semi-gelöste Form transformiert werden. Bei dieser Transformation werden i.A. neue Variablen der Sorte U eingeführt, die aber fast alle (d.h. entweder alle oder alle bis auf eine) in gelöste Form[9] gebracht werden können, wie wir im Folgenden sehen werden.

Lemma 4.4.4. Sei $P = \{\{\!|s_1, \ldots, s_m|S|\!\} =^? \{\!|t_1, \ldots t_n|T|\!\}\}$ ein $\mathcal{E}_{Cl}^{let}$-Unifikationsproblem mit $U_{\mathcal{E}_{Cl}^{let}}(P) \neq \emptyset$. S und T sind Terme deren Wurzelsymbol ungleich $\{\!|\cdot|\cdot|\!\}$ ist.

Dann kann P durch eine endliche Folge von Transformationen in eine semi-gelöste Form S gebracht werden und für alle solche Transformationsfolgen gilt, dass S von folgender Gestalt ist:

$$\{s_{i_1} =^? t_{j_1}, \ldots, s_{i_k} =^? t_{j_k}\} \cup G \cup H$$

mit $i_1, \ldots, i_k \in \{0, \ldots, m\}$, $j_1, \ldots, j_k \in \{0, \ldots, n\}$. Die Menge H ist leer oder enthält endlich viele Gleichungen $N_i =^? u_i$ in gelöster Form. G ist eine endliche Menge von Unifikationsgleichungen, die von der Form von S und T abhängig sind:

1. S und T sind keine Variablen der Sorte U, dann ist $G = \{S =^? T\}$ und es muss $m = n$ gelten, da sonst P keine Lösung besitzt.
2. S und T sind verschiedene Variablen der Sorte U, dann ist $G = \{S =^? \{\!|t_{j_{k+1}}, \ldots, t_{j_n}|N|\!\}, T =^? \{\!|s_{i_{k+1}}, \ldots, s_{i_m}|N|\!\}\}$ mit $m, n \geq k$ und N ist eine neue Variable der Sorte U.
3. $S : U$ ist Variable und $T : U$ ist Variable $(T \neq S)$[10] oder $T : U$ ist ein Term. Dann ist $G = \{S =^? \{\!|t_{j_{k+1}}, \ldots, t_{j_n}|T|\!\}\}$ mit $n \geq k$. Falls $m = n$ gilt, ist auch noch der Fall $G = \{S =^? T\}$ möglich. Oder
4. $T : U$ ist Variable und $S : U$ ist Variable $(S \neq T)$ oder $S : U$ ist ein Term. Dann ist $G = \{T =^? \{\!|s_{i_{k+1}}, \ldots, s_{i_m}|S|\!\}\}$ mit $m \geq k$. Falls $m = n$ gilt ist auch noch der Fall $G = \{T =^? S\}$ möglich.

Beweis.

[9] Ein Variable x ist im Unifikationsproblem P in *gelöster Form*, gdw. x in P genau einmal in einer Gleichung $x = t$ vorkommt und $x \notin Var(t)$.

[10] Für den Fall $T = S$ werden die beiden Variablen durch die Konstante $\emptyset$ ersetzt (*U-Dec*$*$). Wir befinden uns dann in Fall 1.

Durch Induktion über die Anzahl der Variablen m und n mit Fallunterscheidung über die Terme S und T. Dabei müssen alle möglichen Transformationen ausgehend von P betrachtet werden, die P in semi-gelöste Form bringen.

Wir gehen der Einfachheit halber davon aus, dass S nicht in den s_i und T nicht in den t_i vorkommt.[11]

$m = 1, n = 1$**:** $P = \{\{|s_1|S|\} =^? \{|t_1|T|\}\}$ kann transformiert werden zu

$$\begin{array}{ll} a) & \textit{U-Dec i)} \Rightarrow \{s_1 =^? t_1, S =^? T\} \qquad \text{oder} \\ \hline b) & \textit{U-Dec ii)} \Rightarrow \{S =^? \{|t_1|N|\}, \{|s_1|N|\} =^? T\}. \end{array}$$

Sind beide Terme S und T keine Variablen der Sorte U, dann ist $a)$ in semi-gelöster Form (Fall 1) und $b)$ besitzt keine Lösung. Der gleiche Fall gilt, falls einer der beiden Terme keine Variable der Sorte U ist (nach eventueller Orientierung). Sind S, T beides Variablen der Sorte U, dann sind die Probleme $a)$ (Fall 3) und $b)$ (Fall 2) nach einem *Orient*-Schritt für T in semi-gelöster Form.

Um zu sehen, wie die Fälle 3 und 4 des Lemmas zustande kommen und wie die Gleichungen in H aussehen, betrachten wir den Fall:

$m = 1, n = 2$**:** $P = \{\{|s_1|S|\} =^? \{|t_1, t_2|T|\}\}$ kann transformiert werden zu

$$\begin{array}{ll} a) & \textit{U-Dec i)} \Rightarrow \{s_1 =^? t_1, S =^? \{|t_2|T|\}\} \qquad \text{oder} \\ \hline b) & \textit{U-Dec ii)} \Rightarrow \{S =^? \{|t_1|N|\}, \{|s_1|N|\} =^? \{|t_2|T|\}\}. \end{array}$$

Problem $a)$ ist in semi-gelöster Form, (Fall 3) wenn S eine Variable mit Sorte U ist. Problem $b)$ muss noch weiter transformiert werden. Folgende Möglichkeiten bestehen:

$$\begin{array}{ll} c) & \textit{U-Dec i)} \Rightarrow \{S =^? \{|t_1|N|\}, s_1 =^? t_2, N =^? T\} \qquad \text{oder} \\ \hline d) & \textit{U-Dec ii)} \Rightarrow \{S =^? \{|t_1|N|\}, N =^? \{|t_2|N_2|\}, \{|s_1|N_2|\} =^? T\}. \end{array}$$

Falls $S : U$ eine Variable ist, kann Problem $c)$ durch *Variable Elimination* von N in semi-gelöste Form (Fall 3) gebracht werden. Die Gleichung $N =^? T$ ist in der Menge H. Das Unifikationsproblem $d)$ kann ebenfalls in semi-gelöste Form gebracht werden, falls $S, T : U$ Variablen sind und *Variable Elimination* auf N sowie *Orient* auf T angewendet wird: $\{S =^? \{|t_1, t_2|N_2|\}, N =^? \{|t_2|N_2|\}, T =^? \{|s_1|N_2|\}\}$ (Fall 2). Die Gleichung $N =^? \{|t_2|N_2|\}$ befindet sich in H.

Ist S keine Variable der Sorte U, oder S und T sind beides keine Variablen der Sorte U, dann besitzt P keine Lösung.

Für $m = 2, n = 1$ haben wir $T =^? \{|s_i|S|\}$ (Fall 4) anstelle von $S =^? \{|t_i|S|\}$,

[11] Diese Annahme gilt, weil die Unifikationsprobleme, die zur Berechnung von Überlappungen gelöst werden müssen eingeschränkt sind. Siehe dazu Definition 6.3.4 im Kapitel 6 über die Unifikation von Termen mit Kontextvariablen.

ansonsten ist der Fall analog.

$m = 1, n \to n+1$, (Induktionsschritt für n).

Transformiere $P = \{\{|s_1|S|\} =^? \{|t_1, \ldots, t_{n+1}|T|\}\}$ zu

$$\frac{a)\ \ \textit{U-Dec i)} \Rightarrow \{s_1 =^? t_1, S =^? \{|t_2, \ldots, t_{n+1}|T|\}\} \qquad \text{oder}}{b)\ \ \textit{U-Dec ii)} \Rightarrow \{S =^? \{|t_1|N|\}, \{|s_1|N|\} =^? \{|t_2, \ldots, t_{n+1}|T|\}\}.}$$

Wenn $S : U$ eine Variable ist, dann ist $a)$ in semi-gelöster Form (Fall 3), sonst besitzt P keine Lösung. Die erste Gleichung von Problem $b)$ ist in semi-gelöster Form. Auf die zweite Gleichung ist die Induktionsannahme anwendbar, da die Umgebung $\{|t_2, \ldots t_{n+1}|N|\}$ nur noch n Elemente besitzt.

Der Induktionsschritt $m \to m+1$ verläuft analog. Folglich enden alle Transformationsfolgen, die ein Unifikationsproblem zwischen Umgebungen in semi-gelöste Form bringen, mit einem Problem, das einer der im Lemma angegebenen Formen entspricht. □

Bemerkung 4.4.5. Eine Folge von Transformationen, die ein Unifikationsproblem in semi-gelöste Form transformiert, besteht lediglich aus *U-Dec*, *U-Dec∗*, *Orientation* und *Variable Elimination* Transformationen. Insbesondere werden keine anderen Transformationen angewendet, die neue Variablen einführen (etwa *Common Subsort* oder *Weakening*).

Wir halten eine wichtige Tatsache, die zum Beweis der Terminierung notwendig ist, fest: Entweder ist die Menge H aus obigem Lemma (4.4.4) leer (wenn alle Elemente zweier zu unifizierender Umgebungen untereinander unifiziert werden), oder alle Gleichungen der Menge H haben eine der folgenden Formen, wobei $N_i, N : U$ jeweils paarweise verschiedene, neu eingeführte Variablen sind:

- $N_i =^? \{|u_1, \ldots, u_l|\}$ mit $u_i \in \{s_1, \ldots, s_m, t_1, \ldots, t_n\}$ (Fall 1),
- $N_i =^? \{|u_1, \ldots u_l|N|\}$ mit $u_i \in \{s_1, \ldots, s_m, t_1, \ldots, t_n\}$ (Fall 2),
- $N_i =^? \{|t_{j_{k+1}}, \ldots t_{j_n}|T|\}$ (Fall 3),
- $N_i =^? \{|s_{i_{k+1}}, \ldots s_{i_m}|S|\}$ (Fall 4),
- $N_i =^? X$ mit X gleich S oder T (Fälle 2, 3, 4) oder gleich N (Fall 2) oder gleich $\emptyset$ (Fall 1).

D.h. nach einer Folge von Transformationen eines Unifikationsproblems in semi-gelöste Form, sind alle neuen Variablen bis auf N in gelöster Form, unabhängig davon, wie viele Variablen neu eingeführt wurden.

Korollar 4.4.6. Wird ein Unifikationsproblem durch eine beliebige Folge von Transformationen in semi-gelöste Form gebracht, dann enthält die semi-gelöste Form maximal eine neue Variable, die nicht in gelöster Form ist (Fall 2 aus Lemma 4.4.4, in den Fällen 1, 3 und 4 werden keine neuen, nicht gelösten Variablen eingeführt).

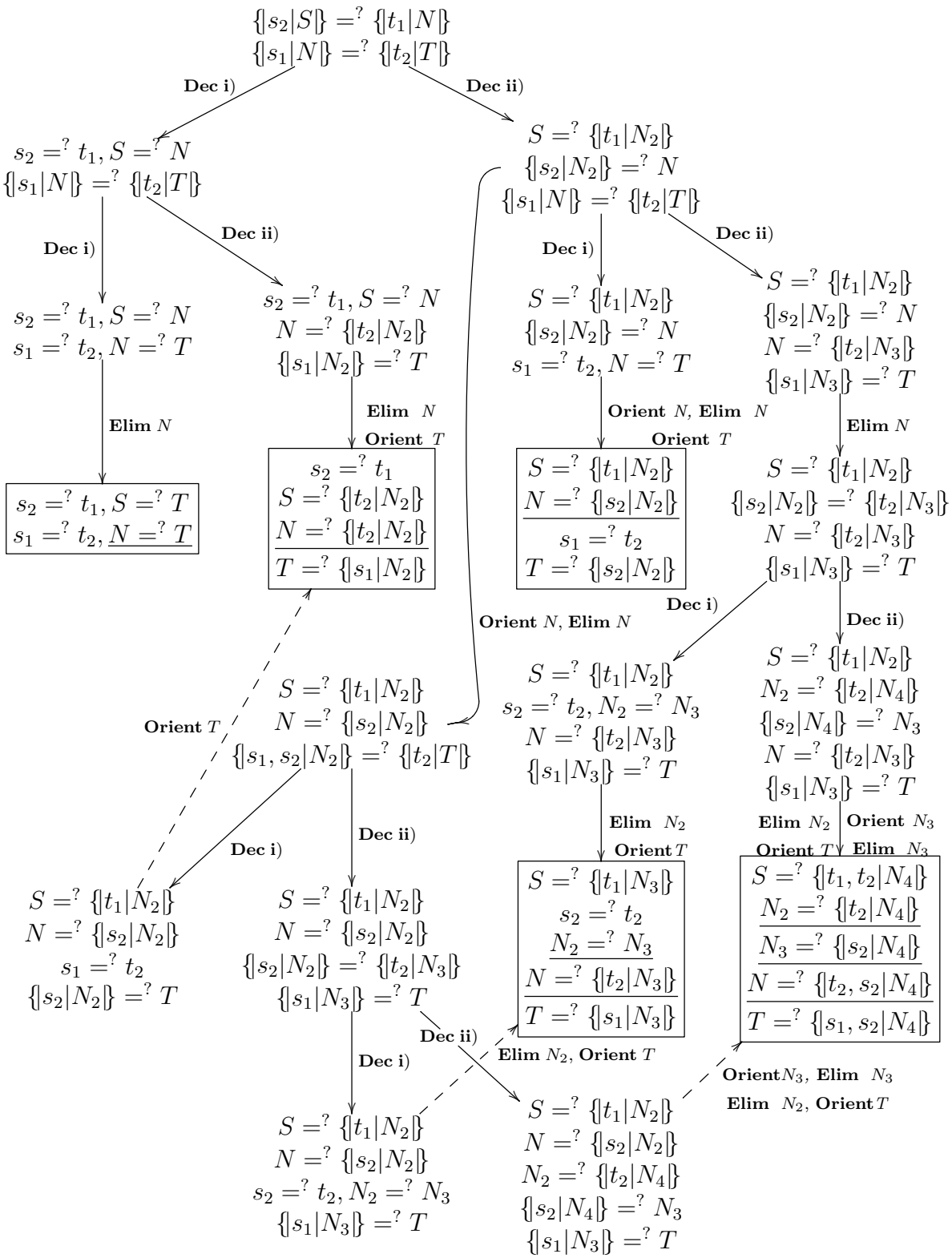


Figure 4.1. Alle möglichen Transformationen von $\{\!|s_1, s_2|S|\!\} =^? \{\!|t_1, t_2|T|\!\}$ nach einem *U-Dec ii*)-Schritt.

Betrachte zur Veranschaulichung der Gleichungen in der Menge H auch

Beispiel 4.4.12 und Abbildung 4.1. Dort sind alle Transformationsmöglichkeiten eines Unifikationsproblems $\{\!|s_1, s_2|S|\!\} =^? \{\!|t_1, t_2|T|\!\}$, für S, T Variablen der Sorte U, nach einem *U-Dec ii*)-Schritt zu sehen (das entspricht dem Fall $m = n = 2$ aus Lemma 4.4.4). Wir gehen der Einfachheit halber davon aus, dass S nicht in den s_i und T nicht in den t_i vorkommt. Die Gleichungen, die sich in H befinden, sind in der Abbildung unterstrichen.

Um die Terminierung des Unifikationsalgorithmus mit der *U-Dec*-Regel zu zeigen, wird der Algorithmus *Unify* aus Abschnitt 3.4.1 folgendermaßen modifiziert: Wende auf ein Unifikationsproblem P eine beliebige Transformationsregel aus Definition 4.4.2 an. Ist die angewendete Unifikationsregel *U-Dec*, die eine Gleichung $e \in P$ zwischen Umgebungen transformiert, dann bringe das Teilproblem e in semi-gelöste Form. An dieser Stelle wird der Nichtdeterminismus des ursprünglichen Algorithmus eingeschränkt, indem alle Transformationen, die e in semi-gelöste Form bringen, vorgezogen werden. Der Grund für dieses Vorgehen liegt darin, dass sich ein Unifikationsproblemen zugeordnetes Komplexitätsmaß nach dieser Folge von Transformationen verringert (was im Beweis von Satz 4.4.7 gezeigt wird). Der modifizierte Unifikationsalgorithmus ist in Abbildung 4.2 zu sehen.

Unify-Cl $(P) =$
  if *es gibt eine Gleichung* $e = (s =^? t) \in P$,
    so dass fuer die Anwendung einer Unifikationsregeln auf e *gilt* $: P \Rightarrow P'$
  then if s *und* t *haben beide* $\{\!| \cdot | \cdot |\!\}$ *als Wurzelsymbol*
    then *transformiere* e *in semi-geloeste Form* e',
      werden dabei Variablen eliminiert,
      dann eliminiere diese aus e *und* $P - \{e\}$.
      Anschliessend rufe *Unify-Cl* $(e' \cup P - \{e\})$ *auf*
    else *Unify-Cl* (P')
  else if P *ist* **in** *geloester Form* **then return** σ_P
    else return P *ist nicht loesbar*

Figure 4.2. *Unify-Cl*

Satz 4.4.7. *Unify-Cl* terminiert für alle $\mathfrak{L}_{Cl}^{let}$-Unifikationsprobleme P.

Beweis. Einem Unifikationsproblem P wird das Komplexitätsmaß (n_1, n_2) zugeordnet, so dass

- n_1 die Anzahl der Variablen in P ist, die nicht gelöst sind, und
- n_2 sei $p = max\{|s| \mid s =^? t \in P\}$. Für $i = 0, \ldots p$ sei $\eta(i)$ die Anzahl der nicht gelösten Gleichungen $s =^? t \in P$, so dass $|s| = i$. Dann ist n_2 die Liste $[\eta(p), \eta(p-1), \ldots, \eta(0)]$.

Es wird folgende (fundierte) Ordnung auf Listen l_1, l_2 definiert:

$$\begin{aligned} l_1 <_{list} l_2 \ , gdw.\ & length(l_1) < length(l_2) \text{ oder} \\ & length(l_1) = length(l_2) \text{ und } head(l_1) < head(l_2) \text{ oder} \\ & length(l_1) = length(l_2), head(l_1) < head(l_2) \text{ und } tail(l_1) < tail(l_2) \end{aligned}$$

Jede Anwendung einer Unifikationsregel bis auf *U-Dec ii)* verkleinert das Tupel bezüglich der lexikographischen Ordnung auf (n_1, n_2):

	n_1	n_2
Tautology	$\geq$	$>$
Orientation	$\geq$	$>$
Decomposition	$\geq$	$>$
Elimination	$>$	
Umg Decomposition i)	$\geq$	$>$
*Subsort**	$>$	
*CommonSubsort**	$>$	

Die Überlegungen hierzu sind analog zum Beweis der Terminierung von *Unify* (Lemma 3.4.17). Weil $\Sigma^{let}_{\{\!|\cdot|\cdot|\!\}}$ eine einfache Signatur ist, vernachlässigen wir wieder die *Weakening*-Regel, die aber für reguläre, elementare Signaturen so modifiziert werden kann, dass eine Anwendung das Komplexitätsmaß verkleinert.

Eine Anwendung der Regel *U-Dec ii)* vergrößert die Menge der nicht gelösten Variablen von P zunächst um 1, da eine neue, nicht direkt eliminierbare Variable eingeführt wird. Die Gleichung e, die zur Anwendung von *U-Dec ii)* geführt hat, ist von der Form $e = \{\!|s_1, \ldots, s_m|S|\!\} =^? \{\!|t_1, \ldots t_n|T|\!\}$. Eine solche Gleichung wird durch *Unify-Cl* in semi-gelöste Form transformiert, deren Gestalt nach Lemma 4.4.4 bekannt ist. Sind S, T keine Variablen der Sorte U, dann sind wir in Fall 1 des Lemmas. Nach Korollar 4.4.6 sind alle neu eingeführten Variablen in gelöster Form, d.h. n_1 hat sich nicht vergrößert. Die Anzahl der Gleichungen des Unifikationsproblems kann angewachsen sein, da neue Gleichungen (in die Menge H) eingeführt werden können. Allerdings werden die beiden Umgebungsterme der Unifikationsgleichung e komplett zerlegt. Durch diese Transformation in semi-gelöste Form verkleinert sich n_2. Diese Überlegung gilt auch, wenn S und T Umgebungsvariablen sind mit $S = T$, da dann beide Variablen durch die Konstante $\emptyset$ ersetzt werden (durch die Regel *U-Dec∗*) und das resultierende Unifikationsproblem in semi-gelöste Form transformiert wird.

Wenn $S \neq T$ beides Variablen sind, dann wird e zu einer semi-gelösten Form der Gestalt aus Fall 2 (bzw. 3 oder 4, siehe unten) des Lemmas 4.4.4 transformiert. Nach Korollar 4.4.6 wurde eine neue Variable eingeführt, die nicht in gelöster Form

ist. Die beiden Variablen S, T sind aus P eliminierbar. Nach diesen beiden *Variable Elimination*-Schritten verringert sich die Anzahl der nicht gelösten Variablen n_1 um 1.

Ist mindestens einer der beiden Terme S oder T eine Variable der Sorte U, dann endet die Transformation mit einem Unifikationsproblem, dessen Gestalt Fall 3 oder 4 aus Lemma 4.4.4 entspricht. Alle neuen Variablen sind in gelöster Form (nach Korollar 4.4.6). Eine der beiden Variablen S oder T ist aus P eliminierbar, wodurch n_1 um 1 verkleinert wird. D.h. nach einer beliebigen Folge von Transformationen einer Gleichung zwischen Umgebungen in semi-gelöste Form und anschließender *Variable Elimination* (die unter Umständen zweimal durchgeführt werden muss), verringert sich das Komplexitätsmaß:

	n_1	n_2	n_3	
U-Dec ii)	$\geq$	$>$		Fall 1 aus Lemma 4.4.4, oder
U-Dec ii)	$>$			Fälle 2,3 oder 4 aus Lemma 4.4.4. $\square$

4.4.2 Vollständigkeit

Die Vollständigkeit aller verwendeten Unifikationsregeln, außer *U-Dec* wurde bereits gezeigt (Lemma 3.4.15). Bei der *U-Dec*-Regel handelt es sich offensichtlich nicht um eine vollständige Transformationsregel. Der Vollständigkeitsbeweis muss deshalb zeigen, dass *Unify-Cl* eine vollständige Unifikationsprozedur in dem Sinne ist, dass für alle $\mathfrak{E}_{Cl}^{let}$-Unifikationsprobleme und für alle $\tau \in U_{\mathfrak{E}_{Cl}^{let}}(P)$ ein Unifikationsproblem S in gelöster Form existiert, so dass P zu S transformierbar ist und $\sigma_S \lesssim_{\mathfrak{E}_{Cl}^{let}} \tau$ gilt. Hierbei kann man sich, wegen der Vorgehensweise des Unifikationsalgorithmus und der bereits gezeigten Vollständigkeit der anderen Unifikationsregeln, auf die Transformationsfolgen von Teilproblemen der Art $\{\!|\cdot|\cdot|\!\} =^? \{\!|\cdot|\cdot|\!\}$ in semi-gelöste Form beschränken. Unter Verwendung von Lemma 4.4.4 kann für diese Transformationsfolgen gezeigt werden, dass sie korrekte (Folgen von) Transformationen sind. Betrachtet man dann den vollständigen Baum aller nichtdeterministischen Verzweigungen der Transformationen (siehe Definition 4.4.11) und verwendet die Terminierung des Unifikationsalgorithmus, kann die Vollständigkeit von *Unify-Cl* gezeigt werden. Dieses Vorgehen ist allerdings technisch etwas aufwendig. Aus diesem Grund vollziehen wir eine Überlegung aus Dovier, Policriti, und Rossi (1998) nach, die die Vollständigkeit auf eine andere Art zeigt.

Dazu lösen wir uns etwas von der durch Termersetzung geprägten Sichtweise auf Identitäten, die wir bisher als Ersetzungsregeln verstanden haben, die es erlauben, in Termen Instanzen einer Seite durch die entsprechende Instanz der anderen Seite zu ersetzen. Wir betrachten das *Cl*-Axiom als Formel der Logik erster Stufe, die

Vertauschbarkeit von Elementen einer Multimenge (d.h. Umgebung) axiomatisiert:

$$Cl := \forall x, y, z : \{\!|x|\{\!|y|z|\!\}|\!\} = \{\!|y|\{\!|x|z|\!\}|\!\}.$$

Ein anderes Axiom, das ebenfalls Gleichheit von Multimengen unter der Eigenschaft der Vertauschbarkeit von Elementen axiomatisiert, ist durch die Formel:

$$\begin{aligned} K := \forall y_1, y_2, v_1, v_2 : \{\!|y_1|v_1|\!\} = \{\!|y_2|v_2|\!\} \Leftrightarrow \\ (y_1 = y_1 \wedge v_1 = v_2) \vee \\ \exists z : (v_1 = \{\!|y_2|z|\!\} \wedge v_2 = \{\!|y_1|z|\!\}). \end{aligned}$$

gegeben.

Die beiden Axiome stehen in folgendem Verhältnis zueinander:

Proposition 4.4.8. Die Axiome K und Cl sind äquivalent.

Beweis. $K \Rightarrow Cl$: Seien $\{\!|x, y|z|\!\}$ und $\{\!|y, x|z|\!\}$ gegeben mit $x \neq y$ (da für $x = y$ die Gleichheit offensichtlich ist). Nach K gilt $\{\!|x, y|z|\!\} = \{\!|y, x|z|\!\}$, gdw. es ein k gibt, so dass $\{\!|y|z|\!\} = \{\!|y|k|\!\}$ und $\{\!|x|z|\!\} = \{\!|x|k|\!\}$. Wähle k gleich z und wir haben $\{\!|x, y|z|\!\} = \{\!|y, x|z|\!\}$.

$Cl \Rightarrow K$: Seien ebenfalls $\{\!|x, y|z|\!\}$ und $\{\!|y, x|z|\!\}$ gegeben mit $x \neq y$. Nach Cl haben wir $\{\!|x, y|z|\!\} = \{\!|y, x|z|\!\}$, d.h. $\{\!|y|z|\!\} = \{\!|y|k|\!\}$ und $\{\!|x|z|\!\} = \{\!|x|k|\!\}$ gilt für $k = z$. □

Satz 4.4.9. *Unify-Cl* ist eine vollständige Unifikationsprozedur für $\mathcal{E}_{Cl}^{let}$ -Unifikationsprobleme.

Beweis. Alle Unifikationsregeln außer *U-Dec* und *U-Dec∗* sind vollständige Transformationen für $\mathcal{E}_{Cl}^{let}$ -Unifikationsprobleme, wie in Lemma 3.4.15 des vorhergehenden Kapitels gezeigt wurde. Die *U-Dec∗*-Regel ist eine vollständige Unifikationsregel, weil eine Unifikationsgleichung $\{\!|s_1, \ldots, s_m|X|\!\} =^? \{\!|t_1, \ldots, t_n|X|\!\}$ eine Lösung besitzt gdw. $\{\!|s_1, \ldots, s_m|\!\} =^? \{\!|t_1, \ldots, t_n|\!\}$ eine Lösung besitzt. Eine Gleichung $\{\!|s_1, \ldots, s_m|\!\} =^? \{\!|t_1, \ldots, t_n|\!\}$ besitzt keine Lösung, wenn $n \neq m$ gilt, d.h. *U-Fail* ist eine vollständige Unifikationsregel. Die Vollständigkeit von *Unify-Cl* folgt aus Proposition 4.4.8 und der strikten Analogie der *U-Dec*-Regel zu Axiom K. □

4.4.3 Unifikationstyp, Berechnung einer vollständigen Menge von Unifikatoren und Komplexität

Satz 4.4.10. Die Gleichungstheorie $\mathcal{E}_{Cl}^{let}$ besitzt den Unifikationstyp endlich.

Beweis. Der Algorithmus *Unify-Cl* zur Lösung von $\mathcal{E}_{Cl}^{let}$ -Unifikationsproblemen terminiert (Satz 4.4.7) und ist vollständig (Satz 4.4.9). Außerdem besitzt der Baum der nichtdeterministischen Verzweigungsmöglichkeiten der Transformationen einen endlichen Verzweigungsgrad (auf jedes endliche $\mathcal{E}_{Cl}^{let}$ -Unifikationsproblem sind nur endlich viele Transformationen anwendbar). Damit folgt, dass $\mathcal{E}_{Cl}^{let}$ den Unifikationstyp endlich besitzt. □

Die Berechnung aller Überlappungen zwischen $\Sigma_{\{|\cdot|\cdot|\}}^{let}$-Termen ist also mit *Unify-Cl* möglich. Der Algorithmus *Unify-Cl* berechnet nichtdeterministisch einen Unifikator für ein Unifikationsproblem aus der vollständigen Menge der Unifikatoren.[12] Um die komplette vollständige Menge von Unifikatoren zu berechnen, betrachten wir die vollständige Menge von Alternativen von Transformationen (Definition 3.4.11). Für ein $\mathcal{E}$-Unifikationsproblem P wird $CU_{\mathcal{E}}(P)$ berechnet, indem alle Transformationsmöglichkeiten $P \Rightarrow P_1 \ldots P \Rightarrow P_n$ betrachtet werden, so dass $CU_{\mathcal{E}}(P) = CU_{\mathcal{E}}(P)|_{Var(P)} \cup \cdots \cup CU_{\mathcal{E}}(P)|_{Var(P)}$ gilt. D.h. der Baum aller nichtdeterministischen Verzweigungsmöglichkeiten von Transformationen wird berechnet. Dazu ist es in $\mathcal{E}_{Cl}^{let}$ ausreichend, anstatt der nichtdeterministischen Wahl einer Transformation bei der Anwendung von *U-Dec*, beide Transformationen *i*) und *ii*) anzuwenden. Anschließend werden beide resultierenden Unifikationsprobleme rekursive weiter transformiert. D.h. für ein Teilproblem $\{|\cdot|\cdot|\} =^? \{|\cdot|\cdot|\}$ werden alle möglichen semi-gelösten Formen berechnet, was durch folgende Modifikation der *U-Dec* Regel geschehen kann:

Definition 4.4.11. Zur Berechnung einer vollständigen Menge von Unifikatoren eines $\mathcal{E}_{Cl}^{let}$-Unifikationsproblems wird die *U-Decompositon*-Regel aus Definition 4.4.2 durch folgende Regel ersetzt:

CU – Decomposition

$$\{\{|t|s|\} =^? \{|t'|s'|\}\} \uplus P \Rightarrow \quad \{\{t =^? t', s =^? s'\} \cup P\} \cup \{\{s =^? \{|t'|N|\}, \{|t|N|\} =^? s'\} \cup P\}$$

wenn $tail(s)$ und $tail(s')$ nicht dieselbe Variable X der Sorte U bezeichne.
N ist eine neue Variable der Sorte U.

Der Algorithmus mit dieser modifizierten Regel berechnet eine vollständige Menge von Unifikatoren für ein Unifikationsproblem, allerdings nicht die minimale vollständige Menge von Unifikatoren, wie an folgendem Beispiel zu sehen ist:

Beispiel 4.4.12. Sei $P = \{\{|x, x|S|\} =^? \{|x|T|\}\}$ ein $\mathcal{E}_{Cl}^{let}$-Unifikationsproblem.

[12] Die Überlegungen hierzu sind analog zu denen in Abschnitt 3.4.1: *Unify-Cl* transformiert ein Unifikationsproblem in gelöste Form, die einen idempotenten Unifikator aus der Menge der vollständigen Unifikatoren repräsentiert.

$$
\begin{array}{rll}
 & \{\{\!|x, x|S|\!\} =^? \{\!|x|T|\!\}\} & = P \\
\hline
\xrightarrow{\textit{U-Dec-C}} & \{\{x =^? x, \{\!|x|S|\!\} =^? T\}, & \\
 & \{\{\!|x|S|\!\} =^? \{\!|x|N_1|\!\}, \{\!|x|N_1|\!\} =^? T\}\} & \\
\hline
\xrightarrow{\textit{Taut}} & \{\{\{\!|x|S|\!\} =^? T\}, & \\
\xrightarrow{\textit{U-Dec-C}} & \{x =^? x, S =^? N_1, \{\!|x|N_1|\!\} =^? T\} & \\
 & \{S =^? \{\!|x|N_2|\!\}, \{\!|x|N_2|\!\} =^? N_1, \{\!|x|N_1|\!\} =^? T\}\} & \\
\hline
\xrightarrow{\textit{Orient}} & \{\{T =^? \{\!|x|S|\!\}\}, & = S_1 \\
\xrightarrow{\textit{Taut, Orient} \times 2, \textit{Elim } N_1} & \{N_1 =^? S, T =^? \{\!|x|S|\!\}\} & = S_2 \\
\xrightarrow{\textit{Orient} \times 2, \textit{Elim } N_1} & \{S =^? \{\!|x|N_2|\!\}, N_1 =^? \{\!|x|N_2|\!\}, T =^? \{\!|x, x|N_2|\!\}\}\} & = S_3
\end{array}
$$

Das Unifikationsproblem P besitzt einen mgu, der zweimal berechnet wird: $\sigma_{S_1} =_{\mathcal{E}_{Cl}^{let}} \sigma_{S_2}[\{S, T\}]$. Für den dritten berechneten Unifikator gilt $\sigma_{S_3} =_{\mathcal{E}_{Cl}^{let}} \sigma_{S_3}\sigma_{S_1}$.

Dass die berechnete vollständig Menge der Unifikatoren nicht minimal ist, stellt für die Berechnung aller Überlappungen zwischen Termen kein Problem dar. Wichtig ist nur, dass alle möglichen (kritischen) Überlappungen berechnet werden. Außerdem werden die zu überlappenden Terme umbenannt und sind deswegen variablendisjunkt. Deshalb kann ein Fall, wie in Beispiel 4.4.12, bei der Berechnung von Überlappungen nicht auftreten. Arenas-Sanchez und Dovier (1997) und Stolzenburg (1999) geben eine Reihe von Verbesserungen des Algorithmus an, um die Anzahl der redundanten Unifikatoren möglichst gering zu halten.

Unifikation von Multimengen ist, ebenso wie Unifikation von *AC*-Funktionssymbolen, NP-vollständig (Kapur & Narendran, 1986; Dovier, Policriti, & Rossi, 1998; Kapur & Narendran, 1992a) und die Anzahl der Unifikatoren kann exponentiell in der Größe der Eingabe sein (Kapur & Narendran, 1992b). Des Weiteren kopiert *Unify-Cl* Terme bei der Elimination von Variablen. Dieses Verhalten kann allerdings verbessert werden durch die Verwendung von Datenstrukturen, die das Kopieren von Termen während der Unifikation vermeiden, wie Multimengen von Unifikationsgleichungen (Martelli & Montanari, 1982, siehe auch Abschnitt 3.4.3) oder die Darstellung von Termen als DAG (Corbin & Bidoit, 1983).

Bei der Berechnung von Überlappungen für Λ^{let} Reduktionsregeln haben wir es nur mit Umgebungen zu tun, die nicht mehr als drei Elemente enthalten (wenn Ketten nicht berücksichtigt werden). Aufgrund dieser geringen Eingabegrößen ist die exponentielle Laufzeit und die exponentielle Anzahl von Unifikatoren (im worst-case) für die Berechnung dieser spezifischen Überlappungen vernachlässigbar.

5 Unifikation von Termen mit Variablenketten

Es gibt zwei Arten von Ketten von `letrec`-Bindungen in Λ^{let}. Die so genannten *Variablenketten* $\{x_i = x_{i-1}\}_{i=m}^{n}$in den (cp)-Reduktionsregeln und die Ketten $\{x_i = R_i^-[x_{i-1}]\}_{i=m}^{n}$, die in Reduktionskontexten auftauchen, die so genannten *Reduktionsketten*. Die Möglichkeiten, diese Ketten bei der Unifikation zu berücksichtigen sind:

1. Für einen zu unifizierenden Ausdruck, der eine Kette enthält, werden Terme mit Variablenketten verschiedener Länge instantiiert. Beispielsweise kann der Ausdruck ($\texttt{letrec}\ \{x_1 = v, \{x_i = x_{i-1}\}_{i=2}^{n} Env\}\ \texttt{in}\ C[x_n]$) übersetzt werden zu:

   - $letrec(\{\!|b(x_1, v)[1]|e|\!\}, C(x_1))$ (Variablenkette der Länge 0),
   - $letrec(\{\!|b(x_1, v), b(x_2, x_1)|e|\!\}, C(x_2))$ (Variablenkette der Länge 1),
   - $letrec(\{\!|b(x_1, v), b(x_2, x_1), b(x_3, x_2)|e|\!\}, C(x_3))$ (Variablenkette der Länge 2),
   - ...

   Aus diesem Vorgehen ergibt sich das Problem, dass Ketten variabler Länge so nicht unifizierbar sind. Es kann nur eine begrenzte Anzahl Instanzen von Reduktionsregeln mit Ketten unterschiedlicher Länge auf Unifizierbarkeit getestet werden. Die Unifikationsprozedur ist somit nicht vollständig. Außerdem wächst die Laufzeit der Unifikation wegen der NP-Vollständigkeit der Cl-Unifikation mit zunehmender Länge der Ketten exponentiell.

2. Die zweite Möglichkeit besteht darin, ein Konstrukt einzuführen, das Variablenketten beliebiger Länge darstellen kann.

Für Variablenketten in Σ^{let} betrachten wir die zweite Alternative. Für Reduktionsketten wird die erste Möglichkeit gewählt.

In diesem Kapitel wird die Methode vorgestellt, um Terme zu unifizieren, die Variablenketten beliebiger Länge enthalten. Dazu erweitern wir in Abschnitt 5.1 die Signatur Σ^{let} um ein Funktionssymbol ch, mit den Variablenketten dargestellt werden können. Für dieses Funktionssymbol werden drei Axiome definiert, durch die

[1] Wir verwenden, wie bisher üblich, für das Funktionssymbol *bind* die Abkürzung b.

sich wesentliche Eigenschaften von Variablenketten beschreiben lassen: Einzelne Bindungen aus können Variablenketten abgespalten und separat unifiziert werden. Der Theoretische Rahmen dieses Kapitels ist, wie im vorhergehenden Kapitel, die Theorie der Gleichungsunifikation.

In Abschnitt 5.1 wird zuerst eine vollständige Unifikationsprozedur für Terme mit Variablenketten skizziert, die allerdings nur zu rekursiven Aufzählung von Unifikatoren verwendet werden kann. Über ihre Terminierung kann keine Aussage getroffen werden. In den folgenden Unterabschnitten stellen wir Überlegungen an, wie sich die Anwendung der Axiome, die für Variablenketten gelten, beschränken lassen, um die Terminierung der Unifikationsprozedur sicherzustellen. Dazu werden die entsprechenden Unifikationsregeln entwickelt.

5.1 Variablenketten in Σ^{let}

Definition 5.1.1. Die Signatur Σ^{let} wird um ein Konstrukt zur Darstellung von Variablenketten beliebiger Länge erweitert. Dazu wird eine neue Sorte K für Variablenketten eingeführt und eine neue Sorte M, die es ermöglicht, Ketten als Bestandteile der `letrec`-Umgebung $\{\!|\cdot|\cdot|\!\}$ darzustellen. Wir notieren nur die relevanten neuen Bestandteile der Signatur.

$$
\begin{aligned}
\Sigma^{let} = \{\ & \textbf{Subsortdeklarationen}: \\
& B, K \sqsubset M \sqsubset U, \\
& \textbf{Funktionsdeklarationen}: \\
& ch : B \to B \to K \qquad (\mathsf{Variablenkette}), \\
& \{\!|\cdot|\cdot|\!\} : M \to U \to U \ \ (\texttt{letrec} - \mathsf{Umgebung}).\ \}
\end{aligned}
$$

Eine Variablenkette in Σ^{let} ist durch ihre Anfangs- und Endbindung definiert. Variablenketten enthalten per Konvention nur `letrec`-Bindungen, die ausschließlich Variablen der Sorte V enthalten. So stellt beispielsweise $ch(b(x_2, x_1), b(x_n, x_{n-1}))$ eine Variablenkette in Σ^{let} dar, wenn x_1, x_2, x_{n-1} und x_n Variablen der Sorte V sind. $ch(b(x_1, s_T), b(x_n, x_{n-1}))$ ist keine Variablenkette, da s eine Variable der Sorte T ist. Variablenketten werden folgendermaßen durch $[\![\ ]\!] : \Lambda^{let} \to \Sigma^{let}$ von Λ^{let} nach Σ^{let} übersetzt:

$$
\begin{aligned}
[\![\{\{x_i = x_{i-1}\}_{i=m}^{n}, x_1 = s_1, \ldots\}]\!] &= \{\!|[\![\{x_i = x_{i-1}\}_{i=m}^{n}]\!]|[\![\{x_1 = s_1, \ldots\}]\!]|\!\} \\
[\![\{x_i = x_{i-1}\}_{i=m}^{n}]\!] &= ch(b([\![x_m]\!], [\![x_{m-1}]\!]), b([\![x_n]\!], [\![x_{n-1}]\!])
\end{aligned}
$$

Das Verhalten von Variablenketten, dass einzelne Bindungen separat betrachtet

(bzw. unifiziert) werden können, lässt sich durch folgende Identitäten beschreiben:

$$
\begin{aligned}
&E_{Split} = \{ \\
&\{\!|ch(b(x_V, y_V), b_B)|e_U|\!\} \approx \{\!|b(x_V, y_V)|\{\!|ch(b(z_V, x_V), b_B)|e_U|\!\}|\!\} && \text{Split A,} \\
&\{\!|ch(b_B, b(x_V, y_V))|e_U|\!\} \approx \{\!|ch(b_B, b(y_V, z_V))|\{\!|b(x_V, y_V)|e_U|\!\}|\!\} && \text{Split E,} \\
&\{\!|ch(b_B, d_B)|e_U|\!\} \approx \{\!|(ch(b_B, b(w_V, x_V))|\{\!|b(y_V, w_V))|\{\!|ch(b(z_V, y_V), d_B)|e_U|\!\}|\!\}|\!\} && \text{Split M.} \\
&\}
\end{aligned}
$$

Eine Definition der Split-Axiome in der Form

$$ch(b(x_V, y_V), b_B) \approx \{\!|b(x_V, y_V)|ch(b(z_V, x_V), b_B)|\!\} \tag{5.1}$$

scheint natürlicher als die oben gewählte, bei der die Ketten in `letrec`-Umgebungen stehen müssen, um aufgespalten zu werden. Allerdings ist die Axiom-Variante (5.1) nicht sort preserving (Definition 4.3.1) und eine Anwendung kann dazu führen, dass das *Cl*-Axiom nicht mehr anwendbar ist. Diese beiden Probleme werden durch die gewählte Definition vermieden. Für alle Terme mit Variablenketten, die zur Berechnung von Überlappungen betrachtet werden müssen, gilt, dass die Variablenketten in `letrec`-Umgebungen vorkommen, d.h. die Split-Axiome anwendbar sind. Außerdem haben alle Variablenketten in diesen Termen die Form $ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))$.

Die drei Split-Axiome erlauben das Abspalten einer Bindung vom Anfang einer Variablenkette (*Split A*), vom Ende einer Variablenkette (*Split E*) oder aus der Mitte einer Kette (*Split M*). Eine Berücksichtigung dieser drei Positionen ist ausreichend: Betrachtet man die linken Seiten der *cp*-Reduktionsregeln, in denen Variablenketten vorkommen, dann sieht man, dass Variablen der Anfangsbindung einer Kette noch einmal in der linken Seite der jeweiligen Reduktionsregel vorkommen. Beispielsweise ($\texttt{letrec}\ \{x_1 = v, \{x_i = x_{i-1}\}_{i=2}^{n}, Env\}\ \texttt{in}\ C[x_n]$), wobei $x_2 = x_1$ die Anfangsbindung der Kette ist und x_1 noch einmal in $x_1 = v$ vorkommt. Ebenso kommt eine Variable aus der Endbindung der Kette noch einmal in der linken Seite der Reduktionsregel vor. Sonstige Variablen der Kette tauchen in der linken Seite der Reduktionsregel nicht mehr auf. Eine Variablenkette komplett in ihre konstituierenden Bindungen zu zerlegen, ist nicht möglich, da Ketten beliebiger Länge betrachtet werden sollen. Deshalb werden durch die drei Split-Axiome die beiden Bindungen einer Kette bei der Unifikation berücksichtigt, die Variablen enthalten, die an anderer Stelle in der linken Seite der Reduktionsregel noch einmal vorkommen und stellvertretend eine Bindung, aus der keine Variable mehr an anderer Stelle vorkommt.

