Higher-Order Narrowing mit definierenden Bäumen und expliziten Substitutionen

von

Hans-Georg Eßer

Diplomarbeit

in Mathematik

vorgelegt der
Mathematisch-Naturwissenschaftlichen Fakultät
– Fachbereich I –
der
Rheinisch-Westfälischen Technischen Hochschule Aachen
im September 1997

Angefertigt am
Lehr- und Forschungsgebiet Informatik II
bei
Professor Dr. rer. nat. Michael Hanus

Zweitgutachter:
Professor Dr. phil. Erich Grädel

Dankwort

Herrn Professor Dr. Michael Hanus danke ich für die Möglichkeit, als Mathematiker eine Diplomarbeit am Lehr- und Forschungsgebiet für Informatik am Lehrstuhl für Informatik II zu schreiben. Die vielen klärenden Gespräche mit ihm halfen mir bei der Bewältigung der gestellten Aufgabe; ebenso danke ich Herrn Professor Dr. Erich Grädel, der sich als Zweitgutachter aus dem Fachbereich Mathematik zur Verfügung gestellt hat.

Besonders danke ich auch Herrn Daniel Briaud, der mit mir verschiedene Aspekte seiner Veröffentlichungen per E-Mail diskutierte [Bri97a] und mir eine noch nicht erschienene Arbeit [Bri97b] zur Verfügung stellte.

Dem Institut National de Recherche en Informatique et en Automatique (INRIA) danke ich für die Zusendung eines Forschungsberichtes [ACCL91].

Für das äußerst sorgfältige Korrekturlesen dieser Arbeit und zahlreiche Anregungen danke ich Frau Dipl.-Math. Birgit Blümer.

Und schließlich möchte ich mich bei meiner Familie bedanken, die mein Studium – nicht nur finanziell – unterstützt hat.

TEX

Diese Arbeit wurde mit dem Textsatzsystem TEX von D. E. Knuth und dem Makropaket LATEX von L. Lamport geschrieben, wobei die Pakete a4, amscd, amsfonts, amsmath, amssymb, array, enumerate, fancyhdr, german, ifthen, index, isolatin1, longtable, mathsymb, theorem und wasysym zum Einsatz kamen. Eine unverzichtbare Hilfe bot der *LATEX-Begleiter* [GMS95], der Ordnung in die verwirrende Vielfalt des LATEX-Paketes brachte.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die funktional-logische Programmierung führt die Vorteile der funktionalen und der logischen Programmierung in einem gemeinsamen Konzept zusammen. Beide Sprachkonzepte sind deklarativ, d.h. ein Programm enthält keine algorithmische Beschreibung des Lösungsweges wie in prozeduralen Sprachen sondern eine Beschreibung des Problemes: funktionale oder kausale Abhängigkeiten werden in einer mathematischen Notation formuliert, und die Semantik eines Programmes kann über diese Notation leichter definiert werden als dies bei prozeduralen Programmen der Fall ist. Dies erleichtert nicht nur die Programmierung selbst, sondern ermöglicht auch die Überprüfung der Korrektheit. In ihren *reinen* Formen sind diese Sprachen frei von Nebeneffekten, die das Verhalten eines Programmes unübersichtlich machen.

Funktionale Programmiersprachen wie Miranda, Haskell, Gofer, LISP, Scheme oder ML erlauben die Definition von Funktionen höherer Ordnung und basieren auf Termersetzungssystemen zur Reduktion funktionaler Ausdrücke. Durch verzögernde Auswertungsstrategien sind unendliche Datenstrukturen möglich, welche immer nur soweit ausgewertet werden, wie dies erforderlich ist. Die Konzepte der Abstraktion und Applikation des λ-Kalküls werden unterstützt, Pattern Matching als Alternative zur Verwendung von Test- und Selektionsoperatoren erlaubt eine vereinfachte Darstellung funktionaler Programme. Funktionen können auch rekursiv definiert werden, so daß alle μ-rekursiven Funktionen und damit alle Turing-berechenbaren Funktionen durch funktionale Programme berechenbar sind.

Logische Programmiersprachen wie Prolog verwenden Klauseln mit logischen Prädikaten und basieren auf Unifikation und dem Resolutionsprinzip von Robinson. Die mathematische Grundlage dieser Programmiersprachen ist die Prädikatenlogik erster Stufe, für die es einen Resolutionsalgorithmus gibt, der die Entscheidbarkeit der Unifikation erster Ordnung ausnutzt. Erweiterungen auf Prädikatenlogik höherer Stufen erfordern auch Unifikation höherer Ordnung, die im allgemeinen unentscheidbar ist. Die Abwesenheit von Funktionsdefinitionen erschwert eine deterministische Auswertung, da die Suche nach Lösungen für eine Anfrage nicht-deterministisch erfolgt.

In der funktional-logischen Programmierung werden diese Konzepte nun zusammengeführt. Operationale Grundlage ist hier das Narrowing, eine Übersicht gibt [Han94b], [Pre95] vergleicht eine Reihe von Narrowing-Verfahren und präsentiert verschiedene Anwendungen. Beispiele funktional-logischer Programmiersprachen sind ALF [HS91], BABEL [MR92], K-LEAF [BGL+87] und SLOG [Fri85]. Die jüngste Entwicklung in diesem Bereich ist die Programmiersprache Curry [HKMN95, Han97b], deren Narrowing-Strategie mit definierenden Bäumen [Ant92, AEH94, HP96] arbeitet. Diese Sprache wurde mittlerweile auch in einer Vorlesung über Deklarative Programmiersprachen an der RWTH Aachen eingesetzt – ein Konzept zur gleichzeitigen Einführung in die funktionale und Logikprogrammierung mit Hilfe von Curry enthält [Han97a].

Beispiel (Symbolisches Differenzieren) Ein funktional-logisches Programm, das symbolisch differenziert, ist durch

$$
\begin{array}{rcl}
\mathit{diff}(\lambda y.y, X) & \to & 1 \\
\mathit{diff}(\lambda y.\sin(F(y)), X) & \to & \cos(F(X)) * \mathit{diff}(\lambda y.F(y), X) \\
\mathit{diff}(\lambda y.\cos(F(y)), X) & \to & (-1) * \sin(F(X)) * \mathit{diff}(\lambda y.F(y), X) \\
\mathit{diff}(\lambda y.\ln(F(y)), X) & \to & \mathit{diff}(\lambda y.F(y), X)/F(X) \\
\mathit{diff}(\lambda y.F(y) + G(y), X) & \to & \mathit{diff}(\lambda y.F(y), X) + \mathit{diff}(\lambda y.G(y), X) \\
\mathit{diff}(\lambda y.F(y) * G(y), X) & \to & \mathit{diff}(\lambda y.F(y), X) * G(X) \\
 & & +\mathit{diff}(\lambda y.G(y), X) * F(X)
\end{array}
$$

gegeben. Auf die Anfrage $\lambda x.\mathit{diff}(\lambda y.\sin(F(x, y)), x) \to^{?} \lambda x.\cos(x)$ sollte als Antwort die Substitution $\sigma = \{F \mapsto \lambda x, y.y\}$ berechnet werden.

Nach jedem Narrowing-Schritt erfolgt eine implizite $\beta\eta$-Reduktion des Terms im λ-Kalkül: so wird etwa $(\lambda x.s)t$ zu $\{x \mapsto t\}s$, und die Substitution $\{x \mapsto t\}$ erfolgt implizit. Dadurch sind die β- und η-Regeln des λ-Kalküls keine Termersetzungsregeln erster Ordnung. λ-Kalküle mit *expliziten Substitutionen* ersetzen die klassischen β- und η-Regeln durch neue Regeln, so daß Terme der Form $t[s]$ entstehen, wobei die eckigen Klammern und s syntaktische Objekte sind. Es werden dann Termersetzungsregeln erster Ordnung eingeführt, mit denen die Substitution bewerkstelligt wird. Dadurch wird ein β- oder η-Reduktionsschritt zu einer Folge von Reduktionsschritten, an deren Ende wieder ein *reiner* Term steht, d.h. ein Term ohne Substitutionsanteil $[s]$. Zwei dieser Kalküle sind $\lambda\sigma$ [ACCL91, DHK95] und $\lambda\upsilon$ [Les94, BBLRD95]. Eine Einführung und Übersicht über verschiedene λ-Kalküle mit expliziten Substitutionen gibt [Ros96] – dieses Papier enthält auch eine sehr ausführliche, teilweise kommentierte Bibliographie.

Aufgabe der vorliegenden Diplomarbeit ist es, einen dieser Kalküle mit expliziten Substitutionen auf das Narrowing-Verfahren LNT [HP96] anzuwenden, um so durch das Ersetzen der impliziten Substitutionen die Grundlage für eine effiziente Implementierung zu schaffen. Wir stellen das Narrowing-Verfahren LNT$\lambda\upsilon$ vor, welches eine modifizierte Version des $\lambda\upsilon$-Kalküls verwendet und dabei äquivalent zu LNT bleibt.

Die Arbeit gliedert sich wie folgt:

- In Kapitel 2 geben wir die grundlegenden Definitionen des (einfach getypten) λ-Kalküls. Wir betrachten Unifikationsprobleme höherer Ordnung und erklären Narrowing-Verfahren.
- Kapitel 3 führt in das Narrowing-Verfahren LNT ein, das in dieser Arbeit weiterentwickelt wird.
- Kapitel 4 stellt λ-Kalküle mit expliziten Substitutionen vor: Wir betrachten die $\lambda\sigma$- und $\lambda\upsilon$-Kalküle, in denen β-Reduktion internalisiert, d.h. zu einer Reduktionsregel auf der syntaktischen Ebene gemacht wird, und geben für letzteren eine Ergänzung um eine explizite η-Regel an.
- In Kapitel 5 kombinieren wir das LNT-Verfahren mit dem $\lambda\upsilon$-Kalkül und weisen die Äquivalenz des neuen Verfahrens zu LNT nach.
- Kapitel 6 schließt die Arbeit mit einer Zusammenfassung und einem Ausblick ab.

Kapitel 2

Grundlagen

2.1 Terme

Konvention 2.1.1 (natürliche Zahlen) Wir setzen $\mathbb{N} := \{1, 2, 3, \ldots\}$. Die Menge $\{0, 1, 2, \ldots\}$ nennen wir $\mathbb{N}_0$. □

Definition 2.1.2 (Signatur, Funktionssymbole, Terme, [BA95])
Es sei Σ eine Menge von Funktionssymbolen, jedem $f \in \Sigma$ sei eine Stelligkeit $n \in \mathbb{N}_0$ zugeordnet. $\Sigma^{(n)}$ bezeichne die Menge der n-stelligen Funktionssymbole aus Σ. Dann heißt Σ eine *Signatur*.

