Zwei Gedanken zur höherstufigen Unifikation - Relevanteste Unifikatoren und Unifikation mit Fallunterscheidung

Martin Kübler

München 2006

Zwei Gedanken zur höherstufigen Unifikation - Relevanteste Unifikatoren und Unifikation mit Fallunterscheidung

Martin Kübler

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Martin Kübler
aus München

München, den 26.07.2004

Erstgutachter: Prof. Dr. Helmut Schwichtenberg
Zweitgutachter: Prof. Dr. Wilfried Buchholz
Externer Gutachter: Prof. Dr. Tobias Nipkow (TU München)
Tag der mündlichen Prüfung: 20.12.2005

Inhaltsverzeichnis

1 Summary

Two Ideas about Higher-Order Unification - Most Relevant Unifiers and Unification with Case Distinction.

Unification is a basic proof principle in mathematics. It means to check whether two expressions could be equalized or not by replacing free variables with terms of the same type. This problem is decidable and well understood if the free variables are only of base types, so they are not allowed to stand for functions or other objects of higher type. The oldest algorithm to solve such first order problems was published by Robinson [34], further improvements lead to a linear algorithm, see e. g. [6],[23] and [28]. The first order unification is used in many applications like type theory [17], rewrite systems [3] or logic programming [36].

In the higher order case, where free variables could also stand for higher order objects like functions or functionals, unification becomes undecible ([15],[18]). Higher order terms are expressed in the simply typed λ-calculus ([4],[5]). This language is strong enough to express all computable functions and is the core of all functionel programming languages [26] and of many theorem provers, see e. g. [29],[9]. Higher order unification is also the base for higher order narrowing [32].

G. Huet has introduced an algorithm for unification in the simply typed λ-calculus [19], which is of course not always terminating. D. Miller found a family of terms, he called them *higher order patterns*, for which the unification problem becomes decidable [24]. Any solvable unification problem of patterns has one unique solution. These patterns are simply typed λ-terms in which any free variable could only have pairwise distinct bound variables as arguments. So e. g. only global propertys of functions like $\forall x.f(x) = 0$ could be defined but not a property like $f(0) = 0$.

In this work I try to relax the pattern restriction. The first idea is to allow more complex arguments for free variables. Under the restrictions of linearity (this means every free variable could only appear once in the problem) I was able to show, that unification of these more complex terms is decidable. Furthermore I give an algorithm to compute a complete set of solutions for solvable unification problems, using a new class of solutions called *most relevant unifiers*.

The second idea is to allow constants as arguments for free variables. Then unique solutions do not longer exist in the λ-calculus. Inspired by the work of Beeson [8] I introduce a calculus with case distinction in the form "if-then-else". I show that this calculus is strongly normalising and confluent and that linear unification problems where any free variable occurs only once have an unique solution in this calculus. Furthermore in contrast to [8] I show that general higher order unification in a calculus with case distinction is undecidable, using the undecidability of the Plotkin-Statman Conjecture [22]. Last but not least I give a correct and complete unification algorithm for non-linear unification problems of these terms.

2 Einleitung

2.1 Was ist Unifikation ?

Erstaunlicherweise stellen sich viele Mathematiker unter Unifikation eine Art Geheimwissenschaft vor, zumindest geben fast alle gerne an, noch nie von Unifikation gehört zu haben. Dies erstaunt um so mehr, da jeder Mathematiker täglich Dinge unifiziert, allerdings ohne explizit darüber nachzudenken. Unifikation bedeutet nichts anderes, als zu überlegen, ob zwei Ausdrücke eine gemeinsame Instanz besitzen, also ob sie „gleich gemacht“ werden können. Eine Instanz entsteht dabei aus einem Term, indem man für die in dem Term auftretenden Variablen andere Terme einsetzt. Man nennt diesen Vorgang „simultane Ersetzung“. Grundlage ist stets eine endliche Abbildung von der Menge der Variablen in die Menge der Terme. Diese Abbildungen heissen „Substitutionen“.

Stellen wir uns vor, wir haben folgende Voraussetzungen zur Verfügung:

$$0 \neq 1$$
$$\forall n \in \mathbb{N}.0 \neq n \Rightarrow 0 < n$$

und wollen daraus die Aussage $0 < 1$ beweisen. Ich weiß, dies ist völlig trivial, aber warum? Wie würde ein Rechner vorgehen, bzw. ein Student im ersten Semester? Das Ziel $0 < 1$ ist eine atomare Formel, um sie beweisen zu können, muß eine „ähnliche“ atomare Formel in den Voraussetzungen existieren. Hier findet man $0 < n$. Wir müssen also dafür sorgen, daß n und 1 gleich werden, wir müssen die Terme n und 1 „unifizieren“. Da n eine Allquantifizierte Variable in den Voraussetzungen ist, können wir einfach $n := 1$ setzen und erhalten $0 \neq 1 \Rightarrow 0 < 1$. Nun müssen wir noch $0 < 1$ zeigen, dies folgt aus der ersten Annahme.

Dies sieht alles furchtbar harmlos aus und ist es auch, so lange die Variablen nur für Elemente erster Stufe stehen, also nicht Mengen oder Funktionen etc. repräsentieren. Diese so genannte „erststufige Unifikation“ ist gründlich erforscht, entscheidbar, es existieren sehr effiziente Unifikationsalgorithmen und die Anwendungen sind vielfältig. Der erste Unifikationsalgorithmus stammt wohl von Robinson [34], noch von exponentieller Komplexität. Quasi-lineare Versionen findet man u. a. in [6] und [23]. In [28] findet man gar eine lineare Version. Dabei gilt sogar, daß für jedes lösbare erststufige Unifikationsproblem eine „beste“ Lösung existiert. Dies ist eine Substitution ϑ, so daß jede andere Lösung die Form $\sigma \circ \vartheta$ mit einer Substitution σ hat. Man nennt so ein ϑ dann den „allgemeinsten Unifikator“.

Anwendung findet die erststufige Unifikation u. a. in der Typtheorie [17], bei Termersetzungssystemen [3] und in den Resolutionsmethoden logischer Programmiersprachen [36] wie z. B. Prolog.

Warum wollen wir also diese schöne Welt verlassen und höherstufige Unifikation betreiben? Manche Probleme benötigen einfach höherstufige Variabeln, betrachten wir ein Beispiel:

Sei $\mathsf{ls}(\mathbb{N})$ die Menge aller Listen natürlicher Zahlen mit den üblichen Konstruktoren [] (leere Liste) und cons (Anhängen eines Elements). Dann definieren wir

$\mathsf{map} : (\mathbb{N} \to \mathbb{N}) \times \mathsf{ls}(\mathbb{N}) \to \mathsf{ls}(\mathbb{N})$ durch Rekursion über $\mathsf{ls}(\mathbb{N})$:

$$\begin{array}{ll} \mathsf{map}(f, []) & := [], \\ \mathsf{map}(f, \mathsf{cons}(a, x)) & := \mathsf{cons}(f(a), \mathsf{map}(f, x)). \end{array}$$

Dies ist eine sehr populäre und weit verbreitete Funktion, die die Anwendung einer Funktion auf eine Liste von Argumenten definiert. Allerdings brauchen wir zur Definition von map offensichtlich eine höherstufige Variable f.

Als Sprache, in der man solche höherstufigen Terme definiert, hat sich das λ-Kalkül durchgesetzt [4]. Dieses ist stark genug, alle berechenbaren Funktionen darzustellen und hat eine sehr einfache Syntax. Ausgehend von Mengen von Variablen und Konstanten reduziert es die gesamte Mathematik auf zwei Operationen, „Anwendung“ und „Abstraktion“ sowie eine Rechenregel namens „β-Reduktion“. Die Anwendung besagt, daß man eine Funktion auf ein Argument anwenden kann, die Abstraktion definiert eine Funktion. Die β-Reduktion regelt das Einsetzen des Arguments in den Funktionsterm. Sei z. B. $*$ eine Konstante und x eine Variable, so ist $*xx$ ein (Anwendungs-)Term. Die Konstante $*$, die eine Funktion darstellt wird auf Argumente angewandt. Es ist dann $\lambda x(*xx)$ ein (Abstraktions-)Term, er beschreibt die Funktion $x \mapsto *xx$. Schließlich kann man diese Funktion $\lambda x(*xx)$ wieder auf eine Argument anwenden, etwa die Zahl 5, und erhält $(\lambda x(*xx))5 = *55$ mittels β-Reduktion.

In dieser Arbeit möchten wir nur den „einfach getypten λ-Kalkül“ [5] betrachten. Dieser entsteht aus dem ungetypten λ-Kalkül, indem wir jeder Variable und jeder Konstante einen Typ, also einen Definitions- und einen Wertebereich zuordnen. Es ist dann die Anwendung nur erlaubt, wenn der Typ des Arguments mit dem Definitionsbereich der Funktion übereinstimmt. Als Definitions- und Wertebereiche lassen wir dabei nur Räume der Gestalt $A \to B$ zu. Im Beispiel des letzten Absatzes könnte etwa der Typ von $*$ gleich $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$ lauten. Dann muß die Variable x den Typ $\mathbb{N}$ haben, die Funktion $\lambda x(*xx)$ hat dann den Typ $\mathbb{N} \to \mathbb{N}$. Der Vorteil des einfach getypten λ-Kalküls liegt darin, daß in ihm die Gleichheit modulo β-Reduktion entscheidbar ist.

Der λ-Kalkül ist Grundlage jeder funktionelen Programmiersprache [26] und höherstufige Unifikation ist u. a. Grundlage vieler Theorembeweiser, z. B. Isabelle [16],[29] oder Minlog [10], und von höherstufigen „narrowing“ [32], einer speziellen Form von Termersetzungssystemen. Erwähnt sei noch, daß man höherstufige Unifikation auch in anderen Kalkülen betreiben kann, etwa in einem namenlosen λ-Kalkül [11] mit expliziten Substitutionen [1],[33]. In diesem so genannten $\lambda\sigma$-Kalkül wurde die Unifikation u. a. in den Arbeiten [13],[14] untersucht.

Höherstufige Unifikation bedeutet also, zwei einfach getypte λ-Terme so umzubauen, daß sie modulo β-Konversion gleich werden. Dies funktioniert i. A. leider nicht. Die höherstufige Unifikation ist unentscheidbar [15],[18]. Trotzdem möchte man sie verwenden, so ist sie z. B. in den Beweissystemen „Minlog“ [9], [10] und „Isabelle“ [29] implementiert. Dazu existiert der Algorithmus von Huet zur Bearbeitung höherstufiger Unifikationsprobleme [19], dieser muß allerdings nicht terminieren. Auch produziert er nicht wirklich Unifikatoren, also komplette Lösungen, sondern nur einen wichtigen Bestandteil der Lösungen. Bestimmte lösbare Probleme läßt er unbearbeitet. Wenn seine Anwendung terminiert, so gibt er immerhin an, ob ein Problem überhaupt lösbar ist oder nicht.

Man kann nun also diesen Algorithmus implementieren und Probleme der Terminierung etwa durch künstliche Schranken verhindern, in der Art „nach 100 Rechenschritten höre auf!“. Dies hat sich in der Praxis als sehr nützlich erwiesen. Allerdings löst dies natürlich nicht das Problem, keine Unifikatoren zu erhalten.

Eine andere Möglichkeit besteht darin, die verwendete Sprache einzuschränken, insbesondere das Auftreten freier Variablen nur in speziellen Formen zu erlauben. Erfolgreich war dabei insbesondere D. Miller [24],[25], der seine Terme „höherstufige Muster“ nennt. In diesen darf eine freie Variable f nur auf paarweise verschiedene, „verbotene“ Variablen angewendet werden, also das Vorkommen von f in einem Term hat immer die Form $fx_1, x_2, \ldots, x_n$ mit $x_i \neq x_j$ und alle x_i verbotene Variablen. Dabei bedeutet „verboten“, daß diese Variablen in einer Lösung eines Unifikationsproblems nicht vorkommen dürfen. Man kann dies interpretieren als: Stellt das Symbol f eine Funktion dar, so kann man nur Eigenschaften für f beschreiben, die global für alle Argumente von f gelten. Also z. B. kann man ausdrücken „Für alle $x \in \mathbb{N}$ gilt $fx = 0$“. Man kann aber nicht ausdrücken „$f0 = 0$“. Für diese Muster existiert ein Unifikationsalgorithmus, der terminiert und im Erfolgsfall sogar einen allgemeinsten Unifikator berechnet. Somit verhalten sich diese Muster bzgl. der Unifikation genauso wie erststufige Terme. Allerdings sind die Einschränkungen, die dafür nötig werden, sehr restriktiv. Benutzt man etwa diesen Algorithmus zur Beweissuche [10], so erhält man häufig die Fehlermeldung: „Die verwendeten Terme sind keine Muster !“

2.2 Resultate

Die gerade erwähnten Muster waren der Ausgangspunkt für meine Überlegungen. Ich wollte Terme mit weniger strikten Einschränkungen finden, für die die Unifiation weiterhin entscheidbar ist. Auch suchte ich nach eindeutigen Lösungen für Unifikationsprobleme, wollte mich mit der Entscheidbarkeit allein nicht zufrieden geben. Dabei kamen mir zwei Gedanken der Verbesserung.

Zum Einen die Idee, die Argumente der freien Variablen komplizierter zu gestalten. Dabei entstanden Terme, in denen freie Variablen sogar Argumente haben dürfen, die selbst wieder freie Variablen enthalten. Solche Terme sind meines Wissens bisher noch nicht Gegenstand von Untersuchungen dieser Art gewesen. Allerdings muß man andere Restriktionen in Kauf nehmen, etwa die Linearität der Probleme, d. h. das jede freie Variable nur an höchstens einer Stelle vorkommen darf. Man erhält für diese Terme dann zwar keinen allgemeinsten Unifikator, aber eine endliche Menge von Lösungen. Dazu habe ich eine Methode entwickelt, wie man eine solche Menge von Lösungen für diese speziellen Terme durch eine „allerbeste“ Lösung ausdrücken kann und diese einen „relevantesten Unifikator“ genannt.

Der andere Gedanke ist, auch Konstanten als Argumente freier Variablen zu erlauben. Dies zerstört die Eindeutigkeit der Lösungen im λ-Kalkül, da dort punktweise Definitionen von Funktionen unmöglich sind. Will man z. B. die Terme $f0$ und 0 mit freier Variable f unifizieren, erhält man als einzige Lösungen für f die Funktionen $x \mapsto x$ und $x \mapsto 0$. Günstiger erscheint als Lösung eine

Funktion der Gestalt

$$x \mapsto \mathsf{Wenn}\, x = 0, \mathsf{dann}\, 0, \mathsf{sonst}\, f'0$$

mit einer neuen Variable f'. Die Idee, einen Kalkül mit Fallunterscheidungen an Stelle des λ-Kalküls zu benutzen, fand ich in der Arbeit von Beeson [8]. Leider musste ich feststellen, daß die dort gemachten Versprechungen nicht ganz der Realität entsprechen. So ist auch die höherstufige Unifikation mit Fallunterscheidung unentscheidbar, in dem entsprechenden Kapitel zeige ich, wie aus ihrer Entscheidbarkeit die Plotkin-Statman Vermutung folgt. Und letztere ist leider falsch. Trotzdem kann man die Fallunterscheidung noch nutzen. Entweder wieder unter einer Linearitätsbedingung, dann findet man sogar beste Lösungen für die oben beschriebenen Terme. Oder ohne diese Bedingung, dann findet man einen vollständigen, korrekten aber nicht mehr terminierenden Unifikationsalgorithmus.

3 Grundlagen

Es folgt ein knapper Überblick über die notwendigen Begriffe.

3.1 Der einfach getypte Lambda Kalkül

In dieser Arbeit wird höherstufige Unifikation als Unifikation von Termen im einfach getypten λ-Kalkül verstanden. Deshalb folgt eine kurze Einführung in diesen Kalkül.

Typen und Terme

Als erstes benötigen wir Typen. Wir beschränken uns hier auf einfache, funktionale Typen. Also auf Räume von Funktionen, Funktionalen etc. über einer Menge von Grundtypen.

Definition. *Gegeben sei eine Menge von Grundtypen, genannt Sorten. Daraus definieren wir die* Typen *wie folgt:*

Jede Sorte ist ein Typ. Sind α und β Typen, so auch $\alpha \to \beta$.

Zur Bezeichnung von Typen haben sich die griechischen Buchstaben α, β, γ durchgesetzt. Sorten bezeichnen wir mit ι. Um Klammern zu sparen vereinbaren wir den Operator $\to$ als rechts assoziativ, also $\alpha_1 \to \alpha_2 \to \alpha_3$ steht für $\alpha_1 \to (\alpha_2 \to \alpha_3)$. Jeder Typ hat dann die Gestalt $\alpha_1 \to \cdots \to \alpha_n \to \iota$ mit einer Sorte ι. Die Menge aller Typen über gegebenen Grundtypen bezeichnen wir mit $\mathcal{T}$.

Die Komplexität der Typen ist von entscheidender Bedeutung. Deshalb ordnen wir jedem Typ α eine *Stufe* $\mathsf{ord}(\alpha)$ [1] zu.

Definition. *Unter der* Stufe *eines Typs α verstehen wir die natürliche Zahl $\mathsf{ord}(\alpha)$, die durch Rekursion über α wie folgt berechnet wird:*

$\mathsf{ord}(\iota) := 1$ *für alle Sorten ι und*

$\mathsf{ord}(\alpha_1 \to \cdots \to \alpha_n) := 1 + \mathsf{max}(\{\mathsf{ord}(\alpha_1), \ldots, \mathsf{ord}(\alpha_n)\})$.

Nun definieren wir die einfach getypten λ-Terme. Stellen Typen z. B. Funktionenräume dar, so kann man die Terme als Funktionen darin auffassen. Wir verwenden dabei eine interne Typisierung, d. h. wir lassen grundsätzlich nur korrekt getypte Terme zu. Dies bedeutet das eine Funktion stets nur auf Argumente ihres Definitionsbereichs angewendet werden darf.

Definition. *Zu jedem Typ α sei eine abzählbare, unendliche Menge von Variablen $V_\alpha = \{ x_n^\alpha \mid n \in \mathbb{N} \}$ gegeben. Es sei $\mathcal{V} := \bigcup_{\alpha \in \mathcal{T}} V_\alpha$ die Menge aller Variablen.*

[1] Vom englischen *order*.

Zu jedem Typ α sei eine endliche Menge $C_\alpha := \{ c_1, \ldots, c_n \mid n \geq 0 \}$[2] von Konstanten gegeben. Es sei $\mathcal{C} := \bigcup_{\alpha \in \mathcal{T}} C_\alpha$ die Menge aller Konstanten.

Dann ist die Menge der einfach getypte λ-Terme *über $\mathcal{V}$ und $\mathcal{C}$ induktiv definiert durch:*

Für alle $\alpha \in \mathcal{T}$ ist jede Variable $x \in V_\alpha$ und jede Konstante $c \in C_\alpha$ ein Term vom Typ α.
Ist r ein Term vom Typ $\alpha \to \beta$ und s ein Term vom Typ α, so ist (rs) ein Term vom Typ β.
Ist r ein Term vom Typ β und $x \in V_\alpha$, so ist $(\lambda x(r))$ ein Term vom Typ $\alpha \to \beta$.

Die Menge aller einfach getypten λ-Terme über $\mathcal{V}$ und $\mathcal{C}$ bezeichnen wir auch mit $\mathsf{T}_\lambda(\mathcal{V},\mathcal{C})$ oder kurz mit T_λ, falls $\mathcal{V}$ und $\mathcal{C}$ keine Rolle spielen. Die Operation (rs) in der zweiten Regel heißt *Anwendung*, die Operation $\lambda x(r)$ in der letzten Zeile heißt *Abstraktion*. Analog dazu heißt ein Term der Gestalt (rs) *Anwendungsterm* und ein Term der Gestalt $(\lambda x(r))$ *Abstraktionsterm*. Zum Einsparen von Klammern vereinbaren wir hier, daß die Anwendung stärker bindet als die Abstraktion und daß mehrere Anwendungen links geklammert, mehrere Abstraktionen rechts geklammert zu lesen sind. Ein Punkt zwischen abstrahierter Variable und Term ersetzt ein Klammerpaar, d. h. $\lambda x, y.rst$ bedeutet $(\lambda x(\lambda y((rs)t)))$. Den Typ eines Terms r bezeichnen wir mit $\tau(r)$. Wir schreiben dann kurz $r^{\tau(r)}$ falls der Typ des Terms von Bedeutung ist und lassen diese *Typdekoration* weg, falls der Typ von r klar oder unwichtig ist.

Bemerkung 3.1 *Bis auf Widerruf sind im Folgenden alle mit „Term“ bezeichneten Objekte einfach getypte λ-Terme gemäß obiger Definition.*

Die Variablen und Konstanten fassen wir unter dem Begriff *Atom* zusammen. Folgende Bezeichnungen wollen wir festlegen: Terme werden mit r,s und t notiert, Konstanten mit c und d, Atome mit A und B. Variablen sollen zunächst einmal mit u,v,w,x,y und z bezeichnet werden. Später werden wir noch unterschiedliche Arten von Variablen kennen lernen, dann werden wir diese Liste sinnvoll ergänzen.

Definition. *Die Menge der* Teilterme $\mathsf{Tt}(r)$ *eines Terms r ist definiert durch Rekursion über r.*

$\mathsf{Tt}(A) = \{A\}$ *für jedes Atom A,* $\mathsf{Tt}(rs) = \{rs\} \cup \mathsf{Tt}(r) \cup \mathsf{Tt}(s)$, $\mathsf{Tt}(\lambda x.r) = \{\lambda x.r\} \cup \mathsf{Tt}(r)$.

Die Menge der echten Teilterme *von r ist dann* $\mathsf{Tt}(r) \setminus \{r\}$.

Variablen in einem Term können gebunden sein oder auch nicht, deshalb definieren wir zu jedem Term r die Menge der in ihm gebundenen und freien Variablen.

Definition. *Sei r ein Term. Die Menge* $\mathsf{FV}(r)$[3] *der freien Variablen und die*

[2] $n = 0$ bedeutet $\mathcal{C}_\alpha = \emptyset$.
[3] Vom englischen *free variable*.

Menge $\mathsf{BV}(r)$[4] *der gebundenen Variablen wird durch Rekursion über* r *definiert:*

$\mathsf{FV}(x) := \{x\}$, $\mathsf{FV}(c) := \emptyset$, $\mathsf{FV}(rs) := \mathsf{FV}(r) \cup \mathsf{FV}(s)$, $\mathsf{FV}(\lambda x.r) := \mathsf{FV}(r) \setminus \{x\}$.

$\mathsf{BV}(x) := \mathsf{BV}(c) := \emptyset$, $\mathsf{BV}(rs) := \mathsf{BV}(r) \cup \mathsf{BV}(s)$, $\mathsf{BV}(\lambda x.r) := \mathsf{BV}(r) \cup \{x\}$.

Nun können wir die Stufe eines Terms definieren. Dieser Begriff trennt die entscheidbare Welt erststufiger Unifikation von der unentscheidbaren Welt der höherstufigen Unifikation.

Definition. *Die Stufe eines Terms* r*, bezeichnet mit* $\mathsf{ord}(r)$[5]*, ist definiert als* $\mathsf{max}\{\, \mathsf{ord}(\tau(x)) \mid x \in \mathsf{FV}(r) \,\}$.

Höherstufige Unifikation bedeutet also nichts anderes, daß in einem Term freie Variablen höherer Stufe auftreten dürfen.

Umbenennung gebundener Variablen, simultane Ersetzung

In einem Term kann die selbe Variable sowohl gebunden als auch frei auftreten, ein störender Umstand. Auch kann der selbe Variablenname mehrfach abstrahiert werden. Um dies zu verhindern führen wir die *Umbenennung gebundener Variablen* ein, oft als α-Konversion bezeichnet. Dazu benötigen wir zunächst den Begriff der *simultanen Ersetzung*[6] von Variablen in einem Term.

Definition. *Seien* $x_1, \ldots, x_n$ *paarweise verschiedene Variablen und* $s_1, \ldots, s_n$ *Terme mit* $\tau(s_i) = \tau(x_i)$ *für alle* $1 \leq i \leq n$*. Die simultane Ersetzung von* $x_1, \ldots, x_n$ *durch* $s_1, \ldots, s_n$ *in einem Term* r *bezeichnen wir mit* $r[\vec{x}_n ::= \vec{s}_n]$ *und definieren sie durch Rekursion über* r*.*

$y[\vec{x}_n ::= \vec{s}_n] := s_i$ *falls* $y = x_i$, $y[\vec{x}_n ::= \vec{s}_n] := y$ *sonst.*
$c[\vec{x}_n ::= \vec{s}_n] := c$.
$rs[\vec{x}_n ::= \vec{s}_n] := r[\vec{x}_n ::= \vec{s}_n]s[\vec{x}_n ::= \vec{s}_n]$.
$(\lambda y.r)[\vec{x}_n ::= \vec{s}_n] := \lambda z.(r[y ::= z])[\vec{x}_n ::= \vec{s}_n]$*, wobei* z *eine neue Variable ist, d. h.* $z \notin \mathsf{FV}(r) \setminus \{y\}$ *und für alle* $1 \leq i \leq n$ *gilt* $z \neq x_i$ *und* $z \notin \mathsf{FV}(s_i)$*.*

Wollen wir nur eine einzelne Variable ersetzen so schreiben wir auch kurz $r_x[s]$ für $r[x ::= s]$. Die letzte Klausel ist so gestaltet, daß das „Einfangen“ freier Variablen nicht möglich ist, d. h. freie Variablen in den s_i bleiben auch nach der Ersetzung freie Variablen, sofern sie nicht gänzlich verschwinden.

Nun können wir definieren, wann zwei Terme sich nur durch Umbenennung gebundener Variablen unterscheiden.

Definition. *Wir definieren* $r \equiv_\alpha s$ *als kompatiblen, reflexiven, symmetrischen und transitiven Abschluß folgender Relation:*

$\lambda x.r \equiv_\alpha \lambda y.r_x[y]$ *für alle Terme* r *und alle Variablen* x*,*y *mit* $\tau(x) = \tau(y)$ *und* $y \notin \mathsf{FV}(r)$*.*

[4] Vom englischen *bound variable*.
[5] Wieder vom englischen *order*.
[6] Auch *simultane Substitution*genannt. Auf Substitutionen gehen wir genauer im folgenden Abschnitt über Unifikation ein.

Dabei bedeutet „kompatibler Abschluß“, daß aus $r \equiv_\alpha r'$ folgt: $rs \equiv_\alpha r's$, $sr \equiv_\alpha sr'$, $\lambda x.r \equiv_\alpha \lambda x.r'$.

Somit können wir jeden Term als Repräsentanten einer ganzen Äquivalenzklasse auffassen. Dies erlaubt uns stets den Term r auszuwählen, der folgende Eigenschaften erfüllt: $\mathsf{BV}(r) \cap \mathsf{FV}(r) = \emptyset$ und jedes $x \in \mathsf{BV}(r)$ ist nur an genau einer Stelle abstrahiert.

Bemerkung 3.2 *Wenn wir im folgenden von dem Term r sprechen, so meinen wir also einen Repräsentanten einer $\equiv_\alpha$ −Äquivalenzklasse mit obiger Variablenbedingung.*

β-Reduktion

Nun müssen wir definieren, wie man ein Argument in eine Funktion einsetzt. Dieser Prozess heißt *β-Reduktion*.

Definition. *Ein Term der Form $(\lambda x.r)s$ heißt* β-Redex. *Die* β-Konversion $\mapsto_\beta$ *eines β-Redex ist definiert durch $(\lambda x.r)s \mapsto_\beta r_x[s]$. Die* β-Reduktion $\to_\beta$ *ist definiert als kompatibler Abschluß von $\mapsto_\beta$:*

Gilt $r \mapsto_\beta s$ so auch $r \to_\beta s$.
Gilt $r \to_\beta s$, so auch $rt \to_\beta st$, $tr \to_\beta ts$ und $\lambda x.r \to_\beta \lambda x.s$.

Es bezeichne dann $\to_\beta^+$ den transitiven Abschluß von $\to_\beta$ und $\to_\beta^*$ den reflexiven Abschluß von $\to_\beta^+$. Ein Term, der keinen β-Redex als Teilterm enthält, heißt in β-Normalform. *Ist s in β-Normalform und gilt $r \to_\beta^* s$, so heißt s* eine β-Normalform *von r.*

Den unbestimmten Artikel „eine“ im letzten Satz der Definition können wir gleich durch den bestimmten Artikel „die“ ersetzen, dank des folgenden bekannten Theorems.

Theorem 3.1 *Die β-Reduktion für einfach getypte λ-Terme ist stark normalisierend und konfluent.*

Beweise dieses Satzes findet man in jedem Buch über das λ-Kalkül. Empfehlenswert ist insbesondere die Lektüre des Artikels von Joachimski/Matthes [20]. Die dortige innovative Beweisidee verwenden wir dann später im Kapitel über Fallunterscheidung.

Der Satz sagt aus, daß wir zu jedem Term r einen eindeutig bestimmten Term s in β-Normalform finden mit $r \to_\beta^* s$. Diese β-Normalform von r bezeichnen wir mit $r\downarrow_\beta$. Terme in β-Normalform kann man wie folgt induktiv charakterisieren:

- Alle Variabeln x und alle Konstanten c sind Terme in β-Normalform.
- Sind $r_1, \ldots, r_n$ Terme in β-Normalform, so ist auch $Ar_1 \cdots r_n$ für jedes passende Atom A in β-Normalform.
- Ist r ein Term in β-Normalform, so ist auch $\lambda x.r$ für jede Variable x ein Term in β-Normalform.

Daß jeder β-normale Term r von dieser Gestalt ist, ist sehr leicht durch Induktion über $r \in \mathsf{T}_\lambda$ zu zeigen. Folglich hat jeder Term in β-Normalform die Gestalt $\lambda x_1, \ldots, x_m.Ar_1 \cdots r_n$ mit ebenfalls β-normalen Termen $r_1, \ldots, r_n$. Abkürzend schreiben wir dafür auch $\lambda\vec{x_m}.A\vec{r_n}$, wobei $m = 0$ bzw. $n = 0$ einen leeren Vektor anzeigt. Das Atom A heißt *Kopf* des Terms, die Terme $r_1, \ldots, r_n$ Argumente des Terms. Noch kürzer schreiben wir $\lambda\vec{x}.A\vec{r}$ falls uns die Anzahl der abstrahierten Variabeln bzw. die Anzahl der Argumente nicht interessiert. Zur Vereinfachung vereinbaren wir für einen Vektor $\vec{r_n}$: $\{\vec{r_n}\} := \{r_1, \ldots, r_n\}$.

Durch die β-Reduktion können Teilterme eines Terms verschwinden. Deshalb wollen wir ab hier vereinbaren, daß ein Term s als Teilterm eines Terms r bezeichnet wird, falls $s \in \mathsf{Tt}(r \downarrow_\beta)$ ist.

Lange Normalform

Nun stört nur noch eins, nämlich daß Atome A mit unterschiedlich vielen Argumenten auftreten können. Genauer gesagt kann ein Atom A vom Typ $\alpha_1 \to \cdots \to \alpha_n \to \iota$ mit bis zu n Argumenten auftreten. Wir wollen erreichen, daß dieses Atom mit genau n Argumenten auftritt, dazu definieren wir den Begriff der η-Äquivalenz.

Definition. *Die Relation $r \equiv_\eta s$ auf β-normalen Termen r und s ist definiert als reflexiver, symmetrischer und transitiver Abschluß folgender Regeln:*

$Ar_1 \cdots r_n \equiv_\eta \lambda x.Ar_1 \cdots r_n x$ für alle $x \in \mathcal{V}$ passenden Typs mit $x \notin \mathsf{FV}(A\vec{r_n})$.

Gilt $r_i \equiv_\eta s_i$ für alle $1 \leq i \leq n$, so auch $A\vec{r_n} \equiv_\eta A\vec{s_n}$.

Gilt $r \equiv_\eta s$, so auch $\lambda x.r \equiv_\eta \lambda x.s$ für alle Variablen x.

Damit ist $\equiv_\eta$ eine Äquivalenzrelation über β-normalen Termen. Für β-normale Terme definieren wir die Relationen $\equiv$ als kompatiblen, symmetrischen, reflexiven und transitiven Abschluß der Relationen $\equiv_\alpha$ und $\equiv_\eta$.

Zu jedem β-normalen Term $r \equiv \lambda\vec{x}.A\vec{r}$ finden wir einen β-normalen Term $s \equiv \lambda\vec{y}.A\vec{s}$ mit $\tau(A\vec{s})$ Grundtyp und $r \equiv s$. Dieses s nennen wir die *lange Normalform* von r. Dies erlaubt uns nun, die Gleichheit zweier Terme zu definieren.

Definition. *Zwei Terme r und s heißen gleich falls $r \downarrow_\beta \equiv s \downarrow_\beta$. Wir schreiben dann $r = s$.*

3.2 Unifikation

Nun wollen wir uns genauer mit Unifikationsproblemen und Substitutionen beschäftigen.

Substitutionen

Definition. Substitutionen *sind Typ erhaltende, endliche Abbildungen von der*

Menge der Variablen in die Menge der Terme. Also $\vartheta\colon \mathcal{V} \to \mathsf{T}_\lambda, x \mapsto r$ *ist eine Substitution, falls* $\tau(\vartheta(x)) = \tau(x)$ *für alle* $x \in \mathcal{V}$ *ist und* $\mathsf{dom}(\vartheta) := \{\, x \in \mathcal{V} \mid \vartheta(x) \neq x \,\}$[7] *eine endliche Menge ist.*

Für Substitutionen verwenden wir griechische Buchstaben $\vartheta, \theta, \varrho, \rho, \sigma$. Die *leere* bzw. *identische* Substitution bezeichnen wir mit ε. Eine Substitution ϑ mit $\mathsf{dom}(\vartheta) = \{x_1, \ldots, x_n\}$ und $\vartheta(x_i) =: r_i$ notiert man als Menge ihrer *Bindungen* $\langle x_i, r_i\rangle$ in der Form $\vartheta := \{\langle x_1, r_1\rangle, \ldots, \langle x_n, r_n\rangle\}$[8]. Es ist also z. B. $\varepsilon = \emptyset$. Weiterhin bezeichnen wir mit $\mathsf{rng}(\vartheta) := \{r_1, \ldots, r_n\}$[9] den *Wertebereich* von ϑ und mit $\mathsf{FV}(\vartheta) := \bigcup_{r \in \mathsf{rng}(\vartheta)} \mathsf{FV}(r)$ die Menge der *freien Variablen* von ϑ. Ist $V \subseteq \mathcal{V}$ eine Menge von Variabeln, so sei die *Beschränkung* einer Substitution ϑ auf V definiert als $\vartheta\,|_V := \{\, \langle f, r\rangle \in \vartheta \mid f \in V \,\}$.

Die Anwendung einer Substitution $\vartheta := \{\langle x_1, r_1\rangle, \ldots, \langle x_n, r_n\rangle\}$ auf einen Term s entspricht der simultanen Ersetzung der Variablen x_i in s durch die Terme r_i. Also

$$\vartheta(s) := (s[\vec{x}_n ::= \vec{r}_n]).$$

Es gilt demnach $\vartheta(rs) = \vartheta(r)\vartheta(s)$ und $\vartheta(\lambda x.r) = \lambda x.\vartheta(r)$ falls $x \notin \mathsf{dom}(\vartheta) \cup \mathsf{FV}(\vartheta)$.

Die Verknüpfung zweier Substitutionen ϑ und ϱ ist einerseits schlicht die Komposition der Abbildungen ϑ und ϱ, also $(\vartheta \circ \varrho)(x) := \vartheta(\varrho(x))$, anderenseits können wir für $\vartheta := \{\langle x_1, r_1\rangle, \ldots, \langle x_n, r_n\rangle\}$ und $\varrho := \{\langle y_1, s_1\rangle, \ldots, \langle y_m, s_m\rangle\}$ definieren: $\vartheta \circ \varrho := \{\langle x_{i_1}, r_{i_1}\rangle, \ldots, \langle x_{i_k}, r_{i_k}\rangle, \langle y_1, \vartheta(s_1)\rangle, \ldots, \langle y_m, \vartheta(s_m)\rangle\}$, wobei $\{x_{i_1}, \ldots, x_{i_k}\} := \{x_1, \ldots, x_n\} \setminus \{y_1, \ldots, y_m\}$ ist.

Bemerkung 3.3 *Seien* ϑ *und* ϱ *Substitutionen mit*

$$\mathsf{dom}(\vartheta) \cap \mathsf{dom}(\varrho) = \mathsf{dom}(\vartheta) \cap \mathsf{FV}(\varrho) = \mathsf{FV}(\vartheta) \cap \mathsf{dom}(\varrho) = \emptyset.$$

Dann gilt $\vartheta \circ \varrho = \varrho \circ \vartheta$.

Beweis. Es gilt für alle $\langle x, r\rangle \in \vartheta$: $\varrho(r) = r$. Ebenso gilt für alle $\langle x, r\rangle \in \varrho$: $\vartheta(r) = r$. Sei $\langle x, r\rangle \in (\vartheta \circ \varrho)$. Dann ist entweder $\langle x, r\rangle \in \vartheta$ und dann wegen $\varrho(r) = r$ auch $\langle x, r\rangle \in (\varrho \circ \vartheta)$ oder $\langle x, r\rangle \in \varrho$ und dann wegen $x \notin \mathsf{dom}(\vartheta)$ auch $\langle x, r\rangle \in (\varrho \circ \vartheta)$. Ganz analog folgt für alle $\langle x, r\rangle \in (\varrho \circ \vartheta)$: $\langle x, r\rangle \in (\vartheta \circ \varrho)$. ■

Substitutionen ϑ und ϱ wie in obiger Bemerkung 3.3 wollen wir *parallel* nennen, i. Z. $\vartheta \| \varrho$.

Lemma 3.1 *Ist* r *ein Term,* ϑ *eine Substitution und* s *die lange Normalform von* r*, so gilt* $\vartheta(r) = \vartheta(s)$.

Beweis. Durch Induktion über $\to_\beta$ zeigen wir zunächst $\vartheta(r) = \vartheta(r\downarrow_\beta)$. Es genügt zu zeigen $\vartheta((\lambda x.r)s) = \vartheta(r_x[s])$. Wir können $x \notin \mathsf{dom}(\vartheta) \cup \mathsf{FV}(\vartheta)$ voraussetzen. Dann ist $\vartheta((\lambda x.r)s) = (\lambda x.\vartheta(r))\vartheta(s) = \{\langle x, \vartheta(s)\rangle\}(\vartheta(r)) = (\vartheta \circ \{\langle x, s\rangle\})(r) = \vartheta(r_x[s])$. Die restlichen Fälle ergeben sich durch Anwendung der Induktionshypothese.

[7] Vom englischen *domain*.
[8] Es gilt dabei stets $i \neq j \iff x_i \neq x_j$.
[9] Vom englischen *range*.

Durch Induktion über $\equiv_\eta$ zeigen wir den Rest. Es genügt zu zeigen $\vartheta(\lambda x.rx) = \vartheta(r)$ falls $x \notin \mathsf{FV}(r) \cup \mathsf{dom}(\vartheta) \cup \mathsf{FV}(\vartheta)$. Dies gilt offensichtlich. Die anderen Fälle ergeben sich wieder durch direkte Anwendung der Induktionshypothese. ■

Definition. *Eine Substitution ϑ heißt* allgemeiner *als eine Substitution ϱ bzgl. einer Menge $V \subseteq \mathcal{V}$, falls es eine Substitution σ gibt mit $(\sigma \circ \vartheta)(x) = \varrho(x)$ für alle $x \in V$. Wir schreiben dann $\vartheta \geq_V \varrho$.*[10]

Für $V := \mathsf{dom}(\vartheta)$ schreiben wir kurz $\vartheta \geq \varrho$.

Unifikationsprobleme, verbotene und flexible Variablen

Definition. *Ein* Unifikationsproblem UP *ist ein Tripel* $(\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *bestehend aus:*

1. *Einer Menge* $\mathrm{U} := \{(r_1, s_1), \ldots, (r_n, s_n)\}$ *von Paaren von Termen gleichen Typs, d. h. für alle* $1 \leq i \leq n$ *gilt* $\tau(r_i) = \tau(s_i)$. *Wir nenne* U *die* Menge der Unifikationspaare.
2. *Einer Menge* forb[11] *von Variablen, genannt* verbotene Variablen.
3. *Einer Menge* flex[12] *von Variablen, genannt* flexible Variablen.

Ein Unifikationsproblem $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *heißt* sauber, *falls* $\mathsf{forb} \cap \mathsf{flex} = \emptyset$ *ist und mit* $\mathsf{FV}(\mathsf{UP}) := \bigcup_{(r,s)\in \mathrm{U}} \mathsf{FV}(r) \cup \mathsf{FV}(s)$ *gilt* $\mathsf{FV}(\mathsf{UP}) \setminus \mathsf{forb} \subseteq \mathsf{flex}$.

Im folgenden wollen wir nur saubere Unifikation betreiben. Dies können wir stets erreichen indem wir zu gegebenen U und forb einfach $\mathsf{flex} := \mathsf{FV}(\mathrm{U}) \setminus \mathsf{forb}$ setzen. Sprechen wir von einem Term in U oder kurz $r \in \mathrm{U}$, so meinen wir, daß ein Term s existiert mit $(r, s) \in \mathrm{U}$ oder $(s, r) \in \mathrm{U}$.