Durch die Einführung der neuen Sorte M und der Anpassung der Sorte von $\{\!|\cdot|\cdot|\!\}$ zu $M \to U \to U$ muss das *Cl*-Axiom, das die Vertauschbarkeit von Elementen in `letrec`-Umgebungen formuliert, zu

$$E_{Cl} = \{\ \{\!|c_M|\{\!|d_M|e_U|\!\}|\!\} \approx \{\!|d_M|\{\!|c_M|e_U|\!\}|\!\}\ \}$$

angepasst werden. Die Kongruenzrelation auf Termen, die nun betrachtet wird, ist $=_{\mathcal{E}}$ mit $\mathcal{E} = (\Sigma^{let}, E_{Cl} \cup E_{Split})$, wobei Σ^{let} über `letrec`-Umgebungen und Variablenketten verfügt. Bei der Unifikation von Σ^{let}-Termen mit `letrec`-Umgebungen und Variablenketten müssen die beiden Gleichungstheorien E_{Cl} und E_{Split} berücksichtigt werden. Um dies formal zu behandeln, wird die Theorie der *Kombination von Unifikationsalgorithmen* (Schmidt-Schauß, 1989b; Baader & Schulz, 1996) benötigt. Dabei kommt erschwerend hinzu, dass die Identitäten *Cl* und *Split*, die die Gleichungstheorien definieren, über der gleichen Signatur Σ^{let} definiert sind, und somit die Signaturen der beiden Theorien nicht disjunkt sind. Der Aspekt der Kombination von Unifikationsalgorithmen für die Theorien *Cl* und *Split* wird hier nicht formal behandelt, sondern wir stützen uns auf die Erfahrungen, die bei der Programmierung des Unifikationsalgorithmus für `letrec`-Umgebungen mit vertauschbaren Elementen und Variablenketten gewonnen wurden. Ein formaler Nachweis, dass Unifikationsalgorithmen für die beiden Theorien tatsächlich problemlos kombiniert werden können, muss noch erbracht werden.

Um die Split-Axiome bei der Unifikation zu berücksichtigen, benötigen wir eine Variante der Mutationsregel (aus Definition 4.2.1), die so genannte *Lazy Paramodulation* (Gallier & Snyder, 1989; Baader & Snyder, 2001), die etwas liberaler bezüglich der Position ist, an der eine Identität angewendet werden kann.

5.2 Unifikation von Termen mit Variablenketten

Definition 5.2.1. Die Regeln zur *Cl*-Unifikation in Σ^{let} (Definition 4.4.2) werden um folgende Regel erweitert (nach Gallier und Snyder (1989)):

Lazy Paramodulation
$\{s[t]\} \uplus P \Rightarrow \{l =^? t, s[r]\} \cup P$
wobei $l \approx r$ eine Version mit neuen Variablen von $l \approx r \in E \cup E^-$ ist und t keine Variable ist. Wenn l keine Variable ist,
dann sind die Wurzelsymbole Symbole von l und t gleich und
auf $l =^? t$ muss als nächstes *Decomposition* angewendet werden.

Beispiel 5.2.2. Wir geben ein ausführliches Beispiel an, wie Terme mit Variablenketten in Σ^{let} mit Hilfe der *Lazy Paramodulation*-Regel (abgekürzt als *LP*) unifiziert werden können. Es wird ein Unifikator aus der vollständigen Menge der Unifikatoren berechnet für die Überlappung einer $(n, cp\text{-}e)$-Reduktion mit einer internen $(cp\text{-}e)$-Reduktion. Die Sorten der verwendeten Variablen sind $x, x_i, y, y_i : V$; $v, w : Ap$; $s, t : T$ und $e, d : U$. Die Sorten der neu eingeführten Variablen sind $e', N_i : U$ und $b_1, b_2 : B$. Die zu unifizierenden Terme enthalten Kontextvariablen

C, R_1^-, R_2^-, was ein Vorgriff auf den Inhalt des Kapitels zur Unifikation mit Kontextvariablen darstellt (Kapitel 6). Die Terme $C(y_n), R_2^-(x)$ und $R_1^-(app(x_m, s))$ können vereinfacht als Terme der Sorte T mit einem entsprechenden Subterm verstanden werden.

$$
\begin{aligned}
&\{letrec(\{\!|b(x_1,v), b(x, R_1^-(app(x_m,s))), ch(b(x_2,x_1), b(x_m,x_{m-1}))|e|\!\}, R_2^-(x))\\
&\quad =^? letrec(\{\!|b(y_1,w), b(y, C(y_n)), ch(b(y_2,y_1), b(y_n,y_{n-1}))|d|\!\}, t)\}\\
&\qquad \xrightarrow{Dec} \{\{\!|b(x_1,v), b(x, R_1^-(app(x_m,s))), ch(b(x_2,x_1), b(x_m,x_{m-1}))|e|\!\}\\
&\qquad\quad =^? \{\!|b(y_1,w), b(y, C(y_n)), ch(b(y_2,y_1), b(y_n,y_{n-1}))|d|\!\},\\
&\qquad\quad R_2^-(x) =^? t\}\\
&\qquad \xrightarrow{U\text{-}Dec\ i)} \{\{\!|b(x, R_1^-(app(x_m,s))), ch(b(x_2,x_1), b(x_m,x_{m-1}))|e|\!\}\\
&\qquad\quad =^? \{\!|b(y, C(y_n)), ch(b(y_2,y_1), b(y_n,y_{n-1}))|d|\!\},\\
&\qquad\quad b(x_1,v) =^? b(y_1,w), R_2^-(x) =^? t\}\\
&\qquad \xrightarrow{LP\ Split\ M} \{\{\!|b(x, R_1^-(app(x_m,s))),\\
&\qquad\quad ch(b_1, b(z_2,z_1)), b(z_3,z_2)), ch(b(z_4,z_3), b_2)|e|\!\}\\
&\qquad\quad =^? \{\!|b(y, C(y_n)), ch(b(y_2,y_1), b(y_n,y_{n-1}))|d|\!\},\\
&\qquad\quad \{\!|ch(b_1,b_2)|e'|\!\} =^? \{\!|ch(b(x_2,x_1), b(x_m,x_{m-1}))|e|\!\}\\
&\qquad\quad b(x_1,v) =^? b(y_1,w), R_2^-(x) =^? t\}\\
&\qquad \xrightarrow{U\text{-}Dec\ i)} \{\{\!|b(x, R_1^-(app(x_m,s))),\\
&\qquad\quad ch(b_1, b(z_2,z_1)), b(z_3,z_2)), ch(b(z_4,z_3), b_2)|e'|\!\}\\
&\qquad\quad =^? \{\!|b(y, C(y_n)), ch(b(y_2,y_1), b(y_n,y_{n-1}))|d|\!\},\\
&\qquad\quad ch(b_1,b_2) =^? ch(b(x_2,x_1), b(x_m,x_{m-1})), e' =^? e\\
&\qquad\quad b(x_1,v) =^? b(y_1,w), R_2^-(x) =^? t\}\\
&\qquad \xrightarrow{U\text{-}Dec\ ii)} \{\{\!|ch(b_1, b(z_2,z_1)), b(z_3,z_2)), ch(b(z_4,z_3), b_2)|e'|\!\} =^? \{\!|b(y, C(y_n))|N_1|\!\},\\
&\qquad\quad \{\!|b(x, R_1^-(app(x_m,s)))|N_1|\!\} =^? \{\!|ch(b(y_2,y_1), b(y_n,y_{n-1}))|d|\!\},\\
&\qquad\quad ch(b_1,b_2) =^? ch(b(x_2,x_1), b(x_m,x_{m-1})), e' =^? e\\
&\qquad\quad b(x_1,v) =^? b(y_1,w), R_2^-(x) =^? t\}\\
&\qquad \xrightarrow{U\text{-}Dec\ ii)} \{\{\!|b(z_3,z_2)), ch(b(z_4,z_3), b_2)|e'|\!\} =^? \{\!|b(y, C(y_n))|N_2|\!\},\\
&\qquad\quad \{\!|ch(b_1, b(z_2,z_1))|N_2|\!\} =^? N_1,\\
&\qquad\quad \{\!|b(x, R_1^-(app(x_m,s)))|N_1|\!\} =^? \{\!|ch(b(y_2,y_1), b(y_n,y_{n-1}))|d|\!\},\\
&\qquad\quad ch(b_1,b_2) =^? ch(b(x_2,x_1), b(x_m,x_{m-1})), e' =^? e\\
&\qquad\quad b(x_1,v) =^? b(y_1,w), R_2^-(x) =^? t\}\\
&\qquad \xrightarrow{U\text{-}Dec\ i)} \{b(z_3,z_2)) =^? b(y, C(y_n)), \{\!|ch(b(z_4,z_3), b_2)|e'|\!\} =^? N_2,\\
&\qquad\quad \{\!|ch(b_1, b(z_2,z_1))|N_2|\!\} =^? N_1,\\
&\qquad\quad \{\!|b(x, R_1^-(app(x_m,s)))|N_1|\!\} =^? \{\!|ch(b(y_2,y_1), b(y_n,y_{n-1}))|d|\!\},\\
&\qquad\quad ch(b_1,b_2) =^? ch(b(x_2,x_1), b(x_m,x_{m-1})), e' =^? e
\end{aligned}
$$

$$
\begin{aligned}
& b(x_1, v) =^? b(y_1, w), R_2^-(x) =^? t\} \\
\xrightarrow{\textit{U-Dec ii)}} \; & \{b(z_3, z_2)) =^? b(y, C(y_n)), \{\!|ch(b(z_4, z_3), b_2)|e'|\!\} =^? N_2, \\
& \{\!|ch(b_1, b(z_2, z_1))|N_2|\!\} =^? N_1, \\
& N_1 =^? \{\!|ch(b(y_2, y_1), b(y_n, y_{n-1}))|N_3|\!\}, \{\!|b(x, R_1^-(app(x_m, s)))|N_3|\!\} =^? d, \\
& ch(b_1, b_2) =^? ch(b(x_2, x_1), b(x_m, x_{m-1})), e' =^? e \\
& b(x_1, v) =^? b(y_1, w), R_2^-(x) =^? t\} \\
\xrightarrow{\textit{Orient } N_1} \; & \{b(z_3, z_2)) =^? b(y, C(y_n)), \{\!|ch(b(z_4, z_3), b_2)|e'|\!\} =^? N_2, \\
\xrightarrow{\textit{Elim } N_1} \; & N_1 =^? \{\!|ch(b_1, b(z_2, z_1))|N_2|\!\}, \\
& \{\!|ch(b_1, b(z_2, z_1))|N_2|\!\} =^? \{\!|ch(b(y_2, y_1), b(y_n, y_{n-1}))|N_3|\!\}, \\
& \{\!|b(x, R_1^-(app(x_m, s)))|N_3|\!\} =^? d, ch(b_1, b_2) =^? ch(b(x_2, x_1), b(x_m, x_{m-1})), \\
& e' =^? e, b(x_1, v) =^? b(y_1, w), R_2^-(x) =^? t\} \\
\xrightarrow{\textit{U-Dec i)}} \; & \{b(z_3, z_2)) =^? b(y, C(y_n)), \{\!|ch(b(z_4, z_3), b_2)|e'|\!\} =^? N_2, \\
& N_1 =^? \{\!|ch(b_1, b(z_2, z_1))|N_2|\!\}, \\
& ch(b_1, b(z_2, z_1)) =^? ch(b(y_2, y_1), b(y_n, y_{n-1})), N_2 =^? N_3, \\
& \{\!|b(x, R_1^-(app(x_m, s)))|N_3|\!\} =^? d, ch(b_1, b_2) =^? ch(b(x_2, x_1), b(x_m, x_{m-1})), \\
& e' =^? e, b(x_1, v) =^? b(y_1, w), R_2^-(x) =^? t\} \\
\xrightarrow{\textit{Dec}} \; & \{z_3 =^? y, z_2 =^? C(y_n), \{\!|ch(b(z_4, z_3), b_2)|e'|\!\} =^? N_2, \\
& N_1 =^? \{\!|ch(b_1, b(z_2, z_1))|N_2|\!\}, \\
& ch(b_1, b(z_2, z_1)) =^? ch(b(y_2, y_1), b(y_n, y_{n-1})), N_2 =^? N_3, \\
& \{\!|b(x, R_1^-(app(x_m, s)))|N_3|\!\} =^? d, ch(b_1, b_2) =^? ch(b(x_2, x_1), b(x_m, x_{m-1})), \\
& e' =^? e, b(x_1, v) =^? b(y_1, w), R_2^-(x) =^? t\} \\
\xrightarrow{\textit{Con Weak}}{}^{2} \; & \{z_3 =^? y, C =^? \square, z_2 =^? y_n, \{\!|ch(b(z_4, z_3), b_2)|e'|\!\} =^? N_2, \\
& N_1 =^? \{\!|ch(b_1, b(z_2, z_1))|N_2|\!\}, \\
& ch(b_1, b(z_2, z_1)) =^? ch(b(y_2, y_1), b(y_n, y_{n-1})), N_2 =^? N_3, \\
& \{\!|b(x, R_1^-(app(x_m, s)))|N_3|\!\} =^? d, ch(b_1, b_2) =^? ch(b(x_2, x_1), b(x_m, x_{m-1})), \\
& e' =^? e, b(x_1, v) =^? b(y_1, w), R_2^-(x) =^? t\} \\
\xrightarrow{\textit{Dec}} \; & \{z_3 =^? y, C =^? \square, z_2 =^? y_n, \{\!|ch(b(z_4, z_3), b_2)|e'|\!\} =^? N_2, \\
& N_1 =^? \{\!|ch(b_1, b(z_2, z_1))|N_2|\!\}, \\
& b_1 =^? b(y_2, y_1), b(z_2, z_1) =^? b(y_n, y_{n-1}), N_2 =^? N_3, \\
& \{\!|b(x, R_1^-(app(x_m, s)))|N_3|\!\} =^? d, ch(b_1, b_2) =^? ch(b(x_2, x_1), b(x_m, x_{m-1})), \\
& e' =^? e, b(x_1, v) =^? b(y_1, w), R_2^-(x) =^? t\} \\
\xrightarrow{\textit{Dec}} \; & \{z_3 =^? y, C =^? \square, z_2 =^? y_n, \{\!|ch(b(z_4, z_3), b_2)|e'|\!\} =^? N_2,
\end{aligned}
$$

[2] An dieser Stelle wird die Unifikationsregel *Context Weakening* aus Kapitel 6 zur Unifikation von Kontextvariablen verwendet: Die Gleichung $z_2 =^? C(y_n)$ besitzt eine Lösung, wenn C gleich dem leeren Kontext $\square$ ist und z_2 gleich y_n.

$$
\begin{aligned}
& N_1 =^? \{\!|ch(b_1, b(z_2, z_1))|N_2|\!\}, \\
& b_1 =^? b(y_2, y_1), b(z_2, z_1) =^? b(y_n, y_{n-1}), N_2 =^? N_3, \\
& \{\!|b(x, R_1^-(app(x_m, s)))|N_3|\!\} =^? d, b_1 =^? b(x_2, x_1), b_2 =^? b(x_m, x_{m-1}), \\
& e' =^? e, b(x_1, v) =^? b(y_1, w), R_2^-(x) =^? t\} \\
\xrightarrow{Elim\ b_1} & \{z_3 =^? y, C =^? \square, z_2 =^? y_n, \{\!|ch(b(z_4, z_3), b_2)|e'|\!\} =^? N_2, \\
& N_1 =^? \{\!|ch(b_1, b(z_2, z_1))|N_2|\!\}, \\
& b_1 =^? b(y_2, y_1), b(z_2, z_1) =^? b(y_n, y_{n-1}), N_2 =^? N_3, \\
& \{\!|b(x, R_1^-(app(x_m, s)))|N_3|\!\} =^? d, b(y_2, y_1) =^? b(x_2, x_1), b_2 =^? b(x_m, x_{m-1}), \\
& e' =^? e, b(x_1, v) =^? b(y_1, w), R_2^-(x) =^? t\} \\
\xrightarrow{Dec\ \times\ 3} & \{z_3 =^? y, C =^? \square, z_2 =^? y_n, \{\!|ch(b(z_4, z_3), b_2)|e'|\!\} =^? N_2, \\
& N_1 =^? \{\!|ch(b_1, b(z_2, z_1))|N_2|\!\}, \\
& b_1 =^? b(y_2, y_1), z_2 =^? y_n, z_1 =^? y_{n-1}, N_2 =^? N_3, \\
& \{\!|b(x, R_1^-(app(x_m, s)))|N_3|\!\} =^? d, y_2 =^? x_2, y_1 =^? x_1, b_2 =^? b(x_m, x_{m-1}), \\
& e' =^? e, x_1 =^? y_1, v =^? w, R_2^-(x) =^? t\} \\
\xrightarrow{Orient\ \times\ 3} & \{z_3 =^? y, C =^? \square, z_2 =^? y_n, N_2 =^? \{\!|ch(b(z_4, z_3), b_2)|e'|\!\}, \\
& N_1 =^? \{\!|ch(b_1, b(z_2, z_1))|N_2|\!\}, \\
& b_1 =^? b(y_2, y_1), z_2 =^? y_n, z_1 =^? y_{n-1}, N_2 =^? N_3, \\
& d =^? \{\!|b(x, R_1^-(app(x_m, s)))|N_3|\!\}, y_2 =^? x_2, y_1 =^? x_1, b_2 =^? b(x_m, x_{m-1}), \\
& e' =^? e, x_1 =^? y_1, v =^? w, t =^? R_2^-(x)\} \\
\xrightarrow{Elim\ N_2} & \{z_3 =^? y, C =^? \square, z_2 =^? y_n, N_2 =^? \{\!|ch(b(z_4, z_3), b_2)|e'|\!\}, \\
& N_1 =^? \{\!|ch(b_1, b(z_2, z_1))|ch(b(z_4, z_3), b_2)|e'|\!\}, \\
& b_1 =^? b(y_2, y_1), z_2 =^? y_n, z_1 =^? y_{n-1}, \{\!|ch(b(z_4, z_3), b_2)|e'|\!\} =^? N_3, \\
& d =^? \{\!|b(x, R_1^-(app(x_m, s)))|N_3|\!\}, y_2 =^? x_2, y_1 =^? x_1, b_2 =^? b(x_m, x_{m-1}), \\
& e' =^? e, x_1 =^? y_1, v =^? w, t =^? R_2^-(x)\} \\
\xrightarrow{Orient\ N_3} & \{z_3 =^? y, C =^? \square, z_2 =^? y_n, N_2 =^? \{\!|ch(b(z_4, z_3), b_2)|e'|\!\}, \\
\xrightarrow{Elim\ N_3} & N_1 =^? \{\!|ch(b_1, b(z_2, z_1))|ch(b(z_4, z_3), b_2)|e'|\!\}, \\
& b_1 =^? b(y_2, y_1), z_2 =^? y_n, z_1 =^? y_{n-1}, N_3 =^? \{\!|ch(b(z_4, z_3), b_2)|e'|\!\}, \\
& d =^? \{\!|b(x, R_1^-(app(x_m, s)))|ch(b(z_4, z_3), b_2)|e'|\!\}, \\
& y_2 =^? x_2, y_1 =^? x_1, b_2 =^? b(x_m, x_{m-1}), e' =^? e, x_1 =^? y_1, v =^? w, t =^? R_2^-(x)\} \\
\xrightarrow{Elim\ z_1, z_2, z_3} & \{z_3 =^? y, C =^? \square, z_2 =^? y_n, N_2 =^? \{\!|ch(b(z_4, y), b(x_m, x_{m-1}))|e|\!\}, \\
\xrightarrow{Elim\ y_1, y_2} & N_1 =^? \{\!|ch(b(x_2, x_1), b(y_n, y_{n-1})), ch(b(z_4, y), b(x_m, x_{m-1}))|e|\!\}, \\
\xrightarrow{Elim\ b_1, b_2} & b_1 =^? b(x_2, x_1), y_n =^? y_n, z_1 =^? y_{n-1}, N_3 =^? \{\!|ch(b(z_4, y), b(x_m, x_{m-1}))|e|\!\}, \\
\xrightarrow{Elim\ e'} & d =^? \{\!|b(x, R_1^-(app(x_m, s))), ch(b(z_4, y), b(x_m, x_{m-1}))|e|\!\}, \\
& y_2 =^? x_2, y_1 =^? x_1, b_2 =^? b(x_m, x_{m-1}), e' =^? e, x_1 =^? x_1, v =^? w, t =^? R_2^-(x)\}
\end{aligned}
$$

Das Unifikationsproblem in der letzten Zeile ist in gelöster Form und repräsentiert einen Unifikator σ für das Ausgangsproblem:

$$\begin{aligned}\sigma = \{\ & y_1 =^? x_1, y_2 =^? x_2,\\ & z_1 =^? y_{n-1}, z_2 =^? y_n, z_3 =^? y, e' =^? e,\\ & v =^? w, C =^? \Box, t =^? R_2^-(x),\\ & d =^? \{\!|b(x, R_1^-(app(x_m, s))), ch(b(z_4, y), b(x_m, x_{m-1}))|e|\!\},\\ & N_1 =^? \{\!|ch(b(x_2, x_1), b(y_n, y_{n-1})), ch(b(z_4, y), b(x_m, x_{m-1}))|e|\!\},\\ & N_2 =^? \{\!|ch(b(z_4, y), b(x_m, x_{m-1}))|e|\!\},\\ & N_3 =^? \{\!|ch(b(z_4, y), b(x_m, x_{m-1}))|e|\!\},\\ & b_1 =^? b(x_2, x_1), b_2 =^? b(x_m, x_{m-1}) \qquad \}.\end{aligned}$$

Anwendung des berechneten Unifikators auf die beiden Ausgangsterme ergibt

$$\begin{aligned}&\sigma(letrec(\{\!|b(y_1, w), b(y, C(y_n)), ch(b(y_2, y_1), b(y_n, y_{n-1}))|d|\!\}, t))\\ &= letrec(\{\!|b(x_1, w), b(y, y_n), ch(b(x_2, x_1), b(y_n, y_{n-1})),\\ &\qquad b(x, R_1^-(app(x_m, s))), ch(b(z_4, y), b(x_m, x_{m-1}))|e|\!\}, R_2^-(x))\end{aligned}$$

und

$$\begin{aligned}&\sigma(letrec(\{\!|b(x_1, v), b(x, R_1^-(app(x_m, s))), ch(b(x_2, x_1), b(x_m, x_{m-1}))|e|\!\}, R_2^-(x)))\\ &= letrec(\{\!|b(x_1, w), b(x, R_1^-(app(x_m, s))), ch(b(x_2, x_1), b(x_m, x_{m-1}))|e|\!\}, R_2^-(x)).\end{aligned}$$

Die beiden Resultatterme sehen nicht gleich aus, sind aber bezüglich $=_{\mathcal{E}}$ (der durch die Split-Axiome und das Cl-Axiom induzierten Kongruenzrelation) gleich. Dies ist offensichtlich, wenn man das *Split M*-Axiom mit den gleichen neuen Variablen, wie auch während der Unifikation anwendet:

$$\begin{aligned}&letrec(\{\!|b(x_1, v), b(x, R_1^-(app(x_m, s))), ch(b(x_2, x_1), b(x_m, x_{m-1}))|e|\!\}, R_2^-(x))\\ &\approx^{Split\ M} letrec(\{\!|b(x_1, w), b(x, R_1^-(app(x_m, s))),\\ &\qquad ch(b_1, b(z_2, z_1)), b(z_3, z_2), ch(b(z_4, z_3), b_2)|e|\!\}, R_2^-(x))\\ &\xrightarrow{\sigma} letrec(\{\!|b(x_1, v), b(x, R_1^-(app(x_m, s))),\\ &\qquad ch(b(x_2, x_1), b(y_n, y_{n-1})), b(y, y_n), ch(b(z_4, y), b(x_m, x_{m-1}))|e|\!\}, R_2^-(x))\end{aligned}$$

Berücksichtigt man die Vertauschbarkeit der Elemente in `letrec`-Umgebungen, sind die beiden substituierten Resultatterme gleich.

Zusammen mit den Regeln *Tautology, Orientation, Decomposition, Variable Elimination, Symbol Clash* und *Occurs Check* beschreibt die Regel *Lazy Paramodulation* eine vollständige Unifikationsprozedur in beliebigen (unsortierten) Gleichungstheorien (Gallier & Snyder, 1989). Allerdings kann über ihr Terminierungsverhalten

keine Aussage gemacht werden. Wir haben hier das Problem, dass die Anwendung eines Axioms nicht eingeschränkt ist und beliebig oft wiederholt werden kann, wenn eine *Decomposition*-Transformation mit dem durch die Anwendung des Axioms neu eingeführten Term möglich wird. Auf eine Variablenkette können demnach beliebig viele *Split A-, Split E-* oder *Split M*-Transformationen angewendet werden.

5.2.1 Unifikation von Bindungen aus Variablenketten mit Bindungen

Um die Terminierung der Unifikation von Termen mit Variablenketten zu gewährleisten, muss nach Möglichkeiten gesucht werden, die Anwendung der Split-Axiome zu beschränken. Dabei machen wir uns die Tatsache zunutze, dass Variablenketten in Termen, die für die Berechnung von Überlappungen unifiziert werden müssen, immer Bestandteile von `letrec`-Umgebungen sind. D.h. die zu unifizierenden Gleichungen zwischen `letrec`-Umgebungen sind immer von der Form

$$\{\!|b_1, \ldots, b_m, c_1, \ldots, c_n|e|\!\} =^? \{\!|b'_1, \ldots, b'_{m'}, c'_1, \ldots c'_{n'}|e'|\!\},$$

wobei $b_i, b'_{i'}$ Terme der Sorte B (d.h. `letrec`-Bindungen), $c_i, c'_{i'}$ Variablenketten und e, e' Umgebungsvariablen der Sorte U sind. Einzelne Bindungen der Variablenketten c_i (bzw. $c'_{i'}$), die durch eine Anwendung von Split-Axiomen abgespalten werden, können unifiziert werden mit

1. Bindungen $b'_{i'}$ (bzw. b_i) oder
2. mit einzelnen Bindungen einer Variablenkette $c'_{i'}$ (bzw. c_i) oder
3. mit der Umgebungsvariablen e' (bzw. e).

Die Anwendung von Split-Axiomen kann nicht beschränkt werden, wenn die beiden letzten Möglichkeiten der Unifikation einzelner Bindungen von Variablenketten erlaubt werden. Aus diesem Grund werden die Unifikationsmöglichkeiten von Bindungen aus Variablenketten eingeschränkt: Einzelne durch die Anwendung von Split-Axiomen abgespaltene Bindungen von Variablenketten werden nur mit Bindungen, der gegenüberliegenden Umgebung unifiziert, die nicht von Variablenketten abgespalten sind. Somit wird für alle Ketten c_i (bzw. $c'_{i'}$) maximal m'-mal (m-mal) ein Split-Axiom angewendet. Außerdem hat eine Variablenkette c_i die Möglichkeit, mit einer Kette $c'_{i'}$ oder der Umgebungsvariablen e' unifiziert zu werden (bzw. kann eine Kette $c'_{i'}$ mit der Umgebungsvariablen e unifiziert werden). Diesen Unifikationsmöglichkeiten wenden wir uns in Abschnitt 5.2.2 zu. Die Beschränkung der Anwendung von Split-Axiomen für Variablenketten kann durch die folgende Unifikationsregeln formuliert werden.

Definition 5.2.3. Wir betrachten die Signatur Σ^{let} mit dem Cl-Funktionssymbol $\{\!|\cdot|\cdot|\!\}$ und dem Funktionssymbol für Variablenketten ch, für das die Split-Axiome

aus Definition 5.1.1 gelten. Die Transformationsregeln zur Unifikation in regulären Signaturen mit Sorten (Definition 3.4.13) werden um die Regeln aus den Abbildungen 5.1 und 5.2 erweitert, um Terme mit Variablenketten zu unifizieren.

Sind mehrere Transformationen auf ein Unifikationsproblem anwendbar, wird eine beliebige ausgewählt.

Die Transformation *Split-R* entspricht der Anwendung eines Split-Axioms auf eine Variablenkette in einer `letrec`-Umgebung auf der rechten Seite einer Unifikationsgleichung. Die abgespaltene Bindung der Variablenkette wird direkt mit einer Bindung der gegenüberliegenden Umgebung unifiziert, die nicht aus einer Variablenkette abgespalten ist. Die Transformationsmöglichkeit *i*) der *Split-R*-Regel entspricht einer Anwendung des *Split A*-Axioms. Die Transformationsmöglichkeiten *ii*) und *iii*) repräsentieren Anwendungen des *Split E*- und des *Split M*-Axioms. Die *Split-L*-Regel repräsentiert analog die Möglichkeiten der Anwendung von Split-Axiomen auf Variablenketten, die in linken Seiten von Unifikationsgleichungen vorkommen. Damit, falls die Unifikation nicht fehlschlägt, die zu unifizierenden Terme syntaktisch gleich sind, muss man sich die Anwendung des entsprechenden Split-Axioms merken, zusammen mit den dadurch neu eingeführten Variablen (siehe dazu Beispiel 5.2.2). Die Split-Regeln markieren alle Variablenketten mit M, die durch Abspaltungen von Bindungen (*Split-X i*)[3]) oder Aufteilen der Kette (*Split-X iii*)) die Anfangsbindung der ursprünglichen Variablenkette nicht mehr enthalten. Diese Markierungen werden später verwendet, um die Unifikationsmöglichkeiten von Variablenketten untereinander zu beschränken (siehe Abschnitt 5.2.2).

Die Regel *U-Decomposition i*) wird nur auf Terme der Sorte B angewendet. Dadurch wird zum jetzigen Zeitpunkt vermieden, dass Variablenketten miteinander unifiziert werden. Die Regeln zur Unifikation von Variablenketten untereinander werden in Abschnitt 5.2.2 beschrieben. *U-Decomposition ii*) ist auch für Variablenketten anwendbar, wodurch die Vertauschbarkeit von Ketten in Umgebungen gewährleistet wird. Für die Anwendung der Regeln *U-Decomposition, Split-R* und *Split-L* muss gelten, dass die zu unifizierenden Umgebungen nicht die gleiche Kontextvariable X der Sorte U enthalten. Ansonsten wird die Regel *U-Decomposition∗* angewendet, die beide Umgebungsvariablen durch die leere Umgebung $\emptyset$ ersetzt. Die Regel *U-Fail* behandelt den Fall, dass die zu unifizierenden Umgebungen keine Umgebungsvariablen der Sorte U enthalten und eine verschiedene Anzahl von Elementen besitzen. In diesem Fall besitzt die Unifikationsgleichung keine Lösung.

Die Beschränkung der Anwendung der Split-Axiome führt dazu, dass einzelne Bindungen von Variablenketten nicht mit Bindungen von anderen Variablenketten unifiziert werden. Für eine Unifikationsgleichung der Form $\{|ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))|e|\} =^? \{|ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))|d|\}$, wobei e, d Variablen der Sorte U sind, ist es bei der Unifikation mit der *Lazy Paramod-*

[3] *Split-X* bezeichnet die beiden Regeln *Split-R* und *Split-L*.

Split – R

$\{\{\!|t|s|\!\}\} =^? \{\{\!|t_1, \ldots, ch(b(x_m, x_{m-1}), b(x_n, x_{n-1})), \ldots, t_k|s'|\!\}\} \uplus P \Rightarrow$

i) $\{t =^? b(x_m, x_{m-1}), s =^? \{\!|t_1, \ldots, ch(b(z, x_m), b(x_n, x_{n-1}))^M, \ldots, t_k|s'|\!\}\} \cup P$
wobei z eine neue Variable der Sorte V ist.

ii) $\{t =^? b(x_n, x_{n-1}), s =^? \{\!|t_1, \ldots, ch(b(x_m, x_{m-1}), b(x_{n-1}, z)), \ldots, t_k|s'|\!\}\} \cup P$
wobei z eine neue Variable der Sorte V ist.

iii) $\{t =^? b(z_3, z_2), s =^? \{\!|t_1, \ldots, ch(b(x_m, x_{m-1}), b(z_2, z_1)),$
$ch(b(z_4, z_3), b(x_n, x_{n-1}))^M, \ldots, t_k|s'|\!\}\} \cup P$
wobei z_1, z_2, z_3, z_4 neue Variablen der Sorte V sind.

Für alle Transformationen muss gelten: t ist ein Term der Sorte B und *tail*(s) sowie *tail*(s') bezeichnen nicht die gleiche Kontextvariable X der Sorte U.

Split – L

$\{\{\!|t_1, \ldots, ch(b(x_m, x_{m-1}), b(x_n, x_{n-1})), \ldots, t_k|s|\!\}\} =^? \{\{\!|t|s'|\!\}\} \uplus P \Rightarrow$

i) $\{t =^? b(x_m, x_{m-1}), s' =^? \{\!|t_1, \ldots, ch(b(z, x_m), b(x_n, x_{n-1}))^M, \ldots, t_k|s|\!\}\} \cup P$
wobei z eine neue Variable der Sorte V ist.

ii) $\{t =^? b(x_n, x_{n-1}), s' =^? \{\!|t_1, \ldots, ch(b(x_m, x_{m-1}), b(x_{n-1}, z)), \ldots, t_k|s|\!\}\} \cup P$
wobei z eine neue Variable der Sorte V ist.

iii) $\{t =^? b(z_3, z_2), s' =^? \{\!|t_1, \ldots, ch(b(x_m, x_{m-1}), b(z_2, z_1)),$
$ch(b(z_4, z_3), b(x_n, x_{n-1}))^M, \ldots, t_k|s|\!\}\} \cup P$
wobei z_1, z_2, z_3, z_4 neue Variablen der Sorte V sind.

Für alle Transformationen muss gelten: t ist ein Term der Sorte B und *tail*(s) sowie *tail*(s') bezeichnen nicht die gleiche Kontextvariable X der Sorte U.

U – Decomposition

$\{\{\!|t|s|\!\}\} =^? \{\!|t'|s'|\!\}\} \uplus P \Rightarrow$ *i)* $\{t =^? t', s =^? s'\} \cup P$
wenn t, t' Terme der Sorte B sind.

ii) $\{s =^? \{\!|t'|N|\!\}, \{\!|t|N|\!\}\} =^? s'\} \cup P$
wobei N eine neue Variable der Sorte U ist.

Für beide Transformationen muss gelten, dass *tail*(s) und *tail*(s') nicht die gleiche Kontextvariable X der Sorte U bezeichnen.

Figure 5.1. Unifikationsregeln für Terme mit `letrec`-Umgebungen und Variablenketten (Teil 1).

U – Decomposition$*$
$\{\{\!|t|s|\!\} =^? \{\!|t'|s'|\!\}\} \uplus P \Rightarrow \{\{\!|t|\widehat{s}|\!\} =^? \{\!|u|\widehat{s'}|\!\}\} \cup P$
wenn $tail(s) = tail(s') = X : U$.
$\widehat{s}, \widehat{s'}$ sind gleich s, s' mit $tail(s), tail(s')$ durch $\emptyset$ ersetzt.
U – Fail
$\{\{\!|t|s|\!\} =^? \{\!|t'|s'|\!\}\} \uplus P \Rightarrow \bot$
wenn $tail(s) = tail(s') = \emptyset$ und
s, s' nicht die gleiche Anzahl von Elementen besitzen.

Figure 5.2. Unifikationsregeln für Terme mit `letrec`-Umgebungen und Variablenketten (Teil 2).

ulation-Regel (aus Definition 5.2.1) möglich, beliebig viele Bindungen aus $ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))$ und $ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))$ durch Anwendung von Split-Axiomen abzuspalten und miteinander zu unifizieren. Die beschränkten *Split-R* und *Split-L* Regeln erlauben dies nicht mehr. Aus diesem Grund ist zu überlegen, wie Variablenketten mit anderen Variablenketten zu unifizieren sind. Eine einfache Anwendung von *U-Decomposition* und *Decomposition*, die für das Beispiel folgendermaßen aussieht

$$\begin{aligned} & \{\{\!|ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))|e|\!\} =^? \{\!|ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))|d|\!\}\} \\ \xrightarrow{\text{U-Dec } i)} & \{ch(b(x_m, x_{m-1}), b(x_n, x_{n-1})) =^? ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1})), e =^? d\} \\ \xrightarrow{Dec} & \{b(x_m, x_{m-1}) =^? b(y_{m'}, y_{m'-1}), b(x_n, x_{n-1}) =^? b(y_{n'}, y_{n'-1}), e =^? d\}, \end{aligned}$$

verletzt die Vollständigkeit der Unifikationsprozedur, weil die Möglichkeit besteht, dass nur einzelne Bindungen der beiden Variablenketten gleich sind und die restlichen unterschiedlichen Teile mit der Umgebungsvariablen e oder d der gegenüberliegenden Umgebung unifiziert werden. Die Regel *U-Decomposition i)* ist so formuliert, dass eine Anwendung auf zwei Variablenketten wie im obigen Beispiel ausgeschlossen ist. Um die Möglichkeiten der teilweisen Gleichheit von Variablenketten bei der Unifikation zu berücksichtigen, werden zusätzliche Unifikationsregeln benötigt. Bevor wir uns den Transformationsregeln zur Unifikation von Variablenketten mit Variablenketten zuwenden, betrachten wir die Form eines Unifikationsproblems zwischen Umgebungen nach einer Folge von Transformationen mit den Regeln aus Definition 5.2.3.

Die Terme, die zur Berechnung von Überlappungen unifiziert werden müssen, enthalten maximal eine Variablenkette pro Umgebung und genau eine Umgebungsvariable der Sorte U. Die Unifikationsgleichungen zwischen Umgebungen sind zu Beginn einer Folge von Transformationen von der Form

$$\{\{\!|b_1, \ldots, b_m, c|e|\!\} =^? \{\!|b'_1, \ldots, b'_{m'}, c'|e'|\!\}\}, \tag{5.2}$$

wobei $b_i, b'_{i'}$ Terme der Sorte B (`letrec`-Bindungen), c, c' Variablenketten (die in den Umgebungen nicht vorhanden sein müssen) und e, e' Variablen der Sorte U sind. Wir gehen hier von dem Fall aus, dass die Ketten c und c' in den Umgebungen enthalten sind. Durch eine (beliebige) Folge von Transformationen mit den Unifikationsregeln aus Definition 5.2.3 kann das Unifikationsproblem (5.2) in die Form:

$$\{\{|c_1, \ldots, c_n|d|\} =^? \{|c'_1, \ldots, c'_{n'}|d'|\}, s_1 =^? t_1, \ldots s_l =^? t_l\} \tag{5.3}$$

gebracht werden, wobei $c_i, c'_{i'}$ Variablenketten, die durch die Anwendung *Split-X iii)* aus c bzw. c' entstanden sind, und d, d' Variablen der Sorte U sind. Die Gleichungen $s_j =^? t_j$ sind Unifikationsgleichungen in semi-gelöster Form und enthalten die $b_i, b'_{i'}$ (siehe Definition 4.4.3). Wir geben an dieser Stelle keinen Beweis für die Behauptung, dass eine Unifikationsgleichung zwischen Umgebungen mit Variablenketten durch eine beliebige Folge von Transformationen mit Regeln aus Definition 5.2.3 in die Form (5.3) gebracht werden kann[4]. Stattdessen geben wir ein Beispiel, wie eine solche Folge von Transformationen für eine Unifikationsgleichung zwischen Umgebungen mit Variablenketten aussehen kann.

Beispiel 5.2.4. Wir betrachten ein Unifikationsproblem zwischen Umgebungen mit Variablenketten mit den Variablensorten $b_i : B$, $x_i, y_{i'}, z_i : V$ und $e, d : U$. Eine Folge von Transformationen, die das gewählte Unifikationsproblem in die Form (5.3) bringt, sieht folgendermaßen aus:

$$\begin{aligned}
&\{\{|b_1, b_2, ch(b(x_2, x_1), b(x_n, x_{n-1}))|e|\} =^? \{|b_3, ch(b(y_2, y_1), b(y_{n'}, y_{n'-1}))|d|\}\} \\
\xrightarrow{\textit{Split-R iii)}}\ &\{b_1 =^? b(z_3, z_2), \{|b_2, ch(b(x_2, x_1), b(x_n, x_{n-1}))|e|\} =^? \\
&\quad \{|b_3, ch(b(y_2, y_1), b(z_2, z_1)), ch(b(z_3, z_4), b(y_{n'}, y_{n'-1}))|d|\}\} \\
\xrightarrow{\textit{Split-L i)}}\ &\{b_1 =^? b(z_3, z_2), b_3 =^? b(x_2, x_1), \{|b_2, ch(b(z_5, x_2), b(x_n, x_{n-1}))|e|\} =^? \\
&\quad \{|ch(b(y_2, y_1), b(z_2, z_1)), ch(b(z_3, z_4), b(y_{n'}, y_{n'-1}))|d|\}\} \\
\xrightarrow{\textit{U-Dec ii)}}\ &\{b_1 =^? b(z_3, z_2), b_3 =^? b(x_2, x_1), \{|ch(b(z_5, x_2), b(x_n, x_{n-1}))|e|\} =^? \\
&\quad \{|ch(b(y_2, y_1), b(z_2, z_1))|N_1|\}, \{|b_2|N_1|\} =^? \{|ch(b(z_3, z_4), b(y_{n'}, y_{n'-1}))|d|\}\} \\
\xrightarrow{\textit{U-Dec ii)}}\ &\{b_1 =^? b(z_3, z_2), b_3 =^? b(x_2, x_1), \{|ch(b(z_5, x_2), b(x_n, x_{n-1}))|e|\} =^? \\
&\quad \{|ch(b(y_2, y_1), b(z_2, z_1))|N_1|\}, N_1 =^? \{|ch(b(z_3, z_4), b(y_{n'}, y_{n'-1}))|N_2|\}, \\
&\quad \{|b_2|N_2|\} =^? d\} \\
\xrightarrow{\textit{Elim } N_1}\ &\{b_1 =^? b(z_3, z_2), b_3 =^? b(x_2, x_1), \{|ch(b(z_5, x_2), b(x_n, x_{n-1}))|e|\} =^? \\
&\quad \{|ch(b(y_2, y_1), b(z_2, z_1)), ch(b(z_3, z_4), b(y_{n'}, y_{n'-1}))|N_2|\}, \\
&\quad N_1 =^? \{|ch(b(z_3, z_4), b(y_{n'}, y_{n'-1}))|N_2|\}, \{|b_2|N_2|\} =^? d\}
\end{aligned}$$

[4] Der Beweis ist ähnlich zum Beweis, dass eine Unifikationsgleichung zwischen Umgebungen in semi-gelöste Form gebracht werden kann (Lemma 4.4.4)

Nach einem *Orient*-Schritt für d entspricht das letzte Unifikationsproblem der Form (5.3).

Die `letrec`-Bindungen zweier zu unifizierender Umgebungen können also durch eine beliebige Folge von *U-Decomposition-, Split-R-, Split-L-, Variable Elimination-* und *Orientation*-Transformationen in Unifikationsgleichungen in semi-gelöster Form "'verschoben"' werden, so dass nur die Gleichungen zwischen Umgebungen nur noch von der Form $\{\!|c_1, \ldots, c_n|d|\!\} =^? \{\!|c'_1, \ldots, c'_{n'}|d'|\!\}$ sind. Wir wenden uns nun den Unifikationsregeln zur Lösung solcher Gleichungen zu.

5.2.2 Unifikation von Variablenketten mit Variablenketten

Wir betrachten zunächst den vereinfachten Fall, dass jede Umgebung nur eine Variablenkette enthält. Prinzipiell müssen zur Lösung einer Unifikationsgleichung der Form

$$\{\!|ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))|e|\!\} =^? \{\!|ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))|d|\!\}$$

alle Möglichkeiten betrachtet werden, wie einzelne Bindungen der beiden Ketten untereinander unifiziert werden können. Bindungen einer Kette, die dabei nicht mit Bindungen der anderen Kette unifiziert werden, können mit der Umgebungsvariablen der gegenüberliegenden Umgebung unifiziert werden. Im Folgenden sollen zwei Kriterien vorgestellt werden, die es ermöglichen, die Anzahl der zu betrachtenden Unifikationsmöglichkeiten bei der Unifikation von Variablenketten untereinander zu beschränken. Das erste Kriterium wird anhand des folgenden Beispiels eingeführt.