Weiter sei $\mathcal{X}$ eine Menge von Variablen. Dann ist $\mathcal{T}(\Sigma, \mathcal{X})$ die kleinste Menge, die $\mathcal{X}$ enthält und unter Funktionsapplikation abgeschlossen ist: d.h. für jedes $f \in \Sigma^{(n)}$ und Terme $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{X})$ gilt auch $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{X})$. $\mathcal{T}(\Sigma, \mathcal{X})$ heißt Menge der Σ-*Terme*. Hat f Stelligkeit 0, schreiben wir f statt $f()$. □

Definition 2.1.3 (Positionen) Eine Position ist eine Folge $n_1. \cdots .n_k$ von Zahlen $n_i \in \mathbb{N}$ oder die leere Sequenz ε. Für einen Term t und eine Position p ist $t|_p$ definiert durch

$$t|_p := \begin{cases} t, & \text{falls } p = \varepsilon, \\ (t_i)|_{p'}, & \text{falls } p = i.p' \text{ und } t = f(t_1, \ldots, t_n), \text{ wobei } i \in \{1, \ldots, n\} \end{cases}$$

$t|_p$ heißt *Teilterm von t an Position p*. Mit $t[s]_p$ wird der Term bezeichnet, der entsteht, indem der Teilterm von t an Position p durch s ersetzt wird. □

2.2 Der einfach getypte λ-Kalkül

2.2.1 λ-Terme

Im folgenden geben wir eine kurze Zusammenfassung der wesentlichen Eigenschaften des *einfach getypten λ-Kalküls*.

Definition 2.2.1 (Typen, Typ-Konstruktor $\to$) Es sei $\mathcal{T}_0$ eine Menge von Grundtypen. Dann definieren wir induktiv die Menge der *Typen* $\mathcal{T}$ als die kleinste Menge, die $\mathcal{T}_0$ und mit $\alpha, \beta \in \mathcal{T}$ auch $(\alpha \to \beta)$ enthält. □

Konvention 2.2.2 Der Typ-Konstruktor $\to$ ist *rechtsassoziativ*, und für $(\alpha_1 \to (\alpha_2 \to \cdots (\alpha_n \to \beta) \ldots))$ schreiben wir kurz $\alpha_1, \ldots, \alpha_n \to \beta$. □

Definition 2.2.3 (Funktionssymbole, Variablen, Atome)
Sei Σ eine Signatur, jedes Symbol $f \in \Sigma$ habe einen eindeutigen Typ $\tau(f) \in \mathcal{T}$. Zu jedem Typ $\alpha \in \mathcal{T}$ existiere eine abzählbar unendliche Menge V_α von *Variablen* dieses Typs. Wir setzen $V(\mathcal{T}) := \bigcup_{\tau \in \mathcal{T}} V_\tau$. $\mathcal{A}(\mathcal{T}) := V(\mathcal{T}) \cup \Sigma$ heißt Menge der *Atome*. □

Definition 2.2.4 ((korrekt) getypte λ-Terme) Die Menge $\mathcal{L}(\mathcal{T})$ der *(korrekt) getypten λ-Terme* ist die kleinste Obermenge $\mathcal{L}$ von $\mathcal{A}(\mathcal{T})$, die folgenden Abschlußeigenschaften genügt:

1. *Applikation*: Für $e_1 \in \mathcal{L}$ vom Typ $\alpha \to \beta$ und $e_2 \in \mathcal{L}$ vom Typ α ist $(e_1 e_2) \in \mathcal{L}$ vom Typ β.
2. *Abstraktion*: Für $e \in \mathcal{L}$ vom Typ β und $x \in V_\alpha$ ist $(\lambda x.e) \in \mathcal{L}$ vom Typ $\alpha \to \beta$.

Wir bezeichnen mit $\tau(e)$ den Typ eines Terms e und schreiben auch $e : \tau$, um auszudrücken, daß e vom Typ τ ist. □

Konvention 2.2.5 Im folgenden gehen wir davon aus, daß alle auftretenden λ-Terme korrekt getypt sind, auch wenn ihre Typen nicht explizit angegeben werden. □

Konvention 2.2.6 Wir legen fest, daß Funktionsapplikation standardmäßig *linksassoziativ* ist, so daß $(\ldots((e_1 e_2) e_3) \ldots e_n)$ kurz als $(e_1 e_2 \ldots e_n)$ geschrieben werden kann. Eine Folge von λ-Abstraktionen $\lambda x_1.(\lambda x_2.(\ldots \lambda x_n.e) \ldots)$ schreiben wir als $\lambda \overline{x_n}.e$. *Klammern* werden weggelassen, wenn dabei die Bedeutung klarbleibt. Der Punkt bezieht soviel Rechtskontext wie möglich mit ein, so daß z.B. $\lambda x.stu$ als $(\lambda x.((st)u))$ interpretiert wird. □

Definition 2.2.7 (kartesische Produktypen) Für $\tau_1, \ldots, \tau_n \in \mathcal{T}$ sei $\tau_1 \otimes \cdots \otimes \tau_n$ der *kartesische Produkttyp* von τ_1 bis τ_n. $\mathcal{T}_\Pi$ sei die kleinste Menge, die $\mathcal{T}$ enthält und unter Bildung von Produkttypen abgeschlossen ist. □

Bemerkung 2.2.8 (Curry-Isomorphismus) Kartesische Typen sind kein Bestandteil des klassischen λ-Kalküls, und wir werden sie hier nicht verwenden. Dies ist aber keine Einschränkung, da wir Produkttypen nur bei mehrstelligen Funktionssymbolen benötigen, und über den *Curry-Isomorphismus*

$$c : (\tau_1 \otimes \cdots \otimes \tau_n \to \tau) \mapsto (\tau_1 \to \cdots \to \tau_n \to \tau)$$

erhalten wir zugehörige Typen, die auch *partielle Applikationen* erlauben. □

Bemerkung 2.2.9 In [Bar84], Seite 6, findet man den Hinweis, die Beobachtung, daß mehrstellige Funktionen auf einstellige reduziert werden können, stamme von Schönfinkel [Sch24]. □

Definition 2.2.10 (Unterterme) Sei $t \in \mathcal{L}(\mathcal{T})$. Dann ist die Menge $Sub(t)$ der *Unterterme* von t induktiv definiert durch:

$$Sub(t) := \begin{cases} \{t\}, & t \text{ atomar}, \\ \{t\} \cup Sub(t_1) \cup Sub(t_2), & t = (t_1\ t_2), \\ \{t\} \cup Sub(e), & t = \lambda x.e \end{cases}$$
□

Definition 2.2.11 (Variablen, Grundterm)
Sei $t \in \mathcal{L}(\mathcal{T})$. Dann ist $Var(t)$ die Menge aller in t vorkommenden Variablen. Gilt $Var(t) = \varnothing$, dann heißt t ein *Grundterm.* □

Definition 2.2.12 (Termgröße, Scope, gebunden, frei, linear)
Die *Termgröße* $|e|$ eines Terms e ist die Anzahl seiner atomaren Unterterme. Eine Variable x tritt *gebunden* im Term e auf, wenn e einen Unterterm $\lambda x.e'$ enthält. In diesem Fall heißt e' der *Gültigkeitsbereich* oder *Scope* dieses bindenden Auftretens von x. Die Menge der gebundenen Variablen von e heißt $BV(e)$. Eine Variable x tritt *frei* im Term e auf, wenn x ein Unterterm von e ist, aber nicht im Scope eines bindenden Auftretens von x auftritt. Die Menge der freien Variablen von e heißt $FV(e)$. e heißt *linear*, falls in ihm keine *freie* Variable mehrfach auftritt. □

Definition 2.2.13 (Ordnung, Sprache der Ordnung n) Für Typen $\tau \in \mathcal{T}$ ist die *Ordnung* $Ord(\tau)$ induktiv definiert als:

$$Ord(\tau) := \begin{cases} 1, & \tau \in \mathcal{T}_0 \\ \max(Ord(\alpha) + 1, Ord(\beta)), & \tau = \alpha \to \beta \end{cases}$$

Die *Ordnung* einer Funktionskonstante oder Variablen ist die Ordnung ihres Typs. Eine *Sprache der Ordnung* n ist eine Sprache, in der Funktionskonstanten höchstens Ordnung $n+1$ und (freie oder gebundene) Variablen höchstens Ordnung n haben.
Dies ist eine Verallgemeinerung der Konvention, daß ein Term erster Ordnung eine Variable oder Konstante ist, ein Term zweiter Ordnung eine Funktion von Variablen (erster Ordnung), etc. □

Konvention 2.2.14 Im folgenden stehen $\alpha, \beta, \gamma, \varphi, \tau$ stets für Typen, b, c für Konstanten eines Grundtyps, f, g, h für Konstanten eines funktionalen Typs, x, y, z für Variablen beliebigen Typs und a für Atome. Freie Variablen funktionalen Typs nennen wir F, G, H, X, Y. λ-Terme heißen e, r, s, t, u, v, w. □

2.2.2 Reduktion

Da wir im folgenden häufig den Begriff einer *Reduktion* verwenden werden, geben wir hier Definitionen der wichtigsten in diesem Zusammenhang auftretenden Begriffe.

Definition 2.2.15 (kompatibel, Kongruenz-, Äquivalenz-, Reduktionsrelation)
Es sei $\Lambda = \mathcal{L}(\mathcal{T})$ die Menge der einfach getypten λ-Terme.

1. Eine binäre Relation R auf Λ ist *kompatibel* (mit den Operationen), falls für alle $(a, a') \in R$, alle $b \in \Lambda$ und alle Variablen x gilt: $(ba, ba') \in R$, $(ab, a'b) \in R$ und $(\lambda x.a, \lambda x.a') \in R$.

2. Eine Äquivalenzrelation auf Λ ist eine binäre Relation R, die folgende drei Eigenschaften hat:

   - Reflexivität: Für alle $a \in \Lambda$: $(a, a) \in R$,
   - Symmetrie: $(a, b) \in R \Rightarrow (b, a) \in R$ und
   - Transitivität: $(a, b) \in R$ und $(b, c) \in R \Rightarrow (a, c) \in R$.

3. Eine *Kongruenzrelation* auf Λ ist eine kompatible Äquivalenzrelation.

4. Eine *Reduktionsrelation* auf Λ ist eine Relation, die kompatibel, reflexiv und transitiv ist. □

Definition 2.2.16 (Reduktionssystem) Es sei A eine nicht-leere Menge und $\rightarrow \subseteq A \times A$ eine binäre Relation auf A. Dann heißt $(A, \rightarrow)$ ein Reduktionssystem. Statt $(a, b) \in \rightarrow$ schreiben wir $a \rightarrow b$. □

Definition 2.2.17 (λ-Konversion, Redex, Normalform, Äquivalenz) Für Terme u, t und eine Variable x sei $\{x \mapsto t\}u$ der Term, der aus u entsteht, wenn man in u jedes freie Auftreten von x durch t ersetzt; $BV(t)$ sei die Menge der gebundenen Variablen in t. Dann gibt es die folgenden drei Regeln der *λ-Konversion*:

- *α-Konversion*: $y \notin FV(t) \cup BV(t) \;\Rightarrow\; (\lambda x.t) \succ_\alpha (\lambda y.(\{x \mapsto y\}t))$
- *β-Konversion*: $((\lambda x.s)t) \succ_\beta \{x \mapsto t\}s$
- *η-Konversion*: $x \notin FV(t) \;\Rightarrow\; (\lambda x.(tx)) \succ_\eta t$

Die Terme auf der linken Seite heißen *Redexe*. Ein Term t, der keinen β-Redex enthält, heißt *β-Normalform*. Entsprechend werden $\eta-$ und $\beta\eta$-Normalformen definiert.