Definition. *Für ein Unifikationsproblem* $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *und eine Substitution ϑ definieren wir* $\vartheta(\mathsf{UP}) := (\mathrm{U}', \mathsf{forb}', \mathsf{flex}')$ *wie folgt:*

- $\mathrm{U}' := \bigcup_{(r,s)\in \mathrm{U}} \{(\vartheta(r), \vartheta(s))\}$.
- $\mathsf{forb}' := \mathsf{forb}$.
- $\mathsf{flex}' := (\mathsf{flex} \setminus \mathsf{dom}(\vartheta)) \cup \mathsf{FV}(\vartheta)$.

Ist UP ein sauberes Unifikationsproblem, so ist $\vartheta(\mathsf{UP})$ genau dann ein sauberes Unifikationsproblem, wenn $\mathsf{FV}(\vartheta) \cap \mathsf{forb} = \emptyset$ ist.

Definition. *Ein Unifikationsproblem* $\mathsf{UP} = (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *heißt* bereit, *falls alle $r \in \mathrm{U}$ Anwendungsterme in β-Normalform sind mit $\tau(r)$ Grundtyp.*

[10] $\geq$, da via ε jede Substitution allgemeiner als sie selbst ist.
[11] Vom englischen *forbidden*.
[12] Vom englischen *flexible*.

Algorithmus 1 Aufbereitung eines Unifikationsproblems.

Sei $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein Unifikationsproblem. Die* Aufbereitung *von* UP *ist definiert durch folgenden Algorithmus:*

- *Ist* UP *bereit, so gib* $(\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *zurück.*
- *Sonst wiederhole bis* UP *bereit ist:*

  Sei $(r, s) \in \mathrm{U}$ *mit* r *oder* s *kein Anwendungsterm vom Grundtyp. Seien* $\tilde{r} := \lambda\vec{x}_n.t_1$, $\tilde{s} := \lambda\vec{y}_n.t_2$ *lange Normalformen von* r *bzw.* s. *Seien* $v_1, \ldots, v_n$ *neue Variablen*[13] *mit jeweils* $\tau(v_i) = \tau(x_i) = \tau(y_i)$. *Dann existieren Terme* r', s' *mit* $r = \lambda\vec{v}_n.r'$, $s = \lambda\vec{v}_n.s'$. *Ersetze* (r, s) *in* U *durch* (r', s') *und ersetze* forb *durch* $\mathsf{forb} \cup \{v_1, \ldots, v_n\}$.

Offensichtlich ist dies ein terminierendes Verfahren, daß schließlich ein bereites Unifikationsproblem zurück gibt. Dieses wollen wir eine *aufbereitete Form* von UP nennen. Dabei ist auch klar, daß sich die aufbereiteten Formen eines Unifikationsproblems nur durch die Namensgebung der verbotenen Variablen unterscheiden. Da diese stets neue Variablen sind spielen die Namen keine Rolle, wir sprechen deshalb auch von *der* aufbereiteten Form von UP.

Definition. *Ein* Unifikator *eines Unifikationsproblems* $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ist eine Substitution* ϑ, *für die gilt:*

1. *Für alle* $(r, s) \in \mathrm{U}$ *ist* $\vartheta(r) = \vartheta(s)$.
2. $\mathsf{dom}(\vartheta) \subseteq \mathsf{flex}$.
3. $\mathsf{FV}(\vartheta) \cap \mathsf{forb} = \emptyset$.
4. $\mathsf{FV}(\vartheta) \cap \mathsf{flex} = \emptyset$.

Eine Unifikator ϑ *von* UP *heißt* allgemeinster Unifikator *, falls für alle Unifikatoren* ϱ *von* UP *gilt* $\vartheta \geq_{\mathsf{flex}} \varrho$. *Eine Menge* M *von Unifikatoren von* UP *heißt* vollständig, *kurz* CSU[14], *falls für jeden Unifikator* ϱ *von* UP *ein Unifikator* $\vartheta \in M$ *existiert mit* $\vartheta \geq_{\mathsf{flex}} \varrho$.

Lemma 3.2 *Sei* UP *eine Unifikationsproblem und* UP' *seine aufbereitete Form. Dann ist jeder Unifikator von* UP *auch Unifikator von* UP' *und umgekehrt.*

Beweis. Sei $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$, $\mathsf{UP}' := (\mathrm{U}', \mathsf{forb}', \mathsf{flex})$. Sei ϑ ein Unifikator von UP. Sei $(r', s') \in \mathrm{U}'$. Dann existiert ein Paar $(\lambda\vec{x}.r, \lambda\vec{y}.s) \in \mathrm{U}$ mit langer Normalform $\lambda\vec{v}.r'$ bzw. $\lambda\vec{v}.s'$. Es gelte dabei $\{\vec{v}\} \notin \mathsf{FV}(\vartheta) \cup \mathsf{dom}(\vartheta)$. Nach dem Lemma 3.1 gilt $\vartheta(\lambda\vec{v}.r') = \vartheta(\lambda\vec{x}.r) = \vartheta(\lambda\vec{y}.s) = \vartheta(\lambda\vec{v}.s')$. Wegen $\vartheta(\lambda\vec{v}.r') = \lambda\vec{v}.\vartheta(r')$ und $\vartheta(\lambda\vec{v}.s') = \lambda\vec{v}.\vartheta(s')$ folgt $\vartheta(r') = \vartheta(s')$.

Die Rückrichtung geht analog. ■

[13] *Neu* bedeutet, daß sie noch nirgends auftreten.
[14] Vom englischen *complete set of unifiers*.

Damit können wir immer mit der aufbereiteten Form eines Unifikationspaares arbeiten. Für ein solches Unifikationsproblem $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ benennen wir die Paare $(A\vec{r}, B\vec{s}) \in \mathrm{U}$ wie folgt:

- *Starr-Starr*: $A, B \in \mathsf{forb} \cup \mathcal{C}$.
- *Flex-Starr*: $A \in \mathsf{flex}$ und $B \in \mathsf{forb} \cup \mathcal{C}$.
- *Starr-Flex*: $A \in \mathsf{forb} \cup \mathcal{C}$ und $B \in \mathsf{flex}$.
- *Flex-Flex*: $A, B \in \mathsf{flex}$.

Analog dazu bezeichnen wir einen Term $r \in \mathrm{U}$ als starr bzw. flexibel, je nachdem ob der Kopf von r in $\mathsf{forb} \cup \mathcal{C}$ oder in flex ist.

Definition. *Ein Unifikationsproblem* $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *heißt* linear*, falls für jede Variable* $f \in \mathsf{flex}$ *gilt: Es gibt höchstens einen Term* $r \in \mathrm{U}$ *mit* $f \in \mathsf{FV}(r)$ *und* f *kommt in diesem* r *an höchstens einer Stelle frei vor.*

Unentscheidbarkeit der höherstufigen Unifikation

Im Gegensatz zur erststufigen Unifikation ist die höherstufige Variante nicht entscheidbar. In erster Linie ist dabei die Arbeit von Goldfarb [15] zu nennen, der zeigt, daß bereits die Entscheidbarkeit der zweitstufigen Version ausreicht um Hilbert's zehntes Problem der diophantischen Gleichungen zu lösen, welches aber unlösbar ist. Schon zuvor hatte G. Huet [18] die Unentscheidbarkeit für drittstufige Terme bewiesen. Anschaulich werden die Schwierigkeiten auch an den folgenden Beispielen.

Sei $\mathsf{UP} := (\{(f(cd), c(fd)), \emptyset, \{f\})$ mit $c, d \in \mathcal{C}$. Definieren wir $c^0 r := r$ und $c^{n+1} r = c(c^n r)$, so ist $\vartheta_n := \{\langle f, \lambda x.c^n x\rangle\}$ für alle $n \in \mathbb{N}$ ein Unifikator von UP. Es gilt aber für alle $i \neq j$: $\vartheta_i \not\geq \vartheta_j$ und $\vartheta_j \not\geq \vartheta_i$. Also kann kein vollständiger, terminierender Algorithmus existieren, der eine vollständige Menge von Unifikatoren berechnet.

Ganz genauso verhält es sich mit $\mathsf{UP} := (\{(f(cdd), c(fd)(fd))\}, \emptyset, \{f\})$, $c, d \in \mathcal{C}$. Hier haben die Unifikatoren die Form

$$\{\langle f, \lambda x.x\rangle\}, \{\langle f, \lambda x.cxx\rangle\}, \{\langle f, \lambda x.c(cxx)(cxx)\rangle\}, \ldots.$$

Und all diese sind wieder nicht mittels $\geq$ vergleichbar.

Es bleibt also festzuhalten, daß höherstufige Unifikation prinzipiell nicht entscheidbar ist und daß lösbare, relativ einfache Unifikationsprobleme existieren, die keine endliche, vollständige Menge von Unifikatoren besitzen.

3.3 Der Algorithmus von Huet

Dieser Algorithmus stellt ein erstes und bis heute grundlegendes Verfahren zur Unifikation dar. Leider muß er im allgemeinen nicht terminieren, was aus der Unentscheidbarkeit folgt. Auch produziert er keine Unifikatoren sondern nur so

genannte Preunifikatoren, da er Flex-Flex-Paare ungelöst läßt. Die Darstellung in diesem Abscnitt ist sehr kompakt und beschränkt sich auf das Wesentliche. Daß wir in dieser Arbeit mit einem modifizierten Begriff von Unifikationsproblemen hantieren, hat keinen entscheidenden Einfluß auf den Algorithmus und die Resultate, die notwendigen Anpassungen sind nicht explizit erwähnt. Ausführliche Informationen zum Algorithmus findet man in der sehr empfehlenswerten Originalarbeit von G. Huet [19].

Der Algorithmus besteht im Wesentlichen aus den zwei Prozeduren SIMPL und MATCH. Die Prozedur SIMPL bearbeitet alle Paare die nicht Flex-Flex oder Flex-Starr sind. Dabei sind zwei starre Terme genau dann unifizierbar, wenn sie die gleichen Köpfe haben und die Argumente jeweils paarweise unifizierbar sind.

Prozedur 3.1 SIMPL(UP).

Sei $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein Unifikationsproblem. Die Prozedur* SIMPL(UP) *ist dann wie folgt definiert:*

1. *Ersetze* UP *durch seine aufbereitete Form (siehe Algorithmus 1).*
2. *Sind alle Paare in* U *entwerder Flex-Flex oder Flex-Starr oder ist* $\mathsf{UP} := \bot$*, so gib* UP *aus und halte an. Sonst gehe zu 3.*
3. *Sei* $(A\vec{r_n}, B\vec{s_m}) \in \mathrm{U}$ *weder Flex-Flex noch Flex-Starr.*
   (a) *Ist* $A \in \mathsf{forb} \cup \mathcal{C}$ *und* $B \in \mathsf{flex}$*, so ersetze* $(A\vec{r}, B\vec{s})$ *in* U *durch* $(B\vec{s}, A\vec{r})$ *und gehe mit diesem neuen* UP *zurück zu 1.*
   (b) *Sind* $A, B \in \mathsf{forb} \cup \mathcal{C}$ *und ist* $A \neq B$*, so setze* $\mathsf{UP} := \bot$ *und gehe zu 2.*
   (c) *Sind* $A, B \in \mathsf{forb} \cup \mathcal{C}$ *und ist* $A = B$*, so ersetze* $(A\vec{r}, B\vec{s})$ *in* U *durch* $(r_1, s_1), \ldots, (r_n, s_n)$ *und gehe zu 1.*

Nun zeigen wir, daß obige Prozedur terminiert und die Menge der Unifikatoren eines Unifikationsproblems nicht ändert.

Lemma 3.3 *Sei* UP *ein Unifikationsproblem. Dann gilt:*

1. *Die Prozedur* SIMPL(UP) *terminiert.*
2. *Ist* $\mathsf{SIMPL}(\mathsf{UP}) = \bot$*, so ist* UP *nicht unifizierbar.*
3. *Ist* $\mathsf{SIMPL}(\mathsf{UP}) \neq \bot$*, so ist eine Substitution ein Unifikator von* UP *gdw. sie Unifikator von* SIMPL(UP) *ist.*

Beweis.

1. Wir definiere dazu ein Termmaß wie folgt: $|Ar_1 \cdots r_n| := n + \sum_{i=1}^{n} |r_i|$. Dann sei $|\mathrm{U}| := \sum_{r \in \mathrm{U}} |r|$. Außerdem sei $\mathsf{sf}(\mathrm{U})$ die Anzahl der Starr-Flex-Paare in U und $\#\mathrm{U}$ einfach die Anzahl der Unifikationspaare in U. So erhalten wir das Tripel $(|\mathrm{U}|, \mathsf{sf}(\mathrm{U}), \#\mathrm{U})$. Auf diesen Tripeln natürlicher Zahlen betrachten wir nun die übliche lexikographische Ordnung. Nun erkennen wir ohne Mühe, daß dann in jedem Schritt der obigen Prozedur SIMPL dieses Tripel echt kleiner wird bzw. die Prozedur mit $\bot$ terminiert.

2. Dies schließen wir unmittelbar aus der Tatsache, daß für einen starren Term $Ar_1 \cdots r_n$ und eine beliebige Substitution ϑ mit $\mathsf{dom}(\vartheta) \cap \mathsf{forb} = \emptyset$ gilt: $\vartheta(Ar_1 \cdots r_n) = A\vartheta(r_1) \cdots \vartheta(r_n)$.
3. Analog zu Fall 2.

■

Nun kommen wir zu den weitaus unangenehmeren Paaren mit einem flexiblen und einem starren Term. Seien $fr_1 \cdots r_m$ und $As_1 \cdots s_n$ gegeben mit $f \in \mathsf{flex}$ und $A \in \mathsf{forb} \cup \mathcal{C}$. Ziel ist es natürlich, den flexiblen Kopf f so zu ersetzen, daß ein Term entsteht der zu einem Term mit dem starren Kopf A reduziert. Am einfachsten geht dies natürlich dann, wenn der starre Kopf im Wertebereich der Substitution vorkommen darf, er also eine Konstante ist. Dann können wir folgende „Imitation“ verwenden, eine Substitution der Form $\langle f, \lambda\vec{x_m}.At_1 \cdots t_n\rangle$, wobei die Gestalt der Terme t_i sich nur aus dem Studium der Beweise der später folgenden Lemmatas erschließt.

Prozedur 3.2 Imitation $\mathsf{IMIT}_{\mathsf{UP}}(r, s)$.

Sei $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein Unifikationsproblem und sei* $(r, s) \in \mathrm{U}$ *mit* $r := fr_1 \cdots r_m$, $s := cs_1 \cdots s_n$, $f \in \mathsf{flex}$ *und* $c \in \mathcal{C}$. *Es seien* $w_1, \ldots, w_m$ *paarweise verschiedene, neue Variablen mit* $\tau(w_i) = \tau(r_i)$ *für alle* $1 \leq i \leq m$ *und* $h_1, \ldots, h_n$ *paarweise verschiedene, neue Variablen mit* $\tau(h_i\vec{w}) = \tau(s_i)$ *für alle* $1 \leq i \leq n$. *Dann definieren wir die* Imitationssubstitution für r und s, *kurz* $\mathsf{IMIT}_{\mathsf{UP}}(r, s)$, *wie folgt:*

$$\mathsf{IMIT}_{\mathsf{UP}}(r, s) := \{\langle f, \lambda\vec{w}.c(h_1\vec{w}) \cdots (h_n\vec{w})\rangle\}.$$

Für ein $(r, s) \in \mathrm{U}$ *mit* $r := fr_1 \cdots r_m$, $s := xs_1 \cdots s_n$, $f \in \mathsf{flex}$ *und* $x \in \mathsf{forb}$ *setzen wir* $\mathsf{IMIT}_{\mathsf{UP}}(r, s) := \emptyset$.

Da uns in aller Regel klar ist, welches Unifikationsproblem vorliegt, lassen wir gerne den Index UP weg und schreiben einfach $\mathsf{IMIT}(r, s)$.

Eine andere Möglichkeit betseht darin, daß für eines der Argumente r_j des flexiblen Terms $\mathsf{hd}(r_j) = A$ oder $\mathsf{hd}(r_j) \in \mathsf{flex}$ ist. Dann kann also das „Vorziehen“ dieses Arguments zum Erfolg führen. Wir „projezieren“ also den flexiblen Term auf sein j-tes Argument.

Prozedur 3.3 Projektion auf das j-te Argument $\mathsf{PROJ}^j_{\mathsf{UP}}(r, s)$.

Sei $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein Unifikationsproblem und sei* $(r, s) \in \mathrm{U}$ *mit* $r := fr_1 \cdots r_m$, $s := As_1 \cdots s_n$, $f \in \mathsf{flex}$, $A \in \mathsf{forb} \cup \mathcal{C}$. *Es seien* $w_1, \ldots, w_m$ *paarweise verschiedene, neue Variablen mit* $\tau(w_i) = \tau(r_i)$ *für alle* $1 \leq i \leq m$. *Sei* $j \in \{1 \ldots, m\}$ *so daß* $\tau(r_j) := \alpha_1 \to \cdots \to \alpha_k \to \tau(r)$ *ist. Es seien* $h_1, \ldots, h_k$ *paarweise verschiedene, neue Variablen mit* $\tau(h_i\vec{w}) = \alpha_i$ *für alle* $1 \leq i \leq k$. *Dann definieren wir die* j-te Projektionssubstitution für r und s. *kurz* $\mathsf{PROJ}^j_{\mathsf{UP}}(r, s)$, *wie folgt:*

$$\mathsf{PROJ}^j_{\mathsf{UP}}(r, s) := \{\langle f, \lambda\vec{w}.w_j(h_1\vec{w}) \cdots (h_k\vec{w})\rangle\}.$$

Für ein $j \in \{1, \ldots, m\}$ *mit* $\tau(r_j) \neq \vec{\alpha}_1 \to \cdots \to \alpha_k \to \tau(r)$ *setzen wir* $\mathsf{PROJ}^j_{\mathsf{UP}}(r, s) := \emptyset$

Auch hier lassen wir falls das Unifikationsproblem klar ist den Index UP weg und schreiben kurz $\mathsf{PROJ}^j(r,s)$.

Alle Projektionen und die Imitation ergeben dann die Menge der „Anpassungssubstitutionen“. Dabei ist zu beachten, daß der Algorithmus alle Möglichkeiten verwendet, obwohl natürlich viele Projektionen offensichtlich in die Irre führen, nämlich diejenigen mit $\mathsf{hd}(r_i) \in \mathsf{forb} \cup \mathcal{C}$ und $\mathsf{hd}(r_i) \neq A$.

Prozedur 3.4 Anpassungssubstitution $\mathsf{MATCH}_{\mathsf{UP}}(r,s)$.

Sei $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein Unifikationsproblem und sei* $(r,s) \in \mathrm{U}$ *mit* $r := fr_1 \cdots r_m$, $s := As_1 \cdots s_n$, $f \in \mathsf{flex}$ *und* $A \in \mathsf{forb} \cup \mathcal{C}$. *Dann ist die Menge der* Anpassungssubstitutionen für r und s, *kurz* $\mathsf{MATCH}_{\mathsf{UP}}(r,s)$ *definiert durch:*

$$\mathsf{MATCH}_{\mathsf{UP}}(r,s) := (\{\mathsf{IMIT}(r,s)\} \cup \{\mathsf{PROJ}^1(r,s)\} \cup \cdots \cup \{\mathsf{PROJ}^m(r,s)\}) \setminus \{\emptyset\}.$$

Auch hier verzichten wir auf den Index UP, falls dies zu keiner Konfusion führt und schreiben kurz $\mathsf{MATCH}(r,s)$.

Das folgende Lemma zeigt uns, daß jede Lösung durch entsprechende Anpassungssubstitutionen erzeugt wird, also zu jedem Unifikator ϑ eines Flex-Starr-Paares findet man eine allgemeinere Anpassungssubstitution σ. Dabei gilt sogar für die Substitution ϱ mit $(\varrho \circ \sigma)(f) = \vartheta(f)$, daß sie „kleiner“ ist als ϑ, im Sinne folgender Definition.

Definition. *Die* Grösse *einer Substitution* $\vartheta := \{\langle f_1, r_1\rangle, \ldots, \langle f_n, r_n\rangle\}$ *messen wir wie folgt:*

$$\nu(\vartheta) := n + \sum_{i=1}^{n} |r_i|.$$

Dabei ist $|r_i|$ definiert wie im Beweis von Lemma 3.3. Diesen Abstieg benötigen wir später im Beweis von Lemma 3.7.

Lemma 3.4 *Sei* $\mathsf{UP} := (\{(r,s)\}, \mathsf{forb}, \mathsf{flex})$ *ein Unifikationsproblem mit* $r := fr_1 \cdots r_m$, $s := As_1 \cdots s_n$, $f \in \mathsf{flex}$ *und* $A \in \mathsf{forb} \cup \mathcal{C}$. *Sei* ϑ *ein beliebiger Unifikator von* UP. *Dann existiert eine eindeutige Substitution* $\sigma \in \mathsf{MATCH}(r,s)$ *mit* $\sigma \geq_{\mathsf{flex}} \vartheta$, *d. h. es existiert eine Substitution* ϱ *mit* $(\varrho \circ \sigma)(f) = \vartheta(f)$ *für alle* $f \in \mathsf{flex}$. *Für dieses* ϱ *gilt außerdem* $\nu(\varrho) < \nu(\vartheta)$.

Beweis. Es sei $\vartheta(f) = \lambda\vec{x_m}.Bt_1 \cdots t_l$. Ist $A \in \mathcal{C}$ und $B = A$, so wählen wir $\sigma := \mathsf{IMIT}(r,s)$, also $\sigma(f) := \lambda\vec{x_m}.A(h_1\vec{x}) \cdots (h_n\vec{x})$. Also haben wir zu definieren: $\varrho(h_i) := \lambda\vec{x}.t_i$ für alle $1 \leq i \leq n$.

Ähnlich behandeln wir den Fall $B = x_i$. Dann wählen wir $\sigma := \mathsf{PROJ}^i(r,s)$. ■

Mit Hilfe der obigen Prozeduren MATCH und SIMPL konstruieren wir nun die so genannten *Anpassungsbäume* oder auch MATCH-*Bäume*. An den Knoten dieser Bäume stehen Unifikationsprobleme, stets in der Form $\mathsf{SIMPL}(\mathsf{UP})$. Die Folgeknoten konstruieren wir mit Hilfe der Anpassungssubstitutionen MATCH, wobei wir ein Unifikationspaar aus UP auswählen, welches wir bearbeiten.

Algorithmus 2 Folgeknoten.

Sei $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein Unifikationsproblem. Zur Konstruktion der Folgeknoten von* UP *in einem Anpassungsbaum gehen wir wie folgt vor:*

1. *Ist* $\mathsf{SIMPL}(\mathsf{UP}) = \bot$*, so beenden wir die Konstruktion und ersetzen* UP *durch* $\bot$*.*
2. *Beinhaltet* $\mathsf{SIMPL}(\mathsf{UP})$ *nur noch flexible Terme, so ersetzen wir* UP *durch* $\mathsf{SIMPL}(\mathsf{UP})$ *und beenden die Konstruktion.*
3. *Für jedes Flex-Starr-Paar* (r, s) *in* $\mathsf{SIMPL}(\mathsf{UP})$ *ist die Menge der Folgeknoten bzgl.* (r, s)*:*

$$\{\, \mathsf{SIMPL}(\vartheta(\mathsf{UP})) \mid \vartheta \in \mathsf{MATCH}_{\mathsf{UP}}(r, s) \,\}$$

Einen *Anpassungsbaum* für ein Unifikationsproblem UP erhalten wir nun, indem wir immer und immer wieder Folgeknoten nach obiger Vorschrift konstruieren. An den Kanten notieren wir dabei immer die verwendete Anpassungssubstitution. Dabei können verschiedene Bäume für ein gegebenes Problem UP entstehen, je nachdem welches Flex-Starr-Parr wir ggf. auswählen. Allerdings spielt dies keine Rolle, wie wir im Folgenden sehen werden. Einen Ast in einem Anpassungsbaum können wir also in der Form

$$\mathsf{UP}_0 \to_{\vartheta_1} \mathsf{UP}_1 \to_{\vartheta_2} \cdots$$

notieren mit $\mathsf{UP}_{i+1} = \mathsf{SIMPL}(\vartheta_i(\mathsf{UP}_i))$.

Zunächst müssen wir uns klar machen, daß dieses Verfahren manches Mal nicht terminiert. Starten wir zum Beispiel mit dem Unifikationsproblem

$$\mathsf{UP} := (\{(fc_1, c_2(fc_1))\}, \emptyset, \{f\})$$

$(c_1, c_2 \in \mathcal{C})$, so erhalten wir via der Imitationssubstitution $\{\langle f, \lambda x.c_2(hx)\rangle\}$ als Folgeknoten

$$(\{(hc_1, c_2(hc_1))\}, \emptyset, \{h\})$$

und so weiter und so fort.

Definition. *Einen endlichen Ast* $\mathsf{UP}_0 \to_{\vartheta_1} \mathsf{UP}_1 \to_{\vartheta_2} \cdots \to_{\vartheta_n} \mathsf{UP}_n$ *nennen wir* erfolgreich*, wenn* $\mathsf{UP}_n \neq \bot$ *ist. Ist* $\mathsf{UP}_0 \to_{\vartheta_1} \mathsf{UP}_1 \to_{\vartheta_2} \cdots \to_{\vartheta_n} \mathsf{UP}_n$ *ein erfolgreicher Ast, so nennen wir* $\vartheta := \vartheta_n \circ \cdots \circ \vartheta_1$ *einen* Preunifikator *von* UP*.*

Für so einen erfolgreichen Ast gilt, daß der letzte Knoten UP_n höchstens noch flexible Terme enthalten kann. Dabei haben wir zu beachten, daß der Preunifikator kein Unifikator zu sein braucht. Erst zusammen mit einem Unifikator des verbliebenen Unifikationsproblem bestehend aus flexiblen Termen erhalten wir einen Unifikator des ursprünglichen Problems. Erfreulicherweise existiert für ein Unifikationsproblem, das nur noch aus flexiblen Termen besteht stets ein Unifikator, wie folgendes Lemma zeigt.

Lemma 3.5 *Sei* $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein Unifikationsproblem in dem alle Terme in* U *flexibel sind. Dann existiert ein Unifikator von* UP*.*

Beweis. Wir konstruieren den Unifikator ϑ von UP wie folgt: Wir reservieren wir für jeden Grundtyp ι eine neue Variable h^ι. Für jede Variable $f \in \mathsf{flex}$ vom Typ $\alpha_1 \to \cdots \to \alpha_n \to \iota$ setzen wir $\vartheta(f) := \lambda x_1, \ldots, x_n.h^\iota$ wobei $\tau(x_i) = \alpha_i$ sein muß. Durch einfaches Nachrechnen erkennen wir schnell, daß ϑ dann ein Unifikator von UP ist. ■

Der im Beweis angegebene Unifikator ist allerdings ein ziemlich rauher Geselle, er ist in nahezu keinem Fall ein allgemeinster Unifikator. Man könnte ihn vielleicht als „brutalsten“ Unifikator charakterisieren. Die Frage wie und ob man für rein flexible Unifikationsprobleme einen allgemeinsten Unifikator erzeugen kann bzw. wenigstens eine vollständige Menge von Unifikatoren, ist Inhalt des folgenden Kapitels über relevanteste Unifikatoren.

Nun zeigen wir aber erst mal, daß unsere Preunifikatoren genau das leisten, was sie versprechen, nämlich uns einer Lösung näher bringen.

Lemma 3.6 *Sei* UP_0 *ein Unifikationsproblem und sei* $\mathsf{UP}_0 \to_{\vartheta_1} \cdots \to_{\vartheta_n} \mathsf{UP}_n$ *ein erfolgreicher Ast in dem Anpassungsbaum für* UP_0*. Sei* σ *ein Unifikator von* UP_n *und sei* $\vartheta := \vartheta_n \circ \cdots \circ \vartheta_1$*. Dann ist* $\sigma \circ \vartheta$ *ein Unifikator von* UP_0*.*

Beweis. Die Existenz von σ folgt aus Lemma 3.5. Mit $\varrho_{n+1} := \sigma$ und $\vartheta_{n+1} := \varepsilon$ definieren wir $\varrho_i := \varrho_{i+1} \circ \vartheta_{i+1}$ für $1 \leq i \leq n$. Dann zeigen wir durch absteigende Induktion über i, daß ϱ_i ein Unifikator von UP_i ist. Der Induktionsanfang gilt nach Voraussetzung. Der Induktionsschluß folgt unmittelbar aus der Induktionsvoraussetzung und Lemma 3.3. ■

Nach der Korrektheit folgt stets die Vollständigkeit. Hier also die Aussage, daß jeder Unifikator einen Preunifikator „enthält“.

Lemma 3.7 *Sei* $\mathsf{UP} := (\{(r, s)\}, \mathsf{forb}, \mathsf{flex})$ *ein lösbares Unifikationsproblem. Sei* θ *ein beliebiger Unifikator von* UP*. Dann existiert in jedem Anpassungsbaum für* UP *ein erfolgreicher Ast, so daß für den zugehörige Preunifikator* ϑ *gilt:* $\vartheta \geq_{\mathsf{flex}} \theta$*.*

Beweis. Sei uns also θ und ein Anpassungsbaum für $\mathsf{UP}_0 := \mathsf{UP}$ gegeben. Wir konstruieren einen Ast $\mathsf{UP}_0 \to_{\vartheta_1} \cdots \to_{\vartheta_n} \mathsf{UP}_i$ und eine Folge von Substitutionen $\sigma_0, \ldots, \sigma_i$, so daß für alle $i \geq 0$ gilt:

1. σ_i ist Unifikator von UP_i oder UP_i enthält nur noch flexible Terme,
2. $\nu(\sigma_{i+1}) < \nu(\sigma_i)$,
3. $\theta(f) = (\sigma_i \circ \vartheta_i \circ \cdots \circ \vartheta_1)(f)$ für alle $f \in \mathsf{flex}$.

Dies alles konstruieren wir mittels Induktion über $i \geq 0$ und zeigen, daß dieser Ast in dem gegebenen Anpassungsbaum liegt. Für den Induktionsanfang wählen wir $\sigma_0 := \theta$.

Im Induktionsschluß $i \to i+1$ ist nach Voraussetzung σ_i ein Unifikator von UP_i. Enthält UP_i nur noch flexible Terme, so beenden wir die Konstruktion und

erhalten alle gewünschten Eigenschaften aus der Induktionshypothese. Sonst sei $\mathsf{UP}_i := (\mathrm{U}_i, \mathsf{forb}_i, \mathsf{flex}_i)$ und $(r_i, s_i) \in \mathrm{U}_i$ mit s_i starr. Nach Lemma 3.4 existiert eine Substitution $\vartheta \in \mathsf{MATCH}_{\mathsf{UP}_i}(r_i, s_i)$ und eine Substitution σ mit $\sigma_i(f) = (\sigma \circ \vartheta)(f)$ für alle $f \in \mathsf{flex}_i$ und $\nu(\sigma) < \nu(\sigma_i)$. Also existiert in dem gegebenen Anpassungsbaum ein Folgeknoten UP' von UP_i mit $\mathsf{UP}_i \rightarrow_\vartheta \mathsf{UP}'$. Wir wählen also $\vartheta_{i+1} := \vartheta$, $\sigma_{i+1} := \sigma$ und $\mathsf{UP}_{i+1} := \mathsf{UP}'$. Nun können wir durch einfaches Nachrechnen die gewünschten Eigenschaften nachweisen. ■

Nun fassen wir nochmal alle Eigenschaften dieses Algorithmus übersichtlich zusammen.

Theorem 3.2 *Sei* $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein Unifikationsproblem. Terminiert die Konstruktion eines Anpassungsbaumes für* UP*, so gilt:*

1. *Existiert kein erfolgreicher Ast im Anpassungsbaum, so ist* UP *nicht lösbar.*
2. *Ist* UP *lösbar und* θ *ein beliebiger Unifikator von* UP*, so existiert in jedem Anpassungabaum für* UP *ein erfolgreicher Ast mit dem dazu gehörigen Preunifikator* ϑ *so daß* $\vartheta \geq_{\mathsf{flex}} \theta$ *gilt.*

3.4 Unifikation von Mustern nach Dale Miller

Muster sind spezielle, stark eingeschränkte Terme, für die die Unifikation entscheidbar ist. Für unifizierbare Terme existieren sogar allgemeinste Unifikatoren.

Entstanden sind diese Terme aus dem Problem (siehe [24][25]) der Unifikation unter gemischten Quantoren. Ist z. B. eine Formel der Gestalt $\forall x \exists y \forall z.A(x, y, z)$ gegeben und soll später mittels Unifikation ein Beweis geführt werden, so ist bei der Instanziierung der Variable y darauf zu achten, daß der für y eingesetzte Term nicht von z abhängt. Wohl aber darf er von x abhängen. Die Lösung ist das Sortieren der Quantoren derart, daß man obige Formel durch $\exists f \forall x \forall z.A(x, fx, z)$ ersetzt. Dabei kommt eine höherstufige Variable f ins Spiel. Nun kommt f in die Menge der flexiblen Variable und x, z in die Menge der verbotenen Variablen. Instanziiert man nun also f mit einem Term $\lambda v.r$, so entspricht dies der Ersetzung von y durch $s = r_v[x]$. Somit ist also das freie Auftreten von x in s möglich, nicht aber das freie Auftreten von z, sofern man sich an die Restriktionen der Unifikatoren bzgl. forb hält. Damit hat man eine interen Repräsentation dieser Quantorenreihenfolge erreicht und muß nicht z. B. über externe Markierungen gehen. [15]

Insbesondere kommen so auch höherstufige Variablen ins Spiel, man benötigt also höherstufige Unifikation, selbst wenn man von Termen erster Stufe ausgegangen ist. Zum Glück sind die Terme in denen höherstufige Variablen auftreten, von sehr einfacher Bauart.

[15] Dabei werden die Allquantifizierten Variablen durch Konstanten ersetzt und diese zusammen mit den freien Variablen (also den Existenzquantifizierten) mit numerischen Marken versehen. In unsrem Fall hätte man die Konstanten c_x und c_z und die Marken 1 für c_x und y sowie 2 für c_z. Erlaubt man nur Instanziierungen von y mit Termen, in denen keine Konstanten mit größerer Markierung als 1 vorkommen, erhält man ebenfalls eine Quantoren respektierende Unifikation. Siehe dazu z. B. [27] oder [21].

Definition. *Ein Term r heißt* Muster *bzgl.* forb *und* flex*, falls für jeden Teilterm der Form $fx_1 \cdots x_n$ mit $f \in \mathsf{flex} \cap \mathsf{FV}(r)$ gilt:*

1. *Für alle $1 \leq i \leq n$ ist $x_i \in \mathsf{forb} \cup \mathsf{BV}(r)$.*
2. *Für alle $1 \leq i \neq j \leq n$ gilt $x_i \neq x_j$.*

Es folgen ein paar einfache Beispiele.

Beispiel. Seien forb und flex gegeben und $f \in \mathsf{flex}$. Dann ist $\lambda x, y.fxy$ ein Muster, aber nicht $\lambda x.fxx$. Der Term fxy ist ein Muster genau dann, wenn $x, y \in \mathsf{forb}$. Ist c eine Konstante, so ist $\lambda x.c(fx)(fx)$ ein Muster. Der Term $c(fx)(\lambda y.fy)$ ist nur genau dann ein Muster, wenn $x \in \mathsf{forb}$ ist.

Nun wollen wir Schritt für Schritt den Unifikationsalgorithmus beschreiben. Zunächst stellen wir fest: Ist $f\vec{x}$ ein flexibles Muster und r ein starrer Term mit $f \in \mathsf{FV}(r)$, so sind beide nicht unifizierbar. Jeder Versuch endet wie die Geschichte vom Hasen und dem Igel. Deshalb benötigen wir einen so genannten „Vorkommenstest“ oder „Occurs-Check“.

Prozedur 3.5 Occurs-Check $\mathsf{oc}(f, r)$.

$$\mathsf{oc}(f, r) := \begin{cases} \mathsf{wahr} & f \in \mathsf{FV}(r) \\ \mathsf{falsch} & f \notin \mathsf{FV}(r) \end{cases}$$

Ist $f\vec{x}$ ein flexibles Muster und r ein starrer Term mit $f \notin \mathsf{FV}(r)$ und wollen wir beide unfizieren, so bleibt uns nur die Möglichkeit für f den Term $\lambda\vec{x}.r$ einzusetzen. Dies geht aber nur, solange $\lambda\vec{x}.r$ keine verbotenen Variablen frei enthält. Existiert also eine Variable $y \in \mathsf{FV}(r) \cap \mathsf{forb}$, welche nicht in $\vec{x}$ enthalten ist, so müssen wir diese erst aus r *herausschneiden*, falls dies möglich ist. Dazu dient die Prozedur *Pruning*. Es ist zu beachten, daß dieses Herausschneiden nur funktionieren kann, wenn y als Argument einer flexiblen Variable auftritt.

Prozedur 3.6 Pruning $\mathsf{Prune}(x, r, \mathsf{forb}, \mathsf{flex})$.

Gelte $x \in \mathsf{FV}(r)$. Sei

$$M = \{\, s \text{ Teilterm von } r \mid x \in \mathsf{FV}(s) \text{ und } \mathsf{hd}(s) \in \mathsf{flex} \,\}.$$

Ist $M = \emptyset$, so sei $\mathsf{Prune}(x, r, \mathsf{forb}, \mathsf{flex}) := \dagger$. Sonst sei $s = fy_1 \cdots y_k x y_{k+1} \cdots y_l \in M$. Sei g eine neue Variable mit $\tau(g\vec{y_l}) = \tau(s)$. Dann ist

$$\mathsf{Prune}(x, r, \mathsf{forb}, \mathsf{flex}) := \{\langle f, \lambda y_1, \ldots, y_k, x, y_{k+1}, \ldots, y_l.g\vec{y_l}\rangle\}.$$

Haben wir nun für $f\vec{x}$ flexibel und r starr festgestellt, daß weder $f \in \mathsf{FV}(r)$ noch verbotene Variablen frei in $\lambda\vec{x}.r$ vorkommen, so unifizieren wir also beide Terme mittels der Substitution $\mathsf{fs}(r, s, \mathsf{forb}, \mathsf{flex})$.

Prozedur 3.7 Flex-Starr $\mathsf{fs}(r, s, \mathsf{forb}, \mathsf{flex})$.

Sei $r = fx_1 \cdots x_n$ *mit* $f \in \mathsf{flex}$, $\mathsf{hd}(r) \in \mathsf{forb} \cup \mathcal{C}$ *und gelte* $\mathsf{forb} \cap \mathsf{FV}(s) \subseteq \{x_1, \ldots, x_n\}$. *Dann ist*

$$\mathsf{fs}(r, s, \mathsf{forb}, \mathsf{flex}) := \{\langle f, \lambda \vec{x_n}.s\rangle\}.$$

Zwei flexible Terme $f\vec{x}$ und $f\vec{y}$ mit gleichen Kopf unifizieren wir, indem wir möglichst viele Argumente behalten, also alle x_i, y_i mit $x_i = y_i$.

Prozedur 3.8 Flex=Flex $\mathsf{ff}(r, s, \mathsf{forb}, \mathsf{flex})$.

Seien $r = fx_1 \cdots x_n$ *und* $s = fy_1 \cdots y_n$ *mit* $f \in \mathsf{flex}$ *gegeben. Sei* $\{i_1, \ldots, i_k\} = \{\, i \in \{1, \ldots, n\} \mid x_i = y_i \,\}$. *Sei* g *eine neue Variable mit* $\tau(gx_{i_1} \cdots x_{i_k}) = \tau(f\vec{x_n})$. *Dann ist*

$$\mathsf{ff}(r, s, \mathsf{forb}, \mathsf{flex}) := \{\langle f, \lambda \vec{x_n}.gx_{i_1} \cdots x_{i_k}\rangle\}.$$

Sind $f\vec{x}$ und $g\vec{y}$ flexible Muster mit $f \neq g$, so können wir alle Argumente aus dem Durchschnitt von $\{\vec{x}\}$ und $\{\vec{y}\}$ behalten.

Prozedur 3.9 Flex $\neq$ Flex $\mathsf{fg}(r, s, \mathsf{forb}, \mathsf{flex})$.

Sei $r = fx_1 \cdots x_m$ *und* $s = gy_1 \cdots y_n$ *mit* $f, g \in \mathsf{flex}$. *Sei* $\{z_1, \ldots, z_k\} = \{x_1, \ldots, x_m\} \cap \{y_1, \ldots, y_n\}$. *Sei* h *eine neue Vraiable mit* $\tau(h\vec{z_k}) = \tau(f\vec{x_m})$. *Dann ist*

$$\mathsf{fg}(r, s, \mathsf{forb}, \mathsf{flex}) := \{\langle f, \lambda \vec{x_m}.h\vec{z_k}\rangle, \langle g, \lambda \vec{y_n}.h\vec{z_k}\rangle\}.$$

Der Algorithmus nach Dale Miller zur Unifikation höhertstufiger Muster besteht nun darin, die obigen Prozeduren in einer sinnvollen Reihenfolge aufzurufen. Außerdem brauchen wir noch die Prozedur SIMPL (Prozedur 3.1) aus dem vorherigen Abschnitt 3.3.