Beispiel 5.2.5. Wir betrachten ein Unifikationsproblem zwischen konkreten Variablenketten, d.h. Ketten die nicht durch das *ch*-Funktionssymbol dargestellt werden. Die Variablen x_i, y_i haben die Sorte V und e, d sind Variablen der Sorte U.

$$\{\!|b(x_2, x_1), b(x_3, x_2), b(x_4, x_3)|e|\!\} =^? \{\!|b(y_2, y_1), b(y_3, y_2), b(y_4, y_3)|d|\!\}$$

Unifiziere folgendermaßen (über Kreuz):

$$\begin{gathered}\{b(x_2, x_1) =^? b(y_3, y_2), b(x_3, x_2) =^? b(y_2, y_1),\\ e =^? \{\!|b(y_4, y_3)|N|\!\}, d =^? \{\!|b(x_4, x_3)|N|\!\}\}\end{gathered}$$

Die Anwendung von *Decomposition-* und *Variable Elimination*-Transformationen resultiert in der Substitution σ:

$$\sigma = \{x_1 \mapsto y_2, x_2 \mapsto y_1, x_3 \mapsto y_2, y_3 \mapsto y_1, e \mapsto \{\!|b(y_4, y_1)|N|\!\}, d \mapsto \{\!|b(x_4, y_2)|N|\!\}\}$$

Die Anwendung der Substitution σ auf einen der beiden Ausgangsterme

$$\sigma(\{\!|b(x_2, x_1), b(x_3, x_2), b(x_4, x_3)|e|\!\}) = \{\!|b(y_1, y_2), b(y_2, y_1), b(x_4, y_2), b(y_4, y_1)|N|\!\}$$

ergibt eine Umgebung, mit einer Schleife in der Variablenkette, die dazu führt, dass eine Redex-Suche, die über die Variablenketten läuft, für diesen Term nicht terminiert: Wir betrachten den Resultatterm in Λ^{let}[5]

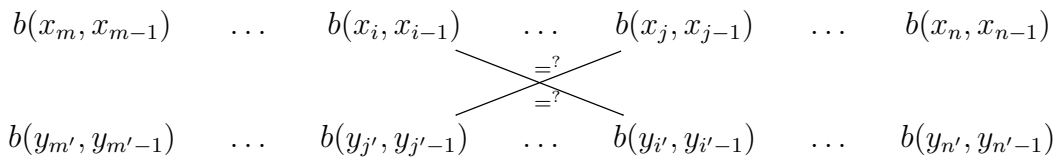
$$[\![\{\!|b(y_1,y_2),b(y_2,y_1),b(x_4,y_2),b(y_4,y_1)|N|\!\}]\!]^- = \{y_1=y_2,y_2=y_1,x_4=y_2,y_4=y_1|N\}$$

und nehmen an, dass der Unwind Algorithmus zur Redex-Suche (aus Definition 2.2.5) die rechte Seite einer Bindung markiert hat:

$$\begin{array}{rl} & \{y_1=y_2,y_2=y_1,x_4=y_2,y_4=y_1^S|N\} \\ \xrightarrow{Unwind} & \{y_1=y_2^S,y_2=y_1,x_4=y_2,y_4=y_1^W|N\} \\ \xrightarrow{Unwind} & \{y_1=y_2^W,y_2=y_1^S,x_4=y_2,y_4=y_1^W|N\} \end{array}$$

In der letzten Zeile versucht Unwind die rechte Seite der Bindung $y_1 = y_2^W$ zu markieren, die bereits mit W markiert ist. Dabei bricht der Algorithmus mit einem Fehler ab.

Generell kommt es zur Schleifenbildung innerhalb einer Variablenkette, die aus der Unifikation zweier Variablenketten $ch(b(x_m,x_{m-1}),b(x_n,x_{n-1}))$ und $ch(b(y_{m'},y_{m'-1}),b(y_{n'},y_{n'-1}))$ resultiert, wenn (mindestens) vier Bindungen folgendermaßen unifiziert werden: $b(x_i,x_{i-1}) =^? b(y_{i'},y_{i'-1})$ und $b(x_j,x_{j-1}) =^? b(y_{j'},y_{j'-1})$ für $i<j, j'<i'$, wie in folgender Abbildung zu sehen:



Diese Art der Unifikation von Variablenkettenbindungen untereinander bezeichnen wir als Kreuzunifikation. Bei der Unifikation von Umgebungen mit Variablenketten werden alle Fälle ausgeschlossen, bei denen einzelne Bindungen der Ketten miteinander kreuz-unifiziert werden.

Das zweite Kriterium, das verwendet wird, um bestimmte Unifikationsmöglichkeiten von Kettenbindungen untereinander auszuschließen, benutzt die Tatsache, dass Variablenketten Bestandteile von `letrec`-Umgebungen sind. D.h. für sie muss nach der Syntaxdefinition von Λ^{let} gelten, dass alle in Kettenbindungen auf der linken Seite vorkommenden Variablen sowie alle anderen in der Umgebung auf linken Seiten von Bindungen vorkommenden Variablen paarweise disjunkt sind. Die Forderung, dass alle Binder einer `letrec`-Umgebung verschiedene Namen haben, wird bei der

[5] Die Abbildung $[\![\ ]\!]^-$ übersetzt einen Term aus Σ^{let} nach Λ^{let}.

Syntaxdefinition des Ursprungskalküls aufgestellt. Sie stellt sicher, dass von der Normalordnung ein eindeutiger Redex gefunden werden kann. Die Unifikation von Umgebungen mit Variablenketten kann diese Bedingung für zu unifizierende Termen verletzen, da sie als Ziel hat, Terme syntaktisch gleich zu machen, und nicht auf syntaktische Nebenbedingungen des Kalküls achtet. Terme, die nach der Unifikation syntaktisch gleich sind, aber nicht der Disjunktheitsanforderung der Variablen in `letrec`-Umgebungen genügen, sind nicht von Interesse, da sie i.A. keinen eindeutigen Normalordnungsredex besitzen.

Definition 5.2.6. Sei

$$\begin{aligned}&\{\!|b(x_1,s_1),\ldots,b(x_l,s_l),\\&\ ch(b(y_{m_1},y_{(m_1-1)}),b(y_{n_1},y_{(n_1-1)})),\ldots,ch(b(y_{m_k},y_{(m_k-1)}),b(y_{n_k},y_{(n_k-1)}))|e|\!\}\end{aligned}$$

eine `letrec`-Umgebung, wobei e eine Variable der Sorte U oder $\emptyset$ ist. Die Umgebung wird als *syntaktisch korrekt* bezeichnet, wenn alle x_i, y_{m_j}, y_{n_j} für $1 \leq i \leq l, 1 \leq j \leq k$ paarweise disjunkt sind. Ist eine `letrec`-Umgebung nicht syntaktisch korrekt, sprechen wir davon, dass sie einen *syntaktischen Fehler* enthält.

Lemma 5.2.7. Es seien zwei Umgebungen $\{\!|ch(b(x_m,x_{m-1}),b(x_n,x_{n-1}))|e|\!\}$ und $\{\!|ch(b(y_{m'},y_{m'-1}),b(y_{n'},y_{n'-1}))|d|\!\}$, wobei e,d Variablen der Sorte U sind, sowie beliebige Bindungen der Variablenketten $b(x_i,x_{i-1}) \in ch(b(x_{m+1},x_m),b(x_n,x_{n-1}))$ und $b(y_{i'},y_{i'-1}) \in ch(b(y_{m'+1},y_{m'}),b(y_{n'},y_{n'-1}))$ ungleich der Anfangsbindung der jeweiligen Kette gegeben. Sei außerdem

$$\begin{aligned}\sigma = \{\ & x_i \mapsto y_{i'}, x_{i-1} \mapsto y_{i'-1},\\ & d \mapsto \{\!|ch(b(x_m,x_{m-1}),b(x_{i-1},x_{i-2})),ch(b(x_{i+1},x_i),b(x_n,x_{n-1}))|N|\!\},\\ & e \mapsto \{\!|ch(b(y_{m'},y_{m'-1},b(y_{i'-1},y_{i'-2})),ch(b(y_{i'+1},y_{i'}),b(y_{n'},y_{n'-1}))|N|\!\}\ \}\end{aligned}$$

die Substitution (wobei N eine neue Variablen der Sorte U ist), die durch die Unifikation von

$$\begin{aligned}&\{b(x_i,x_{i-1}) =^? b(y_{i'},y_{i'-1}),\\ &\ \{\!|ch(b(x_m,x_{m-1}),b(x_{i-1},x_{i-2})),ch(b(x_{i+1},x_i),b(x_n,x_{n-1}))|e|\!\} =^?\\ &\ \{\!|ch(b(y_{m'},y_{m'-1},b(y_{i'-1},y_{i'-2})),ch(b(y_{i'+1},y_{i'}),b(y_{n'},y_{n'-1}))|d|\!\}\}\end{aligned}$$

entsteht. Dann sind $\sigma(\{\!|ch(b(x_m,x_{m-1}),b(x_n,x_{n-1}))|e|\!\})$ und $\sigma(\{\!|ch(b(y_{m'},y_{m'-1}),b(y_{n'},y_{n'-1}))|d|\!\})$ keine syntaktisch korrekten `letrec`-Umgebungen.

Beweis. Durch Anwenden der Substitution σ:

$$
\begin{aligned}
& \sigma(\{\!|ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))|d|\!\}) \\
= \; & \{\!|\sigma(ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1})))|\sigma(d)|\!\}) \\
= \; & \{\!|\sigma(ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))), \\
& \quad \sigma(ch(b(x_m, x_{m-1}), b(x_{i-1}, x_{i-2}))), \sigma(ch(b(x_{i+1}, x_i), b(x_n, x_{n-1})))|N|\!\} \\
= \; & \{\!|ch(\sigma(b(y_{m'}, y_{m'-1})), \sigma(b(y_{n'}, y_{n'-1}))), \\
& \quad ch(\sigma(b(x_m, x_{m-1})), \sigma(b(x_{i-1}, x_{i-2}))), ch(\sigma(b(x_{i+1}, x_i)), \sigma(b(x_n, x_{n-1})))|N|\!\} \\
= \; & \{\!|ch(b(\sigma(y_{m'}), \sigma(y_{m'-1})), b(\sigma(y_{n'}), \sigma(y_{n'-1}))), \\
& \quad ch(b(\sigma(x_m), \sigma(x_{m-1})), b(\sigma(x_{i-1}), \sigma(x_{i-2}))), \\
& \quad ch(b(\sigma(x_{i+1}), \sigma(x_i)), b(\sigma(x_n), \sigma(x_{n-1})))|N|\!\} \\
= \; & \{\!|ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1})), \\
& \quad ch(b(x_m, x_{m-1}), \underline{b(y_{i'-1}, x_{i-2})}), ch(b(x_{i+1}, x_i), b(x_n, x_{n-1}))|N|\!\}
\end{aligned}
$$

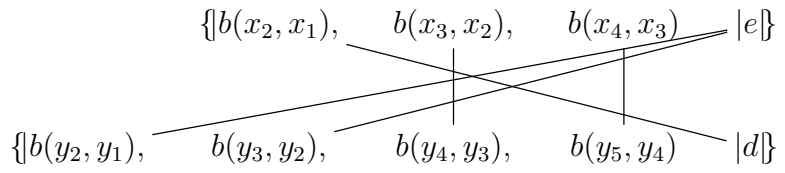
Im resultierenden Term in der letzten Zeile sind die Bindungen $b(y_{i'-1}, x_{i-2})$ und $b(y_{i'-1}, y_{i'-2})$ (in der Variablenkette $ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))$[6]) enthalten. D.h. in der `letrec`-Umgebung kommt zweimal die Variable $y_{i'-1}$ vor und deshalb ist die Umgebung nicht syntaktisch korrekt. □

Das Lemma zeigt, dass bestimmte Arten der Unifikation von Variablenkettenbindungen untereinander in Unifikatoren resultieren, die die Variablenketten zwar syntaktisch gleich machen, aber gleichzeitig einen syntaktischen Fehler in die Umgebungen einführen, in denen die Variablenketten enthalten sind. Das folgende Beispiel illustriert weitere Fälle, durch die syntaktische Fehler bei der Unifikation eingeführt werden können.

Beispiel 5.2.8. Wir betrachten ein Unifikationsproblem zwischen `letrec`-Umgebungen mit konkreten Variablenketten

$$\{\!|b(x_2, x_1), b(x_3, x_2), b(x_4, x_3)|e|\!\} =^? \{\!|b(y_2, y_1), b(y_3, y_2), b(y_4, y_3), b(y_5, y_4)|d|\!\}$$

mit den Variablensorten $x_i, y_i : V$ und $e, d : U$. Die Bindungen der Variablenketten werden nach folgendem Schema unifiziert:



[6] Die Bindung $b(y_{i'-1}, y_{i'-2})$ ist im resultierenden Term nicht unmittelbar sichtbar. Sie wird dies aber, wenn die Aufspaltung der Variablenkette $ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))$ zu $ch(b(y_{m'}, y_{m'-1}, b(y_{i'-1}, y_{i'-2})), b(y_{i'}, y_{i'-1}), ch(b(y_{i'+1}, y_{i'}), b(y_{n'}, y_{n'-1}))$ nachvollzogen wird.

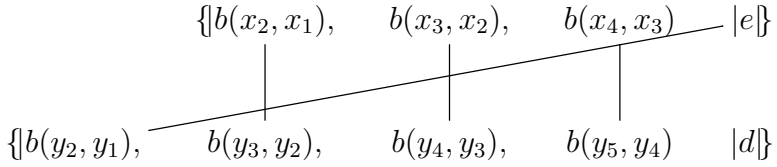
Daraus ergibt sich die Substitution

$$\sigma = \{x_2 \mapsto y_3, x_3 \mapsto y_4, x_4 \mapsto y_5, e \mapsto \{\!|b(y_2, y_1), b(y_3, y_2)|N|\!\}, d \mapsto \{\!|b(y_3, x_1)|N|\!\}\}.$$

Die Anwendung dieser Substitution auf einen der Ausgangsterme resultiert in einer Umgebung, die nicht syntaktisch korrekt ist:

$$\begin{aligned} &\sigma(\{\!|b(x_2, x_1), b(x_3, x_2), b(x_4, x_3)|e|\!\}) \\ = \ &\{\!|\underline{b(y_3, x_1)}, b(y_4, y_3), b(y_5, y_4), b(y_2, y_1), \underline{b(y_3, y_2)}|N|\!\} \end{aligned}$$

Die Einführung eines syntaktischen Fehlers durch die Substitution kann vermieden werden, indem mindestens eine Anfangsbindung einer der beiden Variablenketten unifiziert wird. Beispielsweise ergibt die Unifikation nach dem Schema



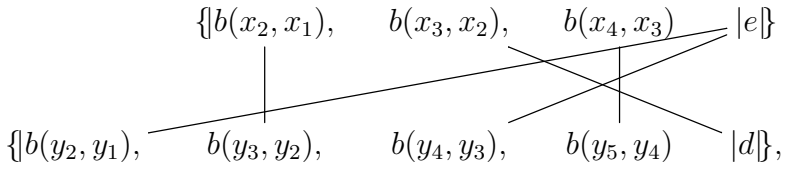
die Substitution

$$\sigma = \{x_1 \mapsto y_2, x_2 \mapsto y_3, x_3 \mapsto y_4, x_4 \mapsto y_5, e \mapsto \{\!|b(y_2, y_1)|N|\!\}\},$$

die keinen syntaktischen Fehler in die substituierte `letrec`-Umgebung einführt:

$$\begin{aligned} &\sigma(\{\!|b(x_2, x_1), b(x_3, x_2), b(x_4, x_3)|e|\!\}) \\ = \ &\{\!|b(y_3, y_2), b(y_4, y_3), b(y_5, y_4), b(y_2, y_1)|N|\!\}. \end{aligned}$$

Um die Einführung eines syntaktischen Fehlers durch einen Unifikator zu vermeiden, müssen die Bindungen einer Variablenkette außerdem durchgehend linearen unifiziert werden. Beispielsweise resultiert die Unifikation nach dem Schema



in der Substitution

$$\begin{aligned} \sigma = \{&x_1 \mapsto y_2, x_2 \mapsto y_3, x_3 \mapsto y_4, x_4 \mapsto y_5, \\ &e \mapsto \{\!|b(y_2, y_1), b(y_4, y_3)|N|\!\}, d \mapsto \{\!|b(y_4, y_3)|N|\!\}\}. \end{aligned}$$

die einen syntaktischen Fehler in die Umgebung einführt

$$\begin{aligned} &\sigma(\{\!|b(x_2, x_1), b(x_3, x_2), b(x_4, x_3)|e|\!\}) \\ = \ &\{\!|b(y_3, y_2), \underline{b(y_4, y_3)}, b(y_5, y_4), b(y_2, y_1), \underline{b(y_4, y_3)}|N|\!\}. \end{aligned}$$

Folgende Bedingungen lassen sich aus dem Verbot der Kreuzunifikation (Beispiel 5.2.5) Lemma 5.2.7 und Beispiel 5.2.8 für die Unifikation von Umgebungen mit Variablenketten ableiten: Es sollen zwei Umgebungen $\{\!|ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))|e|\!\}$ und $\{\!|ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))|d|\!\}$, die jeweils eine Variablenkette enthalten, unifiziert werden. Es gilt entweder:

1. Beide Variablenketten werden jeweils mit den gegenüberliegenden Umgebungsvariablen e bzw. d unifiziert.
2. Oder: Wird eine Kettenbindung $b(x_i, x_{i-1}) \in ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))$ mit einer anderen Kettenbindung $b(y_{i'}, y_{i'-1}) \in ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))$ unifiziert, so müssen die jeweiligen Vorgängerbindungen (falls diese existieren) ebenfalls miteinander unifiziert werden:

   $$b(x_i, x_{i-1}) =^? b(y_{i'}, y_{i'-1}) \Rightarrow b(x_{i-1}, x_{i-2}) =^? b(y_{i'-1}, y_{i'-2}).$$

   Dies geschieht solange, bis die Anfangsbindung einer der beiden Ketten erreicht ist. D.h. mindestens eine Anfangsbindung ist in diesem Fall an der Unifikation der Variablenkettenbindungen beteiligt, wie in folgender Abbildung schematisch zu sehen ist:

   $$\begin{array}{ccccccc} & & b(x_m, x_{m-1}) & \dots & b(x_i, x_{i-1}) & \dots \\ & & \overset{|}{=}^? & \overset{|}{=}^? & \overset{|}{=}^? & \\ b(y_{m'}, y_{m'-1}) & \dots & b(y_{j'}, y_{j'-1}) & \dots & b(y_{i'}, y_{i'-1}) & \dots \end{array}$$

   bzw.

   $$\begin{array}{ccccccc} b(x_m, x_{m-1}) & \dots & b(x_j, x_{j-1}) & \dots & b(x_i, x_{i-1}) & \dots \\ & & \overset{|}{=}^? & \overset{|}{=}^? & \overset{|}{=}^? & \\ & & b(y_{m'}, y_{m'-1}) & \dots & b(y_{i'}, y_{i'-1}) & \dots \end{array}$$

Werden die beiden Variablenketten der Umgebungen so unifiziert, dass keine der beiden Bedingungen gilt, ist das Resultat ein Unifikator σ, der einen syntaktischen Fehler in die beiden Umgebungen einführt.

Aus diesen Überlegungen ergeben sich 13 Möglichkeiten, zwei Umgebungen $\{\!|ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))|e|\!\}$ und $\{\!|ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))|d|\!\}$ mit Variablenketten, die keine weiteren Bindungen enthalten, zu unifizieren. Dazu werden die Variablenketten in Teilketten zerlegt.

Definition 5.2.9. Wir erweitern die Unifikationsregeln aus Definition 5.2.3 um die Regeln aus Abbildung 5.3 und Abbildung 5.4 zur Unifikation von Gleichungen der Form

$$\{\!|ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))|e|\!\} =^? \{\!|ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))|d|\!\},$$

wobei e, d Variablen der Sorte U sind.

Die Aufteilung der Ketten bei der Anwendung einer bestimmten Transformationsmöglichkeit, muss man sich zusammen mit den verwendeten neuen Variablen merken, damit die substituierten Terme syntaktisch gleich sind (siehe dazu auch Definition 5.2.3 sowie die Beispiele 5.2.2 und 5.2.11).

Unify – CH – KMB

$\{\{\!|ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))|e|\!\} =^? \{\!|ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))|d|\!\}\} \uplus P \Rightarrow$

i) $\{ch(b(x_m, x_{m-1}), b(x_n, x_{n-1})) =^? ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1})), e =^? d\} \cup P$

ii) $\{d =^? \{\!|ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))|N|\!\},$
$e =^? \{\!|ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))|N|\!\}\} \cup P$

iii) $\{ch(b(x_m, x_{m-1}), b(z_2, z_1)) =^? ch(b(y_{m'}, y_{m'-1}), b(z'_2, z'_1)),$
$d =^? \{\!|ch(b(z_3, z_2), b(x_n, x_{n-1}))|N|\!\},$
$e =^? \{\!|ch(b(z'_3, z'_2), b(y_{n'}, y_{n'-1}))|N|\!\}\} \cup P$

iv) $\{ch(b(x_m, x_{m-1}), b(z_2, z_1)) =^? ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1})),$
$d =^? \{\!|ch(b(z_3, z_2), b(x_n, x_{n-1}))|e|\!\}\} \cup P$

v) $\{ch(b(x_m, x_{m-1}), b(x_n, x_{n-1})) =^? ch(b(y_{m'}, y_{m'-1}), b(z'_2, z'_1)),$
$e =^? \{\!|ch(b(z'_3, z'_2), b(y_{n'}, y_{n'-1}))|d|\!\}\} \cup P$

Für alle Transformationsmöglichkeiten muss gelten:
Nicht beide Variablenketten sind mit M markiert.
$z_i, z'_i, 1 \leq i \leq 3$ sind jeweils neue Variablen der Sorte V
und N ist eine neue Variable der Sorte U.

Figure 5.3. Unifikationsregeln für `letrec`-Umgebungen mit jeweils eine Variablenkette pro Umgebung (Teil 1).

Die Markierung M wird verwendet, um bestimmte Unifikationsmöglichkeiten auszuschließen, durch die Unifikatoren berechnet werden, die syntaktische Fehler in die zu unifizierenden Umgebungen einführen. Da nach unseren Überlegungen immer die Anfangsbindung einer der beiden Variablenketten an der Unifikation beteiligt sein muss, werden die Fälle von der Unifikation ausgeschlossen, bei denen mit der Anfangsbindung einer Kette unifiziert wird, die mit M markiert ist, d.h. diese Kette enthält nicht mehr die Anfangsbindung der ursprünglichen Ketten. Weshalb es in diesen Fällen zu der Einführung eines syntaktischen Fehlers durch die berechnete Unifikation kommt, ist in Beispiel 5.2.11 erläutert.

Die Transformationsmöglichkeiten für Unifikationsgleichungen der Form $\{\!|ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))|e|\!\} =^? \{\!|ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))|d|\!\}$ aus Definition 5.2.9 können durch Diagramme veranschaulicht werden. Dabei entspricht

Unify − CH − KMR

$\{\{\!|ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))|e|\!\} =^? \{\!|ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))|d|\!\}\} \uplus P \Rightarrow$

i) $\{ch(b(z_3, z_2), b(x_n, x_{n-1})) =^? ch(b(y_{m'}, y_{m'-1}), b(z'_2, z'_1)),$
$d =^? \{\!|ch(b(x_m, x_{m-1}), b(z_2, z_1))|N|\!\},$
$e =^? \{\!|ch(b(z'_3, z'_2), b(y_{m'}, y_{m'-1}))|N|\!\}\} \cup P$

ii) $\{ch(b(z_3, z_2), b(z_5, z_4)) =^? ch(b(y_{m'}, y_{m'-1}), b(z'_2, z'_1)),$
$d =^? \{\!|ch(b(x_m, x_{m-1}), b(z_2, z_1)), ch(b(z_6, z_5), b(x_n, x_{n-1}))|N|\!\},$
$e =^? \{\!|ch(b(z'_3, z'_2), b(y_{n'}, y_{n'-1}))|N|\!\}\} \cup P$

iii) $\{ch(b(z_3, z_2), b(z_5, z_4)) =^? ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1})),$
$d =^? \{\!|ch(b(x_m, x_{m-1}), b(z_2, z_1)), ch(b(z_6, z_5), b(x_n, x_{n-1}))|e|\!\}\} \cup P$

iv) $\{ch(b(z_3, z_2), b(x_n, x_{n-1})) =^? ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1})),$
$d =^? \{\!|ch(b(x_m, x_{m-1}), b(z_2, z_1))|e|\!\}\} \cup P$

Für alle Transformationsmöglichkeiten muss gelten:
Die Variablenkette $ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))$ ist nicht mit M markiert.
$z_i, z'_i, 1 \leq i \leq 6$ sind jeweils neue Variablen der Sorte V
und N ist eine neue Variable der Sorte U.

Unify − CH − KML

$\{\{\!|ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))|e|\!\} =^? \{\!|ch(b(y_{m'}, y_{m'-1}), b(y_{n'}, y_{n'-1}))|d|\!\}\} \uplus P \Rightarrow$

i) $\{ch(b(x_m, x_{m-1}), b(z_2, z_1)) =^? ch(b(z'_3, z'_2), b(y_{n'}, y_{n'-1})),$
$d =^? \{\!|ch(b(z_3, z_2), b(x_n, x_{n-1}))|N|\!\},$
$e =^? \{\!|ch(b(y_{m'}, y_{m'-1}), b(z'_2, z'_1))|N|\!\}\} \cup P$

ii) $\{ch(b(x_m, x_{m-1}), b(z_2, z_1)) =^? ch(b(z'_3, z'_2), b(z'_5, z'_4)),$
$d =^? \{\!|ch(b(z_3, z_2), b(x_n, x_{n-1}))|N|\!\},$
$e =^? \{\!|ch(b(y_{m'}, y_{m'-1}), b(z'_2, z'_1)), ch(b(z'_6, z'_5), b(y_{n'}, y_{n'-1}))|N|\!\}\} \cup P$

iii) $\{ch(b(x_m, x_{m-1}), b(x_n, x_{n-1})) =^? ch(b(z'_3, z'_2), b(z'_5, z'_4)),$
$e =^? \{\!|ch(b(y_{m'}, y_{m'-1}), b(z'_2, z'_1)), ch(b(z'_6, z'_5), b(y_{n'}, y_{n'-1}))|d|\!\}\} \cup P$

iv) $\{ch(b(x_m, x_{m-1}), b(x_n, x_{n-1})) =^? ch(b(z'_3, z'_2), b(y_{n'}, y_{n'-1})),$
$e =^? \{\!|ch(b(y_{m'}, y_{m'-1}), b(z'_2, z'_1))|d|\!\}\} \cup P$

Für alle Transformationsmöglichkeiten muss gelten:
Die Variablenkette $ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))$ ist nicht mit M markiert.
$z_i, z'_i, 1 \leq i \leq 6$ sind jeweils neue Variablen der Sorte V
und N ist eine neue Variable der Sorte U.

Figure 5.4. Unifikationsregeln für `letrec`-Umgebungen mit jeweils einer Variablenkette pro Umgebung (Teil 2).

die Abbildung

$$b(x_m, x_{m-1}) \vdash b(z_2, z_1) \quad b(z_3, z_2) - b(z_5, z_4) \quad b(z_6, z_5) \twoheadrightarrow b(x_n, x_{n-1})$$

der Variablenkette $ch(b(x_m, x_{m-1}), b(x_n, x_{n-1}))$, die in 3 Teilketten zerlegt wurde: Das Anfangsstück der Kette ist $ch(b(x_m, x_{m-1}), b(z_2, z_1))$, das in der Abbildung durch den Pfeil $\vdash$ zwischen Anfangs- und Endbindung der Teilkette gekennzeichnet ist. Das Mittelstück der Kette ist $ch(b(z_3, z_2), b(z_5, z_4))$. Die Anfangs- und Endbindung dieser Teilkette sind durch einen einfachen Strich $-$ verbunden. Das Endstück der Kette ist $ch(b(z_6, z_5), b(x_n, x_{n-1}))$. Hier sind Anfangs- und Endbindung durch den Pfeil $\twoheadrightarrow$ verbunden. Das Diagramm

$$\begin{array}{lll} b(x_m, x_{m-1}) \vdash b(z_2, z_1) & b(z_3, z_2) - b(z_5, z_4) & b(z_6, z_5) \twoheadrightarrow b(x_n, x_{n-1}) \\ & \mid_{=}^{?} \qquad\qquad \mid_{=}^{?} & \\ & b(y_{m'}, y_{m'-1}) \vdash b(z'_2, z'_1) & b(z'_3, z'_2) \twoheadrightarrow d(y_{n'}, y_{n'-1}) \end{array}$$

symbolisiert dann die Transformationsmöglichkeiten *vi)* aus Definition 5.2.9, bei der die Teilketten $ch(b(z_3, z_2), b(z_5, z_4))$ und $ch(b(y_{m'}, y_{m'-1}), b(z'_2, z'_1))$ miteinander und die restlichen Teilketten mit der Umgebungsvariablen der gegenüberliegenden Umgebung unifiziert werden. Wir geben die entsprechenden Diagramme für die einzelnen Transformationsmöglichkeiten der Regel aus Definition 5.2.9 an.

Unify-CH-KMB Die Diagramme, die die einzelnen Transformationsmöglichkeiten der Regel *Unify-CH-KMB* beschreiben, sind:

i)
$$\begin{array}{ll} b(x_m, x_{m-1}) \longmapsto b(x_n, x_{n-1}) \\ \mid_{=}^{?} \qquad \mid_{=}^{?} \\ b(y_{m'}, y_{m'-1}) \longmapsto b(y_{n'}, y_{n'-1}) \end{array}$$

ii)
$$\begin{array}{ll} b(x_m, x_{m-1}) \longmapsto b(x_n, x_{n-1}) & \\ & b(y_{m'}, y_{m'-1}) \longmapsto b(y_{n'}, y_{n'-1}) \end{array}$$

iii)
$$\begin{array}{lll} b(x_m, x_{m-1}) \vdash b(z_2, z_1) & b(z_3, z_2) \twoheadrightarrow b(x_n, x_{n-1}) \\ \mid_{=}^{?} \qquad \mid_{=}^{?} & \\ b(y_{m'}, y_{m'-1}) \vdash b(z'_2, z'_1) & b(z'_3, z'_2) \twoheadrightarrow b(y_{n'}, y_{n'-1}) \end{array}$$

iv)
$$\begin{array}{lll} b(x_m, x_{m-1}) \longmapsto b(z_2, z_1) & b(z_3, z_2) \twoheadrightarrow b(x_n, x_{n-1}) \\ \mid_{=}^{?} \qquad \mid_{=}^{?} & \\ b(y_{m'}, y_{m'-1}) \mapsto b(y_{n'}, y_{n'-1}) & \end{array}$$

v)
$$\begin{array}{lll} b(x_m, x_{m-1}) \mapsto b(x_n, x_{n-1}) & \\ \mid_{=}^{?} \qquad \mid_{=}^{?} & \\ b(y_{m'}, y_{m'-1}) \longmapsto b(z'_2, z'_1) & b(z'_3, z'_2) \twoheadrightarrow b(y_{n'}, y_{n'-1}) \end{array}$$

Unify-CH-KMR Die Diagramme, die die einzelnen Transformationsmöglichkeiten der Regel *Unify-CH-KMR* beschreiben, sind:

i)
$$\begin{array}{lccccl}
b(x_m, x_{m-1}) \vdash\!\!- b(z_2, z_1) & b(z_3, z_2) & \longrightarrow & b(x_n, x_{n-1}) & & \\
& \Vert^{?} & & \Vert^{?} & & \\
& b(y_{m'}, y_{m'-1}) & \vdash\!\!\relbar\relbar & b(z'_2, z'_1) & & b(z'_3, z'_2) \to b(y_{n'}, y_{n'-1})
\end{array}$$

ii)
$$\begin{array}{lccccl}
b(x_m, x_{m-1}) \vdash\!\!- b(z_2, z_1) & b(z_3, z_2) & \relbar\relbar & b(z_5, z_4) & & b(z_6, z_5) \to b(x_n, x_{n-1}) \\
& \Vert^{?} & & \Vert^{?} & & \\
& b(y_{m'}, y_{m'-1}) & \vdash\!\!- & b(z'_2, z'_1) & & b(z'_3, z'_2) \to b(y_{n'}, y_{n'-1})
\end{array}$$

iii)
$$\begin{array}{lccccl}
b(x_m, x_{m-1}) \vdash\!\!- b(z_2, z_1) & b(z_3, z_2) & \relbar\relbar\relbar & b(z_5, z_4) & & b(z_6, z_5) \to b(x_n, x_{n-1}) \\
& \Vert^{?} & & \Vert^{?} & & \\
& b(y_{m'}, y_{m'-1}) & \mapsto & b(y_{n'}, y_{n'-1}) & &
\end{array}$$

iv)
$$\begin{array}{lccc}
b(x_m, x_{m-1}) \vdash\!\!- b(z_2, z_1) & b(z_3, z_2) & \longrightarrow & b(x_n, x_{n-1}) \\
& \Vert^{?} & & \Vert^{?} \\
& b(y_{m'}, y_{m'-1}) & \mapsto & b(y_{n'}, y_{n'-1})
\end{array}$$

Unify-CH-KML Die Diagramme, die die einzelnen Transformationsmöglichkeiten der Regel *Unify-CH-KML* beschreiben, sind:

i)
$$\begin{array}{lccccl}
& b(x_m, x_{m-1}) & \relbar\relbar & b(z_2, z_1) & & b(z_3, z_2) \to b(x_n, x_{n-1}) \\
& \Vert^{?} & & \Vert^{?} & & \\
b(y_{m'}, y_{m'-1}) \vdash\!\!- b(z'_2, z'_1) & b(z'_3, z'_2) & \longrightarrow & b(y_{n'}, y_{n'-1}) & &
\end{array}$$

ii)
$$\begin{array}{lccccl}
& b(x_m, x_{m-1}) & \vdash\!\!- & b(z_2, z_1) & & b(z_3, z_2) \to b(x_n, x_{n-1}) \\
& \Vert^{?} & & \Vert^{?} & & \\
b(y_{m'}, y_{m'-1}) \vdash\!\!- b(z'_2, z'_1) & b(z'_3, z'_2) & \relbar\relbar & b(z'_5, z'_4) & & b(z'_6, z'_5) \to b(y_{n'}, y_{n'-1})
\end{array}$$

iii)
$$\begin{array}{lccccl}
& b(x_m, x_{m-1}) & \mapsto & b(x_n, x_{n-1}) & & \\
& \Vert^{?} & & \Vert^{?} & & \\
b(y_{m'}, y_{m'-1}) \vdash\!\!- b(z'_2, z'_1) & b(z'_3, z'_2) & \relbar\relbar\relbar & b(z'_5, z'_4) & & b(z'_6, z'_5) \to b(y_{n'}, y_{n'-1})
\end{array}$$

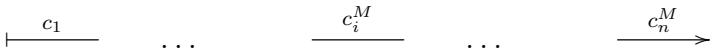
iv)
$$\begin{array}{lccc}
& b(x_m, x_{m-1}) & \mapsto & b(x_n, x_{n-1}) \\
& \Vert^{?} & & \Vert^{?} \\
b(y_{m'}, y_{m'-1}) \vdash\!\!- b(z'_2, z'_1) & b(z'_3, z'_2) & \longmapsto & b(y_{n'}, y_{n'-1})
\end{array}$$

Wir benötigen noch Regeln, um Gleichungen zwischen Umgebungen zu unifizieren, die mehrere Variablenketten pro Umgebung enthalten, wobei alle vorkommenden Ketten durch die Aufspaltung einer Ursprungskette (durch Anwendung von *Split-X*

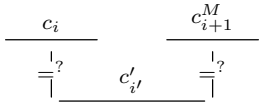
iii)) entstanden sind. D.h. es müssen Gleichungen der Form

$$\{\{|c_1, \ldots, c_n|e|\} =^? \{|c'_1, \ldots, c'_{n'}|d|\}\}$$

unifiziert werden, wobei $c_i, c'_{i'}$ Variablenketten sind, von denen maximal eine Kette nicht mit M markiert ist. Diese nicht markierte Variablenkette enthält die Anfangsbindung der ursprünglichen Variablenkette. Durch die Aufspaltung der Variablenketten liegen die Teilketten in einer Umgebung in der Form:



linear hintereinander. Eine Unifikation der Teilketten nach dem Schema



wobei die Teilketten c_i und $c'_{i'}$ mit M markiert sein können, ist nicht möglich, da durch die resultierende Substitution ein syntaktischer Fehler in die Umgebungen eingeführt wird. Es werden also jeweils einzelne Teilketten miteinander unifiziert. Dabei darf immer nur höchstens eine Teilkette mit M markiert sein.

Definition 5.2.10. Die Regeln zur Unifikation von Termen mit je einer Variablenkette pro Umgebung aus Definition 5.2.3 und Definition 5.2.9 werden um die Regel aus Abbildung 5.5 erweitert, um Unifikationsgleichungen der Form

$$\{\{|c_1, \ldots, c_n|e|\} =^? \{|c'_1, \ldots, c'_{n'}|d|\}\}$$

zu lösen. Dabei sind $c_i, c'_{i'}$ Variablenketten, die jeweils durch die Anwendung von *Split-X iii*) aus einer Ursprungskette entstanden sind, und e, d sind Variablen der Sorte U.

Die Transformationsmöglichkeit *i*) der *Unify-CHS*-Regel behandelt den Fall, dass die Teilkette, die die Anfangsbindung der ursprünglichen Variablenkette enthält (und deshalb nicht mit M markiert ist) mit einer beliebigen Teilkette der gegenüberliegenden Umgebung unifiziert wird. Die resultierende Gleichung $\{|c_i|N_1|\} =^? \{|c'_{i'}|N_2|\}$ kann anschließend mit einer der Regeln *Unify-CH-KMB, Unify-CH-KMR* oder *Unify-CH-KML* transformiert werden. Die restlichen Teilketten der Umgebung werden mit der Umgebungsvariablen der gegenüberliegenden Umgebung unifiziert. Der Fall $\{|c_i^M|N_1|\} =^? \{|c_{i'}^{\prime M}|N_2|\}$ dass beide Variablenkette mit M markiert sind, muss nicht betrachtet werden, da keine der beiden Ketten die Anfangsbindung der ursprünglichen Kette enthält, führt die berechnete Substitution für diesen Fall einen syntaktischen Fehler in die Umgebungen ein. Die Transformationsmöglichkeit *ii*) deckt den Fall ab, dass die Variablenketten der beiden Umgebungen disjunkt sind.

Unify – CHS

$\{\{\!|c_1, \ldots, c_n|e|\!\} =^? \{\!|c'_1, \ldots, c'_{n'}|d|\!\}\} \uplus P \Rightarrow$

i) $\{\{\!|c_i|N_1|\!\} =^? \{\!|c'_{i'}|N_2|\!\}, e =^? \{\!|c'_1, \ldots, c'_{i'-1}, c'_{i'+1}, \ldots, c'_{n'}|N_1|\!\},$
$d =^? \{\!|c_1, \ldots, c_{i-1}, c_{i+1} \ldots, c_n|N_2|\!\}\} \cup P$
wenn eine der beiden Variablenketten $c_i, c'_{i'}$ oder beide Ketten nicht mit M markiert sind. N_1, N_2 sind neue Variablen der Sorte U.

ii) $\{e =^? \{\!|c'_1, \ldots, c'_{n'}|N|\!\}, d =^? \{\!|c_1, \ldots, c_n|N|\!\}\} \cup P$
wenn alle Variablenketten $c_i, c'_{i'}$ mit M markiert sind.
N ist eine neue Variable der Sorte U.

Figure 5.5. Unifikationsregel für `letrec`-Umgebungen mit mehreren Teilketten pro Umgebung, die durch Aufspaltung einer Ursprungskette entstanden sind.

Beispiel 5.2.11. Wir geben ein Beispiel, wie die Unifikation einer Variablenkette, die nicht mit M markiert ist, mit einer Variablenkette, die mit M markiert ist, in einem Unifikator resultiert, der einen syntaktischen Fehler in die Umgebungen einführt, in denen die Ketten enthalten sind. Die Sorten der Variablen des Unifikationsproblems sind $x_i, y_i, z_i : V$, $e, d : U$ und $b_1 : B$.

$$
\begin{aligned}
& \{\{\!|ch(b(x_2, x_1), b(x_n, x_{n-1}))|e|\!\} =^? \{\!|b_1, ch(b(y_2, y_1), b(y_{n'}, y_{n'-1}))|d|\!\}\} \\
\xrightarrow{\textit{Split-L iii)}} & \{b_1 =^? b(z_3, z_2), \\
& \quad \{\!|ch(b(y_2, y_1), b(y_{n'}, y_{n'-1}))|d|\!\} =^? \\
& \quad \{\!|ch(b(x_2, x_1), b(z_2, z_1)), ch(b(z_4, z_3), b(x_n, x_{n-1}))^M|e|\!\}\} \\
\xrightarrow{\textit{Unify-CHS i)}} & \{b_1 =^? b(z_3, z_2), \\
& \quad \{\!|ch(b(y_2, y_1), b(y_{n'}, y_{n'-1}))|N_1|\!\} =^? \{\!|ch(b(z_4, z_3), b(x_n, x_{n-1}))^M|N_2|\!\} \\
& \quad e =^? N_2, d =^? \{\!|ch(b(x_2, x_1), b(z_2, z_1))|N_1|\!\}\} \\
\xrightarrow{\textit{U-CH-KMR iv)}}{}^{7} & \{b_1 =^? b(z_3, z_2), \\
& \quad ch(b(z_7, z_6), b(y_{n'}, y_{n'-1})) =^? ch(b(z_4, z_3), b(x_n, x_{n-1}))^M \\
& \quad N_2 =^? \{\!|ch(b(y_2, y_1), b(z_6, z_5))|N_1|\!\} \\
& \quad e =^? N_2, d =^? \{\!|ch(b(x_2, x_1), b(z_2, z_1))|N_1|\!\}\} \\
\xrightarrow{\textit{Dec}} & \{b_1 =^? b(z_3, z_2), \\
& \quad b(z_7, z_6) =^? b(z_4, z_3), b(y_{n'}, y_{n'-1}) =^? b(x_n, x_{n-1})
\end{aligned}
$$

[7] Die Regel *Unify-CH-KMR iv)* wird auf die Variablenkette $ch(b(z_4, z_3), b(x_n, x_{n-1}))^M$ angewendet, die auf der rechten Seite einer Unifikationsgleichung steht und mit M markiert ist. Nach Defintion 5.2.9 der Unifikationsregeln ist diese Anwendung nicht zulässig. Das Beispiel soll illustrieren, weshalb eine solche Anwendung der *Unify-CH-KMR*-Regel durch die Definition der Regel ausgeschlossen wird.

$$
\begin{aligned}
& N_2 =^? \{\!|ch(b(y_2, y_1), b(z_6, z_5))|N_1|\!\} \\
& e =^? N_2, d =^? \{\!|ch(b(x_2, x_1), b(z_2, z_1))|N_1|\!\}\} \\
\xrightarrow{Dec \times 2} & \{b_1 =^? b(z_3, z_2), \\
& z_7 =^? z_4, z_6 =^? z_3, y_{n'} =^? x_n, y_{n'-1} =^? x_{n-1} \\
& N_2 =^? \{\!|ch(b(y_2, y_1), b(z_6, z_5))|N_1|\!\} \\
& e =^? N_2, d =^? \{\!|ch(b(x_2, x_1), b(z_2, z_1))|N_1|\!\}\} \\
\xrightarrow{Elim\ z_6, N_2} & \{b_1 =^? b(z_3, z_2), \\
& z_7 =^? z_4, z_6 =^? z_3, y_{n'} =^? x_n, y_{n'-1} =^? x_{n-1} \\
& N_2 =^? \{\!|ch(b(y_2, y_1), b(z_3, z_5))|N_1|\!\} \\
& e =^? \{\!|ch(b(y_2, y_1), b(z_3, z_5))|N_1|\!\}, d =^? \{\!|ch(b(x_2, x_1), b(z_2, z_1))|N_1|\!\}\}
\end{aligned}
$$

Das Unifikationsproblem in der letzten Zeile ist in gelöster Form und repräsentiert einen Unifikator σ. Im Term $\{\!|b_1, ch(b(y_2, y_1), b(y_{n'}, y_{n'-1}))|d|\!\}$ wird die Kette analog zur Anwendung der Regel *Unify-CH-KMR iv)* bei der Unifikation zu $\{\!|b_1, ch(b(y_2, y_1), b(z_6, z_5)), ch(b(z_7, z_6), b(y_{n'}, y_{n'-1}))|d|\!\}$ aufgespalten. Jetzt kann σ angewendet werden:

$$
\begin{aligned}
& \sigma(\{\!|b_1, ch(b(y_2, y_1), b(z_6, z_5)), ch(b(z_7, z_6), b(y_{n'}, y_{n'-1}))|d|\!\}) \\
= \ & \{\!|\underline{b(z_3, z_2)}, ch(\sigma(b(y_2, y_1)), \underline{b(z_3, z_5)}), \sigma(ch(b(z_7, z_6), b(y_{n'}, y_{n'-1})))|\sigma(d)|\!\}.
\end{aligned}
$$

Die unterstrichenen Bindungen markieren die Stellen, an denen σ einen syntaktischen Fehler in die `letrec`-Umgebung einführt.