Falls $e[s]$ ein λ-Term e mit Unterterm s an einer festen Stelle ist und $e[t]$ den Term bezeichnet, der aus $e[s]$ hervorgeht, indem man diesen Unterterm s durch t ersetzt, wobei s und t vom gleichen Typ sind, dann können wir die Relation $\rightarrow_\alpha$ definieren durch

$$e[s] \rightarrow_\alpha e[t] \iff s \succ_\alpha t.$$

Genauso definieren wir Relationen $\rightarrow_\beta$ und $\rightarrow_\eta$.

Außerdem setzen wir $\rightarrow_{\beta\eta} := \rightarrow_\beta \cup \rightarrow_\eta$ (d.h. $e_1 \rightarrow_{\beta\eta} e_2 \iff e_1 \rightarrow_\beta e_2 \vee e_1 \rightarrow_\eta e_2$). Dann erklären wir für jede dieser Relationen $\rightarrow$ auf übliche Weise die *symmetrische Hülle* $\leftrightarrow$, die *transitive Hülle* $\rightarrow^+$, und die symmetrische, reflexive und transitive Hülle $\overset{*}{\longleftrightarrow}$. Die Relationen $\overset{*}{\longleftrightarrow}_\beta, \overset{*}{\longleftrightarrow}_\eta$ und $\overset{*}{\longleftrightarrow}_{\beta\eta}$ heißen β-, η- und *βη-Äquivalenz*. □

Man zeigt leicht, daß diese Regeln der λ-Konversion *typerhaltend* sind.

Wir übernehmen die Definitionen der Begriffe *konfluent* und *lokal konfluent* aus [BA95].

Definition 2.2.18 (konfluent, lokal konfluent) Es sei $(A, \rightarrow)$ ein Reduktionssystem. $\rightarrow$ heißt *konfluent*, falls

$$\forall x, y_1, y_2 \in A \text{ mit } x \rightarrow^* y_1, x \rightarrow^* y_2 : \exists z \in A : \; y_1 \rightarrow^* z, y_2 \rightarrow^* z.$$

$\rightarrow$ heißt *lokal konfluent*, falls

$$\forall x, y_1, y_2 \in A \text{ mit } x \rightarrow y_1, x \rightarrow y_2 : \exists z \in A : \; y_1 \rightarrow^* z, y_2 \rightarrow^* z.$$

$$
\begin{tikzcd}
 & y_1 \arrow[dr, "*"] & \\
x \arrow[ur, "*"] \arrow[dr, "*"'] & & z \\
 & y_2 \arrow[ur, "*"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & y_1 \arrow[dr, "*"] & \\
x \arrow[ur] \arrow[dr] & & z \\
 & y_2 \arrow[ur, "*"'] &
\end{tikzcd}
$$

konfluent lokal konfluent

□

Bei der lokalen Konfluenz fordern wir also nur, daß Terme nach einem Schritt des „Auseinandergehens“ wieder zusammengeführt werden können.

Definition 2.2.19 (stark normalisierend)
Ein Reduktionssystem $(A, \to)$ heißt *stark normaliserend* oder *terminierend*, falls für jedes $x \in A$ jede Reduktionsfolge $(x \to x_1 \to x_2 \to \ldots)$ endlich ist, d.h. es gibt ein $n \in \mathbb{N}$, so daß x_n keinen $\to$-Redex enthält.

Zu der Eigenschaft, daß für ein festes $x \in A$ jede Reduktionsfolge $(x \to x_1 \to x_2 \to \ldots)$ endlich ist, sagen wir auch, daß x *stark $\to$-normalisierend* ist. □

Bemerkung 2.2.20 (Lemma von Newman, [BA95, New42])
Für terminierende Reduktionssysteme, d.h. solche, in denen es keine unendlichen Folgen von Regelanwendungen gibt, fallen die Begriffe Konfluenz und lokale Konfluenz zusammen. □

Definition 2.2.21 (Church-Rosser-Eigenschaft) Es sei $(A, \to)$ ein Reduktionssystem. $\to$ hat die *Church-Rosser-Eigenschaft*, falls

$$\forall x, y \in A \text{ mit } x \overset{*}{\longleftrightarrow} y : \ \exists z \in A : \ x \to^* z \text{ und } y \to^* z.$$

$$
\begin{tikzcd}
x \arrow[dd, leftrightarrow, "*"] \arrow[dr, "*"] & \\
 & z \\
y \arrow[ur, "*"'] &
\end{tikzcd}
$$

Church-Rosser

□

Definition 2.2.22 (substituierbar) Ein Term s heißt für x in t *substituierbar*, wenn für jeden Unterterm $\lambda y.t'$ von t gilt: $y \in FV(s) \Rightarrow x \notin FV(t')$. □

Konvention 2.2.23 Im folgenden betrachten wir nur Terme, für die die Mengen der freien Variablen und der gebundenen Variablen disjunkt sind; alle Vergleiche von λ-Termen werden modulo α-Konversion erfolgen (d.h. wir interessieren uns nicht für simple Umbenennungen von Variablen). Dies erlaubt uns die Darstellung von λ-Bindern mit generischen Variablen $x_1, \ldots, x_k$, wo dies keine Verwirrung erzeugt. □

Definition 2.2.24 (getypter β- bzw. $\beta\eta$-Kalkül) Unter dem *getypten β-Kalkül* bzw. *getypten $\beta\eta$-Kalkül* verstehen wir den Kalkül, in dem nur die β-Regel bzw. β- und η-Regel zugelassen sind. □

Es folgen zwei wesentliche Eigenschaften dieses Kalküls.

Satz 2.2.25 (Starke Normalisierung, [HS86])
Jede Folge von $\beta\eta$-Reduktionen ist endlich. (D.h., daß ausgehend von einem Term t_0 für jede Folge von Reduktionsschritten $t_0 \rightarrow_{\beta\eta} t_1 \rightarrow_{\beta\eta} \cdots \rightarrow_{\beta\eta} t_i \rightarrow_{\beta\eta} \cdots$ ein n existiert, so daß t_n keinen $\beta\eta$-Redex mehr enthält, und damit bricht die Folge nach dem n-ten Schritt ab.) □

Wir werden im weiteren Verlauf dieser Arbeit Kalküle vorstellen, die die β- und η-Regeln durch neue Regeln mit *expliziten Substitutionen* ersetzen. Dabei interessiert uns dann, ob die Eigenschaft der starken Normalisierung erhalten bleibt.

Definition 2.2.26 (Bewahrung der starken Normalierung) Ein Reduktionssystem $(A, \rightarrow)$ (mit einer Grundmenge A, auf der β- und η-Reduktion definiert sind) *bewahrt starke Normalisierung* (*PSN: preserves strong normalisation*), falls jeder Term, der stark $\rightarrow_\beta$-normalisierend ist, auch stark $\rightarrow$-normalisierend ist.

Satz 2.2.27 (Church, Rosser, [CR35, HS86])
$\rightarrow_{\beta\eta}$ hat die Church-Rosser-Eigenschaft. (D.h.: Wenn zwei Terme $\beta\eta$-äquivalent sind, dann können sie auf den gleichen Term reduziert werden.) □

Wir erhalten daraus das Resultat, daß es zu jedem Term eine (bis auf α-Konversion) eindeutige $\beta\eta$-Normalform gibt. Man kann zwei Terme auf Äquivalenz testen, indem man ihre Normalformen ausrechnet und vergleicht.

Die beiden obigen Sätze gelten entsprechend für β- bzw. η-Konversion.

Definition 2.2.28 (β-Normalform) Für einen Term t sei $t{\downarrow_\beta}$ seine eindeutige β-Normalform. □

Dann erhalten wir als Anwendung der obigen Sätze: $s \overset{*}{\longleftrightarrow}_\beta t \iff s{\downarrow_\beta} = t{\downarrow_\beta}$.

Definition 2.2.29 (Kopfsymbol, atomar)
In einem Term $t = \lambda x_1 \ldots x_n.(a\ e_1 \ldots e_m)$ (mit Termen $a, e_1, \ldots, e_n$) heißt a das *Kopfsymbol* und wird mit $Head(t)$ bezeichnet. Ein Unterterm a in einem Term t heißt *atomar*, falls a eine Funktionskonstante, eine gebundene Variable oder eine in t freie Variable ist. □

Konvention 2.2.30 Wir setzen i.a. voraus, daß Terme in β-Normalform sind, wenn nichts anderes erwähnt wird. Jeder Term in β-Normalform kann in der Form $\lambda x_1 \ldots x_n(ae_1 \ldots e_m)$ (für ein $n \geq 0$) dargestellt werden, wobei der Kopf a *atomar* ist und die Terme $e_1, \ldots, e_m$ ebenfalls diese Form haben. In Analogie zur Notation der Terme erster Ordnung schreiben wir einen solchen Term $\lambda x_1 \ldots x_n.a(e_1, \ldots, e_m)$. □

Definition 2.2.31 (Binder, Matrix) In einem λ-Term $\lambda\overline{x_n}.a(\overline{e_m})$ heißen $\lambda\overline{x_n}$ *Binder* und $a(\overline{e_m})$ *Matrix* des Terms. □

Definition 2.2.32 (starr, flexibel) Ein Term, dessen Kopf eine Funktionskonstante oder eine gebundene Variable ist, heißt ein *starrer* Term. Ist der Kopf eine freie Variable, heißt der Term *flexibel.* □

Beispiel 2.2.33 Der Term $\lambda x.F(\lambda y.y(x, a), c)$ ist flexibel, aber sein Unterterm $\lambda y.y(x, a)$ ist starr.

Lemma 2.2.34 ([Bar84])
Für zwei Terme s und t gilt:

$$s \rightarrow^*_{\beta\eta} t \iff \exists u\ s \rightarrow^*_{\beta} u \rightarrow^*_{\eta} t.$$ □

Dies ermöglicht uns, in zwei Schritten auf $\beta\eta$-Äquivalenz zu testen: (1) Reduktion der Terme auf β-Normalform und (2) Test dieser Normalformen auf η-Äquivalenz ($s \overset{*}{\longleftrightarrow}_{\beta\eta} t \iff s{\downarrow_\beta} \overset{*}{\longleftrightarrow}_{\eta} t{\downarrow_\beta}$). Somit läßt sich η-Konversion „ausfaktorisieren", indem wir η-Äquivalenzklassen von Termen betrachten. Im folgenden suchen wir nach einer Darstellung dieser Klassen durch kanonische Vertreter.