Algorithmus 3 *Sei* $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein Muster-Unifikationsproblem. Zur Unifikation setze* $\theta := \varepsilon$, $V := \mathsf{flex}$ *und führe folgende Schritte aus.*

1. *Ersetze* UP *durch* $(\mathrm{U} \setminus \{\, (r, s) \in \mathrm{U} \mid r = s \,\}, \mathsf{forb}, \mathsf{flex})$ *und gehe zu 2.*
2. *Ersetze* UP *durch* $\mathsf{SIMPL}(\mathsf{UP})$ *(siehe Prozedur 3.1).*
3. *Ist* $\mathsf{UP} = \bot$, *so gib* $\bot$ *aus. Ist* $\mathrm{U} = \emptyset$, *so gib* $\theta \mid_V$ *aus.*
4. *Sonst wähle ein Paar* $(r, s) \in \mathrm{U}$ *und unterscheide folgende Fälle:*
   (a) *Ist* $r = fx_1, \cdots x_n$ *flexibel und* s *starr, so unterscheide folgende Fälle:*
      (i) *Ist* $\mathsf{oc}(f, r) = \mathsf{wahr}$, *so ersetze* UP *durch* $\bot$ *und gehe zu 3.*
      (ii) *Ist* $\mathsf{oc}(f, r) = \mathsf{falsch}$ *und existiert ein* $y \in \mathsf{FV}(r) \cap \mathsf{forb} \setminus \{\vec{x_n}\}$, *so rufe* $\mathsf{Prune}(y, r, \mathsf{forb}, \mathsf{flex})$ *auf. Ist* $\mathsf{Prune}(y, r, \mathsf{forb}, \mathsf{flex}) = \dagger$, *so erzetze* UP *durch* $\bot$ *und gehe zu 3. Ist* $\mathsf{Prune}(y, r, \mathsf{forb}, \mathsf{flex}) = \vartheta$, *so erstze* UP *durch* $\vartheta(\mathsf{UP})$, θ *durch* $\vartheta \circ \theta$ *und gehe zu 1.*

(iii) *Ist* $\mathsf{oc}(f,r) = \mathsf{falsch}$ *und* $\mathsf{FV}(r) \cap \mathsf{forb} \setminus \{x_1,\ldots,x_n\} = \emptyset$*, so sei* $\vartheta := \mathsf{fs}(r,s,\mathsf{forb},\mathsf{flex})$*. Ersetze nun* UP *durch* $\vartheta(\mathsf{UP})$*,* θ *durch* $\vartheta\circ\theta$ *und gehe zu 1.*

(b) *Sind* r *und* s *flexibel, so unterscheide folgende Fälle:*

(i) *Ist* $\mathsf{hd}(r) = \mathsf{hd}(s)$*, so sei* $\vartheta := \mathsf{ff}(r,s,\mathsf{forb},\mathsf{flex})$*. Ersetze* UP *durch* $\vartheta(\mathsf{UP})$*,* θ *durch* $\vartheta\circ\theta$ *und gehe zu 1.*

(ii) *Ist* $\mathsf{hd}(r) \neq \mathsf{hd}(s)$*, so sei* $\vartheta := \mathsf{fg}(r,s,\mathsf{forb},\mathsf{flex})$*. Ersetze* UP *durch* $\vartheta(\mathsf{UP})$*,* θ *durch* $\vartheta\circ\theta$ *und gehe zu 1.*

Der fachkundige Leser mag hier den im Original so genannten ξ-Schritt vermissen. Dieser ist bereits in die Bereitstellung aus der Prozedur SIMPL eingebaut. Dies alles erscheint auf den ersten Blick recht kompliziert, deshalb betrachten wir erst mal ein Beispiel.

Beispiel. Sei $\mathsf{UP} = (\{(c(\lambda x.c(fxy)), cg)\}, \{y\}, \{f,g\})$ mit $c \in \mathcal{C}$. Es sei ι ein Grundtyp, α ein beliebiger Typ und $\tau(x) = \tau(y) = \alpha$, $\tau(g) = \alpha \to \iota$, $\tau(c) = (\alpha \to \iota) \to \iota$. Wir rufen nun $\mathsf{SIMPL}(\mathsf{UP})$ auf den Plan. Zunächst werden die starren Köpfe entfernt. Wir erhalten

$$(\{(\lambda x.c(fxy), g)\}, \{y\}, \{f,g\}).$$

Nun ist der rechte Term g noch kein Anwendungsterm von einem Grundtyp. Also expandieren wir diesen zu $\lambda v.gv$ mit neuer Variable v^α und benennen x und v jeweils in die neue Variable w^α um. Anschließend streichen wir die λ-Prefixe und stecken w in die Menge forb. Unser Problem sieht nun wie folgt aus:

$$(\{(c(fwy), gw)\}, \{y,w\}, \{f,g\}).$$

Nun sortieren wir noch und erhalten als endgültiges Resultat

$$\mathsf{UP}_1 := \mathsf{SIMPL}(\mathsf{UP}) = (\{(gw, c(fwy))\}, \{y,w\}, \{f,g\}).$$

Wir müssen nun also den flexiblen Term gw mit dem starren Term $c(fwy)$ unifizieren. Aber Halt. Erst prüfen wir ob $\mathsf{forb} \cap \mathsf{FV}(c(fwy)) \subseteq \{w\}$. Offenbar nicht, da die verbotene Variable y ebenfalls vorkommt. Diese müssen wir also herausschneiden. Dazu rufen wir $\mathsf{Prune}(y, c(fwy), \{w,y\}, \{f,g\})$ auf. Wir erhalten als Menge flexiblen Teilterme, die y enthalten: $M = \{fwy\}$. Also ist

$$\mathsf{Prune}(y, c(fwy), \{w,y\}, \{f,g\}) = \{\langle f, \lambda w, y.hw\rangle\},$$

wobei h den Typ $\alpha \to \alpha \to \iota$ haben soll. Wenden wir diese Substitution auf UP_1 an, so erhalten wir

$$\mathsf{UP}_2 = (\{(gw, c(hw))\}, \{y,w\}, \{g,h\}).$$

Jetzt können wir die Prozedur $\mathsf{fs}(gw, c(hw), \{y,w\}, \{g,h\})$ aufrufen und erhalten als Antwort $\{\langle g, \lambda w.c(hw)\rangle\}$. Angewandt auf UP_2 ergibt dies

$$\mathsf{UP}_3 = (\{(c(hw), c(hw))\}, \{y,w\}, \{h\}).$$

und damit nach Durchlaufen von Schritt 1 des Algorithmus:

$$\mathsf{UP}_4 = (\emptyset, \{y,w\}, \{h\}).$$

Also geben wir jetzt

$$\theta = \{\langle g, \lambda w.c(hw)\rangle\} \circ \{\langle f, \lambda w, y.hw\rangle\} = \{\langle g, \lambda w.c(hw)\rangle, \langle f, \lambda w, y.hw\rangle\}$$

als allgemeinsten Unifikator von UP aus.

Nun aber zu den allgemeinen Eigenschaften des Algorithmus, als da wären Terminierung, Korrektheit und Vollständigkeit.

Lemma 3.8 *Der obige Algorithmus 3 terminiert.*

Beweis. Dazu definieren wir ein passendes Maß auf $\mathsf{UP} = (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$. Für einen β-normalen Term $\lambda\vec{x_m}.A\vec{r_n}$ definieren wir

$$\|\lambda\vec{x_m}.A\vec{r_n}\| := \begin{cases} m & A \in \mathsf{flex} \\ m + n + \sum_{i=1}^{n} \|r_i\| & A \in \mathsf{forb} \cup \mathcal{C} \end{cases}$$

Desweiteren sei k die Anzahl der Paare in U, l die Größe von flex, m die Anzahl der freien Vorkommen von flexiblen Variablen in U und p die Anzahl der Vorkommen verbotener Variablen als Argumente flexibler Variablen. Dann definieren wir das Maß $\|\mathsf{UP}\| := (l, m, p, \sum_{(r,s)\in \mathrm{U}} \|r\| + \|s\|, k)$. Nun rechnen wir einfach nach, daß bei jedem Durchlaufen von Schritt 2 im Algorithmus 3 das Unifikationsproblem kleiner geworden ist bzgl. der lexikographischen Ordnung auf $\mathbb{N}^5$. ■

Wir können also die Anwendung von Algorithmus 3 auf ein Musterunifikationsproblem beschreiben durch die Abfolge einzelner Schritte

$$\mathsf{UP}_0 \Rightarrow_{\vartheta_1} \mathsf{UP}_1 \Rightarrow_{\vartheta_2} \cdots \Rightarrow_{\vartheta_n} \mathsf{UP}_n$$

mit $\mathsf{UP}_n = \bot$ oder $\mathsf{UP}_n = (\mathrm{U}_n, \mathsf{forb}_n, \mathsf{flex}_n)$ mit $\mathrm{U}_n = \emptyset$. Sei für alle $1 \leq i \leq n-1$ $(\mathrm{U}_i, \mathsf{forb}_i, \mathsf{flex}_i) = \mathsf{SIMPL}(\vartheta_i(\mathrm{U}_{i-1}))$. Dann ist $\mathsf{UP}_i = (\mathrm{U}_i \setminus \{\,(r,s) \in \mathrm{U}_i \mid r = s\,\}, \mathsf{forb}_i, \mathsf{flex}_i)$. Ist $\mathsf{UP}_n = \emptyset$, so gilt dies auch für $i = n$.

Lemma 3.9 *Ist* $\mathsf{UP} \Longrightarrow_\vartheta \mathsf{UP}'$ *ein Arbeitsschritt des Algorithmus 3 und* θ *ein Unifikator von* UP'*, so ist* $\theta \circ \vartheta$ *ein Unifikator von* UP*.*

Beweis. Sei $\mathsf{UP} = (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ und $(r,s) \in \mathrm{U}$. Offenbar ist $\mathsf{UP}' \neq \bot$. Da θ ein Unifikator von $\mathsf{SIMPL}(\vartheta(\mathsf{UP}))$ ist , ist θ auch ein Unifikator von $\vartheta(\mathsf{UP})$ (siehe Lemma 3.3). Also folgt für alle $(r,s) \in \mathrm{U}$: $(\theta \circ \vartheta)(r) = (\theta \circ \vartheta)(s)$. ■

Lemma 3.10 *Sei* $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein Muster-Unifikationsproblem und sei* $\mathsf{UP} \Longrightarrow_\vartheta \mathsf{UP}' \neq \bot$ *ein Arbeitsschritt des Algorithmus 3. Sei* θ *ein Unifikator von* UP*. Dann existiert ein Unifikator* σ *von* UP' *mit* $\theta(f) = (\sigma \circ \vartheta)(f)$ *für alle* $f \in \mathsf{flex}$*.*

Beweis. Betrachten wir das Paar $(r,s) \in \mathrm{U}$, welches zur Konstruktion von ϑ führte. Es gilt $\theta(r) = \theta(s)$. Unterscheiden wir also die einzelnen Fälle:

(a) Sei $r = fx_1 \cdots x_n$ flexibel und s starr. Aus $\theta(f\vec{x}) = \theta(s)$ folgt $\theta(f) = \lambda x_1, \ldots, x_n.\theta(s)$. Also muß wegen $\mathsf{FV}(\theta) \cap \mathsf{forb} = \emptyset$ gelten $\mathsf{FV}(\theta(s)) \cap \mathsf{forb} \subseteq \{x_1, \ldots, x_n\}$. Außerdem $f \notin \mathsf{FV}(s)$.

Pruning. Also $r = fx_1 \cdots x_n$ und s ein starrer Term sowie $f \notin \mathsf{FV}(s)$ und $y \in \mathsf{FV}(s) \cap \mathsf{forb} \setminus \{x_1, \ldots, x_n\}$. Ist y kein Argument einer flexiblen Variable, so ist $\mathsf{Prune}(y, s, \mathsf{forb}, \mathsf{flex}) = \dagger$. Allerdings ist dann $y \in \mathsf{FV}(\varrho(s))$ für jede Substitution ϱ mit $\mathsf{dom}(\varrho) \subseteq \mathsf{flex}$. Folglich kann kein Unifikator von UP existieren.

Also muß jedes Vorkommen von y die Form $g\vec{x}y\vec{z}$ haben mit $g \in \mathsf{flex}$. Es ist $\mathsf{Prune}(y, s, \mathsf{forb}, \mathsf{flex}) = \{\langle g, \lambda\vec{x}, y, \vec{z}.h\vec{x}\vec{z}\rangle\}$. Sei $\theta(g) = \lambda\vec{x}, y, \vec{z}.t$. Dann ist $y \notin \mathsf{FV}(t)$. Also definieren wir $\sigma(h) := \lambda\vec{x}, \vec{z}.t$ und $\sigma = \theta$ sonst. Dann ist $\sigma \circ \vartheta = \theta$ und σ ist ein Unifikator von UP'.

Gilt $\mathsf{FV}(s) \cap \mathsf{forb} \subseteq \{x_1, \ldots, x_n\}$, so ist $\mathsf{fs}(r, s, \mathsf{forb}, \mathsf{flex}) = \{\langle f, \lambda\vec{x}.s\rangle\}$. Nach den einführenden Worten zu diesem Fall ist $\theta(f) = \lambda\vec{x}.\theta(s)$. Also setzten wir $\sigma(g) = \theta(g)$ für alle $g \in \mathsf{dom}(\theta) \setminus \{f\}$. Dann ist offensichtlich σ ein Unifikator von UP' und $\sigma \circ \vartheta = \theta$.

(b) Die anderen Fälle können ganz analog, nur sehr viel einfacher nachgerechnet werden.

■

Theorem 3.3 *Sei* UP *ein Muster-Unifikationsproblem. Wenden wir Algorithmus 3 auf* UP *an, so gilt:*

1. *Die Anwendung des Algorithmus terminiert.*
2. *Erhalten wir die Antwort* $\bot$*, so ist* UP *nicht unifizierbar.*
3. *Erhalten wir als Antwort die Substitution* ϑ*, so ist dies ein allgemeinster Unifikator von* UP*.*

Beweis. Die Terminiereung war Inhalt von Lemma 3.8. Der zweite Teil folgt aus den Eigenschaften der Prozedur SIMPL, siehe Lemma 3.3, und dem Fall über den Schritt „pruning“ im Lemma 3.10.

Zum dritten Teil betrachten wir also die Anwendung des Algorithmus

$$\mathsf{UP} \Rightarrow_{\vartheta_1} \mathsf{UP}_1 \Rightarrow_{\vartheta_2} \cdots \Rightarrow_{\vartheta_n} \mathsf{UP}_n$$

mit $\mathsf{UP}_n = (\emptyset, \mathsf{forb}_n, \mathsf{flex}_n)$. Es ist $\vartheta = \vartheta_n \circ \cdots \circ \vartheta_1$. ϑ ist ein Unifikator, denn ε ist ein Unifikator von UP_n, also nach Lemma 3.9 ist ϑ_n ein Unifikator von UP_{n-1}, also ist $\vartheta_{n-1} \circ \vartheta_n$ ein Unifikator von UP_{n-2} usw.

Sei umgekehrt θ ein beliebiger Unifikator von UP. Dann existiert nach Lemma 3.10 ein Unifikator σ_1 von UP_1 mit $\sigma_1 \circ \vartheta_1 = \theta$. Treiben wir dieses Spiel weiter bin UP_n, so erhalten wir einen Unifikator σ_n von UP_n (jede Substitution deren Definitionsbereich in flex_n liegt ist Unifikator von UP_n) mit $\sigma_n \circ \vartheta_n = \sigma_{n-1}$. Es ist aber $\sigma_{n-1} \circ \vartheta_{n-1} = \sigma_{n-2}$ usw. Also $\sigma_n \circ \vartheta = \theta$, folglich $\vartheta \geq_{\mathsf{flex}} \theta$. ■

Also haben wir für die recht einfachen höherstufigen Muster das selbe Verhalten bzgl. der Unifikation wie für erststufige Terme nachgewiesen, nämlich Entscheidbarkeit und die Existenz allgemeinster Unifikatoren. In den nun folgenden Kapiteln wollen wir uns der Frage zuwenden, ob Muster bzgl. dieser Eigenschaften schon das Ende der Fahnenstange darstellen. Dazu wird der Begriff der Muster auf zwei Arten ausgedehnt. Einmal indem man komplexere Argumente für flexible Variablen zuläßt deren Köpfe paarweise verschiedene verbotene Variablen sind. Dies führt zum Begriff der relevantesten Unifikatoren, siehe das folgende Kapitel. Zum Zweiten untersuchen wir, ob man nicht auch Konstanten als Argumente zulassen kann. Dazu benötigen wir dann ein Kalkül mit Fallunterscheidung, siehe letztes Kapitel.

4 Relevanteste Unifikatoren

Oft besitzen Unifikationsprobleme keinen allgemeinsten Unifikator, aber eine endliche, vollständige Menge von Unifikatoren. Die Elemente dieser Menge kann man u. U. doch wieder durch einen einzigen Unifikator ausdrücken. Besonders dann, wenn sie sich nur durch Bindungen unterscheiden, die für den jeweils anderen Unifikator keine Rolle spielen. Dazu ein Beispiel.

Beispiel. Betrachten wir das Unifikationsproblem

$$\mathsf{UP} := (\{(f_1 y(xg), f_2 z(xc))\}, \{x, y, z\}, \{f_1, f_2, g\}).$$

Setzen wir $\tau(g) = \tau(c)$ voraus, so bilden die folgenden Substitutionen ϑ_1 und ϑ_2 eine vollständige Menge von Unifikatoren:

$$\begin{aligned} \vartheta_1 &:= \{\langle f_1, \lambda y, v.hv\rangle, \langle f_2, \lambda z, v.hv\rangle, \langle g, c\rangle\}, \\ \vartheta_2 &:= \{\langle f_1, \lambda y, v.h'\rangle, \langle f_2, \lambda z, v.h'\rangle\}, \end{aligned}$$

wobei h, h' neue Variablen seien. Es gilt weder $\vartheta_1 \geq \vartheta_2$ noch $\vartheta_2 \geq \vartheta_1$. Aber mit der Substitution

$$\sigma := \{\langle h, \lambda v.h'\rangle\}$$

erhalten wir $(\sigma \circ \vartheta_1)(f_i) = \vartheta_2(f_i)$ für $i = 1, 2$, aber $(\sigma \circ \vartheta_1)(g) = c \neq g = \vartheta_2(g)$. Es ist aber $(\sigma \circ \vartheta)(f_1 y(xg)) = h'$, d. h. die Ersetzung von g durch c „verschwindet“. Die Bindung $\langle g, c\rangle$ ist also nicht mehr relevant für diese Lösung, wie sie auch nicht relevant ist für ϑ_2.

Einen Unifikator wie ϑ_1 in obigen Beispiel wollen wir besonders auszeichnen und *relevantesten* Unifikator von UP nennen. Zunächst definieren wir, was eine relevante Bindung und der relevante Anteil einer Substitution sind.

Definition. *Sei r ein Term und $\vartheta := \{\langle f_1, r_1\rangle, \ldots, \langle f_n, r_n\rangle\}$ eine Substitution. Eine Bindung $\langle f_i, r_i\rangle \in \vartheta$ heißt* relevant *bzgl. r, falls $f_i \in \mathsf{flex} \cap \mathsf{FV}(\vartheta \setminus \{\langle f_i, r_i\rangle\}(r))$ ist. Der* relevante Anteil ϑ^{R}_r *von ϑ bzgl. r ist definiert durch*

$$\vartheta^{\mathsf{R}}_r := \{\, \langle f, r\rangle \in \vartheta \mid \langle f, r\rangle \textit{relevant bzgl. } r \,\}.$$

Für ein Unifikationsproblem $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ heißt eine Bindung $\langle f_i, r_i\rangle \in \vartheta$ relevant *bzgl. UP, falls es ein $r \in \mathrm{U}$ gibt so daß $\langle f_i, r_i\rangle$ relevant ist bzgl. r. Der* relevante Anteil $\vartheta^{\mathsf{R}}_{\mathsf{UP}}$ *von ϑ bzgl. UP ist definiert durch*

$$\vartheta^{\mathsf{R}}_{\mathsf{UP}} := \{\, \langle f, r\rangle \in \vartheta \mid \langle f, r\rangle \textit{relevant bzgl. } \mathsf{UP} \,\}.$$

Wir bemerken, daß für einen Unifikator ϑ eines Unifikationsproblems UP auch $\vartheta^{\mathsf{R}}_{\mathsf{UP}}$ ein Unifikator von UP ist. Nun wollen wir zur Definition des relevantesten Unifikators schreiten.

Definition. *Ein Unifikator ϑ eines Unifikationsproblems $\mathsf{UP} = (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$*

heißt relevantester *Unifikator (kurz* mru[16]*), falls zu jedem beliebigen Unifikator* ϱ *von* UP *eine Substitution* σ *existiert mit* $\mathsf{dom}(\sigma) \subseteq \mathsf{FV}(\vartheta)$ *und*

$$(\sigma \circ \vartheta)^{\mathsf{R}}_{\mathsf{UP}} \geq_{\mathsf{flex}} \varrho.$$

Zu einem beliebigen Unifikator θ bekommt man also einen allgemeineren Unifikator, indem man dem relevantesten Unifikator ϑ eine Substitution „vorschaltet“ und dann die irrelevant gewordenen Bindungen streicht. Folglich bildet die Menge

$$\{\, (\sigma \circ \vartheta)^{\mathsf{R}}_{\mathsf{UP}} \mid \mathsf{dom}(\sigma) \subseteq \mathsf{FV}(\vartheta) \,\}$$

eine vollständige Menge von Unifikatoren. Interessant sind nun natürlich die Fälle, in denen endlich viele Substitutionen σ ausreichen. In den folgenden Abschnitten wollen wir ein solches Problem betrachten, also Terme, für die wir einen relevantesten Unifikator berechnen können und für die man dann eine endliche, vollständige Menge von Unifikatoren erzeugen kann.

4.1 Ein Beispiel für die Existenz relevantester Unifikatoren

Wir wollen also in diesem Abschnitt eine Familie von Termen definieren, für die ein relevantester Unifikator existiert und einen Algorithmus zu dessen Berechnung angeben.

Zunächst müssen wir uns klar machen, daß kein Weg am Algorithmus von Huet vorbei führt. Schließlich ist dieser ja vollständig und die dort auftretenden unterschiedlichen Preunifikatoren können nicht durch einen „relevanteren“ ersetzt werden. Denn die Unterschiede betreffen starre Köpfe von Termen, gegen diese ist mittels Substitutionen kein Kraut gewachsen. Also müssen unsere Terme so definiert werden, daß der Algorithmus von Huet terminiert und ein eindeutiges Ergebnis liefert.

Zweitens ergibt sich bald, daß das Betrachten linearer Unifikationsprobleme ausreichend kompliziert ist, Also sollten unsere Terme so gestrickt sein, daß ein lineares Unifikationsproblem nach Durchlaufen des Algorithmus von Huet linear bleibt. Dies ist eine sehr starke Forderung, immerhin besteht dieser Algorithmus maßgeblich darin, Argumentterme vielfach zu kopieren.

Drittens sollten die flexiblen Terme so gestaltet sein, daß man für sie einen relevantesten Unifikator berechnen kann. Es ist also wenig verwunderlich, daß die folgende Definition etwas unübersichtlich erscheinen mag.

Definition. *Ein* Muster mit erweiterten Argumenten *bzgl.* forb *und* flex*, kurz* PExA[17]*, ist ein Term* r *in dem jedes Auftreten* $\lambda\vec{x}.f\vec{r_n}$ *eines* $f \in \mathsf{flex}(r)$ *und jedes Auftreten* $\lambda\vec{y}.c\vec{s_m}$ *einer Konstanten* c *folgenden Restriktionen unterliegt:*

(a) *Fall* $\mathsf{ord}(\tau(f)) \leq 2$*: Für alle* $1 \leq i \leq n$ *sei* $r_i := A\vec{t}$ *mit* $A \in \mathsf{forb} \cup \mathsf{BV}(r)$.

[16] Vom englischen *most relevant unifier*

[17] Aus dem englischen **P***attern with* **Ex***tended* **A***rguments.*

(b) *Fall* $\mathsf{ord}(\tau(f)) \geq 3$*: Für alle* $1 \leq i \leq n$ *ist* $r_i = z_i$ *mit* $z_i \in \mathsf{forb} \cup \mathsf{BV}(r)$.

(c) *Für alle* $1 \leq i, j \leq n$ *gilt* $\mathsf{hd}(r_i) \neq \mathsf{hd}(r_j)$.

(d) *Für alle* $1 \leq i \leq m$ *sei* $s_i := \lambda\vec{v}.A\vec{t}$ *mit* $A \in \mathsf{forb} \cup \mathsf{BV}(r) \setminus \{\vec{v}\}$.

(e) *Für alle* $1 \leq i, j \leq m$ *gilt* $\mathsf{hd}(s_i) \neq \mathsf{hd}(s_j)$.

Ein PExA ist also ein Term, in dem zweitstufige, flexible Variablen auch zusammengesetzte Terme als Argumente haben dürfen, so lange diese paarweise verschiedene, verbotene Variablen als Kopf haben. Als Preis dafür zahlen wir, daß die Argumente von Konstanten ebenfalls dieser Restriktion unterliegen.

Selbst lineare Unifikationsprobleme bestehend aus PExA müssen keinen allgemeinsten Unifikator besitzen, wie das Beispiel

$$\mathsf{UP} := (\{(f_1 y(xg), f_2 z(xc))\}, \{x, y, z\}, \{f_1, f_2, g\}).$$

von zuvor gezeigt hat. Die folgende kleine Aussage zeigt eine erste erfreuliche Eigenschaft dieser Terme und dient auch zur Illustration des Begriffs „PExA“.

Proposition 4.1 *Seien* forb *und* flex *gegeben mit* $\mathsf{forb} \cap \mathsf{flex} = \emptyset$*. Sei* $r := \lambda\vec{x}.fr_1 \cdots r_n$ *ein PExA mit* $f \in \mathsf{flex}$ *und* ϑ *eine Substitution mit* $\mathsf{dom}(\vartheta) \subseteq \mathsf{flex}$*,* $\mathsf{FV}(\vartheta) \cap \mathsf{forb} = \emptyset$ *und* $\vartheta(f) = \lambda\vec{y_n}.s$*. Dann gilt für alle* $1 \leq i \leq n$*:* $y_i \in \mathsf{FV}(s)$ *gdw.* $\vartheta(r_i)$ *ein Teilterm von* $\vartheta(r)$ *ist.*

Beweis. Dazu müssen wir nur beachten, daß die Terme $r_1, \ldots, r_n$ keine Abstraktionsterme sind und daß $\mathsf{hd}(r_i) \in \mathsf{forb}$ ist für alle $1 \leq i \leq n$. Sei $t_i := \vartheta(r_i)$ $(1 \leq i \leq n)$. Dann sind auch die Terme $t_1, \ldots, t_n$ keine Abstraktionsterme und es ist $\mathsf{hd}(t_i) = \mathsf{hd}(r_i)$ für alle $1 \leq i \leq n$. Es gilt $\vartheta(r) = s[\vec{y_n} ::= \vec{t_n}]$.

Für die Richtung „dann“ bemerken wir, daß bei der simultanen Ersetzung kein neuer β-Redex entsteht, da die Terme t_i keine Abstraktionsterme sind. Also nimmt jeder Term t_i den Platz von y_i in s ein.

Für die Richtung „wenn“ stellen wir fest, daß $\mathsf{hd}(t_i) \notin \mathsf{FV}(s)$ sein kann. Ist also t_i ein Teilterm von $\vartheta(r)$, also insbesondere $\mathsf{hd}(t_i) \in \mathsf{FV}(\vartheta(r))$, so kann dieser nur durch die simultane Ersetzung von y_i durch t_i dorthin gelangt sein. Folglich muß $y_i \in \mathsf{FV}(s)$ sein. ■

Wir können also mit einer Substitution ϑ genau steuern, welche Argumente eines flexiblen Terms behalten werden sollen und welche nicht.

Nun beginnen wir mit der Konstruktion unseres Algorithmus zur Berechnung relevantester Unifikatoren für lineare PExA-Unifikationsprobleme. Zunächst betrachten wir den Algorithmus von Huet und überlegen uns, was dieser aus einem solchen Unifikationsproblem macht.

Lemma 4.1 *Sei* $\mathsf{UP} = (\{(r, s)\}, \mathsf{forb}, \mathsf{flex})$ *ein lineares PExA Unifikationsproblem mit* $(r, s) := (fr_1 \cdots r_m, As_1 \cdots s_n)$ *und* $f \in flex$*,* $A \notin \mathsf{flex}$*. Dann terminiert die Konstruktion eines Anpassungsbaumes für* UP*.*

Beweis. Zum Beweis der Terminierung benutzen wir das Maß

$$\|Ar_1 \cdots r_n\|_{\mathsf{UP}} := \begin{cases} n + \sum_{i=1}^{n} \|r_i\|_{\mathsf{UP}} & A \in \mathsf{forb} \cup \mathcal{C} \\ \sum_{i=1}^{n} \|r_i\|_{\mathsf{UP}} & A \in \mathsf{flex} \end{cases}$$

Wir beweisen das Lemma nun mit Induktion über $\|r\|_{\mathsf{UP}} + \|s\|_{\mathsf{UP}}$. Dazu müssen wir die folgenden Fälle unterscheiden.

1. Fall $\mathsf{ord}(\tau(f)) > 2$. Dann ist $r_i = x_i \in \mathsf{forb}$ für alle $1 \leq i \leq m$ und $x_i \neq x_j$ für alle $1 \leq i \neq j \leq m$. Wir unterscheiden nun:
   (a) $A \in \mathsf{forb}$. Dann ist $\mathsf{IMIT}(r, s) = \emptyset$. Folgende Unterfälle für treten auf:
      (i) $A \neq x_i$ für alle $1 \leq i \leq m$. Dann führt uns jede Projektion $\mathsf{PROJ}^i(r, s)$ zu dem Folgeknoten $\bot$.
      (ii) Sonst gibt es genau ein x_i mit $x_i = A$. Und nur die Projektion $\mathsf{PROJ}^i(r, s)$ führt uns nicht zu $\bot$, zumindest nicht unmittelbar. Es ist
      $$\vartheta := \mathsf{PROJ}^i(r, s) := \{\langle f, \lambda \vec{x_m}.x_i(h_1\vec{x_m}) \cdots (h_n\vec{x_m})\rangle\}$$
      und $\vartheta(r) = A(h_1\vec{x_m}) \cdots (h_n\vec{x_m})$. Wir erhalten für den Folgeknoten
      $$\mathsf{UP}' = (\{(h_1\vec{x_m}, s_1), \ldots, (h_n\vec{x_m}, s_n)\}, \mathsf{forb}, \mathsf{flex}')$$
      mit $\mathsf{flex}' = \mathsf{flex} \setminus \{f\} \cup \{h_1, \ldots, h_n\}$. Dies ist offensichtlich wieder ein lineares PExA Unifikationsproblem, da die neuen Variablen $h_1, \ldots, h_n$ paarweise verschieden sind.
      Ist $n > 0$, so gilt für alle $1 \leq i \leq n$: $0 = \|h_i\vec{x_m}\|_{\mathsf{UP}'} = \|f\vec{x_m}\|_{\mathsf{UP}}$ und $\|s_i\|_{\mathsf{UP}'} < n + \sum_{i=1}^{n} \|s_i\|_{\mathsf{UP}} = \|A\vec{s_n}\|_{\mathsf{UP}}$. Also terminiert nach Induktionsvoraussetzung die Konstruktion des Anpassungsbaumes für alle Unifikationsprobleme
      $$\mathsf{UP}_i := (\{(h_i\vec{x}, s_i)\}, \mathsf{forb}, \mathsf{flex}), (1 \leq i \leq n).$$
      Damit terminiert wegen der Linearität auch die Konstruktion für UP'.
      Ist $n = 0$, so ist $\tau(A)$ ein Grundtyp, folglich haben wir
      $$\mathsf{PROJ}^i(r, s) := \{\langle f, \lambda \vec{x_m}.x_i\rangle\}.$$
      Als Folgeknoten erhalten wir dann $\mathsf{UP}' = (\emptyset, \mathsf{forb}, \mathsf{flex} \setminus \{f\})$, die Konstruktion terminiert also mit einem erfolgreichen Ast.
   (b) $A \in \mathcal{C}$. Dann ergibt jede Projektion PROJ^i einen Folgeknoten $\bot$. Es ist also nur $\mathsf{IMIT}(r, s)$ Erfolg versprechend. Mit
   $$\vartheta := \mathsf{IMIT}(r, s) = \{\langle f, \lambda \vec{x}.A(h_1\vec{x}) \cdots (h_n\vec{x})\rangle$$
   und $\vartheta(r) = A(h_1\vec{x_m}) \cdots (h_n\vec{x_m})$ erhalten wir als Folgeknoten
   $$\mathsf{UP}' = (\{(h_1\vec{x}, s_1), \ldots, (h_n\vec{x}, s_n)\} \cup \mathrm{U}, \mathsf{forb}, \mathsf{flex}')$$
   mit $\mathsf{flex}' = \mathsf{flex} \setminus \{f\} \cup \{h_1, \ldots, h_n\}$. Dies ist ein Folgeknoten gleicher Bauart wie im obigen Fall $A \in \mathsf{forb}$, wir können mit unserem Beweis also in gleicher Weise fortfahren.

2. Fall $\mathsf{ord}(f) = 2$. Für alle $1 \leq i \leq m$ ist $\mathsf{hd}(r_i) \in \mathsf{forb}$ und $\tau(r_i)$ ein Grundtyp. Außerdem ist für alle $1 \leq i \neq j \leq m$: $\mathsf{hd}(r_i) \neq \mathsf{hd}(r_j)$. Wir unterscheiden zwei Fälle:

   (a) $A \in \mathsf{forb}$. Dann ist $\mathsf{IMIT}(r, s) = \emptyset$. Folgende Unterfälle sind zu beachten:

      (i) $A \neq \mathsf{hd}(r_i)$ für alle $1 \leq i \leq m$. Dann führen alle Projektionen $\mathsf{PROJ}^i(r, s)$ zum Folgeknoten $\bot$.

      (ii) $A = \mathsf{hd}(r_i)$ für ein $i \in \{1, \ldots, m\}$. Dann führen alle $\mathsf{PROJ}^j(r, s)$ mit $j \neq i$ zum Folgezustand $\bot$. Sei $r_i = At_1 \cdots t_n$. Da $\tau(r_i)$ ein Grundtyp ist, haben wir

      $$\mathsf{PROJ}^i(r, s) = \{\langle f, \lambda\vec{x}.x_i\rangle\}$$

      und $\mathsf{PROJ}^i(r, s)(r) = r_i$. Als Folgeknoten erhalten wir das Unifikationsproblem $\mathsf{SIMPL}(\mathsf{UP}')$ mit

      $$\mathsf{UP}' = (\{(t_1, s_1), \ldots, (t_n, s_n)\}, \mathsf{forb}, \mathsf{flex} \setminus \{f\}).$$

      Ist $n > 0$, so erhalten wir $\|t_i\|_{\mathsf{UP}'} < n + \sum_{i=1}^n \|t_i\|_{\mathsf{UP}} = \|r_i\|_{\mathsf{UP}} \leq \|r\|_{\mathsf{UP}}$ und $\|s_i\|_{\mathsf{UP}'} < n + \sum_{i=1}^n \|s_i\|_{\mathsf{UP}} = \|s\|_{\mathsf{UP}}$. Außerdem haben wir nach Lemma 3.3 für jedes Paar $(t', s') \in \mathsf{SIMPL}(\mathsf{UP}')$ ein Paar $(t_i, s_i) \in \mathsf{UP}'$ mit $\|t'\|_{\mathsf{SIMPL}(\mathsf{UP}')} + \|s'\|_{\mathsf{SIMPL}(\mathsf{UP}')} \leq \|t_i\|_{\mathsf{UP}'} + \|s_i\|_{\mathsf{UP}'} < \|r\|_{\mathsf{UP}} + \|s\|_{\mathsf{UP}}$. Somit terminiert nach Induktionsvoraussetzung die Konstruktion des Anpassungsbaumes für alle Paare $(t', s') \in \mathsf{SIMPL}(\mathsf{UP}')$, damit wegen der Linearität auch die Konstruktion des Baumes für $\mathsf{SIMPL}(\mathsf{UP}')$, also auch diejenige für UP.
      Im Fall $n = 0$ ist $\tau(A)$ ein Grundtyp, also $r_i = A = s$ und damit ist der Folgeknoten $\mathsf{UP}' = (\emptyset, \mathsf{forb}, \mathsf{flex} \setminus \{f\})$. Also terminiert die Konstruktion in einem erfolgreichen Ast.

   (b) $A \in \mathcal{C}$. Wegen $\mathsf{hd}(r_i) \in \mathsf{forb}$ für alle $1 \leq i \leq m$ führen alle Projektionen zu dem Folgeknoten $\bot$, also genügt es, $\mathsf{IMIT}(r, s)$ näher zu betrachten. Wir haben

   $$\mathsf{IMIT}(r, s) = \{\langle f, \lambda\vec{x}.A(h_1\vec{x}) \cdots (h_n\vec{x})\rangle\}$$

   und

   $$\mathsf{IMIT}(r, s)(r) = A(h_1\vec{r_m}) \cdots (h_n\vec{r_m}).$$

   Als Folgeknoten erhalten wir demnach

   $$\mathsf{UP}' = (\{(h_1\vec{r}, s_1), \ldots, (h_n\vec{r}, s_n)\}, \mathsf{forb}, \mathsf{flex} \setminus \{f\} \cup \{h_1, \ldots, h_n\}).$$

   Dies ist zwar ein PExA-, aber kein lineares Unifikationsproblem mehr. Da $\mathsf{hd}(s_i) \in \mathsf{forb}$ für alle $1 \leq i \leq n$, ist $\mathsf{IMIT}(h_i\vec{r}, s_i) = \emptyset$ für alle $1 \leq i \leq n$.
   Betrachten wir zuerst die Möglichkeit, daß für ein $1 \leq k \leq n$ $\mathsf{hd}(s_k) \neq \mathsf{hd}(r_i)$ ist für alle $1 \leq i \leq m$. Dann hat UP' für alle Projektionen $\mathsf{PROJ}^i(h_k\vec{r}, s_k)$ den Folgeknoten $\bot$ und somit endet jeder Ast erfolglos.
   Gelte nun für alle $1 \leq k \leq n$: $\mathsf{hd}(s_k) = \mathsf{hd}(r_{i_k})$ für ein $1 \leq i_k \leq m$. Der Index i_k ist eindeutig, da für alle $1 \leq i \neq j \leq m$ gilt: $\mathsf{hd}(r_i) \neq \mathsf{hd}(r_j)$. Für alle $1 \leq k \neq l \leq n$ ist $i_k \neq i_l$, da $\mathsf{hd}(s_k) \neq \mathsf{hd}(s_l)$.

Betrachten wir nun der Reihe nach die Paare $(h_k r_1 \cdots r_m, s_k)$, $1 \leq k \leq n$, so ist

$$\mathsf{PROJ}^{j_k}(h_k\vec{r}, s_k) = \{\langle h_k, \lambda\vec{x_m}.x_{j_k}\rangle\}$$

und $\mathsf{PROJ}^{j_k}(h_k\vec{r}, s_k)(h_k\vec{r}) = r_{j_k}$.
Als n-ter Folgeknoten von UP' und einziger Ast, der bis dahin nicht in $\perp$ endet, ergibt sich also

$$\mathsf{UP}'' := \mathsf{SIMPL}((\{(r_{j_1}, s_1), \ldots, (r_{j_n}, s_n)\}, \mathsf{forb}, \mathsf{flex} \setminus \{h_1, \ldots h_n\})).$$

Und dies ist tatsächlich wieder ein lineares PExA-Unifikationsproblem und ganz analog wie im obigen Fall $A \in \mathsf{forb}$ gilt auch für alle $(r', s') \in \mathsf{UP}''$: $\|r'\|_{\mathsf{UP}''} + \|s'\|_{\mathsf{UP}''} < \|r\|_{\mathsf{UP}} + \|s\|_{\mathsf{UP}}$. Wir setzen dan fort wie in den Fällen zuvor.

■

Damit ist die Hauptarbeit erledigt, folgender Satz bzw. Folgerung faßt nochmal alles übersichtlich zusammen.

Folgerung 4.1 *Sei* UP *ein lineares PExA-Unifikationsproblem. Dann terminiert der Algorithmus von Huet angewandt auf* UP. *Die Frage, ob ein solches Problem lösbar ist, ist also entscheidbar. Ist* UP *lösbar, so erhalten wir in jedem Anpassungsbaum genau einen erfolgreichen Ast. Dieser endet in einem linearen PExA-Unifikationsproblem.*

Beweis. Die Terminierung folgt aus der Terminierung der Prozedur SIMPL, siehe Lemma 3.3 und obigen Lemma 4.1, da wegen der Linearität die Unifikationspaare in UP eins nach dem Anderen abgearbeitet werden können. Die Eindeutigkeit des erfolgreichen Astes und die Linearität des letzten Knotens kann man im Beweis des vorherigen Lemmas 4.1 erkennen, es existiert immer nur höchstens ein Folgeknoten ungleich $\perp$. ■

Nun müssen wir also noch Unifikatoren für die flexiblen Unifikationsprobleme finden. Wegen der Linearität können wir dabei jedes Unifikationspaar isoliert betrachten. Grundsätzlich gehen wir in etwa so vor wie bei den Mustern, d. h. wir müssen überlegen, welche Argumente der flexiblen Terme wir behalten können und welche nicht.

Sind z. B. fr und gs zwei flexible PExA gleichen Typs und wollen wir diese unifizieren, so können wir die Argumente r und s in jeden Fall dann behalten, wenn r und s unfizierbar sind. Ein Unifikator ϑ hätte dann die Form $\vartheta(f) = \lambda x.hx$ sowie $\vartheta(g) = \lambda x.hx$ und $\vartheta(r) = \vartheta(s)$.

Leider reicht es nicht aus, nur die unifizierbaren Paare zu behalten. Sei in obigen Beispiel etwa $r = x$ und $s = xc$ mit einer verbotenen Variable x und einer Konstanten c. Dann sind r und s offensichtlich nicht unifizierbar, sie haben ja nicht mal den gleichen Typ. Allerdings ist dann eine Substitution ϑ der Gestalt $\vartheta(f) = \lambda x.h$ und $\vartheta(g) = \lambda y.h$ nicht die beste Lösung. Besser ist nämlich $\vartheta(f) = \lambda x.h(xc)$ und $\vartheta(g) = \lambda y.hy$.