5.2.3 Terminierung

Die Unifikationsprozedur, die durch die Transformationsregeln aus den Definitionen 5.2.3, 5.2.9 und 5.2.10 beschrieben wird, terminiert für Unifikationsprobleme mit Termen, die maximal eine Variablenkette pro Umgebung enthalten. Außerdem dürfen in den zu unifizierenden Termen alle Variablen der Sorte U, K oder M nur genau einmal vorkommen, damit es durch eine Anwendung von *Variable Elimination* nicht zu einer Vervielfachung von Variablenketten in einer Umgebung kommen kann.[8] Die Überlegung zur Terminierung der Unifikationsprozedur für solche eingeschränkten Unifikationsprobleme ist folgende: Zur Unifikation von Termen die eine Variablenkette enthalten, müssen Unifikationsprobleme der Form

$$\{\{\!|b_1, \ldots, b_m, c|e|\!\} =^? \{\!|b'_1, \ldots, b'_{m'}, c'|e'|\!\}\}$$

[8] Die Unifikationsprobleme, die zur Berechnung von Überlappungen gelöst werden müssen, erfüllen diese Beschränkung. Siehe hierzu auch: Definition 6.3.4 von eingeschränkten Unifikationsproblemen.

gelöst werden, wobei $b_i, b'_{i'}$ `letrec`-Bindungen, c, c' Variablenketten und e, e' Variablen der Sorte U sind. Dazu wird zunächst eine Folge von Transformationen mit Regeln aus Definition 5.2.3 auf das Unifikationsproblem angewendet, die die Bindungen $b_i, b'_{i'}$ nach dem Schema der *Cl*-Unifikation unifizieren (dieser Prozess terminiert nach Satz 4.4.7) oder die Bindungen werden durch die Anwendung von *Split-X* mit Bindungen aus Variablenketten unifiziert. Die Anwendung von *Split-X* auf eine Kette c oder c' ist dabei durch die Anzahl der Bindungen in der gegenüberliegenden Umgebung beschränkt. Auch diese Folge von Transformationen terminiert, da die Anzahl der enthalten Bindungen m und m' endlich ist. Durch diese Folge von Transformationen wird das Unifikationsproblem in die Form

$$\{\{|c_1, \ldots, c_n|d|\} =^? \{|c'_1, \ldots, c'_{n'}|d'|\}, s_1 =^? t_1, \ldots s_l =^? t_l\}.$$

gebracht. $c_i, c'_{i'}$ sind durch die Anwendung von *Split-X iii*) aus c bzw. c' entstandenen Teilketten und d, d' sind Variablen der Sorte U. Die Gleichungen $s_j =^? t_j$ sind Unifikationsgleichungen in semi-gelöster Form (Definition 4.4.3) und enthalten die $b_i, b'_{i'}$. Ein Unifikationsproblem in dieser Form enthält maximal eine Variablenkette pro Umgebung, die nicht mit M markiert ist, und kann durch eine Anwendung von *Unify-CHS* u.U. gefolgt von der Anwendung einer der Regeln aus Definition 5.2.9 und einer endlichen Folge von *Variable Elimination*-Schritten in semi-gelöste Form gebracht werden.

Ausblick Die Behandlung der Unifikation von Termen mit Variablenketten in diesem Kapitel ist eher informell. An vielen Stellen, an den eigentlich Beweise von Aussagen nötig sind, werden lediglich Beispiele gegeben. Insbesondere die Vollständigkeit der Unifikationsprozedur wird nicht ausführlich untersucht. Es wird vermutet, dass die beschriebene Unifikationsprozedur, die sich aus der Beschränkung von Anwendungen der Split-Axiome ergibt, eine Eigenschaft besitzt, die man als Überlappungs-Vollständigkeit bezeichnen könnte. Damit ist gemeint, dass für die alle für die Berechnung von Überlappungen relevanten Unifikator von der Unifikationsprozedur gefunden werden.

6 Unifikation von Termen mit Kontextvariablen in $\sum^{let}$

Λ^{let}-Reduktionsregeln enthalten Kontextvariablen als zusätzliches Konstrukt, das bisher nicht behandelt wurde. Diese müssen auch in Σ^{let} dargestellt und bei der Unifikation berücksichtigt werden. Das kann geschehen, indem eine zusätzliche Menge von Kontextvariablen betrachtet wird. Terme mit Kontextvariablen sind aufgebaut wie wohlsortierte Terme mit zusätzlichen Kontextvariablen. Kontexte sind Terme, die an einer Position einen speziellen Loch-Operator ($\Box$) enthalten, in den Terme oder Kontexte eingesetzt werden können. Kontextvariablen in Termen können von Substitutionen durch Kontexte ersetzt werden. Syntaktisch können Kontextvariablen wie Funktionssymbole der Stelligkeit 2 behandelt werden. Da wir Signaturen mit Sorten betrachten, besitzen Kontexte und Kontextvariablen eine Sorte $R \to S$.

Unifikation von Termen mit Kontextvariablen ist eine Variante der Unifikation zweiter Ordnung, da Kontexte als λ-Terme mit genau einem Vorkommen einer gebundenen Variablen betrachtet werden können. D.h. ein Kontext c entspricht dem Term $\lambda\Box.t$, wobei die gebundene Variable $\Box$ genau einmal in t vorkommt. Unifikation zweiter Ordnung ist (ebenso wie Unifikation höherer Ordnung) als unentscheidbar bekannt (Goldfarb, 1981). Die Entscheidbarkeit der Unifikation von Termen mit Kontextvariablen ist ein offenes Problem. Gelten für ein Unifikationsproblem mit Kontextvariablen bestimmte Einschränkungen, kann die Entscheidbarkeit gezeigt werden. Schmidt-Schauß (1994) zeigt, dass unter bestimmten Strukturbeschränkungen der Unifikationsgleichungen (so genannte stratifizierte Gleichungen) das Unifikationsproblem entscheidbar wird. Das Unifikationsproblem ist auch entscheidbar, wenn nur zwei Kontextvariablen vorkommen (Schmidt-Schauß & Schulz, 2002) oder wenn alle Kontextvariablen maximal zweimal in einem Unifikationsproblem vorkommen (Levy, 1996). Comon (1998) zeigt, dass Unifikation mit Kontextvariablen entscheidbar ist, wenn alle in einem Unifikationsproblem vorkommenden Kontextvariablen C auf den gleichen Term t angewandt werden.

In diesem Kapitel beschäftigen wir uns mit der Unifikation von Termen mit Kontextvariablen in Σ^{let}. Das Kapitel ist folgendermaßen aufgebaut: In Abschnitt 6.1 werden Kontexte, Kontextvariablen und Terme mit Kontexten eingeführt. Diese Konstrukte werden bezüglich einer Signatur mit Sorten definiert. Die Signatur Σ^{let} wird erweitert, um die entsprechenden Kontextvariablen (für allgemeine Kontexte, Oberflächen-, Reduktions- und schwache Reduktionskontexte) aus Λ^{let} darstellen zu

können. Durch die Definition von Kontextvariablen für Terme mit Sorten existieren in Σ^{let} mehr Kontextvariablen als im Ursprungskalkül Λ^{let}. Einige Probleme, die sich daraus ergeben, werden kurz diskutiert. Um diesen Problemen zu begegnen, werden *zulässige* Σ^{let}*-Terme* definiert, die nur bestimmte Kontextvariablen enthalten dürfen.

Im anschließenden Abschnitt 6.2 werden Substitutionen erweitert zu Abbildungen, die Variablen auf Terme und Kontextvariablen auf Kontexte abbilden. Wir werden sehen, dass Wohlsortiertheit einer Substitution σ in Σ^{let} als Kriterium nicht ausreicht, um alle Einschränkungen, die für Σ^{let}-Kontexte nach der Definition von Kontexten in Λ^{let} gelten müssen, darzustellen. Als wesentliches Kriterium für (wohlsortierte) Substitutionen wird *Wohlstrukturiertheit* eingeführt. Diese verlangt, dass für alle Komponenten $\{C \mapsto c\}$ einer Substitution die Struktur des Kontextes c der Sorte der Kontextvariablen C entspricht.

In Abschnitt 6.3 befassen wir uns mit der Unifikation von Σ^{let}-Termen, die Kontextvariablen enthalten. Das Vorgehen ist analog zum Bisherigen: Ein Unifikationsproblem wird schrittweise in gelöste Form transformiert. Es wird festgestellt, dass alle Σ^{let}-Unifikationsprobleme, die zur Berechnung (kritischer) Überlappungen gelöst werden müssen, einer Beschränkung unterliegen, unter der sie entscheidbar sind (der Beschränkung von Comon (1998)). Im Anschluss daran wird der Unifikationsprozess für Σ^{let}-Terme mit Kontextvariablen beschrieben, der in zwei Schritten abläuft: Zuerst wird mit Unifikationsregeln, die sich an den Regeln von Comon orientieren, ein wohlsortierter Unifikator für ein Unifikationsproblem berechnet. Die Vollständigkeit und die Terminierung dieser Unifikationsprozedur wird hier nur angedeutet, der komplette Beweis wird nicht gegeben (er ist in der zitierten Arbeit von Comon zu finden). Anschließend wird der berechnete Unifikator an Einschränkungen angepasst, die sich durch die Definition von Kontexten in Λ^{let} ergeben. Dieser Prozess der *Wohlstrukturierung* wird in Abschnitt 6.3.1 beschrieben. Abschließend wird ein ausführliches Beispiel für die Berechnung einer vollständigen Menge von Unifikatoren für ein Σ^{let}-Unifikationsproblem gegeben.

In diesem Kapitel wird die Links-Kommutativität des Funktionssymbols $\{\!|\, \cdot \mid \cdot \,|\!\}$ ignoriert. Es wird als zweistelliges Funktionssymbol behandelt, für das weiter keine Bedingungen gelten. Des weiteren werden in diesem Kapitel keine Variablenketten betrachtet. Die Signatur $\Sigma^{let}_{\{|\cdot|\cdot|\}}$ wird im verkürzt als Σ^{let} geschrieben.

6.1 Kontexte und Kontextvariablen

Um Terme mit Kontextvariablen darzustellen, wird als zusätzliches syntaktisches Konstrukt eine Menge von Kontextvariablen CX benötigt. Kontextvariablen werden durch die Symbole $C, C_1, C'D, E, \ldots$ bezeichnet. Alle Kontextvariablen $C \in CX$ haben die Stelligkeit 2. Signaturen mit Sorten werden erweitert um die Abbildung

$\mathbb{S}_{CX} : CX \to S_\Sigma \times S_\Sigma$, die jeder Kontextvariablen ein Paar von Sorten zuordnet. Im weiteren Verlauf schreiben wir $\mathbb{S}$ und meinen damit entweder die Abbildung $\mathbb{S}_X : X \to S_\Sigma$ oder $\mathbb{S}_{CX} : CX \to S_\Sigma \times S_\Sigma$, je nachdem ob das Argument eine Variable oder eine Kontextvariable ist. Ist C eine Kontextvariable, so dass $\mathbb{S}(C) = (R, S)$, dann schreiben wir auch $C_{(R,S)}$ oder $C : R \to S$. Für eine Signatur Σ mit Sorten enthält die Menge der Kontextvariablen für alle Sortensymbole $R, S \in S_\Sigma$ abzählbar unendliche viele Kontextvariablen der Sorte (R, S).

Für Paare (R, S) von Sorten definieren wir $\underset{\sim}{\sqsubset}$ folgendermaßen:

$$(R, S) \underset{\sim}{\sqsubset} (R', S') \text{ gdw. } R \underset{\sim}{\sqsubset} R' \text{ und } S \underset{\sim}{\sqsubset} S'.$$

Wir bezeichnen mit $\underset{\sim}{\sqsubset}_\Sigma$ den reflexiven, transitiven Abschluss von $\underset{\sim}{\sqsubset}_\Sigma$ und schreiben wie gewohnt kurz $\underset{\sim}{\sqsubset}$.

Die Definition der Σ-Terme der Sorte S wird um Kontextvariablen erweitert.

Definition 6.1.1. Sei Σ eine Signatur mit Sorten, X eine Menge von Variablen und CX eine Menge von Kontextvariablen, so dass $\Sigma \cap X \cap CX = \emptyset$. Die Menge $T(\Sigma, S, X, CX)$ aller Σ-*Terme der Sorte* S über X und CX ist induktiv definiert durch

i) $x \in T(\Sigma, S, X, CX)$, wenn $x \in X$ mit $\mathbb{S}(x) \underset{\sim}{\sqsubset} S$.

ii) $t \in T(\Sigma, S, X, CX)$, wenn $t : R \in \Sigma$ und $R \underset{\sim}{\sqsubset} S$.

iii) $\{x \mapsto r\}t \in T(\Sigma, S, X, CX)$, wenn $t \in T(\Sigma, S, X, CX)$, $r \in T(\Sigma, R, X, CX)$ und $R \underset{\sim}{\sqsubset} \mathbb{S}(x)$.

iv) $C(t) \in T(\Sigma, S, X, CX)$, wenn $C \in CX$ eine Kontextvariable ist mit $\mathbb{S}_{CX}(C) = (R', S')$, so dass $S' \underset{\sim}{\sqsubset} S$ und $t \in T(\Sigma, R, X, CX)$, so dass $R \underset{\sim}{\sqsubset} R'$.

Die ersten drei Punkte der Definition entsprechen der bisherigen Definition von Σ-Termen. Neu hinzu kommt, dass Σ-Terme der Sorte S gebildet werden können, durch die Anwendung einer Kontextvariablen $C \in CX$ der Sorte (R', S') auf einen Term mit einer Sorte kleiner oder gleich R'. Dabei muss S' eine Subsorte von S sein.

Definition 6.1.2. Sei Σ eine Signatur mit Sorten, X eine Menge von Variablen und CX eine Menge von Kontextvariablen, so dass $\Sigma \cap X \cap CX = \emptyset$. Die Menge $C(\Sigma, (R, S), X, CX)$ aller Σ-*Kontext der Sorte* (R, S) über X und CX ist induktiv definiert durch

i) $\Box \in C(\Sigma, (R, S), X, CX)$, für alle $R, S \in S_\Sigma$.

ii) $C \in C(\Sigma, (R,S), X, CX)$, wenn $C \in CX$ mit $\mathbb{S}_{CX}(C) \sqsubseteq_{\sim} (R,S)$.

iii) $t[\Box]_p \in C(\Sigma, (R,S), X, CX)$, wenn $t \in T(\Sigma, S', X, CX)$ keine Variable $x \in X$ ist, so dass $S' \sqsubseteq_{\sim} S$ und $p \in Pos(t)$, so dass $t|_p = s : R' \sqsubseteq_{\sim} R$.

Kontexte der Sorte (R,S) sind entweder Kontextvariablen mit einer Sorte (R',S') kleiner gleich (R,S), oder der spezielle (leere) Kontext $\Box$, der für alle Sorten R,S alle Paare (R,S) als Sorte besitzt. Ein neuer Kontext der Sorte (R,S) kann aus einem Term t der Sorte S' kleiner oder gleich S konstruiert werden Dies geschieht, indem in t (an einer Position p) ein Subterm s mit einer Sorte R' kleiner oder gleich R durch den Kontext $\Box$ ersetzt wird. D.h. ein Kontext ist entweder eine Kontextvariable, oder ein Term mit (genau) einem Loch $\Box$. Um Kontexte zu bezeichnen, verwenden wir die Symbole $c, c_1, c', d, \ldots$.

Im Gegensatz zu Variablen $x \in X$, die in der Baumdarstellung eines Terms nur als Blätter vorkommen, können Kontextvariablen als beliebige Knoten im Baum (mit maximal einem Kind) vorkommen. In Abbildung 6.1 ist ein beispielhafter Kontext in seiner Baumdarstellung zu sehen.

Figure 6.1. Baumdarstellung des Σ-Kontextes $f(x, f(C(y), D(\Box)))$.

Aus der Definition der Subsorten-Quasiordung auf Tupel von Sorten und der Definition von Kontexten folgt, direkt:

Korollar 6.1.3. Sei Σ eine Signatur mit Sorten.

i) Für alle Paare von Sorten $(R,S), (R',S') \in S_\Sigma \times S_\Sigma$ gilt: $(R',S') \sqsubset (R,S)$ impliziert $C(\Sigma, (R',S'), X, CX) \subseteq C(\Sigma, (R,S), CX)$.

ii) Für Kontextvariablen gilt: $C \in T(\Sigma, (R,S), X) \Leftrightarrow \mathbb{S}(C) \sqsubseteq_{\sim} (R,S)$.

In das Loch eines Kontextes können Terme oder Kontexte (einer entsprechenden Sorte) eingesetzt werden.

Definition 6.1.4. Sei $t[\Box]_p : R \to S$ ein Σ-Kontext, und $s : R' \sqsubseteq_{\sim} R$ ein Σ-Term.

- Wenn $p \neq \epsilon$ gilt, dann ergibt die Anwendung des Kontextes auf den Term

  $$t[\Box]_p(s) := t[s]_p$$

  einen Σ-Term der Sorte S. Wenn $s : R' \to S'$ ein Σ-Kontext ist, so dass $S' \sqsubseteq_{\sim} R$, dann ergibt die Anwendung einen Σ-Kontext der Sorte $R' \to S$.

- Wenn $p = \epsilon$ gilt, dann ergibt die Anwendung des Kontextes auf den Term

  $$\Box(s) := s$$

  ein Term der Sorte R'. Ist s ein Kontext der Sorte $R' \to S'$, dann ergibt die Anwendung ein Kontext der Sorte $R' \to S'$.

Wenn c_1 und c_2 Kontexte sind, dann bezeichnen wir $c_1(c_2)$ als Kontextschachtelung.

Definition 6.1.5. Wir definieren die *Menge* $T(\Sigma, X, \mathrm{CX})$ *aller wohlsortierten* Σ-*Terme*, die um eine Menge von Kontextvariablen erweitert ist, als

$$\bigcup_{S \in S_\Sigma} \{T(\Sigma, S, X, CX)\}$$

und die *Menge* $C(\Sigma, X, \mathrm{CX})$ *aller wohlsortierten* Σ-*Kontexte* als

$$\bigcup_{S \in (S_\Sigma \times S_\Sigma)} \{C(\Sigma, S, X, CX)\}.$$

Die *Sorte eines Kontextes* c ist definiert als die Menge

$$S_\Sigma(c) := \{S \in (S_\Sigma \times S_\Sigma) \mid c \in T(\Sigma, S, X, CX)\}$$

und die *Sorte einer Kontextvariablen* C ist definiert als

$$S_\Sigma(C) = \{S \in (S_\Sigma \times S_\Sigma) \mid \mathbb{S}(C) \sqsubseteq_{\sim} S\}.$$

Eine Signatur Σ wird als *regulär* bezeichnet, wenn $(S_\Sigma, \sqsubseteq_{\sim})$ und $(S_\Sigma \times S_\Sigma, \sqsubseteq_{\sim})$ partiell geordnete Mengen sind und für jeden Term t und jeden Kontext c die Mengen $S_\Sigma(t)$ und $S_\Sigma(c)$ ein kleinstes Element besitzen. Das kleinste Element wird mit $LS_\Sigma(t)$ bzw. $LS_\Sigma(c)$ bezeichnet. Ist eine Signatur regulär, so schreiben wir für die partielle Ordnung auf der Menge der Sortensymbole $\sqsubseteq$ anstatt $\sqsubseteq_{\sim}$. Wir beschränken uns im weiteren Verlauf dieses Kapitels auf die Betrachtung regulärer Signaturen.

Definition 6.1.6. Die Signatur Σ^{let} wird verändert, um die entsprechenden Kontextklassen aus Λ^{let} darstellen zu können. Dazu werden neue Sorten L, Ap für `letrec`-Terme und Applikationen eingeführt. Die Sorte A, die bisher Abstraktionen repräsentierte, wird in Ab umbenannt. Die Funktionsdeklarationen werden entsprechend angepasst. Als neue Sorten werden außerdem eingeführt:

- Die Sorte ST repräsentiert Λ^{let}-Oberflächenkontexte, in die ein Term mit einer Subsorte von T eingesetzt wurde. Σ^{let}-Oberflächenkontexte sind Σ^{let}-Kontexte der Sorte $R \to S$ mit $(R, S) \sqsubseteq (T, ST), S \neq V$. Eine Σ^{let}-Oberflächenkontextvariable ist eine Kontextvariable $D \in CX$ der Sorte $T \to ST$.
- Die Sorte RT repräsentiert Λ^{let}-Reduktionskontexte, in die ein Term mit einer Subsorte von T eingesetzt wurde. Σ^{let}-Reduktionskontexte sind Σ^{let}-Kontexte der Sorte $R \to S$ mit $(R, S) \sqsubseteq (T, RT), S \neq V$. Eine Σ^{let}-Reduktionskontextvariable ist eine Kontextvariable $D \in CX$ der Sorte $T \to RT$.
- Die Sorte RWT dient zur Darstellung von schwachen Λ^{let}-Reduktionskontexten, in die ein Term mit einer Subsorte von T eingesetzt wurde. Wir bezeichnen Σ^{let}-Kontexte der Sorte $R \to S$ mit $(R, S) \sqsubseteq (T, RWT), S \neq V$ als schwache Σ^{let}-Reduktionskontexte. Eine schwache Σ^{let}-Reduktionskontextvariable ist eine Kontextvariable $D \in CX$ der Sorte $T \to RWT$.
- Zur Darstellung von allgemeinen Λ^{let}-Kontexten in Σ^{let} wird keine zusätzliche Sorte benötigt. Sie werden durch Σ^{let}-Kontexte der Sorte $R \to S$ mit $(R, S) \sqsubseteq (T, T), S \neq V$ repräsentiert. Eine allgemeine Σ^{let}-Kontextvariable ist eine Kontextvariable $D \in CX$ der Sorte $T \to T$.

Für alle Σ^{let}-Kontexte der Sorte (R, S) wird $S = V$ verboten, weil es keine Funktionsdeklaration der Sorte V in Σ^{let} gibt. Die Subsortendeklarationen werden so getroffen, dass sie der Semantik des Λ^{let}-Kalküls entsprechen. So ist beispielsweise die Sorte der schwachen Reduktionskontexte eine (echte) Subsorte der Sorte der Reduktionskontexte.

$$
\begin{aligned}
\Sigma^{let} = \{\ & \textbf{Subsortdeklarationen}: \\
& Ab, Ap, V, L \sqsubset T, \\
& Ap, L \sqsubset ST \sqsubset T, \\
& Ap, L \sqsubset RT \sqsubset ST, \\
& Ap \sqsubset RWT \sqsubset RT, \\
& B \sqsubset U, \\
& \textbf{Funktionsdeklarationen}: \\
& abs : V \to T \to Ab \quad (\text{Abstraktion}), \\
& app : T \to T \to Ap \quad (\text{Applikation}), \\
& letrec : U \to T \to L \quad (\texttt{letrec}), \\
& \{\!|\cdot|\cdot|\!\} : B \to U \to U \quad (\texttt{letrec} - \text{Umgebung}), \\
& bind : V \to T \to B \quad (\texttt{letrec} - \text{Bindung}), \\
& \emptyset : U \quad (\text{leere } \texttt{letrec} - \text{Umgebung})\ \}
\end{aligned}
$$

Die erweiterte Signatur ist immer noch *einfach*, erfüllt die *maximal-sorts-condition* und hat den *Grad 1*, d.h. ihr Unifikationstyp (ohne Kontextvariablen) ist eindeutig (siehe Kapitel 3). Abbildung 6.1.6 zeigt ein Hasse-Diagramm der Sortstruktur.

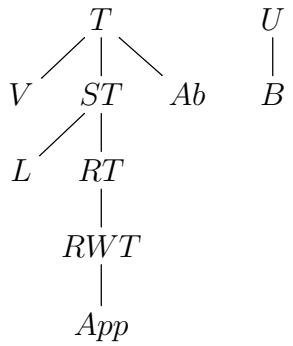


Figure 6.2. Hasse-Diagramm der Sortstruktur von Σ^{let}.

Die in Abschnitt 3.3.2 definierte Abbildung $[\![\ ]\!] : \Lambda^{let} \to T(\Sigma^{let}, X)$ wird folgendermaßen zu einer Abbildung $[\![\ ]\!] : \Lambda^{let} \to T(\Sigma^{let}_{\{\!|\cdot|\cdot|\!\}}, X)$ erweitert, um Kontextvariablen zu übersetzen:

$$
\begin{aligned}
[\![C[s]]\!] &= [\![C]\!]([\![s]]\!]) \quad (\text{analog für } S, R \text{ oder } R^-) \\
[\![C]]\!] &= C : T \to T \\
[\![S]]\!] &= S : T \to ST \\
[\![R]]\!] &= R : T \to RT \\
[\![R^-]]\!] &= R^- : T \to RWT
\end{aligned}
$$

Die Übersetzung, die Σ^{let}-Terme in Λ^{let}-Ausdrücke abbildet, wird mit $[\![\ ]\!]^-$ bezeichnet.

Bemerkung 6.1.7. Der Kontextbegriff in Σ^{let} weist bezüglich des Kontextbegriffs in Λ^{let} eine Reihe von Problemen auf:

i) Nach Korollar 6.1.3 ist $abs(\Box, s) : V \to Ab$ wegen $(V, Ab) \sqsubset (T, T)$ auch ein Kontext der Sorte $T \to T$, die nach unserem Verständnis allgemeine Kontexte repräsentiert. Allerdings darf nach Definition 2.2.1 das Loch in einem allgemeinen Kontext nicht an einer Position in einem Ausdruck vorkommen, an der nur Variablen stehen dürfen. So ist beispielsweise auch $letrec(b(\Box, s), t) : V \to T$ eine Instanz eine Σ^{let}-Kontextes, die keine Entsprechung eines Λ^{let}-Kontextes besitzt. Diesem Problem kann man begegnen, indem man Löcher in Σ^{let}-Termen an Positionen verbietet, an denen nur Terme der Sorte V stehen dürfen. Allerdings kann man damit nicht das allgemeine Problem lösen, das auch für andere Kontextklassen gilt. Beispielsweise ist $app(s, \Box) : T \to Ap$ in Σ^{let} wegen $(T, Ap) \sqsubseteq (T, RT)$ ein RT- und ein RWT-Kontext, aber in Λ^{let} stellt der Kontext keinen gültigen Reduktionskontext dar.

ii) Das Einsetzen in Löcher und die Definition von Kontexten erlauben es in Σ^{let}, Kontexte zu konstruieren, die zwar die (Sub-) Sorte einer bestimmten Kontextklasse haben, aber im Sinne der Λ^{let}-Kontexte keine Kontexte der entsprechenden Kontextklasse mehr sind. Betrachte beispielsweise die Σ^{let}-Kontexte $c_1 = app(\Box, s) : T \to Ap$ und $c_2 = abs(x, \Box) : T \to Ab$. Einsetzen von c_2 in c_1 ergibt $app(abs(x, \Box), t) : T \to Ap$ (nach Definition 6.1.4). Diese Sorte ist in Σ^{let} eine Subsorte der Sorte von Oberflächen- und (schwachen) Reduktionskontexten. Der Kontext $c_1(c_2)$ ist aber in Λ^{let} weder ein Oberflächenkontext noch ein (schwacher) Reduktionskontext. In Σ^{let} lässt sich für Kontexte $c_1 : R_1 \to S_1$, $c_2 : R_2 \to T_2$ die Sorte der Anwendung $c_1(c_2) : R \to S$, die der Λ^{let}-Kontextklasse entspricht, folgendermaßen bestimmen: Nach Definition 6.1.4 gilt $R = R_2$ und für die Sorte S haben wir $S = lub(S_1, S_2)$. Für die Anwendung $app(abs(x, \Box), t)$ obiger Beispielkontexte ergibt sich so die Sorte $(T, lub(Ap, Ab)) = (T, T)$, die der Sorte der Kontextvariablen entspricht, die allgemeine Λ^{let}-Kontexte repräsentieren.

iii) Eine Unterscheidung zwischen ST und RT Kontexten ist nur anhand der Sorten in Σ^{let} nicht möglich.

Um einigen der geschilderten Problemen zu begegnen, definieren wir:

Definition 6.1.8. Ein Σ^{let}-Term t wird als *zulässig* bezeichnet, gdw. für alle Kontextvariablen C in t gilt $\mathbb{S}(C) = (T, T)$ oder $\mathbb{S}(C) = (T, ST)$ oder $\mathbb{S}(C) = (T, RT)$ oder $\mathbb{S}(C) = (T, RWT)$.

Ein Σ^{let}-Kontext c wird als *zulässig* bezeichnet, gdw. für einen zulässigen Term t, der Term $c(t)$ zulässig ist und c keine Löcher an Positionen besitzt, an denen

nur Variablen der Sorte V oder Terme einer Subsorte von U stehen dürfen. D.h. $\forall p \in Pos(c) : c|_p = abs(x,s) \Rightarrow x \neq \Box$ und $c|_p = bind(x,s) \Rightarrow x \neq \Box$, sowie $c|_p = letrec(s,t) \Rightarrow s \neq \Box$ und $c|_p = \{\!|s|t|\!\} \Rightarrow s \neq \Box, t \neq \Box$.

Das erste Problem aus Bemerkung 6.1.7 wird durch diese Definition gelöst: Es werden nur Kontextvariablen in zulässigen Σ^{let}-Termen erlaubt, die eine Entsprechung in Λ^{let} besitzen (d.h. Kontextvariablen für allgemeine Kontexte, für Oberflächenkontexte, und (schwache) Reduktionskontexte). Außerdem sind Löcher an Positionen in Termen, an denen ausschließlich Variablen der Sorte V oder Terme mit einer Subsorte von U stehen dürfen, in zulässigen Termen und Kontexten verboten.

Für zulässige Terme und Kontexte gilt, dass sie eine Entsprechung in Λ^{let}-Ausdrücken besitzen.

Lemma 6.1.9. Sei t ein zulässig Σ^{let}-Term, dann ist $[\![t]\!]^- \in \Lambda^{let}$.

Beweis. Durch strukturelle Induktion über den Aufbau von zulässigen Termen und Kontexten und der Anwendung von $[\![\ ]\!]^-$. $\Box$

Im folgenden Abschnitt wenden wir uns den Problemen 2 und 3 aus Bemerkung 6.1.7 zu.

6.2 Wohlstrukturierte Substitutionen in Σ^{let}

Es werden Substitutionen für Terme und Kontexte mit Kontextvariablen eingeführt zunächst für den Fall einer Signatur $\overline{\Sigma}$ ohne Sorten.

Definition 6.2.1. Analog zu einer Substitution $\sigma_1 : X \to T(\overline{\Sigma}, X, CX)$, die Variablen auf Terme abbildet, definieren wir eine Substitution $\sigma_2 : CX \to C(\overline{\Sigma}, X, CX)$, die Kontextvariablen auf Kontexte abbildet, so dass die Menge $\{C \in CX \mid \sigma_2 C \neq C\}$ endlich ist. Ein Paar von Substitutionen (σ_1, σ_2) wird erweitert zu $\widehat{\sigma} : T(\overline{\Sigma}, X, CX) \to T(\overline{\Sigma}, X, CX)$ und $\widetilde{\sigma} : C(\overline{\Sigma}, X, CX) \to C(\overline{\Sigma}, X, CX)$, so dass

- $\widehat{\sigma}|_X = \sigma_1$ und $\widetilde{\sigma}|_{CX} = \sigma_2$,
- $\widehat{\sigma}(f(t_1, \ldots, t_n)) = f(\widehat{\sigma}t_1, \ldots, \widehat{\sigma}t_n)$ für alle $t_1, \ldots, t_n \in T(\overline{\Sigma}, X, CX)$ und alle $f \in \overline{\Sigma}^n$,
- $\widetilde{\sigma}(f(t_1, \ldots, t_{i-1}, c, t_{i+1}, \ldots t_n)) = f(\widehat{\sigma}t_1, \ldots, \widehat{\sigma}t_{i-1}, \widetilde{\sigma}c, \widehat{\sigma}t_{i+1}, \ldots \widehat{\sigma}t_n)$ für alle t_1, $\ldots$, $t_n \in T(\overline{\Sigma}, X, CX)$, alle $c \in C(\overline{\Sigma}, X, CX)$, alle $f \in \overline{\Sigma}^n$ und alle $1 \leq i \leq n$,
- $\widetilde{\sigma}(c(\widehat{\sigma}t)) = \widehat{\sigma}(c(t))$ und $\widetilde{\sigma}(c_1(\widetilde{\sigma}c_2)) = \widetilde{\sigma}(c_1(c_2))$ für alle $c, c_1, c_2 \in C(\overline{\Sigma}, X, CX)$ und $t \in T(\overline{\Sigma}, X, CX)$,
- $\widetilde{\sigma}(\Box) = \Box$.

Im weiteren Verlauf wird zur Vereinfachung der Notation nicht zwischen (σ_1, σ_2) und $(\widetilde{\sigma},\widehat{\sigma})$ unterschieden. Das Paar wird kurz als σ bezeichnet. Die *Domain* von σ ist definiert als die Vereinigung von $Dom_X(\sigma) := \{x \in X \mid \sigma x \neq x\}$ und $Dom_{CX}(\sigma) := \{C \in CX \mid \sigma C \neq C\}$. Die *Range* von σ ist die Menge der Terme in $T(\overline{\Sigma}, X, CX) \cup C(\overline{\Sigma}, X, CX)$, die Bilder der Variablen in $Dom(\sigma)$ sind. Wir schreiben

$$\{x_1 \mapsto t_1, \ldots, x_m \mapsto t_m, C_1 \mapsto c_1, \ldots, C_n \mapsto c_n\}$$

mit $t_1, \ldots, t_m \in T(\overline{\Sigma}, X, CX)$ und $c_1, \ldots, c_n \in C(\overline{\Sigma}, X, CX)$ für die Substitution mit der Domain $\{x_1, \ldots, x_m, C_1, \ldots, C_n\}$. Die Menge aller unsortierten Substitutionen wird als $Sub_{\overline{\Sigma}}$ bezeichnet.

Wir bemerken, dass Substitutionen keine Löcher in Terme einführen. D.h ein Term t kann durch Anwendung einer Substitution σ nicht zu einem Kontext gemacht werden. Ebenso wenig kann ein Kontext c durch Anwendung von σ zu einem Kontext mit mehr als einem Loch gemacht werden. Wir definieren unter welcher Bedingung Substitutionen *wohlsortiert* sind.

Definition 6.2.2. Die Menge der *wohlsortierten Substitutionen* Sub_Σ ist definiert als:

$$Sub_\Sigma := \{\sigma = (\sigma_1, \sigma_2) \in Sub_{\overline{\Sigma}} \mid S_\Sigma(\sigma_1 x) \supseteq S_\Sigma(x), S_\Sigma(\sigma_2 C) \supseteq S_\Sigma(C)\}.$$

Für reguläre Signaturen ist dies gleichbedeutend mit

$$Sub_\Sigma := \{\sigma = (\sigma_1, \sigma_2) \in Sub_{\overline{\Sigma}} \mid LS_\Sigma(\sigma_1 x) \sqsupseteq LS_\Sigma(x), LS_\Sigma(\sigma_2 C) \sqsupseteq LS_\Sigma(C)\}.$$

Unter Punkt 2 der Bemerkung 6.1.7 haben wir gesehen, dass beispielsweise ein Kontext der Form $c = app(abs(x, \Box), t) : T \to Ap$ eine Subsorte der Sorte von Σ^{let}-Oberflächen- und (schwachen) Reduktionskontexten besitzt. Allerdings ist c in Λ^{let} weder ein Oberflächenkontext noch ein (schwacher) Reduktionskontext. Das Kriterium der Wohlsortiertheit reicht an dieser Stelle nicht aus, um beispielsweise eine Substitution der Form $\{R_{(T,RT)} \mapsto app(abs(x, \Box), t)\}$, die bezüglich der Λ^{let}-Kontextsyntax keine gültig Lösung darstellt, zu vermeiden. Wir definieren deshalb:

Definition 6.2.3. Eine wohlsortierte Substitution

$$\sigma = (\sigma_1 : X \to T(\Sigma^{let}, X, CX), \sigma_2 : CX \to C(\Sigma^{let}, X, CX))$$

wird als *wohlsortiert bezüglich der Λ^{let}-Kontextsyntax* (oder einfach kurz *wohlstrukturiert*) bezeichnet, gdw.

i) alle Terme und alle Kontexte der Range von σ zulässig und alle Kontextvariablen in $Dom(\sigma_2)$ zulässig sind. Und

ii) für alle $C \in Dom(\sigma_2)$ mit $\sigma_2(C) = c$ und für alle Kontexte c_1, c_2, deren Wurzelsymbole ungleich *bind* und $\{\!|\cdot|\cdot|\!\}$ sind, so dass $c = c_1(c_2)$, gilt

$$snd(LS_{\Sigma^{let}}(c_1)) \sqsupseteq snd(\mathbb{S}(C)) \text{ und } snd(LS_{\Sigma^{let}}(c_2)) \sqsupseteq snd(LS_{\Sigma^{let}}(C))$$

Wobei *snd* eine Funktion ist, die wenn ein Tupel gegeben ist, das zweite Element des Tupels zurück gibt, $snd(R, S) = S$. Und

iii) für alle $C \in Dom(\sigma_2)$ der Sorte $(R, S) \sqsubseteq (T, ST)$ mit $\sigma_2(C) = c$ gilt, wenn $c|_p = abs(x, t)$, dann enthält t kein Loch, d.h. $t \in T(\Sigma^{let}, X, CX)$ für alle $p \in Pos(c)$. Und

iv) für alle $C \in Dom(\sigma_2)$ der Sorte $(R, S) \sqsubseteq (T, RT)$ mit $\sigma_2(C) = c$ gilt, wenn $c|_p = app(s, t)$, dann enthält t kein Loch für alle $p \in Pos(c)$. Und

v) für alle $C \in Dom(\sigma_2)$ der Sorte (T, RT) mit $\sigma_2(C) = c$ gilt, wenn $c|_p = letrec(s, t)$ und einer der beiden Terme s, t ein Kontext ist, dann ist $p = \epsilon$. Außerdem wird die Bedingung 2 verschärft zu: Für alle Kontexte c_1, c_2, deren Wurzelsymbole ungleich *bind* und $\{\!|\cdot|\cdot|\!\}$ sind, so dass $c = c_1(c_2)$, gilt $snd(LS_{\Sigma^{let}}(c_1)), snd(LS_{\Sigma^{let}}(c_2)) \sqsupseteq RWT$.

Punkt 2 der Definition stellt sicher, dass in der Range einer Substitution die Kontexte bezüglich der Sorte der Kontextvariablen in der Domain und der Λ^{let}-Definition von Kontexten richtig geschachtelt sind. Die Bedingungen 3 und 4 der Definition stellen sicher, dass in Σ^{let}-Kontexten die Löcher nicht an Positionen stehen, an denen sie für die entsprechenden Λ^{let}-Kontexte nicht erlaubt sind. Bedingung 5 garantiert, dass Reduktionskontexte r in Σ^{let} eine der folgenden zwei Formen haben: Entweder $r = r_1(r_2)$ und alle Kontexte r_1, r_2 sind schwache Reduktionskontexte (der Sorte RWT) oder $r = letrec(s, t)$ und s oder t enthalten als Subkontexte nur schwache Reduktionskontexte.[1]

Beispiel 6.2.4. Die Substitution $\{R_{(T,RT)} \mapsto app(abs(x, \Box), t)\}$ ist nicht wohlstrukturiert, da für $c_1 = app(\Box, t), c_2 = abs(x, \Box)$ gilt: $snd(LS_{\Sigma^{let}}(c_2)) = Ab \not\sqsupseteq RT = snd(LS_{\Sigma^{let}}(R))$. Die Substitution $\{R_{(T,RT)} \mapsto app(C_{(T,T)}(\Box), t)\}$ ist nicht wohlstrukturiert wegen $T \not\sqsupseteq RT$. Im Gegensatz zum vorhergehenden Beispiel kann sie zu einer wohlstrukturierten Substitution gemacht werden, indem die Kontextvariable C instantiiert wird $\{C \mapsto C'_{(T,RT)}\}\{R_{(T,RT)} \mapsto app(C_{(T,T)}(\Box), t)\}$. Die Substitution $\{R_{(T,RT)} \mapsto app(s, \Box)\}$ ist nicht wohlstrukturiert, weil $app(s, \Box)|_\epsilon = app(s, \Box)$ gegen Bedingung 4 aus obiger Definition verstößt.

Die zweite Bedingung aus Definition 6.2.3 berücksichtigt keine Kontextschachtelungen der Form $c_1(c_2)$, wenn wenigstens einer der beiden Kontexte als Wurzelsymbol

[1] Damit r für den Fall $r = letrec(s, t)$ einen Reduktionskontext gemäß der Λ^{let}-Definition von Reduktionskontexten darstellt, muss i.A. eine noch stärkere Bedingung gelten (siehe die Definition 2.2.3 von Reduktionkontexten in Σ^{let}). Da diese jedoch sehr umständlich zu formulieren ist, kommen wir zu einem späteren Zeitpunkt noch einmal darauf zurück (in Beispiel 6.3.8).

$\{\!|\cdot|\cdot|\!\}$ oder *bind* besitzt. Der Grund dafür ist folgender: Wie wir schon bemerkt haben (Bemerkung 6.1.7 und Definition 6.1.8) dürfen in zulässigen Termen keine Löcher an Positionen auftauchen, an denen ausschließlich Terme einer Subsorte von U stehen können. Bei der Betrachtung aller Kontextschachtelungen können allerdings solche nicht zulässigen Kontexte auftreten. Betrachte beispielsweise den Kontext $letrec(\{\!|b(x,\Box)|X|\!\},t)$, der sich in $c_1 = letrec(\Box,t)$ und $c_2 = \{\!|b(x,\Box)|X|\!\}$ zerlegen lässt, wobei c_1 kein zulässiger Kontext ist. Deshalb wird nur die Zerlegung in zulässigen Kontexte

$c_1 = letrec(\{\!|b(x,\Box)|X|\!\},t)$, $c_2 = \Box$ betrachtet. Diese ist ausreichend um eine Schachtelung der Kontexte gemäß der Λ^{let}-Kontextsyntax sicherzustellen.

Proposition 6.2.5. Sei σ eine wohlstrukturierte Substitution und t ein zulässiger Σ^{let}-Term. Dann ist σt ein zulässiger Σ^{let}-Term.

Beweis. Folgt direkt aus der Definition der wohlstrukturierten Substitutionen, da sie keine Kontextvariablen einführen, die nicht zulässig sind und keine Löcher an Positionen einführen, an denen nur Variablen der Sorte V oder Terme einer Subsorte von U erlaubt sind. □

Proposition 6.2.6. Sei $t \stackrel{?}{=} s$ ein Σ^{let}-Unifikationsproblem zwischen zulässigen Termen s,t und σ eine wohlstrukturierte Lösung des Unifikationsproblems. Dann gilt $[\![\sigma s]\!]^-, [\![\sigma t]\!]^- \in \Lambda^{let}$.

Beweis. Die wohlstrukturierte Substitution σ erhält die Zulässigkeit der Terme s und t (Proposition 6.2.5) und nach Lemma 6.1.9 gilt für die zulässigen Terme σs und σt: $[\![\sigma s]\!]^-, [\![\sigma t]\!]^- \in \Lambda^{let}$. □

Proposition 6.2.7. Seien σ und τ wohlstrukturierte Substitutionen. Dann ist auch $\sigma\tau$ eine wohlstrukturierte Substitution.

Beweis. Seien $\sigma = \{x_1 \mapsto s_1, \ldots, x_m \mapsto s_n, C_1 \mapsto c_1, \ldots, C_n \mapsto c_n\}$ und $\tau = \{y_1 \mapsto t_1, \ldots, y_k \mapsto t_k, D_1 \mapsto d_1, \ldots, D_l \mapsto d_l\}$ wohlstrukturiert. Dann ist $\sigma\tau = \{y_1 \mapsto \sigma t_1, \ldots, y_k \mapsto \sigma t_k, D_1 \mapsto \sigma d_1, \ldots, D_l \mapsto \sigma d_l\} \cup \{x_i \mapsto s_i, C_j \mapsto c_j \mid x_i, C_j \in Dom(\sigma) - Dom(\tau)\}$. Die Komponenten $\{y_i \mapsto \sigma t_i\}, 1 \leq i \leq k$ sind wohlstrukturiert, weil y_i, t_i zulässige Terme sind (nach Voraussetzung) und σ wohlstrukturiert ist, und die Anwendung σt_i nach Proposition 6.2.5 die Zulässigkeit der Terme erhält.

Jetzt ist noch zu zeigen, dass die Komponenten $\{D_i \mapsto \sigma d_i \mid 1 \leq i \leq l\}$ wohlstrukturiert sind, d.h. für alle d'_1, d'_2 deren Wurzelsymbole ungleich $\{\!|\cdot|\cdot|\!\}$ und *bind* sind, mit $d'_1(d'_2) = d_i$ muss $snd(LS_{\Sigma^{let}}(\sigma d'_1)), snd(LS_{\Sigma^{let}}(\sigma d'_2)) \sqsupseteq D_i$ gelten. Falls d'_1 und d'_2 keine Kontextvariablen aus der Domain von σ sind, gilt die Bedingung, weil die Sorte von Kontexten in einer einfachen Signatur nicht von der Sorte der Subterme abhängig ist. Sei $d'_1 = C_j$ eine Kontextvariable aus der Domain von σ, dann folgt $snd(LS_{\Sigma^{let}}(\sigma d'_1)) \sqsupseteq D_i$ aus der Wohlsortiertheit von σ:

$snd(LS_{\Sigma^{let}}(\sigma d'_1)) = snd(LS_{\Sigma^{let}}(\sigma C_j)) \sqsubseteq snd(LS_{\Sigma^{let}}(\sigma c_j))$. Die Überlegung ist analog, falls d'_2 eine Kontextvariable aus der Domain von σ ist, bzw. falls d'_1, d'_2 beides Kontextvariablen aus der Domain von σ sind.

Außerdem gilt für alle Kontextvariablen $D_i \in Dom(\tau)$, die eine Subsorte von (T, ST) besitzen, dass in d_i nur Kontextvariablen vorkommen, die eine Subsorte von (T, ST) besitzen (da τ wohlstrukturiert ist; Punkt 2 von Definition 6.2.3). Da auch σ wohlstrukturiert ist, führt die Anwendung σd_i in d_i keine Terme der Form $abs(x, t)$, so dass t ein Kontext ist. Die Überlegung gilt gleichermaßen für die Bedingungen 4 und 5.