Definition 2.2.35 (η-expandiert) Sei $e = \lambda\overline{x_n}.a(\overline{e_m})$ ein Term in β-Normalform vom Typ $\alpha_1, \ldots, \alpha_n, \alpha_{n+1}, \ldots, \alpha_{n+k} \rightarrow \beta$ mit Grundtyp $\beta \in \mathcal{T}_0$. Die *η-expandierte Form* von e, bezeichnet mit $e{\uparrow^\eta}$, erhält man durch Hinzunahme k neuer Variablen passenden Typs zu Binder und Matrix des Terms und rekursives Anwenden der selben Entwicklung auf die Unterterme, so daß man

$$e{\uparrow^\eta} = \lambda x_1 \ldots x_n x_{n+1} \ldots x_{n+k}.a(e_1{\uparrow^\eta}, \ldots, e_m{\uparrow^\eta}, x_{n+1}{\uparrow^\eta}, \ldots, x_{n+k}{\uparrow^\eta})$$

erhält, wo $\tau(x_{n+i}) = \alpha_{n+i}$ für $1 \leq i \leq k$. □

Definition 2.2.36 (lange $\beta\eta$-Normalform) Die *lange $\beta\eta$-Normalform* $t{\updownarrow^\eta_\beta}$ eines Terms t ist die η-expandierte Form der β-Normalform von t. □

Die η-expandierte Form ist die Normalform unter der inversen Operation zur η-Reduktion (so daß $e{\uparrow^\eta} \rightarrow^*_\eta e$) und ist *nur* definiert, wenn e bereits in β-Normalform ist. Man kann zeigen, daß in η-expandierter Form jedes Atom auf so viele Argumente angewandt wird, wie sein Typ erlaubt, und daß die Typen der Matrizen aller Unterterme Grundtypen sind. Diese Form ist nützlicher als die η-Normalform, weil so die Typen des Terms und aller Unterterme expliziter sind. Daher ist es eine angenehme syntaktische Konvention für die Darstellung der Kongruenzklassen aller Terme, die modulo der η-Regel gleich sind. Durch Terminduktion kann man zeigen, daß diese expandierten Terme immer existieren und (bis auf α-Konversion) eindeutig sind, so daß für zwei Terme s und t in β-Normalform gilt: $s \xleftrightarrow{*}_{\eta} t \iff s{\uparrow^{\eta}} = t{\uparrow^{\eta}}$. Damit erhält der Satz von Church und Rosser die folgende Form:

Satz 2.2.37 ([Hue76])
Für Terme s und t gilt: $s \xleftrightarrow{*}_{\beta\eta} t \iff s{\updownarrow^{\eta}_{\beta}} = t{\updownarrow^{\eta}_{\beta}}$. □

Definition 2.2.38 ($\mathcal{L}_{exp}, \mathcal{L}_{\eta}$) $\mathcal{L}_{exp}$ sei die Menge aller η-expandierten Formen, d.h. $\mathcal{L}_{exp} := \{\, e{\updownarrow^{\eta}_{\beta}} \mid e \in \mathcal{L} \,\}$.
$\mathcal{L}_{\eta}$ sei die kleinste Teilmenge von $\mathcal{L}$, die $\mathcal{L}_{exp}$ enthält und unter Funktionsanwendung und λ-Abstraktion abgeschlossen ist, d.h. für $e_1, e_2 \in \mathcal{L}_{\eta}$ sind auch $(e_1 e_2)$ und $\lambda x.e_1$ in $\mathcal{L}_{\eta}$. □

Die wesentlichen Eigenschaften von $\mathcal{L}_{exp}$ und $\mathcal{L}_{\eta}$, die uns erlauben, uns auf η-expandierte Formen zu beschränken, sind:

Lemma 2.2.39 (Abschlußeigenschaften, [Hue76])
Für jede Variable x und jedes Paar e, e' von Termen passenden Typs gilt:

1. $e, e' \in \mathcal{L}_{exp} \implies (\lambda x.e) \in \mathcal{L}_{exp}$ *und* $(ee'){\downarrow_{\beta}} \in \mathcal{L}_{exp}$;
2. $e \in \mathcal{L}_{\eta} \implies e{\downarrow_{\beta}} \in \mathcal{L}_{exp}$;
3. $e, e' \in \mathcal{L}_{\eta} \implies (\lambda x.e) \in \mathcal{L}_{\eta}$ *und* $(ee') \in \mathcal{L}_{\eta}$;
4. $e \in \mathcal{L}_{\eta}, e \rightarrow^{*}_{\beta} e' \implies e' \in \mathcal{L}_{\eta}$;
5. $e, e' \in \mathcal{L}_{\eta} \implies \{x \mapsto e\}e' \in \mathcal{L}_{\eta}$. □

Diese *Abschlußeigenschaften* von $\mathcal{L}_{\eta}$ (von denen nicht alle von der Menge der η-Normalformen erfüllt werden) erlauben formal, daß wir die η-Regel im folgenden Abschnitt über Unifikation implizit lassen, indem wir eine Methode der Unifikation höherer Ordnung in der Sprache $\mathcal{L}_{\eta}$ entwickeln, und daß wir explizit β-Konversion nur als Berechnungsregel betrachten.

Wir formalisieren nun den generellen Begriff der *Substitution* von λ-Termen für freie Variablen im $\beta\eta$-Kalkül. Danach zeigen wir, wie diese über $\mathcal{L}_{exp}$ spezialisiert werden kann.

Definition 2.2.40 (Substitution, Träger, eingeführte Variablen)
Eine *Substitution* ist eine (totale) Funktion $\sigma : V \to \mathcal{L}$, so daß $\sigma(x) \neq x$ nur für endlich viele $x \in V$ gilt, und σ typerhaltend ist, d.h. $\tau(\sigma(x)) = \tau(x)$ für alle $x \in V$.
Ist σ eine Substitution, dann heißt $Dom(\sigma) := \{x \mid \sigma(x) \neq x\}$ der *Träger* (*domain*, *support*) von σ. Die Substitution mit leerem Träger heißt *Identitätssubstitution* oder einfach *Identität* und wird mit Id bezeichnet. Die Menge der von σ *eingeführten Variablen* ist $Rng(\sigma) := I(\sigma) := \bigcup_{x \in Dom(\sigma)} FV(\sigma(x))$ (*range*). □

Substitutionen sind also totale Funktionen, die fast überall trivial sind (d.h. nur auf einer endlichen Menge von Variablen nicht-trivial).

Wenn der Träger einer Substitution σ die Menge $\{x_1, \ldots, x_n\}$ ist und $t_i = \sigma(x_i)$, dann schreiben wir $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ und $\sigma(u) = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}u$.

Definition 2.2.41 (umbenennend, Einschränkung)
Eine Substitution ρ heißt *umbenennend weg von* W, falls $\rho(x)$ für jedes $x \in Dom(\rho)$ eine Variable ist, $I(\rho) \cap W = \varnothing$ und für $x, y \in Dom(\rho)$ gilt: $\rho(x) \overset{*}{\underset{\eta}{\longleftrightarrow}} \rho(y) \implies x = y$. Wenn W unwichtig ist, heißt ρ einfach *umbenennend*. Die *Einschränkung* einer Substitution σ auf eine Menge W', bezeichnet als $\sigma|_{W'}$, ist die Substitution σ', für die gilt: $\sigma' = \sigma$ auf W', $\sigma' = Id$ sonst.
Statt „umbennende Substitution“ sagen wir auch kurz *Umbenennung*. □

Definition 2.2.42 (Fortsetzung $\widehat{\sigma}$) Sei $\sigma_{-x} := \sigma|_{Dom(\sigma)\setminus\{x\}}$. Für jede Substitution σ definiere die *Fortsetzung* $\widehat{\sigma}$ durch:

1. (1) $\widehat{\sigma}(x) = \sigma(x), \forall x \in V$;
2. (2) $\widehat{\sigma}(a) = a, \forall a \in \Sigma$;
3. (3) $\widehat{\sigma}(\lambda x.e) = \lambda x.\widehat{\sigma_{-x}}(e)$;
4. (4) $\widehat{\sigma}((e_1 e_2)) = (\widehat{\sigma}(e_1)\widehat{\sigma}(e_2))$. □

Somit läßt eine Substitution außer den *freien* Variablen eines Terms alles fest. Im folgenden werden wir σ und $\widehat{\sigma}$ identifizieren.

Man beachte: Durch unsere Annahme, daß die Mengen der freien und gebundenen Variablen stets disjunkt sind, kann es durch Anwendung einer Substitution nie zu einem „*variable capture*“[1] kommen.

Man kann leicht zeigen, daß der Typ eines Terms durch Substitutionen nicht geändert wird.

[1] Variable capture tritt auf, wenn nach dem Anwenden der Substitution eine ursprünglich freie Variable gebunden wird: in $\{y \mapsto x\}\lambda x.y$ darf die freie Variable y nicht durch x ersetzt werden.

Bemerkung 2.2.43 Es ist wichtig, daß mit $\sigma(e)$ stets das Resultat nach Anwenden der Substitution σ auf e *ohne* β-Reduktion gemeint ist; mit $\sigma(e)\downarrow_\beta$ bezeichnen wir das Resultat von Anwendung der Substitution und anschließender β-Reduktion. Diese unübliche Trennung zwischen Substitution und anschließender β-Reduktion ist nützlich, da wir später die genauen Effekte dieser beiden Operationen auf λ-Terme untersuchen wollen. □

Definition 2.2.44 (Vereinigung, Komposition zweier Substitutionen) Die *Vereinigung* der zwei Substitutionen σ und θ mit disjunkten Trägern ($Dom(\sigma) \cap Dom(\theta) = \emptyset$) ist definiert durch:

$$(\sigma \cup \theta)(x) := \begin{cases} \sigma(x), & x \in Dom(\sigma) \\ \theta(x), & x \in Dom(\theta) \\ x, & \text{sonst} \end{cases}$$

Die *Komposition* (*Hintereinanderausführung*) von σ und θ ist die Substitution $\sigma \circ \theta$, die durch $\sigma \circ \theta(x) := \widehat{\theta}(\sigma(x))$ definiert ist. *Achtung*: Die Schreibweise ist anders als gebräuchlich: In $\sigma_1 \circ \cdots \circ \sigma_n(x)$ wird die am weitesten links stehende Substitution σ_1 zuerst ausgeführt. □

Definition 2.2.45 (gleich über W, allgemeiner über W, unabhängig) Es sei W eine Menge von Variablen. Zwei Substitutionen σ, θ heißen *gleich über* W, in Zeichen $\sigma = \theta\ [W]$, falls $\forall x \in W, \sigma(x) = \theta(x)$. σ und θ heißen *β-gleich über* W, in Zeichen $\sigma =_\beta \theta\ [W]$, falls $\forall x \in W, \sigma(x) \xleftrightarrow{*}_\beta \theta(x)$ bzw. $\sigma(x)\downarrow_\beta = \theta(x)\downarrow_\beta$. Die Relationen $=_\eta$ und $=_{\beta\eta}$ werden entsprechend unter Nutzung von $\xleftrightarrow{*}_\eta$ und $\xleftrightarrow{*}_{\beta\eta}$ definiert.
σ heißt *allgemeiner als θ über W*, in Zeichen $\sigma \leq \theta\ [W]$, falls es eine Substitution η gibt, so daß $\theta = \sigma \circ \eta\ [W]$, und wir definieren $\sigma \leq_\beta \theta\ [W]$, falls es eine Substitution η' gibt, so daß $\theta =_\beta \sigma \circ \eta'\ [W]$. Analog werden $\leq_\eta$ und $\leq_{\beta\eta}$ definiert. Wenn W die Menge aller Variablen ist, lassen wir das „$[W]$" weg.
σ und θ heißen *unabhängig*, falls weder $\sigma \leq_{\beta\eta} \theta$ noch $\theta \leq_{\beta\eta} \sigma$. □