Ein solches Paar (x, xc) wollen wir „kritisch“ nennen.

Definition. *Sei* $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein lineares PExA-Unifikationsproblem und sei* $(f\vec{r_m}, g\vec{s_n}) \in \mathrm{U}$ *mit* $f, g \in \mathsf{flex}$. *Ein Paar* (r_i, s_j) *heißt* kritisches Argumentpaar in UP, *falls* $\mathsf{hd}(r_i) = \mathsf{hd}(s_j)$ *aber* $\tau(r_i) \neq \tau(s_j)$ *ist. Das Unifikationsproblem* UP *heißt* kritisch, *falls ein kritisches Argumentpaar in* UP *existiert, sonst* unkritisch

Kritische Argumentpaare können nur auftreten, wenn zwei flexible Terme $f\vec{r}$ und $g\vec{s}$ zusammen treffen, für die genau einer der beiden Köpfe f, g einen Typ der Ordnung 2 hat. Deshalb haben kritische Argumentpaare die folgende Gestalt:

Lemma 4.2 *Sei* $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein lineares PExA-Unifikationsproblem. Dann gilt für jedes kritische Argumentpaar* $(A\vec{r_m}, A\vec{s_n})$ *in* UP*: Entweder* $m = 0$ *oder* $n = 0$.

Beweis. Sei also $(A\vec{r_m}, A\vec{s_n})$ kritische Argumentpaar in UP. Also gibt es ein flex-flex-Paar $(f\vec{r'}, g\vec{s'})$ in UP und Indizes i, j mit $r'_i = A\vec{r_m}$ und $s'_j = A\vec{s_n}$. Wir unterscheiden folgende Fälle abhängig von $\mathsf{ord}(\tau(f))$ bzw. $\mathsf{ord}(\tau(g))$.

Fall $\mathsf{ord}(\tau(f)) = \mathsf{ord}(\tau(g)) = 2$. Dann muß $\mathsf{ord}(\tau(A\vec{r_m})) = \mathsf{ord}(\tau(A\vec{s_n})) = 1$ sein, also auch $\tau(A\vec{r_m}) = \tau(A\vec{s_n})$. Somit kann dies kein kritisches Argumentpaar sein.

Fall $\mathsf{ord}(\tau(f)) \geq 3$ und $\mathsf{ord}(\tau(g)) \geq 3$. Dann ist nach der Definition von PExA $m = n = 0$ und ein kritisches Argumentpaar kann nicht existieren.

Also muß entweder $\mathsf{ord}(\tau(f)) = 2$ oder $\mathsf{ord}(\tau(g)) = 2$ und $\mathsf{ord}(\tau(f)) \neq \mathsf{ord}(\tau(g))$ sein. Damit ist aber nach Definition der PExA entweder $n = 0$ oder $m = 0$. ■

Nun wollen wir uns zunächst flexiblen, linearen PExA-Unifikationsproblemen ohne kritischen Argumentpaaren zuwenden. Dann betseht die Idee zur Konstruktion eines relevantesten Unifikators in der Tat darin, alle Argumente zu behalten, die einen passenden Partner im anderen Term besitzen, mit dem sie unifizierbar sind.

Prozedur 4.1 *Sei* $\mathsf{UP} := (\{(f\vec{r_m}, g\vec{s_n})\}, \mathsf{forb}, \mathsf{flex})$ *ein unkritisches, lineares PExA-Unifikationsproblem mit* $f, g \in \mathsf{flex}$. *Seien* $\vec{x_m}$, $\vec{y_n}$ *zwei Listen von Variablen mit* $\tau(x_i) = \tau(r_i)$ *und* $\tau(y_j) = \tau(s_j)$ *für alle* $1 \leq i \leq m$ *und alle* $1 \leq j \leq n$. *Seien* $\vec{v}$ *und* $\vec{w}$ *Namen für zwei zunächst leere Listen.*

Dann berechnen wir $\mathsf{pff}(\mathsf{UP})$ *mittels folgender Vorschrift:*

- *Wir konstruieren das Unifikationsproblem* $\mathsf{UP}' = (\mathrm{U}', \mathsf{forb}, \mathsf{flex})$ *indem wir setzen:*

$$\mathrm{U}' := \{\, (t, t') \in \{r_1, \ldots, r_m\} \times \{s_1, \ldots, s_n\} \mid \mathsf{hd}(t) = \mathsf{hd}(t') \,\}.$$

- *Für jedes Paar* $(r_i, s_j) \in \mathrm{U}'$ *entscheiden wir mittels des Algorithmus von Huet, ob* $(\{(r_i, s_j)\}, \mathsf{forb}, \mathsf{flex})$ *lösbar ist. Falls nein, ersetzten wir* U' *durch* $\mathrm{U}' \setminus \{(r_i, s_j)\}$. *Falls ja, erweitern wir* $\vec{v}$ *zu* $\vec{v}, x_i$ *und* $\vec{w}$ *zu* $\vec{w}, y_j$.

- *Nun definieren wir* $\vartheta := \{\langle f, \lambda\vec{x}.h\vec{v}\rangle, \langle g, \lambda\vec{y}.h\vec{w}\rangle\}$ *wobei* h *eine neue Variable passenden Typs ist.*

- *Ersetze in* UP' *die Menge* flex *durch* $(\mathsf{flex} \cup \{h\}) \setminus \{f, g\}$.
- *Es ist nun* $\mathsf{pff}(\mathsf{UP}) := (\vartheta, \mathsf{UP}')$.

Dabei sei bemerkt, daß der Typ von h so beschaffen sein muß, daß $\tau(h\vec{v}) = \tau(f\vec{r})$ ist. Es folgt dann wegen $\tau(v_i) = \tau(w_i)$ auch $\tau(h\vec{w}) = \tau(g\vec{s})$.

Nun würden also anschließend die Probleme in UP' gelöst werden. Natürlich ist es dafür von Vorteil, sich schon mal die bereits berechneten Preunifikatoren zu behalten. Dies als Anmerkung zu einer Implementierung.

Die nun folgenden Lemmatas zeigen im Wesentlichen, daß jeder Unifikator eben nur solche Argumente behalten kann, die auch obige Prozedur behält.

Lemma 4.3 *Sei* $\mathsf{UP} := (\{(f\vec{r_m}, g\vec{s_n})\}, \mathsf{forb}, \mathsf{flex})$ *ein unkritisches und lineares PExA-Unifikationsproblem mit* $f, g \in \mathsf{flex}$. *Es sei* $\mathsf{pff}(\mathsf{UP}) = (\vartheta, \mathsf{UP}')$ *mit* $\vartheta(f) = \lambda\vec{x_m}.h\vec{v_k}$ *und* $\vartheta(g) = \lambda\vec{y_n}.h\vec{w_k}$. *Sei* θ *ein Unifikator von* UP *mit* $\theta(f) = \lambda\vec{x_m}.r$ *und* $\theta(g) = \lambda\vec{y_n}.s$. *Dann gilt* $\lambda\vec{v_k}.r = \lambda\vec{w_k}.s$.

Beweis. Es ist $\tau(v_i) = \tau(w_i)$ für alle $1 \leq i \leq k$. Also gilt auf alle Fälle $\lambda\vec{v}.r = \lambda\vec{w}.r[\vec{v_k} ::= \vec{w_k}]$. Es genügt uns also zu zeigen: $r[\vec{v_k} ::= \vec{w_k}] = s$.

Dies gedenken wir durch Induktion über r zu tun.

Induktionsanfang: $r \equiv A \in \mathcal{V} \cup \mathcal{C}$.

Fall $A \notin \{\vec{x_m}\}$. Dann ist $A \notin \mathsf{forb}$, also gilt für alle $1 \leq i \leq n$: $s_i \neq A$, da ja $\mathsf{hd}(s_i) \in \mathsf{forb}$ ist. Wir haben $\theta(f\vec{r_m}) = A = \theta(g\vec{s_n}) = (\lambda\vec{y_n}.s)\vec{s_n}$. Es muß $\{y_1, \ldots, y_n\} \cap \mathsf{FV}(s) = \emptyset$ sein, da sonst wegen Proposition 4.1 ein Term $\theta(s_i)$ Teilterm von $\theta(g\vec{s_n})$ wäre und damit $\mathsf{hd}(s_i) \neq A$ in $\theta(g\vec{s_n})$ vorkäme. Also muß $s = A$ sein.

Fall $A \in \{\vec{x_m}\}$. Sei etwa $A = x_i$. Dann ist $\theta(f\vec{r_m}) = \theta(r_i)$ mit $\mathsf{hd}(\theta(r_i)) = \mathsf{hd}(r_i) \in \mathsf{forb}$. Folglich $\mathsf{hd}(s) \neq \mathsf{hd}(r_i)$. Es ist aber $(\lambda\vec{y_n}.s)\vec{s_n} = \theta(r_i)$, also insbesondere $\mathsf{hd}(\theta(g\vec{s_n})) = \mathsf{hd}(r_i)$. Also muß ein und kann nur ein s_j in $\{s_1, \ldots, s_n\}$ existieren mit $\mathsf{hd}(s_j) = \mathsf{hd}(r_i)$. Da keine kritisches Argumentpaare existieren gilt $\tau(s_j) = \tau(r_i)$, also $\tau(y_j) = \tau(x_i)$. Es muß dann $s = y_j$ sein, denn nur so erhalten wir $\tau(\theta(g\vec{s_n})) = \tau(r_i)$. Dann ist $\theta(g\vec{s_n}) = \theta(s_j)$. Also sind r_i und s_j unifizierbar, damit existiert ein l mit $v_l = x_i$ und $w_l = y_j$. Somit $x_i[v_1 ::= w_1, \ldots, v_k ::= w_k] = y_j$.

Induktionsschluß: Wegen $t_1t_2[\vec{v_k} ::= \vec{w_k}] = t_1[\vec{v_k} ::= \vec{w_k}]t_2[\vec{v_k} ::= \vec{w_k}]$ und $(\lambda z.t)[\vec{v_k} ::= \vec{w_k}] = \lambda z.t[\vec{v_k} ::= \vec{w_k}]$ bei geschickter Wahl von z, ist dieser nur noch eine direkte Anwendung der Induktionshypothesen. ■

Nach Studium dieses Beweises ist uns aufgefallen:

Folgerung 4.2 *Sei* $\mathsf{UP} := (\{(f\vec{r_m}, g\vec{s_n})\}, \mathsf{forb}, \mathsf{flex})$ *ein unkritisches, lineares PExA-Unifikationsproblem mit* $f, g \in \mathsf{flex}$. *Es sei* $\mathsf{pff}(\mathsf{UP}) = (\vartheta, \mathsf{UP}')$ *mit* $\vartheta(f) = \lambda\vec{x_m}.h\vec{v_k}$ *und* $\vartheta(g) = \lambda\vec{y_n}.h\vec{w_k}$. *Sei* θ *ein Unifikator von* UP *mit* $\theta(f) = \lambda\vec{x_m}.r$ *und* $\theta(g) = \lambda\vec{y_n}.s$. *Dann gilt* $x_i \in \mathsf{FV}(r)$ *genau dann, wenn* $y_j \in \mathsf{FV}(s)$ *und* θ *ein Unifikator von* (r_i, s_j) *ist.*

Lemma 4.4 *Sei* $\mathsf{UP} := (\{(f\vec{r_m}, g\vec{s_n})\}, \mathsf{forb}, \mathsf{flex})$ *ein unkritisches und lineares PExA-Unifikationsproblem mit* $f, g \in \mathsf{flex}$*. Es sei* $\mathsf{pff}(\mathsf{UP}) = (\vartheta, \mathsf{UP}')$*. Dann gilt für jeden Unifikator* θ *von* UP*:* $\vartheta \geq \theta$*.*

Beweis. Sei uns also UP und ein beliebiger Unifikator θ von UP gegeben. Sei $\theta(f) = \lambda\vec{x_m}.r$ und $\theta(g) = \lambda\vec{y_n}.s$. Sei $\vartheta(f) = \lambda\vec{x_m}.h\vec{v}$, $\vartheta(g) = \lambda\vec{y_n}.h\vec{w}$. Nach Lemma 4.3 gilt $\lambda\vec{v}.r = \lambda\vec{w}.s$. Nach Folgerung 4.2 ist ein $x_i \in \mathsf{FV}(r)$ gdw. $y_j \in \mathsf{FV}(s)$ und r_i, s_j unifizierbar sind. Damit ist aber x_i in $\vec{v}$ bzw. y_j in $\vec{w}$ enthalten. Also setzen wir $\sigma(h) := \lambda\vec{v}.r$ und erhalten $\sigma(\vartheta(f)) = \lambda\vec{x_m}.r = \theta(f)$ und $\sigma(\vartheta(g)) = \lambda\vec{y_n}.(\lambda\vec{v}.r)\vec{w} = \lambda\vec{y_n}.(\lambda\vec{w}.s)\vec{w} = \lambda\vec{y_n}.s = \theta(g)$. ■

Achtung. Diese Aussage heißt nicht, daß unsere Prozedur pff einen allgemeinsten Unifikator berechnet. Denn dieses ϑ aus dem obigen Lemma ist ja gar kein Unifikator, es ist nur eine Art „Auswahlsubstitution“ für die flexiblen Köpfe.

Nun betrachten wir ein kritisches Argumentpaar etwas genauer. Wegen der Linearität unserer Unifikationsprobleme, genügt es, dabei immer nur ein Unifikationspaar mit genau einem kritischen Argumentpaar zu betrachten.

Lemma 4.5 *Sei* $\mathsf{UP} := (\{(fx, g(x\vec{s_n}))\}, \mathsf{forb}, \mathsf{flex})$ *ein lineares PExA-Unifikationsproblem mit* $f, g \in \mathsf{flex}$*. Sei* θ *ein Unifikator von* UP *mit* $\theta(f) = \lambda x.r$ *und* $\theta(g) = \lambda y.s$*. Es sei* $t_i := \theta(s_i)$ *für alle* $1 \leq i \leq n$*. Dann gilt:*

1. $x \in \mathsf{FV}(r)$ *gdw.* $y \in \mathsf{FV}(s)$*.*
2. $r = s_y[x\vec{t_n}]$*.*
3. *Ist* $x \in \mathsf{FV}(r)$*, so ist* $\mathsf{FV}(t_i) \cap \mathsf{forb} = \emptyset$ *für alle* $1 \leq i \leq n$*.*

Beweis.

1. Ist $x \in \mathsf{FV}(r)$, so ist auch $x \in \mathsf{FV}(\theta(fx))$. Da $x \in \mathsf{forb}$ ist, kann x nicht in $\mathsf{FV}(\theta)$ sein, also auch nicht in $\mathsf{FV}(s)$. Folglich muß $y \in \mathsf{FV}(s)$ sein, da nur so $x \in \mathsf{FV}(\theta(g(x\vec{s_n}))$ möglich ist. Rückrichtung analog.
2. Mit $t_i = \theta(s_i)$ für alle $1 \leq i \leq n$ erhalten wir $s_y[x\vec{t}] = \theta(g(x\vec{s})) = \theta(fx) = (\lambda x.r)x = r$.
3. Dies folgt unmittelbar aus Aussage 2, da sonst $\mathsf{FV}(\vartheta) \cap \mathsf{forb} \neq \emptyset$ wäre und somit ϑ kein Unifikator von UP sein könnte.

■

Zur relevantesten Unifikation müssen wir also ggf. verbotene Variablen aus Termen entfernen, damit wir sie in einem Unifikator verwenden können. Dazu benötigen wir also wieder eine Prozedur zum Ausschneiden bzw. „prunen“ von verbotenen Variablen. Allerdings ist die Situation nun etwas komplizierter als bei Mustern. Wollen wir z. B. die Variable y aus dem PExA $f(x(gy))$ entfernen, so ist natürlich die Substitution $\{\langle g, \lambda y.h\rangle\}$ „besser“ als die Substitution $\{\langle f, \lambda z.h\rangle\}$.

Dazu definieren wir erst mal eine Funktion, die überprüft ob verbotene Variablen überhaupt entfernt werden können oder nicht, also ob verbotene Variablen als Argumente flexibler Variablen auftreten oder nicht.

Definition. *Sei* $r = \lambda\vec{x}.A\vec{r_m}$ *ein lineares PExA bzgl. gegebener Mengen* forb *and* flex. *Sei* $V \subseteq$ forb *mit* $V \cap \mathsf{BV}(r) = \emptyset$. *Dann definieren wir die Funktion* Vcheck *durch:*

$$\mathsf{Vcheck}(r, V) := \begin{cases} \mathsf{wahr} & A \in \mathsf{flex} \\ \mathsf{falsch} & A \in V \\ \bigwedge_{i=1}^{m} \mathsf{Vcheck}(r_i, V) & \textit{sonst.} \end{cases}$$

Das folgende Lemma zeigt, daß obige Prozedur Vcheck das Versprochene leistet.

Lemma 4.6 *Sei* r *ein lineares PExA bzgl.* forb *und* flex *und sei* $V \subseteq$ forb *mit* $V \cap \mathsf{BV}(r) = \emptyset$. *Ist* $\mathsf{Vcheck}(r, V) = \mathsf{falsch}$, *so gilt für jede Substitution* ϑ *mit* $\mathsf{dom}(\vartheta) \subseteq$ flex*:* $V \cap \mathsf{FV}(\vartheta(r)) \neq \emptyset$.

Beweis. Wir beweisen dies durch Induktion über r. Am Anfang sei $r = \lambda\vec{x}.A$ mit $A \notin \{x_1, \ldots, x_n\}$. Da $\mathsf{Vcheck}(r, V) = \mathsf{falsch}$ ist, muß $A \in V$ sein. Also gilt für jede Substitution ϑ mit $\mathsf{dom}(\vartheta) \subseteq$ flex: $\vartheta(r) = \lambda\vec{x}.A$. Folglich ist $A \in V \cap \mathsf{FV}(\vartheta(r))$.

Sei nun $r = \lambda\vec{x}.Ar_1 \cdots r_n$ mit $V \cap \mathsf{BV}(r) = \emptyset$ und $\mathsf{Vcheck}(r, V) = \mathsf{falsch}$. Also ist $A \notin$ flex. Demnach gilt für jede Substitution ϑ mit $\mathsf{dom}(\vartheta) \subseteq$ flex: $\vartheta(r) = \lambda\vec{x}.A\vartheta(r_1) \cdots \vartheta(r_n)$. Ist $A \in V$, so ist also auch $A \in V \cap \mathsf{FV}(\vartheta(r))$. Ist $A \notin V$, so existiert ein $r_i \in \{r_1, \ldots, r_n\}$ mit $\mathsf{Vcheck}(r_i, V) = \mathsf{falsch}$. Nach Induktionsvoraussetzung gilt $V \cap \mathsf{FV}(\vartheta(r)) \supseteq V \cap \mathsf{FV}(\vartheta(r_i)) \neq \emptyset$. ■

Sind $\vartheta_1, \ldots, \vartheta_n$ Substitutionen, so sei $\bigcirc_{i=1}^{n}\vartheta_i := \vartheta_1 \circ \cdots \circ \vartheta_n$. Es bezeichne † eine „unmögliche“ Substitution. Diese habe für alle Substitutionen ϑ die Eigenschaft, daß $\dagger \circ \vartheta = \vartheta \circ \dagger = \dagger$ ist.

Dann können wir die Prozedur Prune definieren. Sie versucht stets, möglichst weit im Inneren des Terms zu operieren, so wie im einführenden Beispiel illustriert.

Definition. *Es sei* $r := \lambda\vec{x}.A\vec{r_n}$ *ein lineares PExA bzgl.* forb *und* flex. *und* $V \subseteq$ forb *mit* $V \cap \mathsf{BV}(r) = \emptyset$. *Dann definieren wir die* Beschneidungssubstitution $\mathsf{Prune}(r, V)$ *rekursiv wie folgt:*

- $A \in$ flex*:*

  $$\mathsf{Prune}(r, V) := \{\langle A, \lambda\vec{y_n}.gy_{i_1} \cdots y_{i_k}\rangle\} \circ \bigcirc_{j=1}^{k}\mathsf{Prune}(r_{i_j}, V)$$

  wobei $\{r_{i_1}, \ldots, r_{i_k}\} = \{\, r \in \{r_1, \ldots, r_n\} \mid \mathsf{Vcheck}(r, V) = \mathsf{wahr} \,\}$ *und* g *eine neue Variable passenden Typs ist.*

- $A \in V$*:* $\mathsf{Prune}(r, V) := \dagger$

- *Sonst:* $\mathsf{Prune}(r, V) := \mathsf{Prune}(r_1, V) \circ \cdots \circ \mathsf{Prune}(r_n, V)$

Eine ganz einfache Folgerung von Lemma 4.6 ist:

Lemma 4.7 *Sei r ein lineares PExA bzgl.* forb *and* flex *und $V \subseteq$ forb mit $V \cap \mathsf{BV}(r) = \emptyset$. Sei $\mathsf{Prune}(r, V) = \dagger$. Dann gilt für jede Substitution ϑ mit $\mathsf{dom}(\vartheta) \subseteq$ flex: $V \cap \mathsf{FV}(\vartheta(r)) \neq \emptyset$.*

Nun ist zu zeigen, daß die berechnete Beschneidungssubstitution tatsächlich die „beste“ Lösung ist.

Lemma 4.8 *Sei r ein lineares PExA bzgl.* forb *and* flex *und $V \subseteq$ forb mit $V \cap \mathsf{BV}(r) = \emptyset$. Sei ϑ eine Substitution mit $\mathsf{dom}(\vartheta) \subseteq$ flex, so daß $V \cap \mathsf{FV}(\vartheta(r)) = \emptyset$ ist. Dann ist $\mathsf{Prune}(r, V) \geq \vartheta$.*

Beweis. Nach Lemma 4.7 ist $\mathsf{Prune}(r, V) \neq \dagger$. Es sei $r = \lambda\vec{x}.Ar_n^{\rightarrow}$. Wir fahren in bewährter Manier mit Induktion über r fort. Im Anfang sei $r = \lambda\vec{x}.A$. Es ist $A \notin V$. Ist $A \in \mathcal{C}$, so ist $\mathsf{Prune}(r, V) = \varepsilon \geq \vartheta$. Ist $A \in$ flex, so ist $\mathsf{Prune}(r, V) = \{\langle A, g\rangle\} \geq \vartheta$.

Im Induktionsschluß sei nun $r = \lambda\vec{x}.Ar_n^{\rightarrow}$. Ist $A \in \mathcal{C}$, so folgt die Aussage unmittelbar aus der Induktionshypothese, denn $\mathsf{Prune}(r, V) = \mathsf{Prune}(r_1, V) \cup \cdots \cup \mathsf{Prune}(r_n, V)$ (wegen der Linearität gilt für alle $i \neq j$: $\mathsf{Prune}(r_i, V) \| \mathsf{Prune}(r_j, V)$) und $\mathsf{Prune}(r_i, V) \geq \vartheta$ (IH) für alle $1 \leq i \leq n$. Ist $A \in$ flex, so sei

$$\mathsf{Prune}(r, V) := \{\langle A, \lambda y_n^{\rightarrow}.gy_{i_1} \cdots y_{i_k}\rangle\} \circ \bigcirc_{j=1}^{k} \mathsf{Prune}(r_{i_j}, V)$$

und $\vartheta(A) = \lambda y_n^{\rightarrow}.s$. Nach Lemma 4.6 und Proposition 4.1 gilt

$$\mathsf{FV}(s) \cap \{y_1, \ldots, y_n\} \subseteq \{y_{i_1}, \ldots, y_{i_k}\}.$$

Nach Induktionshypothese ist $\mathsf{Prune}(r_{i_j}, V) \geq \vartheta$ für alle $1 \leq j \leq k$. Mit $\sigma(g) := \lambda y_{i_1}, \ldots, y_{i_k}.s$ gilt $(\sigma \circ \mathsf{Prune}(r, V))(A) = \vartheta(A)$, folglich wegen der Linearität unseres Unifikationsproblems und der daraus folgenden Parallelität der Substitutionen $\mathsf{Prune}(r_{i_j}, V)$, $1 \leq j \leq k$, folgt $\mathsf{Prune}(r, V) \geq \vartheta$. ∎

Nach all diesen Vorbereitungen können wir nun zwei flexible PExA mit kritischem Argumentpaar unifizieren.

Prozedur 4.2 Bearbeitung kritischer Argumentpaare $\mathsf{kp}((r, s), V)$.

Sei uns folgendes gegeben: Ein kritisches Argumentpaar $(r, s) := (x, x\vec{s_l})$ mit $l > 0$ und $V \subseteq$ forb. Dann geben wir folgendes aus:

- *Falls $\mathsf{Prune}(s_j, V) = \dagger$ ist für ein $1 \leq j \leq l$: $\mathsf{kp}((x, x\vec{s_l}), \mathsf{forb}) := \bot$.*
- *Sonst sei für alle $1 \leq j \leq l$: $t_j := \mathsf{Prune}(s_j, V)(s_j)$ und $\vartheta = \mathsf{Prune}(s_1, V) \circ \cdots \circ \mathsf{Prune}(s_l, V)$. Dann definieren wir $\mathsf{kp}((x, x\vec{s_l}), \mathsf{forb}) := (xt_1 \cdots t_l, \vartheta)$.*

Der Sinn dieser Prozedur erschließt sich nach dem Studium des folgenden Lemmas.

Lemma 4.9 *Sei $\mathsf{UP} := (\{(fx, g(x\vec{s_l}))\}, \mathsf{forb}, \mathsf{flex})$ ein lineares PExA-Unifikationsproblem mit $f, g \in$ flex, $x \in$ forb und $l > 0$. Sei $V :=$ forb $\setminus \{x\}$. Dann gilt:*

1. *Ist* $\mathsf{kp}((x, x\vec{s_l}), V) = \bot$, *so ist* $\vartheta := \{\langle f, \lambda x.h\rangle, \langle g, \lambda y.h\rangle\}$ *mit einer neuen Variable* h *ein allgemeinster Unifikator von* UP.

2. *Ist* $\mathsf{kp}((x, x\vec{s_l}), V) = (t, \varrho)$, *so ist* $\vartheta := \{\langle f, \lambda x.ht\rangle, \langle g, h\rangle\} \circ \varrho$ *mit einer neuen Variable* h *ein relevantester Unifikator von* UP.

Beweis. Sei θ ein Unifikator von UP mit $\theta(f) = \lambda x.r$ und $\theta(g) = \lambda y.s$. Sei $V = \mathsf{forb} \setminus \{x\}$.

1. In diesem Fall ist also $\mathsf{Prune}(s_j, V) = \dagger$ für mindestens ein $1 \leq j \leq l$. Nach Lemma 4.7 ist damit $\mathsf{FV}(\theta(s_j)) \cap \mathsf{forb} \neq \emptyset$. Damit folgt nach Lemma 4.5 daß $x \notin \mathsf{FV}(r)$ und $y \notin \mathsf{FV}(s)$ ist. Ebenfalls aus Lemma 4.5 folgt damit $r = s$. Also haben wir mit $\sigma(h) := r$: $\sigma(\vartheta(f)) = \lambda x.r = \theta(f)$ und $\sigma(\vartheta(g)) = \lambda y.s = \theta(g)$.

2. Hier müssen wir zwei Fälle unterscheiden.

   (a) $x \notin \mathsf{FV}(r)$. Nach Lemma 4.5 gilt dann $y \notin \mathsf{FV}(s)$ und $r = s$. Wir setzen $\sigma(h) := \lambda y.s$ und erhalten $\sigma(\vartheta(f)) = \lambda x.(\lambda y.s)t = \lambda x.r = \theta(f)$ sowie $\sigma(\vartheta(g)) = \lambda y.s = \theta(g)$. Es ist $\mathsf{dom}((\sigma \circ \vartheta)^{\mathsf{R}}_{\mathsf{UP}}) = \{f, g\}$ und damit $(\sigma \circ \vartheta)^{\mathsf{R}}_{\mathsf{UP}} \geq_{\mathsf{flex}} \theta$.

   (b) $x \in \mathsf{FV}(r)$. Nach Lemma 4.5 ist $y \in \mathsf{FV}(s)$ und $r = s_y[x\theta(s_1) \cdots \theta(s_l)]$. Nach Lemma 4.7 existieren Substitutionen $\sigma_1, \ldots, \sigma_l$ mit $\mathsf{dom}(\sigma_i) \subseteq \mathsf{FV}(\mathsf{Prune}(s_i, V))$ und $(\sigma_i \circ \mathsf{Prune}(s_i, V)) \geq \theta\,|_{\mathsf{FV}(s_i)}$ für alle $1 \leq i \leq l$. Wegen der Linearität folgt für $\sigma := \sigma_1 \circ \cdots \circ \sigma_l$: $\sigma \circ \varrho \geq \theta$. Also existiert ein σ' mit $\sigma'(\sigma(\varrho(A))) = \theta(A)$ für alle $A \in \mathsf{dom}(\varrho)$. Erweitern wir noch σ' durch $\sigma'(h) = \lambda y.s$, so erhalten wir für f, g:

   $$\begin{aligned}\sigma'(\sigma(\vartheta(f))) = \quad & \lambda x.(\lambda y.s)(x\sigma'(\sigma(\varrho(s_1))) \cdots \sigma'(\sigma(\varrho(s_l)))) = \\ & \lambda x.s_y[x\theta(s_1) \cdots \theta(s_l)] = \theta(f)\end{aligned}$$

   und $\sigma'(\sigma(\vartheta(g))) = \lambda y.s = \theta(g)$.

■

Nun setzten wir beide Prozeduren, also die für Terme ohne kritische Argumentpaare und die Prozedur kp zusammen zum „Flex-Flex-Schritt für lineare PExA“.

Prozedur 4.3 *Seien* $f\vec{r_m}$ *and* $g\vec{s_n}$ *flexible, lineare PExA bzgl.* forb *and* flex *ohne gemeinsame flexible Variablen. Seien* $\vec{x_m}$ *und* $\vec{y_n}$ *Listen von Variablen mit* $\tau(x_i) = \tau(r_i)$ *and* $\tau(y_j) = \tau(s_j)$ *für alle* $1 \leq i \leq m$ *and* $1 \leq j \leq n$. *Dabei soll für alle* i, j *gelten:* $r_i = u \in \mathcal{V}$ *bzw.* $s_j = v \in \mathcal{V}$ *gdw.* $x_i = u$ *bzw.* $y_j = v$. *Seien* $\vec{X}$ *und* $\vec{Y}$ *zwei leere Listen. Sei* $\theta := \varepsilon$, $V :=$ flex *und* $\mathrm{U}' := \emptyset$. *Dann ist der* Flex-Flex-Schritt *durch folgenden Algorithmus definiert:*

1. *Setze* $M := \{\, (r_i, s_j) \mid \mathsf{hd}(r_i) = \mathsf{hd}(s_j) \,\}$.

2. *Führe folgende Schritte für jedes Paar* $(r_i, s_j) \in M$ *aus:*

   (a) *Falls* r_i *und* s_j *unifizierbar sind: Setze* $\vec{X} ::= \vec{X}, x_i$, $\vec{Y} ::= \vec{Y}, y_j$, $\mathrm{U}' ::= \mathrm{U}' \cup \{(r_i, s_j)\}$.

(b) *Falls* $(r_i, s_j) := (x_i, x_i\vec{s_l'})$ *ein kritisches Argumentpaar ist mit* $l > 0$, *und* $(t, \varrho) := \mathsf{kp}(x_i, x_i\vec{s_l'}, \mathsf{forb} \setminus \{x_1, \ldots, x_m\})$ *ist, so ersetze* $\vec{X}$ *durch* $(\vec{X}, t)$, $\vec{Y}$ *durch* $(\vec{Y}, y_j)$ *und* θ *durch* $\varrho \circ \theta$. *Ist* $\mathsf{kp}(x_i, x_i\vec{s_l'}, \mathsf{forb} \setminus \{x_1, \ldots, x_m\}) = \bot$, *so mache nichts.*

(c) *Falls* $(r_i, s_j) := (y_j\vec{r_k'}, y_j)$ *ein kritisches Argumentpaar ist mit* $k > 0$, *und ist* $(t, \varrho) := \mathsf{kp}(y_j, y_j\vec{r_k'}, \mathsf{forb} \setminus \{y_1, \ldots, y_n\})$, *so ersetzte* $\vec{X}$ *durch* $(\vec{X}, x_i)$, $\vec{Y}$ *durch* $(\vec{Y}, t)$ *und* θ *durch* $\varrho \circ \theta$. *Ist* $\mathsf{kp}(y_j, y_j\vec{r_k'}, \mathsf{forb} \setminus \{y_1, \ldots, y_n\}) = \bot$, *so mache nichts.*

(d) *Sind* r_i *und* s_j *nicht unifizierbar und kein kritisches Argumentpaar, so tue nichts.*

3. *Nun definiere* $\sigma := \{\langle f, \lambda\vec{x_m}.h\vec{X}\rangle, \langle g, \lambda\vec{y_n}.h\vec{Y}\rangle\}$ *mit einer neuen, passenden Variable* h *und* $\mathsf{flex}' ::= (\mathsf{flex} \setminus \mathsf{dom}(\theta)) \cup \mathsf{FV}(\theta)$.

4. *Gib nun* $(\sigma \circ \theta) \mid_V$, flex', U' *aus.*

Damit haben wir alle Zutaten für einen vollständigen Algorithmus zur Berechnung relevantester Unifikatoren für lineare PExA-Unifikationsprobleme zusammen.

Algorithmus 4 Der Unifikationsalgorithmus für lineare PExA-Unifikationsprobleme. *Sei* $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein lineares PExA Unifikationsproblem. Zur Konstruktion eines relevantesten Unifikators führe folgenden Algorithmus aus:*

1. *Setze* $\theta := \varepsilon$, $V := \mathsf{flex}$.

2. *Führe folgende Schritte aus, bis* $\mathrm{U} = \emptyset$ *oder* $\mathsf{UP} = \bot$ *ist:*

   (a) *Wähle ein Paar* $(r, s) \in \mathrm{U}$.

   (b) *Ist* r *oder* s *ein starrer Term, so wende den Algorithmus von Huet auf* (r, s) *an. Sind* r *und* s *nicht unifizierbar, so setze* $\mathsf{UP} := \bot$. *Sonst sei* σ *der errechnete Pre-Unifikator und* U' *das resultierende flexible Unifikationsproblem. Dann setze* $\theta ::= \sigma \circ \theta$ *und* $\mathrm{U} ::= \mathrm{U} \setminus \{(r, s)\} \cup \mathrm{U}'$.

   (c) *Sind* r *und* s *flexible Terme, so rufe die Prozedur 4.3 auf mit* (r, s), forb *and* flex. *Sei seine Antwort* σ, U' *und* flex'. *Dann setze* $\theta ::= \sigma \circ \theta$, $\mathrm{U} ::= \mathrm{U} \setminus \{(r, s)\} \cup \mathrm{U}'$ *und* $\mathsf{flex} ::= \mathsf{flex}'$.

3. *Ist* $\mathrm{U} = \emptyset$, *so gib* $\theta \mid_V$ *aus. Ist* $\mathsf{UP} = \bot$, *so gib* $\bot$ *aus.*

Theorem 4.1 *Sei* $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein lineares PExA Unifikationsproblem.*

- *Die Anwendung des Algorithmus 4 auf* UP *terminiert.*
- UP *ist genau dann nicht lösbar, wenn der obige Unifikationsalgorithmus angewandt auf* UP *mit* $\bot$ *antwortet.*
- *Gibt obiger Algorithmus angewandt auf* UP *eine Substitution* ϑ *aus, so ist dieses* ϑ *ein relevantester Unifikator von* UP.

Beweis. Die Terminierung folgt aus dem Lemma 4.1 und der Tatsache, daß die in der Prozedur 4.3 erzeugte Menge von Unifikationspaaren aus „kleineren“ Termen als die ursprüngliche Menge besteht. Die zweite Aussage folgt aus den Tatsachen über den Huet'schen Algorithmus, siehe Kapitel 3.3. Die letzte Aussage folgt aus der Eindeutigkeit des Anpassungsbaumes (Folgerung 4.1) sowie den Lemmatas über die Unifikation flexibler, linearer PExA-Unifikationsprobleme (Lemma 4.4 und Lemma 4.9). ■

Jetzt warten wir gespannt auf den Praxistest. Also ein Beispiel.

Beispiel. Kehren wir zunächst zu unserem Ausgangsbeispiel zurück, also $\mathsf{UP} := (\{(f_1 y(xg), f_2 z(xc))\}, \{x, y, z\}, \{f_1, f_2, g\})$. Wir setzten also $\vartheta := \varepsilon$ und $V := \{f_1, f_2, g\}$. Da nur eine Unifikationspaar existiert wählen wir dieses aus und stellen fest, daß beide Terme darin flexibel sind. Also rufen wir die Prozedur 4.3 auf:

Es ist $x_1 := y$, $y_1 := z$ und es sei x_2 eine Variable mit $\tau(x_2) = \tau(xg)$ und y_2 eine Variable mit $\tau(y_2) = \tau(xc)$. Wir haben $M := \{(xg, xc)\}$. Dieses Paar ist unifizierbar, also erhalten wir $\vec{X} := x_2$, $\vec{Y} := y_2$ und $\mathrm{U}' := \{(xg, xc)\}$. Wir definieren also $\sigma := \{\langle f_1, \lambda y, x_2.hx_2\rangle, \langle f_2, \lambda z, y_2.hy_2\rangle\}$. Außerdem ist $\mathsf{flex}' = \{g, h\}$.

Es ist nun $\vartheta := \sigma \circ \varepsilon = \sigma$, $\mathsf{UP} = (\{(xg, xc)\}, \{x, y, z\}, \{g, h\})$. Wir machen daraus $\mathsf{SIMPL}(\mathsf{UP}) = (\{(g, c)\}, \{x, y, z\}, \{g, h\})$. Es folgt $\mathsf{IMIT}(g, c) = \{\langle g, c\rangle\}$ und damit der erfolgreiche Ast im Anpassungsbaum, dieser endet sogar in einer leeren Menge von Unifikationsproblemen.

Es ist also dann $\vartheta = \{\langle g, c\rangle, \langle f_1, \lambda y, x_2.hx_2\rangle, \langle f_2, \lambda z, y_2.hy_2\rangle\}$, $U = \emptyset$.

Also ist $\vartheta = \{\langle g, c\rangle, \langle f_1, \lambda y, x_2.hx_2\rangle, \langle f_2, \lambda z, y_2.hy_2\rangle\}$ der berechnete, relevanteste Unifikator.

Machen wir es uns etwas komplizierter. Sei

$$\mathsf{UP} = (\{(f(u_1(f' u_3(u_1(f'' v))))u_2(u_3 v), g u_2 u_3 u_1)\}, \{u_1, u_2, u_3, v\}, \{f, f', f'', g\})$$

mit $c \in \mathcal{C}$. Es handelt sich um ein flex-flex-Paar mit einer zweitstufigen Variablen f. Wir starten also wieder mit der Prozedur 4.3.

Es sei x_1 eine Variable mit $\tau(x_1) = \tau(u_1(f' u_3(u_1(f'' v))))$, $x_2 := u_2$, x_3 eine Variable mit $\tau(x_3) = \tau(u_3 v)$, $y_1 := u_2$, $y_2 := u_3$ und $y_3 := u_1$. Nun bilden wir die Menge der Argumentpaare mit gleichem Kopf:

$$M = \{(u_1(f' u_3(u_1(f'' v))), u_1), (u_2, u_2), (u_3 v, u_3)\}.$$

Das erste Paar ist offensichtlich ein kritisches Argumentpaar, wir rufen

$$\mathsf{kp}(u_1, u_1(f' u_3(u_1(f'' v))), \{v\}).$$

Es sei zu beachten, daß $y_3 = u_1$ und $\mathsf{forb} \setminus \{u_2, u_3, u_1\} = \{v\}$ ist.

Es folgt der Aufruf von $\mathsf{Prune}(f' u_3(u_1(f'' v)), \{v\})$. Da $f' \in \mathsf{flex}$ ist, müssen wir $\mathsf{Vcheck}(u_3, \{v\})$ und $\mathsf{Vcheck}(u_1(f'' v), \{v\})$ berechnen. Es ist $\mathsf{Vcheck}(u_3, \{v\}) = \mathsf{wahr}$, da leere Konjunktionen als wahr betrachtet werden. Es ist

$$\mathsf{Vcheck}(u_1(f'' v), \{v\}) = \mathsf{Vcheck}(f'' v, \{v\}) = \mathsf{wahr},$$

da $f'' \in \mathsf{flex}$. Es ist $\mathsf{Prune}(u_3, \{v\}) = \varepsilon$ und

$$\mathsf{Prune}(u_1(f''v), \{v\}) = \mathsf{Prune}(f''v, \{v\}) = \{\langle f'', \lambda w.h\rangle\}.$$

Folglich ist

$$\mathsf{Prune}(f'u_3(u_1(f''v)), \{v\}) = \{\langle f', \lambda w_1, w_2.h'w_1w_2\rangle, \langle f'', \lambda w.h\rangle\} =: \varrho.$$

Also ist $t = h'u_3(u_1h)$ und

$$\mathsf{kp}(u_1, u_1(f'u_3(u_1(f''v))), \{v\}) = (u_1t, \varrho).$$

Folglich setzen wir $\vec{X} := x_1$, $\vec{Y} := u_1t$, $\vartheta := \varrho$.

Das zweite Paar aus M, nämlich (u_2, u_2), ist offensichtlich unifizierbar. Also ist nun $\vec{X} = x_1, u_2$ und $\vec{Y} = u_1t, u_2$ und $\mathrm{U}' = \{(u_2, u_2)\}$.