D.h. für die Komposition $\sigma\tau$ zweier wohlstrukturierter Substitutionen gelten die Bedingungen aus Definition 6.2.3, folglich ist $\sigma\tau$ wohlstrukturiert. □

Definition 6.2.8. Seien σ und τ wohlsortierte Substitutionen und $W \subseteq X \cup CX$ eine Menge von Variablen. Die Substitution σ ist *allgemeiner auf* W als die Substitution τ, geschrieben als $\sigma \lesssim_\Sigma \tau[W]$, wenn es eine wohlsortierte Σ^{let}-Substitution δ gibt, so dass gilt $\tau = \delta\sigma[W]$.

Da die Komposition von wohlstrukturierten Σ^{let}-Substitutionen σ, τ wieder wohlstrukturiert ist, kann man analog definieren: $\sigma \lesssim^{wstr}_{\Sigma^{let}} \tau[W]$, gdw. es eine wohlstrukturierte Substitution δ gibt, so dass $\tau = \delta\sigma[W]$.

6.3 Unifikation von Σ^{let}-Termen mit Kontextvariablen

Das Vorgehen zur Lösung eines Unifikationsproblems mit Kontextvariablen ist analog zu den bisher vorgestellten Verfahren: Ein Unifikationsproblem wird durch sukzessive Anwendung von Transformationen in gelöste Form gebracht, aus der ein Unifikator für das Problem direkt abgelesen werden kann. Da die Terme nun Kontextvariablen enthalten, passen wir die entsprechenden Begriffe dementsprechend an. Außerdem muss für Σ^{let}-Unifikationsprobleme berücksichtigt werden, dass nur wohlstrukturierte Substitutionen als Lösungen in Frage kommen.

Definition 6.3.1. Eine *Termgleichung* ist eine Gleichung $s =^? t$ zwischen zwei Termen $s, t \in T(\Sigma, X, CX)$ und eine *Kontextgleichung* ist eine Gleichung $C =^? c$ zwischen einer Kontextvariablen $C \in CX$ und einem Kontext $c \in C(\Sigma, X, CX)$. Ein Σ-*Unifikationsproblem mit Kontextvariablen* ist eine endliche Menge von Term- und Kontextgleichungen $P = \{s_1 =^? t_1, \ldots, s_m =^? t_m, C_1 =^? c_1, \ldots, C_n =^? c_n\}$. Ein Σ-*Unifikator* von P ist eine wohlsortierte Substitution σ, so dass $\sigma t_1 = \sigma s_i$ für $i = 1, \ldots, m$ und $\sigma C_j = \sigma c_j$ für $j = 1, \ldots, n$. Die Menge aller wohlsortierten Unifikatoren von P wird als $U_\Sigma(P)$ bezeichnet

$$U_\Sigma(P) := \{\sigma \in Sub_\Sigma |\ \sigma s_i = \sigma t_i \text{ für } i = 1, \ldots, n \text{ und } C_j = c_j \text{ für } j = 1, \ldots, m\}.$$

P ist *lösbar* wenn $U_\Sigma(P) \neq \emptyset$. Das spezielle Unifikationsproblem $\bot$ besitzt keine Lösung. Mit $Var(P)$ wird die Menge der in P auftauchenden Variablen und Kontextvariablen bezeichnet.

Ist P ein Σ^{let}-Unifikationsproblem, dann muss außerdem gelten:

- Die Term- und Kontextgleichungen in P enthalten nur zulässige Terme.
- Ein Unifikator σ von P ist eine wohlstrukturierte Substitution.
- Die Menge aller Unifikatoren von P besteht aus der Menge aller wohlstrukturierten Unifikatoren, die P lösen.

Man startet bei der Transformation eines Unifikationsproblems mit Kontextvariablen immer mit einem Problem, das nur Termgleichungen enthält. Während der Transformation können Kontextgleichungen eingeführt werden, auf die als einzige Unifikationsregel die Eliminierung von Kontextvariablen angewandt wird. Aus diesem Grund sind Kontextgleichungen auf die Form $C = c, C \in CX, c \in C(\Sigma, X, CX)$ beschränkt. Wenn wir im weiteren Verlauf dieses Kapitels von Unifikationsproblemen sprechen, sind Unifikationsprobleme mit Kontextvariablen gemeint.

Als einzige Änderung der Definition einer (minimalen) vollständigen Menge von Unifikatoren für ein Unifikationsproblem mit Kontextvariablen, betrachten wir die Instantiierungs-Quasiordnung modulo der Menge von Variablen und Kontextvariablen, die in einem Unifikationsproblem vorkommen.

Definition 6.3.2. Eine *vollständige Menge von Unifikatoren* für ein Σ-Unifikationsproblem mit Kontextvariablen P ist eine Menge (wohlsortierter) Substitutionen $CU_\Sigma(P)$, so dass

- $CU_\Sigma(P) \subseteq U_\Sigma(P)$ und
- für alle $\tau \in U_\Sigma(P)$ gibt es $\sigma \in CU_\Sigma(P)$, so dass $\sigma \lesssim_\Sigma \tau[Var(P)]$.

Eine *minimale vollständige Menge von Σ-Unifikatoren* für ein Σ-Unifikationsproblem mit Kontextvariablen P ist eine vollständige Menge von Unifikatoren $MU_\Sigma(P)$, so dass

- für alle $\sigma, \tau \in MU_\Sigma(P)$ gilt: $\sigma \lesssim_\Sigma \tau[Var(P)]$ impliziert $\sigma =_\Sigma \tau[Var(P)]$.

Eine Substitution σ ist ein *allgemeinster Σ-Unifikator* von P gdw. $\{\sigma\}$ eine minimale vollständige Menge von Lösungen für P ist.

Ist P ein Σ^{let}-Unifikationsproblem, dann muss außerdem gelten, dass die (minimale) vollständige Menge nur wohlstrukturierte Substitutionen enthält.

Definition 6.3.3. Ein Unifikationsproblem mit Kontextvariablen

$$S = \{x_1 =^? t_1, \ldots x_m =^? t_m, C_1 =^? c_1, \ldots C_n =^? c_n\}$$

ist in *gelöster Form*, gdw. alle Variablen x_i und Kontextvariablen C_j paarweise verschieden sind und nicht in t_i und c_j vorkommen und $LS_\Sigma(x_i) \sqsubseteq LS_\Sigma(t_i), LS_\Sigma(C_i) \sqsubseteq$

$LS_\Sigma(c_i)$ gilt. In diesem Fall definiert man die Lösung von S als

$$\sigma_S = \{x_1 \mapsto t_1, \ldots x_m \mapsto t_m, C_1 \mapsto c_1, \ldots C_n \mapsto c_n\}.$$

Für ein Σ^{let}-Unifikationsproblem muss außerdem gelten, dass die S zugeordnete Lösung σ_S eine wohlstrukturierte Substitution ist.

Ein Unifikationsproblem in gelöster Form repräsentiert einen mgu für dieses Problem.

Die Entscheidbarkeit von Unifikationsproblemen mit Kontextvariablen ist ein offenes Problem. Gelten bestimmte Einschränkungen für Unifikationsprobleme, so kann deren Entscheidbarkeit gezeigt werden. Eine Beschränkung, unter der Unifikationsprobleme mit Kontextvariablen entscheidbar sind, ist folgende von Comon (1998) angegebene Bedingung (die so genannte Comon-Einschränkung): Für alle Kontextvariablen C eines Unifikationsproblems gilt, dass alle Vorkommen von C auf denselben Term t angewandt werden. Comon gibt für solche eingeschränkten Unifikationsprobleme einen vollständigen und terminierenden Unifikationsalgorithmus an. Dabei berücksichtigt er auch Sorten, allerdings ist sein Ansatz diesbezüglich schwer mit dem hier gewählten vergleichbar, da er Sorten in Form von Constraints verwendet, die zusätzliche Bestandteile von Unifikationsproblemen sind. Wir gehen daher hier nicht weiter auf seine Methode zur Behandlung von Sorten ein, wollen uns aber an seinem Ansatz zur Unifikation von Termen mit Kontextvariablen orientieren. Dazu halten wir zunächst fest, dass die Σ^{let}-Unifikationsprobleme, an deren Lösung wir vornehmlich interessiert sind, die Einschränkung von Comon erfüllen: Für alle linken Seiten der Λ^{let}-Unifikationsregeln gilt, dass sie ausschließlich paarweise verschiedene Kontextvariablen enthalten, d.h. die Einschränkung von Comon ist offensichtlich erfüllt. Außerdem werden zur Berechnung von Überlappungen die beiden zu unifizierenden Terme so umbenannt, dass sie variablendisjunkt sind (siehe Abschnitt 7.2). Für alle Σ^{let}-Unifikationsprobleme, die zur Berechnung von Überlappungen gelöst werden müssen, gilt sogar eine noch stärkere Einschränkung.

Definition 6.3.4. Sei P ein Σ^{let}-Unifikationsproblem mit Kontextvariablen zwischen zulässigen Termen. Das Problem P wird als *eingeschränkt* bezeichnet, gdw. alle in P vorkommenden Kontextvariablen genau einmal in P vorkommen und alle in P vorkommenden Variablen, einer beliebigen Sorte R ungleich V, genau einmal in P vorkommen.

Variablen der Sorte V dürfen in einem eingeschränkten Problem mehr als einmal vorkommen. Das mehrfache Vorkommen von Variablen dieser Sorte ist unproblematisch, weil es durch die Elimination von Variablen der Sorte V nicht zu einer Vervielfachung von Kontextvariablen in einem Unifikationsproblem kommen kann (da Termgleichungen der Form $x_V =^? C(t)$ nicht in wohlsortierter, gelöster Form

sind). Unter obiger Beschränkung wird somit bei der Transformation von Σ^{let}-Unifikationsproblemen vermieden, dass sich Kontextvariablen durch Elimination in einem Problem vervielfachen. Diese Tatsache stellt die Terminierung des Unifikationsalgorithmus sicher.

Proposition 6.3.5. Alle Σ^{let}-Unifikationsprobleme, die zur Berechnung aller (kritischen) Überlappungen von linken Seiten der Λ^{let}-Reduktionsregeln gelöst werden müssen, sind eingeschränkte Unifikationsprobleme.

Beweis. Folgt durch Betrachtung der linken Seiten der Λ^{let}-Reduktionsregeln (bzw. der Σ^{let}-Reduktionsregeln aus Kapitel 7) und weil die Regeln zur Berechnung von Überlappungen so umbenannt werden, dass sie variablendisjunkt sind. □

Wir betrachten nun die wesentlichen Unifikationsregeln zur Transformation von Unifikationsproblemen mit Kontextvariablen nach Comon (1998). Zur Vereinfachung der Notation wird angenommen, dass die zugrunde liegende Signatur einfach ist und alle Funktionssymbole eine Stelligkeit von zwei besitzen (was in Σ^{let} der Fall ist). Die Unifikationsregeln, die Comon verwendet, kombinieren die Regeln *Variable Elimination* (für Variablen und Kontextvariablen) und *Decomposition* in einer *Merge*-Regel, weil eine Eliminierung von (Kontext-) Variablen im Allgemeinen die Comon-Einschränkung von Unifikationsproblemen nicht erhält. Da wir für Σ^{let}-Unifikationsprobleme aber eine noch stärkere Einschränkung (Def. 6.3.4) als die von Comon betrachten, können wir die Eliminierung von (Kontext-) Variablen beibehalten.

Definition 6.3.6. Sei Σ eine einfache Signatur, so dass alle Funktionssymbole in Σ die Stelligkeit zwei besitzen und P ein eingeschränktes Σ-Unifikationsproblem mit Kontextvariablen. Die Regeln zur Unifikation in regulären Signaturen aus Definition 3.4.13 werden um die in Abbildung 6.3 dargestellten Regeln erweitert.

Split

$\{f(t_1, t_2) =^? C(s)\} \uplus P \Rightarrow$

i) $\{C =^? \Box, f(t_1, t_2) =^? s\} \cup P$

ii) $\{C =^? f(C'(\Box), t_2), t_1 =^? C'(s)\} \cup P$
wenn $f(x_1, x_2)$ die Funktionsdeklaration von f ist und $LS_\Sigma(C) = (R, S)$,
dann ist $C' : R \to LS_\Sigma(x_1)$ eine neue Kontextvariable.

iii) $\{C =^? f(t_1, C'(\Box)), t_2 =^? C'(s)\} \cup P$
wenn $f(x_1, x_2)$ die Funktionsdeklaration von f ist und $LS_\Sigma(C) = (R, S)$,
dann ist $C' : R \to LS_\Sigma(x_2)$ eine neue Kontextvariable.

Context Decompositon

$\{C(s) =^? D(t)\} \uplus P \Rightarrow$

i) $\{s =^? D'(t), D =^? C(D'(\Box))\} \cup P$
wenn $LS_\Sigma(C) = (R_C, S_C)$ und $LS_\Sigma(D) = (R_D, S_D)$,
dann ist $D' : R_D \to R_C$ eine neue Kontextvariable.

ii) $\{t =^? C'(s), C =^? D(C'(\Box))\} \cup P$
wenn $LS_\Sigma(C) = (R_C, S_C)$ und $LS_\Sigma(D) = (R_D, S_D)$,
dann ist $C' : R_C \to R_D$ eine neue Kontextvariable.

iii) $\{C =^? E(f(C'(\Box), D'(t))), D =^? E(f(C'(s), D'(\Box)))\} \cup P$
wenn $glb(LS_\Sigma(C(s)), LS_\Sigma(D(t))) = S$ existiert, $f(x_1, x_2) : S_f$
eine Funktionsdeklaration ist, und $LS_\Sigma(C) = (R_C, S_C),\ LS_\Sigma(D) = (R_D, S_D)$.
Dann sind $C' : R_C \to LS_\Sigma(x_1),\ D' : R_D \to LS_\Sigma(x_2)$ und
$E : S_f \to S,$ neue Kontextvariablen.

iv) $\{C =^? E(f(D'(t), C'(\Box))), D =^? E(f(D'(\Box), C'(s)))\} \cup P$
wenn $glb(LS_\Sigma(C(s)), LS_\Sigma(D(t))) = S$ existiert, $f(x_1, x_2) : S_f$
eine Funktionsdeklaration ist, und $LS_\Sigma(C) = (R_C, S_C),\ LS_\Sigma(D) = (R_D, S_D)$.
Dann sind $C' : R_C \to LS_\Sigma(x_2),\ D' : R_D \to LS_\Sigma(x_1)$ und
$E : S_f \to S,$ neue Kontextvariablen.

Context Variable Elimination (**sorted**)

$\{C =^? c\} \uplus P \Rightarrow \{C =^? c\} \cup \{C \mapsto c\}P$
wenn $C \notin Var(c)$ und $LS_\Sigma(c) \sqsubseteq LS_\Sigma(C)$.

Figure 6.3. Regeln zur Unifikation von eingeschränkten Unifikationsproblemen mit Kontextvariablen nach Comon (1998).

Weitere Regeln, die zur Unifikation benötigt werden sind:

Context Orientation

$\{C(s) =^? t\} \uplus P \Rightarrow \{t =^? C(s)\} \cup P$

wenn t ein Term ist, dessen Wurzelsymbol keine Kontextvariable ist.

Context Weakening

$\{x =^? C(s)\} \uplus P \Rightarrow$

wenn $x \notin C(s)$ und $LS_\Sigma(x) \not\sqsubseteq LS_\Sigma(C(s))$

i) $\{C =^? \Box, x =^? s\} \cup P$

ii) $\{x =^? f(x_1, x_2), f(x_1, x_2) =^? C(s)\} \cup P$

wenn $f(x_1, x_2) : S$ eine Funktionsdeklaration (mit neuen Variablen) ist, so dass $S = glb(LS_\Sigma(x), LS_\Sigma(C(t)))$.

Ist auf ein Unifikationsproblem eine Regel anwendbar, bei der mehr als eine Transformationsmöglichkeit gegeben ist $(i, ii, \dots)$, dann wird eine beliebige Alternative ausgewählt, für die das Unifikationsproblem die Voraussetzungen erfüllt.

Die Sorten der neuen Kontextvariablen, eingeführt in den Regeln *Split* und *Context Decomposition*, werden so gewählt, dass sie der Definition von Kontexten (6.1.2) und der Einschränkung, die sich für die Sorten durch die Einsetzung in Kontexte ergibt (Definition 6.1.4), entsprechen.

Die Unifikationsprozedur, die durch die Regeln aus Definition 6.3.6 definiert ist, berechnet für ein Σ^{let}-Unifikationsproblem mit Kontextvariablen P einen wohlsortierten Unifikator σ, der i.A. nicht wohlstrukturiert ist. Falls für P ein wohlstrukturierter Unifikator existiert, kann dieser berechnet werden, indem eine Substitution δ bestimmt wird, so dass $\delta\sigma$ ein wohlstrukturierter Unifikator für P ist (ähnlich der Berechnung eines Weakenings für eine unsortierte Substitution in Proposition 4.3.6). Durch die Aufteilung der Berechnung von wohlstrukturierten Unifikatoren in zwei Schritte, kann auch die Vollständigkeit und die Terminierung der Unifikation in zwei Schritten gezeigt werden: Zuerst wird die Vollständigkeit und Terminierung der durch die Regeln aus Definition 6.3.6 beschriebenen Unifikationsprozedur gezeigt. Da diese auf Einschränkungen der Unifikationsregeln von Comon (1998) basiert, kann dazu auf dessen Beweise zurückgegriffen werden. Anschließend kann die Vollständigkeit und die Terminierung des Verfahrens gezeigt werden, das versucht, einen berechneten nicht wohlstrukturierten Unifikator wohlzustrukturieren.

Wie wir bereits bemerkt haben, ist Comons Ansatz bezüglich der Behandlung von Sorten nicht direkt mit dem hier gewählten vergleichbar. Um auf seinen Beweis der Vollständigkeit und Terminierung zurückgreifen zu können, sollte deshalb eine modifizierte Version der Unifikationsregeln aus Definition 6.3.6 verwendet werden, die allen neu eingeführten Kontextvariablen die *TOP*-Sorte einer Signatur zuweist (vgl.

Lemma 4.3.7). Vollständigkeit und Terminierung kann dann mit Comons Methoden gezeigt werden. Zur Wohlsortierung ist anschließend noch die Berechnung eines Weakenings notwendig. Um die Präsentation des Verfahrens zur Wohlstrukturierung in Σ^{let} nicht zu kompliziert zu gestalten, wurden die Unifikationsregeln in Definition 6.3.6 direkt so formuliert, dass eine Sorte für neu eingeführte Kontextvariablen gemäß Definitionen 6.1.2 und 6.1.4 gewählt wird.

Wir skizzieren kurz die Überlegung, weshalb die beiden Regeln *Split* und *Context Decomposition* aus Definition 6.3.6 eine vollständige und terminierende Unifikationsprozedur beschreiben. Die kompletten Beweise sind in Comon (1998) zu finden. Auf die zusätzlichen Regeln, die wegen der expliziten Berücksichtigung der Sorten vorhanden sind, wird ebenfalls kurz eingegangen.

Vollständigkeit Betrachte die Regel *Split*. Sei σ eine Substitution, so dass $\sigma f(t_1, t_2) = \sigma C(s) = \sigma C[\sigma s]_p$ gilt. Für $p = \epsilon$ muss $\sigma C = \Box$ und $\sigma f(t_1, t_2) = \sigma s$ gelten (Fall i) der *Split*-Regel)[2]. Ist $p = iq$ ungleich ϵ, dann folgt $p = 12$ oder $p = 21$, weil alle Funktionssymbole in Σ zweistellig sind. Für den ersten Fall ist das Loch von C in einem Subterm von t_1, d.h. wir haben $\sigma C = \sigma f(C'(\Box), t_2)$ und $\sigma t_1 = \sigma C'(s)$ (Fall ii) der *Split*-Regel). Der Fall $p = 21$ wird analog durch *Split* iii) abgedeckt.

Betrachte die Regel *Context Decomposition*. Sei σ eine Substitution, so dass $\sigma C(s) = \sigma D(t)$ gilt. D.h. nach der Definition von Substitutionen haben wir $\sigma C(\sigma s) = \sigma D(\sigma t)$, was $\sigma C[\sigma s]|_{p_1} = \sigma D[\sigma t]|_{p_2}$ entspricht. Entweder sind p_1 und p_2 vergleichbare Positionen, oder die beiden Positionen sind nicht vergleichbar. Falls die Positionen vergleichbar sind, haben wir $p_1 \leq p_2$, was *Context Decomposition* i) entspricht, oder $p_2 \leq p_1$, was *Context Decomposition* ii) entspricht. Sind die beiden Position p_1, p_2 der Löcher in σC und σD parallel (d.h. nicht vergleichbar), dann sei f das Funktionssymbol, das an der kleinsten Position steht, an der p_1 und p_2 vergleichbar sind. Der Kontext E steht direkt über dieser Position. Die Terme s und t stehen dann an nicht vergleichbaren Positionen unterhalb der Position von f in neuen Kontexten C' und D'. Dieser Fall wird durch die Transformationsmöglichkeiten *Context Decomposition* iii) und iv) abgedeckt.

Die Anwendung der zusätzlichen Regeln *Context Variable Elimination*, *Context Orientation* und *Sorted Fail Context* verändern die Menge der Lösungen nicht (die Regeln sind analog zu den jeweiligen Regeln für Variablen aus Definition 3.4.13).

Die Regel *Context Weakening* behandelt den Fall, dass eine Gleichung zwischen einer Variablen und der Anwendung einer Kontextvariablen auf einen Term in gelöster, aber nicht wohlsortierter Form ist. Dann kann entweder die Kontextvariable zum leeren Kontext instantiiert werden (i), oder es gibt eine Termdeklaration, deren

[2]Diese Regel wird auch für die Σ^{let}-Unifikation von Gleichungen der Form $\emptyset =^? C(s)$ verwendet. Da das Konstantensymbol nullstellig ist $\emptyset$, d.h. es kann kein Loch enthalten.

Sorte eine untere Schranke der Sorten der beiden Terme ist, so dass die Variable instantiiert werden kann (*ii*). In Σ^{let} tritt dieser Fall beispielsweise für Gleichungen der Form $x_{Ap} =^? C_{T,T}(t)$ auf. Durch Anwendung von *Context Weakening* erhält man $\{x_{Ap} =^? abs(y_V, s_T), abs(y_V, s_T) =^? C_{T,T}(t)\}$.

Terminierung Der Terminierungsbeweis von Comon (1998) wird hier nicht nachvollzogen. Wir wollen nur auf einen wichtigen Punkt hinweisen: Die Eingeschränktheit (Def. 6.3.4) eines Unifikationsproblems wird durch die Transformationen *Split* und *Context Decomposition* i.A. nicht erhalten, weil neue Kontextvariablen an zwei Positionen in ein Unifikationsproblem eingeführt werden. Betrachte beispielsweise *Split ii*): $\{f(t_1, t_2) =^? C(t)\} \uplus P \Rightarrow \{C =^? f(C'(\Box), t_2), t_1 =^? C'(s)\} \cup P$. Im zweiten Unifikationsproblem taucht die neue Kontextvariable C' zweimal auf. Allerdings kann die Gleichung $C =^? f(C'(\Box), t_2)$ eliminiert werden (oder das Problem besitzt keine Lösung). Da C keine neu eingeführte Kontextvariable ist, kommt C in P wegen der Eingeschränktheit nicht mehr vor. D.h. die Eliminierung von C führt nicht zu einer Vervielfachung von C', und da die Gleichung in gelöster Form ist, hat nur noch das (einzelne) Vorkommen von C' in der Gleichung $t_1 =^? C'(s)$ Einfluss auf den weiteren Verlauf der Unifikation (das Teilproblem ohne die Gleichungen in gelöster Form bleibt in beschränkter Form). Diese Überlegung gilt analog für die anderen Transformationen, die neue Kontextvariablen mehr als einmal einführen.

6.3.1 Wohlstrukturierung von Σ^{let}-Unifikatoren

Für ein Σ^{let}-Unifikationsproblem P wird durch die oben beschriebene Unifikationsprozedur ein Unifikator σ aus der vollständigen Menge der Unifikatoren von P berechnet, der wohlsortiert, aber i.A. nicht wohlstrukturiert ist. Es wird ein allgemeinster Unifikator σ bezüglich der Instantiierungs-Quasiordnung auf wohlsortierten Substitutionen berechnet, jedoch nicht bezüglich der Ordnung auf wohlstrukturierten Substitutionen. Wenn P einen wohlstrukturierten Unifikator besitzt, dann kann aus σ ein solcher gewonnen werden, indem eine Substitution δ[3] berechnet wird, so dass $\delta\sigma$ eingeschränkt auf die in P vorkommenden Variablen wohlstrukturiert ist. Dazu instantiiert die Substitution δ neu eingeführte Kontextvariablen in $Ran(\sigma)$, die gegen Bedingungen aus Definition 6.2.3 verstoßen.

Lemma 6.3.7. Sei P ein eingeschränktes Σ^{let}-Unifikationsproblem und σ ein (wohlsortierter) Unifikator von P. Wenn P einen wohlstrukturierten Unifikator besitzt, dann kann eine Substitution δ berechnet werden, so dass $\delta\sigma|_{Var(P)}$ ein wohlstrukturierter Unifikator von P ist.

[3] Die nicht wohlstrukturiert, insbesondere nicht wohlsortiert sein muss.

Beweis. Sei σ ein nicht wohlstrukturierter Unifikator eines Σ^{let}-Unifikationsproblems P. Gehe folgendermaßen vor, um σ wohlzustrukturieren:

i) Sei D eine Kontextvariable in $Ran(\sigma)$, die nicht zulässig ist. Wenn $LS_{\Sigma^{let}}(D)$ eine Subsorte von $(T,T),(T,ST),(T,RT)$ oder (T,RWT) ist, dann sei (R',S') die kleinste dieser vier Sorten für die $LS_{\Sigma^{let}}(D) \sqsubseteq (R',S')$ gilt und sei D' eine neue Kontextvariable der Sorte (R',S'). Setze $\delta = \{D \mapsto D'\}$ und starte erneut bei Punkt 1 mit $\delta\sigma$. Ist $LS_{\Sigma^{let}}(D)$ keine Subsorte einer der Sorten $(T,T),(T,ST),(T,RT)$ oder (T,RWT), dann wähle eine Funktionsdeklaration $f(x_1,x_2)$ aus Σ^{let}, so dass $LS_{\Sigma^{let}}(D) \sqsubseteq LS_{\Sigma^{let}}(f(\Box,x_2))$ oder $LS_{\Sigma^{let}}(D) \sqsubseteq LS_{\Sigma^{let}}(f(x_1,\Box))$.[4] Existiert für D keine Funktionsdeklaration mit dieser Eigenschaft, dann stoppe mit der Antwort: P besitzt keine wohlstrukturierte Lösung. Wir gehen o.B.d.A. vom ersten Fall aus, der zweite funktioniert analog. Unterscheide folgende Fälle:

- Wenn die Sorte S von $f(x_1,x_2)$ eine Subsorte von T ist ($S \sqsubseteq T$), dann sei $D' : S \to S_D$, mit $(R_D,S_D) = LS_{\Sigma^{let}}(D)$, eine neue Kontextvariable und $\delta = \{D \mapsto D'(f(\Box,x_2))\}$. Beginne den Prozess der Wohlstrukturierung erneut mit $\delta\sigma$.
- Wenn die Sorte von $f(x_1,x_2)$ keine Subsorte von T ist, dann sei $D' : R_D \to LS_\Sigma(x_1)$ eine neue Kontextvariable mit $(R_D,S_D) = LS_{\Sigma^{let}}(D)$ und $\delta = \{D \mapsto f(D'(\Box),x_2))\}$. Beginne den Prozess der Wohlstrukturierung erneut mit $\delta\sigma$.

Sind alle Variablen in der Range von σ zulässig, dann fahre mit folgendem Punkt fort.

ii) Überprüfe für alle Kontextvariablen D in $Dom(\sigma)$ mit $\{D \mapsto d\}$ und $LS_{\Sigma^{let}}(D) = (R,S)$, ob die Bedingungen 3 und 4 aus Definition der wohlstrukturierten Substitutionen (Def. 6.2.3) erfüllt sind. Ist eine der Bedingungen nicht erfüllt, dann breche mit der Antwort ab: P besitzt keine wohlstrukturierte Lösung. Wenn D eine Kontextvariable der Sorte (T,RT) ist, dann prüfe, ob die Bedingung 5 aus Definition 6.2.3 für d gilt ($\forall p \in Pos(d) : d_p = letrec(s,t) \Rightarrow p = \epsilon$, wenn s oder t ein Kontext ist). Gilt die Bedingung nicht, sei p die Position in $Pos(d)$, so dass $d|_p = letrec(s,t)$ und einer der beiden Terme s,t ist ein Kontext. Wenn alle Terme an Positionen $q < p$ als Wurzelsymbole Kontextvariablen $C_1,C_2,\ldots \notin Var(P)$ besitzen, dann sei $\delta = \{C_1 \mapsto \Box, C_2 \mapsto \Box, \ldots\}$ und starte den Prozess der Wohlstrukturierung erneut mit $\delta\sigma$. Steht an einer Position q ein Kontext mit einem Funktionssymbol als Wurzelsymbol, dann stoppe mit der Antwort: P besitzt keinen wohlstrukturierten Unifikator.

Als Letztes ist noch zu prüfen, ob die Subkontexte in d bezüglich der Λ^{let}-Kontextsyntax richtig geschachtelt sind (Bedingung 2 und Bedingung

[4] Die Wahl ist für alle nicht zulässigen Kontextvariablen eindeutig (wegen der Struktur von Σ^{let}).

5 aus Definition 6.2.3). Wenn für alle Kontexte d_1, d_2 mit Wurzelsymbolen ungleich $\{\!|\cdot|\cdot|\!\}$ oder *bind*, so dass $d = d_1(d_2)$ die Bedingung $snd(LS_{\Sigma^{let}}(d_1)), snd(LS_{\Sigma^{let}}(d_2)) \sqsupseteq S$ gilt, dann stoppe und gebe $\sigma|_{Var(P)}$ als wohlstrukturierte Substitution zurück. Seien d_1, d_2 Kontexte, für die diese Bedingung nicht gilt, dann unterscheide folgende Fälle:

- d_1 ist ein Kontext der Sorte $R_1 \to S_1$, so dass $S_1 \sqsubseteq S$ gilt. Wenn d_2 als Wurzelsymbol eine Kontextvariable $D_2 : R_2 \to S_2$ besitzt, so dass $D_2 \notin Var(P)$ gilt und $glb(R_1, S) = S'$ existiert, dann sei $D_2' : R_2 \to S'$[5] und $\delta = \{D_2 \mapsto D_2'\}$. Starte erneut unter Punkt 1 mit $\delta\sigma$. Hat d_2 als Wurzelsymbol ein Funktionssymbol oder eine Kontextvariable $D_2 \in Var(P)$ oder existiert $glb(R_1, S)$ nicht, dann stoppe mit der Antwort: P besitzt keine wohlstrukturierte Lösung.
- d_1 ist ein Kontext der Sorte $R_1 \to S_1$ mit $S_1 \not\sqsubseteq S$. Wenn das Wurzelsymbol von d_1 eine Kontextvariable D_1 ist, so dass $D_1 \notin Var(P)$ gilt und $glb(S_1, S) = S'$ existiert, dann sei $D_1' : R_1 \to S'$[6] eine neue Kontextvariable und $\delta = \{D_1 \mapsto D_1'\}$. Starte erneut unter Punkt 1 mit $\delta\sigma$. Besitzt d_1 als Wurzelsymbol ein Funktionssymbol oder eine Kontextvariable $D_1 \in Var(P)$ oder existiert $glb(S_1, S)$ nicht, dann stoppe mit der Antwort: P besitzt keinen wohlstrukturierten Unifikator.

Die Substitution δ erhält man durch die Komposition der in den einzelnen Schritten berechneten δ. □

Für den im Beweis beschriebenen Prozess der Wohlstrukturierung eines (wohlsortierten) Unifikators σ für ein eingeschränktes Σ^{let}-Unifikationsproblem P gilt:

- Der Prozess terminiert: Da σ eine Substitution ist, müssen nur endlich viele nicht zulässige Kontextvariablen instantiiert werden. Die Sorten der neuen Kontextvariablen (D') werden dabei so gewählt, dass sie zulässig sind (Punkt 1). Um die Bedingungen 3, 4 und 5 aus Definition 6.2.3 zu überprüfen, müssen endlich viele Subterme in der Range von σ inspiziert werden. Um die Bedingung 2 aus Definition 6.2.3 zu überprüfen, müssen endlich viele Kontextschachtelungen $d_1(d_2)$ inspiziert werden (Punkt 2).
- Der Prozess ist vollständig: Die Menge der wohlstrukturierten Unifikatoren des Σ^{let}-Unifikationsproblems wird durch den Prozess nicht verändert.
- Der Prozess berechnet einen wohlstrukturierten Unifikator aus der vollständigen Menge der Unifikatoren des Σ^{let}-Unifikationsproblems P: Die

[5] Hat die Kontextvariable D für die betrachtete Komponente $\{D \mapsto d\}$ die Sorte $T \to RT$ und $glb(R_1, RWT)$ existiert, dann sei D_2' eine neue Kontextvariable der Sorte $R_2 \to RWT$ (Bedingung 5 aus Definition 6.2.3).

[6] Bzw. $D_1' : R_1 \to RWT$, falls $D : T \to RT$ gilt und $glb(S_1, S)$ existiert (Bedingung 5 aus Definition 6.2.3).

Substitution σ ist ein wohlsortierter Unifikator von P. Diese Substitution wird gemäß der Definition wohlstrukturierter Substitutionen angepasst.

Als Endresultat halten wir fest: Das eingeschränkte Σ^{let}-Unifikationsproblem mit Kontextvariablen ist entscheidbar. Eine vollständige Menge von wohlstrukturierten Unifikatoren kann folgendermaßen berechnet werden: Berechne zuerst mit der durch die Regeln in Definition 6.3.6 beschriebenen Unifikationsprozedur eine vollständige Menge von wohlsortierten Unifikatoren für ein Σ^{let}-Unifikationsproblem P. Berechne dabei die vollständige Menge von Alternativen von Transformationen. Dieses Verfahren terminiert und ist vollständig. Besitzt P wohlstrukturierte Lösungen, so können diese aus der vollständigen Menge der Unifikatoren durch das im Beweis von Lemma 6.3.7 beschriebene Verfahren gewonnen werden. Auch dieser Prozess terminiert und ist vollständig.

Es wird ein ausführliches Beispiel zur Berechnung von wohlstrukturierten Unifikatoren in zwei Schritten gegeben.

Beispiel 6.3.8. Es sei folgendes Σ^{let}-Unifikationsproblem gegeben: $P = \{C(s_T) =^? R(t_T)\}$, mit den Kontextvariablensorten $C : T \to T$ und $R : T \to RT$. Wir betrachten die Berechnung der Menge aller wohlstrukturierten Unifikatoren für P.

Anwendung der Regeln *Context Decomposition* i) und ii) auf P ergibt die beiden Probleme
$\{s_T =^? R'(t), R =^? C(R'(\Box))\}$, mit $R' : T \to T$ und
$\{t_T =^? C'(s), C =^? R(C'(\Box))\}$, mit $C' : T \to T$.
Das zweite Problem ist in gelöster, wohlstrukturierter From. Das erste Problem ist in gelöster Form und es ist wohlsortiert, allerdings ist es nicht wohlstrukturiert, da für $R =^? C(R'(\Box))$ und $c_1 = C(\Box), c_2 = R'(\Box)$ gilt $snd(LS_{\Sigma^{let}}(C)) = T \not\supseteq RT = snd(LS_{\Sigma^{let}}(R))$ und da $C \in Var(P)$ ist, kann die Sorte der Kontextvariablen nicht angepasst werden.

Die Regeln *Context Decomposition* iii) und iv) sind auf P anwendbar, da die größte untere Schranke $glb(LS_{\Sigma^{let}}(C(s_T)), LS_{\Sigma^{let}}(R(t_T))) = glb(T, RT) = RT$ existiert. Jetzt müssen alle Funktionssymbole $f \in \Sigma^{let}$ betrachtet werden.

- Sei $f = app : T \to T \to Ap$. Die Anwendung von *Context Decomposition* iii) und iv) ergibt die Unifikationsprobleme
  $\overset{iii)}{\longrightarrow} \{C =^? E(app(C'(\Box), R'(t_T))), R =^? E(app(C'(s_T), R'(\Box)))\} := P_1,$
  $\overset{iv)}{\longrightarrow} \{C =^? E(app(R'(t_T), C'(\Box))), R =^? E(app(R'(\Box), C'(s_T)))\} := P_2.$
  Mit den Variablensorten $E : Ap \to RT, C' : T \to T$ und $R' : T \to T$. Beide Probleme sind in gelöster Form und wohlsortiert und beide Probleme enthalten die nicht zulässige Kontextvariable E, deren Sorte (Ap, RT) eine Subsorte der Σ^{let}-Kontextvariablen Sorten $(T, T), (T, ST)$ und (T, RT) ist. Die kleinste dieser Sorten ist (T, RT), d.h. wir instantiieren E mit einer neuen zulässigen Kontextvariablen dieser Sorte und setzten die Wohlstrukturierung

mit $\{E \mapsto E'_{(T,RT)}\}P_i, i = 1, 2$ fort. Jetzt müssen die Bedingungen zwei bis fünf aus Definition 6.2.3 überprüft werden. Für P_1 haben wir für die rechte Seite der zweiten Gleichung $E'(app(C'(s_T), R'(\Box))|_1 = app(C'(s_T), R'(\Box))$ und weil $R : T \to RT$ ist, wird die Bedingungen 4 aus Definition 6.2.3 nicht erfüllt. Das erste Problem repräsentiert somit keine wohlstrukturierte Lösung für P. Im Unifikationsproblem P_2 ist Bedingung 5 aus Definition 6.2.3 an zwei Stellen verletzt: Für $c_1 = E'(app(\Box, C'(s_T)))$ mit der Sorte $T \to RT$ und $c_2 = R'(\Box)$ mit der Sorte $T \to T$, gilt $snd(LS_{\Sigma^{let}}(c_2)) = T \not\sqsupseteq RWT$. Da aber $R' \notin Var(P)$ gilt, kann die Sorte von R' angepasst werden, indem $R'' : T \to glb(RWT, T)$ eine neue Kontextvariable ist. Außerdem ist $snd(LS_{\Sigma^{let}}(E')) = RT \not\sqsupseteq RWT$. Da aber $E' \notin Var(P)$ gilt, instantiieren wir E' mit einer neuen Variablen $E'' : T \to RWT$. Dann repräsentiert $\{R'_{(T,T)} \mapsto R^-_{(T,RWT)}, E'_{(T,RT)} \mapsto E''_{(T,RWT)}\}P_2$ eine wohlstrukturierte Lösung für P.

- Sei $f = abs : V \to T \to Ab$. Durch die Anwendung von *Context Decomposition iii)* und *iv)* erhält man die Probleme

  $\overset{iii)}{\longrightarrow} \{C =^? E(abs(C'(\Box), R'(t_T))), R =^? E(abs(C'(s_T), R'(\Box)))\} \quad := P_1$

  mit Variablensorten $E : Ab \to RT$, $C' : T \to V$, $R' : T \to T$.

  $\overset{iv)}{\longrightarrow} \{C =^? E(abs(R'(t_T), C'(\Box))), R =^? E(abs(R'(\Box), C'(s_T)))\} \quad := P_2$

  mit Variablensorten $E : Ab \to RT$, $C' : T \to T$, $R' : T \to V$.

  Beide Unifikationsprobleme enthalten Kontextvariablen, die nicht zulässig sind. In P_1 sind E und C' nicht zulässig, in P_2 sind E und R' nicht zulässig. Für die Sorte $T \to V$ der beiden Variablen existiert keine Funktionsdeklaration in $f \in \Sigma^{let}$, so dass $(T, V) \sqsubseteq LS_{\Sigma^{let}}(f(\Box, x_2))$ oder $(T, V) \sqsubseteq LS_{\Sigma^{let}}(f(x_1, \Box))$. Die beiden Probleme sind deshalb nicht wohlstrukturierbar.

- Sei $f = letrec : U \to T \to L$. Die Anwendung von *Context Decomposition iii)* und *iv)* resultiert in den Problemen

  $\overset{iii)}{\longrightarrow} \{C =^? E(letrec(C'(\Box), R'(t_T))), R =^? E(letrec(C'(s_T), R'(\Box)))\} := P_1$

  mit Variablensorten $E : L \to RT$, $C' : T \to U$, $R' : T \to T$.

  $\overset{iv)}{\longrightarrow} \{C =^? E(letrec(R'(t_T), C'(\Box))), R =^? E(letrec(R'(\Box), C'(s_T)))\} := P_2$

  mit Variablensorten $E : L \to RT$, $C' : T \to T$, $R' : T \to U$.

  Das Problem P_1 enthält zwei nicht zulässige Variablen E und C', die mit zulässigen Kontexten instantiiert werden können: E wird analog zu obigem Fall $f = app$ instantiiert. Für C' betrachte die Funktionsdeklaration $b(x_V, u_T)$, für die gilt $LS_{\Sigma^{let}}(C') = (T, U) \sqsubseteq LS_{\Sigma^{let}}(b(x_V, \Box))$. Die Sorte U ist keine Subsorte von T, deshalb sei C'' eine neue Kontextvariable der Sorte $T \to LS_{\Sigma^{let}}(u_T) = T \to T$ und wir überprüfen, ob $\{C' \to b(x_V, C''(\Box)), E \mapsto E'_{(T,RT)}\}P_1 = P'_1 =$
  $\{C =^? E'(letrec(b(x_V, C''(\Box)), R'(t_T))), R =^? E'(letrec(b(x_V, C''(s_T)), R'(\Box)))\}$
  wohlstrukturiert ist. Die zweite Gleichung von P'_1 verstößt an zwei Stellen

gegen Bedingung 5 aus Definition 6.2.3. An der Position $p = 1$ des Kontextes $E'(letrec(b(x_V, C''(s_T)), R'(\Box)))$ steht ein Kontext, der als Wurzelsymbol das Funktionssymbol *letrec* besitzt. Der Kontext an der Position $q = \epsilon, q < p$ besitzt als Wurzelsymbol eine Kontextvariable $E' \notin Var(P)$, die durch den leeren Kontext instantiiert werden kann. Außerdem haben wir für $c_1 = E'(letrec(b(x_V, C''(s_T)), \Box)), c_2 = R'(\Box)$, dass $snd(LS_{\Sigma^{let}}(c_2)) = T \not\sqsupseteq RWT$ gilt. Weil $R' \notin Var(P)$ gilt, kann die Sorte von R' angepasst werden, so dass $\{R'_{(T,T)} \mapsto R^-_{(T,RWT)}, E'_{(T,RT)} \mapsto \Box\}P'_1$ eine wohlstrukturierte Lösung von P repräsentiert.

Für P_2 sind $R' : T \to U$ und $E : L \to T$ keine zulässigen Kontextvariablen, die auf die gleiche Weise wie oben beschrieben mit einem zulässigen Kontext instantiiert werden können. Als Resultat erhalten wir:
$\{C =^? letrec(b(x_V, R^-(t)), C'(\Box)), R =^? letrec(b(x_V, R^-(\Box)), C'(s)\} =: P'_2$
mit den Variablensorten $C' : T \to T$, $R^- : T \to RWT$. Das Problem repräsentiert eine wohlstrukturierte Lösung, allerdings ist der Kontext $letrec(b(x_V, R^-(\Box)), C'(s))$ kein Reduktionskontext bezüglich der Definition von Reduktionskontexten in Λ^{let}. Damit der Kontext einen Λ^{let}-Reduktionskontext entspricht, muss C' eine Kontextvariable der Sorte $T \to RWT$ sein und $s = x_V$ gelten. Diese Bedingung kann durch den Algorithmus zur Wohlstrukturierung von Substitutionen berücksichtigt werden. Sie wurde nicht mit in die Beschreibung des Verfahrens aufgenommen, um die Notation nicht weiter zu verkomplizieren. Im vorliegenden Programm, das die Unifikation für Σ^{let}-Kontextvariablen implementiert wird diese Bedingung berücksichtigt.