Der Vergleich von Substitutionen modulo β-, η- und $\beta\eta$-Konversion wird durch das folgende Lemma gerechtfertigt, welches durch einfache Terminduktion bewiesen werden kann:

Lemma 2.2.46 ([SG89])
Es seien σ und θ zwei beliebige Substitutionen, so daß entweder $\sigma =_\beta \theta$, $\sigma =_\eta \theta$ oder $\sigma =_{\beta\eta} \theta$. Dann gilt für jeden Term u (jeweils) $\sigma(u) \underset{\beta}{\overset{*}{\longleftrightarrow}} \theta(u)$, $\sigma(u) \underset{\eta}{\overset{*}{\longleftrightarrow}} \theta(u)$ bzw. $\sigma(u) \underset{\beta\eta}{\overset{*}{\longleftrightarrow}} \theta(u)$. □

Definition 2.2.47 (normalisiert) Eine Substitution θ heißt *normalisiert*, falls für jede Variable $x \in Dom(\theta)$ gilt: $\theta(x) \in \mathcal{L}_{exp}$. □

O.B.d.A. können wir annehmen, daß es in keiner normalisierten Substitution eine Bindung $x \mapsto x\uparrow^\eta$ für eine Variable x gibt. Eine normalisierte, umbenennende Substitution hat die Form $\{x_i \mapsto y_1\uparrow^\eta, \ldots, x_n \mapsto y_n\uparrow^\eta\}$; wendet man diese Substitution an und β-reduziert dann, so ist der Effekt eine Umbenennung der Variablen $x_1, \ldots, x_n$ in $y_1, \ldots, y_n$. Das folgende Korollar von Lemma 2.15 zeigt, warum es sinnvoll ist, normalisierte Substitutionen zu verwenden:

Korollar 2.2.48 ([SG89]) *Ist θ eine normalisierte Substitution und $e \in \mathcal{L}_{exp}$, dann gilt $\theta(e) \in \mathcal{L}_\eta$ und $\theta(e)\downarrow_\beta \in \mathcal{L}_{exp}$.* □

Wenn σ und θ normalisiert sind, dann gilt $\sigma =_{\beta\eta} \theta \iff \sigma = \theta$. Wenn θ zu θ' normalisiert wird, dann gilt $\theta' =_{\beta\eta} \theta$.

Konvention 2.2.49 Ab jetzt nehmen wir i.a. an, daß Substitutionen normalisiert sind. Dadurch können wir Substitutionen modulo η-Äquivalenz vergleichen ($\rightarrow$ rausfaktorisieren), so daß wir z.B. statt $\leq_{\beta\eta}$ $\leq_\beta$ verwenden können.
Die Komposition zweier normalisierter Substitutionen ist nicht automatisch normalisiert, z.B. ist $\{F \mapsto \lambda x.G(a)\} \circ \{G \mapsto \lambda y.y\} = \{F \mapsto \lambda x.(\lambda y.y)a, G \mapsto \lambda y.y\}$, und nicht $= \{F \mapsto \lambda x.a, G \mapsto \lambda y.y\}$. □

Definition 2.2.50 (idempotent) Eine Substitution σ heißt *idempotent*, falls $\sigma \circ \sigma =_{\beta\eta} \sigma$. □

Eine hinreichende Bedingung für Idempotenz gibt das

Lemma 2.2.51 ([SG89])
Eine Substitution σ ist idempotent, wenn $I(\sigma) \cap Dom(\sigma) = \varnothing$. □

Daß wir o.B.d.A. nur mit idempotenten Substitutionen arbeiten können, besagt das folgende Resultat, welches zeigt, daß jede Substitution (bis auf Umbenennungen) zu einer idempotenten Substitution äquivalent ist.

Lemma 2.2.52 ([Sny88])
Für jede Substitution σ und eine Menge von Variablen $W \supset Dom(\sigma)$ gibt es eine idempotente Substitution σ', so daß $Dom(\sigma) = Dom(\sigma'), \sigma \leq_{\beta\eta} \sigma'$ und $\sigma' \leq_{\beta\eta} \sigma$ $[W]$. □

Definition 2.2.53 (Multimenge, Vereinigung von Multimengen)
Eine *Multimenge* über einer Menge A ist eine ungeordnete Folge von Elementen von A, in der ein Element auch mehrmals auftreten darf. Formaler ist eine Multimenge über A eine Funktion $M : A \rightarrow \mathbb{N}_0$, so daß ein Element $a \in A$ exakt n-mal in M auftritt, falls $M(a) = n$. a gehört nicht zu M, wenn $M(a) = 0$, und wir schreiben $a \in M \iff M(a) > 0$. Die *Vereinigung* von zwei Multimengen M_1, M_2, in Zeichen $M_1 \cup M_2$, ist definiert durch $(M_1 \cup M_2)(a) := M_1(a) + M_2(a)$. □

Um Verwechslung von Mengen und Multimengen zu vermeiden, werden wir immer genau angeben, wann ein Objekt eine Multimenge sein soll. Man beachte, daß Mengen-Vereinigung und Multimengen-Vereinigung unterschiedliche Begriffe sind, da z.B. für eine nichtleere Multimenge A gilt: $A \cup A \neq A$.

2.3 Unifikation

2.3.1 Einführung

Die Unifikation beschäftigt sich mit dem Problem, zu zwei Termen e_1, e_2 eine Substitution σ zu finden, so daß $\sigma(e_1)$ und $\sigma(e_2)$ gleich modulo einer Gleichheitstheorie sind. Wir präsentieren den Ansatz aus [SG89], in dem – ausgehend von λ-Termen erster Ordnung – ein Transformationssystem zur Unifikation von *einfach getypten* [Chu40] λ-Termen höherer Ordnung hergeleitet wird.

Beispiel 2.3.1 Seien $e_1 = f(x, f(h(x, g(x)), x'))$, $e_2 = f(x, f(h(f(y), z), y'))$. Wir schreiben diese beiden Terme als Termpaar

$$\langle f(x, f(h(x, g(x)), x')), f(x, f(h(f(y), z), y'))\rangle.$$

Nun haben beide Terme das Funktionssymbol f als Kopfsymbol. Eine Substitution kann dieses Symbol nicht ändern, also gilt: θ ist genau dann ein Unifikator für das obige Paar, wenn θ paarweise die Unterterme (hier: Argumente von f) unifiziert, also

$$\langle x, x\rangle, \langle f(h(x, g(x)), x'), f(h(f(y), z))\rangle, \langle x', y'\rangle.$$

Aus diesem Beispiel ergibt sich die generelle Transformationsregel

$$\{\ \langle f(u_1, \ldots, u_n), f(v_1, \ldots, v_n)\rangle\ \} \cup S \implies \{\ \langle u_1, v_1\rangle, \ldots, \langle u_n, v_n\rangle\ \} \cup S,$$

die wir *Termzerlegung* nennen. Wenn wir diese Regel noch zweimal anwenden, erhalten wir

$$\langle x, x\rangle, \langle x, f(y)\rangle, \langle g(x), z\rangle, \langle x', y'\rangle.$$

Nun ist das Paar $\langle x, x\rangle$ bereits unifiziert, und wir können auf diese Information verzichten. Es ergibt sich

$$\langle x, f(y)\rangle, \langle g(x), z\rangle, \langle x', y'\rangle.$$

Die entsprechende allgemeine Regel ist

$$\{\ \langle u, u\rangle\ \} \cup S \implies S$$

Diese beiden Transformationen vereinfachen ein System, ohne die Menge der Lösungen zu beeinflussen: Die Lösungsmenge ist invariant unter diesen Transformationen.

Jeder mögliche Unifikator muß x auf einen Term abbilden, der mit dem Funktionssymbol f anfängt. Mit anderen Worten: die Bindung für x wird die Form $f(t)$ für einen Term t haben. Wir führen nun eine partielle Bindung für x ein, da wir noch nicht die endgültige Bindung kennen, wir ergänzen also $\langle x, f(x_1)\rangle$ mit einer neuen Variablen x_1:

$$\langle x, f(x_1)\rangle, \langle x, f(y)\rangle, \langle g(x), z\rangle, \langle x', y'\rangle.$$

Nun eliminieren wir x im Rest des Systems, indem wir überall x durch $f(x_1)$ ersetzen:

$$\langle x, f(x_1)\rangle, \langle f(x_1), f(y)\rangle, \langle g(f(x_1)), z\rangle, \langle x', y'\rangle.$$

Nach einem weiteren Termzerlegungsschritt erhalten wir

$$\langle x, f(x_1)\rangle, \langle x_1, y\rangle, \langle g(f(x_1)), z\rangle, \langle x', y'\rangle.$$

Insgesamt nennen wir dies einen *Imitationsschritt*. Die allgemeine Imitationsregel zum partiellen Auflösen nach einer Variablen ist:

Wenn x im Term $f(t_1, \ldots, t_n)$ nicht auftritt, dann

$$\begin{aligned}&\{\langle x, f(t_1, \ldots, t_n)\rangle\} \cup S\\ &\qquad \Longrightarrow \{\langle x, f(y_1, \ldots, y_n)\rangle, \langle f(y_1, \ldots, y_n), f(t_1, \ldots t_n)\rangle\} \cup S',\end{aligned}$$

wo die y_i neue Variablen sind, die nirgends auftauchen und $S' = \{x \mapsto f(y_1, \ldots, y_n)\}S$. (Bemerkung: Wenn x in $f(t_1, \ldots, t_n)$ auftauchen würde, wäre das System nicht unifizierbar.) Damit diese Regel nicht mehrfach angewendet wird, erlauben wir ihre Anwendung nur, wenn x in S vorkommt (vergleiche Definition 2.3.7).

Allgemein: Ähnlich dem Einsetzungsverfahren bei Gleichungssystemen brauchen wir eine Methode, um unser System nach einzelnen Variablen aufzulösen: Wenn wir ein Paar $\langle x, t\rangle$ haben, in dem x in t nicht vorkommt, dann können wir in allen anderen Paaren x durch t ersetzen:

$$\{\langle x, t\rangle\} \cup S \Longrightarrow \{\langle x, t\rangle\} \cup \{x \mapsto t\}S$$

wo $\{x \mapsto t\}S$ erhalten wird, indem auf jeden Term in S die Substitution $\{x \mapsto t\}S$ angewendet wird. Wir nennen diese Regel *Variablenelimination* und erhalten in unserem Beispiel

$$\langle x, f(y)\rangle, \langle x_1, y\rangle, \langle g(f(y)), z\rangle, \langle x', y'\rangle.$$

Wir sagen: das Paar $\langle x, t\rangle$ ist in einem System S gelöst, wenn x weder in t noch in S auftritt. In diesem Sinne ist das letzte System gelöst, da alle Paare gelöst sind. Als Unifikator können wir nun $\{x \mapsto f(y), z \mapsto g(f(y)), x' \mapsto y'\}$ angeben.