Das dritte Paar ist wieder ein kritisches, diesmal erhalten wir aber

$$\mathsf{Prune}(u_3v, \{v\}) = \mathsf{Prune}(v, \{v\}) = \dagger,$$

also ist $\mathsf{kp}(u_3v, u_3, \{v\}) = \bot$, folglich ändern wir nichts.

Nun definieren wir

$$\sigma := \{\langle f, \lambda x_1, u_2, x_3.\tilde{h}x_1u_2\rangle, \langle g, \lambda u_2, u_3, u_1.\tilde{h}(u_1(h'u_3(u_1h)))u_2\rangle\}$$

und erhalten

$$\begin{aligned}\theta = \sigma \circ \vartheta = \quad & \{\langle f, \lambda x_1, u_2, x_3.\tilde{h}x_1u_2\rangle,\\ & \langle g, \lambda u_2, u_3, u_1.\tilde{h}(u_1(h'u_3(u_1h)))u_2\rangle,\\ & \langle f', \lambda w_1, w_2.h'w_1w_2\rangle,\\ & \langle f'', \lambda w.h\rangle\}.\end{aligned}$$

und $\mathsf{flex}' = \{h, h', \tilde{h}\}$.

Damit kehren wir von der Prozedur 4.3 zurück und erhalten nun $\vartheta = \theta$ und $\mathsf{UP} = (\{(u_2, u_2), \{u_1, u_2, u_3, v\}, \{h, h', \tilde{h}\}$. UP ist lösbar mit Unifikator ε, also ist jenes θ der relevanteste Unifikator von UP.

Nun stellt sich uns natürlich die Frage nach der Linearität, ist sie denn wirklich notwendig? Folgendes Beispiel sei dazu gewählt:

$$\mathsf{UP} = (\{(f(xg), f(xc)), (f'(xg), f'(xd))\}, \{x\}, \{f, f', g\})$$

mit $c, d \in \mathcal{C}$. Beginnen wir mit dem ersten Problem, so erhalten wir als relevantesten Unifikator:

$$\{\langle f, \lambda y.hy\rangle, \langle g, c\rangle\}.$$

Angewandt auf das zweite Paar erhalten wir das Problem

$$\mathsf{UP}' = (\{(f'(xc), f'(xd))\}, \{x\}, \{f', h\}).$$

Dies hat den relevantesten Unifikator

$$\{\langle f', \lambda z.h'\rangle\}.$$

Alles zusammen erhalten wir

$$\theta_1 = \{\langle f, \lambda y.hy\rangle, \langle g, c\rangle, \langle f', \lambda z.h'\rangle\}.$$

Beginnen wir aber mit dem zweiten Paar un dem dafür relevantesten Unifikator

$$\{\langle f', \lambda z.h''z\rangle, \langle g, d\rangle\},$$

so erhalten wir nach Anwendung auf das erste Paar

$$\mathsf{UP}' = (\{(f(xd), f(xc))\}, \{x\}, \{f, h''\})$$

mit der relevantesten Lösung

$$\{\langle f, \lambda y.\tilde{h}\rangle\}.$$

und alles zusammen

$$\theta_2 = \{\langle f', \lambda z.h''z\rangle, \langle g, d\rangle, \langle f, \lambda y.\tilde{h}\rangle\}.$$

Leider existiert nun weder eine Substitution σ_1 mit $(\sigma_1 \circ \theta_1)^{\mathsf{R}}_{\mathsf{UP}} \geq \theta_2$ noch eine Substitution σ_2 mit $(\sigma_2 \circ \theta_2)^{\mathsf{R}}_{\mathsf{UP}} \geq \theta_1$. Möglicherweise bilden θ_1 und θ_2 eine Art „vollständige Menge von relevantesten Unifikatoren“.

Ein weiteres Beispiel bringt nun auch die Prozeduren des Huet'schen Algorithmus ins Spiel sowie den Fall unifizierbarer Argumente flexibler Variablen.

Beispiel. Sei $\mathsf{UP} :=$

$$(\{(f(z_1(f'(z_3f''))z_4))z_2, c(z_1(g(z_1(g'z_3))))))\}, \{z_1, z_2, z_3, z_4\}, \{f, f', f'', g, g'\}).$$

Es sind also f und g zweitstufige Variablen, sei $\tau(f) = \iota \to \iota \to \iota$ und $\tau(g) = \iota \to \iota$. Sei $\theta := \varepsilon$.

Wir starten mit MATCH, hier genauer mit

$$\vartheta := \mathsf{IMIT}(f(z_1(f'(z_3f''))z_4))z_2, c(z_1(g(z_3(g'z_3))))))) = \{\langle f, \lambda x_1, x_2.c(hx_1x_2)\rangle\}.$$

Angewandt auf UP ergibt sich $\vartheta(\mathsf{UP}) =$

$$(\{(c(h(z_1(f'(z_3f''))z_4))z_2), c(z_1(g(z_3(g'z_3))))))), \{z_1, z_2, z_3, z_4\}, \{h, f', f'', g, g'\}).$$

Wir setzen $\theta = \{\langle f, \lambda x_1, x_2.c(hx_1x_2)\rangle\}$. Es folgt $\mathsf{SIMPL}(\mathsf{UP}) =$

$$(\{(h(z_1(f'(z_3f''))z_4))z_2, z_1(g(z_3(g'z_3))))\}, \{z_1, z_2, z_3, z_4\}, \{h, f', f'', g, g'\}).$$

Es geht weiter mit

$$\vartheta := \mathsf{PROJ}^1(h(z_1(f'(z_3f''))z_4))z_2, z_1(g(z_3(g'z_3)))) = \{\langle h, \lambda x_1, x_2.x_1\rangle\}.$$

Wir erhalten dann

$$\vartheta(\mathsf{UP}) = (\{(z_1(f'(z_3f''))z_4), z_1(g(z_3(g'z_3)))), \{z_1, z_2, z_3, z_4\}, \{f', f'', g, g'\})$$

und

$$\theta = \{\langle h, \lambda x_1, x_2.x_1\rangle, \langle f, \lambda x_1, x_2.cx_1\rangle\}.$$

Weiter geht's mit

$$\mathsf{SIMPL}(\mathsf{UP}) = (\{(f'(z_3 f'')z_4, g(z_3(g' z_3)))\}, \{z_1, z_2, z_3, z_4\}, \{f', f'', g, g'\}).$$

Nun kommt ein flex-flex-Schritt. Die Menge der Argumente mit gleichem Kopf ist $M = \{(z_3 f'', z_3(g' z_3))\}$ und dieses Paar ist unifizierbar, da $f'', g' \in \mathsf{flex}$ sind und beide Terme den gleichen Typ haben. Also haben wir als Antwort der Prozedur 4.3:

$$\begin{aligned} &\vartheta = \{\langle f', \lambda x_1, x_2.h' x_1\rangle, \langle g, \lambda y_1.h' y_1\rangle\},\\ &\mathsf{flex}' = \{f'', g', h'\},\\ &\mathrm{U}' = \{(z_3 f'', z_3(g' z_3))\}. \end{aligned}$$

Wir machen also weiter mit dem Unifikationsproblem

$$\mathsf{UP} = (\{(z_3 f'', z_3(g' z_3))\}, \{z_1, z_2, z_3, z_4\}, \{f'', g', h'\})$$

und haben nun

$$\theta = \{\langle f', \lambda x_1, x_2.h' x_1\rangle, \langle g, \lambda y_1.h' y_1\rangle, \langle h, \lambda x_1, x_2.x_1\rangle, \langle f, \lambda x_1, x_2.c x_1\rangle\}.$$

Weiter mit SIMPL, wir erhalten

$$\mathsf{SIMPL}(\mathsf{UP}) = (\{(f'', g' z_3)\}, \{z_1, z_2, z_3, z_4\}, \{f'', g', h'\}).$$

Also erneut Prozedur 4.3. Es ist $M = \emptyset$, folglich erhalten wir

$$\begin{aligned} &\vartheta = \{\langle f'', h''\rangle, \langle g', \lambda y_1.h''\rangle\},\\ &\mathsf{flex}' = \{h', h''\},\\ &\mathrm{U}' = \emptyset. \end{aligned}$$

Dies führt nun zum Unifikationsproblem

$$\mathsf{UP} = (\emptyset, \{z_1, z_2, z_3, z_4\}, \{h', h''\})$$

und θ wird zu

$$\begin{aligned} \{\quad &\langle f'', h''\rangle, \langle g', \lambda y_1.h''\rangle\\ &\langle f', \lambda x_1, x_2.h' x_1\rangle, \langle g, \lambda y_1.h' y_1\rangle, \langle h, \lambda x_1, x_2.x_1\rangle, \langle f, \lambda x_1, x_2.c x_1\rangle \quad \}. \end{aligned}$$

Wir sind also fertig und erhalten als relevantesten Unifikator:

$$\begin{aligned} \theta\,|_{\{f, f', f'', g, g'\}} = \{\quad &\langle f'', h''\rangle, \langle g', \lambda y_1.h''\rangle\\ &\langle f', \lambda x_1, x_2.h' x_1\rangle, \langle g, \lambda y_1.h' y_1\rangle, \langle f, \lambda x_1, x_2.c x_1\rangle \quad \}. \end{aligned}$$

4.2 Vom MRU zur CSU

Nun wollen wir aus einem gegebenen relevantesten Unifikator eines linearen PExA-Unifikationsproblems eine vollständige Menge von Unifikatoren ablesen. Wir werden sehen, daß dies auf recht einfache Art und Weise möglich ist und wir dazu nicht einmal das ursprüngliche Unifikationsproblem benötigen.

Zunächst wollen wir zwei Arten von Substitutionen besonders benennen.

Definition. *Eine Substitution π heißt* offene Projektion *bzgl.* forb *und* flex*, falls für alle $\langle f, r\rangle \in \pi$ gilt: $r = \lambda\vec{x}.g\vec{y}$ mit $g \in \mathcal{V}$, $g \notin \mathsf{forb}$, $\{\vec{y}\} \subseteq \{\vec{x}\}$ und für alle $i \neq j$ ist $y_i \neq y_j$.*

Der Name „offene Projektion“ beschreibt sehr gut die Wirkung der Anwendung solcher Substitutionen auf einen Term $fr_1 \cdots r_n$. Ist $\pi(f) = \lambda\vec{x_n}.gx_{i_1} \cdots x_{i_k}$, so ist $\pi(f\vec{r_n}) = gr_{i_1} \cdots r_{i_k}$.

Definition. *Eine Substitution ϱ heißt* argumenterhaltend*, falls für alle $\langle f, r\rangle \in \varrho$ gilt: $r = \lambda\vec{x}.s$ mit $\{\vec{x}\} \subseteq \mathsf{FV}(s)$.*

Der Name „argumenterhaltend“ soll insbesondere den Gegensatz zur offenen Projektion betonen. Allerdings kann für einen Term $fr_1 \cdots r_n$ und eine argumenterhaltende Substitution $\varrho(f) = \lambda\vec{x_n}.s$ es durchaus vorkommen, daß einige der Argumente r_i doch verschwinden. Sei z. B. der Term $fr(\lambda y.d)$ gegeben und $\varrho(f) = \lambda x, z.zx$. Dann ist ϱ zwar Argumenterhaltent, aber $\varrho(fr(\lambda y.d)) = (\lambda y.d)r = d$. Also verschwindet der Argumentterm r. Es sei aber hier schon darauf hingewiesen, daß dies bei PExAs nicht möglich ist, dies folgt aus der Proposition 4.1.

Nun folgt, daß jede Substitution aus einer offenen Projektion und einer argumenterhaltenden Substitution zusammengesetzt werden kann.

Proposition 4.2 *Ist ϑ eine Substitution, so existiert eine offene Projektion π mit $\mathsf{dom}(\pi) = \mathsf{dom}(\vartheta)$ und eine argumenterhaltende Substitution ϱ mit $\mathsf{dom}(\varrho) \subseteq \mathsf{FV}(\pi)$, so daß $\varrho \circ \pi = \vartheta$ ist.*

Beweis. Für jede Bindung $\langle f, \lambda\vec{x}.s\rangle \in \vartheta$ sei $\pi(f) := \lambda\vec{x}.g\vec{y}$ mit $\{\vec{y}\} = \{\vec{x}\} \cap \mathsf{FV}(s)$ und passender neuen Variable g und $\varrho(g) := \lambda\vec{y}.s$. ■

Dies nutzen wir nun aus, um zu beweisen, daß wir zur Konstruktion einer vollständigen Menge von Unifikatoren aus einem relevantesten Unifikator nur offene Projektionen betrachten müssen.

Theorem 4.2 *Sei ϑ ein relevantester Unifikator für ein lineares PExA-Unifikationsproblem $\mathsf{UP} = (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ und sei θ ein beliebiger Unifikator von UP. Dann existiert eine offene Projektion π mit $\mathsf{dom}(\pi) \subseteq \mathsf{FV}(\vartheta)$, so daß $(\pi \circ \vartheta)^{\mathsf{R}}_{\mathsf{UP}} \geq_{\mathsf{flex}} \theta$ ist.*

Beweis. Es existiert eine Substitution σ mit $(\sigma \circ \vartheta)^{\mathsf{R}}_{\mathsf{UP}} \geq_{\mathsf{flex}} \theta$ und $\mathsf{dom}(\sigma) \subseteq \mathsf{FV}(\vartheta)$. Nach Proposition 4.2 ist $\sigma = \varrho \circ \pi$ mit einem argumenterhaltenden ϱ und einer offenen Projektion π sowie $\mathsf{dom}(\pi) = \mathsf{dom}(\sigma)$ und $\mathsf{dom}(\varrho) \subseteq \mathsf{FV}(\pi)$. Wir zeigen $\mathsf{dom}(\sigma \circ \vartheta)^{\mathsf{R}}_{\mathsf{UP}} = \mathsf{dom}(\pi \circ \vartheta)^{\mathsf{R}}_{\mathsf{UP}}$.

„$\supseteq$“. Sei $f \in \mathsf{dom}((\pi \circ \vartheta)^{\mathsf{R}}_{\mathsf{UP}})$. Zunächst stellen wir fest, daß $f \in \mathsf{dom}(\vartheta)$ sein muß, da die Variablen in $\mathsf{dom}(\sigma)$ allesamt neue Variablen sind, die in den Termen von UP gar nicht vorkommen. Es sei $\langle f, t\rangle \in \vartheta$ und $\vartheta' = \vartheta \setminus \langle f, t\rangle$. Dann existiert ein $r \in \mathsf{UP}$ mit $f \in \mathsf{FV}((\pi \circ \vartheta')(r))$. Nun bleibt zu zeigen: $f \in \mathsf{FV}(\varrho(s))$ mit $s = \pi(\vartheta'(r))$. Es ist s ein lineares PExA. Induktion über s.

Induktionsanfang $s \equiv A$. Nach Voraussetzung ist dann $A = f$, folglich wegen $f \notin \mathsf{dom}(\varrho)$: $\varrho(s) = f$.

Induktionsschluß: Fall $s = As_1 \cdots s_n$. Unterfall $A \notin \mathsf{dom}(\varrho)$: Es gilt $\varrho(s) = A\varrho(s_1) \cdots \varrho(s_n)$. Ist nun $A = f$, so sind wir fertig. Ist $f \in \mathsf{FV}(s_i)$ für ein i, so ist nach Induktionsvoraussetzung auch $f \in \mathsf{FV}(\varrho(s_i))$.

Unterfall $A \in \mathsf{dom}(\varrho)$. Dann ist $A \neq f$. Sei $f \in \mathsf{FV}(s_i)$ für ein i. Dann ist nach Induktionsvoraussetzung $f \in \mathsf{FV}(\varrho(s_i))$. Es sei $\varrho(A) = \lambda\vec{y_n}.t$ mit $\{\vec{y_n}\} \subseteq \mathsf{FV}(t)$. Nach Lemma 4.1 ist dann $\varrho(s_i)$ Teilterm von $\varrho(s)$, also auch $f \in \mathsf{FV}(\varrho(s))$.

Ist $s = \lambda x.t$, so folgt die Aussage direkt aus der Induktionshypothese, da wir $x \neq f$ und $\varrho(\lambda x.t) = \lambda x.\varrho(t)$ annehmen können.

„$\subseteq$“. Umgekehrt gilt $\mathsf{dom}(\sigma \circ \vartheta)^{\mathsf{R}}_{\mathsf{UP}} \subseteq \mathsf{dom}(\pi \circ \vartheta)^{\mathsf{R}}_{\mathsf{UP}}$, da $\sigma = \varrho \circ \pi$ ist. ■

Folgerung 4.3 *Ist ϑ ein relevantester Unifikator eines linearen PExA-Unifikationsproblems UP, so ist*

$$\{\, (\pi \circ \vartheta)^{\mathsf{R}}_{\mathsf{UP}} \mid \pi \text{ offene Projektion mit } \mathsf{dom}(\pi) = \mathsf{FV}(\vartheta) \,\}$$

eine vollständige Menge von Unifikatoren für UP.

Betrachten wir den relevantesten Unifikator θ eines flexiblen, linearen PExA-Unifikationspaares $(f\vec{r_m}, g\vec{s_n})$, so hat dieser die Form

$$\{\langle f, \lambda\vec{x_n}.h(x_{i_1}\vec{t}) \cdots (x_{i_k}\vec{t})\rangle, \langle g, \lambda\vec{y_m}.h(y_{i'_1}\vec{t}) \cdots (y_{i'_k}\vec{t})\rangle\} \cup \vartheta_1 \cup \cdots \cup \vartheta_k$$

wobei eine Substitutionen ϑ_i entweder eine Ausschneidesubstitutionen (im Falle eines kritischen Argumentpaares) oder ein relevantester Unifikator eines Argumentpaares ist.

Definieren wir nun eine offene Projektion π mit $\pi(h) = \lambda\vec{v_k}.h'v_{j_1} \cdots v_{j_l}$, so folgt aus der Linearität des Unifikationsproblems

$$\begin{aligned}(\pi \circ \theta)^{\mathsf{R}}_{\mathsf{UP}} \quad &= \{\langle f, \lambda\vec{x_n}.h(x_{i_{j_1}}\vec{t}) \cdots (x_{i_{j_l}}\vec{t})\rangle, \langle g, \lambda\vec{y_m}.h(y'_{i_{j_1}}\vec{t}) \cdots (y'_{i_{j_l}}\vec{t})\rangle\} \\ &\cup (\pi \circ \vartheta_{j_1})^{\mathsf{R}}_{\mathsf{UP}} \cup \cdots \cup (\pi \circ \vartheta_{j_l})^{\mathsf{R}}_{\mathsf{UP}}.\end{aligned}$$

So können wir also eine vollständige Menge von Unifikatoren berechnen, ohne das ursprüngliche Problem noch zu kennen. Die Kenntnis des relevantesten Unifikators genügt, sofern wir uns merken, zu welchem Argumentpaar ein ϑ_k gehört.

Beispiel. Im letzten Abschnitt hatten wir das Beispiel $\mathsf{UP} :=$

$$(\{(f(z_1(f'(z_3f'')z_4))z_2, c(z_1(g(z_1(g'z_3))))))\}, \{z_1, z_2, z_3, z_4\}, \{f, f', f'', g, g'\}).$$

mit dem relevantesten Unifikator

$$\begin{aligned}\theta = \{ \quad &\langle f'', h''\rangle, \langle g', \lambda y_1.h''\rangle \\ &\langle f', \lambda x_1, x_2.h'x_1\rangle, \langle g, \lambda y_1.h'y_1\rangle, \langle f, \lambda x_1, x_2.cx_1\rangle \quad \}.\end{aligned}$$

behandelt. Wir zerlegen θ in drei Teile, nämlich

$$\{\langle f, \lambda x_1, x_2.cx_1\rangle \cup (\{\langle f', \lambda x_1, x_2.h'x_1\rangle, \langle g, \lambda y_1.h'y_1\rangle\} \cup \{\langle f'', h''\rangle, \langle g', \lambda y_1.h''\rangle\}_1)\}.$$

Dabei soll der Index 1 angeben, daß dieser Teil zu dem ersten Arguumentpaar des flexiblen Paares gehört. Es ist $\mathsf{FV}(\theta) = \{h', h''\}$. Die einzige „sinnvolle“, offene Projektion ist $\pi = \{\langle h', \lambda x.\tilde{h}\rangle\}$. Es ist

$$(\pi \circ \theta)^{\mathsf{R}}_{\mathsf{UP}} = \{\langle f, \lambda x_1, x_2.cx_1\rangle, \langle f', \lambda x_1, x_2.\tilde{h}\rangle, \langle g, \lambda y_1.\tilde{h}\rangle\},$$

da der Teil $\{\langle f'', h''\rangle, \langle g', \lambda y_1.h''\rangle\}$ aufgrund der Projektion, welches das erste Argumentpaar löscht, wegfällt. Wir erhalten als vollständige Menge von Unifikatoren:

$$\{\quad \{\langle f'', h''\rangle, \langle g', \lambda y_1.h''\rangle, \langle f', \lambda x_1, x_2.h'x_1\rangle, \langle g, \lambda y_1.h'y_1\rangle, \langle f, \lambda x_1, x_2.cx_1\rangle\},$$
$$\{\langle f, \lambda x_1, x_2.cx_1\rangle, \langle f', \lambda x_1, x_2.\tilde{h}\rangle, \langle g, \lambda y_1.\tilde{h}\rangle\}\}.$$

Mitunter ist es nicht nötig, alle offenen Projektionen zu betrachten. Sei z. B. das Unifikationsproblem

$$\mathsf{UP} = (\{(f(xf')yz, g(xc)y)\}, \{x, y, z\}, \{f, g, f'\})$$

mit $c \in \mathcal{C}$ gegeben. Dann ist

$$\theta = \{\langle f, \lambda x_1, y, z.hx_1y\rangle, \langle g, \lambda x_1, y.hx_1y\rangle, \langle f', c\rangle\}$$

der relevanteste Unifikator von UP. Nun hätten wir als mögliche offene Projektion $\pi_1 = \{\langle h, \lambda x_1, y.h'x_1\rangle\}$, $\pi_2 = \{\langle h, \lambda x_1, y.h''y\rangle\}$ und $\pi_3 = \{\langle h, \lambda x_1, y.h'''\rangle\}$. Es ist

$$(\pi_1 \circ \theta)^{\mathsf{R}}_{\mathsf{UP}} = \{\langle f, \lambda x_1, y, z.h'x_1\rangle, \langle g, \lambda x_1, y.h'x_1\rangle, \langle f', c\rangle\},$$
$$(\pi_2 \circ \theta)^{\mathsf{R}}_{\mathsf{UP}} = \{\langle f, \lambda x_1, y, z.h''y\rangle, \langle g, \lambda x_1, y.h''y\rangle\},$$
$$(\pi_3 \circ \theta)^{\mathsf{R}}_{\mathsf{UP}} = \{\langle f, \lambda x_1, y, z.h'''\rangle, \langle g, \lambda x_1, y.h'''\rangle\}.$$

Es folgt offensichtlich $\theta \geq (\pi_1 \circ \theta)^{\mathsf{R}}_{\mathsf{UP}}$ und $(\pi_2 \circ \theta)^{\mathsf{R}}_{\mathsf{UP}} \geq (\pi_3 \circ \theta)^{\mathsf{R}}_{\mathsf{UP}}$. Also genügt als vollständige Menge von Unifikatoren: $\{\theta, (\pi_2 \circ \theta)^{\mathsf{R}}_{\mathsf{UP}}\}$. Dies liegt daran, daß das zweite Argumentpaar (y, y) als Unifikator die leer Substitution ε hat. Wenn wir also in der offenen Projektion für h das Argument y weglassen, ändern wir nicht viel, genauer gesagt bleibt der Domain bei der Reduktion auf relevante Bindungen erhalten. Fassen wir diese Erkenntnis zu einer Proposition zusammen:

Proposition 4.3 *Sei θ ein relevantester Unifikator für ein lineares PExA-Unifikationsproblem UP und seien π_1 und π_2 offene Projektionen mit $\mathsf{dom}(\pi_i) \subseteq \mathsf{FV}(\theta)$ $(i = 1, 2)$, so daß $\mathsf{dom}((\pi_1 \circ \theta)^{\mathsf{R}}_{\mathsf{UP}}) = \mathsf{dom}((\pi_2 \circ \theta)^{\mathsf{R}}_{\mathsf{UP}})$. Dann existiert eine offene Projektion π mit $\mathsf{dom}(\pi) \subseteq \mathsf{FV}(\theta)$ und $(\pi \circ \theta)^{\mathsf{R}}_{\mathsf{UP}} \geq (\pi_i \circ \theta)^{\mathsf{R}}_{\mathsf{UP}}$ für $i = 1, 2$.*

Beweis. Sei $f \in \mathsf{FV}(\theta)$, $\pi_1(f) = \lambda\vec{x_n}.gx_{i_1} \cdots x_{i_k}$ und $\pi_2(f) = \lambda\vec{x_n}.g'x_{j_1} \cdots x_{j_l}$. Sei $\{\vec{y}\} = \{x_{i_1}, \ldots, x_{i_k}\} \cup \{x_{j_1}, \ldots, x_{j_l}\}$. Dann definieren wir $\pi(f) := \lambda\vec{x_n}.h\vec{y}$ mit neuer, pasender Variable h und stellen fest, daß π das Gewünschte leistet. ■

Ist also $(f\vec{r}, g\vec{s})$ ein flexibles, lineares Paar von PExAs und θ der relevanteste Unifikator davon mit

$$\theta = \{\langle f, \lambda\vec{x}.h\vec{r_k'}\rangle, \langle g, \lambda\vec{y}.h\vec{s_k'}\rangle\} \cup \vartheta_1 \cdots \vartheta_k$$

und gelte $\vartheta_{i_1} = \cdots = \vartheta_{i_l} = \varepsilon$, so genügt es für h nur diejenigen offenen Projektionen $\langle h, \lambda \vec{z_k}.h' z_{j_1} \cdots z_{j_m} \rangle$ zu betrachten, für die $\{i_1, \ldots, i_l\} \subseteq \{j_1, \ldots, j_m\}$ ist.

Beispiel. Im letzten Abschnitt hatten wir das Unifikationsproblem

$$\mathsf{UP} = (\{(f(u_1(f' u_3(u_1(f'' v))))u_2(u_3 v), g u_2 u_3 u_1)\}, \{u_1, u_2, u_3, v\}, \{f, f', f'', g\})$$

betrachtet, dieses hat den relevantesten Unifikator

$$\theta = \{\langle f, \lambda x_1, u_2, x_3.\tilde{h} x_1 u_2 \rangle, \langle g, \lambda u_2, u_3, u_1.\tilde{h}(u_1(h' u_3(u_1 h)))u_2 \rangle\} \cup \vartheta_1 \cup \vartheta_2$$

mit $\vartheta_1 = \{\langle f'', \lambda w.h \rangle\}$ und $\vartheta_2 = \varepsilon$. Wir haben uns hier die reine Namensumbenennung für f' erspart. Zur Berechnung einer vollständigen Menge von Unifikatoren müssen wir also alle möglichen offenen Projektionen für $\tilde{h}$ betrachten. Dies sind $\langle \tilde{h}, \lambda x_1, x_2.h_1 x_1 \rangle$, $\langle \tilde{h}, \lambda x_1, x_2.h_2 x_2 \rangle$ und $\langle \tilde{h}, \lambda x_1, x_2.h_3 \rangle$. Von diesen Projektionen ist aber nach der letzten Proposition nur die zweite, also $\langle \tilde{h}, \lambda x_1, x_2.h_2 x_2 \rangle$ interessant, da nur diese die Variable x_2 enthält und $\vartheta_2 = \varepsilon$ ist. Damit erhalten wir als vollständige Menge von Unifikatoren:

$$\begin{array}{ll} \{ & \{\langle f, \lambda x_1, u_2, x_3.\tilde{h} x_1 u_2 \rangle, \langle g, \lambda u_2, u_3, u_1.\tilde{h}(u_1(h' u_3(u_1 h)))u_2 \rangle, \langle f'', \lambda w.h \rangle\}, \\ & \{\langle f, \lambda x_1, u_2, x_3.h_2 u_2 \rangle, \langle g, \lambda u_2, u_3, u_1.h_2 u_2 \rangle\}\}. \end{array}$$

5 Unifikation mit Fallunterscheidung

Anhand eines Beispiels erläutern wir den Unterschied zwischen der konventionellen Unifikation und der Unifikation mit Fallunterscheidung.

Sei ein Unifikationsproblem $\mathsf{UP} = (\{(f0, 1)\}, \emptyset, \{f\})$ gegeben mit Konstanten $0 \neq 1$. Der Algorithmus von Huet liefert als einzige Lösung die Substitution $\vartheta = \{\langle f, \lambda x.1\rangle\}$. Anderenseits kann man obiges Unifikationsproblem auch als Gleichung verstehen, mit der man eine Eigenschaft einer Funktion f beschreibt, nämlich daß f an der Stelle 0 den Wert 1 annimmt. Legt man diese Gleichung eine Menschen (Mathematiker) vor, der von formaler Unifikation keine Ahnung hat, so wird er f wohl wie folgt definieren:

$$f(x) := \begin{cases} 1 & x = 0 \\ \tilde{f}(x) & x \neq 0 \end{cases} \qquad \text{(L1)}$$

Ein Informatiker würde dies übersetzen in ein Programm der Gestalt

$$\lambda x.\mathsf{Wenn}\, x = 0, \mathsf{dann}\, 1, \mathsf{sonst}\, \tilde{f}x$$

mit einem nicht näher spezifizierten $\tilde{f}$, das anzeigt, daß die Funktion für Eingaben ungleich 0 noch undefiniert ist. Den gravierenden Unterschied erkennt man nun, wenn man ein zweites Paar $(f1, 2)$ mit einer Konstanten $2 \neq 1$ hinzunimmt. Es ist $\vartheta(f1) = 1 \neq 2$, also ist das zusammengesetzte Problem $\mathsf{UP}' = (\{(f0, 1), (f1, 2)\}, \emptyset, \{f\})$ im einfach getypten λ-Kalkül unlösbar.

Dies widerspricht natürlich jeder Anschauung. Wendet wir obige „menschliche“ Lösung (L1) auf das zweite Paar an, so erhalten wir das neue Paar $(\tilde{f}1, 2)$ und als Lösung dessen

$$\tilde{f}(x) := \begin{cases} 2 & x = 1 \\ \hat{f}(x) & x \neq 1 \end{cases} \qquad \text{(L2)}$$

Setzen wir nun (L1) und (L2) zusammen, so entsteht

$$f(x) := \begin{cases} 1 & x = 0 \\ 2 & x = 1 \\ \hat{f}(x) & x \neq 0, 1 \end{cases}$$

Oder wie es der Programmierer sagen würde:

$$\lambda x.\mathsf{Wenn}\, x = 0, \mathsf{dann}\, 1, \mathsf{sonst}\, (\mathsf{Wenn}\, x = 1, \mathsf{dann}\, 2, \mathsf{sonst}\, \hat{f}x).$$

Der Unterschied zwischen konventioneller Unifikation und Unifikation mit Fallunterscheidung liegt also darin, das letztere die punktweise Definition von Funktionen erlaubt.

Auch ein weiteres populäres Problem der Unifikation wird eindeutig lösbar, nämlich $\mathsf{UP} = (\{(f0, 0)\}, \emptyset, \{f\})$. Die konventionelle Unifikation liefert uns zwei Lösungen, als da wären

$$\vartheta_1 = \{\langle f, \lambda x.x\rangle\} \qquad \text{und} \qquad \vartheta_2 = \{\langle f, \lambda x.0\rangle\}.$$

Leider gilt weder $\vartheta_1 \geq \vartheta_2$, noch $\vartheta_2 \geq \vartheta_1$. Ein allgemeinster Unifikator im einfach getypten λ-Kalkül existiert also nicht.

Mit Hilfe einer Fallunterscheidung können wir dieses Problem umgehen. Wir definieren

$$\theta := \{\langle f, \lambda x.\mathsf{Wenn}\, x = 0, \mathsf{dann}\, 0, \mathsf{sonst}\, f'x\rangle\}$$

und stellen fest

$$\theta(f0) = (\mathsf{Wenn}\, 0 = 0, \mathsf{dann}\, 0, \mathsf{sonst}\, f'0) = 0.$$

Ausserdem gilt tatsächlich $\theta \geq \vartheta_i (i = 1, 2)$ in einem Kalkül mit Fallunterscheidung. Wir definieren dazu $\sigma_1 := \{\langle f', \lambda y.y\rangle\}$ und $\sigma_2 := \{\langle f', \lambda y.0\rangle\}$. Dann erhalten wir:

$$\begin{array}{lll} (\sigma_1 \circ \theta)(f) & = \sigma_1(\lambda x.\mathsf{Wenn}\, x = 0, \mathsf{dann}\, 0, \mathsf{sonst}\, f'x) & \\ & = \lambda x.\mathsf{Wenn}\, x = 0, \mathsf{dann}\, 0, \mathsf{sonst}\, (\lambda y.y)x & \\ & = \lambda x.\mathsf{Wenn}\, x = 0, \mathsf{dann}\, 0, \mathsf{sonst}\, x & \\ & = \lambda x.x & = \vartheta_1(f) \end{array}$$

wobei die letzte Gleichheit darauf beruht, daß im Falle $x = 0$ eben x gleich 0 ist. Analog erkennen wir

$$\begin{array}{lll} (\sigma_2 \circ \theta)(f) & = \sigma_2(\lambda x.\mathsf{Wenn}\, x = 0, \mathsf{dann}\, 0, \mathsf{sonst}\, f'x) & \\ & = \lambda x.\mathsf{Wenn}\, x = 0, \mathsf{dann}\, 0, \mathsf{sonst}\, (\lambda y.0)x & \\ & = \lambda x.\mathsf{Wenn}\, x = 0, \mathsf{dann}\, 0, \mathsf{sonst}\, 0 & \\ & = \lambda x.0 & = \vartheta_2(f). \end{array}$$

Nun wollen wir erst mal einen Kalkül mit Fallunterscheidung definieren und seine elementaren Eigenschaften beweisen, nämlich daß er stark normalisierend und konfluent ist. Danach beweisen wir, daß Unifikation mit Fallunterscheidung generell unentscheidbar ist. Allerdings sehen wir anschließend, daß es durchaus sinnvolle Anwendungen geben kann, z. B. um den Musterbegriff aus Kapitel 2 zu erweitern. Dabei müssen wir aber zwei Fälle unterscheiden, nämlich ob wir nur lineare Unifikationsprobleme zulassen wollen oder auch solche, wo flexible Variablen mehrfach auftreten können.

5.1 Lambda Kalkül mit beschränkter Fallunterscheidung

Wir wollen einen Kalkül angeben, in dem wir Unifikation mit Fallunterscheidung definieren können. Ziel ist es, den Begriff Muster so zu erweitern, daß wir auch Konstanten als Argumente flexibler Variablen zulassen. Deshalb genügt es uns, die Fallunterscheidung nur in der Form

$$\mathsf{Wenn}\, r = c, \mathsf{dann}\, s, \mathsf{sonst}\, t$$

mit einer Konstanten c zu erlauben. Dies hat den großen Vorteil, daß die Frage der Gleichheit von Konstanten einfach zu entscheiden ist.

Dazu definieren wir zunächst Terme mit Fallunterscheidung, dann eine erweiterte β-Reduktion (mit Regeln für die Fallunterscheidung) und zeigen, daß dieser Kalkül stark normalisierend und konfluent ist. Danach definieren wir eine η-Gleichheit auf Termen in β-Normalform und erhalten schließlich, daß wie im λ-Kalkül jeder term eine lange Normalform besitzt.

Definition. *Zu jedem Typ α sei eine abzählbar unendliche Menge von Variablen $V_\alpha = \{ x_n^\alpha \mid n \in \mathbb{N} \}$ gegeben. Es sei $\mathcal{V} := \bigcup_{\alpha \in \mathcal{T}} V_\alpha$ die Menge aller Variablen.*

Zu jedem Typ α sei eine endliche Menge $C_\alpha := \{ c_1, \ldots, c_n \mid n \geq 0 \}$ von Konstanten gegeben. Es sei $\mathcal{C} := \bigcup_{\alpha \in \mathcal{T}} C_\alpha$ die Menge aller Konstanten.

Dann ist die Menge der einfach getypte $\lambda\mathsf{D}$-Terme *über $\mathcal{V}$ und $\mathcal{C}$ induktiv definiert durch:*

Für alle $\alpha \in \mathcal{T}$ ist jede Variable $x \in V_\alpha$ und jede Konstante $c \in C_\alpha$ ein $\lambda\mathsf{D}$-Term vom Typ α.
Ist r ein $\lambda\mathsf{D}$-Term vom Typ $\alpha \to \beta$ und s ein $\lambda\mathsf{D}$-Term vom Typ α, so ist (rs) ein $\lambda\mathsf{D}$-Term vom Typ β.
Ist r ein $\lambda\mathsf{D}$-Term vom Typ β und $x \in V_\alpha$, so ist $(\lambda x r)$ ein $\lambda\mathsf{D}$-Term vom Typ $\alpha \to \beta$.
Ist $c \in C_\alpha$, r ein $\lambda\mathsf{D}$-Term vom Typ α und sind s und t $\lambda\mathsf{D}$-Terme vom Typ β, so ist $\mathsf{D}_c(r,s,t)$ ein $\lambda\mathsf{D}$-Term vom Typ β.

Es steht also $\mathsf{D}_c(r,s,t)$ für den Ausdruck

$$\mathsf{Wenn}\, r = c, \mathsf{dann}\, s, \mathsf{sonst}\, t.$$

Es gelte auch wieder, daß Konstanten mit unterschiedlichen Namen tatsächlich verschieden sind, insbesondere soll nun stets $c \neq d$ sein. Die Menge der *Atome* ist wieder die Menge $\mathcal{C} \cup \mathcal{V}$. Für Atome verwenden wir die Buchstaben $A, B, \ldots$.

Proposition 5.1 *Die Terme des einfach getypten $\lambda\mathsf{D}$-Kalküls gehorchen folgender Grammatik, wobei eine korrekte Typisierung vorausgesetzt wird:*

$$\begin{aligned} \lambda\mathsf{D} \ni r, s, s', t, \vec{t} ::= \quad & A\vec{t} \mid \lambda x.r \mid D_c(r,s,t) \quad (r \notin \mathcal{C}) \mid \\ & (\lambda x.r)s\vec{t} \mid \mathsf{D}_c(c,r,s) \mid \mathsf{D}_c(d,r,s) \mid \mathsf{D}_c(r,s,s')t\vec{t}. \end{aligned}$$

Beweis. Offenbar sind alle Terme die nach dieser Grammatik gebaut werden $\lambda\mathsf{D}$-Terme. Die Rückrichtung geht durch einfache Induktion über die Definition der $\lambda\mathsf{D}$-Terme. ■

Die Definition der Mengen der freien und gebundenen Variablen müssen wir in sinnvoller Weise erweitern, indem wir folgende Klausel jeweils hinzufügen:

$$\begin{aligned} \mathsf{FV}(\mathsf{D}_c(r,s,t)) &:= \mathsf{FV}(r) \cup \mathsf{FV}(s) \cup \mathsf{FV}(t) \\ \mathsf{BV}(\mathsf{D}_c(r,s,t)) &:= \mathsf{BV}(r) \cup \mathsf{BV}(s) \cup \mathsf{BV}(t). \end{aligned}$$

Auch die Definition der simultanen Ersetzung erweitern wir in natürlicher Weise um

$$\mathsf{D}_c(r,s,t)[\vec{x}_n ::= \vec{t}_n] := \mathsf{D}_c(r[\vec{x}_n ::= \vec{t}_n], s[\vec{x}_n ::= \vec{t}_n], t[\vec{x}_n ::= \vec{t}_n]).$$

Die Äquivalenzrelation $\equiv_\alpha$ (Umbenennung gebundener Variablen) erweitern wir um die Klausel $D_c(r,s,t) \equiv_\alpha D_c(r',s',t')$ falls $r \equiv_\alpha r'$, $s \equiv_\alpha s'$ und $t \equiv_\alpha t'$.

Nun kommen wir zu den Reduktionen.

Definition. *Die Konversionen $\mapsto$ bestehen aus folgenden Regeln $\mapsto_*$:*

$$\begin{array}{ll}(\beta) & (\lambda x.r)s \mapsto_\beta r_x[s] \\ (\pi) & \mathsf{D}_c(r',r,s)t \mapsto_\pi \mathsf{D}_c(r',rt,st) \\ (\delta_1) & \mathsf{D}_c(c,r,s) \mapsto_{\delta_1} r \\ (\delta_2) & \mathsf{D}_c(d,r,s) \mapsto_{\delta_2} s\end{array}$$

Es sei dann $\to$ der kompatible Abschluß von $\mapsto$, also aus $r \mapsto r'$ folgt $r \to r'$ und aus $r \to r'$ folgt auch

$$rs \to r's, sr \to sr', \lambda x.r \to \lambda x.r', \mathsf{D}_c(r,s,t) \to \mathsf{D}_c(r',s,t),$$
$$\mathsf{D}_c(s,r,t) \to \mathsf{D}_c(s,r',t), \mathsf{D}_c(s,t,r) \to \mathsf{D}_c(s,t,r').$$

Es sei dann $\to^*$ der reflexiv-transitive Abschluß von $\to$.

Definition. *Ein Term ist* in $\lambda\mathsf{D}$-Normalform*, falls keine Reduktion in ihm möglich ist. Sind r, s $\lambda\mathsf{D}$-Terme und ist s in Normalform und gilt $r \to^* s$, so heißt s* eine Normalform *von r.*

Im folgenden wollen wir nun beweisen, daß unser $\lambda\mathsf{D}$-Kalkül ein stark normalisierendes Kalkül ist. Dazu beginnen wir mit der üblichen Verträglichkeit von Reduktion und simultaner Ersetzung.