- Sei $f = \{\!|\cdot|\cdot|\!\} : B \to U \to U$. Die Anwendung von *Context Decomposition* *iii*) und *iv*) ergibt die Unifikationsprobleme
  $\underset{\longrightarrow}{iii)}$ $\{C =^? E(\{\!|C'(\Box)|R'(t_T)|\!\}), R =^? E(\{\!|C'(s_T)|R'(\Box)|\!\})\}$ $:= P_1$
  mit Variablensorten $E : U \to RT$, $C' : T \to B$, $R' : T \to U$.
  $\underset{\longrightarrow}{iv)}$ $\{C =^? E(\{\!|R'(t_T)|C'(\Box)|\!\}), R =^? E(\{\!|R'(\Box)|C'(s_T)|\!\})\}$ $:= P_2$
  mit Variablensorten $E : U \to RT$, $C' : T \to U$, $R' : T \to B$.
  Beide Probleme enthalten Kontextvariablen, die nicht zulässig sind, aber durch zulässige Terme instantiiert werden können. Betrachte das Problem P_1:
  - Wir haben $LS_{\Sigma^{let}}(E) = (U, RT) \sqsubseteq LS_{\Sigma^{let}}(letrec(\Box, u_T)))$ und $L \sqsubseteq T$. Deshalb setzen wir $\delta_1 = \{E \mapsto E'(letrec(\Box, u_T))\}$, wobei E' die Sorte $L \to RT$ hat.
  - Es gilt $LS_{\Sigma^{let}}(C') = (T, B) \sqsubseteq LS_{\Sigma^{let}}(b(x_V, \Box)))$ und $B \not\sqsubseteq T$. Deshalb setze $\delta_2 = \{C' \mapsto b(x_V, C''(\Box))\}$ mit $C'' : T \to T$.
  - Es gilt $LS_{\Sigma^{let}}(R') = (T, U) \sqsubseteq LS_{\Sigma^{let}}(b(y_V, \Box)))$ und $B \not\sqsubseteq T$. Deshalb setze $\delta_3 = \{C' \mapsto b(y_V, R''(\Box))\}$ mit $R'' : T \to T$.

Die Anwendung $(\delta_1 \cup \delta_2 \cup \delta_3)P$ ergibt das Problem
$\{C =^? E'(letrec(\{\!|b(x_V, C''(\Box))|b(y_V, R''(t_T))|\!\}, u_T)),$
$R =^? E'(letrec(\{\!|b(x_V, C''(s_T))|b(y_V, R''(\Box))|\!\}, u_T))\} =: P_1',$
das jetzt auf Wohlstrukturiertheit geprüft werden muss. Dabei muss die Sorte der Variablen R'' angepasst werden und E' muss mit dem leeren Kontext instantiiert werden: $\{R''_{(T,T)} \mapsto R^-_{(T,RWT)}), E' \mapsto \Box\}P_1' =$
$\{C =^? letrec(\{\!|b(x_V, C''(\Box))|b(y_V, R^-(t_T))|\!\}, u_T),$
$R =^? letrec(\{\!|b(x_V, C''(s_T))|b(y_V, R^-(\Box))|\!\}, u_T)\}$
Das Problem ist in wohlstrukturierter, gelöster Form. Allerdings ist auch hier der Kontext $letrec(\{\!|b(x_V, C''(s_T))|b(y_V, R^-(\Box))|\!\}, u_T)$ kein Reduktionskontext in Λ^{let}. Damit der Kontext ein Reduktionskontext ist, muss außerdem $u_T = R'^-[y_V]$ gelten (auch dieser Fall wird in der Implementierung korrekt berücksichtigt).

Für das Problem P_2 erfolgt die Wohlstrukturierung analog.

- Sei $f = b : V \to T \to B$. Dieser Fall lässt sich analog zu Fall $f = abs$ nicht wohlstrukturieren.

Die vollständige Menge der wohlstrukturierten Unifikatoren für das Unifikationsproblem $\{C(s_T) =^? R(t_T)\}$ besteht somit aus folgenden Substitutionen:

i) $\{t_T \mapsto C'(s), C \mapsto R(C'(\Box))\}$

ii) $\{C \mapsto E''(app(R^-(t_T), C'(\Box))), R \mapsto E''(app(R^-(\Box), C'(s_T)))\}$

iii) $\{C \mapsto letrec(b(x_V, C''(\Box)), R^-(t_T)), R \mapsto letrec(b(x_V, C''(s_T)), R^-(\Box))\}$

iv) $\{C \mapsto letrec(b(x_V, R^-(t)), C'(\Box)), R \mapsto letrec(b(x_V, R^-(\Box)), C'(s))\}$
mit $C' : T \to RWT$ und $s = x_V$

v) $\{C \mapsto letrec(\{\!|b(x_V, C''(\Box))|b(y_V, R^-(t_T))|\!\}, u_T),$
$R \mapsto letrec(\{\!|b(x_V, C''(s_T))|b(y_V, R^-(\Box))|\!\}, u_T)\}$
mit $u_T = R'^-[y_V]$

vi) $\{C \mapsto letrec(\{\!|b(y_V, R^-(t_T))|b(x_V, C''(\Box))|\!\}, u_T),$
$R \mapsto letrec(\{\!|b(y_V, R^-(\Box))|b(x_V, C''(s_T))|\!\}, u_T)\}$
mit $u_T = R'^-[y_V]$

7 Berechnung von Überlappungen für Gabeldiagramme in Σ^{let}

Um einen vollständigen Satz von Gabeldiagrammen für eine interne Reduktion zu berechnen, müssen zunächst alle möglichen Überlappungen zwischen no-Reduktionen und der internen Reduktion bestimmt werden (vgl. Abschnitt 2.6.1). In diesem Kapitel wird der Begriff der *Überlappung* von Reduktionsregeln formal definiert. Dazu orientieren wir uns an dem Begriff der Überlappung, der in der Theorie der Termersetzungssysteme verwendet wird (Baader & Nipkow, 1998; Bezem et al., 2003).

Zuerst wird in Abschnitt 7.1 der Σ^{let}-Kalkül vorgestellt, der verwendet wird, um Überlappungen zu berechnen. Die Signatur Σ^{let}, deren Definition sich über die Kapitel 3, 4, 5 und 6 erstreckt, wird zusammengefasst angegeben, ebenso wie die Übersetzung $[\![\ ]\!]$, die Λ^{let}-Ausdrücke in Σ^{let}-Terme übersetzt. Anschließend werden Reduktionsregeln und die standardisierte Form der Auswertung, die Normalordnungsreduktion, für Σ^{let}-Terme definiert. Die Reduktionsregeln in Σ^{let} bilden die Reduktionsregeln aus Λ^{let} nach. Die Normalordnungsreduktion in Σ^{let} entspricht einer eingeschränkten Normalordnungsreduktion in Λ^{let}.

Im Abschnitt 7.2 wird der Begriff der Überlappung formal definiert und eine Methode zur Berechnung von Überlappungen in Σ^{let} angegeben.

7.1 Der Σ^{let}-Kalkül

Die Signatur Σ^{let} bildet die Basis, um Terme mit Sorten, Kontextvariablen, einem links-kommutativen Funktionssymbol $\{\!|\cdot|\cdot|\!\}$ und Variablenketten darzustellen und zu unifizieren.

Definition 7.1.1. Die Definition der Signatur Σ^{let} und die Übersetzung $[\![\ ]\!]$, die Λ^{let}-Ausdrücke in Meta-Notation (die für die Definition von Reduktionsregeln in Λ^{let} verwendet wird, (siehe Abschnitt 2.3) in Σ^{let}-Termen übersetzt, ist in Abbildung 7.1 zu sehen. Die Umkehrabbildung, die Σ^{let}-Terme in Λ^{let}-Ausdrücke übersetzt, wird mit $[\![\ ]\!]^{-}$ bezeichnet.

$\Sigma^{let} = \{$ **Subsortdeklarationen** :

$Ab, Ap, V, L \sqsubset T, \quad Ap, L \sqsubset ST \sqsubset T, \quad Ap, L \sqsubset RT \sqsubset ST,$
$Ap \sqsubset RWT \sqsubset RT, \quad B, K \sqsubset M \sqsubset U,$

Funktionsdeklarationen :

$abs : V \to T \to Ab$ (Abstraktion),
$app : T \to T \to Ap$ (Applikation),
$letrec : U \to T \to L$ (`letrec`),
$\{\!|\cdot|\cdot|\!\} : M \to U \to U$ (`letrec` − Umgebung).
$ch : B \to B \to K$ (Variablenkette),
$bind : V \to T \to B$ (`letrec` − Bindung),
$\emptyset : U$ (leere `letrec` − Umgebung) $\}$

$$
\begin{aligned}
[\![x]\!] &= x_V \\
[\![v]\!] &= v_A \\
[\![s]\!] &= s_T \\
[\![t]\!] &= t_T \\
[\![Env]\!] &= e_U \\
[\![\lambda x.s]\!] &= abs([\![x]\!], [\![s]\!]) \\
[\![(s\ t)]\!] &= app([\![s]\!], [\![t]\!]) \\
[\![(\texttt{letrec}\ \{Env\}\ \texttt{in}\ s)]\!] &= letrec([\![Env]\!], [\![s]\!]) \\
[\![(\texttt{letrec}\ \{Env\}\ \texttt{in}\ s)]\!] &= letrec([\![Env]\!], [\![s]\!]) \\
[\![\{Env, x_1 = s_1, \ldots, x_n = s_n\}]\!] &= [\![\{x_1 = s_1, \ldots, x_n = s_n, Env\}]\!] \\
[\![\{x_1 = s_1, \ldots, Env, \ldots, x_n = s_n\}]\!] &= [\![\{x_1 = s_1, \ldots, x_n = s_n, Env\}]\!] \\
[\![\{x_1 = s_1, x_2 = s_2, \ldots, x_n = s_n\}]\!] &= \{\!|\ [\![x_1 = s_1]\!] \mid [\![\{x_2 = s_2, \ldots, x_n = s_n\}]\!]\ |\!\} \\
[\![\{\{x_i = x_{i-1}\}_{i=m}^{n}, x_1 = s_1, \ldots\}]\!] &= \{\!|[\![\{x_i = x_{i-1}\}_{i=m}^{n}]\!]|[\![\{x_1 = s_1, \ldots\}]\!]|\!\} \\
[\![\{x = s, Env\}]\!] &= \{\!|[\![x = s]\!]|[\![Env]\!]|\!\} \\
[\![\{x = s\}]\!] &= \{\!|[\![x = s]\!]|\emptyset|\!\} \\
[\![x = s]\!] &= bind([\![x]\!], [\![s]\!]) \\
[\![\{x_i = x_{i-1}\}_{i=m}^{n}]\!] &= ch(bind([\![x_m]\!], [\![x_{m-1}]\!]), bind([\![x_n]\!], [\![x_{n-1}]\!]) \\
[\![C[s]]\!] &= [\![C]\!]([\![s]\!]) \quad \text{(analog für } S, R \text{ oder } R^-\text{)} \\
[\![C]\!] &= C : T \to T \\
[\![S]\!] &= S : T \to ST \\
[\![R]\!] &= R : T \to RT \\
[\![R^-]\!] &= R^- : T \to RWT
\end{aligned}
$$

Figure 7.1. Die Signatur Σ^{let} und die Übersetzung $[\![\ ]\!]$.

Definition 7.1.2. Zu einer Signatur Σ ist eine *Reduktionsregel* ein Paar (l, r) von Σ-Termen. Reduktionsregeln werden geschrieben als $l \to r$ und falls sie einen Namen ρ besitzen auch als $\rho : l \to r$.

Zu der Reduktionsregel $\rho : l \to r$ wird eine Instanz von l (d.h. σl für eine Substitution σ) als ρ-*Redex* bezeichnet.

Definition 7.1.3. Die Reduktionsregeln des Σ^{let}-Kalküls sind in Abbildung 7.2 zu sehen.

(llet-in)	$letrec(e_1, letrec(e_2, s)) \longrightarrow letrec(\{\!\lfloor e_1 \vert e_2 \rfloor\!\}, s)$
(llet-e)	$letrec(\{\!\lfloor b(x, letrec(e_2, s)) \vert e_1 \rfloor\!\}, t) \longrightarrow letrec(\{\!\lfloor b(x, s), e_2 \vert e_1 \rfloor\!\}, t)$
(cp-in)	$letrec(\{\!\lfloor b(x_1, v), ch(b(x_2, x_1), b(x_n, x_{n-1})) \vert e \rfloor\!\}, C(x_n)) \longrightarrow letrec(\{\!\lfloor b(x_1, v), ch(b(x_2, x_1), b(x_n, x_{n-1})) \vert e \rfloor\!\}, C(v))$
(cp-e)	$letrec(\{\!\lfloor b(x_1, v), ch(b(x_2, x_1), b(x_n, x_{n-1})), b(y, C(x_n)) \vert e \rfloor\!\}, t) \longrightarrow letrec(\{\!\lfloor b(x_1, v), ch(b(x_2, x_1), b(x_n, x_{n-1})), b(y, C(v)) \vert e \rfloor\!\}, t)$
(lapp)	$app(letrec(e, s), t) \longrightarrow letrec(e, app(s, t))$
(lbeta)	$app(abs(x, s), t) \longrightarrow letrec(\{\!\lfloor b(x, t) \vert \emptyset \rfloor\!\}, s)$

Figure 7.2. Reduktionsregeln des Σ^{let}-Kalküls.

Die Reduktionsregeln des Σ^{let}-Kalküls ergeben sich aus der Übersetzung der Reduktionsregeln des Λ^{let}-Kalküls durch die Abbildung $[\![\ ]\!]$. Bei der Übersetzung der Regeln (*lapp*) und (*lbeta*) aus Λ^{let} wird der umschließende Reduktionskontext C aufgrund der angepassten Definition der Reduktionsrelation (Definition 7.1.5) nicht mit übersetzt. Beispielsweise wird die linke Seite der (*lbeta*) Regel $C[((\lambda x.s)\ t)]$ aus Λ^{let} übersetzt zu $app(abs(x, s), t)$. Die rechten Seiten der Σ^{let}-Reduktionsregeln (*llet-in*) und (*llet-e*) enthalten Terme, die nicht wohlsortiert sind: In $letrec(\{\!\lfloor e_1 \vert e_2 \rfloor\!\}, s)$ und $letrec(\{\!\lfloor b(x, s), e_2 \vert e_1 \rfloor\!\}, t)$ sind jeweils zwei Umgebungsvariablen e_1 und e_2 der Sorte U in den `letrec`-Umgebungen enthalten, was nach der Definition des Funktionssymbols $\{\!\lfloor \cdot \vert \cdot \rfloor\!\} : M \to U \to U$ nicht zulässig ist. Allerdings ist dies hier nicht problematisch, da für die Berechnung von Überlappungen für Gabeldiagramme keine rechten Seiten von Reduktionsregeln unifiziert werden müssen.

Definition 7.1.4. Die Abbildungen 7.3 und 7.4 zeigen die eno-Reduktionsregeln des Σ^{let}-Kalküls.

Der Präfix "'e"' soll symbolisieren, dass die Reduktionsregeln in Σ^{let} nur eine Teilmenge der in Λ^{let} möglichen no-Reduktionen beschreiben. Die eno-Reduktionsregeln in Σ^{let} sind eingeschränkt auf Reduktionsketten der Länge ≤ 3.

Die Vereinigung der Regeln (*eno, llet-e i*) wird als (*eno, llet-e*), die von (*eno, cp-e i*) als (*eno, cp-e*), die von (*eno, lapp i.j*) als (*eno, lapp*), und die Vereinigung von (*eno, lbeta i.j*) wird als (*eno, lbeta*) bezeichnet, für $i = 1, 2, 3, j = 1, 2, 3, 4$.

(eno,llet-in) $letrec(e_1, letrec(e_2, s)) \longrightarrow letrec(\{\!|e_1|e_2|\!\}, s)$

(eno,llet-e 0) $letrec(\{\!|b(x, letrec(e_2, s))|e_1|\!\}, R^-(x))$
$\longrightarrow letrec(\{\!|b(x, s)), e_2|e_1|\!\}, R^-(x))$

(eno,llet-e 1) $letrec(\{\!|b(x, letrec(e_2, s)), b(x_1, R_1^-(x))|e_1|\!\}, R^-(x_1))$
$\longrightarrow letrec(\{\!|b(x, s)), b(x_1, R_1^-(x)), e_2|e_1|\!\}, R^-(x_1))$

(eno,llet-e 2) $letrec(\{\!|b(x, letrec(e_2, s)), b(x_1, R_1^-(x)), b(x_2, R_2^-(x_1))|e_1|\!\}, R^-(x_2))$
$\longrightarrow letrec(\{\!|b(x, s)), b(x_1, R_1^-(x)), b(x_2, R_2^-(x_1)), e_2|e_1|\!\}, R^-(x_2))$

(eno,llet-e 3) $letrec(\{\!|b(x, letrec(e_2, s)), b(x_1, R_1^-(x)), b(x_2, R_2^-(x_1)),$
$b(x_3, R_3^-(x_2))|e_1|\!\}, R^-(x_3))$
$\longrightarrow letrec(\{\!|b(x, s)), b(x_1, R_1^-(x)), b(x_2, R_2^-(x_1)),$
$b(x_3, R_3^-(x_2)), e_2|e_1|\!\}, R^-(x_3))$

(eno,cp-in) $letrec(\{\!|b(x_1, v), ch(b(x_2, x_1), b(x_n, x_{n-1}))|e|\!\}, R^-(x_n))$
$\longrightarrow letrec(\{\!|b(x_1, v), ch(b(x_2, x_1), b(x_n, x_{n-1}))|e|\!\}, R^-(v))$

(eno,cp-e 0) $letrec(\{\!|b(x_1, v), ch(b(x_2, x_1), b(x_n, x_{n-1})),$
$b(y, R_0^-(app(x_n, s)))|e|\!\}, R^-(y))$
$\longrightarrow letrec(\{\!|b(x_1, v), ch(b(x_2, x_1), b(x_n, x_{n-1})),$
$b(y, R_0^-(app(v, s)))|e|\!\}, R^-(y))$

(eno,cp-e 1) $letrec(\{\!|b(x_1, v), ch(b(x_2, x_1), b(x_n, x_{n-1})), b(y, R_0^-(app(x_n, s))),$
$b(y_1, R_1^-(y))|e|\!\}, R^-(y_1))$
$\longrightarrow letrec(\{\!|b(x_1, v), ch(b(x_2, x_1), b(x_n, x_{n-1})), b(y, R_0^-(app(v, s))),$
$b(y_1, R_1^-(y))|e|\!\}, R^-(y_1))$

(eno,cp-e 2) $letrec(\{\!|b(x_1, v), ch(b(x_2, x_1), b(x_n, x_{n-1})), b(y, R_0^-(app(x_n, s))),$
$b(y_1, R_1^-(y)), b(y_2, R_2^-(y_1))|e|\!\}, R^-(y_2))$
$\longrightarrow letrec(\{\!|b(x_1, v), ch(b(x_2, x_1), b(x_n, x_{n-1})), b(y, R_0^-(app(v, s))),$
$b(y_1, R_1^-(y)), b(y_2, R_2^-(y_1))|e|\!\}, R^-(y_2))$

(eno,cp-e 3) $letrec(\{\!|b(x_1, v), ch(b(x_2, x_1), b(x_n, x_{n-1})), b(y, R_0^-(app(x_n, s))),$
$b(y_1, R_1^-(y)), b(y_2, R_2^-(y_1)), b(y_3, R_3^-(y_2))|e|\!\}, R^-(y_3))$
$\longrightarrow letrec(\{\!|b(x_1, v), ch(b(x_2, x_1), b(x_n, x_{n-1})), b(y, R_0^-(app(v, s))),$
$b(y_1, R_1^-(y)), b(y_2, R_2^-(y_1)), b(y_3, R_3^-(y_2))|e|\!\}, R^-(y_3))$

Figure 7.3. Die eno-Reduktionsregeln des Σ^{let}-Kalküls (Teil 1).

(eno,lapp 1) $R^-(app(letrec(e,s),t)) \longrightarrow R^-(letrec(e,app(s,t)))$

(eno,lapp 2) $letrec(e_1, app(letrec(e_2,s),t)) \longrightarrow letrec(e_1, letrec(e_2, app(s,t)))$

(eno,lapp 3.0)
$$letrec(\{\!|b(x,app(letrec(e_2,s),t))|e_1|\!\}, R^-(x)) \longrightarrow letrec(\{\!|b(x,letrec(e_2,app(s,t))|e_1|\!\}, R^-(x))$$

(eno,lapp 3.1)
$$letrec(\{\!|b(x,app(letrec(e_2,s),t)), b(x_1,R_1^-(x))|e_1|\!\}, R^-(x_1)) \longrightarrow letrec(\{\!|b(x,letrec(e_2,app(s,t)), b(x_1,R_1^-(x))|e_1|\!\}, R^-(x_1))$$

(eno,lapp 3.2)
$$letrec(\{\!|b(x,app(letrec(e_2,s),t)), b(x_1,R_1^-(x)), b(x_2,R_2^-(x_1))|e_1|\!\}, R^-(x_2)) \longrightarrow letrec(\{\!|b(x,letrec(e_2,app(s,t)), b(x_1,R_1^-(x)), b(x_2,R_2^-(x_1))|e_1|\!\}, R^-(x_2))$$

(eno,lapp 3.3)
$$letrec(\{\!|b(x,app(letrec(e_2,s),t)), b(x_1,R_1^-(x)), b(x_2,R_2^-(x_1)), b(x_3,R_3^-(x_2))|e_1|\!\}, R^-(x_3)) \longrightarrow letrec(\{\!|b(x,letrec(e_2,app(s,t)), b(x_1,R_1^-(x)), b(x_2,R_2^-(x_1)), b(x_3,R_3^-(x_2))|e_1|\!\}, R^-(x_3))$$

(eno,lbeta 1) $R^-(app(abs(x,s),t)) \longrightarrow R^-(letrec(\{\!|b(x,t)|\emptyset|\!\}, s))$

(eno,lbeta 2) $letrec(e, (app(abs(x,s),t))) \longrightarrow letrec(e, letrec(\{\!|b(x,t)|\emptyset|\!\}, s))$

(eno,lbeta 3.0)
$$letrec(\{\!|b(y,app(abs(x,s),t))|e|\!\}, R^-(y)) \longrightarrow letrec(\{\!|b(y,letrec(\{\!|b(x,t)|\emptyset|\!\}, s)|e|\!\}, R^-(y))$$

(eno,lbeta 3.1)
$$letrec(\{\!|b(y,app(abs(x,s),t)), b(y_1,R_1^-(y))|e|\!\}, R^-(y_1)) \longrightarrow letrec(\{\!|b(y,letrec(\{\!|b(x,t)|\emptyset|\!\}, s), b(y_1,R_1^-(y))|e|\!\}, R^-(y_1))$$

(eno,lbeta 3.2)
$$letrec(\{\!|b(y,app(abs(x,s),t)), b(y_1,R_1^-(y)), b(y_2,R_2^-(y_1))|e|\!\}, R^-(y_2)) \longrightarrow letrec(\{\!|b(y,letrec(\{\!|b(x,t)|\emptyset|\!\}, s), b(y_1,R_1^-(y)), b(y_2,R_2^-(y_1))|e|\!\}, R^-(y_2))$$

(eno,lbeta 3.3)
$$letrec(\{\!|b(y,app(abs(x,s),t)), b(y_1,R_1^-(y)), b(y_2,R_2^-(y_1)), b(y_3,R_3^-(y_2))|e|\!\}, R^-(y_3)) \longrightarrow letrec(\{\!|b(y,letrec(\{\!|b(x,t)|\emptyset|\!\}, s), b(y_1,R_1^-(y)), b(y_2,R_2^-(y_1)), b(y_3,R_3^-(y_2))|e|\!\}, R^-(y_3))$$

Figure 7.4. Die eno-Reduktionsregeln des Σ^{let}-Kalküls (Teil 2).

Die eno-Reduktionsregeln des Σ^{let}-Kalküls ergeben sich durch die Übersetzung der Λ^{let}-Ausdrücke, die durch die verschiedenen Fälle der Definition der Normalordnungsreduktion in Λ^{let} bestimmt sind (Definition 2.4.2). Die eno-Reduktionsregeln in Σ^{let} decken nur einen Teil der möglichen no-Reduktionen aus Λ^{let} ab. In Λ^{let} können no-Redexe mit Reduktionsketten beliebiger Länge der Form $\{x_i = R_i^-[x_{i-1}]\}_{i=m}^n$ vorkommen. Solche Reduktionsketten beliebiger Länge können in Σ^{let} nicht dargestellt werden. Deshalb werden im Σ^{let}-Kalkül nur Reduktionsregeln mit Reduktionsketten einer Länge ≤ 3 für die Berechnung der Überlappungen verwendet.

Definition 7.1.5. Eine Σ^{let}-*Reduktion* $\rightarrow$ ist folgendermaßen definiert: $s \rightarrow t$, gdw. es eine Reduktionsregel $l \rightarrow r$, eine Substitution σ und eine Position $p \in Pos(s)$ gibt, so dass $s|_p = \sigma(l)$ und $t = s[\sigma(r)]_p$.

Eine *eno*-Σ^{let}-*Reduktion* $\xrightarrow{eno}$ ist definiert durch: $s \xrightarrow{eno} t$, gdw. es eine eno-Reduktionsregel $l \rightarrow r$ und eine Substitution σ gibt, so dass $s = \sigma(l)$ und $t = \sigma(r)$ gilt.

Eine iX-Σ^{let}-*Reduktion* $\xrightarrow{iX}$ für $X \in \{\mathcal{C}, \mathcal{S}, \mathcal{R}\}$ ist definiert durch: $s \xrightarrow{iX} t$, gdw. es eine Reduktionsregel $l \rightarrow r$, eine Substitution σ und eine Position $p \in Pos(s)$ gibt, so dass $[\![s[\Box]_p]\!]^-$ ein Kontext der Kontextklasse X ist, $s|_p = \sigma(l)$ sowie $t = s[\sigma(r)]_p$ gilt und wenn s eno-reduzierbar zu t' ist ($s \xrightarrow{eno} t'$), dann gilt $t' \neq t$.

7.2 Überlappungen von Reduktionsregeln in Σ^{let}

Wir haben folgende Situation für Gabeldiagramme in Σ^{let}:

$$\begin{array}{ccc} s & \xrightarrow{iS, l_2 \rightarrow r_2} & t_2 \\ {\scriptstyle eno, l_1 \rightarrow r_1}\downarrow & & \\ t_1 & & \end{array}$$

wobei $l_1 \rightarrow s_1$ eine eno-Reduktionsregel ist und es eine Substitution σ_1 gibt, so dass $s = \sigma_1 l_1$ und $t_1 = \sigma r$ gilt. Die Reduktion $\xrightarrow{iS, l_2 \rightarrow r_2}$ ist eine interne Reduktion in einem Oberflächenkontext. D.h. $l_2 \rightarrow r_2$ ist eine Reduktionsregel und es gibt eine Position $p \in Pos(s)$ und eine Substitution σ_2, so dass $s|_p = \sigma_2 l_2$ und $[\![s[\Box]_p]\!]^-$ ein Oberflächenkontext ist. Dann ist $t_2 = s[\sigma_2 r_2]_p$ und es gilt $t_1 \neq t_2$.

Abhängig von der Position p (bzw. von der Form des Subterms $\sigma(l_1|_p)$) lassen sich verschiedene Fälle unterscheiden.

Fall 1.1 Der interne-Redex $\sigma_2 l_2$ überlappt nicht direkt mit l_1, sondern ist in σ_1 enthalten. D.h. es gilt $p = q_1 q_2$, so dass q_1 eine Variablen-Position von l_1 ist. Dann hat der eno-Redex $\sigma_1 l_1$ die Form:

Diese Überlappung wird als *Schachtelung* der Redexe bezeichnet: Der interne Redex kommt innerhalb einer Variablen des eno-Redex vor. Für diesen Fall kann das Gabeldiagramm immer auf die gleiche einfache Art geschlossen werden (siehe Abbildung 7.5 und Lemma 2.6.7).

Figure 7.5. Schachtelung von Redexen. Ist $\sigma_1 r_1[\Box]_{p'}$ ein Reduktionskontext, dann gilt $A = eno$, sonst ist $A = iS$.

Zum Schließen des Gabeldiagramms ist auf der rechten Seite der Gabel nach der Reduktion des internen Redex $\sigma_2 l_2$ eine eno-Reduktion von $\sigma_1 l_1$ notwendig. Diese ist möglich, da $\sigma_1 l_1$ ein eno-Redex ist und die Reduktion des internen Redex den eno-Redex nicht zerstört, weil sich die beiden Redexe keine Funktionssymbole teilen. Der resultierende Ausdruck ist $r = \sigma_1 r_1[\sigma_2 r_2]_{p'}$ ($p' \in Pos(\sigma r_1)$ ist die Position, des internen Redex nach der eno-Reduktion). Nach der eno-Reduktion auf der linken Seite der Gabel zu t_1, kommen folgende Fälle in Betracht, um t_1 (durch eine interne oder eine eno-Reduktion) zu r zu reduzieren: Sei $p' \in Pos(\sigma_1 r_1)$ die Position, so

dass $\sigma_1(r_1|_{p'}) = \sigma_2 l_2$ gilt. Wenn $[\![\sigma_1 r_1[\Box]_{p'}]\!]^-$ ein Reduktionskontext ist, dann kann $t_1 = \sigma_1 r_1[\sigma_2 l_2]_{p'}$ durch eine eno-Reduktion von $\sigma_2 l_2$ zu r reduziert werden. Ist $\sigma_1 r_1[\Box]_{p'}$ kein Reduktionskontext, kann t_1 durch eine iS-Reduktion von $\sigma_2 l_2$ zu r reduziert werden.

Fall 1.2. Die beiden linken Seiten der Reduktionsregeln l_1 und l_2 überlappen, d.h. $p \in Pos(l_1)$ und $l_1|_p$ ist keine Variable und $\sigma_1(l_1|_p) = \sigma_2 l_2$. In diesem Fall ist σl_2 ein nicht trivialer Subterm von σl_1 an der Position p. Dann hat $\sigma_1 l_1$ folgende Form:

Dieser Fall ist eine echte so genannte *kritische Überlappung* der beiden Redexe. Die beiden Redexe teilen sich mindestens ein Funktionssymbol. Dieser Fall der Überlappung ist eine Instanz eines *kritischen Paares*. Informell kann man ein kritisches Paar verstehen als das Resultat der Unifikation der linken Seite einer (internen) Reduktionsregel mit einem Nicht-Variablen-Subterm einer linken Seite einer anderen (eno) Reduktionsregel.

Definition 7.2.1. Seien $l_i \to r_i, i = 1, 2$ zwei Reduktionsregeln mit umbenannten Variablen, so dass $Var(l_1, r_1) \cap Var(l_2, r_2) = \emptyset$. Sei $p \in Pos(l_1)$ eine Position in l_1, so dass $l_1|_p$ keine Variable ist und C eine vollständige Menge von Unifikatoren für das Unifikationsproblem $l_1|_p =^? l_2$. Dann ist die *Menge der kritischen Paare* definiert durch $\{\langle \sigma r_1, \sigma(l_1)[\sigma r_2]_p)\rangle \mid \sigma \in C\}$. Elemente aus der Menge der kritischen Paare, können durch das Diagramm

$$\begin{array}{ccc} \sigma l_1 & \xrightarrow{l_2 \to r_2} & \sigma(l_1)[\sigma r_2]_p \\ {\scriptstyle l_1 \to r_1}\downarrow & & \\ \sigma r_1 & & \end{array}$$

dargestellt werden. Wenn zwei Reduktionsregeln eine Menge von kritischen Paaren erzeugen, sagt man, die beiden Reduktionsregeln *überlappen*.

Für eine eno-Reduktionsregel $l_1 \to r_1$ und eine Reduktionsregel $l_2 \to r_2$ und alle Positionen $p \in Pos(l_1)$, so dass p keine Position im Rumpf einer Abstraktion ist, beschreiben die jeweiligen Mengen kritischer Paare alle Überlappungen zwischen der eno-Reduktion, die durch $l_1 \to r_1$ gegeben ist und der iS-Reduktion, die durch $l_2 \to r_2$ gegeben ist.

Zur Berechnung aller Überlappungen einer iS-Reduktion mit eno-Reduktionen geht man folgendermaßen vor: Für alle eno-Reduktionsregeln $l_1 \to r_1$ und eine Reduktionsregel $l_2 \to r_2$ (die interne Reduktion) und für alle Positionen $p \in Pos(l_1)$, so

dass $l_1|_p$ keine Variable und kein Subterm im Rumpf einer Abstraktion ist, werden variablendisjunkte Varianten von $l_1|_p$ und l_2 unifiziert. Die so bestimmten Mengen der kritischen Paare beschreiben alle Überlappungen der internen Reduktion in einem Oberflächenkontext, die durch die $l_2 \to r_2$ bestimmt ist, mit eno-Reduktionen.

Zur Unifikation wird die Unifikationsprozedur verwendet, die sich aus der Vereinigung

- der Unifikationsregeln für Terme mit Sorten (aus Definition 3.4.13),
- der Unifikationsregeln für Terme mit Kontextvariablen (Definition 6.3.6) und
- der Unifikationsregeln für Terme mit dem links-kommutativen Funktionssymbol $\{\!|\cdot|\cdot|\!\}$ und dem Funktionssymbol *ch* für Variablenketten beliebiger Länge (Definitionen 5.2.3, 5.2.9 und 5.2.10)

ergibt. Die Unifikationsprozeduren zur Unifikation von Termen mit Sorten, Termen mit Kontextvariablen und Termen mit einem links-kommutativen Funktionssymbol sind vollständig. Von der Unifikationsprozedur für Terme mit Variablenketten wird angenommen, dass sie überlappungs-vollständig ist[1]. D.h. die Menge der kritischen Paare kann durch die Unifikationsprozedur, die sich aus der Vereinigung der Regeln der einzelnen Unifikationsprozeduren ergibt, berechnet werden.

Aufgrund der Eingeschränktheit der eno-Reduktion in Σ^{let} gegenüber der no-Reduktion in Λ^{let} wird mit dieser Methode nur eine Teilmenge aller Überlappung berechnet, die in Λ^{let} möglich sind.

[1] D.h. Es wird angenommen, dass alle Unifikatoren, die zur Berechnung aller Überlappung notwendig sind, durch die Unifikationsprozedur berechnet werden.

8 Implementierung und Ergebnisse

Im Rahmen der vorliegenden Arbeit ist eine Implementierung entstanden, die zu einer gegebenen iS-Reduktion eine vollständige Menge von Gabeldiagrammen durch die Überlappung mit eno-Reduktionen (eingeschränkten no-Reduktionen, siehe Definition 7.1.4) berechnet. Der Kern des Programms, das in der funktionalen Programmiersprache Haskell entwickelt wurde, besteht aus einem Unifikationsalgorithmus zur Unifikation von Termen mit Sorten, Kontextvariablen mit Sorten, einem links-kommutativen Funktionssymbol und Variablenketten beliebiger Länge.

Der Quelltext des Haskell-Programms ist unter folgender URL abrufbar:

`http://www.xylon.de/rau/`.

Zur Ausführung des Programms wird der Haskell Compiler *GHC* in einer Version $\geq$ 6.10.1 benötigt, der unter der URL `http://www.haskell.org/ghc/` erhältlich ist.
Wird das Programm compiliert oder in den GHC-Interpreter geladen, muss als Option `--fglasgow-exts` angegeben werden.

Die Implementierung ist zu Beginn der Bearbeitung des Themas in einer explorativen Phase entstanden. Zu diesem Zeitpunkt waren die verschiedenen Konzepte, die es bei der Unifikation zu berücksichtigen gilt, teilweise noch in anderer Form formuliert. Beispielsweise wurden die in Λ^{let}-Reduktionsregeln (Definition 2.3.1) enthaltenen Variablen $s, t, x, y, Env, R^-, R, C$ und v als Metavariablen bezeichnet. Der Begriff der Metavariablen war informell definiert: Für eine bestimmte Metavariable können Ausdrücke oder andere Metavariablen eines entsprechenden Typs eingesetzt werden. D.h. für s, t können beliebige Ausdrücke, für x, y Variablen, für Env `letrec`-Umgebungen, für R^-, R, C beliebige Kontexte einer entsprechenden Kontextklasse und für v Abstraktionen eingesetzt werden. Das Konzept der Metavariablen ist dem Konzept der Variablen mit Sorten sehr ähnlich. So entsprechen die Metavariablen s, t Variablen der Sorte T, Metavariablen x, y entsprechen Variablen der Sorte V, usw. Allerdings ist in der Literatur, die sich mit Unifikationstheorien befasst, von Termen und Variablen mit Sorten und nicht von Metavariablen die Rede. In der frühen Phase der Bearbeitung wurde das Programm basierend auf dem Begriff von Metavariablen entwickelt.

Des weiteren verwendet das Programm zur Darstellung der zu unifizierenden Objekte Ausdrücke, die in ihrer Gestalt eher der Syntax von Λ^{let}-Ausdrücken als der

Syntax von Σ^{let}-Termen entsprechen. Ausdrücke und Terme sind verwandte Begriffe, allerdings hat das `letrec`-Konstrukt in Λ^{let} keine feste Stelligkeit. Im theoretischen Teil dieser Arbeit wurden deshalb die Funktionssymbole *letrec* und $\{\!|\cdot|\cdot|\!\}$ der Signatur Σ^{let} eingeführt, die über eine feste Stelligkeit verfügen. In der Literatur wird Unifikation im Rahmen von Termen und nicht von Ausdrücken behandelt. Im Programm werden die Ausdrücke bei der Unifikation im wesentlichen wie Terme behandelt, allerdings werden keine expliziten Terme sondern Ausdrücke unifiziert.

Die Regeln des Programms, die zur Unifikation der in Λ^{let}-Reduktionsregeln enthaltenen Konstrukten verwendet werden, sind durch eine intuitive Herangehensweise entstanden. Die Überlegungen, die zum Entwurf der Unifikationsregeln angestellt wurden, waren immer von der Art: Gegeben eine Unifikationsgleichung zwischen Ausdrücken, welche Möglichkeiten bestehen, die Ausdrücke zu unifizieren. Für die Gleichung

$$(\texttt{letrec}\ \{Env_1, x_1 = s_1\}\ \texttt{in}\ s) =^? (\texttt{letrec}\ \{Env_2, y_1 = t_1\}\ \texttt{in}\ t)$$

besteht beispielsweise nur die Möglichkeit die beiden `letrec`-Umgebungen untereinander sowie s mit t zu unifizieren. Diese Unifikationsmöglichkeiten entsprechen der Anwendung der Regel *Decomposition*, wenn Terme unifiziert werden. Da `letrec` in Λ^{let} ebenso wenig ein Funktionssymbol ist, wie die Konstrukte Applikation und Abstraktion, wird bei der Unifikation von Ausdrücken für alle möglichen Kombinationen von Konstrukten in Unifikationsgleichungen eine separate Regel benötigt, die der *Decomposition*-Regel für Terme entspricht:

Dec Abs
$\{(\lambda x.s) =^? (\lambda y.t)\} \uplus P \Rightarrow \{x =^? y, s =^? t\} \cup P$
Dec App
$\{(f\ s) =^? (g\ t)\} \uplus P \Rightarrow \{f =^? g, s =^? t\} \cup P$
Dec `letrec`
$\{(\texttt{letrec}\ \{Env_1\}\ \texttt{in}\ s) =^? (\texttt{letrec}\ \{Env_2\}\ \texttt{in}\ t)\} \uplus P \Rightarrow$
$\quad \{Env_1 =^? Env_2, s =^? t\} \cup P$
Symbol Clash `letrec`**/App**
$\{(\texttt{letrec}\ \{Env\}\ \texttt{in}\ t) =^? (\lambda x.s)\} \uplus P \Longrightarrow \bot$
...

Das intuitive Vorgehen beim Entwurf der Unifikationsregeln für das Programm, die Verwendung von Metavariablen und die syntaktische Repräsentation in Form von Ausdrücken resultiert im wesentlichen in Regeln, die den Unifikationsregeln entsprechen, die im theoretischen Teil dieser Arbeit beschrieben werden. Allerdings weicht die Formulierung der einzelnen Regeln des Programms teilweise von Formulierungen ab, die in der Unifikationsliteratur gebräuchlich sind. Aus diesem Grund war es nach der Entwicklung des Programms schwierig, Resultate aus dem

Bereich der Unifikationstheorien zum Beweis der Terminierung und Vollständigkeit des programmierten Unifikationsalgorithmus zu verwenden. Da diese beiden Eigenschaften aber zentral für die Berechnung aller möglichen Überlappungen für Gabeldiagramme ist, wurde in der zweiten Phase der Bearbeitung des Themas ein stärkerer Bezug zu existierenden Theorien hergestellt, um formale Aussagen über die Eigenschaften des Unifikationsalgorithmus zu treffen. Daraus resultiert eine gewisse Diskrepanz zwischen dem Programm und den vorgestellten Theorien, die aufgrund der Beschränkung des Bearbeitungszeitraums im Rahmen dieser Arbeit nicht mehr behoben werden konnte. Die Anpassung des Programms an die vorgestellten Unifikationstheorien muss in zukünftigen Arbeiten erfolgen. Die Diskrepanz bezieht sich nur auf die Formulierung der Unifikationsregeln: Die Unifikationsregeln des Programms sind nicht an allen Stellen exakt so formuliert, wie die Unifikationsregeln der vorgestellten Theorien. Die Unifikationsmöglichkeiten, die durch die Regeln des Programms beschrieben werden, decken trotzdem alle Unifikationsmöglichkeiten ab, die durch die Regeln der verschiedenen Theorien gegeben sind. Um diese Behauptung zu stützen, geben wir eine Reihe von Beispielaufrufen wichtiger Funktionen des Programms an, die verwendete Unifikationsregeln illustrieren, und vergleichen sie mit den Ergebnissen, die sich aus den Regeln der Unifikationstheorien ergeben.

8.1 Implementierung

Zuerst werden kurz die wichtigsten Datenstrukturen vorgestellt, die im Programm zur Darstellung von Ausdrücken verwendet werden. Dann werden die zentralen Funktionen des Programms besprochen, die Unifikationsgleichungen transformieren, kritische Paare und eine vollständige Menge von Gabeldiagrammen berechnen.

8.1.1 Datenstrukturen für Ausdrücke

Folgende Datenstrukturen werden zur Darstellung von Metavariablen im Programm verwendet:

data $Tmv = Tmv\ VName$
data $Vmv = Vmv\ VName$
data $Wmv = Wmv\ VName$
data $Emv = Emv\ VName$
data $Cmv = Cmv\ VName\ CType$.

Alle Metavariablen haben einen Namen *VName*, der durch ein Paar des Typs $(String, Integer)$ dargestellt wird. Der String bezeichnet den Namen der Variablen,

der Integer-Wert wird zum einfachen Umbenennen von Variablen im Programm verwendet. Wir geben die Bedeutung der einzelnen Datenstrukturen an.

- Die Datenstruktur *Tmv* (abgekürzt für Term bzw. Ausdrucks-Metavariable) repräsentiert Metavariablen für Ausdrücke. Die entsprechenden Variablen in Σ^{let} sind von der Sorte T.
- Die Datenstruktur *Vmv* (Variablen-Metavariable) repräsentiert Metavariablen für Variablen. Die entsprechenden Variablen in Σ^{let} sind von der Sorte V.
- Die Datenstruktur *Wmv* (Werte-Metavariable) repräsentiert Metavariablen für Abstraktionen. Die entsprechenden Variablen in Σ^{let} sind von der Sorte *App*.
- Die Datenstruktur *Emv* (Umgebungs-Metavariable) repräsentiert Metavariablen für `letrec`-Umgebungen. Die entsprechenden Variablen in Σ^{let} sind von der Sorte U.
- Die Datenstruktur *Cmv* (Kontext-Metavariable) repräsentiert Metavariablen für Kontexte. Der Konstruktor der Datenstruktur verfügt über ein zusätzliches Argument *CType*, das angibt, zu welcher Kontextklasse die Kontext-Metavariable gehört:

  $$\textbf{data}\ \mathit{CType} = C \mid S \mid R \mid \mathit{Rw}.$$

  Eine Kontextvariable mit *CType C* entspricht einer Σ^{let}-Kontextvariablen der Sorte $T \to T$. Es gelten analoge Entsprechungen von Kontext-Metavariablen mit einem anderen *CType* zu den jeweiligen Σ^{let}-Kontextvariablen.

Ausdrücke werden im Programm durch die Datenstruktur

$$\begin{aligned}\textbf{data}\ \mathit{MAusdr} = {} & \mathit{MTtmv}\ \mathit{Tmv}\\ \mid {} & \mathit{MTvmv}\ \mathit{Vmv}\\ \mid {} & \mathit{MTwmv}\ \mathit{Wmv}\\ \mid {} & \mathit{MTC}\ \mathit{Cmv}\ \mathit{MAusdr}\\ \mid {} & \mathit{MApp}\ \mathit{MAusdr}\ \mathit{MAusdr}\\ \mid {} & \mathit{MAbs}\ \mathit{Vmv}\ \mathit{MAusdr}\\ \mid {} & \mathit{MLet}\ \mathit{Env}\ \mathit{MAusdr}\end{aligned}$$

dargestellt. Der Präfix *M* soll verdeutlichen, dass es sich um Ausdrücke mit Metavariablen handelt. Ein Ausdruck im Programm ist entweder eine bestimmte Metavariable, repräsentiert durch die Konstruktoren *MTtmv, MTvmv* und *MTwmv*, die jeweils eine entsprechende Metavariable als Argument erwarten, oder eine Kontext-Metavariable angewandt auf einen Ausdruck (*MTC*). Des Weiteren kann es sich bei einem Ausdruck um eine Applikation (*MApp*), eine Abstraktion (*MAbs*) oder einen `letrec`-Ausdruck (*MLet*) handeln. Der `letrec` Datenkonstruktor erwartet als zweites Argument eine Datenstruktur des Typs *Env*. `letrec`-Umgebungen sind im Programm als drei-Tupel definiert:

$$\textbf{type}\ \mathit{Env} = ([\mathit{Emv}], [\mathit{Chain}], [\mathit{Bind}])$$

bestehend aus einer Liste von Umgebungs-Metavariablen, einer Liste von Variablenketten und einer Liste von `letrec`-Bindungen. `letrec`-Umgebungen enthalten zwar eine Liste von Umgebungs-Metavariablen, der Unifikationsalgorithmus des Programms unifiziert aber nur solche Umgebungen, die maximal eine Umgebungs-Metavariable enthalten (analog zu der Beschränkung, dass Umgebungen $\{\!|\ \cdot\ |\ \cdot\ |\!\}$ in Σ^{let} nur eine Variable der Sorte U enthalten dürfen).