Bis hier haben wir nur Terme erster Ordnung behandelt. Im wesentlichen wollen wir aber Terme höherer Ordnung unifizieren: dies sind Terme, bei denen Variablen auch einen funktionalen Typ haben können. Dazu benutzen wir den λ-Kalkül.

Beispiel 2.3.2 Wir betrachten $S = \{\langle F(f(a)), f(F(a))\rangle\}$, wobei F eine funktionale Variable ist (z.B. $int \to int$). Ein Unifikator ist $\theta = \{F \mapsto \lambda x.f(x)\}$:

$$\theta(F(f(a))) = (\lambda x.f(x))f(a) \to_\beta f(f(a))\,{}_\beta\!\leftarrow f((\lambda x.f(x))a) = \theta(f(F(a))).$$

Es gibt aber noch weitere Unifikatoren, z.B. $\theta = \{F \mapsto \lambda x.f^k(x)\}$ für $k \geq 0$.

Im Fall erster Ordnung haben wir partielle Bindungen der Form $\{x \mapsto f(y_1, \ldots, y_n)\}$ betrachtet, wo $y_1, \ldots, y_n$ Variablen (erster Ordnung) waren. Wir müssen nun die Imitationsbindung $f(y_1, \ldots, y_n)$ auf den Fall höherer Ordnung verallgemeinern. Dazu betrachten wir partielle Bindungen der Form

$$\{F \mapsto \lambda x_1 \ldots x_k.a(Y_1(x_1, \ldots, x_k), \ldots, Y_n(x_1, \ldots, x_k))\},$$

wo $Y_1, \ldots Y_n$ Variablen (höherer Ordnung) sind und a atomar (d.h. Konstante oder Variable) ist. Dabei kann a auch eine funktionale Variable sein, und an die Stelle der y_i treten Ausdrücke $Y_i(x_1, \ldots, x_k)$, da die Unterterme auch Funktionen in $x_1, \ldots x_k$ sein können.

In unserem zweiten Beispiel hätte eine partielle Bindung für F, die das Symbol f imitiert, die Form $\lambda x.f(Y(x))$, so daß wir $\{\langle F(f(a)), f(F(a))\rangle\}$ in

$$\{\langle F, \lambda x.f(Y(x))\rangle, \langle f(Y(f(a))), f(f(Y(a)))\rangle\}$$

transformieren würden. (Hierbei haben wir bereits einen β-Reduktionsschritt angewandt.) Nach Termzerlegung ergibt sich

$$\{\langle F, \lambda x.f(Y(x))\rangle, \langle Y(f(a)), f(Y(a))\rangle\}.$$

Leider reicht diese Imitationsregel nicht aus, um die Bindungen aufzubauen: Suchen wir nun eine partielle Bindung für Y, stehen wir vor dem gleichen Problem wir bei F, wenn wir immer weiter imitieren, erhalten wir eine nicht abbrechende Folge von Transformationen. Dieses Problem ergibt sich, da Terme höherer Ordnung Variablen als erstes Symbol haben können, und unsere Transformationen müssen Bindungen wie z.B. $\lambda x.x$ finden können. Wenn wir für $\lambda x_1 \ldots x_k$ kurz $\lambda\overline{x_k}$ schreiben, erhält unsere neue Regel für das Finden partieller Bindungen (grob) die Form:

$$\{\langle \lambda\overline{x_k}.F(u_1, \ldots, u_n), \lambda\overline{x_k}.a(v_1, \ldots, v_m)\rangle\} \cup S \Longrightarrow$$

$$\{\langle F, t\rangle\} \cup \sigma\Big(\{\langle \lambda\overline{x_k}.F(u_1, \ldots, u_n), \lambda\overline{x_k}.a(v_1, \ldots, v_m)\rangle\} \cup S\Big),$$

wo a ein Funktionssymbol, eine Konstante oder Variable ist, und t entweder eine Imitationsbindung (d.h. $t = \lambda\overline{y_n}.a(Y_1(\overline{y_n}), \ldots, Y_m(\overline{y_n})))$ oder eine Projektionsbindung (d.h. $t = \lambda\overline{y_n}.y_i(Y_1(\overline{y_n}), \ldots, Y_q(\overline{y_n}))$[2] für ein $i, 1 \leq i \leq n$) ist und $\sigma = \{F \mapsto t\}$.
In unserem Beispiel können wir $\{\langle F(f(a)), f(F(a))\rangle\}$ durch eine Projektionsbindung in

$$\{\langle F, \lambda x.x\rangle, \langle F(f(a)), f(F(a))\rangle\}$$

transformieren. Wenn wir dann die Substitution $\{F \mapsto \lambda x.x\}$ (mit folgender β-Reduktion) anwenden, erhalten wir

$$\{\langle F, \lambda x.x\rangle, \langle f(a), f(a)\rangle\}.$$

Nach Entfernen des trivialen Paars erhalten wir das gelöste System $\{\langle F, \lambda x.x\rangle\}$.

2.3.2 Transformationen im Fall erster Ordnung

Wir werden nun Unifikation von Termen erster Ordnung definieren und eine abstrakte Sichtweise des Unifikationsprozesses als ein System nicht-deterministischer Regeln zur Umformung eines Unifikationsproblems in eine explizite Darstellung seiner Lösung (falls lösbar) präsentieren; im nächsten Abschnitt werden wir dies auf den Fall höherer Ordnung ausdehnen. Dieser Ansatz stammt aus [MM82], tauchte aber implizit schon in Herbrands Arbeit [Her30] auf.
Alle Terme in diesem Abschnitt sind Terme erster Ordnung, also gibt es keine λ-Abstraktionen, keine Variablen im Kopf von Termen, und für jeden Term t ist $FV(t)$ die Menge *aller* Variablen in t. Jeder Term erster Ordnung ist trivialerweise in $\mathcal{L}_{exp}$.

Unsere Darstellung der Unifikationsprobleme ist die folgende:

Definition 2.3.3 (Term-Paar, Unifikator, Term-System)
Ein *Term-Paar* (oder *Paar*) ist eine Multimenge von zwei Termen (Schreibweise $\langle s, t\rangle$). Eine Substitution θ heißt ein *Standard-Unifikator* (oder einfach *Unifikator*) des Paares $\langle s, t\rangle$, falls $\theta(s) = \theta(t)$. Ein *Term-System* (oder *System*) ist eine Multimenge solcher Paare, und eine Substitution θ ist ein Unifikator eines Systems, wenn sie jedes Paar unifiziert. Die Menge der Unifikatoren eines Systems S wird mit $U(S)$ bezeichnet, und wenn S nur aus dem Paar $\langle s, t\rangle$ besteht, bezeichnen wir die Menge seiner Unifikatoren mit $U(s, t)$. □

[2] Die Anzahl q der Variablen $\overline{Y_q}$ ergibt sich aus dem Typ von y_i: Wenn $\tau(y_i) = \alpha_1, \ldots, \alpha_q \to \beta$ ist, führen wir hier q neue Variablen $Y_j : \alpha_j$ ein.

Definition 2.3.4 (mgu (allgemeinster Unifikator))
Eine Substitution σ ist ein *mgu* (*most general unifier*) oder *allgemeinster Unifikator* eines Systems S, wenn

1. $Dom(\sigma) \subseteq FV(S)$;
2. $\sigma \in U(S)$;
3. Für jedes $\theta \in U(S)$ gilt $\sigma \leq \theta$. □

Für unifizierbare Systeme gibt es immer mgu's, und diese sind eindeutig bis auf Umbenennungen. Wir werden daher von *dem* mgu eines Systems sprechen, in Zeichen: $mgu(S)$.

Definition 2.3.5 (gelöst, ungelöst) Ein Paar $\langle x, t\rangle$ ist in *gelöster Form* in einem System S, und x heißt in diesem System eine *gelöste* Variable, wenn x eine Variable ist, die nirgendwo anders in S auftaucht; insbesondere $x \notin FV(t)$. Ein System ist in gelöster Form, wenn alle seine Paare in gelöster Form sind. Eine Variable ist *ungelöst*, wenn sie in S auftritt aber nicht gelöst ist. □

Man beachte, daß ein System in gelöster Form immer eine *Menge* (keine doppelten Einträge) von Paaren ist. Die Bedeutung von Systemen in gelöster Form zeigt:

Lemma 2.3.6 ([SG89])
Sei $S = \{\ \langle x_1, t_1\rangle, \ldots, \langle x_n, t_n\rangle\ \}$ ein System in gelöster Form. Die Substitution $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ ist ein idempotenter mgu von S. Außerdem gilt für jeden Unifikator $\theta \in U(S)$, daß $\theta = \sigma \circ \theta$. □

Streng genommen ist die Substitution σ hier mehrdeutig für den Fall, daß mindestens ein Paar in S aus zwei gelösten Variablen besteht; aber da mgu's als eindeutig bis auf Umbenennungen betrachtet werden und solche Paare beliebig umbenannt werden können, bezeichnen wir diese Substitution mit σ_S. Als Spezialfall gilt $\sigma_\varnothing = Id$.

Wir zerlegen die Suche nach mgu's folgendermaßen:
Wenn $\theta(u) = \theta(v)$, dann ist entweder (i) $u = v$ und keine Unifizierung nötig, oder (ii) $u = f(u_1, \ldots u_n)$ und $v = f(v_1, \ldots, v_n)$ für ein Funktionssymbol $f \in \Sigma$, und $\theta(u_i) = \theta(v_i)$ für $1 \leq i \leq n$, oder (iii) u eine Variable, die nicht in $FV(v)$ liegt oder umgekehrt. Wenn u eine Variable ist, die nicht in $FV(v)$ ist, dann ist $\{u \mapsto v\} \in U(u, v)$ und $\{u \mapsto v\} \leq \theta$. Dehnen wir dies auf Systeme von Paaren aus, haben wir ein Transformationssystem zur Suche nach mgu's:

Definition 2.3.7 (System der Transformationsregeln $\mathcal{ST}$) Sei S ein beliebiges System (evtl. leer), $f \in \Sigma$ und u, v zwei Terme. Dann besteht das *Transformationssystem* $\mathcal{ST}$ aus den folgenden Transformationen:

$$\{\ \langle u,u\rangle\ \}\cup S \implies S \tag{1}$$
$$\{\ \langle f(u_1,\ldots,u_n), f(v_1,\ldots,v_n)\rangle\ \}\cup S$$
$$\implies \{\ \langle u_1,v_1\rangle,\ldots,\langle u_n,v_n\rangle\ \}\cup S \tag{2}$$
$$\{\ \langle x,v\rangle\ \}\cup S \implies \{\ \langle x,v\rangle\ \}\cup\sigma(S), \tag{3}$$

wo x eine Variable ist, die in S auftaucht, so daß $x \notin FV(v)$ und $\sigma = \{x \mapsto v\}$.
□

Wir sollten uns daran erinnern, daß Systeme Multimengen sind – also sind die hier auftretenden Vereinigungen Multimengen-Vereinigungen. Damit bedeuten die Regeln, auf der linken Seite ein einzelnes Paar zu isolieren, das transformiert werden soll. Die Regeln (1) - (3) entsprechen den Fällen (i) - (iii) aus der Vorbemerkung. Transformation (2) heißt *Termzerlegung*, und Transformation (3) heißt *Variablenelimination*. Wir schreiben $\theta \in Unify(S)$, wenn es eine Folge von Transformationen

$$S \implies \ldots \implies S'$$

gibt, so daß S' in gelöster Form ist und $\theta = \sigma_{S'}$. (Wenn keine der Transformationen angewandt werden kann und das System nicht bereits in gelöster Form ist, schlägt die hier angegebene Prozedur fehl.)