Proposition 5.2 *Seien r und s $\lambda\mathsf{D}$-Terme. Aus $r \to r'$ folgt:*

1. $s_x[r] \to^* s_x[r']$.
2. $r_x[s] \to r'_x[s]$.

Beweis.

1. Die Aussage folgt sehr schnell durch Induktion über s.
2. Beweis durch Induktion über $r \to r'$. Induktionsanfang $r \mapsto r'$.

   Fall β: $r = (\lambda y.t)t'$. Sei o. E. $y \neq x$ und $y \notin \mathsf{FV}(s)$. Dann ist $r_x[s] = (\lambda y.t_x[s])t'_x[s] \mapsto (t_x[s])_y[t'_x[s]] = (t_y[t'])_x[s] = r'_x[s]$.

   Fälle π, δ_1, δ_2. Direkt aus der Definition der simultanen Ersetzung. Der Induktionsschluß ist dann nur noch einfache Anwendung der Induktionshypothese.

■

Zum Beweis der starken Normalisierung wollen wir ähnlich vorgehen wie Joachimski und Matthes in [20]. Dazu definieren wir erst mal eine Grammatik, die nur noch normale $\lambda\mathsf{D}$-Terme enthält.

Definition. *Die Menge* nf *sei durch folgende Grammatik definiert:*

$$\mathsf{nf} \ni r, s, t, \vec{r} ::= A\vec{r} \mid \lambda x.r \mid \mathsf{D}_c(r,s,t) \ (r \notin \mathcal{C})$$

Lemma 5.1 *Die Menge* nf *enthält genau alle normalen Terme von* $\lambda\mathsf{D}$*.*

Beweis. Offensichtlich sind alle Terme in nf in Normalform. Durch einfache Induktion über r in $\lambda\mathsf{D}$ zeigt man, daß alle normalen Terme in nf sind. ■

Der nächste Schritt ist nun, den Begriff der starken Normalisierung einen induktiven Charakter zu geben wie in [2].

Definition. *Für einen* $\lambda\mathsf{D}$*-Term* r *definieren wir* $r \Downarrow$ *durch Induktion über* $\rightarrow$*:*

$$r \Downarrow \quad :\Longleftrightarrow \quad r \in \mathsf{nf} \text{ oder für alle } r' \text{ mit } r \rightarrow r' \text{ gilt } r' \Downarrow .$$

Definition. *Eine Folge von Termen* $r_1, r_2, \ldots$ *heißt* Reduktionsfolge beginnend mit r_1*, falls gilt:* $r_1 \rightarrow r_2 \rightarrow \cdots$.

Lemma 5.2 *Für einen* $\lambda\mathsf{D}$*-Term* r *gilt* $r \Downarrow$ *gdw. jede Reduktionsfolge beginnend mit* r *endlich ist.*

Beweis. „$\Rightarrow$“: Durch Induktion über $r \Downarrow$. Ist $r \in \mathsf{nf}$, so existiert keine Reduktionsfolge beginnend mit r. Sonst gilt nach Induktionsvoraussetzung für jedes r' mit $r \rightarrow r'$: Die Reduktionsfolgen beginnend mit r' sind endlich. Folglich ist jede Reduktionsfolge beginnend mit r endlich.

„$\Leftarrow$“: Durch Induktion über die Länge der längsten, möglichen Reduktionsfolge beginnend mit r. Ist diese 0, so ist $r \in \mathsf{nf}$, also $r \Downarrow$. Ist diese $n+1$, so gilt für alle r' mit $r \rightarrow r'$: Die längste, mögliche Reduktionsfolge beginnend mit r' hat eine Länge $\leq n$. Folglich nach Induktionsvoraussetzung $r' \Downarrow$. Damit nach Definition auch $r \Downarrow$. ■

Wir nennen einen $\lambda\mathsf{D}$-Term r dann „stark normalisierend“, wenn gilt: $r \Downarrow$. Für die folgende Proposition und für alles weitere wollen wir vereinbaren, daß für einen Vektor $\vec{s} = s_1, \ldots, s_n$ der Ausdruck $\vec{s} \Downarrow$ steht für: „$s_1 \Downarrow$ und $s_2 \Downarrow$ und ... und $s_n \Downarrow$“.

Proposition 5.3 *1. Aus* $r\vec{s} \Downarrow$ *folgt* $r \Downarrow$ *und* $\vec{s} \Downarrow$*.*

2. Für alle $A \in \mathcal{C} \cup \mathcal{V}$ *gilt* $A \Downarrow$*.*

3. Aus $r \Downarrow$ *folgt* $\lambda x.r \Downarrow$*.*

4. Aus $r, s, t \Downarrow$ *folgt* $\mathsf{D}_c(r, s, t) \Downarrow$*.*

5. Aus $r, s \Downarrow$ *folgt*

(i) $rs \Downarrow$.

(ii) $r_x[s] \Downarrow$.

Beweis.

1. Angenommen r oder ein s_i ist nicht stark normalisierend (o. E. sei r nicht stark normalisierend). Dann definiert die unendliche Reduktionfolge $r \to r_1 \to \ldots$ auch eine unendliche Reduktionsfolge von $r\vec{s}$, also ist $r\vec{s}$ nicht stark normalisierend.

2. Da $A \in \mathsf{nf}$ für jedes $A \in \mathcal{C} \cup \mathcal{V}$ folgt $A \Downarrow$.

3. Jede Reduktion in $\lambda x.r$ findet in r statt.

4. Eine unendliche Reduktionsfolge beginnend mit $\mathsf{D}_c(r, s, t)$ würde eine unendliche Reduktionsfolge beginnend mit r, s oder t benötigen.

5. Durch simultane Induktion über $\mathsf{ord}(\tau(s))$ beweisen wir beide Aussagen. Dabei benutzen wir eine Nebeninduktion über die maximale Länge von Reduktionsbäumen beginnend mit r. Es gelte also für alle r' mit $r \to r'$: $r's \Downarrow$ und $r'_x[s] \Downarrow$. Wir machen eine weitere Induktion über r und unterscheiden folgende Fälle. Dabei ist zu beachten, daß Teilterme von r keine längeren Reduktionsfolgen als r haben können.

   Fall $r = A\vec{r_n}$. Dann gilt nach 1 $\vec{r_n} \Downarrow$. Folglich auch $Ar_1 \cdots r_n s \Downarrow$, da jede Reduktion nur in den r_i oder in s statt findet. Also gilt (i). Für Teil (ii) gilt nach Nebeninduktionshypothese (ii) (über r) für alle $1 \leq i \leq n$: $r_i[x ::= s] \Downarrow$. Ist $A \neq x$, so sind wir fertig, da jede unendliche Reduktionsfolge beginnend mit $Ar_1[x ::= s] \cdots r_n[x ::= s]$ eine unendliche Reduktionsfolge beginnend mit einem $r_i[x ::= s]$ benötigen würde. Ist $A = x$, so folgt

   $$r_x[s] = sr_1[x ::= s] \cdots r_n[x ::= s].$$

   Für alle $1 \leq i \leq n$ ist $\mathsf{ord}(\tau(r_i[x ::= s])) < \mathsf{ord}(\tau(s))$, also gilt nach der Hauptinduktionshypothese (i) n-mal angewandt: $r_x[s] \Downarrow$.

   Fall $r = \lambda y.r'$. Für Teil (i) folgt nach Nebeninduktionshypothese (ii) (über r): $rs = r'_y[s] \Downarrow$. Für Teil (ii) folgt aus der Nebeninduktionshypothese (ii) (über r): $r'_x[s] \Downarrow$. Also wegen $r_x[s] = \lambda y.r'_x[s]$ und Teil 3 dieser Proposition auch $r_x[s] \Downarrow$.

   Fall $r = D_c(r', s', t)$ mit $r' \notin \mathcal{C}$. Hier folgen beide Aussagen unmittelbar aus der Nebeninduktionshypothese (über r) und Teil 4 dieser Proposition.

   Fall $r = (\lambda y.r_1)r_2\vec{t}$. Es gilt also $r_1, r_2, \vec{t} \Downarrow$ und auch $r_1[y ::= r_2]\vec{t} \Downarrow$. Also gilt nach Nebeninduktionshypothese (i) (über $r \Downarrow$) auch $r_1[y ::= r_2]\vec{t}s \Downarrow$ und damit folgt unmittelbar auch $rs \Downarrow$, da eine unendliche Reduktionsfolge von rs eine unendliche Reduktionsfolge von $r_1[y ::= r_2]\vec{t}s$ benötigen würde.

   Für Aussage (ii) ist

   $$r_x[s] = (\lambda y.r_1[x ::= s])r_2[x ::= s]t_1[x ::= s] \cdots t_n[x ::= s].$$

   Wir können o. E. annehmen, daß $y \notin \mathsf{FV}(s)$ ist. Nach Nebeninduktionshypothese (ii) (über r) gilt $r_1[x ::= s], r_2[x ::= s], t_1[x ::= s], \ldots, t_n[x ::= s] \Downarrow$. Also sind alle unmittelbaren Redukte abgedeckt bis auf $(r_1[x ::= s])[y ::= r_2[x ::= s]]t_1[x ::= s] \cdots t_n[x ::= s]$. Dies ist aber gleich dem Term $(r_1[y ::= r_2]\vec{t})_x[s]$, also nach Nebeninduktionshypothese (ii) (über $r \Downarrow$) gilt $r_x[s] \Downarrow$.

Fall $r = \mathsf{D}_c(c, r_1, r_2)$. Nach NIH (i) gilt $r_1 s \Downarrow$ und $r_2 s \Downarrow$. Also folgt auch $rs = \mathsf{D}_c(c, r_1, r_2)s \Downarrow$. Es gilt $r_1[x ::= s] \Downarrow$ und $r_2[x ::= s] \Downarrow$ nach NIH (ii). Folglich auch $r_x[s] = \mathsf{D}_c(c, r_1[x ::= s], r_2[x ::= s]) \Downarrow$.

Fall $r = \mathsf{D}_c(d, r_1, r_2)$ analog zu $r = \mathsf{D}_c(c, r_1, r_2)$.

Fall $r = \mathsf{D}_c(r_1, r_2, r_3)t\vec{t}$. Dann folgt (i) $rs = \mathsf{D}_c(r_1, r_2, r_3)t\vec{t}s \Downarrow$, da wir nicht ewig in s reduzieren können und damit in jedem Ast des Reduktionsbaumes irgendwann ein Schritt der Gestalt $rs' \to r's'$ mit $s \to^* s'$, also auch $s' \Downarrow$, vorkommt. Dann greift aber die Nebeninduktionshypothese (i). Für Teil (ii) ist

$$r_x[s] = \mathsf{D}_c(r_1[x ::= s], r_2[x ::= s], r_3[x ::= s])t_x[s]t_1[x ::= s] \cdots t_n[x ::= s].$$

Nach NIH (ii) (Induktion nach r) gilt

$$r_1[x ::= s], r_2[x ::= s], r_3[x ::= s], t_x[s], t_1[x ::= s], \ldots, t_n[x ::= s] \Downarrow .$$

Nach NIH (ii) (Induktion nach $r \Downarrow$) gilt auch

$$\mathsf{D}_c(r_1[x ::= s], (r_2 t)_x[s], (r_3 t)_x[s])t_1[x ::= s] \cdots t_n[x ::= s] \Downarrow .$$

Damit sind alle Redukte abgedeckt, es folgt $r_x[s] \Downarrow$.

■

Theorem 5.1 *Alle Terme in $\lambda\mathsf{D}$ sind stark normalisierend.*

Beweis. Für einen Term r in $\lambda\mathsf{D}$ zeigen wir zunächst durch Induktion über r und mit Hilfe der Proposition 5.3 $r \Downarrow$. Durch Induktion über $r \Downarrow$ zeigen wir dann, daß jede Reduktionsfolge beginnend mit r endlich ist. ■

Nun fehlt noch die Konfluenz. Dazu genügt uns wegen der starken Normalisierung, die lokale Konfluenz aller Reduktionsfolgen eines Terms nachzuweisen.

Lemma 5.3 *Gelte für einen $\lambda\mathsf{D}$-Term r: $r \to r'$ und $r \to r''$. Dann existiert ein $\lambda\mathsf{D}$-Term s mit $r' \to^* s$ und $r'' \to^* s$.*

Beweis. Wir beweisen dies durch Induktion über $r \to r'$. Der Induktionsanfang ist dann $r \mapsto r'$. Hier müssen wir jetzt natürlich die einzelnen Möglichkeiten getrennt untersuchen.

β. Also $r \equiv (\lambda x.s)t$ und $r' \equiv s_x[t]$. Es sei $r'' \equiv (\lambda x.s')t'$ mit entweder $s \to s'$ oder $t \to t'$[18]. Nach Proposition 5.2 gilt dann $r' \equiv s_x[t] \to^* s'_x[t']$ und $r'' \equiv (\lambda x.s')t' \mapsto_\beta s'_x[t']$.

π. Also ist $r \equiv D_c(r_1, r_2, r_3)t$ und $r' \equiv D_c(r_1, r_2 t, r_3 t)$. Unterfall: $r_1 \notin \mathcal{C}$. Sei $r'' \equiv D_c(r'_1, r'_2, r'_3)t'$ mit entweder $r_1 \to r'_1$ oder $r_2 \to r'_2$ oder $r_3 \to r'_3$ oder $t \to t'$. Es gilt dann offensichtlich $r' \to^* D_c(r'_1, r'_2 t', r'_3 t')$ und $r'' \mapsto_\pi D_c(r'_1, r'_2 t', r'_3 t')$.

Unterfall $r_1 = c$. Hier ist insbesondere der Fall $r'' \equiv r_2 t$ interessant. Es folgt aber unmittelbar $r' \equiv \mathsf{D}_c(c, r_2 t, r_3 t) \mapsto_{\delta_1} r_2 t$. Genauso im Unterfall $r_1 = d \neq c$.

[18] Gilt $s \to s'$, so sei $t' :\equiv t$ und umgekehrt. Ähnliches gelte auch in allen folgenden Fällen.

δ_1. Es ist also $r \equiv D_c(c,s,t)$, $r' \equiv s$ und es sei $r'' \equiv \mathsf{D}_c(c,s',t')$ mit entweder $s \to s'$ oder $t \to t'$. Dann gilt offensichtlich $r' \to^* s'$ und $r'' \mapsto_{\delta_1} s'$.

δ_2. Es ist also $r \equiv D_c(d,s,t)$, $r' \equiv t$ und es sei $r'' \equiv \mathsf{D}_c(d,s',t')$ mit entweder $s \to s'$ oder $t \to t'$. Dann gilt offensichtlich $r' \to^* t'$ und $r'' \mapsto_{\delta_2} t'$.

Der Induktionsschluß folgt dann direkt aus der Induktionshypothese. ■

Theorem 5.2 *Der Kalkül $\lambda\mathsf{D}$ ist stark normalisierend und konfluent.*

Beweis. Die starke Normalisierung haben wir in Theorem 5.1 bewiesen, die Konfluenz folgt aus der lokalen Konfluenz von Lemma 5.3. ■

Wir bezeichnen dann mit $r\downarrow$ die $\lambda\mathsf{D}$-Normalform eines $\lambda\mathsf{D}$-Terms r. Die Äquivalenzrelation $\equiv_\eta$ definieren wir durch die Regeln

$$\begin{array}{ll} r \equiv_\eta \lambda x.rx & \text{falls } r \in \mathsf{nf} \text{ und } x \notin \mathsf{FV}(r) \\ \mathsf{D}_c(x,r,s) \equiv_\eta \mathsf{D}_c(x, r_x[c]\downarrow, s) & \mathsf{D}_c(x,r,s) \in \mathsf{nf} \text{ und } x \in \mathsf{FV}(r). \end{array}$$

Es gelte $r \equiv_{\eta\alpha} s$ falls $r \equiv_\eta s$ oder $r \equiv_\alpha s$. Es ist dann $=_{\eta\alpha}$ definiert als reflexiver, symmetrischer, transitiver und kompatibler Abschluß von $\equiv_{\eta\alpha}$ über Terme in Normalform. Der kompatible Abschluß bedeutet dabei: Aus $r =_{\eta\alpha} r'$ folgt auch

$$\begin{array}{l} A\vec{s}r\vec{t} =_{\eta\alpha} A\vec{s}r'\vec{t}, \quad \lambda x.r =_{\eta\alpha} \lambda x.r', \quad \mathsf{D}_c(r,s,t) =_{\eta\alpha} \mathsf{D}_c(r',s,t), \\ \mathsf{D}_c(s,r,t) =_{\eta\alpha} \mathsf{D}_c(s,r',t), \quad \mathsf{D}_c(s,t,r) =_{\eta\alpha} \mathsf{D}_c(s,t,r'). \end{array}$$

Definition. *Für $\lambda\mathsf{D}$-Terme r und s definieren wir: $r =_{\beta\eta} s$ falls $r\downarrow =_{\eta\alpha} s\downarrow$.*

Nun müssen wir noch den Fall behandeln, daß in einer Fallunterscheidung sich die beiden Alternativen nicht wirklich unterscheiden. Wir wollen mit Sicherheit die Terme $\mathsf{D}_c(x,c,c)$ und den Term c identifizieren. Noch genauer wollen wir sagen können: $\mathsf{D}_c(x,r,s) = s$ falls $s_x[c] = r_x[c]$. Da wir aber noch keine Gleichheit für $\lambda\mathsf{D}$ haben, müssen diese wie folgt definieren.

Definition. *Wir definieren $r =^i_\mathsf{D} s$ durch Induktion über i:*

$$r =^0_\mathsf{D} s \text{ falls } r =_{\beta\eta} s.$$

Im Fall $i+1$ definieren wir zunächst die Relation $\equiv^{i+1}_\mathsf{D}$ und dann $=^{i+1}_\mathsf{D}$ als Abschluß von $\equiv^{i+1}_\mathsf{D}$ wie folgt:

$$D_c(x,r,s) \equiv^{i+1}_\mathsf{D} s \text{ falls } s_x[c] =^i_\mathsf{D} r_x[c].$$

Es sei dann $=^{i+1}_\mathsf{D}$ definiert als reflexiv, symmetrischer, transitiver und kompatibler Abschluß von $\equiv^{i+1}_\mathsf{D} \cup =^i_\mathsf{D}$. Es gilt also $r =^{i+1}_\mathsf{D} r'$ falls $r \equiv^{i+1}_\mathsf{D} r'$ oder $r =^i_\mathsf{D} r'$ und aus $r =^{i+1}_\mathsf{D} r'$ folgt auch

$$\begin{array}{l} A\vec{s}r\vec{t} =^{i+1}_\mathsf{D} A\vec{s}r'\vec{t}, \quad \lambda x.r =^{i+1}_\mathsf{D} \lambda x.r', \quad \mathsf{D}_c(r,s,t) =^{i+1}_\mathsf{D} \mathsf{D}_c(r',s,t), \\ \mathsf{D}_c(s,r,t) =^{i+1}_\mathsf{D} \mathsf{D}_c(s,r',t), \quad \mathsf{D}_c(s,t,r) =^{i+1}_\mathsf{D} \mathsf{D}_c(s,t,r'). \end{array}$$

Es gilt stets für alle r, s und i: Aus $r =^i_{\mathsf{D}} s$ folgt $r =^{i+1}_{\mathsf{D}} s$. Wir schreiben dann $r =_{\mathsf{D}} s$ falls es ein i gibt mit $r =^i_{\mathsf{D}} s$.

Beispiel.

$$\mathsf{D}_c(x,c,x) =^1_{\mathsf{D}} x$$
$$\mathsf{D}_c(x,c,\mathsf{D}_d(y,c,x)) =^2_{\mathsf{D}} \mathsf{D}_d(y,c,x)$$

Für die zweite Gleichung muß also gelten: $\mathsf{D}_d(y,c,x)_x[c] =^1_{\mathsf{D}} c$. Dies folgt aus $\mathsf{D}_d(y,c,x)_x[c] \equiv \mathsf{D}_d(y,c,c) =^1_{\mathsf{D}} c$ wegen $c_y[d] =^0_{\mathsf{D}} c$.

Die Gleichheit $=_{\mathsf{D}}$ ist leider nicht voll kompatibel zur simultanen Ersetzung. Ist etwa $t = \lambda y.y$, so ist zwar $\mathsf{D}_c(x,c,x) =_{\mathsf{D}} x$, aber

$$\mathsf{D}_c(x,c,x)_x[t] = \mathsf{D}_c(\lambda y.y, c, \lambda y.y) \neq_{\mathsf{D}} \lambda y.y.$$

Allerdings gilt für spezielle x bzw. spezielle t die Kompatibilität $r =_{\mathsf{D}} s \Rightarrow r_x[t] =_{\mathsf{D}} s_x[t]$. Einige davon wollen wir nun beschreiben.

Lemma 5.4 *Aus $r =_{\mathsf{D}} s$ folgt $r_x[A] =_{\mathsf{D}} s_x[A]$ für alle $x \in \mathcal{V}$ und $A \in \mathcal{C} \cup \mathcal{V}$.*

Beweis. Es genügt zu zeigen, daß für alle i aus $r =^i_{\mathsf{D}} s$ folgt $r_x[A] =_{\mathsf{D}} s_x[A]$. Induktion über i. IA $i = 0$. Also $r =_{\beta\eta} s$.

Wir zeigen durch Nebeninduktion über den kompatiblen Abschluß: $r =_{\beta\eta} s \Rightarrow r_x[A] =_{\beta\eta} s_x[A]$. Für den Nebeninduktionsanfang genügt, daß aus

$$\mathsf{D}_c(y,r',s') =_{\beta\eta} \mathsf{D}_c(y, r'_y[c]\downarrow, s')$$

auch

$$\mathsf{D}_c(y,r',s')_x[A] =_{\beta\eta} \mathsf{D}_c(y, r'_y[c]\downarrow, s')_x[A]$$

folgt. Wir verzichten auf den Index $\beta\eta$.

Im Fall $x \neq y$ erhalten wir

$$\begin{array}{l}\mathsf{D}_c(y,r',s')_x[A] = \\ \mathsf{D}_c(y, r'_x[A], s'_x[A]) = \\ \mathsf{D}_c(y, (r'_x[A])_y[c]\downarrow, s'_x[A]) = \\ \mathsf{D}_c(y, (r'_y[c]\downarrow)_x[A_y[c]]\downarrow, s'_x[A]) = \\ \mathsf{D}_c(y, (r'_y[c]\downarrow)_x[A], s'_x[A]) = \\ \mathsf{D}_c(y, r'_y[c]\downarrow, s')_x[A],\end{array}$$

wobei wir verwenden:

$$(r'_x[A])_y[c]\downarrow = (r'_x[A_y[c]])_y[c] = (r'_y[c])_x[A_y[c]] = (r'_y[c]\downarrow)_x[A_y[c]].$$

Dies folgt aus der Konfluenz der Reduktionen in $\lambda\mathsf{D}$ und der Verträglichkeit der Reduktionen mit simultaner Ersetzung (Proposition 5.2).

Im Fall $x = y$ und $A \in \mathcal{V}$ handelt es sich um eine ganz schlichte Umbenennung der Variable y in A, die Aussage folgt unmittelbar. Ist $x = y$ und $A \in \mathcal{C}$, so ist $\mathsf{D}_c(y,r',s')_y[A] = \mathsf{D}_c(A, r'_y[A], s'_y[A])$. Ist nun $A = c$, so folgt

$\mathsf{D}_c(A, r'_y[A], s'_y[A]) = r'_y[c]$ und dies entspricht der rechten Seite. Im Fall $A = d \neq c$ folgt $\mathsf{D}_c(A, r'_y[A], s'_y[A]) = s'_y[d] = \mathsf{D}_c(y, r'_y[c]\downarrow, s')_y[d]$.

Für die zweite Regel $r \equiv \lambda y.ry$ folgt die Aussage wie im λ-Kalkül. Der Nebeninduktionsschluß folgt direkt aus der Hypothese mittels der Konfluenz von $\lambda\mathsf{D}$ (Theorem 5.2) und Proposition 5.2.

Für den Induktionsschluß gelte also $r =_{\mathsf{D}}^{i+1} s$. Gilt dabei schon $r =_{\mathsf{D}}^{i} s$, so nutzen wir direkt die Induktionshypothese. Sonst nutzen wir eine Nebeninduktion über den kompatiblen Abschluß von $=_{\mathsf{D}}^{i+1}$ und müssen also zeigen: Gilt $\mathsf{D}_c(y, r, s) \equiv^{i+1} s$ wegen $r_y[c] =_{\mathsf{D}}^{i} s_y[c]$, so gilt entweder $\mathsf{D}_c(y, r, s)_x[A] \equiv_{\mathsf{D}}^{i+1} s_x[A]$ oder $\mathsf{D}_c(y, r, s)_x[A] =_{\mathsf{D}}^{i} s_x[A]$. Dazu unterscheiden wir folgende Fälle:

Fall $x \neq y$. Dann folgt nach Hauptinduktionsvoraussetzung

$$(r_x[A])_y[c] = (r_y[c])_x[A_y[c]] =_{\mathsf{D}}^{i} (s_y[c])_x[A_y[c]] = (s_x[A])_y[c].$$

Also auch $\mathsf{D}_c(y, r_x[A], s_x[A]) \equiv_{\mathsf{D}}^{i+1} s_x[A]$.

Fall $x = y$ und $A = z \in \mathcal{V}$. Dann ist $\mathsf{D}_c(y, r, s)_x[z] = \mathsf{D}_c(z, r_y[z], s_y[z])$ und es folgt mit Hauptinduktionshypothese:

$$(r_y[z])_z[c] = (r_y[c])_z[c] =_{\mathsf{D}}^{i} (s_y[c])_z[c] = (s_y[z])_z[c].$$

Also auch $\mathsf{D}_c(z, r_y[z], s_y[z]) \equiv_{\mathsf{D}}^{i+1} s_y[z]$.

Fall $x = y$ und $A = c$. Dann ist $\mathsf{D}_c(y, r, s)_x[A] = \mathsf{D}_c(c, r_y[c], s_y[c]) =_{\mathsf{D}}^{0} r_y[c] =_{\mathsf{D}}^{i} s_y[c]$.

Fall $x = y$ und $A = d \in \mathcal{C}$. Dann ist $\mathsf{D}_c(y, r, s)_x[A] = \mathsf{D}_c(d, r_y[d], s_y[d]) =_{\mathsf{D}}^{0} s_y[d]$.

Der Nebeninduktionsschluß folgt nun unmittelbar aus der Nebeninduktionshypothese. ■

Definition. *Ein Term $r \in \mathsf{nf}$ heißt* gut*, wenn für jeden Teilterm $\mathsf{D}_c(s, t, t')$ von r gilt: $s \in \mathcal{V}$. Ein Term r heißt* gut*, wenn $r\downarrow$ gut ist.*

Um zu überprüfen, ob ein Term $r_x[s]$ gut ist, hilft der folgende Begriff der D-*Variablen* eines Terms r, $\mathsf{DV}(r)$:

$$\mathsf{DV}(r) := \{\, x \in \mathsf{FV}(r) \mid r\downarrow \text{ besitzt einen Teilterm der Gestalt } \mathsf{D}_c(x, r', r'') \,\}.$$

Es ist leicht einzusehen, daß für eine guten Term r und einen normalen Term s der Term $r_x[s]$ genau dann gut ist, wenn gilt: $x \in \mathsf{DV}(r) \Rightarrow s \in \mathcal{C} \cup \mathcal{V}$.

Lemma 5.5 *Seien r und s gute Terme mit $r =_{\mathsf{D}} s$. Sei $x \in \mathcal{V}$ und t ein Term, so daß $r_x[t]$ und $s_x[t]$ gut sind. Dann gilt auch $r_x[t] =_{\mathsf{D}} s_x[t]$.*

Beweis. Ist $t = A \in \mathcal{C} \cup \mathcal{V}$, so siehe das vorherige Lemma 5.4. Sei also t ungleich jeder Variablen oder Konstanten. Wir zeigen, daß für alle i gilt: Aus $r =_{\mathsf{D}}^{i} s$ folgt $r_x[t] =_{\mathsf{D}}^{i} s_x[t]$. Induktion über i. IA $i = 0$. Also $r =_{\beta\eta} s$.

Wir zeigen durch Nebeninduktion über den kompatiblen Abschluß: $r =_{\beta\eta} s \Rightarrow r_x[t] =_{\beta\eta} s_x[t]$. Für den Nebeninduktionsanfang genügt, daß aus $\mathsf{D}_c(y, r', s') =_{\beta\eta} \mathsf{D}_c(y, r'_y[c]\downarrow, s')$ auch $\mathsf{D}_c(y, r', s')_x[t] =_{\beta\eta} \mathsf{D}_c(y, r'_y[c]\downarrow, s')_x[t]$ folgt. Nach obiger Bemerkung muß $x \neq y$ sein. Wir verzichten auf den Index $\beta\eta$.

$$\begin{array}{l}\mathsf{D}_c(y, r', s')_x[t] = \\ \mathsf{D}_c(y, r'_x[t], s'_x[t]) = \\ \mathsf{D}_c(y, (r'_x[t])_y[c]\downarrow, s'_x[t]) = \\ \mathsf{D}_c(y, (r'_y[c]\downarrow)_x[t_y[c]]\downarrow, s'_x[t]) = \\ \mathsf{D}_c(y, (r'_y[c]\downarrow)_x[t], s'_x[t]) = \\ \mathsf{D}_c(y, r'_y[c]\downarrow, s')_x[t],\end{array}$$

wobei wir verwenden:

$$(r'_x[t])_y[c]\downarrow = (r'_x[t])_y[c] = (r'_y[c])_x[t_y[c]] = (r'_y[c]\downarrow)_x[t_y[c]].$$

Dies folgt aus der Konfluenz der Reduktionen in $\lambda\mathsf{D}$ und der Verträglichkeit der Reduktionen mit simultaner Ersetzung (Proposition 5.2).

Induktionsschluß. Nebeninduktion über den kompatiblen Abschluß $=_{\mathsf{D}}^{i+1}$. Nebeninduktionsanfang: Zu zeigen ist: Aus $\mathsf{D}_c(y, r, s) \equiv_{\mathsf{D}}^{i+1} s$ wegen $r_y[c] =_{\mathsf{D}}^{i} s_y[c]$ folgt $\mathsf{D}_c(y, r, s)_x[t] \equiv^{i+1} s_x[t]$. Da nach Voraussetzung $\mathsf{D}_c(y, r, s)_x[t]$ ein guter Term ist, folgt $x \neq y$. Außerdem müssen dann auch $r_x[t]$ und $s_x[t]$ gute Terme sein. Folglich sind auch $(r_x[t])_y[c] = (r_y[c])_x[t_y[c]]$ und $(s_x[t])_y[c] = (s_y[c])_x[t_y[c]]$ gute Terme, also folgt nach HIH $(r_y[c])_x[t_y[c]] =_{\mathsf{D}}^{i} (s_y[c])_x[t_y[c]]$. Damit nach Definition auch $\mathsf{D}_c(y, r, s)_x[t] = \mathsf{D}_c(y, r_x[t], s_x[t]) =_{\mathsf{D}}^{i+1} s_x[t]$. Nebeninduktionsschluß direkt aus Nebeninduktionshypothese. ∎

Im folgenden soll nun $r = s$ immer $r =_{\mathsf{D}} s$ bedeuten. Zu einem normalen Term r finden wir immer einen normalen Term s mit $r = s$ und folgenden Eigenschaften:

1. Für jeden Teilterm der Gestalt $A\vec{r}$ von s gilt $\tau(A\vec{r})$ ist Grundtyp.
2. Für jeden Teilterm der Gestalt $\mathsf{D}_c(x, r', s')$ von s gilt $x \notin \mathsf{FV}(r')$ und $s'_x[c] \neq r'_x[c]$.

Ein solches s nennen wir dann *in langer Normalform* bzw. eine *lange Normalform von* r. Im Folgenden wollen wir, falls nicht anders notiert, immer annehmen, daß die vorkommenden Terme in langer Normalform sind.

5.2 Unentscheidbarkeit der Unifikation mit Fallunterscheidung

Wir wollen in diesem Kapitel die Frage untersuchen, ob durch das Hinzufügen der Fallunterscheidung die Unifikation entscheidbar wird, d. h. ob in einem Kalkül mit Fallunterscheidung ein vollständiger, immer terminierender Algorithmus existiert, der, falls vorhanden, stets eine vollständige Menge von Unifikatoren berechnet. Wie die Überschrift bereits suggeriert, ist dies nicht der Fall.

Um dies zu zeigen, benutzen wir die Frage der λ-Definierbarkeit. Die Entscheidbarkeit dieser Frage, welche auch *Plotkin-Statman- Vermutung* genannt wird, wurde erst kürzlich von R. Loader [22] widerlegt. Wir zeigen im folgenden, daß aus der Existenz endlicher, vollständiger Mengen von Unifikatoren für jedes lösbare Unifikationsproblem in einem sehr speziellen Kalkül mit Fallunterscheidung die Plotkin-Statman-Vermutung folgt.

Definition. *Der $\lambda\delta$-Kalkül entsteht aus dem $\lambda\mathsf{D}$-Kalkül indem wir die letzte Termregel wie folgt abändern:*

Ist $c \in \mathcal{C}_\alpha$, r ein Term vom Typ α und sind $d_1, d_2 \in \mathcal{C}_\beta$, so ist $D_c(r, d_1, d_2)$ ein $\lambda\delta$-term vom Typ β.

Der $\lambda\delta$-Kalkül unterscheidet sich vom $\lambda\mathsf{D}$-Kalkül dadurch, daß in beiden Alternativen einer Fallunterscheidung nur Konstanten stehen dürfen.

Nun wollen wir genau definieren, was die Plotkin-Statman Vermutung ist. Dazu sei uns ein endliches Standardmodell $\mathcal{M}$ gegeben und eine Interpretation, die jedem λ-Term r einen Wert $[|r|]$ in $\mathcal{M}$ zuordnet. Das λ-Definitionsproblem kann man dann wie folgt definieren:

Existiert für jede Funktion $a \in \mathcal{M}$ vom Typ α ein geschlossener λ-Term r mit $[|r|] = a$?

Die Entscheidbarkeit dieser Frage, also die Behauptung ein Algorithmus würde existieren, der uns jenen Term r passend zu einer Funktion a berechnet, nennt man dann die *Plotkin-Statman Vermutung.*

Wer es genauer wissen möchte, den verweise ich auf die Arbeiten von Plotkin [30],[31] und Statman [35]. Empfehlenswert sind auch die Arbeiten von Barendregt [5] und Wolfram [37].

Die Widerlegung dieser Vermutung gelang R. Loader in [22] dadurch, daß er die Äquivalenz der Plotkin-Statman Vermutung zu einem unentscheidbaren Wortproblem zeigte.

Theorem 5.3 *Die Plotkin-Statman Vermutung ist falsch.*

Dabei kann man ein Wortproblem so beschreiben: Sei eine formale Grammatik und seien zwei Wörter einer Sprache gegeben. Ist das eine Wort nun aus dem anderen mittels der Grammatik ableitbar oder nicht ? Die Unentscheidbarkeit solcher Wortprobleme wurde von M. Davies in [12] gezeigt. Zur Vollständigkeit sei noch erwähnt, daß auch entscheidbare Wortprobleme existieren, eine kleine Übersicht findet man z. B. in dem Buch von Becker und Weisspfennig [7], die auch eine sehr formale, algebraische Charakterisierung dieser Probleme geben.

Definition. *Seien $r^{\alpha\to\iota}$ und t^ι zwei geschlossene $\lambda\delta$-Terme. Die Frage: „Finde ich einen geschlossenen λ-Term s^α mit $rs = t$?“ heißt* Anpassungsproblem *im $\lambda\delta$-Kalkül.*

Man beachte: Gesucht wird tatsächlich ein Term s im einfach getypten λ-Kalkül, kein $\lambda\delta$-Term. Die Gleichheit in $rs = t$ bezieht sich dabei natürlich auf die Gleichheit im $\lambda\delta$-Kalkül.

Theorem 5.4 *Die Plotkin-Statman Vermutung ist äquivalent zur Entscheidbarkeit des Anpassungsproblems im λδ-Kalkül.*

Einen Beweis findet man z. B. im Buch von Wolfram [37]. Die für uns wichtige Richtung, daß das Anpassungsproblem die λ-Definierbarkeit impliziert, folgt aus der Tatsache, daß jede Funktion in einem endlichen Standardmodell die Interpretation eines λδ-Terms ist. Wollen wir also zu einer Funktion g vom Typ α einen entsprechenden λ-Term s mit $[|s|] = g$ finden, so definieren wir zwei Funktionen f und h vom Typ $\alpha \to \iota$ bzw. ι mit $f(x) = h$ gdw. $x = g$. Dann exisitieren λδ-Terme r und t, deren Interpretation f bzw. h ist. Also erhalten wir s aus der Entscheidbarkeit des Anpassungsproblems für λδ angewandt auf r und t.

Lemma 5.6 *Ist im λδ- Kalkül die Unifikation entscheidbar und existiert für jedes lösbare Unifikationsproblem eine endliche, vollständige Menge von Unifikatoren, so ist auch das Anpassungsproblem in λδ entscheidbar.*

Beweis. Dazu seien $r^{\alpha\to\iota}$ und t^{ι} geschlossene λδ-Terme. Dann entspricht das Anpassungsproblem dem Unifikationsproblem $(\{(rx, t)\}, \emptyset, \{x\})$. Somit können wir die Suche nach einem Term s mit $rs = t$ gleich setzen mit dem Versuch, rx und t zu unifizieren. Nach Voraussetzung bekommen wir dafür eine endliche, vollständige Menge von Unifikatoren $\vartheta_1, \ldots, \vartheta_n$. Ist diese Menge leer, also rx und t nicht unifizierbar, so ist das Anpassungsproblem negativ entschieden. Sonst gelte $\vartheta_i(x) = s_i$. Da für alle $1 \le i \le n : rs_i = t$ und t ein geschlossener Term ist, muß auch $\mathsf{FV}(rs_i) = \emptyset$ sein. Folglich spielen Instanziierungen freier Variablen in s_i keine Rolle, es folgt: Für jede Substitution σ ist auch $\sigma \circ \vartheta_i$ ein Unifikator.[19]

Das Anpassungsproblem reduziert sich nun auf die Frage, ob es ein $1 \le i \le n$ und eine Substitution σ gibt, so daß $\sigma(s_i) \downarrow$ ein λ-Term ist, also keine Fallunterscheidungen mehr besitzt. Ist bereits ein $s_i \downarrow$ ein λ-Term, so ist das Anpassungsproblem positiv entschieden. Sonst seien $s_1 \downarrow, \ldots, s_n \downarrow$ keine λ-Terme, jedes enthalte also noch einen Teilterm der Gestalt $\mathsf{D}_c(y, d_1, d_2)$ mit $d_1 \neq d_2$. Die Frage ist nun, wann wir diesen Teilterm durch eine Substitution entfernen können.

Sei dazu nun konkret $s_i = s_i \downarrow$ gegeben und $t' = \mathsf{D}_c(y, d_1, d_2)$ Teilterm von s_i. Ist $y \in \mathsf{FV}(s_i)$, so setzen wir $\sigma(y) := c$. Damit verschwindet jede Fallunterscheidung, in der y an entsprechender Stelle steht. Ist allerdings $y \notin \mathsf{FV}(s_i)$, so bleibt noch die Hoffnung, daß ein Teilterm $fs'_1 \cdots s'_n$ von s_i existiert mit $f \in \mathsf{FV}(s_i)$ und t' Teilterm eines s'_j. Dann können wir ganz t' entfernen, indem wir setzen: $\sigma(f) := \lambda\vec{x_n}.z$, wobei z eine neue Variable passenden Typs ist. Dieses Vorgehen ist etwas brutal, aber wir sind hier ja nicht an besonders allgemeinen oder eleganten Lösungen interessiert. Kommt t' allerdings nicht in einer solchen Situation in s_i vor, und ist $y \notin \mathsf{FV}(s_i)$, so wird t' auch Teilterm von $\sigma(s_i) \downarrow$ bleiben für alle Substitutionen σ.

Finden wir in jedem $s_1 \downarrow, \ldots, s_n \downarrow$ eine Fallunterscheidung, die nicht entfernt werden kann, so ist das Anpassungsproblem negativ entschieden. Sonst sei σ und s_j gefunden mit: $\sigma(s_j) \downarrow$ ist ein λ-Term. Dann ist das Anpassungsproblem positiv entschieden mit $s = \sigma(s_j) \downarrow$. ■

[19] In λD folgt nicht automatisch aus $r = s$ auch $\sigma(r) = \sigma(s)$.

Theorem 5.5 *Im $\lambda\delta$-Kalkül muß ein Unifikationsproblem existieren, das keine endliche, vollständige Menge von Unifikatoren besitzt oder dessen Unlösbarkeit nicht in endlicher Zeit entschieden werden kann.*

Beweis. Nach Theorem 5.3 ist die Plotkin-Statman Vermutung falsch. Damit folgt wegen Theorem 5.4, daß das Anpassungsproblem in $\lambda\delta$ unentscheidbar ist. Also muß nach Lemma 5.6 mindestens ein Unifikationsproblem in $\lambda\delta$ existieren, das keine endliche, vollständige Menge von Unifikatoren besitzt oder dessen Unlösbarkeit nicht in endlicher Zeit entschieden werden kann. ■

Wir haben die obigen Überlegungen für den $\lambda\delta$-Kalkül durchgeführt. Unschwer einzusehen ist, daß dieser Beweis für jede Erweiterung des $\lambda\delta$-Kalküls möglich ist, insbesondere auch für das λD-Kalkül. Da jedes Kalkül mit sinnvoller Fallunterscheidung wohl nicht darum herum kommt, eine solche Erweiterung zu sein, kann man folgern, daß also die Einführung von Fallunterscheidungen nicht zur Entscheidbarkeit des Unifikationsproblems beträgt.