Im Programm sind `letrec`-Bindungen definiert durch

$$\textbf{data}\ \mathit{Bind} = \mathit{Vmv} :=: \mathit{MAusdr}.$$

Der infix Konstruktor :=: bindet einen Ausdruck an eine Variablen-Metavariable.

Variablenketten werden durch die Datenstruktur

$$\textbf{data}\ \mathit{Chain} = \mathit{Chain}\ \mathit{CHmv}\ \mathit{Bind}\ [(\mathit{BindPos}, \mathit{Bind})]\ \mathit{Bind}$$

dargestellt. Als zweites und viertes Argument erwartet der Kettenkonstruktor jeweils eine `letrec`-Bindung, die Anfangs- und Endbindung der Variablenkette symbolisieren. Das erste Argument des Konstruktors ist eine Ketten-Metavariable, die zusammen mit dem dritten Argument, einer Liste von Tupeln bestehend aus einer Position (A,E oder M) und einer Bindung, dazu dienen, sich abgespaltene Bindungen und Aufteilungen der Kette zu merken (siehe Kapitel 5). Ketten-Metavariablen sind analog zu den anderen Metavariablen definiert:

$$\textbf{data}\ \mathit{CHmv} = \mathit{CHmv}\ \mathit{VName}.$$

Für eine Ketten-Metavariable können Variablenketten oder andere Ketten-Metavariablen eingesetzt werden.

Reduktionsregeln werden im Programm durch Paare $(\mathit{MAusdr}, \mathit{MAusdr})$ repräsentiert. Die noe-Reduktionsregeln und die Reduktionsregeln, die zur Berechnung aller Überlappungen verwendet werden, entsprechen den in Kapitel 7 definierten Regeln des Σ^{let}-Kalküls (Definition 7.1.2 und Definition 7.1.4). Im Haskell-Interpreter werden die Regeln durch den Aufruf der Funktionen *norules* bzw. *isrules* angezeigt.

Die meisten Datenstrukturen verfügen über eine Druckfunktion für eine besser lesbare Darstellung im Haskell-Interpreter, wie beispielsweise

```
> MLet ([Emv ("E", 1)],[],
        [Vmv ("x", 1) :=: MApp (MTtmv (Tmv ("s", 1)))
                               (MTvmv (Vmv ("x", 1)))])
       (MTvmv (Vmv ("v", 1)))
> (letrec ([E1],[],[x1=@(s1,x1)]) in v1).
```

Nach obigem Schema wird die Auswertung eines Haskell-Ausdrucks im Interpreter präsentiert: Hinter dem ersten `>` stehet der auszuwertenden Ausdruck, hinter dem zweiten `>` das Resultat der Auswertung.

8.1.2 Funktionen

Die Funktion *solve* mit dem Typ

$$solve :: GL \rightarrow \ Either\ [[GL]]\ Subst$$

erhält als Argument eine Unifikationsgleichung zwischen Ausdrücken (repräsentiert durch die Datenstruktur *GL*) und wendet eine Transformation auf die Unifikationsgleichung an. Der Rückgabewert ist entweder eine Liste von Listen von Unifikationsgleichungen (die vollständige Menge von Alternativen, siehe Definition 3.4.6) oder eine Substitution, falls die eingegebene Gleichung in gelöster Form ist. Für die eingegebene Unifikationsgleichung müssen die Bedingungen eingeschränkter Unifikationsprobleme (Definition 6.3.4) gelten. Insbesondere darf jede Kontext-Metavariable nur einmal vorkommen. Außerdem darf jede `letrec`-Umgebung in der Gleichung maximal eine Variablenkette enthalten, es sei denn die Variablenketten sind durch Aufspaltung aus einer Ursprungskette entstanden (siehe Kapitel 5). Die Ausführung der Funktion *solve* ist in eine Zustands-Monade gekapselt, um auf einfache und modulare Weise neue Variablennamen zu erzeugen, die von manchen Unifikationsregeln benötigt werden. Der tatsächliche Typ von *solve* ist deshalb

$$solve :: GL \rightarrow \ State\ Integer\ (Either\ [[GL]]\ Subst).$$

D.h. *solve* gibt einen Zustand zurück, der aus einer Integer-Zahl, die für die Generierung neuer Variablennamen verwendet wird, und einer entweder der vollständigen Menge von Alternativen oder einer Substitution besteht.

Wir geben eine Reihe von Beispielen, die illustrieren, dass die Unifikationsregeln des Programms kompatibel mit den Unifikationsregeln der vorgestellten Theorien sind.

Betrachte die Gleichung $\{app(s_1, t_1) =^? app(s_2, t_2)\}$ in Σ^{let} die durch einen *Decomposition*-Schritt (aus Definition 3.4.13) transformiert wird zu $\{s_1 =^? t_1, s_2 =^? t_2\}$. Ein Aufruf von *solve* mit dem Unifikationsproblem als Argument sieht folgendermaßen aus:

```
> run $ solve (MApp (MTtmv (Tmv ("s", 1))) (MTtmv (Tmv ("t", 1))) =?
               MApp (MTtmv (Tmv ("s", 2))) (MTtmv (Tmv ("t", 2))))
> Left [[s1 =? s2,t1 =? t2]].
```

Dabei ist *run* eine Hilfsfunktion, um die Zustands-Monade für *solve* zu initialisieren. Der Datenkonstruktor `Left` signalisiert, dass der Rückgabewert die vollständige Menge (Liste) von Alternativen ist, die in diesem Fall nur aus einer Liste von Unifikationsgleichungen besteht.

Das Σ^{let}-Unifikationsproblem $\{s_T =^? letrec(e, t)\}$ ist in gelöster, wohlsortierter Form. Der Aufruf von *solve* für diese Problem resultiert in einer Substitution, die den `letrec`-Ausdruck an die Variable bindet. Die Rückgabe einer Substitution wird durch den Datenkonstruktor `Right` signalisiert.

```
> run $ solve ((MTtmv (Tmv ("s", 1))) =?
               (MLet ([Emv ("E", 1)],[],[]) (MTtmv (Tmv ("t", 1)))))
> Right Tmv ("s",1) -> (letrec ([E1],[],[]) in t1)
```

Auf das Σ^{let}-Unifikationsproblem $\{x_v =^? letrec(e,t)\}$ besitzt keine Lösung (*Sorted Fail Fun* aus Definition 3.4.13). Analog dazu der Aufruf von *solve*:

```
> run $ solve ((MTvmv (Vmv ("x", 1))) =?
               (MLet ([Emv ("E", 1)],[],[]) (MTtmv (Tmv ("t", 1)))))
> Left [],
```

wobei der `Left[]` der Wert ist, der das Unifikationsproblem repräsentiert dass keine Lösung besitzt.

Das Σ^{let}-Unifikationsproblem $\{s =^? letrec(e,s)\}$ resultiert in einem *Occurs Check*, dazu passend haben wir

```
> run $ solve ((MTtmv (Tmv ("s", 1))) =?
               (MLet ([Emv ("E", 1)],[],[]) (MTtmv (Tmv ("s", 1)))))
> Right ***
  Exception: Occurs: Tmv ("s",1) in (letrec ([E1],[],[]) in s1).
```

Die vollständige Menge von Alternativen für das Σ^{let}-Unifikationsproblem

$$\{\{\!|b(x_1,s_1),b(x_2,s_2)|e_1|\!\} =^? \{\!|b(y_1,t_1),b(y_2,t_2)|e_2|\!\}\}$$

ist durch die Transformationsmöglichkeiten von *U-Decomposition* (aus Definition 4.4.2) gegeben:

$$\{\{\{\!|b(x_2,s_2)|e_1|\!\} =^? \{\!|b(y_1,t_1)|N|\!\}\},\{\{\!|b(x_1,s_1)|N|\!\} =^? \{\!|b(y_2,t_2)|e_2|\!\}\}\}$$

Im Programm wird eine Variante der *U-Decomposition*-Regel verwendet, wie an folgendem Beispiel zu sehen ist.

```
> run $ solve
  (([Emv ("E", 1)],[],[Vmv ("x", 1) :=: MTtmv (Tmv ("s", 1)),
                        Vmv ("x", 2) :=: MTtmv (Tmv ("s", 2))]) =?
   ([Emv ("E", 2)],[],[Vmv ("y", 1) :=: MTtmv (Tmv ("t", 1)),
                        Vmv ("y", 2) :=: MTtmv (Tmv ("t", 2))]))
> Left
1 [[x1=s1 =? y1=t1, ([E1],[],[x2=s2]) =? ([E2],[],[y2=t2])],
2  [x1=s1 =? y2=t2, ([E1],[],[x2=s2]) =? ([E2],[],[y1=t1])],
3  [([E2],[],[]) =? ([N],[],[x1=s1]),
    ([E1],[],[x2=s2]) =? ([N],[],[y1=t1,y2=t2])]]
```

Die Liste der vollständigen Alternativen enthält ein zusätzliches Unifikationsproblem (Zeile 2), das durch die Σ^{let}-Unifikationsregeln erst nach einer Folge von Transformationen berechnet wird (siehe Transformation in semi-gelöste Form in Lemma 4.4.4). Die im Programm verwendete Unifikationsregel zieht diese Transformationsmöglichkeit vor. Das Unifikationsproblem in Zeile 3 ist entsprechend angepasst, um die Berechnung einiger redundanter Unifikatoren zu vermeiden.

Für das Σ^{let}-Unifikationsproblem $\{\{[\![b(x_1, s_1)|e_1]\!] =^? [\![ch(b(y_2, y_1), b(y_4, y_3))|e_2]\!]\}$ ergibt sich die vollständigen Menge von Alternativen durch die Anwendung der Regel *Split-R* und *U-Decompositon* (Definition 5.2.3):

$$
\begin{aligned}
&\{\{e_1 =^? [\![ch(b(y_2, y_1), b(y_4, y_3))|N]\!], e_2 =^? [\![b(x_1, s_1)|N]\!]\},\\
&\{b(x_1, s_1) =^? b(y_2, y_1), e_1 =^? [\![ch(b(z, y_2), b(y_4, y_3))^M|e_2]\!]\}\\
&\{b(x_1, s_1) =^? b(y_4, y_3), e_1 =^? [\![ch(b(y_2, y_1), b(y_3, z))|e_2]\!]\}\\
&\{b(x_1, s_1) =^? b(z_3, z_2), e_1 =^? [\![ch(b(y_2, y_1), b(z_2, z_1)), ch(b(z_4, z_3), b(y_4, y_3))^M|e_2]\!]\}\}
\end{aligned}
$$

Durch *solve* werden exakt diese Alternativen berechnet:

```
> run $ solve
  (([Emv ("E", 1)],[],[Vmv ("x", 1) :=: MTtmv (Tmv ("s", 1))]) =?
   ([Emv ("E", 2)],
    [Chain (CHmv ("CH", 1))
           (Vmv ("y", 2) :=: MTvmv (Vmv ("y", 1))) []
           (Vmv ("y", 4) :=: MTvmv (Vmv ("y", 3)))],[]))
> Left
1 [[([E2],[],[]) =? ([N],[],[x1=s1]),
    ([E1],[],[]) =? ([N],[CH1(y2=y1,[],y4=y3)],[])],
2  [x1=s1 =? y2=y1, CH1 =? [CH11(y13=y2,[(A,y2=y1)],y4=y3)],
    ([E1],[],[]) =? ([E2],[CH11(y13=y2,[(A,y2=y1)],y4=y3)],[])],
3  [x1=s1 =? y4=y3, CH1 =? [CH11(y2=y1,[(E,y4=y3)],y3=y14)],
    ([E1],[],[]) =? ([E2],[CH11(y2=y1,[(E,y4=y3)],y3=y14)],[])],
4  [x1=s1 =? y14=y13, CH1 =? [CH11(y2=y1,[],y13=y15),
                              CH12(y16=y14,[(M,y14=y13)],y4=y3)],
   ([E1],[],[]) =? ([E2],[CH11(y2=y1,[],y13=y15),
                          CH12(y16=y14,[(M,y14=y13)],y4=y3)],[])]]
```

Die Unifikationsprobleme in den Zeilen 2, 3 und 4 enthalten jeweils eine zusätzliche Gleichungen, beispielsweise in Zeile 2: `CH1 =? [CH11(y13=y2,[(A,y2=y1)],y4=y3)]`. In diesen Gleichungen steht auf der linken Seite eine Ketten-Metavariable und auf der rechten Seite eine Liste von Variablenketten. Die Gleichungen sind per Definition in gelöster Form und dienen dazu, sich die Abspaltung oder Aufteilung von Variablenketten zu merken (vgl. Kapitel 5), damit die substituierten Terme syntaktisch gleich werden. In

der Kette `CH11(y13=y2,[(A,y2=y1)],y4=y3)` ist beispielsweise vermerkt, dass an der Anfangsposition der ursprünglichen Kette (`A`) die Bindung `y2=y1` abgespalten wurde. Die Markierungen `A` und `M` entsprechen der Markierung M der Unifikationsregeln für Variablenketten. Variablenketten die mit `E` markiert sind, werden wie unmarkierte Variabelketten behandelt.

Als letztes Beispiel betrachten wir das Σ^{let}-Unifikationsproblem mit Kontextvariablen $\{C_{(T,T)}(t_T) =^? R_{(T,RT)}(s_T)\}$[1], das auch ausführlich am Ende des Kapitels 6 behandelt wird (in Beispiel 6.3.8).

```
> run $ solve ((MTC (Cmv ("C", 1) C) $ MTtmv (Tmv ("s", 1))) =?
               (MTC (Cmv ("R", 1) R) $ MTtmv (Tmv ("t", 1))))
> Left
1 [[C1c =? R11-[@(R13-[t1],C8c[ ])],
    R1r =? R11-[@(R13-[ ],C8c[s1])]],
2  [C1c =? (letrec ([E5],[],[x6=C8c[ ],y7=R3-[t1]]) in R2-[y7]),
    R1r =? (letrec ([E5],[],[x6=C8c[s1],y7=R3-[ ]]) in R2-[y7])],
3  [C1c =? (letrec ([E5],[],[x6=C8c[ ]]) in R3-[t1]),
    R1r =? (letrec ([E5],[],[x6=C8c[s1]]) in R3-[ ])],
4  [C1c =? (letrec ([E5],[],[x6=R3-[t1]]) in R4-[ ]),
    R1r =? (letrec ([E5],[],[x6=R3-[ ]]) in R4-[s1]), s1 =? x6],
5  [C1c =? R1r[C8c[ ]], t1 =? C8c[s1]]]
```

Kontext-Metavariablen, die allgemeine Kontexte darstellen, werden im Interpreter mit einem Suffix `c` gedruckt. Kontext-Metavariablen aus der Klasse der Oberflächenkontexte tragen den Suffix `s`, Kontext-Metavariablen der Klasse der Reduktionskontexte den Suffix `r` und für Metavariablen aus der Klasse der schwachen Reduktionskontext wird der Suffix `-` verwendet. Die vollständige Menge der alternativen, die durch das Programm berechnet wird, entspricht der im Beispiel 6.3.8 angegeben vollständigen Menge. Allerdings mit der Einschränkung, dass in der vom Programm berechneten Menge nur ein Unifikationsproblem der Form

```
C1c =? (letrec ([E5],[],[x6=C8c[ ],y7=R3-[t1]]) in R2-[y7]),
R1r =? (letrec ([E5],[],[x6=C8c[s1],y7=R3-[ ]]) in R2-[y7])
```

enthalten ist (in Zeile 2) und dass zusätzliche in Beispiel 6.3.8 vorkommenden Problem

```
C1c =? (letrec ([E5],[],[y7=R3-[t1]],x6=C8c[ ]) in R2-[y7]),
R1r =? (letrec ([E5],[],[y7=R3-[ ]],x6=C8c[s1],) in R2-[y7])
```

[1] In dem Problem ist C eine allgemeine Kontextvariable und R eine Reduktionskontextvariable.

nicht berechnet wird. Der Grund dafür ist, dass von den beiden Problemen wegen der Vertauschbarkeit von Elementen in `letrec`-Umgebungen nur eines in der Vollständigen Menge der Alternativen benötigt wird. Damit das Vertauschbarkeits-Axiom (*Cl*) anwendbar ist, werden im Programm zusätzlich neue Umgebungs-Metavariablen (`E5`) in den `letrec`-Umgebungen instantiiert. Außerdem werden die Nebenbedingungen, die für die Wohlstrukturiertheit der berechneten Unifikatoren gelten müssen korrekt beachtet. So ist beispielsweise in den Gleichung

```
1 C1c =? (letrec ([E5],[],[x6=R3-[t1]]) in R4-[ ]),
2 R1r =? (letrec ([E5],[],[x6=R3-[ ]]) in R4-[s1]),
3 s1 =? x6
```

`R4-` eine Kontext-Metavariable der Klasse der schwachen Reduktionskontexte und es gilt `s =? x6`. D.h. `(letrec ([E5],[],[x6=R3-[ ]]) in R4-[s1])` stellt einen Reduktionskontext dar und deshalb ist die Gleichung in Zeile 2 wohlstrukturiert.

Die Funktion *unify* mit dem Typ

$$unify :: MAusdr \to MAusdr \to State\ Integer\ [Subst]$$

berechnet für zwei gegebene Ausdrücke eine vollständige Menge von Unifikatoren. Dazu ruft sie rekursiv die Funktion *solve* auf. Zur einfachen Erzeugung von neuen Variablen, ist auch die Funktion *solve* in einer Zustands-Monade gekapselt. Um als Rückgabewert lediglich die vollständige Menge von Unifikatoren zu erhalten, wird die Funktion im Haskell-Interpreter folgenderma"sen aufgerufen:

```
> State.evalState (unify Ausdruck1 Ausdruck2) i
```

wobei i eine Integer-Zahl ist, die größer ist als alle Integer-Zahlen, die in Variablennamen in `Ausdruck1` oder `Ausdruck2` vorkommen.

Die Funktion *cpss* berechnet für eine Liste von Reduktionen und eine Reduktion alle kritischen Paare (Definition 7.2.1) zwischen der Reduktion und den Reduktionen. Ihr Typ ist:

$$\begin{aligned} cpss :: &[(String, (MAusdr, MAusdr))] \\ &\to (String, (MAusdr, MAusdr)) \\ &\to [(String, MAusdr, MAusdr, MAusdr)]. \end{aligned}$$

Die Reduktionen, die *cpss* als Argument erwartet, stehen in einem Tupel $(String, (MAusdr, MAusdr))$, wobei der String den Namen der jeweiligen Reduktionsregel enthält, die durch das Paar $(MAusdr, MAusdr)$ repräsentiert wird. Die Rückgabe der Funktion ist eine Menge von kritischen Paaren, die durch eine Liste von vier-Tupeln $[(String, MAusdr, MAusdr, MAusdr)]$ dargestellt wird. Der String enthält die Namen der Reduktionsregeln, die das kritische Paar erzeugen. Darauf

folgt der Ausdruck, in dem die Überlappung auftritt. Die beiden letzten Ausdrücke stellen das kritische Paar dar, wobei der erste der beiden Ausdrücke der eno-reduzierte und der zweite der iS-reduzierte Ausdruck ist. Ein Beispielaufruf von *cpss*, der nur die ersten beiden kritischen Paare für eine interne (*llet-in*)-Reduktion ausgibt (was durch das vorangestellte `take 2 $` erreicht wird) ist:

```
> take 2 $ cpss norules ("iS,llet-in",isIletin)
> 1 [("eno,lletin iS,llet-in",
      (letrec ([E1],[],[]) in (letrec ([E2],[],[]) in r1)),
      (letrec ([E1,E2],[],[]) in r1),
      (letrec ([E1,E2],[],[]) in r1)),
  2  ("eno,lletin iS,llet-in",
      (letrec ([E1],[],[]) in (letrec ([E2],[],[]) in
                               (letrec ([E7],[],[]) in r6))),
      (letrec ([E1,E2],[],[]) in (letrec ([E7],[],[]) in r6)),
      (letrec ([E1],[],[]) in (letrec ([E2,E7],[],[]) in r6)))]
```

Das kritische Paar in Zeile 1 erzeugt eine triviale Überlappung, d.h. der eno-Redex und der interne Redex (der in diesem Fall kein interner Redex ist) sind identisch. Das kritische Paar in Zeile 2 erzeugt eine echte Überlappung.

Eine vollständige Menge von Gabeldiagrammen für alle internen Reduktionen des Σ^{let}-Kalküls wird durch die Funktion *completeFDSet* mit dem Typ

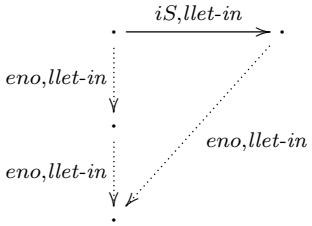
$$\begin{aligned} \mathit{completeFDSet} :: & [(\mathit{String}, (\mathit{MAusdr}, \mathit{MAusdr}))] \\ & \to [(\mathit{String}, (\mathit{MAusdr}, \mathit{MAusdr}))] \\ & \to [([(\mathit{String}, \mathit{MAusdr})], [(\mathit{String}, \mathit{MAusdr})])] \end{aligned}$$

berechnet. Die Funktion erwartet als erstes Argument eine Liste von eno-Reduktionsregeln und als zweites Argument eine Liste von Reduktionsregeln (die internen Reduktionsregeln). Die Listen von Reduktionsregeln enthalten wieder Paare, bestehend aus dem Namen der Reduktionsregel und der jeweiligen Regel. Der Rückgabewert von *completeFDSet* ist ein vollständiger Satz von Gabeldiagrammen für die internen Reduktionen, die als zweites Argument übergeben wurden. Der vollständige Satz wird dargestellt durch eine Liste von Paaren $[([(\mathit{String}, \mathit{MAusdr})], [(\mathit{String}, \mathit{MAusdr})])]$. Die Elemente des Tupels sind Listen von Paaren. In diesen Paaren steht der String für den Namen einer Reduktionsregel und der Ausdruck für ein Redukt, dass aus der Anwendung der benannten Regel resultiert. Das erste Element des Tupels $([(\mathit{String}, \mathit{MAusdr})], [(\mathit{String}, \mathit{MAusdr})])$ repräsentiert die Reduktionsfolge auf der linken und unteren Seite des Gabeldiagramms, das zweite Element stellt die Reduktionsfolge auf der oberen und rechten Seite des Gabeldiagramms dar. Wir geben ein Beispiel:

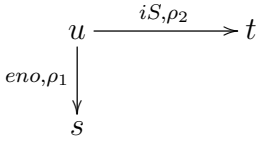
```
> take 2 $ completeFDSet norules isrules
```

```
> 1 [([("eno,lletin",(letrec ([E6,E7],[],[]) in r6))],
      [("iS,lletin",(letrec ([E6,E7],[],[]) in r6))]),
  2 ([("eno,lletin",
       (letrec ([E1,E6],[],[]) in (letrec ([E7],[],[]) in r6))),
      ("eno,lletin",
       (letrec ([E1,E6,E7],[],[]) in r6))],
     [("iS,lletin",
       (letrec ([E1],[],[]) in (letrec ([E6,E7],[],[]) in r6))),
      ("eno,lletin",
       (letrec ([E1,E6,E7],[],[]) in r6))])]
```

Die Ausgabe in Zeile 1 bezieht sich auf eine triviale Überlappung, für die kein Gabeldiagramm notwendig ist. Das Paar in Zeile 2 korrespondiert mit dem Gabeldiagramm



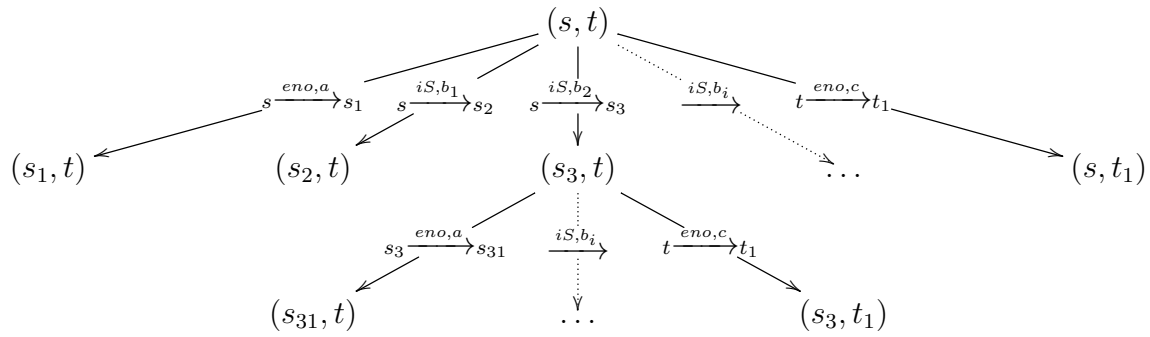
Zum Schließen von Gabeldiagrammen wird in der Funktion *completeFDSet* folgenderma"sen vorgegangen: Sei (s, t) ein kritisches Paar für eine eno-Reduktionsregel ρ_1 und einer (interne) Reduktionsregel ρ_2 und sei s der eno-reduzierte Ausdruck und t der Ausdruck der durch die interne Reduktion reduziert wurde:



D.h. zum Schließen des Diagramms können auf t nur eno-Reduktionen angewendet werden und auf s eno- oder interne Reduktionen. Generiere einen Suchbaum mit dem Wurzelknoten (s, t). Für jeden Knoten (s_i, t_j) im Baum erzeuge all seine Kinderknoten

- (s_l, t_j), wenn $s_i \xrightarrow{eno,a} s_l$ für eine eno-Reduktionsregel a gilt. Und
- (s_l, t_j) für alle internen Reduktionsmöglichkeiten in Oberflächenkontexten $s_i \xrightarrow{iS,b} s_l$. Und
- (s_i, t_k), wenn $t_j \xrightarrow{eno,c} t_k$ für eine eno-Reduktionsregel c gilt.

Folgende Abbildung zeigt das Schema eines solchen Suchbaums.



Der so erzeugte Baum wird mittels Tiefensuche mit einer iterativen Tiefenschranke durchsucht nach einem Knoten (s_i, t_j), so dass die Ausdrücke syntaktisch gleich sind (modulo der durch das Cl-Axiom und die Split-Axiome erzeugte Kongruenzrelation). Man merke sich den Pfad vom Wurzelknoten zu diesem Knoten. Er gibt die kürzesten Folgen von Reduktionen an, so dass das Gabeldiagramm geschlossen wird.

Die Reduktion von Ausdrücken im Programm wird durch Matching implementiert.

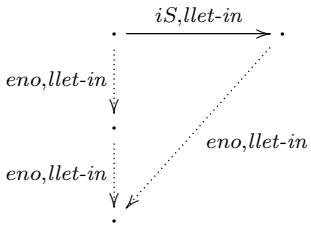
8.2 Ergebnisse der Berechnung von Gabeldiagrammen

Wir geben die vollständigen Gabeldiagramme von Gabeldiagrammen für iS-Reduktionen an, die durch das Programm berechnet werden. Für interne *lapp* und *lbeta* Reduktionen ergeben sich keine kritischen Paare. Das entspricht der Tatsache, dass zum Beweis der Korrektheit der Programmtransformationen *lapp,lbeta* keine Diagramme benötigt werden (siehe Proposition 2.5.5).

8.3 Vollständiger Satz von Gabeldiagrammen für (iS,llet)

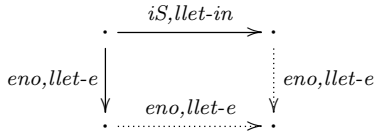
8.3.1 Vollständiger Satz von Gabeldiagrammen für (iS,llet-in)

Für die Überlappungen einer (iS, *llet-in*)-Reduktion mit eno-Reduktionen berechnet das Programm insgesamt 11 Gabeldiagramme, die teilweise redundant sind. Wir geben einige berechnete Gabeldiagramme, zugehörigen kritischen Paaren und die schließenden Reduktionsfolgen an.

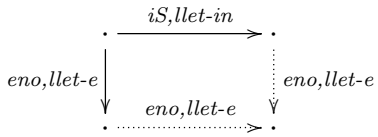


$$
\begin{array}{l}
(\texttt{letrec}\ \{E_1\}\ \texttt{in}\ (\texttt{letrec}\ \{E_6\}\ \texttt{in}\ (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ r_6))) \\
\hline
\xrightarrow{eno,lletin} (\texttt{letrec}\ \{E_1, E_6\}\ \texttt{in}\ (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ r_6)) \\
\xrightarrow{eno,lletin} (\texttt{letrec}\ \{E_1, E_6, E_7\}\ \texttt{in}\ r_6) \\
\hline
\xrightarrow{iS,lletin} (\texttt{letrec}\ \{E_1\}\ \texttt{in}\ (\texttt{letrec}\ \{E_6, E_7\}\ \texttt{in}\ r_6)) \\
\xrightarrow{eno,lletin} (\texttt{letrec}\ \{E_1, E_6, E_7\}\ \texttt{in}\ r_6)
\end{array}
$$

Die folgenden vier Diagramme ergeben sich zwar aus unterschiedlichen Überlappungen, können aber immer durch das gleiche Gabeldiagramm geschlossen werden. Sie entstehen durch die Überlappung mit Varianten einer eno-Reduktionsregel, die jeweils Reduktionsketten unterschiedlicher Länge enthalten.



$$
\begin{array}{l}
(\texttt{letrec}\ \{E_1 x_1 = (\texttt{letrec}\ \{E_6\}\ \texttt{in}\ (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ r_6))\}\ \texttt{in}\ R_1^-[x_1]) \\
\hline
\xrightarrow{eno,llete0} (\texttt{letrec}\ \{E_1, E_6, x_1 = (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ r_6)\}\ \texttt{in}\ R_1^-[x_1]) \\
\xrightarrow{eno,llete0} (\texttt{letrec}\ \{E_1, E_6, E_7, x_1 = r_6\}\ \texttt{in}\ R_1^-[x_1]) \\
\hline
\xrightarrow{iS,lletin} (\texttt{letrec}\ \{E_1, x_1 = (\texttt{letrec}\ \{E_6, E_7\}\ \texttt{in}\ r_6)\}\ \texttt{in}\ R_1^-[x_1]) \\
\xrightarrow{eno,llete0} (\texttt{letrec}\ \{E_1, E_6, E_7, x_1 = r_6\}\ \texttt{in}\ R_1^-[x_1])
\end{array}
$$



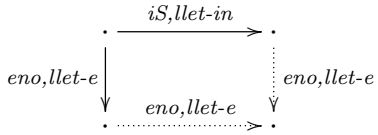
$(\texttt{letrec}\ E_1, nx_1 = nR_1^-[@(x_1, u_1)], x_1 = (\texttt{letrec}\ \{E_6,\}\ \texttt{in}\ (\texttt{letrec}\ \{E_7,\}\ \texttt{in}\ r_6))$
$\quad \texttt{in}\ R_1^-[nx_1])$

$\xrightarrow{eno,llete1}$ $(\texttt{letrec}\ E_1, E_6, nx_1 = nR_1^-[@(x_1, u_1)], x_1 = (\texttt{letrec}\ \{E_7,\}\ \texttt{in}\ r_6)$
$\quad \texttt{in}\ R_1^-[nx_1])$

$\xrightarrow{eno,llete1}$ $(\texttt{letrec}\ \{E_1, E_6, E_7, nx_1 = nR_1^-[@(x_1, u_1)], x_1 = r_6\}\ \texttt{in}\ R_1^-[nx_1])$

$\xrightarrow{iS,lletin}$ $(\texttt{letrec}\ E_1, nx_1 = nR_1^-[@(x_1, u_1)], x_1 = (\texttt{letrec}\ \{E_6, E_7,\}\ \texttt{in}\ r_6)$
$\quad \texttt{in}\ R_1^-[nx_1])$

$\xrightarrow{eno,llete1}$ $(\texttt{letrec}\ \{E_1, E_6, E_7, nx_1 = nR_1^-[@(x_1, u_1)], x_1 = r_6\}\ \texttt{in}\ R_1^-[nx_1])$



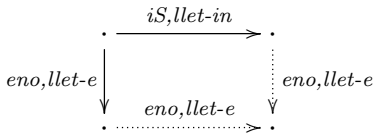
$(\texttt{letrec}\ E_1, nx_1 = nR_1^-[x_1], nx_2 = nR_2^-[nx_1],$
$\quad x_1 = (\texttt{letrec}\ \{E_6,\}\ \texttt{in}\ (\texttt{letrec}\ \{E_7,\}\ \texttt{in}\ r_6))$
$\quad \texttt{in}\ R_1^-[nx_2])$

$\xrightarrow{eno,llete2}$ $(\texttt{letrec}\ E_1, E_6, nx_1 = nR_1^-[x_1], nx_2 = nR_2^-[nx_1],$
$\quad x_1 = (\texttt{letrec}\ \{E_7,\}\ \texttt{in}\ r_6)$
$\quad \texttt{in}\ R_1^-[nx_2])$

$\xrightarrow{eno,llete2}$ $(\texttt{letrec}\ \{E_1, E_6, E_7, nx_1 = nR_1^-[x_1], nx_2 = nR_2^-[nx_1], x_1 = r_6\}\ \texttt{in}\ R_1^-[nx_2])$

$\xrightarrow{iS,lletin}$ $(\texttt{letrec}\ E_1, nx_1 = nR_1^-[x_1], nx_2 = nR_2^-[nx_1],$
$\quad x_1 = (\texttt{letrec}\ \{E_6, E_7,\}\ \texttt{in}\ r_6)$
$\quad \texttt{in}\ R_1^-[nx_2])$

$\xrightarrow{eno,llete2}$ $(\texttt{letrec}\ \{E_1, E_6, E_7, nx_1 = nR_1^-[x_1], nx_2 = nR_2^-[nx_1], x_1 = r_6\}\ \texttt{in}\ R_1^-[nx_2])$



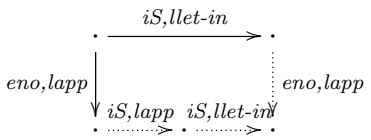
$(\texttt{letrec}\ E_1, nx_1 = nR_1^-[x_1], nx_2 = nR_2^-[nx_1], nx_3 = nR_3^-[nx_2],$
$\quad x_1 = (\texttt{letrec}\ \{E_6,\}\ \texttt{in}\ (\texttt{letrec}\ \{E_7,\}\ \texttt{in}\ r_6))$
$\quad \texttt{in}\ R_1^-[nx_3])$

$\xrightarrow{eno,llete3}$ $(\texttt{letrec}\ E_1, E_6, nx_1 = nR_1^-[x_1], nx_2 = nR_2^-[nx_1], nx_3 = nR_3^-[nx_2],$
$\quad x_1 = (\texttt{letrec}\ \{E_7,\}\ \texttt{in}\ r_6)$
$\quad \texttt{in}\ R_1^-[nx_3])$

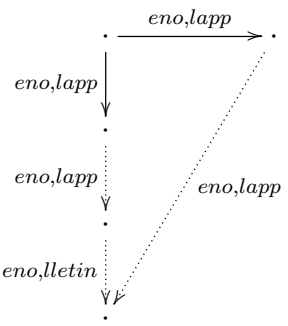
$\xrightarrow{eno,llete3}$ $(\texttt{letrec}\ E_1, E_6, E_7, nx_1 = nR_1^-[x_1], nx_2 = nR_2^-[nx_1], nx_3 = nR_3^-[nx_2],$
$\quad x_1 = r_6$
$\quad \texttt{in}\ R_1^-[nx_3])$

$\xrightarrow{iS,lletin}$ $(\texttt{letrec}\ E_1, nx_1 = nR_1^-[x_1], nx_2 = nR_2^-[nx_1], nx_3 = nR_3^-[nx_2],$
$\quad x_1 = (\texttt{letrec}\ \{E_6, E_7,\}\ \texttt{in}\ r_6)$
$\quad \texttt{in}\ R_1^-[nx_3])$

$\xrightarrow{eno,llete3}$ $(\texttt{letrec}\ E_1, E_6, E_7, nx_1 = nR_1^-[x_1], nx_2 = nR_2^-[nx_1], nx_3 = nR_3^-[nx_2],$
$\quad x_1 = r_6$
$\quad \texttt{in}\ R_1^-[nx_3])$



bzw.



$R_3^-[@((\texttt{letrec}\ \{E_6\}\ \texttt{in}\ (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ r_6)), s_1)]$

$\xrightarrow{eno,lapp1}$ $R_3^-[(\texttt{letrec}\ \{E_6\}\ \texttt{in}\ @((\texttt{letrec}\ \{E_7\}\ \texttt{in}\ r_6), s_1))]$

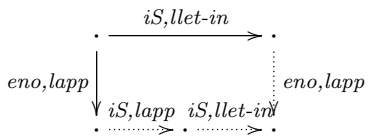
$\xrightarrow{iS \vee eno,lapp}$ $R_3^-[(\texttt{letrec}\ \{E_6\}\ \texttt{in}\ (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ @(r_6, s_1)))]$

$\xrightarrow{iS \vee eno,lletin}$ $R_3^-[(\texttt{letrec}\ \{E_6, E_7\}\ \texttt{in}\ @(r_6, s_1))]$

Die beiden letzten Reduktionen sind eno, falls $R_3^- = [\cdot]$, sonst iS.

$\xrightarrow{iS,lletin}$ $R_3^-[@((\texttt{letrec}\ \{E_6, E_7\}\ \texttt{in}\ r_6), s_1)]$

$\xrightarrow{eno,lapp1}$ $R_3^-[(\texttt{letrec}\ \{E_6, E_7\}\ \texttt{in}\ @(r_6, s_1))]$



$(\texttt{letrec}\ \{E_3\}\ \texttt{in}\ R_3^-[@((\texttt{letrec}\ \{E_6\}\ \texttt{in}\ (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ r_6)), s_1)])$

$\xrightarrow{eno,lapp2}$ $(\texttt{letrec}\ \{E_3\}\ \texttt{in}\ R_3^-[(\texttt{letrec}\ \{E_6\}\ \texttt{in}\ @((\texttt{letrec}\ \{E_7\}\ \texttt{in}\ r_6), s_1))])$

$\xrightarrow{iS,lapp}$ $(\texttt{letrec}\ \{E_3\}\ \texttt{in}\ R_3^-[(\texttt{letrec}\ \{E_6\}\ \texttt{in}\ (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ @(r_6, s_1)))])$

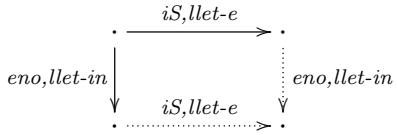
$\xrightarrow{iS,lletin}$ $(\texttt{letrec}\ \{E_3\}\ \texttt{in}\ R_3^-[(\texttt{letrec}\ \{E_6, E_7\}\ \texttt{in}\ @(r_6, s_1))])$

$\xrightarrow{iS,lletin}$ $(\texttt{letrec}\ \{E_3\}\ \texttt{in}\ R_3^-[@((\texttt{letrec}\ \{E_6, E_7\}\ \texttt{in}\ r_6), s_1)])$

$\xrightarrow{eno,lapp2}$ $(\texttt{letrec}\ \{E_3\}\ \texttt{in}\ R_3^-[(\texttt{letrec}\ \{E_6, E_7\}\ \texttt{in}\ @(r_6, s_1))])$

8.3.2 Vollständiger Satz von Gabeldiagrammen für (iS,llet-e)

Für die Überlappungen einer $(iS, \textit{llet-e})$-Reduktion mit eno-Reduktionen berechnet das Programm insgesamt 35 Gabeldiagramme. Wir geben die 5 Fälle an, die unterschiedliche Diagramme ergeben.