Wählt man $S = \{\ \langle u,v\rangle\ \}$, dann kann man mit diesem System einen Unifikator für zwei Terme u, v suchen.

Beispiel 2.3.8 $\langle f(x,g(a,y)),\ f(x,g(y,x))\rangle$
$\implies_2 \langle x,x\rangle, \langle g(a,y), g(y,x)\rangle$
$\implies_1 \langle g(a,y), g(y,x)\rangle$
$\implies_2 \langle a,y\rangle, \langle y,x\rangle$
$\implies_3 \langle a,y\rangle, \langle a,x\rangle$.

Diese Transformationen erhalten die logisch invarianten Eigenschaften eines Unifikationsproblems in folgendem Sinn:

Lemma 2.3.9 ([SG89])
Falls S durch eine Transformation aus $\mathcal{ST}$ in S' übergeht, dann gilt $U(S) = U(S')$. □

Hierbei ist der entscheidende Punkt, daß die wichtigste Eigenschaft eines Unifikationsproblems, seine Lösungsmenge, unter diesen Transformationen erhalten bleibt. Also ist unsere Methode, ein solches Problem in eine triviale (gelöste) Form zu transformieren, in der die Existenz eines mgu offensichtlich ist, korrekt.

Satz 2.3.10 (Korrektheit, [SG89])
Falls $S \implies^* S'$ mit S' in gelöster Form, dann gilt $\sigma_{S'} \in U(S)$. (D.h.: Wenn das Verfahren mit einem gelösten System abbricht, dann ist der ablesbare Unifikator auch Unifikator des Ausgangssystems S.) □

Satz 2.3.11 (Vollständigkeit, [SG89])
Sei $\theta \in U(S)$. Dann muß jede Folge von Transformationen

$$S = S_0 \Longrightarrow S_1 \Longrightarrow S_2 \Longrightarrow \ldots$$

nach endlich vielen Schritten mit einer gelösten Form S' abbrechen, so daß $\sigma_{S'} \leq \theta$. (D.h.: Das Verfahren terminiert immer und liefert einen Unifikator, der allgemeiner als jeder beliebige Unifikator, also ein mgu ist.) □

Setzt man diese zwei Sätze zusammen, sieht man, daß $\mathcal{ST}$ immer einen mgu für ein unifizierbares System von Termen findet.

Manchmal ist es nützlich, mit idempotenten Unifikatoren zu arbeiten, die umbenennend weg von einer Menge „geschützter“ Variablen sind, aber allgemeinst über der Menge der Variablen des Ausgangssystems sind. Die folgende Definition präzisiert dies:

Definition 2.3.12 (mgu weg von W) Es sei S ein System und W eine Menge „*geschützt*er“ Variablen. Dann heißt eine Substitution σ ein *mgu von S weg von W* (kurz: $mgu(S)[W]$), falls

1. $Dom(\sigma) \subseteq FV(S)$ und $I(\sigma) \cap (W \cup Dom(\sigma)) = \varnothing$;
2. $\sigma \in U(S)$;
3. Für jedes $\theta \in U(S)$ gilt $\sigma \leq \theta\ [FV(S)]$.

(D.h.: ein $mgu(S)[W]$ ist ein idempotenter $mgu(S)$, der keine der Variablen in W einführt.) □

Lemma 2.3.13 ([Sny88])
Sei S ein unifizierbares System und W eine geschützte Menge von Variablen. Dann existiert eine Substitution σ, die ein $mgu(S)[W]$ ist. □

2.3.3 Transformationen im Fall höherer Ordnung

Unifikationsprobleme höherer Ordnung sind komplexer, da es Variablen funktionalen Typs gibt und wir mit Scope und gebundenen Variablen arbeiten müssen. Aus dieser zusätzlichen syntaktischen Komplexität ergeben sich einige wichtige Konsequenzen: Zunächst ist die Unifikation von Termen höherer Ordnung unentscheidbar. Dann gibt es keine mgu's mehr, und wir müssen den komplexeren Begriff der *vollständigen Menge von Unifikatoren* einführen. Schließlich kann der Suchraum beim Unterproblem, zwei flexible Terme zu unifizieren, unendlich verzweigen, was eine vernünftige Implementierung unmöglich macht.

Definition 2.3.14 (Unifikator) Die Begriffe Paar und System von Termen übertragen sich vom Fall erster Ordnung. Eine Substitution θ ist ein *Unifikator* für zwei λ-Terme e_1 und e_2, falls $\theta(e_1) \overset{*}{\longleftrightarrow}_{\beta\eta} \theta(e_2)$. θ unifiziert ein System S, wenn θ jedes Paar aus S unifiziert. Die Menge aller Unifikatoren für S heißt $U(S)$, und wie bisher schreiben wir kurz $U(s,t)$, falls $S = \{ \langle s,t \rangle \}$. □

Diese Definition ist allgemeiner, als wir sie benötigen werden, da wir unseren Ansatz in $\mathcal{L}_\eta$ entwickeln werden, um η-Konversion ausfaktorisieren zu können (vgl. Abschnitt 2). Daher sagen wir für zwei Terme $s,t \in \mathcal{L}_\eta$, daß eine normalisierte Substitution θ in $U(s,t)$ liegt, falls $\theta(s) \overset{*}{\longleftrightarrow}_{\beta} \theta(t)$ bzw. falls $\theta(s){\downarrow_\beta} = \theta(t){\downarrow_\beta}$.
Ein Termpaar in einem System S ist gelöst, wenn es die Form $\langle x{\uparrow^\eta}, t \rangle$ hat, wobei die Variable x in S nur einmal auftaucht; S ist gelöst, wenn jedes Paar in S gelöst ist. In Abweichung von der Nutzung der η-expandierten Form werden wir Paare der Form $\langle x{\uparrow^\eta}, t \rangle$ als $\langle x, t \rangle$ darstellen, um ihre Entsprechung zu Bindungen $x \mapsto t$ in Substitutionen (im Fall erster Ordnung) hervorzuheben.

Beispiel 2.3.15 Sei $u = f(a, g(\lambda x.G(\lambda y.x(b))))$ und $v = F(\lambda x.x(z))$.
Dann liegt $\theta = \{F \mapsto \lambda x_2.f(a, g(x_2)), G \mapsto \lambda x_3.x_3(z_2), z \mapsto b\}$ in $U(u,v)$, da $\theta(u){\downarrow_\beta} = \theta(v){\downarrow_\beta}$:
$\theta(u) = \{F \mapsto \lambda x_2.f(a, g(x_2)), G \mapsto \lambda x_3.x_3(z_2), z \mapsto b\} f(a, g(\lambda x.G(\lambda y.x(b))))$
$= f(a, g(\lambda x.[(\lambda x_3.x_3(z_2))(\lambda y.x(b))]))$
$\rightarrow_\beta f(a, g(\lambda x.[(\lambda y.x(b))z_2]))$
$\rightarrow_\beta f(a, g(\lambda x.x(b)))$ und
$\theta(v) = \{F \mapsto \lambda x_2.f(a, g(x_2)), G \mapsto \lambda x_3.x_3(z_2), z \mapsto b\} F(\lambda x.x(z))$
$= (\lambda x_2.f(a, g(x_2)))(\lambda x.x(b))$
$\rightarrow_\beta f(a, g(\lambda x.x(b)))$

Definition 2.3.16 (Unifikationsproblem (n-ter Ordnung)) Es sei Σ eine Menge von Funktionskonstanten. Dann ist das *Unifikationsproblem* für die von Σ erzeugte Sprache $\mathcal{L}$, für beliebige Terme $e, e' \in \mathcal{L}$ zu entscheiden, ob die Menge $U(e,e')$ nicht-leer ist. Das *Unifikationsproblem n-ter Ordnung* ist, das Unifikationsproblem für eine beliebige Sprache n-ter Ordnung zu entscheiden.
□

Satz 2.3.17 (Unentscheidbarkeit, [Gol81])
Das Unifikationsproblem zweiter Ordnung ist unentscheidbar. □

Neben der Unentscheidbarkeit ergibt sich das Problem, daß mgu's nicht mehr existieren müssen. So haben z.B. die zwei Terme $F(a)$ und a die Unifikatoren $\{F \mapsto \lambda x.a\}$ und $\{F \mapsto \lambda x.x\}$, aber es gibt keinen Unifikator, der allgemeiner als beide ist. Dies führt uns zu einer Erweiterung des Begriffs $mgu(S)[W]$ auf die Fälle höherer Ordnung, in der wir *vollständige Unifikatormengen* betrachten.

Definition 2.3.18 (Vollständige Unifikatormenge)
Zu einem System S und einer endlichen Menge „geschützter“ Variablen W heißt eine Menge U von normalisierten Substitutionen *vollständige Unifikatormenge für S weg von W* oder kurz $CSU(S)[W]$ (CSU = *complete set of unifiers*), falls

1. Für alle $\sigma \in U$ gilt $Dom(\sigma) \subseteq FV(S)$ und $I(\sigma) \cap (W \cup Dom(\sigma)) = \emptyset$,
2. $U \subseteq U(S)$,
3. Für jedes normalisierte $\theta \in U(S)$ gibt es ein $\sigma \in U$, so daß $\sigma \leq_\beta \theta[FV(S)]$.

Die erste Bedingung heißt *Reinheitsbedingung*, die zweite *Kohärenzbedingung* und die letzte *Vollständigkeitsbedingung*. Besteht S nur aus dem Paar $\langle u, v\rangle$, schreiben wir kurz $CSU(u, v)$ $[W]$. Wenn es auf W nicht ankommt, lassen wir „$[W]$“ weg. □

Daß wir beim ausschließlichen Betrachten normalisierter Substitutionen die Allgemeinheit nicht verlieren, kann man daran sehen, daß jede Substitution zu einer normalisierten Substitution $\beta\eta$-gleich ist. Wir geben nun für diesen neuen Zusammenhang eine Version von Lemma 2.3.13, die zeigt, daß auch Bedingung 1 der obigen Definition die Allgemeinheit nicht einschränkt.