Damit kurz zu der Arbeit von Beeson [8]. Dort behauptet er, einen terminierenden Algorithmus für die Unifikation in einem Kalkül mit Fallunterscheidung gefunden zu haben. Sein Fehler liegt in der Tatsache, daß sein Algorithmus nicht vollständig ist, da er keine Berechnungsschritte für Terme, die bereits eine Fallunterscheidung beinhalten angibt. Dies ist aber notwendig, so bald die Unifikationsprobleme nicht mehr linear sind.

Sei z. B. $\mathsf{UP} = (\{(f0, 0), (fx, 0)\}, \{x\}, \{f\})$. Für das erste Paar erhalten wir als Unifikator $\langle f, \lambda x.D_0(x, 0, f'x)\rangle$ und damit das neue Unifikationsproblem $\mathsf{UP}' = (\{(D_0(x, 0, f'x), 0)\}, \{x\}, \{f'\})$. Wir können nun den Konstruktor D nicht wie eine beliebige Konstante behandeln (wie in [8]) und folglich behaupten, daß dieses Unifikationsproblem unlösbar ist. Erkennen wir doch rasch, daß als Lösung $\langle f', \lambda x.0\rangle$ in Frage kommt.

Also müßten Prozeduren definiert werden, die Unifikationspaare welche Fallunterscheidungen beinhalten, lösen. Dies kann allerdings, wie in diesem Abschnitt gezeigt, nicht zur Glückseligkeit eines terminierenden, vollständigen und korrekten Unifikationsalgorithmus führen. Nicht einmal in so harmlosen Kalkülen wie dem λD- bzw. $\lambda\delta$-Kalkül. Insofern wollen wir uns zunächst auf lineare Unifikationsprobleme konzentrieren.

5.3 Muster mit konstanten Argumenten - Linearer Fall

Nun wollen wir Unifikation mit Fallunterscheidung an einem konkreten Beispiel durchführen. Wir definieren dazu Terme, die sich von den Mustern dadurch unterscheiden, daß flexible Variablen auch Konstanten als Argumente bekommen können. Wie im vorherigen Abschnitt bereits angedeutet, beschränken wir uns zunächst auf lineare Unifikationsprobleme, also solche Probleme, bei denen jede flexible Variable höchstens ein freies Vorkommen hat. Wir werden im Folgenden einen terminierenden, korrekten und vollständigen Unifikationsalgorithmus angeben.

Definition. *Ein λ-Term r heißt* $\mathcal{C}$-Muster *bzgl.* flex *und* forb*, falls für alle Teilterme $\lambda\vec{y_m}.As_1 \cdots s_n$ mit $A \in \mathsf{flex} \cap \mathsf{FV}(r)$ gilt:*

1. *Für alle* $1 \leq i \leq n$*:* $s_i = A_i \in \mathcal{C} \cup \mathsf{forb} \cup \mathsf{BV}(r)$.
2. *Für alle* $1 \leq i \neq j \leq n$*:* $A_i \in \mathcal{V}$ *impliziert* $A_i \neq A_j$.

Für verschachtelte Fallunterscheidungen der Gestalt

$$\mathsf{D}_{c_1}(r_1, (\mathsf{D}_{c_2}(r_2, \ldots (\mathsf{D}_{c_k}(r_k, r, s_k) \ldots), s_2), s_1)$$

schreiben wir kurz $\mathsf{D}_{\vec{c_k}}(\vec{r_k}, r, \vec{s_k})$.

Nun wollen wir die einzelnen Schritte, die zur Unifikation solcher $\mathcal{C}$-Muster nötig sind, einzeln vorstellen. Wir gehen dabei von einem linearen Unifikationsproblem aus. Der nicht-lineare Fall folgt im nächsten Abschnitt. Das Vorgehen ähnelt dabei sehr dem der Musterunifikation. Nur daß wir jetzt Fallunterscheidungen benutzen, falls wir auf konstante Argumente flexibler Variablen treffen.

Zunächst kann es ja passieren, daß die Argumente bunt gemischt vorkommen, also Variabeln und Konstanten in beliebiger Reihenfolge. Solche Unordnung wollen wir nicht dulden und sortieren deshalb immer die Argumente so, daß erst die Konstanten kommen, dann die Variablen. Also statt $fxcy$ wollen wir $gcxy$ haben. Dies geht mittels einer Substitution $\langle f, \lambda x, v, y.gvxy \rangle$. Man beachte, daß wir dabei die Lösbarkeit und die Lösungen nicht ernsthaft beeinträchtigen, läßt sich obige Bindung doch mittels $\langle g, \lambda v, x, y.fxvy \rangle$ wieder rückgängig machen.

Prozedur 5.1 Sortieren der Argumente $\mathsf{sort}(r, \mathsf{forb}, \mathsf{flex})$.

Sei $r = fA_1 \cdots A_n$ *ein* $\mathcal{C}$*-Muster,* $f \in \mathsf{flex}$*,* $\{A_{i_1}, \ldots, A_{i_k}\} = \{A_1, \ldots, A_n\} \cap \mathcal{C}$ *und* $\{A_{j_1}, \ldots, A_{j_l}\} = \{A_1, \ldots, A_n\} \cap \mathcal{V}$*. Seien* $x_1, \ldots, x_n \in \mathcal{V}$ *mit* $\tau(x_i) = \tau(A_i)$ *für alle* $1 \leq i \leq n$*. Dann ist*

$$\mathsf{sort}(r, \mathsf{forb}, \mathsf{flex}) := \{\langle f, \lambda \vec{x_n}.gx_{i_1} \cdots x_{i_k} x_{j_1} \cdots x_{j_l} \rangle\}$$

mit einer neuen Variable g *passenden Typs.*

Wir brauchen erneut eine Prüfung, ob eine flexible Variable f in einem Term r frei vorkommt.

Prozedur 5.2 Vorkommensüberprüfung $\mathsf{oc}(f, r)$.

$$\mathsf{oc}(f, r) := \begin{cases} \mathsf{wahr} & \textit{falls } f \in \mathsf{FV}(r) \\ \mathsf{falsch} & \textit{falls } f \notin \mathsf{FV}(r) \end{cases}$$

Auch kann es wieder nötig werden, störende verbotene Variablen zu entfernen. Dies geht ganz analog zu den Mustern.

Prozedur 5.3 Pruning $\mathsf{Prune}(x, r, \mathsf{forb}, \mathsf{flex})$.

Gelte $x \in \mathsf{FV}(r)$*. Sei*

$$M = \{\, s \textit{ Teilterm von } r \mid x \in \mathsf{FV}(s) \textit{ und } \mathsf{hd}(s) \in \mathsf{flex} \,\}.$$

Ist $M = \emptyset$*, so sei* $\mathsf{Prune}(x, r, \mathsf{forb}, \mathsf{flex}) := \dagger$.

Sonst sei $s = fA_1 \cdots A_k x A_{k+1} \cdots A_l \in M$. *Seien* $v_1, \ldots, v_l$ *Variablen mit* $\tau(v_i) = \tau(A_i)$. *Sei* g *eine neue Variable mit* $\tau(g\vec{v_l}) = \tau(s)$. *Dann ist*

$$\mathsf{Prune}(x, r, \mathsf{forb}, \mathsf{flex}) := \{\langle f, \lambda v_1, \ldots, v_k, x, v_{k+1}, \ldots, v_l . g\vec{v_l}\rangle\}.$$

Jetzt wird es interessant. Einen flexiblen Term $f\vec{c}\vec{x}$ und einen starren Term s unifizieren wir, indem wir für f einen Term einsetzen, der grob gesagt bedeutet: Sind die ersten Argumente gleich den Konstanten $\vec{c}$ und die restlichen Argumente beliebig, so muß der Term zu s konvergieren.

Prozedur 5.4 Flexibel-Starr $\mathsf{fs}(r, s, \mathsf{forb}, \mathsf{flex})$.

Gegeben seien zwei $\mathcal{C}$*-Muster* $r = fc_1 \cdots c_k x_1 \cdots x_l$ *und* s *bzgl.* forb *und* flex *mit* $f \in \mathsf{flex}$ *und* s *starr.*

Seien $y_1, \ldots, y_k$ *Variablen mit* $\tau(y_i) = \tau(c_i)$ *und* $f_1, \ldots, f_k$ *neue Variablen gleichen Typs wie* f. *Gelte* $\mathsf{forb} \cap \mathsf{FV}(s) \subseteq \{x_1, \ldots, x_l\}$. *Mit* $r_i := f_i \vec{y_k} \vec{x_l}$ *definieren wir*

$$\mathsf{fs}(r, s, \mathsf{forb}, \mathsf{flex}) := \begin{cases} \{\langle f, \lambda \vec{y_k}, \vec{x_l} . \mathsf{D}_{\vec{c_k}}(\vec{y_k}, s, \vec{r_k})\rangle\} & k > 0 \\ \{\langle f, \lambda \vec{x_l} . s\rangle\} & k = 0 \end{cases}$$

Und schließlich zu den flexiblen. Dabei können nicht zwei gleiche Köpfe auftreten. Wollen wir z. B. die Terme $f25xy$ und $g4y$ unifizieren, wobei die Zahlen als Konstante gelesen werden sollen, so ersetzten wir f durch eine Funktion der Gestalt

$$u, v, x, y \mapsto \mathsf{Wenn}\, u = 2, \mathsf{dann}\, (\mathsf{Wenn}\, v = 5, \mathsf{dann}\, hy, \mathsf{sonst}\, f'uvxy), \mathsf{sonst}\, f''uvxy$$

und g durch eine Funktion der Gestalt

$$w, y \mapsto \mathsf{Wenn}\, w = 4, \mathsf{dann}\, hy, \mathsf{sonst}\, g'wy.$$

Eingesetzt reduziert beides zu hy.

Prozedur 5.5 Flexibel≠Flexibel $\mathsf{fg}(r, s, \mathsf{forb}, \mathsf{flex})$.

Gegeben seien zwei $\mathcal{C}$*-Muster* $r = fc_1 \cdots c_k x_1 \cdots x_l$ *und* $s = gd_1 \cdots d_m y_1 \cdots y_n$ *bzgl.* forb *und* flex *mit* $f, g \in \mathsf{flex}$. *Es sei* $\{\vec{z}\} = \{\vec{x_l}\} \cap \{\vec{y_n}\}$.

Im Fall $k + m > 0$ *seien* $f_1, \ldots, f_k$ *neue Variablen gleichen Typs wie* f *und* $g_1, \ldots, g_m$ *neue Variablen gleichen Typs wie* g, *sowie* h *eine neue Variable mit* $\tau(h\vec{z}) = \tau(r)$. *Wir definieren dann*

$$\mathsf{fg}(r, s, \mathsf{forb}, \mathsf{flex}) := \{\langle f, \lambda \vec{v_k} x_l . \mathsf{D}_{\vec{c_k}}(\vec{v_k}, h\vec{z}, \vec{r_k})\rangle, \langle g, \lambda \vec{w_m} y_n . \mathsf{D}_{\vec{d_m}}(\vec{w_m}, h\vec{z}, \vec{s_m})\rangle\}$$

mit $r_i := f_i \vec{v_k} \vec{x_l}$ *und* $s_i := g_i \vec{w_m} \vec{y_n}$.

Im Fall $k + m = 0$ *setzen wir wie bei Mustern*

$$\mathsf{fg}(r, s, \mathsf{forb}, \mathsf{flex}) := \{\langle f, \lambda \vec{x_l} . h\vec{z}\rangle, \langle g, \lambda \vec{y_n} . h\vec{z}\rangle\}.$$

Nun setzen wir die obigen Prozeduren einfach noch zusammen und erhalten unseren Unifikationsalgorithmus.

Algorithmus 5 *Sei* $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein lineares* $\mathcal{C}$*-Muster Unifikationsproblem. Sei* $\theta := \varepsilon$ *und* $V := \mathsf{flex}$*.*

1. *Ersetze* UP *durch* $(\mathrm{U} \setminus \{\, (r,s) \in \mathrm{U} \mid r = s \,\}, \mathsf{forb}, \mathsf{flex})$*.*
2. *Ersetze* UP *durch* $\mathsf{SIMPL}(\mathsf{UP})$*.*
3. *Ist* $\mathsf{UP} = \bot$*, so gib* $\bot$ *aus und beende, ist* $\mathrm{U} = \emptyset$*, so gib* $\theta\,|_V$ *aus und beende.*
4. *Wähle ein Paar* $(r,s) \in \mathrm{U}$ *und unterscheide folgende Fälle:*
   (a) *Fall* r *flexibel und* s *starr:*
      (i) *Ist* $\mathsf{oc}(\mathsf{hd}(r), s) = \mathsf{wahr}$*, so setzte* $\mathsf{UP} := \bot$ *und gehe zu 3.*
      (ii) *Ist* r *noch nicht sortiert und* $\vartheta = \mathsf{sort}(r, \mathsf{flex})$*, so ersetze* UP *durch* $\vartheta(\mathsf{UP})$ *und* θ *durch* $\vartheta \circ \theta$ *und gehe zu 1.*
      (iii) *Sei* $r = f\vec{c}\vec{x}$*. Existiert ein* $y \in (\mathsf{forb} \cap \mathsf{FV}(s)) \setminus \{\vec{x}\}$*, so rufe* $\mathsf{Prune}(y, s, \mathsf{forb}, \mathsf{flex})$ *auf. Ist* $\mathsf{Prune}(y, s, \mathsf{forb}, \mathsf{flex}) = \dagger$*, so setze* $\mathsf{UP} := \bot$ *und gehe zu 3. Sonst sei* $\vartheta = \mathsf{Prune}(y, s, \mathsf{forb}, \mathsf{flex})$*. Ersetze* UP *durch* $\vartheta(\mathsf{UP})$ *und* θ *durch* $\vartheta \circ \theta$ *und gehe zu 1.*
      (iv) *Sei* $r = f\vec{x}\vec{x}$ *und gelte* $\mathsf{oc}(\mathsf{hd}(r), s) = \mathsf{falsch}$ *sowie* $\mathsf{forb} \cap \mathsf{FV}(s) \subseteq \{\vec{x}\}$*. Dann sei* $\vartheta = \mathsf{fs}(r, s, \mathsf{forb}, \mathsf{flex})$*. Ersetze* UP *durch* $\vartheta(\mathsf{UP})$ *und* θ *durch* $\vartheta \circ \theta$ *und gehe zu 1.*
   (b) *Fall* r *und* s *flexibel.*
      (i) *Ist* r *oder* s *noch unsortiert, so sei* $\vartheta := \mathsf{sort}(r, \mathsf{flex}) \circ \mathsf{sort}(s, \mathsf{flex})$*. Ersetze* θ *durch* $\vartheta \circ \theta$ *und* UP *durch* $\vartheta(\mathsf{UP})$ *und gehe zu 1.*
      (ii) *Sind* r *und* s *sortiert, so sei* $\vartheta = \mathsf{fg}(r, s, \mathsf{forb}, \mathsf{flex})$*. Ersetze* θ *durch* $\vartheta \circ \theta$ *und* UP *durch* $\vartheta(\mathsf{UP})$ *und gehe zu 1.*

Wir können also die Anwendung unseres Algorithmus als Folge von Arbeitsschritten in der Form

$$\mathsf{UP}_0 \Longrightarrow_{\vartheta_1} \mathsf{UP}_1 \Longrightarrow_{\vartheta_2} \ldots \Longrightarrow_{\vartheta_n} \mathsf{UP}_n \Longrightarrow_{\vartheta_{n+1}} \cdots$$

schreiben. Dabei soll $\mathsf{UP}_i = \mathsf{SIMPL}(\vartheta_i(\mathsf{UP}_{i-1}))$ sein. Wir beweisen nun im Folgenden die zu der Musterunifikation analogen Aussagen über Terminierung, Korrektheit und Vollständigkeit.

Lemma 5.7 *Sei* $\mathsf{UP} \Longrightarrow_\vartheta \mathsf{UP}'$ *ein Arbeitsschritt des Algorithmus 5. Ist* UP *ein lineares* $\mathcal{C}$*-Muster Unifikationsproblem so auch* UP'*.*

Beweis. Dazu betrachte man einfach die berechneten Substitutionen und beachte, daß im Falle der Einführung einer Fallunterscheidung in den Prozeduren „Flexibel-Starr“ bzw. „Flexibel ≠ Flexible“ die Variablen im Domain sonst niegends auftreten. ∎

Lemma 5.8 *Der Algorithmus 5 terminiert.*

Beweis. Wir verwenden das Maß aus dem Lemma 3.3. Dieses wird bei der Anwendung von SIMPL auf UP echt kleiner. Aber auch die anderen Schritte verringern es, da ein Unifikationspaar gestrichen wird und die anderen unverändert bleiben. ■

Lemma 5.9 *Sei* $\mathsf{UP} \Longrightarrow_\vartheta \mathsf{UP}'$ *ein Arbeitsschritt des Algorithmus 5. Sei* θ *ein Unifikator von* UP'*. Dann ist* $\theta \circ \vartheta$ *ein Unifikator von* UP*.*

Beweis. Wir untersuchen dazu die Fälle, in denen sich unser Algorithmus von der Muster Unifikation unterscheidet, also Konstanten als Argumente flexibler Variablen auftreten. Ist dies der Fall, so beachten wir die Linearitätsbedingung und stellen erfreut fest, daß jene flexible Variable dann nur an genau einer Stelle auftritt. Es sei an das Lemma 3.9 erinnert.

Fall „Flexibel-Starr“: Sei $(r, s) \in \mathrm{U}$ mit $r = fc_1 \cdots c_k x_1 \cdots x_l$ mit $f \in \mathsf{flex}$, $k > 0$ und s starr. Dann tritt f nirgends sonst mehr in U frei auf. Gelte $\mathsf{forb} \cap \mathsf{FV}(s) \subseteq \{x_1, \ldots, x_l\}$. Seien $y_1, \ldots, y_k$ Variablen mit $\tau(y_i) = \tau(c_i)$ und $f_1, \ldots, f_k$ neue Variablen gleichen Typs wie f. Mit $r_i := f_i \vec{y_k} \vec{x_l}$ für alle $1 \leq i \leq k$ haben wir

$$\vartheta := \{\langle f, \lambda \vec{y_k}, \vec{x_l}.\mathsf{D}_{\vec{c_k}}(\vec{y_k}, s, \vec{r_k})\rangle\}.$$

Es ist also $\vartheta(f\vec{c}\vec{x}) = \lambda\vec{x}.s\vec{x} = s$. Wegen $\mathsf{UP}' = (\mathrm{U} \setminus \{(r, s)\}, \mathsf{forb}, \mathsf{flex}')$ ist $\theta \circ \vartheta$ ein Unifikator von UP.

Fall „Flexibel≠Flexibel“. Sei $(r, s) \in \mathrm{U}$ mit

$$r = fc_1 \cdots c_k x_1 \cdots x_l, \quad s = gd_1 \cdots d_m y_1 \cdots y_n,$$

$k + m > 0$ und $f, g \in \mathsf{flex}$. Es sei $\{\vec{z}\} = \{\vec{x_l}\} \cap \{\vec{y_n}\}$. Es seien $f_1, \ldots, f_k$ neue Variablen gleichen Typs wie f und $g_1, \ldots, g_m$ neue Variablen gleichen Typs wie g, sowie h eine neue Variable mit $\tau(h\vec{z}) = \tau(r)$. Es ist dann

$$\vartheta := \{\langle f, \lambda \vec{v_k} x_l.\mathsf{D}_{\vec{c_k}}(\vec{v_k}, h\vec{z}, \vec{r_k})\rangle, \langle g, \lambda \vec{w_m} y_n.\mathsf{D}_{\vec{d_m}}(\vec{w_m}, h\vec{z}, \vec{s_m})\rangle\}$$

mit $r_i := f_i \vec{v_k} \vec{x_l}$ für alle $1 \leq i \leq k$ und $s_j := g_j \vec{w_m} \vec{y_n}$ für alle $1 \leq j \leq m$. Es folgt $\vartheta(r) = h\vec{z} = \vartheta(s)$ und damit ist $\theta \circ \vartheta$ ein Unifikator von UP, da alle anderen Unifikationspaare von UP auch in UP' vorkommen. ■

Lemma 5.10 *Sei* $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *und* $\mathsf{UP} \Longrightarrow_\vartheta \mathsf{UP}'$ *ein Arbeitsschritt des Algorithmus 5. Ist* θ *ein Unifikator von* UP*, so existiert ein Unifikator* θ' *von* UP' *mit* $\theta(f) = (\theta' \circ \vartheta)(f)$ *für alle* $f \in \mathsf{flex}$*.*

Beweis. Auch hier sei auf das entsprechende Lemma 3.10 für die Muster-Unifikation verwiesen. Interessant sind hier also wieder die neuen Fälle, die eine Fallunterscheidung beinhalten.

Fall „Flexibel-Starr“: Sei $(r, s) \in \mathrm{U}$ mit $r = fc_1 \cdots c_k x_1 \cdots x_l$ mit $f \in \mathsf{flex}$, $k > 0$ und s starr. Dann tritt f nirgends sonst mehr in U frei auf. Gelte $\mathsf{forb} \cap \mathsf{FV}(s) \subseteq \{x_1, \ldots, x_l\}$. Seien $y_1, \ldots, y_k$ Variablen mit $\tau(y_i) = \tau(c_i)$ und $f_1, \ldots, f_k$ neue Variablen gleichen Typs wie f. Mit $r_i := f_i \vec{y_k} \vec{x_l}$ ist dann

$$\vartheta := \{\langle f, \lambda \vec{y_k}, \vec{x_l}.\mathsf{D}_{\vec{c_k}}(\vec{y_k}, s, \vec{r_k})\rangle\}.$$

Gelte $\theta(f) = \lambda\vec{y_k}\vec{x_l}.r'$. Es ist also $r'[\vec{y}_k ::= \vec{c}_k] = \theta(s)$. Definieren wir $\theta'(g) = \theta(g)$ für alle $g \in \mathsf{flex} \setminus \{f\}$ und

$$\theta'(f_i) := \lambda\vec{y_k}\vec{x_l}.r'[\vec{y_{i-1}} ::= \vec{c_{i-1}}],$$

so folgt $(\theta' \circ \vartheta)(f) = \lambda\vec{y_k}\vec{x_l}.r' = \theta(f)$. Um dies zu verstehen, betrachten wir zunächst die „innerste“ Fallunterscheidung in $\vartheta(f)$: $\mathsf{D}_{c_k}(y_k, s, r_k)$. Es ist dann

$$\theta'(\mathsf{D}_{c_k}(y_k, s, r_k)) = \mathsf{D}_{c_k}(y_k, \theta(s), r'[y_1 ::= c_1, \ldots, y_{k-1} ::= c_{k-1}]).$$

Wegen $r'[\vec{y}_k ::= \vec{c}_k] = \theta(s)$ folgt also $\mathsf{D}_{c_k}(y_k, \theta(s), r'[y_1 ::= c_1, \ldots, y_{k-1} ::= c_{k-1}]) = r'[y_1 ::= c_1, \ldots, y_{k-1} ::= c_{k-1}]$. Die nächste Fallunterscheidung ist also gleich diesem Term:

$$\mathsf{D}_{c_{k-1}}(y_{k-1}, r'[\vec{y}_{k-1} ::= \vec{c}_{k-1}], r'[\vec{y}_{k-2} ::= \vec{c}_{k-2}]),$$

Fahren wir auf diese Weise fort bis $k = 1$, so bleibt zum Schluß $\theta'(f_1\vec{y_k}\vec{x_l}) = r'$ übrig.

Fall „Flexibel≠Flexibel“: Sei $(r, s) \in \mathrm{U}$ mit

$$r = fc_1 \cdots c_k x_1 \cdots x_l, \quad s = gd_1 \cdots d_m y_1 \cdots y_n$$

und $f, g \in \mathsf{flex}$ sowie $k+m > 0$. Es sei $\{\vec{z}\} = \{\vec{x_l}\} \cap \{\vec{y_n}\}$. Es seien $f_1, \ldots, f_k$ neue Variablen gleichen Typs wie f und $g_1, \ldots, g_m$ neue Variablen gleichen Typs wie g, sowie h eine neue Variable mit $\tau(h\vec{z}) = \tau(r)$. Es ist dann

$$\vartheta := \{\langle f, \lambda\vec{v_k}x_l.\mathsf{D}_{\vec{c_k}}(\vec{v_k}, h\vec{z}, \vec{r_k})\rangle, \langle g, \lambda\vec{w_m}y_n.\mathsf{D}_{\vec{d_m}}(\vec{w_m}, h\vec{z}, \vec{s_m})\rangle\}$$

mit $r_i := f_i\vec{v_k}\vec{x_l}$ für $1 \leq i \leq k$ und $s_j := g_j\vec{w_m}\vec{y_n}$ für $1 \leq j \leq m$. Gelte $\theta(f) = \lambda\vec{v_k}\vec{x_l}.r'$ und $\theta(g) = \lambda\vec{w_m}\vec{y_n}.s'$. Dann ist

$$r'[v_1 ::= c_1, \ldots, v_k ::= c_k] = s'[w_1 ::= d_1, \ldots, w_m ::= d_m].$$

Außerdem gilt wegen $(\mathsf{dom}(\theta) \cup \mathsf{FV}(\theta)) \cap \mathsf{forb} = \emptyset$:

$$\mathsf{forb} \cap \mathsf{FV}(r') \subseteq \{x_1, \ldots, x_l\} \cap \mathsf{FV}(r') \subseteq \{\vec{z}\} \text{ und}$$
$$\mathsf{forb} \cap \mathsf{FV}(s') \subseteq \{y_1, \ldots y_n\} \cap \mathsf{FV}(s') \subseteq \{\vec{z}\}.$$

Wir setzen nun $\theta'(h) := \lambda\vec{z}.r'[v_1 ::= c_1, \ldots, v_k ::= c_k]$, $\theta'(X) = \theta(X)$ für alle $X \in \mathsf{flex} \setminus \{f, g\}$ sowie $\theta'(f_i) := r'[v_1 ::= c_1, \ldots, v_{i-1} ::= c_{i-1}]$ für $1 \leq i \leq k$ und $\theta'(g_i) := s'[w_1 ::= d_1, \ldots, w_{i-1} ::= d_{i-1}]$ für $1 \leq i \leq m$. Dann ist $\vartheta(\theta'(f)) = \theta(f)$ sowie $\vartheta(\theta'(g)) = \theta(g)$. Dazu betrachten wir zunächst wieder die innersten Fallunterscheidungen von $\vartheta(\theta'(f))$:

$\mathsf{D}_{c_k}(v_k, r'[v_1 ::= c_1, \ldots, v_k ::= c_k], r'[v_1 ::= c_1, \ldots, v_{k-1} ::= c_{k-1}]) =$
$r'[v_1 ::= c_1, \ldots, v_{k-1} ::= c_{k-1}]$
wegen $(r'[v_1 ::= c_1, \ldots, v_{k-1} ::= c_{k-1}])_{v_k}[c_k] = r'[v_1 ::= c_1, \ldots, v_k ::= c_k]$.

Und die innerste Fallunterscheidung von $\vartheta(\theta'(g))$:

$\mathsf{D}_{d_m}(w_m, r'[v_1 ::= c_1, \ldots, v_k ::= c_k], s'[w_1 ::= d_1, \ldots, w_{m-1} ::= d_{m-1}]) =$
$s'[w_1 ::= d_1, \ldots, w_{m-1} ::= d_{m-1}]$
wegen
$(s'[w_1 ::= d_1, \ldots, w_{m-1} ::= d_{m-1}])_{w_m}[d_m] = r'[v_1 ::= c_1, \ldots, v_k ::= c_k]$.

Fahren wir sukzessive fort, so erhalten wir als letzte Fallunterscheidungen

$$\mathsf{D}_{c_1}(v_1, r'_{v_1}[c_1], r') = r' \text{ und } \mathsf{D}_{d_1}(w_1, s'_{w_1}[d_1], s') = s'.$$

Es folgt $\vartheta(\theta'(f)) = \lambda \vec{v_k}\vec{x_l}.r'$ und $\vartheta(\theta'(g)) = \lambda \vec{w_m}\vec{y_n}.s'$. Offensichtlich ist θ' ein Unifikator von UP'. ■

Theorem 5.6 *Sei* UP *ein lineares* $\mathcal{C}$*-Muster Unifikationsproblem. Wenden wir den Algorithmus 5 auf* UP *an, so gilt:*

1. *Der Algorithmus terminiert.*
2. *Ist* UP *ein lösbares Unifikationsproblem, so ist die berechnete Substitution* θ *ein allgemeinster Unifikator bzgl. des* $\lambda\mathsf{D}$*-Kalküls für* UP*.*
3. UP *ist genau dann nicht lösbar, wenn der Algorithmus mit* $\perp$ *antwortet.*

Beweis. Die Terminierung war Inhalt des Lemmas 5.8.

Ist UP ein lösbares Unifikationsproblem, so sei $\mathsf{UP} \Rightarrow_{\vartheta_1} \mathsf{UP}_1 \Rightarrow_{\vartheta_2} \cdots \Rightarrow_{\vartheta_n} \mathsf{UP}_n$ der Ablauf des Algorithmus mit $\theta = \vartheta_n \circ \cdots \circ \vartheta_1$ als berechnete Antwort und $\mathsf{UP}_n = \emptyset$.

Es ist ε also ein Unifikator von UP_n. nach Lemma 5.9 ist dann $\varepsilon \circ \vartheta_n = \vartheta_n$ ein Unifikator von UP_{n-1}. Wenden wir dieses Lemma noch $n-1$-mal an, so erhalten wir, daß θ ein Unifikator von UP ist.

Sei nun σ ein beliebiger Unifikator von UP. Dann existiert ein Unifikator σ_1 von UP_1 mit $(\sigma_1 \circ \vartheta_1)(f) = \sigma(f)$ für alle $f \in \mathsf{dom}(\vartheta)$ und $\sigma_1(g) = \sigma(g)$ für alle $g \in \mathsf{dom}(\sigma) \setminus \mathsf{dom}(\vartheta)$. Wenden wir dieses Lemma nun $n-1$-mal an, so erhalten wir die Existenz eines Unifikator σ_n von $\mathsf{UP}_n = \emptyset$ mit $(\sigma_n \circ \theta)(f) = \sigma(f)$ für alle $f \in \mathsf{dom}(\theta)$. Also nach Definition $\theta \geq \sigma$.

Endet unsere Berechnung hingegen mit $\mathsf{UP}_n = \perp$, also UP_n ist unlösbar, so muß auch UP unlösbar sein. Sonst hätte man mit n-maliger Anwendung von Lemma 5.9 einen Widerspruch. ■

Nun wollen wir einige Beispiele berechnen.

Beispiel. Sei $c \in \mathcal{C}$ und sei uns als Unifikationsproblem gegeben:

$$\mathsf{UP} := (\{(fx_1cx_2, gx_2c)\}, \{x_1, x_2\}, \{f, g\}).$$

Wir haben $\theta := \varepsilon$ und starten also mit dem Sortieren. Wir erhalten

$$\begin{aligned}\vartheta = \quad & \mathsf{sort}(fx_1, c, x_2, \{f, g\}) \circ \mathsf{sort}(gx_2c, \{f, g\}) = \\ & \{\langle f, \lambda u, v, w.f'vuw\rangle, \langle g, \lambda w, v.g'vw\rangle\}.\end{aligned}$$

und setzten fort mit

$$\vartheta(\mathsf{UP}) = (\{(f'cx_1x_2, g'cx_2), \{x_1, x_2\}, \{f', g'\})$$

und $\theta = \vartheta$.

Nun folgt die Prozedur „Flexible ≠ Flexibel“. Dazu benötigen wir erst mal $\{x_2\} = \{x_1, x_2\} \cap \{x_2\}$ und Variabeln f_1 und g_1 mit $\tau(f_1) = \tau(f')$ sowie $\tau(g_1) = \tau(g')$. Außerdem sei h eine Variable mit $\tau(hx_2) = \tau(f'cx_1x_2)$. Dann erhalten wir $\vartheta := \mathsf{fg}(f'cx_1x_2, g'cx_2, \{x_1, x_2\}, \{f', g'\}) =$

$$\{\langle f', \lambda v, x_1, x_2.\mathsf{D}_c(v_1, hx_2, f_1vx_1x_2)\rangle, \langle g', \lambda v, x_2.\mathsf{D}_c(v, hx_2, g_1vx_2)\rangle\}.$$

Es ist dann

$$\vartheta(\mathsf{UP}) = (\{(hx_2, hx_2)\}, \{x_1, x_2\}, \{h, f_1, f_2\})$$

und also im nächsten Schritt $U = \emptyset$. Also erhalten wir als allgemeinsten Unifikator

$$(\vartheta \circ \theta) \mid_{\{f,g\}} = \{\langle f, \lambda u, v, w.\mathsf{D}_c(v, hw, f_1vuw)\rangle, \langle g, \lambda w, v.\mathsf{D}_c(v, hw, g_1vw)\rangle\}.$$

Beispiel. Seien $c, d \in \mathcal{C}$ und sei

$$\mathsf{UP} := (\{(fcdxy, c(gxz))\}, \{x, y, z\}, \{f, g\}).$$

Sei $\theta := \varepsilon$ und $V := \{f, g\}$.

Hier handelt es sich also um ein „Flexibel-Starr-Paar“. Erst mal stellen wir fest: $\mathsf{oc}(f, c(gxz)) = \mathsf{falsch}$. Der flexible Term ist bereits sortiert, also müssen wir $\mathsf{forb} \cap \mathsf{FV}(c(gxz)) = \{x, z\}$ betrachten und stellen leider fest, daß $\{x, z\} \not\subseteq \{x, y\}$ ist. Es stört also die Variable z. Folglich kommt nun

$$\vartheta := \mathsf{Prune}(z, c(gxz), \{x, y, z\}, \{f, g\}) = \{\langle g, \lambda u_1, u_2.g'u_1\rangle\}.$$

Wir fahren fort mit

$$\vartheta(\mathsf{UP}) = (\{(fcdxy, c(g'x))\}, \{x, y, z\}, \{f, g'\})$$

und $\theta := \vartheta$.

Nun haben wir keine störenden, verbotenen Variablen mehr frei in dem starren Term, wir können also die Prozedur „Flexibel-Starr“ aufrufen. Dazu benötigen wir Variablen f_1, f_2 gleichen Typs wie f. Es ist dann

$$\begin{aligned}\vartheta := \quad & \mathsf{fs}(fcdxy, c(g'x), \{x, y, z\}, \{f, g'\}) = \\ & \{\langle f, \lambda v, w, x, y.\mathsf{D}_c(v, \mathsf{D}_d(w, c(g'x), f_2vwxy), f_1vwxy)\rangle\}.\end{aligned}$$

Wir setzen fort mit

$$\vartheta(\mathsf{UP}) = (\{(c(g'x), c(g'x))\}, \{x, y, z\}, \{f_1, f_2, g'\}),$$

also erhalten wir demnächst $U = \emptyset$, folglich ist dann

$$\begin{aligned}(\vartheta \circ \theta) \mid_V = \quad & \{\langle f, \lambda v, w, x, y.\mathsf{D}_c(v, \mathsf{D}_d(w, c(g'x), f_2vwxy), f_1vwxy)\rangle, \\ & \langle g, \lambda u_1, u_2.g'u_1\rangle\}\}\end{aligned}$$

der allgemeinste Unifikator von UP.

So könnte man nun ewig fortfahren, allerdings führt die Linearität dazu, daß die Beispiele recht langweilig sind. Verzichten wir auf diese Einschränkung, so müssen wir wie bereits im vorherigen Abschnitt angedeutet, Unifikationsprozeduren für Terme definieren, die Fallunterscheidungen beinhalten.

5.4 Muster mit konstanten Argumenten - Nicht Linear

Wir müssen nun also noch Prozeduren erfinden für Terme mit Fallunterscheidungen. Und natürlich eine Prozedur für zwei flexible Terme mit gleichem Kopf. Dazu müssen wir zunächst die Definition der $\mathcal{C}$-Muster erweitern zu $\mathsf{D}\mathcal{C}$-Mustern, da nun auch Fallunterscheidungen in dem Unifikationsproblem vorkommen können. Wir geben dann einen vollständigen und korrekten Algorithmus an, der allerdings nicht terminieren musß. Dieses Verhalten erinnert stark an den Algorithmus von Huet aus Abschnitt 2.3.

Definition. *Ein* $\lambda\mathsf{D}$*-normaler Term* r *heißt* $\mathsf{D}\mathcal{C}$-Muster *bzgl.* flex *und* forb*, falls für alle Teilterme* $\lambda\vec{y_m}.As_1 \cdots s_n$ *mit* $A \in \mathsf{flex} \cap \mathsf{FV}(r)$ *und alle Teilterme der Gestalt* $D_c(r', s, t)$ *gilt:*

1. *Für alle* $1 \leq i \leq n$*:* $s_i = A_i \in \mathcal{C} \cup \mathsf{forb} \cup \mathsf{BV}(r) \setminus \{y_1, \ldots, y_m\}$,
2. *Für alle* $1 \leq i \neq j \leq n$*:* $s_i = x_i \in \mathcal{V}$ *impliziert* $s_i \neq s_j$,
3. $r' \in \mathsf{forb} \cup \mathsf{BV}(r)$.

Ein $\lambda\mathsf{D}$*-Term* r *heißt* $\mathsf{D}\mathcal{C}$-Muster *bzgl.* flex *und* forb*, falls seine Normalform* $r\downarrow$ *ein* $\mathsf{D}\mathcal{C}$-Muster *bzgl.* flex *und* forb *gemäß obiger Definition ist.*

Insbesondere dürfen in normalen Termen Fallunterscheidungen nur die Form $\mathsf{D}_c(x, s, t)$ haben mit einer verbotenen bzw. gebundenen Variable x. Jedes $\mathsf{D}\mathcal{C}$-Muster r ist also ein guter Term. Es ist $\mathsf{DV}(r) \subseteq \mathsf{forb}$ für alle $\mathsf{D}\mathcal{C}$-Muster r. Folglich gilt für alle $\mathsf{D}\mathcal{C}$-Muster r und alle Substitutionen ϑ mit $(\mathsf{dom}(\vartheta) \cup \mathsf{FV}(\vartheta)) \cap \mathsf{forb} = \emptyset$: $\vartheta(r)$ ist ein guter Term und somit ist die Gleichheit $=$ verträglich mit den Substitutionen, die wir für Unifikationsprobleme zulassen. Es folgt also für alle $\mathsf{D}\mathcal{C}$-Muster r, s und alle Substitutionen ϑ: Aus $r = s$ folgt $\vartheta(r) = \vartheta(s)$. (Siehe Lemmatas 5.4 und 5.5).

Die folgenden Prozeduren müssen dann einfach dem Algorithmus 5 hinzugefügt werden.

Zuerst fällt uns auf, daß wir die Prozedur Prune erweitern müssen, nämlich auf den Fall, daß eine D-Variable x entfernt werden soll. Die kann gelingen, falls wir es schaffen, beide Alternativen in der Fallunterscheidung zu unifizieren, genauer daß wir für $\mathsf{D}_c(x, s, t)$ versuchen müssen, $s_x[c]$ und $t_x[c]$ zu unifizieren.

Prozedur 5.6 Pruning-DV $\mathsf{PruneDV}(x, r, \mathsf{UP})$.

Sei $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$*,* $r \in \mathrm{U}$ *und* $x \in \mathsf{forb}$*. Sei* $x \in \mathsf{FV}(r)$ *und existiere ein Teilterm* $D_c(x, s, t)$ *von* r*. Dann definieren wir*

$$\mathsf{PruneDV}(x, r, \mathsf{UP}) := (\{(s_x[c], t_x[c])\} \cup \mathrm{U}, \mathsf{forb}, \mathsf{flex}).$$

Dazu möchten wir gleich ein Lemma beweisen, welches zeigt, daß mit Prune und $\mathsf{PruneDV}$ alle Möglichkeiten zum Entfernen verbotener Variablen ausgeschöpft werden.

Lemma 5.11 *Sei r ein DC-Muster bzgl.* forb *und* flex *in langer Normalform und sei $x \in \mathsf{FV}(r) \cap \mathsf{forb}$ mit $x \notin \mathsf{DV}(r)$ und*

$$M := \{\, s \text{ Teilterm von } r \mid x \in \mathsf{FV}(s) \text{ und } \mathsf{hd}(s) \in \mathsf{flex} \,\} = \emptyset.$$

Dann gilt für jede Substitution ϑ mit $\mathsf{dom}(\vartheta) \cap \mathsf{forb} = \emptyset$: $x \in \mathsf{FV}(\vartheta(r))$.

Beweis. Sei ϑ eine beliebige Substitution mit $\mathsf{dom}(\vartheta) \cap \mathsf{forb} = \emptyset$.

Beweis durch Induktion über die Anzahl von Fallunterscheidungen in r. Induktionsanfang: In r gibt es keine Fallunterscheidung. Da $M = \emptyset$, ist jeder Teilterm der x frei enthält starr und folglich kann x nicht entfernt werden.