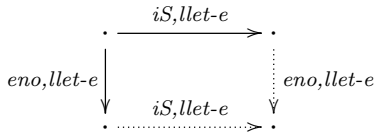
$(\texttt{letrec}\ \{E_6, x_6 = (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ s_6)\}\ \texttt{in}\ (\texttt{letrec}\ \{E_2\}\ \texttt{in}\ r_1))$

$\xrightarrow{eno,lletin}$ $(\texttt{letrec}\ \{E_6, E_2, x_6 = (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ s_6)\}\ \texttt{in}\ r_1)$

$\xrightarrow{iS,llete}$ $(\texttt{letrec}\ \{E_6, E_2, E_7, x_6 = s_6\}\ \texttt{in}\ r_1)$

$\xrightarrow{iS,llete}$ $(\texttt{letrec}\ \{E_6, E_7, x_6 = s_6\}\ \texttt{in}\ (\texttt{letrec}\ \{E_2\}\ \texttt{in}\ r_1))$

$\xrightarrow{eno,lletin}$ $(\texttt{letrec}\ \{E_6, E_7, E_2, x_6 = s_6\}\ \texttt{in}\ r_1)$



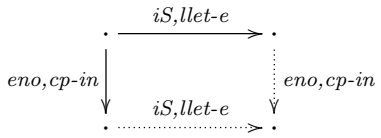
$(\texttt{letrec}\ \{E_8, x_1 = (\texttt{letrec}\ \{E_2\}\ \texttt{in}\ s_1), x_6 = (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ s_6)\}\ \texttt{in}\ R_1^-[x_1])$

$\xrightarrow{eno,llete0}$ $(\texttt{letrec}\ \{E_8, E_2, x_1 = s_1, x_6 = (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ s_6)\}\ \texttt{in}\ R_1^-[x_1])$

$\xrightarrow{iS,llete}$ $(\texttt{letrec}\ \{E_8, E_2, E_7, x_1 = s_1, x_6 = s_6\}\ \texttt{in}\ R_1^-[x_1])$

$\xrightarrow{iS,llete}$ $(\texttt{letrec}\ \{E_8, E_7, x_1 = (\texttt{letrec}\ \{E_2\}\ \texttt{in}\ s_1), x_6 = s_6\}\ \texttt{in}\ R_1^-[x_1])$

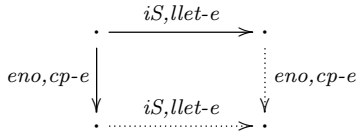
$\xrightarrow{eno,llete0}$ $(\texttt{letrec}\ \{E_8, E_7, E_2, x_1 = s_1, x_6 = s_6\}\ \texttt{in}\ R_1^-[x_1])$



$$(\texttt{letrec}\ E_9, CH_1(cx_2 = cx_1, cx_4 = cx_3), cx_1 = v_1, x_6 = (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ s_6)$$
$$\texttt{in}\ R_1^-[cx_4])$$

$$\xrightarrow{eno,cpin} (\texttt{letrec}\ E_9, CH_1(cx_2 = cx_1, cx_4 = cx_3), cx_1 = v_1, x_6 = (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ s_6)$$
$$\texttt{in}\ R_1^-[v_1])$$
$$\xrightarrow{iS,llete} (\texttt{letrec}\ \{E_9, E_7, CH_1(cx_2 = cx_1, cx_4 = cx_3), cx_1 = v_1, x_6 = s_6\}\ \texttt{in}\ R_1^-[v_1])$$

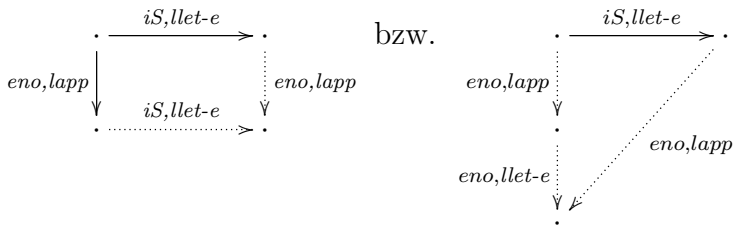
$$\xrightarrow{iS,llete} (\texttt{letrec}\ E_9, E_7, CH_1(cx_2 = cx_1, cx_4 = cx_3), cx_1 = v_1, x_6 = s_6$$
$$\texttt{in}\ R_1^-[cx_4])$$
$$\xrightarrow{eno,cpin} (\texttt{letrec}\ \{E_9, E_7, CH_1(cx_2 = cx_1, cx_4 = cx_3), cx_1 = v_1, x_6 = s_6\}\ \texttt{in}\ R_1^-[v_1])$$



$$(\texttt{letrec}\ E_{14}, CH_2(cy_2 = cy_1, cy_4 = cy_3), cy_1 = v_2, x_1 = R_2^-[@(cy_4, s_2)],$$
$$x_6 = (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ s_6)$$
$$\texttt{in}\ R_1^-[x_1])$$

$$\xrightarrow{eno,cpe0} (\texttt{letrec}\ E_{14}, CH_2(cy_2 = cy_1, cy_4 = cy_3), cy_1 = v_2, x_1 = R_2^-[@(v_2, s_2)],$$
$$x_6 = (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ s_6)$$
$$\texttt{in}\ R_1^-[x_1])$$
$$\xrightarrow{iS,llete} (\texttt{letrec}\ E_{14}, E_7, CH_2(cy_2 = cy_1, cy_4 = cy_3), cy_1 = v_2, x_1 = R_2^-[@(v_2, s_2)],$$
$$x_6 = s_6$$
$$\texttt{in}\ R_1^-[x_1])$$

$$\xrightarrow{iS,llete} (\texttt{letrec}\ E_{14}, E_7, CH_2(cy_2 = cy_1, cy_4 = cy_3), cy_1 = v_2, x_1 = R_2^-[@(cy_4, s_2)],$$
$$x_6 = s_6$$
$$\texttt{in}\ R_1^-[x_1])$$
$$\xrightarrow{eno,cpe0} (\texttt{letrec}\ E_{14}, E_7, CH_2(cy_2 = cy_1, cy_4 = cy_3), cy_1 = v_2, x_1 = R_2^-[@(v_2, s_2)],$$
$$x_6 = s_6$$
$$\texttt{in}\ R_1^-[x_1])$$



$R_3^-[@((\texttt{letrec}\ \{E_6, x_6 = (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ s_6)\}\ \texttt{in}\ r_6), s_1)]$

$\xrightarrow{eno,lapp1}\ R_3^-[(\texttt{letrec}\ \{E_6, x_6 = (\texttt{letrec}\ \{E_7\}\ \texttt{in}\ s_6)\}\ \texttt{in}\ @(r_6, s_1))]$

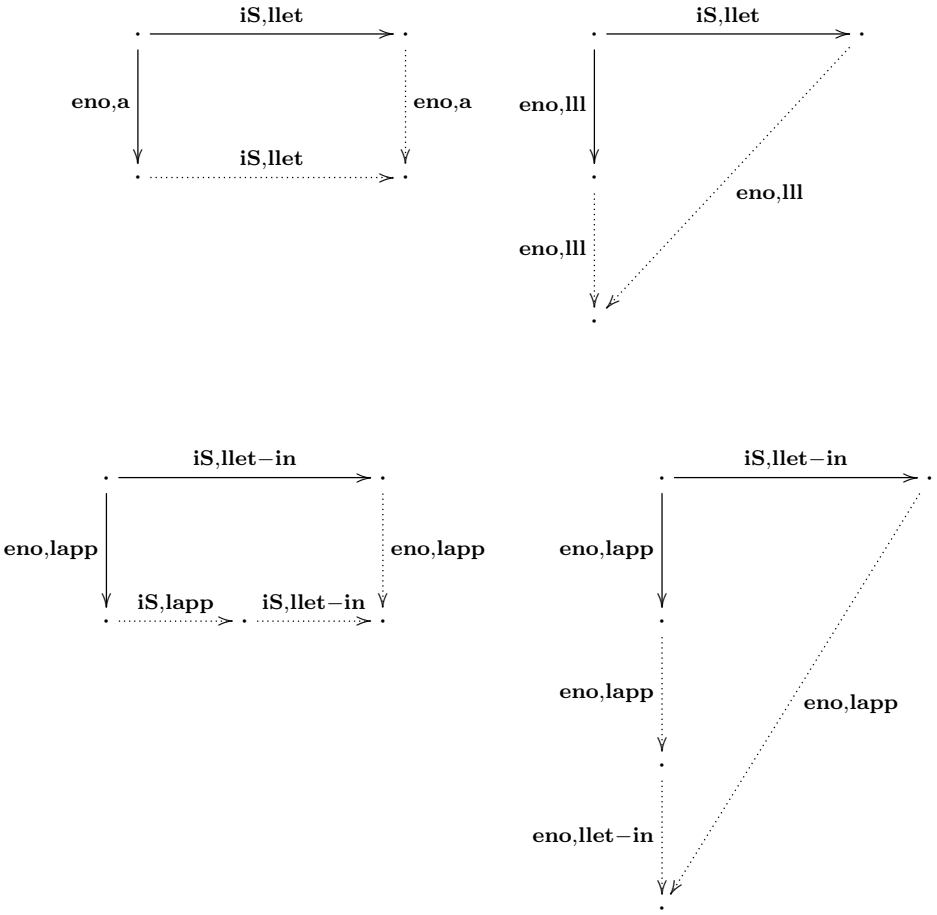
$\xrightarrow{iS \vee no,llete}\ R_3^-[(\texttt{letrec}\ \{E_6, E_7, x_6 = s_6\}\ \texttt{in}\ @(r_6, s_1))]$

Die letzte Reduktion ist no, falls $R_3^- = [\cdot]$, sonst iS.

$\xrightarrow{iS,llete}\ R_3^-[@((\texttt{letrec}\ \{E_6, E_7, x_6 = s_6\}\ \texttt{in}\ r_6), s_1)]$

$\xrightarrow{eno,lapp1}\ R_3^-[(\texttt{letrec}\ \{E_6, E_7, x_6 = s_6\}\ \texttt{in}\ @(r_6, s_1))]$

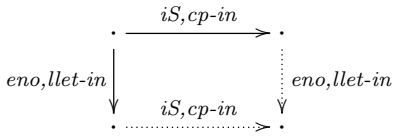
Durch die Zusammenfassung der einzelnen Diagramme ergibt sich der berechnete vollständige Satz von Gabeldiagrammen für $(iS, llet)$:



8.4 Vollständiger Satz von Gabeldiagrammen für (iS,cp)

8.4.1 Vollständiger Satz von Gabeldiagrammen für (iS,cp-in)

Das Programm berechnet 63 Gabeldiagramme für eine $(iS, cp-in)$-Reduktion. Wir geben die Diagramme an, die sich unterscheiden.



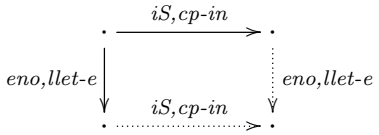
$(\texttt{letrec}\ \{E_6, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6\}\ \texttt{in}\ (\texttt{letrec}\ \{E_2\}\ \texttt{in}\ C^C_{10}[cx_9]))$

$\xrightarrow{eno,lletin}$ $(\texttt{letrec}\ \{E_6, E_2, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6\}\ \texttt{in}\ C^C_{10}[cx_9])$

$\xrightarrow{iS,cpin}$ $(\texttt{letrec}\ \{E_6, E_2, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6\}\ \texttt{in}\ C^C_{10}[v_6])$

$\xrightarrow{iS,cpin}$ $(\texttt{letrec}\ E_6, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6$
$\texttt{in}\ (\texttt{letrec}\ \{E_2\}\ \texttt{in}\ C^C_{10}[v_6]))$

$\xrightarrow{eno,lletin}$ $(\texttt{letrec}\ \{E_6, E_2, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6\}\ \texttt{in}\ C^C_{10}[v_6])$



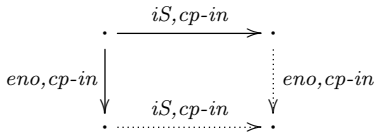
$(\texttt{letrec}\ E_{10}, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6, x_1 = (\texttt{letrec}\ \{E_2\}\ \texttt{in}\ s_1)$
$\texttt{in}\ R^-_{18}[@(R^-_{20}[x_1], C^C_{17}[cx_9])])$

$\xrightarrow{eno,llete.0}$ $(\texttt{letrec}\ E_{10}, E_2, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6, x_1 = s_1$
$\texttt{in}\ R^-_{18}[@(R^-_{20}[x_1], C^C_{17}[cx_9])])$

$\xrightarrow{iS,cpin}$ $(\texttt{letrec}\ E_{10}, E_2, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6, x_1 = s_1$
$\texttt{in}\ R^-_{18}[@(R^-_{20}[x_1], C^C_{17}[v_6])])$

$\xrightarrow{iS,cpin}$ $(\texttt{letrec}\ E_{10}, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6,$
$x_1 = (\texttt{letrec}\ \{E_2\}\ \texttt{in}\ s_1)$
$\texttt{in}\ R^-_{18}[@(R^-_{20}[x_1], C^C_{17}[v_6])])$

$\xrightarrow{eno,llete.0}$ $(\texttt{letrec}\ E_{10}, E_2, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6, x_1 = s_1$
$\texttt{in}\ R^-_{18}[@(R^-_{20}[x_1], C^C_{17}[v_6])])$



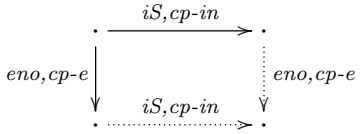
$(\texttt{letrec}\ E_{21}, CH_1(cx_2 = cx_6, cx_4 = cx_3), CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6$
$\quad \texttt{in}\ R^-_{147}[@(R^-_{149}[cx_4], C^C_{146}[cx_9])])$

$\xrightarrow{eno,cpin}$ $(\texttt{letrec}\ E_{21}, CH_1(cx_2 = cx_6, cx_4 = cx_3),$
$\qquad CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6$
$\qquad \texttt{in}\ R^-_{147}[@(R^-_{149}[v_6], C^C_{146}[cx_9])])$

$\xrightarrow{iS,cpin}$ $(\texttt{letrec}\ E_{21}, CH_1(cx_2 = cx_6, cx_4 = cx_3),$
$\qquad CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6$
$\qquad \texttt{in}\ R^-_{147}[@(R^-_{149}[v_6], C^C_{146}[v_6])])$

$\xrightarrow{iS,cpin}$ $(\texttt{letrec}\ E_{21}, CH_1(cx_2 = cx_6, cx_4 = cx_3),$
$\qquad CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6$
$\qquad \texttt{in}\ R^-_{147}[@(R^-_{149}[cx_4], C^C_{146}[v_6])])$

$\xrightarrow{eno,cpin}$ $(\texttt{letrec}\ E_{21}, CH_1(cx_2 = cx_6, cx_4 = cx_3)$
$\qquad CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6$
$\qquad \texttt{in}\ R^-_{147}[@(R^-_{149}[v_6], C^C_{146}[v_6])])$



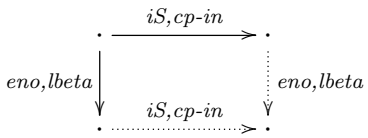
$(\texttt{letrec}\ E_{25}, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6, x_1 = R^-_2[@(cx_9, s_2)]$
$\quad \texttt{in}\ R^-_{162}[@(R^-_{164}[x_1], C^C_{161}[cx_9])])$

$\xrightarrow{eno,cpe.0}$ $(\texttt{letrec}\ E_{25}, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6, x_1 = R^-_2[@(v_6, s_2)]$
$\qquad \texttt{in}\ R^-_{162}[@(R^-_{164}[x_1], C^C_{161}[cx_9])])$

$\xrightarrow{iS,cpin}$ $(\texttt{letrec}\ E_{25}, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6, x_1 = R^-_2[@(v_6, s_2)]$
$\qquad \texttt{in}\ R^-_{162}[@(R^-_{164}[x_1], C^C_{161}[v_6])])$

$\xrightarrow{iS,cpin}$ $(\texttt{letrec}\ E_{25}, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6, x_1 = R^-_2[@(cx_9, s_2)]$
$\qquad \texttt{in}\ R^-_{162}[@(R^-_{164}[x_1], C^C_{161}[v_6])])$

$\xrightarrow{eno,cpe.0}$ $(\texttt{letrec}\ E_{25}, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6, x_1 = R^-_2[@(v_6, s_2)]$
$\qquad \texttt{in}\ R^-_{162}[@(R^-_{164}[x_1], C^C_{161}[v_6])])$



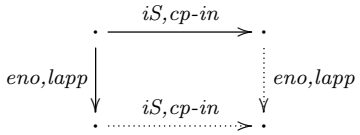
$(\texttt{letrec}\ \{E_6, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6\}\ \texttt{in}\ @((\lambda x_1.C_{19}^C[cx_9]), r_1))$

$\xrightarrow{eno,lbeta2}$ $(\texttt{letrec}\ E_6, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6$
$\texttt{in}\ (\texttt{letrec}\ \{x_1 = r_1\}\ \texttt{in}\ C_{19}^C[cx_9]))$

$\xrightarrow{iS,cpin}$ $(\texttt{letrec}\ E_6, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6$
$\texttt{in}\ (\texttt{letrec}\ \{x_1 = r_1\}\ \texttt{in}\ C_{19}^C[v_6]))$

$\xrightarrow{iS,cpin}$ $(\texttt{letrec}\ E_6, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6$
$\texttt{in}\ @((\lambda x_1.C_{19}^C[v_6]), r_1))$

$\xrightarrow{eno,lbeta2}$ $(\texttt{letrec}\ E_6, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6$
$\texttt{in}\ (\texttt{letrec}\ \{x_1 = r_1\}\ \texttt{in}\ C_{19}^C[v_6]))$



$R_3^-[@((\texttt{letrec}\ \{E_6, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6\}\ \texttt{in}\ C_6^C[cx_9]), s_1)]$

$\xrightarrow{eno,lapp1}$ $R_3^-[(\texttt{letrec}\ \{E_6, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6\}\ \texttt{in}\ @(C_6^C[cx_9], s_1))]$

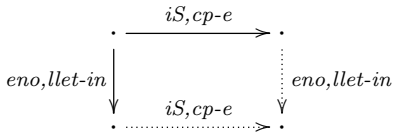
$\xrightarrow{iS,cpin}$ $R_3^-[(\texttt{letrec}\ \{E_6, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6\}\ \texttt{in}\ @(C_6^C[v_6], s_1))]$

$\xrightarrow{iS,cpin}$ $R_3^-[@((\texttt{letrec}\ \{E_6, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6\}\ \texttt{in}\ C_6^C[v_6]), s_1)]$

$\xrightarrow{eno,lapp1}$ $R_3^-[(\texttt{letrec}\ \{E_6, CH_6(cx_7 = cx_6, cx_9 = cx_8), cx_6 = v_6\}\ \texttt{in}\ @(C_6^C[v_6], s_1))]$

8.4.2 Vollständiger Satz von Gabeldiagrammen für (iS,cp-e)

Das Programm berechnet 222 Gabeldiagramme für eine $(iS, cp-e)$-Reduktion. Wir geben die Diagramme an, die sich unterscheiden.



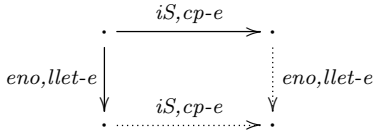
$(\texttt{letrec}\ E_7, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7, y_6 = C_7^C[cy_9]$
$\quad \texttt{in}\ (\texttt{letrec}\ \{E_2\}\ \texttt{in}\ r_1))$

$\xrightarrow{eno,lletin}\ (\texttt{letrec}\ \{E_7, E_2, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7, y_6 = C_7^C[cy_9]\}\ \texttt{in}\ r_1)$

$\xrightarrow{iS,cpe}\ (\texttt{letrec}\ \{E_7, E_2, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7, y_6 = C_7^C[v_7]\}\ \texttt{in}\ r_1)$

$\xrightarrow{iS,cpe}\ (\texttt{letrec}\ E_7, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7, y_6 = C_7^C[v_7]$
$\quad \texttt{in}\ (\texttt{letrec}\ \{E_2\}\ \texttt{in}\ r_1))$

$\xrightarrow{eno,lletin}\ (\texttt{letrec}\ \{E_7, E_2, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7, y_6 = C_7^C[v_7]\}\ \texttt{in}\ r_1)$



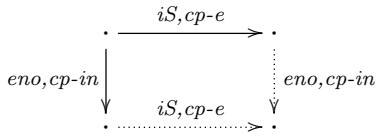
$(\texttt{letrec}\ E_7, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7, y_6 = (\texttt{letrec}\ \{E_2\}\ \texttt{in}\ C_{17}^C[cy_9])$
$\quad \texttt{in}\ R_1^-[y_6])$

$\xrightarrow{eno,llete.0}\ (\texttt{letrec}\ E_7, E_2, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7, y_6 = C_{17}^C[cy_9]$
$\quad \texttt{in}\ R_1^-[y_6])$

$\xrightarrow{iS,cpe}\ (\texttt{letrec}\ E_7, E_2, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7, y_6 = C_{17}^C[v_7]$
$\quad \texttt{in}\ R_1^-[y_6])$

$\xrightarrow{iS,cpe}\ (\texttt{letrec}\ E_7, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7,$
$\quad y_6 = (\texttt{letrec}\ \{E_2\}\ \texttt{in}\ C_{17}^C[v_7])$
$\quad \texttt{in}\ R_1^-[y_6])$

$\xrightarrow{eno,llete.0}\ (\texttt{letrec}\ E_7, E_2, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7, y_6 = C_{17}^C[v_7]$
$\quad \texttt{in}\ R_1^-[y_6])$



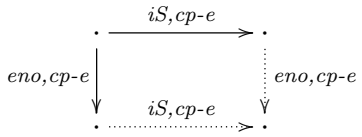
$(\texttt{letrec}\ \{E_{235}, CH_{18}(cx_2 = cy_6, (cx_4 = cx_3), cx_3 = cx_{21}), cy_6 = v_7\}\ \texttt{in}\ R_1^-[cx_4])$

$\xrightarrow{eno,cpin}$ $(\texttt{letrec}\ E_{235}, CH_{18}(cx_2 = cy_6, (cx_4 = cx_3), cx_3 = cx_{21}), cy_6 = v_7$
$\texttt{in}\ R_1^-[v_7])$

$\xrightarrow{iS,cpe}$ $(\texttt{letrec}\ \{E_{235}, CH_{18}(cx_2 = cy_6, cx_3 = cx_{21}), cx_4 = v_7, cy_6 = v_7\}\ \texttt{in}\ R_1^-[v_7])$

$\xrightarrow{iS,cpe}$ $(\texttt{letrec}\ \{E_{235}, CH_{18}(cx_2 = cy_6, cx_3 = cx_{21}), cx_4 = v_7, cy_6 = v_7\}\ \texttt{in}\ R_1^-[cx_4])$

$\xrightarrow{eno,cpin}$ $(\texttt{letrec}\ \{E_{235}, CH_{18}(cx_2 = cy_6, cx_3 = cx_{21}), , cx_4 = v_7, cy_6 = v_7\}\ \texttt{in}\ R_1^-[v_7])$



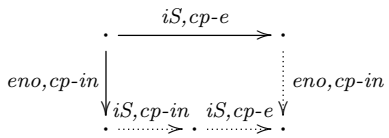
$(\texttt{letrec}\ E_{897}, CH_{680}(cy_2 = cy_6, (cy_4 = cy_3), cy_3 = cy_{683}),$
$cy_6 = v_7, x_1 = R_2^-[@(cy_4, s_2)]$
$\texttt{in}\ R_1^-[x_1])$

$\xrightarrow{eno,cpe.0}$ $(\texttt{letrec}\ E_{897}, CH_{680}(cy_2 = cy_6, (cy_4 = cy_3), cy_3 = cy_{683}),$
$cy_6 = v_7, x_1 = R_2^-[@(v_7, s_2)]$
$\texttt{in}\ R_1^-[x_1])$

$\xrightarrow{iS,cpe}$ $(\texttt{letrec}\ E_{897}, CH_{680}(cy_2 = cy_6, cy_3 = cy_{683}),$
$cy_4 = v_7, cy_6 = v_7, x_1 = R_2^-[@(v_7, s_2)]$
$\texttt{in}\ R_1^-[x_1])$

$\xrightarrow{iS,cpe}$ $(\texttt{letrec}\ E_{897}, CH_{680}(cy_2 = cy_6, cy_3 = cy_{683}),$
$cy_4 = v_7, cy_6 = v_7, x_1 = R_2^-[@(cy_4, s_2)]$
$\texttt{in}\ R_1^-[x_1])$

$\xrightarrow{eno,cpe.0}$ $(\texttt{letrec}\ E_{897}, CH_{680}(cy_2 = cy_6, cy_3 = cy_{683}),$
$cy_4 = v_7, cy_6 = v_7, x_1 = R_2^-[@(v_7, s_2)]$
$\texttt{in}\ R_1^-[x_1])$



$$(\texttt{letrec}\ E_{630}, CH_1(cx_2 = y_6, cx_4 = cx_3), CH_7(cy_7 = cy_6, cy_9 = cy_8),$$
$$cy_6 = v_7, y_6 = (\lambda x_{618}.C^C_{617}[cy_9])$$
$$\texttt{in}\ R_1^-[cx_4])$$

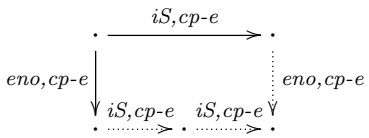
$$\xrightarrow{eno,cpin} (\texttt{letrec}\ E_{630}, CH_1(cx_2 = y_6, cx_4 = cx_3), CH_7(cy_7 = cy_6, cy_9 = cy_8),$$
$$cy_6 = v_7, y_6 = (\lambda x_{618}.C^C_{617}[cy_9])$$
$$\texttt{in}\ R_1^-[(\lambda x_{618}.C^C_{617}[cy_9])])$$

$$\xrightarrow{iS,cpin} (\texttt{letrec}\ E_{630}, CH_1(cx_2 = y_6, cx_4 = cx_3), CH_7(cy_7 = cy_6, cy_9 = cy_8),$$
$$cy_6 = v_7, y_6 = (\lambda x_{618}.C^C_{617}[cy_9])$$
$$\texttt{in}\ R_1^-[(\lambda x_{618}.C^C_{617}[v_7])])$$

$$\xrightarrow{iS,cpe} (\texttt{letrec}\ E_{630}, CH_1(cx_2 = y_6, cx_4 = cx_3), CH_7(cy_7 = cy_6, cy_9 = cy_8),$$
$$cy_6 = v_7, y_6 = (\lambda x_{618}.C^C_{617}[v_7])$$
$$\texttt{in}\ R_1^-[(\lambda x_{618}.C^C_{617}[v_7])])$$

$$\xrightarrow{iS,cpe} (\texttt{letrec}\ E_{630}, CH_1(cx_2 = y_6, cx_4 = cx_3), CH_7(cy_7 = cy_6, cy_9 = cy_8),$$
$$cy_6 = v_7, y_6 = (\lambda x_{618}.C^C_{617}[v_7])$$
$$\texttt{in}\ R_1^-[cx_4])$$

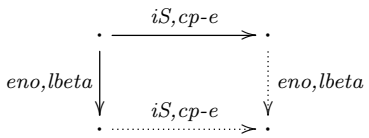
$$\xrightarrow{eno,cpin} (\texttt{letrec}\ E_{630}, CH_1(cx_2 = y_6, cx_4 = cx_3), CH_7(cy_7 = cy_6, cy_9 = cy_8),$$
$$cy_6 = v_7, y_6 = (\lambda x_{618}.C^C_{617}[v_7])$$
$$\texttt{in}\ R_1^-[(\lambda x_{618}.C^C_{617}[v_7])])$$



$$(\texttt{letrec}\ E_{1301}, CH_2(cy_2 = y_6, cy_4 = cy_3), CH_7(cy_7 = cy_6, cy_9 = cy_8),$$
$$cy_6 = v_7, x_1 = R_2^-[@(cy_4, s_2)], y_6 = (\lambda x_{1280}.C^C_{1279}[cy_9])$$
$$\texttt{in}\ R_1^-[x_1])$$

$$\xrightarrow{eno,cpe.0} (\texttt{letrec}\ E_{1301}, CH_2(cy_2 = y_6, cy_4 = cy_3), CH_7(cy_7 = cy_6, cy_9 = cy_8),$$
$$cy_6 = v_7, x_1 = R_2^-[@((\lambda x_{1280}.C^C_{1279}[cy_9]), s_2)], y_6 = (\lambda x_{1280}.C^C_{1279}[cy_9])$$
$$\texttt{in}\ R_1^-[x_1])$$
$$\xrightarrow{iS,cpe} (\texttt{letrec}\ E_{1301}, CH_2(cy_2 = y_6, cy_4 = cy_3), CH_7(cy_7 = cy_6, cy_9 = cy_8),$$
$$cy_6 = v_7, x_1 = R_2^-[@((\lambda x_{1280}.C^C_{1279}[v_7]), s_2)], y_6 = (\lambda x_{1280}.C^C_{1279}[cy_9])$$
$$\texttt{in}\ R_1^-[x_1])$$
$$\xrightarrow{iS,cpe} (\texttt{letrec}\ E_{1301}, CH_2(cy_2 = y_6, cy_4 = cy_3), CH_7(cy_7 = cy_6, cy_9 = cy_8),$$
$$cy_6 = v_7, x_1 = R_2^-[@((\lambda x_{1280}.C^C_{1279}[v_7]), s_2)], y_6 = (\lambda x_{1280}.C^C_{1279}[v_7])$$
$$\texttt{in}\ R_1^-[x_1])$$

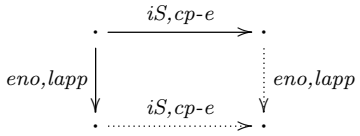
$$\xrightarrow{iS,cpe} (\texttt{letrec}\ E_{1301}, CH_2(cy_2 = y_6, cy_4 = cy_3), CH_7(cy_7 = cy_6, cy_9 = cy_8),$$
$$cy_6 = v_7, x_1 = R_2^-[@(cy_4, s_2)], y_6 = (\lambda x_{1280}.C^C_{1279}[v_7])$$
$$\texttt{in}\ R_1^-[x_1])$$
$$\xrightarrow{eno,cpe.0} (\texttt{letrec}\ E_{1301}, CH_2(cy_2 = y_6, cy_4 = cy_3), CH_7(cy_7 = cy_6, cy_9 = cy_8),$$
$$cy_6 = v_7, x_1 = R_2^-[@((\lambda x_{1280}.C^C_{1279}[v_7]), s_2)], y_6 = (\lambda x_{1280}.C^C_{1279}[v_7])$$
$$\texttt{in}\ R_1^-[x_1])$$



$$(\texttt{letrec}\ E_7, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7, y_6 = @((\lambda y_{27}.C^C_{28}[cy_9]), r_1)$$
$$\texttt{in}\ R^-_3[y_6])$$

$$\xrightarrow{eno,lbeta3.0}\ (\texttt{letrec}\ E_7, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7,$$
$$y_6 = (\texttt{letrec}\ \{y_6 = r_1\}\ \texttt{in}\ C^C_{28}[cy_9])$$
$$\texttt{in}\ R^-_3[y_6])$$
$$\xrightarrow{iS,cpe}\ (\texttt{letrec}\ E_7, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7,$$
$$y_6 = (\texttt{letrec}\ \{y_6 = r_1\}\ \texttt{in}\ C^C_{28}[v_7])$$
$$\texttt{in}\ R^-_3[y_6])$$

$$\xrightarrow{iS,cpe}\ (\texttt{letrec}\ E_7, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7,$$
$$y_6 = @((\lambda y_{27}.C^C_{28}[v_7]), r_1)$$
$$\texttt{in}\ R^-_3[y_6])$$
$$\xrightarrow{eno,lbeta3.0}\ (\texttt{letrec}\ E_7, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7,$$
$$y_6 = (\texttt{letrec}\ \{y_6 = r_1\}\ \texttt{in}\ C^C_{28}[v_7])$$
$$\texttt{in}\ R^-_3[y_6])$$

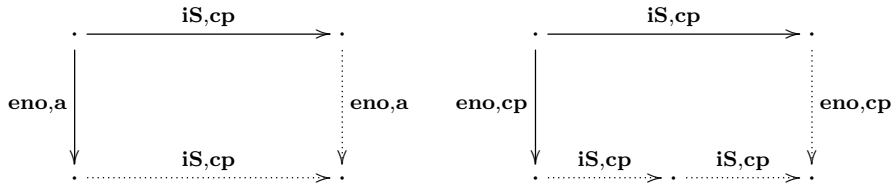


$$R^-_3[@((\texttt{letrec}\ \{E_7, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7, y_6 = C^C_7[cy_9]\}\ \texttt{in}\ r_6), s_1)]$$

$$\xrightarrow{eno,lapp1}\ R^-_3[\,(\texttt{letrec}\ E_7, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7, y_6 = C^C_7[cy_9]\,]$$
$$\texttt{in}\ @(r_6, s_1))$$
$$\xrightarrow{iS,cpe}\ R^-_3[\,(\texttt{letrec}\ E_7, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7, y_6 = C^C_7[v_7]\,]$$
$$\texttt{in}\ @(r_6, s_1))$$

$$\xrightarrow{iS,cpe}\ R^-_3[@(\,(\texttt{letrec}\ E_7, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7, y_6 = C^C_7[v_7]\,, s_1)]$$
$$\texttt{in}\ r_6)$$
$$\xrightarrow{eno,lapp1}\ R^-_3[\,(\texttt{letrec}\ E_7, CH_7(cy_7 = cy_6, cy_9 = cy_8), cy_6 = v_7, y_6 = C^C_7[v_7]\,]$$
$$\texttt{in}\ @(r_6, s_1))$$

Durch die Zusammenfassung der einzelnen Diagramme ergibt sich der berechnete vollständige Satz von Gabeldiagrammen für (iS, cp):


iS,cp
eno,a
eno,a
iS,cp
iS,cp
eno,cp
eno,cp
iS,cp
iS,cp

9 Zusammenfassung und Ausblick

9.1 Zusammenfassung

In dieser Arbeit wird ein einfacher λ-Kalkül mit einem `letrec`-Konstrukt betrachtet, der auf einer Teilmenge des LR-Kalküls aus Schmidt-Schauß et al. (2007) basiert. Der verwendete Kalkül wird als Λ^{let}-Kalkül bezeichnet und besteht aus einer Sprache $L_{\Lambda^{let}}$, deren Elemente Ausdrücke sind, aus Reduktionsregeln, die Ausdrücke transformieren und einer standardisierten Form der Auswertung, der Normalordnung. Der Kalkül wird in Kapitel 2 definiert. In diesem Kapitel wird außerdem das zentrale semantische Konzept des Kalküls, die kontextuelle Gleichheit von Ausdrücken und darauf aufbauend die Korrektheit von Programmtransformationen definiert. Als wesentliches Hilfsmittel zum Beweis der Korrektheit von Programmtransformationen dienen vollständige Sätze von Gabel- und Vertauschungsdiagrammen.

Das Ziel dieser Arbeit ist es, eine Methode zu entwickeln, mit der vollständige Sätze von Gabeldiagrammen für den Λ^{let}-Kalkül berechnet werden können. Die Methode basiert auf der Berechnung von Überlappungen von Reduktionsregeln, wozu Reduktionsregeln des Λ^{let}-Kalküls unifiziert werden müssen.

Der Hauptteil der Arbeit (Kapitel 3, 4, 5 und 6) befasst sich mit den Unifikationsmethoden, die zur Unifikation von verschiedenen Konstrukten, die in Λ^{let}-Reduktionsregeln enthalten sind, benötigt werden. Im Verlauf der vier Kapitel wird sukzessiv eine Signatur Σ^{let} definiert, mit deren Hilfe Konstrukte, die in Λ^{let}-Reduktionsregeln vorkommen, als Terme über der Signatur Σ^{let} dargestellt und unifiziert werden können.

Die Reduktionsregeln in Λ^{let} enthalten Variablen verschiedener Sorten. In Kapitel 3 werden Terme mit Sorten definiert und das Verfahren zur Unifikation von Termen mit Sorten beschrieben, das auf der Arbeit von (Schmidt-Schauß, 1989a) basiert. Es wird gezeigt, dass der Unifikationsalgorithmus terminiert und vollständig ist.

Kapitel 4 befasst sich mit der Unifikation von Funktionssymbolen, deren Argumente vertauschbar sind. Diese Unifikationsmethode wird benötigt, da die Elemente von `letrec`-Umgebungen in Λ^{let} vertauschbar sind und diese Vertauschbarkeit bei der Unifikation berücksichtigt werden muss. Es wird ein Unifikationsalgorithmus aus der Theorie der Unifikation von Mengen (Dovier, Piazza, & Rossi, 2000) vorgestellt, der terminiert und vollständig ist.

`letrec`-Umgebungen in Λ^{let}-Reduktionsregeln enthalten in manchen Fällen ein Konstrukt, um Variablenketten beliebiger Länge darzustellen. Die Methoden, die zur Unifikation von Variablenketten verwendet werden, sind in Kapitel 5 beschrieben.

Als letztes Konstrukt, das es bei der Unifikation zu berücksichtigen gilt, enthalten Λ^{let}-Reduktionsregeln Kontextvariablen mit Sorten. Deren Unifikation erfolgt wie in Kapitel 6 dargelegt. Dabei orientieren wir uns hauptsächlich an der Arbeit von (Comon, 1998) und skizzieren dessen Überlegungen zur Vollständigkeit und Terminierung der Unifikationsprozedur.

In Kapitel 7 wird der Begriff der Überlappung von Reduktionsregeln formal definiert. Dazu fassen wir zuerst die Definition der Signatur Σ^{let}, die zur Darstellung und Unifikation von Λ^{let}-Reduktionsregeln verwendet wird, zusammen. Dann werden Reduktionsregeln für Terme über dieser Signatur und eine standardisierte Form der Auswertung, analog zu den entsprechenden Definitionen des Λ^{let}-Kalküls definiert. Die in Σ^{let} definierte Normalordnungsreduktion stellt eine Einschränkung der Normalordnungsreduktion aus Λ^{let} dar. Abschließend beschreiben wir, wie alle Überlappungen für einen vollständigen Satz von Gabeldiagrammen berechnet werden können.

Das abschließende Kapitel 8 befasst sich mit der Implementierung, die im Rahmen dieser Arbeit entstanden ist. Sie berechnet vollständige Sätze von Gabeldiagrammen. Wir skizzieren einige Datenstrukturen des Programms und geben Beispiele für die Arbeitsweise der wichtigsten Funktionen. Anschließend geben wir die vollständigen Sätze von Gabeldiagrammen an, die das Programm berechnet.

9.2 Ausblick

Abschließend zeigen wir einige Erweiterungen und Verbesserungen auf, die durch zukünftige Forschung erreicht werden könnten.

9.2.1 Anpassung der Implementierung an die Unifikationstheorien

Wie wir in Kapitel 8 zu Beginn dargelegt haben, entspricht die Implementierung, die im Rahmen der Arbeit entstanden ist nicht in allen Aspekten den vorgestellten Unifikationstheorien. Um eine höhere Sicherheit bezüglich der Terminierung und der Vollständigkeit des programmierten Unifikationsalgorithmus zu erhalten, ist eine Anpassung des Programms an die Unifikationstheorien notwendig. Dazu ist das Programm so zu verändern, dass anstatt Ausdrücken mit Metavariablen Terme mit Sorten zur Unifikation verwendet werden. Außerdem sind die einzelnen Unifikationsregeln des Programms so abzuändern, dass sie den Regeln der Theorien

entsprechen.

Das Programm kann auch noch an verschieden anderen Stellen erweiterte werden. Beispielsweise um ein Modul, das es erlaubt eine Menge von Gabeldiagrammen einzugeben. Anhand der Berechnung aller Überlappungen kann dann getestet werden, ob die eingegebene Menge von Diagrammen vollständig ist. Eine weitere, wünschenswerte Funktion ist die Berechnung einer vollständigen Menge von Vertauschungsdiagrammen.

9.2.2 Offene Fragen zur Unifikation von Termen mit Variablenketten

Die Behandlung der Unifikation von Termen mit Variablenketten in Kapitel 5 ist eher informell. An vielen Stellen, an den eigentlich Beweise von Aussagen nötig sind, werden lediglich Beispiele gegeben. Insbesondere die Vollständigkeit der Unifikationsprozedur wird nicht ausführlich untersucht. Hierzu sind weitere Untersuchungen notwendig.

9.2.3 Berechnung von Diagrammen für andere Kalküle

Die Berechnung von Gabeldiagrammen erfolgt in dieser Arbeit für den einfachen Λ^{let}-Kalkül, der über die Konstrukte Applikation, Abstraktion und `letrec` verfügt. In der Literatur zur Korrektheit von Programmtransformationen existieren eine Vielzahl von Kalkülen, die über einen größeren Sprachumfang verfügen (Schmidt-Schauß, 2003; Schmidt-Schauß et al., 2007; Sabel, 2008). Des weiteren gibt es Kalküle, in denen die Normalordnungsreduktion in einer anderen Form als in Λ^{let} definiert wird (Schmidt-Schau"ß, 2007). Für diese Kalküle ist es interessant zu untersuchen, ob sich die hier vorgestellten Methoden verwenden lassen, um für sie vollständige Sätze von Gabeldiagrammen zu berechnen.

Literaturverzeichnis

Abramsky, S. (1990). The Lazy Lambda Calculus. In D. A. Turner (Hrsg.), *Research Topics in Functional Programming* (S. 65–116). Reading, MA: Addison-Welsey.

Arenas-Sanchez, P., & Dovier, A. (1997). A minimality study for set unification. *Journal of Functional and Logic Programming*, *7*, 1–49.

Ariola, Z. M., & Felleisen, M. (1997). The call-by-need lambda calculus. *Journal of functional programming*, *7*(03), 265–301.

Baader, F. (1991). Unification, weak unification, upper bound, lower bound, and generalization problems. In *Proceedings of the 4th international conference on Rewriting techniques and applications table of contents* (S. 86–97). Springer-Verlag New York, Inc. New York, NY, USA.

Baader, F., & Nipkow, T. (1998). *Term Rewriting and All That*. Cambridge University Press.

Baader, F., & Schulz, K. U. (1996). Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures. *Journal of Symbolic Computation*, *21*, 211–243.

Baader, F., & Snyder, W. (2001). Unification Theory. In *Handbook of Automated Reasoning* (Bd. I, S. 445–532). Elsevier Science.

Barendregt, H. (1984). *The Lambda Calculus: Its Syntax and Semantics* (2 Aufl.). North Holland.

Bezem, M., Klop, J. W., & Vrijer, R. de. (2003). *Term Rewriting Systems*.

Boudet, A., Contejean, E., & Devie, H. (1990). A new AC unification algorithm with an algorithm for solving systems of diophantine equations. In *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on e* (S. 289–299).

Büttner, W. (1986). Unification in the Data Structure Sets. In *Proceedings of the 8th International Conference on Automated Deduction* (S. 470–488). Springer-Verlag.

Comon, H. (1998). Completion of Rewrite Systems with Membership Constraints Part II: Constraint Solving. *Journal of Symbolic Computation*, *25*(4), 421–453.

Corbin, J., & Bidoit, M. (1983). *A Rehabilitation of Robinson's Unification Algorithm. Information Processing 83*. North-Holland.

Dantsin, E., & Voronkov, A. (1999). Bag and set unification. In *FOSSACS* (Bd. 99, S. 17).

Dovier, A., Piazza, C., & Rossi, G. (2000). *A uniform approach to constraintsolving for lists, multisets, compact lists* (Tech. Rep.). and sets. Technical Report, Dipartimento di Matematica, Universita di Parma.

Dovier, A., Policriti, A., & Rossi, G. (1998). A Uniform Axiomatic View of Lists, Multisets, and Sets, and the Relevant Unification Algorithms. *Fundamenta Informaticae*, *36*(2-3), 201–234.

Dovier, A., Pontelli, E., & Rossi, G. (1998). *Set Unification Revisited* (Tech. Rep.). NMSU-CSTR-9817, Dept. of Computer Science, New Mexico State University, USA. October. Submitted.

Dovier, A., Pontelli, E., & Rossi, G. (2006). Set unification. *Theory and Practice of Logic Programming*, *6*(06), 645–701.

Fages, F., & Huet, G. (1986). Complete sets of unifiers and matchers in equational theories. *Theor. Comput. Sci.*, *43*(2-3), 189–200.

Fortenbacher, A. (1985). An algebraic approach to unification under associativity and commutativity. In *Rewriting Techniques and Applications* (S. 381–397).

Gallier, J. H., & Snyder, W. (1989). Complete Sets of Transformations for General E-Unification. *TCS*, *67*(2&3), 203–260.

Goldfarb, W. D. (1981). Undecidability of the Second-Order Unification Problem. *THEORET. COMP. SCI.*, *13*(2), 225–230.

Huber, M. (2000). *Jonah: Ein System zur Validierung von Reduktionsdiagrammen in nichtdeterministischen Lambda-Kalk¨ulen mit Let-Ausdrücken, Letrec-Ausdrücken und Konstruktoren.* Unver"offentlichte Diplomarbeit, Johann

Wolfgang Goethe-Universität Frankfurt am Main.

Kapur, D., & Narendran, P. (1986). NP-completeness of the set unification and matching problems. In *Proc. of the 8th international conference on Automated deduction table of contents* (S. 489–495). Springer-Verlag New York, Inc. New York, NY, USA.

Kapur, D., & Narendran, P. (1992a, Oktober). Complexity of unification problems with associative-commutative operators. *Journal of Automated Reasoning*, *9*(2), 261–288.

Kapur, D., & Narendran, P. (1992b). Double-exponential complexity of computing a complete set of AC-unifiers. In *Logic in Computer Science, 1992. LICS '92., Proceedings of the Seventh Annual IEEE Symposium on* (S. 11–21).

Kutzner, A. (2000). Ein nichtdeterministischer call-by-need Lambda-Kalkl mit erratic Choice: Operationale Semantik, Programmtransformationen und Anwendungen. *Doktorarbeit, Johann Wolfgang Goethe-Universitat, Frankfurt am Main, April*.

Levy, J. (1996). Linear Second-Order Unification. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications* (S. 332–346). Springer-Verlag.

Lincoln, P., & Christian, J. (1989). Adventures in associative-commutative unification (A summary). In *9th International Conference on Automated Deduction* (S. 358–367).

Livesey, M., & Siekmann, J. H. (1976). *Unification of AC-terms (bags) and ACI-terms (sets)* (Tech. Rep.). Internal report, University of Essex, 1975. Also published as Technical Report 3-76, Universitat Karlsruhe, 1976.

Martelli, A., & Montanari, U. (1982). An Efficient Unification Algorithm. *ACM Trans. Program. Lang. Syst.*, *4*(2), 258–282.

Nutt, W. (1991). The unification hierarchy is undecidable. *Journal of Automated Reasoning*, *7*(3), 369–381.

Paterson, M. S., & Wegman, M. N. (1976). Linear unification. In *Proceedings of the eighth annual ACM symposium on Theory of computing* (S. 181–186). Hershey, Pennsylvania, United States: ACM.

Sabel, D. (2003). *Realisierung der Ein-/Ausgabe in einem Compiler für Haskell bei*

Verwendung einer nichtdeterministischen Semantik. Unver"offentlichte Diplomarbeit, Johann Wolfgang Goethe-Universität Frankfurt am Main.

Sabel, D. (2008). *Semantics of a Call-by-Need Lambda Calculus with McCarthy's amb for Program Equivalence*. Unver"offentlichte Dissertation, Johann Wolfgang Goethe-Universität Frankfurt am Main.

Schmidt-Schauß, M. (1989a). *Computational Aspects of an Order-Sorted Logic with Term Declarations, volume 395 of LNAI*. Springer.

Schmidt-Schauß, M. (1989b). Unification in a combination of arbitrary disjoint equational theories. *Journal of Symbolic Computation*, *8*, 51–99.

Schmidt-Schauß, M. (1994). *Unification of Stratified Second Order Terms*. Fachbereich Informatik, Univ.

Schmidt-Schauß, M. (2003). *FUNDIO: A Lambda-Calculus with a letrec, case, Constructors, and an IO-Interface: Approaching a Theory of unsafePerformIO* (Bd. 16; Tech. Rep.). Johann Wolfgang Goethe-Universität Frankfurt am Main.

Schmidt-Schauß, M. (2006). *Skript zur Vorlesung "'Funktionale Programmierung: Programmtransformationen"' im Wintersemester 2006/2007.*

Schmidt-Schau"ß, M. (2007). Correctness of Copy in Calculi with Letrec. *Lecture Notes in Computer Science*, *4533*, 329.

Schmidt-Schauß, M., Sabel, D., & Schuetz, M. (2007). Safety of Nöcker's strictness analysis. *Journal of Functional Programming*, *18*(04), 503–551.

Schmidt-Schauß, M., & Schulz, K. U. (2002). Solvability of Context Equations with Two Context Variables is Decidable. *Journal of Symbolic Computation*, *33*(1), 77–122.

Stickel, M. E. (1975). A complete unification algorithm for associative-commutative functions. 4th International Joint Conf. on Artificial Intelligence. *Tbilisi, USSR, Proceedings of the 4th International Joint Conference of Artificial Intelligence*, 71–82.

Stickel, M. E. (1981). A Unification Algorithm for Associative-Commutative Functions. *J. ACM*, *28*(3), 423–434.

Stolzenburg, F. (1999). An Algorithm for General Set Unification and Its Complex-

ity. *Journal of Automated Reasoning*, *22*(1), 45–63.

Walther, C. (1988). Many-sorted unification. *J. ACM*, *35*(1), 1–17.