Induktionsschluß. Sei $r = D_c(y, t, t')$. Dann ist also $x \in \mathsf{FV}(t)$ oder $x \in \mathsf{FV}(t')$. Nach Induktionsvoraussetzung gilt $x \in \mathsf{FV}(\vartheta(t))$ oder $x \in \mathsf{FV}(\vartheta(t'))$. Letzte Möglichkeit: Es ist $x \in \mathsf{FV}(t)$ und $\vartheta(t')_y[c] = \vartheta(t)_y[c]$. Dann ist $\vartheta(r) = \vartheta(t')$. Aber leider ist dann x auch in $\mathsf{FV}(\vartheta(t'))$, wegen $\vartheta(t')_y[c] = \vartheta(t'_y[c]) = \vartheta(t_y[c]) = \vartheta(t)$, da r in langer Normalform und somit $y \notin \mathsf{FV}(t)$ ist. ■

Wir betrachten nun zwei flexiblen Termen gleichen Kopfs. Dazu ein Beispiel: Stellen wir uns vor, wir möchten die Terme $f0$ und $f1$ unifizieren, wobei $0 \neq 1$ Konstanten unserer Sprache seien. Gesucht ist also eine Funktion, die an der Stelle 0 und an der Stelle 1 den gleichen Wert annimmt, sonst undefiniert ist, also etwas der Gestalt

$$x \mapsto \mathsf{Wenn}\, x = 0, \mathsf{dann}\, r, \mathsf{sonst}\, (\mathsf{Wenn}\, x = 1, \mathsf{dann}\, r, \mathsf{sonst}\, f'x).$$

Dies lösen wir, indem wir in solchen Fällen das Problem zu einem Problem zweier flexibler Terme mit verschiedenen Köpfen machen, in unsrem Beispiel also zunächst f definieren als

$$x \mapsto \mathsf{Wenn}\, x = 0, \mathsf{dann}\, f_1 x, \mathsf{sonst}\, f_2 x.$$

Prozedur 5.7 Flexibel = Flexibel $\mathsf{ff}(r, s, \mathsf{forb}, \mathsf{flex})$.

Seien $r = fc_1 \cdots c_k x_1 \cdots x_l$ und $s = fs_1 \cdots s_m$ zwei DC-Muster bzgl. forb *und* flex *mit $f \in \mathsf{flex}$. Seien $\vec{v_k}$ Variablen mit $\tau(v_i) = \tau(c_i)$ für alle $1 \leq i \leq k$.*

Fall: Es existiert ein Index $i \in \{1, \ldots, k\}$ mit $s_i = d \in \mathcal{C}$ und $d \neq c_i$. Dann definieren wir

$$\mathsf{ff}(r, s, \mathsf{forb}, \mathsf{flex}) := \{\langle f, \lambda \vec{v_k} \vec{x_l}.\mathsf{D}_{c_i}(v_i, f_1 \vec{v_k} \vec{x_l}, f_2 \vec{v_k} \vec{x_l})\rangle\},$$

wobei f_1, f_2 neue Variablen gleichen Typs wie f seien.

Sonst sei $\{\vec{z}\} \subseteq \{\vec{v_k}\} \cup \{\vec{x_l}\}$ mit $v_i \in \{\vec{z}\}$ gdw. $c_i = s_i$ und $x_j \in \{\vec{z}\}$ gdw. $x_j = s_j$ für alle $1 \leq i \leq k$ und $1 \leq j \leq l$. Wir definieren dann

$$\mathsf{ff}(r, s, \mathsf{forb}, \mathsf{flex}) := \{\langle f, \lambda \vec{v_k} \vec{x_l}.h\vec{z}\rangle\}$$

mit neuer, passender Variable h.

Nun wollen wir einen flexiblen Term mit einer Fallunterscheidung unifizieren. Dabei gehen wir so vor, daß wir den Fallunterscheidungsterm wie einen starren Term behandeln.

Prozedur 5.8 Flex-D $\mathsf{fD}(r, s, \mathsf{forb}, \mathsf{flex})$.

Sei $r = f\vec{c_k}\vec{x_l}$ *mit* $f \in \mathsf{flex}$ *und* $s = \mathsf{D}_c(x, s', s'')$. *Sei* $\mathsf{forb} \cap \mathsf{FV}(s) \subseteq \{\vec{x_l}\}$. *Dann definieren wir:*

$$\mathsf{fD}(r, s, \mathsf{forb}, \mathsf{flex}) := \{\langle f, \lambda\vec{v_k}\vec{x_l}.D_{\vec{c_k}}(\vec{v_k}, s, \vec{t_k})\rangle\}$$

mit $t_i := f_i\vec{v_k}\vec{x_l}$ *und* f_i *ist eine neue Variable mit* $\tau(f_i) = \tau(f)$ *für alle* $1 \leq i \leq k$.

Nun überlegen wir uns, was wir bei zwei gleichen Fallunterscheidungen machen, also zwei Termen der Gestalt $D_c(x, \cdot, \cdot)$. Offensichtlich müssen wir dann jeweils die beiden Alternativterme, also die beiden „dann“- und die beiden „sonst“-Terme unifizieren.

Prozedur 5.9 $\mathsf{D} = \mathsf{D}\ \mathsf{dd}(\mathsf{UP})$.

Sei $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein* DC*-Muster-Unifikationsproblem und* $(r, s) \in \mathrm{U}$ *mit* $r = D_c(x, r_1, r_2)$ *und* $s = D_c(x, s_1, s_2)$. *Dann ist*

$$\mathsf{dd}(\mathsf{UP}) = ((\mathrm{U} \setminus \{(r, s)\}) \cup \{(r_1, s_1), (r_2, s_2)\}, \mathsf{forb}, \mathsf{flex}).$$

Was passiert nun, wenn eine Fallunterscheidung $\mathsf{D}_c(x, r, s)$ einen starren Term t trifft? Dazu muß wohl die Fallunterscheidung so bearbeitet werden, daß sie verschwindet. Also daß für $\mathsf{D}_c(x, r, s)$ ein ϑ gefunden wird mit $\vartheta(s)_x[c] = r$. Außerdem muß dann natürlich auch noch $\vartheta(s)$ gleich $\vartheta(t)$ sein.

Prozedur 5.10 D-Starr $\mathsf{ds}(\mathsf{UP})$.

Sei $\mathsf{UP} = (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein* DC*-Muster-Unifikationsproblem und* $(r, s) \in \mathrm{U}$ *mit* $r = D_c(x, r', r'')$ *und* s *ein starrer Term. Dann definieren wir*

$$\mathsf{ds}(\mathsf{UP}) := ((\mathrm{U} \setminus \{(r, s)\}) \cup \{(r', r''_x[c]), (r'', s)\}, \mathsf{forb}, \mathsf{flex}).$$

Und nun noch der Fall zweier Fallunterscheidungen $D_c(x, r', r'')$ und $D_d(y, s', s'')$ mit $c \neq d$. Dabei müssen wir beide Fallunterscheidungen los werden, also ist eine Substitution ϑ gesucht mit: $\vartheta(r'')_x[c] = \vartheta(r')$, $\vartheta(s'')_y[d] = \vartheta(s')$ und $\vartheta(r'') = \vartheta(s'')$.

Prozedur 5.11 $\mathsf{D} \neq \mathsf{D}\ \mathsf{dnd}(\mathsf{UP})$.

Sei $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein* DC*-Muster-Unifikationsproblem und* $(r, s) \in \mathrm{U}$ *mit* $r = D_c(x, r', r'')$ *und* $s = D_d(y, s', s'')$ *und* $c \neq d$. *Dann definieren wir*

$$\mathsf{dnd}(\mathsf{UP}) = ((\mathrm{U} \setminus \{(r, s)\}) \cup \{(r''_x[c], r'), (s''_y[d], s'), (r'', s'')\}, \mathsf{forb}, \mathsf{flex}).$$

Nun setzen wir also den Algorithmus aus den obigen Prozeduren und aus den Prozeduren des vorherigen Abschnittes zusammen.

Algorithmus 6 *Sei* $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein* DC*-Muster Unifikationsproblem. Sei* $\theta := \varepsilon$ *und* $V := \mathsf{flex}$.

1. *Ersetze* UP *durch* $(\mathrm{U} \setminus \{\, (r, s) \in \mathrm{U} \mid r = s \,\}, \mathsf{forb}, \mathsf{flex})$.

2. *Ersetze* UP *durch* $\mathsf{SIMPL}(\mathsf{UP})$.

3. *Ist* $\mathsf{UP} = \bot$, *so gib* $\bot$ *aus und beende, ist* $\mathrm{U} = \emptyset$, *so gib* $\theta\,|_V$ *aus und beende.*

4. *Wähle ein Paar* $(r, s) \in \mathrm{U}$ *und unterscheide folgende Fälle:*

   (a) *Fall* r *flexibel und* s *starr:*

      (i) *Ist* $\mathsf{oc}(\mathsf{hd}(r), s) = \mathsf{wahr}$, *so setzte* $\mathsf{UP} := \bot$ *und gehe zu 3.*

      (ii) *Ist* r *noch nicht sortiert und* $\vartheta := \mathsf{sort}(r, \mathsf{flex})$, *so ersetze* UP *durch* $\vartheta(\mathsf{UP})$ *und* θ *durch* $\vartheta \circ \theta$ *und gehe zu 1.*

      (iii) *Sei* $r = f\vec{c}\vec{x}$. *Existiert ein* $y \in (\mathsf{forb} \cap \mathsf{FV}(s)) \backslash \{\vec{x}\}$, *so unterscheide folgende Fälle:*
      $y \notin \mathsf{DV}(s)$*: Ist* $\mathsf{Prune}(y, s, \mathsf{forb}, \mathsf{flex}) = \dagger$, *so setze* $\mathsf{UP} := \bot$ *und gehe zu 3. Sonst sei* $\vartheta := \mathsf{Prune}(y, s, \mathsf{forb}, \mathsf{flex})$. *Ersetze* UP *durch* $\vartheta(\mathsf{UP})$ *und* θ *durch* $\vartheta \circ \theta$ *und gehe zu 1.*
      $y \in \mathsf{DV}(s)$*: Ersetze* UP *durch* $\mathsf{PruneDV}(y, s, \mathsf{UP})$ *und gehe zu 1.*

      (iv) *Sei* $r = f\vec{c}\vec{x}$ *und gelte* $\mathsf{oc}(\mathsf{hd}(r), s) = \mathsf{falsch}$ *sowie* $\mathsf{forb} \cap \mathsf{FV}(s) \subseteq \{\vec{x}\}$. *Dann sei* $\vartheta = \mathsf{fs}(r, s, \mathsf{forb}, \mathsf{flex})$. *Ersetze* UP *durch* $\vartheta(\mathsf{UP})$ *und* θ *durch* $\vartheta \circ \theta$ *und gehe zu 1.*

   (b) *Fall* r *flexibel und* s *von der Form* $D_c(x, s', s'')$*:*

      (i) *Ist* $\mathsf{oc}(\mathsf{hd}(r), s) = \mathsf{wahr}$, *so setzte* $\mathsf{UP} := \bot$ *und gehe zu 3.*

      (ii) *Ist* r *noch nicht sortiert und* $\vartheta = \mathsf{sort}(r, \mathsf{flex})$, *so ersetze* UP *durch* $\vartheta(\mathsf{UP})$ *und* θ *durch* $\vartheta \circ \theta$ *und gehe zu 1.*

      (iii) *Sei* $r = f\vec{c}\vec{x}$. *Existiert ein* $y \in (\mathsf{forb} \cap \mathsf{FV}(s)) \backslash \{\vec{x}\}$, *so unterscheide folgende Fälle:*
      $y \notin \mathsf{DV}(s)$*: Ist* $\mathsf{Prune}(y, s, \mathsf{forb}, \mathsf{flex}) = \dagger$, *so setze* $\mathsf{UP} := \bot$ *und gehe zu 3. Sonst sei* $\vartheta := \mathsf{Prune}(y, s, \mathsf{forb}, \mathsf{flex})$. *Ersetze* UP *durch* $\vartheta(\mathsf{UP})$ *und* θ *durch* $\vartheta \circ \theta$ *und gehe zu 1.*
      $y \in \mathsf{DV}(s)$*: Ersetze* UP *durch* $\mathsf{PruneDV}(y, s, \mathsf{UP})$ *und gehe zu 1.*

      (iv) *Gelte* $\mathsf{oc}(\mathsf{hd}(r), s) = \mathsf{falsch}$ *und* $\mathsf{forb} \cap \mathsf{FV}(s) \subseteq \{\vec{x}\}$. *Dann sei* $\vartheta = \mathsf{fD}(r, s, \mathsf{forb}, \mathsf{flex})$. *Ersetze* UP *durch* $\vartheta(\mathsf{UP})$ *und* θ *durch* $\vartheta \circ \theta$ *und gehe zu 1.*

   (c) *Fall* r *und* s *flexibel mit* $\mathsf{hd}(r) \neq \mathsf{hd}(s)$.

      (i) *Ist* r *oder* s *noch unsortiert, so sei* $\vartheta := \mathsf{sort}(r, \mathsf{flex}) \circ \mathsf{sort}(s, \mathsf{flex})$. *Ersetze* θ *durch* $\vartheta \circ \theta$ *und* UP *durch* $\vartheta(\mathsf{UP})$ *und gehe zu 1.*

      (ii) *Sind* r *und* s *sortiert, so sei* $\vartheta = \mathsf{fg}(r, s, \mathsf{forb}, \mathsf{flex})$. *Ersetze* θ *durch* $\vartheta \circ \theta$ *und* UP *durch* $\vartheta(\mathsf{UP})$ *und gehe zu 1.*

   (d) *Fall* r *und* s *flexibel mit* $\mathsf{hd}(r) = \mathsf{hd}(s)$.

      (i) *Ist* r *noch nicht sortiert, so sei* $\vartheta := \mathsf{sort}(r, \mathsf{flex})$. *Ersetze* UP *durch* $\vartheta(\mathsf{UP})$ *und* θ *durch* $\vartheta \circ \theta$ *und gehe zu 1.*

      (ii) *Ist* r *sortiert, so sei* $\vartheta := \mathsf{ff}(r, s, \mathsf{forb}, \mathsf{flex})$. *Ersetze* UP *durch* $\vartheta(\mathsf{UP})$ *und* θ *durch* $\vartheta \circ \theta$ *und gehe zu 1.*

   (e) *Ist* $r = D_c(x, r_1, r_2)$ *und* $s = D_c(x, s_1, s_2)$, *so ersetze* UP *durch* $\mathsf{dd}(\mathsf{UP})$ *und gehe zu 1.*

   (f) *Ist* $r = D_c(x, r_1, r_2)$ *und* s *ein starrer Term, so ersetze* UP *durch* $\mathsf{ds}(\mathsf{UP})$ *und gehe zu 1.*

(g) *Ist* $r = D_c(x, r_1, r_2)$ *und* $s = D_d(y, s_1, s_2)$ *mit* $c \neq d$, *so ersetze* UP *durch* $\mathsf{dnd}(\mathsf{UP})$ *und gehe zu 1.*

Wir schreiben wieder $\mathsf{UP} \Rightarrow_\vartheta \mathsf{UP}'$ für einen Arbeitsschritt des Algorithmus mit berechneter Substitutiuon ϑ. Nun müssen wir noch die Lemmatas 5.7, 5.9 und 5.10 erweitern um diese neuen Fälle. Diese Lemmatas gelten auch für nichtlineare Probleme, in den Beweisen im letzten Abschnitt muß man nur für die nicht bearbeiteten Unifikationspaare statt „$(r, s) \in \mathrm{U}$ gdw. $(r, s) \in \mathrm{U}'$“ schreiben: „$(r, s) \in \mathrm{U}$ gdw. $(\vartheta(r), \vartheta(s)) \in \mathrm{U}'$“.

Lemma 5.12 *Sei* $\mathsf{UP} \Longrightarrow_\vartheta \mathsf{UP}'$ *ein Arbeitsschritt des Algorithmus 6. Ist* UP *ein* DC*-Muster- Unifikationsproblem so auch* UP'.

Beweis. Die folgt unmittelbar aus der Tatsache, daß die Argumente einer flexiblen Variable nur Konstanten oder verbotene bzw. gebundene Variablen sind. ■

Lemma 5.13 *Sei* $\mathsf{UP} \Longrightarrow_\vartheta \mathsf{UP}'$ *ein Arbeitsschritt des Algorithmus 6. Sei* θ *ein Unifikator von* UP'. *Dann ist* $\theta \circ \vartheta$ *ein Unifikator von* UP.

Beweis. Wir zeigen also diese Aussage für die neu hinzu gekommenen Prozeduren.

Fall „Flexibel-D“. Geht ganz analog zum Fall „Flexibel-Starr“ in Lemma 5.9.

Fall „PruneDV“. Es ist $\mathrm{U} \subseteq \mathrm{U}'$.

Fall „Flexibel = Flexibel“. Im ersten Fall finden wir zu jedem Paar $(r, s) \in \mathrm{U}$ ein Paar $(r', s') \in \mathrm{U}$ mit $r' = \vartheta(r)$ und $s' = \vartheta(s)$. Folglich gilt für alle $(r, s) \in \mathrm{U}$: $\theta(\vartheta(r)) = \theta(r') = \theta(s') = \theta(\vartheta(s))$.

Im zweiten Fall ist ϑ offensichtlich ein Unifikator von $(f\vec{c}\vec{x}, f\vec{s})$. Außerdem gilt für alle anderen Paare (r, s) in UP, daß $(\vartheta(r), \vartheta(s))$ in UP' ist. Also folgt wieder wie oben $\theta \circ \vartheta$ ist Unifikator von UP.

Fall „D = D“. Es ist $\vartheta = \varepsilon$, also müssen wir zeigen, daß θ ein Unifikator von UP ist. Sei $(D_c(x, r, r'), D_c(x, s, s'))$ in UP ersetzt worden durch $(r, s), (r's')$ in UP'. Also ist θ ein Unifikator von (r, s) und (r', s'). Somit folgt $\theta(D_c(x, r, r')) = D_c(x, \theta(r), \theta(r')) = D_c(x, \theta(s), \theta(s')) = \theta(D_c(x, s, s')$.

Fall „D-Starr“. Wegen $\vartheta = \varepsilon$ müssen wir zeigen, daß θ ein Unifikator von UP ist. Sei $(D_c(x, r, r'), s)$ aus UP ersetzt worden durch $(r'_x[c], r), (r', s)$ in UP'. Dann folgt $\theta(D_c(x, r, r')) = D_c(x, \theta(r), \theta(r')) = \theta(r') = \theta(s)$, da $\theta(r')_x[c] = \theta(r'_x[c]) = \theta(r)$.

Fall „D ≠ D“. Es ist wieder $\vartheta = \varepsilon$, also müssen wir zeigen, daß θ ein Unifikator von UP ist. Es sei also $(D_c(x, r, r'), D_d(y, s, s'))$ in UP ersetzte worden durch $(r'_x[c], r), (s'_y[d], s), (r', s')$ in UP'. Es folgt $\theta(D_c(x, r, r')) = D_c(x, \theta(r), \theta(r')) = \theta(r') = \theta(s') = D_d(y, \theta(s), \theta(s')) = \theta(D_d(y, s, s'))$, da $\theta(r')_x[c] = \theta(r'_x[c]) = \theta(r)$ und $\theta(s')_y[c] = \theta(s'_y[c]) = \theta(s)$ ist. ■

Lemma 5.14 *Sei* $\mathsf{UP} \Longrightarrow_\vartheta \mathsf{UP}'$ *ein Arbeitsschritt des Algorithmus 6. Sei* $\mathsf{UP} := (\mathrm{U}, \mathsf{forb}, \mathsf{flex})$ *ein lösbares Unifikationsproblem und sei* θ *ein Unifikator von* UP. *Dann existiert ein Unifikator* θ' *von* UP' *mit* $\theta(f) = (\theta' \circ \vartheta)(f)$ *für alle* $f \in \mathsf{flex}$.

Beweis. Wieder betrachten wir die neu hinzugekommenen Prozeduren.

Fall „Flexibel-D“. Analog zum Fall „Flexibel-Starr“ in Lemma 5.10.

Fall „PruneDV“. Ist das Paar $(f\vec{c}\vec{x}, s)$ in U mit s entweder starr oder von der Form $D_c(z, s', s'')$ und $y \in \mathsf{DV}(s)$ mit $y \notin \{\vec{x}\}$, so kann y nicht in $\theta(f\vec{c}\vec{x})$ liegen. Also muß y aus s entfernt werden und da $y \in \mathsf{DV}(s)$, existiert ein Teilterm der Gestalt $D_d(y, t, t')$. Folglich kann dies nur gelingen, indem man $\theta(t')_y[d] = \theta(t'_y[d]) = \theta(t)$ erreicht. Somit ist θ auch Unifikator von UP'.

Fall „Prune“. Es sei nur erwähnt, daß im Falle $\mathsf{Prune}(.,.,.,.) = \bot$ das Problem UP nicht unifizierbar ist, siehe dazu Lemma 5.11.

Fall „Flexibel = Flexibel“. Im ersten Fall sei θ also ein Unifikator von $f\vec{c_k}\vec{x_l}$ und $f\vec{s_m}$. Es ist $\vartheta(f) = \lambda\vec{v_k}\vec{x_l}.D_{c_i}(v_i, f_1\vec{v_k}\vec{x_l}, f_2\vec{v_k}\vec{x_l})$. Wir definieren $\theta'(f_1) = \theta'(f_2) = \theta(f)$ und $\theta'(g) = \theta(g)$ für alle $g \in \mathsf{dom}(\theta) \setminus \{f\}$. Dann folgt

$$\begin{aligned}(\theta' \circ \vartheta)(f) = \quad & \theta'(\lambda\vec{v_k}\vec{x_l}.D_{c_i}(v_i, f_1\vec{v_k}\vec{x_l}, f_2\vec{v_k}\vec{x_l}) = \\ & \lambda\vec{v_k}\vec{x_l}.D_{c_i}(v_i, \theta(f)\vec{v_k}\vec{x_l}, \theta(f)\vec{v_k}\vec{x_l}) = \\ & \lambda\vec{v_k}\vec{x_l}.\theta(f)\vec{v_k}\vec{x_l} = \\ & \theta(f).\end{aligned}$$

Also ist θ' ein Unifikator von UP'.

Im zweiten Fall ist $\vartheta(f) = \lambda\vec{v_k}\vec{x_l}.h\vec{z}$, wobei $\{\vec{z}\} \subseteq \{\vec{v_k}\} \cup \{\vec{x_l}\}$ mit v_i in $\vec{z}$ gdw. $s_i = c_i$ und x_j in $\vec{z}$ gdw. $s_j = x_j$. Sei $\theta(f) = \lambda\vec{v_k}, \vec{x_l}.t$. Dann folgt $(\{\vec{v_k}\} \cup \{\vec{x_l}\}) \cap \mathsf{FV}(t) \subseteq \{\vec{z}\}$. Denn ist $v_i \in \mathsf{FV}(t)$, so ist c_i in $\theta(f\vec{c_k}\vec{x_l})$ und an gleicher Stelle s_i in $\theta(f\vec{s_m})$. Folglich $s_i \in \mathcal{C}$ und $s_i = c_i$, sonst wäre der erste Fall der Prozedur „Flexibel = Flexibel“ zur Anwendung gekommen. Ist $x_j \in \mathsf{FV}(t)$, so muß $s_j = x_j$ sein, da beide an gleicher Position vorkommen. Wir können also ohne Gefahr $\theta'(h) := \lambda\vec{z}.t$ und $\theta'(g) := \theta(g)$ für alle $g \in \mathsf{dom}(\theta) \setminus \{f\}$ definieren. Dann ist

$$\theta'(\vartheta(f)) = \theta'(\lambda\vec{v_k}\vec{x_l}.h\vec{z}) = \lambda\vec{v_k}\vec{x_l}.\theta'(h)\vec{z} = \lambda\vec{v_k}\vec{x_l}.t = \theta(f)$$

und somit θ' ein Unifikator von UP'.

Fall „D = D“. Wegen $\theta(D_c(x, r, s)) = D_c(x, \theta(r), \theta(s))$ ist θ auch ein Unifikator von UP'.

Fall „D-Starr“. Sei also θ ein Unifikator von $D_c(x, r, r')$ und s mit $\mathsf{hd}(s) \in \mathsf{forb} \cup \mathcal{C}$. Also muß $\theta(D_c(x, r, r'))$ so sein daß $\theta(r')_x[c] = \theta(r'_x[c]) = \theta(r)$ und $\theta(r') = \theta(s)$ ist. Somit ist θ auch Unifikator von UP'.

Fall „D ≠ D“. Es gilt $\theta(D_c(x, r, r')) = \theta(D_d(y, s, s'))$. Also muß gelten $\theta(r')_x[c] = \theta(r'_x[c]) = \theta(r)$ und $\theta(s')_y[d] = \theta(s'_y[d]) = \theta(s)$. Es folgt $\theta(r') = \theta(D_c(x, r, r')) = \theta(D_d(y, s, s')) = \theta(s')$. Somit ist θ auch Unifikator von UP'. ∎

Theorem 5.7 *Sei* UP *ein* DC*-Muster-Unifikationsproblem und terminiere die Anwendung des Algorithmus 6 auf* UP. *Dann gilt:*

1. *Ist die Antwort $\bot$, so ist* UP *unlösbar.*
2. *Ist die Antwort eine Substitution ϑ, so ist dies ein allgemeinster Unifikator von* UP *im* $\lambda\mathsf{D}$*-Kalkül.*

Beweis. Ganz analog zu Theorem 5.6 zeigt man beide Aussagen mit Hilfe der Lemmata 5.13 und 5.14. ■

Es folgen einige Beispiele. Erst mal kümmern wir uns um die Frage der Terminierung.

Beispiel. Sei

$$\mathsf{UP} := (\{(\mathsf{D}_c(x, c, fx), c)\}, \{x\}, \{f\})$$

ein DC-Musterunifikationsproblem mit $c \in \mathcal{C}$. Es folgt die D-Starr Prozedur, wir erhalten

$$\mathsf{ds}(\mathsf{UP}) = (\{(c, fc), (fx, c)\}, \{x\}, \{f\}).$$

Die Prozedur SIMPL dreht das erste Paar einfach herum zu (fc, c), es folgt die Prozedur Flexibel-Starr:

$$\vartheta := \mathsf{fs}(fc, c, \{x\}, \{f\}) = \{\langle f, \lambda y.\mathsf{D}_c(y, c, f'y)\rangle\}.$$

Wir fahren fort mit

$$\vartheta(\mathsf{UP}) = (\{(\mathsf{D}_c(x, c, f'x), c)\}, \{x\}, \{f'\})$$

und dies kommt uns recht bekannt vor. Also terminiert diese Art der Anwendung nicht. Bearbeiten wir allerdings in $\mathsf{ds}(\mathsf{UP})$ zuerst das Paar (fx, c), so erhalten wir die Lösung $\vartheta := \{\langle f, \lambda x.c\rangle\}$ als allgemeinsten Unifikator von UP.

Beispiel. Betrachten wir

$$\mathsf{UP} = (\{(fc, fd), (fx, gc)\}, \{x\}, \{f, g\}).$$

Sei $\theta := \emptyset$.

Das erste Paar ergibt die Lösung

$$\vartheta := \mathsf{ff}(fc, fd, \{x\}, \{f, g\}) = \{\langle f, \lambda y.D_c(y, f_1y, f_2y)\rangle\}.$$

mit $\tau(f_1) = \tau(f_2) = \tau(f)$. Es geht also weiter mit

$$\vartheta(\mathsf{UP}) = (\{(f_1c, f_2d), (D_c(x, f_1x, f_2x), gc)\}, \{x\}, \{f_1, f_2, g\}).$$

und $\theta := \vartheta$.

Das erste Paar erfordert nun $\vartheta :=$

$$\mathsf{fg}(f_1c, f_2d, \{x\}, \{f_1, f_2, g\}) = \{\langle f_1, \lambda y.D_c(y, h, f'y)\rangle, \langle f_2, \lambda y.D_d(y, h, f''y)\rangle\}.$$

mit passender Variable h. Wegen

$$\vartheta(D_c(x, f_1x, f_2x)) = \quad D_c(x, D_c(x, h, f'x), D_d(x, h, f''x)) = \\ D_c(x, h, D_d(x, h, f''x))$$

erhalten wir demnach

$$\vartheta(\mathsf{UP}) = (\{(h,h),(D_c(x,h,D_d(x,h,f''x)),gc)\},\{x\},\{g,h,f',f''\})$$

und $\theta =$

$$\{\langle f_1, \lambda y.D_c(y,h,f'y)\rangle, \langle f_2, \lambda y.D_d(y,h,f''y)\rangle, \langle f, \lambda y.D_c(y,h,D_d(y,h,f''y))\rangle\}.$$

Nach dem Entfernen des gelösten Paares (h,h) geht es weiter mit

$$\mathsf{UP} ::= \mathsf{SIMPL}(\mathsf{UP}) = (\{(gc, D_c(x,h,D_d(x,h,f''x)))\},\{x\},\{g,h,f',f''\}). \quad (*)$$

Vor dem Aufruf der Prozedur Flex-D muß die verbotene Variable x entfernt werden, diese ist eine D-Variable, wir setzten also mit

$$\begin{array}{l}\mathsf{PruneDV}(x, D_c(x,h,D_d(x,h,f''x)), \mathsf{UP}) = \\ (\{(h,f''c),(gc,D_c(x,h,D_d(x,h,f''x)))\},\{x\},\{g,h,f',f''\})\end{array}$$

fort, da $D_d(x,h,f''x)_x[c] = f''c$.

Das erste Paar ergibt $\vartheta := \mathsf{fg}(h,f''c,\{x\},\{g,h,f',f''\}) =$

$$\{\langle h,h'\rangle, \langle f'', \lambda x.D_c(x,h',f'''x)\rangle\}.$$

Und es geht weiter mit $\vartheta(\mathsf{UP}) =$

$$(\{(h',h'),(gc,D_c(x,h',D_d(x,h',D_c(x,h',f'''x))))\},\{x\},\{g,f',f''',h'\}).$$

Es gilt

$$D_d(x,h',D_c(x,h',f'''x))_x[c] = D_d(c,h',D_c(c,h',f'''c)) = h'$$

also $D_c(x,h',D_d(x,h',D_c(x,h',f'''x))) = D_d(x,h',D_c(x,h',f'''x))$. Wir haben also nun alles zusammen

$$\mathsf{UP} = (\{(gc, D_d(x,h',D_c(x,h',f'''x)))\},\{x\},\{g,f',f''',h'\}).$$

Wir stecken also offensichtlich in einer Sackgasse und würden bis zum Sankt Nimmerleinstag so weitermachen. Dabei gibt es doch eine Lösung, man nehme einfach für das Problem $(*)$: $\langle f'', \lambda x.h\rangle$. Dann reduziert obiges einfach zu dem Paar (gc,h) und wir erhalten die Gesamtlösung

$$\{\langle f, \lambda x.h\rangle, \langle g, \lambda y.D_c(y,h,g'y)\rangle\}.$$

Das sollte unser Algorithmus auch schaffen, wenn er terminiert. Dazu beginnen wir einfach mit dem zweiten Paar unseres Problems

$$\mathsf{UP} := (\{(fc,fd),(fx,gc)\},\{x\},\{f,g\}).$$

Die wird mittels der Prozedur Flex $\neq$ Flex mit der Substitution

$$\vartheta := \{\langle f, \lambda x.h\rangle, \langle g, \lambda y.D_c(y,h,g'y)\rangle\}$$

unifiziert und also ergibt sich

$$\vartheta(\mathsf{UP}) = (\{(h,h),(h,h)\},\{x\},\{h\}).$$

Fertig. Der allgemeinste Unifikator ist also jenes ϑ.

Beide Beispiele zeigen an, daß es auf die Reihenfolge der Unifikationspaare ankommen kann. Dabei scheint es so, daß es von Vorteil ist, immer die Paare mit den wenigsten Konstanten als Argumente zu betrachten, da dann auch weniger Fallunterscheidungen eingeführt werden. Diese Beobachtung läßt hoffen, daß eine Verwendung dieses Algorithmus in der Praxis durchaus sinnvoll sein kann, etwa so wie die praktische Anwendung des Algorithmus von Huet.

6 Schlußbemerkungen

In dieser Arbeit habe ich mich intensiv mit der Unifikation von λ-Termen beschäftigt. Mein Ausgangspunkt war die Musterunifikation nach D. Miller (siehe Kapitel 2.4) und der bekannte Unifikationsalgorithmus von Huet (siehe Kapitel 2.3). Ziel war es, die strengen Restriktionen von Mustern aufzuweichen, und trotzdem noch entscheidbare und effektiv lösbare Probleme zu erhalten. Dabei habe ich zwei bisher unbekannte Techniken zur Unifikation entdeckt:

Relevanteste Unifikatoren (Kapitel 3) stellen eine effiziente Möglichkeit dar, Lösungen entscheidbarer Probleme zu berechnen. Bisher waren spezielle Algorithmen für Probleme mit endlichen Lösungsmengen nicht bekannt. Insbesondere die vollständige Berechnung einer solchen Lösungsmenge war nie Ziel der Untersuchungen, man begnügte sich in der Regel mit dem Nachweis der Existenz und der Berechnung einzelner Lösungen bzw. einzelner Preunifikatoren.

Unifikation mit Fallunterscheidung (Kapitel 4) ermöglicht die punktweise Definition von Funktionen. Dadurch werden im λ-Kalkül unlösbare Probleme lösbar, Probleme mit mehreren Lösungen erhalten u. U. allgemeinste Lösungen. Allerdings ist die Unifikation mit Fallunterscheidung i. A. nicht entscheidbar (Kapitel 4.2), aber für spezielle Probleme kann sie wertvolle Dienste leisten (Kapitel 4.3 und 4.4).

Davon ausgehend stellen sich natürlich weitere Fragen.

- Was erhält man, wenn man die Linearität erweiterter Muster in Kapitel 3 nicht mehr fordert? Gibt es Möglichkeiten, die Linearität auf bestimmte Variablen zu beschränken?
- Kann man die Fallunterscheidung statt über Konstanten auch über geschlossene Terme formulieren? Kann man Terme der Gestalt $\mathsf{D}_r(s,t,t')$ mit $\mathsf{FV}(r)=\emptyset$ zulassen?
- Führt der letzte Punkt dann zu komplizierteren Mustern, etwa wenn jede flexible Variable beliebige, geschlossene Terme als Argumente besitzen kann?
- Wie kann man beide Ideen vereinigen? Kann man daraus einen Algorithmus konstruieren, der PExA unifiziert, in denen auch Konstanten als Argumente flexibler Variablen zugelassen sind?
- Gibt es Heuristiken, die einen praktischen Einsatz des nicht terminierenden Algorithmus zur Unifikation von $\mathsf{D}\mathcal{C}$-Mustern ermöglicht?
- Wie kann man diese Ideen in anderen Kalkülen, etwa in Kalkülen ohne Variablennamen oder in Kalkülen mit expliziten Substitutionen formulieren?

Diese Liste ließe sich sicherlich noch ergänzen. Einige der Fragen scheinen mir einfach zu beantworten zu sein. So glaube ich, daß die Erweiterung der Fallunterscheidung auf geschlossenen Terme keine Probleme bringt. Andere sehen sehr kompliziert aus. So bezweifele ich, ob man die Linearität in Kapitel 3 tatsächlich aufgeben kann und immer noch entscheidbare Unifikationsprobleme erhält.

Damit beende ich meine Überlegungen und danke dem Leser für seine Aufmerksamkeit.

Literatur

[1] M. Abadi, L. Cardelli, P.-L. Curien, J.-J. Levi : *Explicit substitutions.* Journal of Functional Programming 1(4):375-416, 1991.

[2] T. Altenkirch: *A formalization of the strong normalization proof for system F in LEGO.* M. Bezem and J. Groote (eds.): Typed Lambda-Calculi and Applications, LNCS 664: 13-28, 1993.

[3] F. Baader and T. Nipkow: *Term Rewriting and All That.* Cambridge University Press, 1998.

[4] H. Barendregt: *The Lambda-Calculus: Its Syntax And Semantics.*, North-Holland, Amsterdam, 1984.

[5] H.Barendregt: *Lambda Calculi with Types.* S. Abramsky, D. M. Gabbay, T. S. E. Maibaum (eds.): Handbook of Logic in Computer Science Vol. II, Oxford University Press, 1992.

[6] L. D. Baxter: *The Complexity of Unification.* PhD thesis, University of Waterloo, Ontario, Canada, 1976.

[7] Th. Becker und V. Weispfenning: *Gröbner Bases.* Springer, 1993.

[8] M. Beeson: *Unification in lambda-calculus with if-then-else.* C. Kirchner and H. Kirchner (eds.): Automated Deduction, CADE-15, LNAI 1421: 103-118, 1998.

[9] H. Benl, U. Berger, M. Seisenberger, H. Schwichtenberg, W. Zuber: *Proof theory at work: Program development in the Minlog system.* W. Bibel and P. H. Schmitt (eds.): Automated Deduction Vol.II, Kluwer, 1998.

[10] *Das Minlog System.* `http://www.minlog-system.de`

[11] N. de Bruijn: *Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem.* Indagationes Mathematicae 34(5): 381-392, 1972.

[12] M. Davies: *Unsolvable Problems.* In J. Barwise (ed): *The Handbook of Mathematical Logic.* North-Holland, 1977.

[13] G.Dowek, T.Hardin, C.Kirchner: *Higher Order Unification via Explicit Substitutions.* Rapport de recherche INRIA Lorraine Nr.2709, 1995. (http://www.inria.fr/RRRT/publications-eng.html)

[14] G.Dowek, T.Hardin, C.Kirchner, F.Pfenning: *Unification via Explicit Substitutions: The Case of Higher Order Patterns.* Rapport de recherche Nr.3591 INRIA Rocquencourt, 1998. (http://www.inria.fr/RRRT/publications-eng.html)

[15] D. Goldfarb: *The undecidability of the second-order unification problem.* Theoretical Computer Science 13: 225-230, 1981.

[16] M. J. C. Gordon, T. F. Melham: *Introduction to HOL: a theorem proving environment for higher order logic.* Cambridge University Press, 1993.

[17] J. R. Hindley: *Basic Simple Type Theory.* Cambridge Tracts in Theoretical Computer Science 42, Cambridge University Press, 1997.

[18] G. Huet: *The undecidability of unification in third order logic.* Information and Control, 22: 257-267 ,1973.

[19] G. Huet: *A unification algorithm for typed λ-calculus.* Theoretical Computer Science 1: 27-57, 1975.

[20] F. Joachimski and R. Matthes: *Short proofs of normalization for the simply-typed lambda-calculus, permutative conversions and Gödel's T.* Archive for Mathematical Logic, 2003.

[21] M. Kübler: *Unifikation von Termen höherer Stufe.* Diplomarbeit am mathematischen Institut der Universität München, 1997.

[22] R. Loader: *The Undecidability of λ-definability.* C. Anthony Andersen, M. Zeleny (eds.): Logic, Meaning and Computation: Essays in Memory of Alonzo Church. (331-342) , Kluwer, 2001.

[23] A. Martelli and U. Montanari: *An Efficient Unification Algorithm.* ACM Trans. Programming Languages and Systems 4(2): 258-282, 1982.

[24] D.Miller: *A logic programming language with λ-abstraction, function variables and simple unification.* Journal of Logic and Computation 1(4): 497-536 , 1991.

[25] D.Miller: *Unification under mixed prefixes.* Journal of Symbolic Computation Vol. 14: 321-358, 1992.

[26] J. C. Mitchell: *Foundations for Programming Languages.* MIT Press, 1996.

[27] G. Nadathur, D. Miller: *Higher Order Logic Programming.* D. Gabbay, C. Hogger, J. Robinson (eds.): Handbook of Logic in AI and Logic Programming (Vol. 5). S. 499-590, Clarendon Press Oxford, 1998.

[28] M. S. Paterson and M. N. Wegman: *Linear Unification.* Journal Computer and System Sciences 16, 158-167, 1978.

[29] L. C. Paulsen: *Isabelle: A generic Theorem Prover.* LNCS 828, Springer, 1994.

[30] G. D. Plotkin: *λ-definability and Logic Relations.* Memorandum SAI-RM-4, School of Artificial Intelligence, University of Edinburgh, 1973.

[31] G. D. Plotkin: *λ-definability in the Full Type Hierarchy.* In J. P. Seldin and J. R. Hindley (eds): Combinatory Logic,Lambda Calculus and Formalism. Academic Press, 1980.

[32] Ch. Prehofer: *Solving higher-order equations: From logic to programming.* Technical Report I9508, Technische Universität München, 1995.

[33] A. Rios: *Contributions à l'étude des λ- calculus avec des substitutions explictes.* Thèse de Doctorat d'Université, U.Paris VII, 1993.

[34] J. A. Robinson: *A Machine-Oriented Logic Based on the Resolution Principle.* Journal ACM (12): 23-41, 1965.

[35] R. Statman: *Completeness, Invariance and λ-definability.* Journal of Symbolic Logic 47(1): 17-26, 1982.

[36] A. S. Troelstra and H. Schwichtenberg: *Basic Proof Theory.* Cambridge Tracts in Theoretical Computer Science 43, Cambridge University Press, 1996.

[37] Wolfram: *The Clausal Theory Of Types.* Cambridge Tracts in Theoretical Computer Science vol. 21, Cambridge University Press, 1993.

Index

Lebenslauf von Martin Kübler

Person

Geboren am 30.08.1971 in München, ledig.

Schule

1977-1979 Grundschule an der Stuntzstraße, Bogenhausen.

1979-1981 Grundschule an der Sambergerstraße, Solln.

1981-1990 Staatliches Gymnasium Pullach.

29.06.1990 Abitur.

Militär

10.1990-09.1991 Wehrdienst. Entlassung als Obergefreiter der Reserve.

Studium

WS 1991-WS 1997 Studium der Mathematik mit Nebenfach Informatik an der Universität München.

23.12.1997 Erlangung des akademischen Grades Diplom-Mathematiker (Univ).

Forschung

07.1998-06.2001 Stipendiat des Graduiertenkollegs *Logik in der Informatik* der Deutschen Forschungsgemeinschaft (DFG).

10.2001-08.2004 Wissenschaftlicher Mitarbeiter am Mathematischen Institut der Universität München.

München, den 26.07.2004.