Lemma 2.3.19 ([SG89])
Für ein beliebiges System S, eine Substitution θ und eine Menge W geschützter Variablen gibt es zu jedem $\theta \in U(S)$ eine normalisierte Substitution σ, so daß

1. *$Dom(\sigma) \subseteq FV(S)$ und $I(\sigma) \cap (W \cup Dom(\sigma)) = \emptyset$,*
2. *$\sigma \in U(S)$,*
3. *$\sigma \leq_{\beta\eta} \theta[FV(S)]$ und $\theta \leq_{\beta\eta} \sigma[FV(S)]$.* □

Dies zeigt, daß für beliebige S und W die Menge aller normalisierenden Unifikatoren, die Bedingungen 1 und 2 von Definition 2.3.18 erfüllen, ein $CSU(S)[W]$ ist. Insbesondere verlieren wir nicht die Allgemeinheit, wenn wir im folgenden nur normalisierte idempotente Unifikatoren θ mit $Dom(\theta) \cap I(\theta) = \emptyset$ betrachten, was unsere Darstellung vereinfachen wird.
Zum Schluß untersuchen wir die Bedeutung von Systemen in gelöster Form in $\mathcal{L}_\eta$:

Lemma 2.3.20 ([SG89])
Wenn $S = \{\ \langle x_1, t_1\rangle, \ldots, \langle x_n, t_n\rangle\ \}$ ein System in gelöster Form ist, dann ist $\{\sigma_S\}$ ein $CSU(S)[W]$ für jedes W mit $W \cap FV(S) = \emptyset$. □

Wir kommen nun zum entscheidenden Teil dieses Abschnittes, in dem wir das Transformationssystem $\mathcal{ST}$ auf den Fall höherer Ordnung verallgemeinern.

Wir werden den Prozeß der Unifikation höherer Ordnung wie folgt untersuchen: Ohne Einschränkung der Allgemeinheit wollen wir annehmen, daß u und v zwei λ-Terme in $\mathcal{L}_{exp}$ sind und daß θ ein idempotenter normalisierter Unifikator für u und v ist. Also gibt es Folgen von Reduktionen in β-Normalform: $\theta(u) \rightarrow^*_\beta w \leftarrow^*_\beta \theta(v)$. (Man beachte, daß diese Folge trivial ist, wenn alle von dieser Substitution erzeugten Instanzen von erster Ordnung sind, da es dann keine β-Reduktion gibt.) Wir werden diese Folge *top-down* untersuchen, wobei wir die folgenden fünf Fälle unterscheiden:

(A) $u = v$: keine Unifizierung nötig (in den übrigen Fällen gelte also $u \neq v$).

(B) In beiden Termen verändert keine Substitution den Kopf:
Dann gilt $Head(u) = Head(v)$, und wegen $u \neq v$ haben wir $|u|, |v| > 0$. Es seien $u = \lambda\overline{x_k}.a(\overline{u_n})$, $w = \lambda\overline{x_k}.a(\overline{w_n})$ und $v = \lambda\overline{x_k}.a(\overline{v_n})$, wobei $n > 0$ und entweder $a \in \Sigma$, $a = x_i$ für ein $i : 1 \leq i \leq k$, oder a ist eine freie Variable mit $a \notin Dom(\theta)$. Es muß dann gelten: $\theta(\lambda\overline{x_k}.u_i) \rightarrow^*_\beta \lambda\overline{x_k}.w_i \leftarrow^*_\beta \theta(\lambda\overline{x_k}.v_i)$ für $1 \leq i \leq n$, d.h. die Unterterme von u und v werden paarweise von θ unifiziert.

(C) Die Terme haben die Form $u = \lambda\overline{x_k}.F(\overline{x_k})$, $v = \lambda\overline{x_k}.v'$ für eine Variable F und einen Term v', wobei $F \notin FV(v)$:
In diesem Fall muß gelten $\theta(\lambda\overline{x_k}.F(\overline{x_k})) \stackrel{*}{\longleftrightarrow}_\beta \theta(\lambda\overline{x_k}.v')$ mit $F \notin FV(v)$, und wenn $\theta = \{F \mapsto \lambda\overline{y_k}.t\} \cup \theta'$, dann gilt wegen $\theta(F) \leftarrow_\beta \theta(\lambda\overline{x_k}.F(\overline{x_k}))$, daß $\theta(F) \stackrel{*}{\longleftrightarrow}_\beta \theta(\lambda\overline{x_k}.v')$, wobei F nicht in v' auftritt. Somit können wir dieselbe Argumentation wie im Fall erster Ordnung anwenden: Setzen wir $\sigma = \{F \mapsto \lambda\overline{x_k}.v'\}$, dann gilt $\theta =_\beta \sigma \circ \theta$, da σ und $\sigma \circ \theta$ sich nur bei F unterscheiden, aber $\theta(F) \stackrel{*}{\longleftrightarrow}_\beta \theta(\lambda\overline{x_k}.v') = \sigma \circ \theta(F)$.
Dies zeigt, daß ein Termpaar dieser Form einen mgu besitzt. (Z.B. werden $\lambda x.F(x)$ und $\lambda x.f(x,z)$ durch $\theta = \{F \mapsto \lambda y.f(y,a), z \mapsto a\}$ unifiziert, aber $\sigma = \{F \mapsto \lambda y.f(y,z)\}$ ist ein mgu.) Dies ist eine Verallgemeinerung der Variablenelimination für den Fall höherer Ordnung, da u (bis auf η-Äquivalenz) einfach eine Variable ist, die nicht in $FV(v)$ auftaucht.

(D) Im Kopf nur eines der Terme findet eine Substitution statt:
Dieser Term sei o.B.d.A. u (d.h. $Head(w) = Head(v)$). Dann sei $u = \lambda\overline{x_k}.F(\overline{u_n})$, $v = \lambda\overline{x_k}.a(\overline{v_m})$ für ein Atom $a \neq F$, welches entweder eine Funktionskonstante, eine gebundene Variable oder eine freie Variable $\notin Dom(\theta)$ ist. Um die beiden Terme zu unifizieren, muß an einer Stelle der β-Reduktionsfolge von $\theta(u)$ nach w der Kopf von u zu a werden. Dafür gibt es zwei Möglichkeiten: Entweder Imitation des Kopfes von v durch Substitution eines Termes für F, dessen Kopf a ist, oder Substitution eines Termes für F, der auf einen Unterterm von v projiziert (d.h. es gibt in v

einen Unterterm, dessen Kopf dann gematcht werden kann). Projektion ist nur möglich, wenn F vom Typ höherer Ordnung ist. Wir betrachten im folgenden diese beiden Fälle.

- *Imitation*: Die Substitution für F matcht das Kopfsymbol von v durch Imitieren des Kopfsymbols a, wo $a \in \Sigma$ oder $a \notin Dom(\theta)$ eine freie Variable ist (wie in Beispiel 2.3.15: $u = f(a, g(\lambda x.G(\lambda y.x(b))))$, $v = F(\lambda x.x(z))$, $\theta = \{F \mapsto \lambda x_2.f(a, g(x_2)), G \mapsto \lambda x_3.x_3(z_2), z \mapsto b\}$). Also gilt $\theta(F) = \lambda\overline{z_n}.a(\overline{r_m})$ für Terme $\overline{r_m}$, und wir haben eine Reduktionsfolge der Form
$$\begin{aligned}\theta(u) &= \theta(\lambda\overline{x_k}.((\lambda\overline{z_n}.a(\overline{r_m}))\overline{u_n}))\\ &\to_\beta \theta(\lambda\overline{x_k}.a(\overline{r'_m}) \overset{*}{\longleftrightarrow}_\beta \theta(\lambda\overline{x_k}.a(\overline{v_m})),\end{aligned}$$
wo $r'_i = \{z_1 \mapsto u_1, \ldots, z_n \mapsto u_n\}r_i$ für $1 \leq i \leq m$. (Wegen der Idempotenz von θ instanziieren wir in dieser Folge zur Anschauung den Term u nur teilweise mit der Bindung für den Kopf F.)
- *Projektion*: Die Substitution für F versucht, das Kopfsymbol a von v durch Projektion auf einen Unterterm von u zu matchen. Es gibt drei Arten, dies zu tun, in Abhängigkeit vom Kopfsymbol des Terms, auf den projiziert wird.
Zunächst könnte ein Unterterm von u den Kopf a besitzen und das Matchen erlauben: z.B. werden $F(\lambda x.f(x, c))$ und $f(b, c)$ durch die Substitution $\{F \mapsto \lambda y.y(b)\}$ (hier ist also f das Kopfsymbol) unifiziert.
Die zweite Möglichkeit für eine Projektion ist, daß vielleicht ein Unterterm von u flexibel ist (d.h. sein Kopf ist eine freie Variable) – dann können wir versuchen, diesen Kopf mit v zu matchen. Z.B. werden $F(\lambda x.G(x, a))$ und b durch $\{F \mapsto \lambda y.y(b), G \mapsto \lambda x_1 x_2.x_1\}$ unifiziert.
Die dritte Möglichkeit ist, daß der Unterterm selbst eine Projektion ist, und nach einer Reihe von Reduktionsschritten entsteht ein Term, der entweder flexibel ist oder dessen Kopf a ist, so daß wir matchen können. Z.B. unifiziert $\theta = \{F \mapsto \lambda y_1.y_1(\lambda y_2.y_2(a))\}$ die beiden Terme $u = F(\lambda x_1.x_1(\lambda x_2.f(x_2)))$ und $v = f(a)$ auf folgende Art:
$\theta(u) = [\lambda y_1.y_1(\lambda y_2.y_2(a))]\lambda x_1.x_1(\lambda x_2.f(x_2))$
$\to_\beta [\lambda x_1.x_1(\lambda x_2.f(x_2))]\lambda y_2.y_2(a)$
$\to_\beta (\lambda y_2.y_2(a))\lambda x_2.f(x_2)$
$\to_\beta (\lambda x_2.f(x_2))a$
$\to_\beta f(a) = \theta(f(a))$.
Wenn wir den Kopf eines flexiblen Terms $u = \lambda\overline{x_k}.F(\overline{u_n})$ durch eine Projektion ersetzen, sind wir durch den Typ von F darauf be-

schränkt, auf einen Unterterm u_i zu projizieren, der den Typ von u erhält. Insbesondere – da wir F nur durch einen Term vom gleichen Typ ersetzen können, und da Unifikation nur für Terme des gleichen Typs definiert ist – gilt: Ist $\tau(u) = \tau(v) = \alpha_1, \ldots, \alpha_k \to \beta$, dann muß $\tau(u_i)$ ein Typ der Form $\gamma_1, \ldots, \gamma_m \to \beta$ sein, damit das Ergebnis der Projektion den Typ von u erhält. Also muß der Typ der Matrix (in $t = \lambda\overline{x_k}.e$ ist e die Matrix von t) von u_i gleich dem von u sein, und die Substitution muß für alle Variablen im λ-Binder von u_i Argumente zur Verfügung stellen. Wenn also $\theta(F) = \lambda\overline{z_n}.z_i(\overline{r_{m'}})$ für ein $i : 1 \leq i \leq n$ ist, dann muß u_i die Form $u_i = \lambda\overline{y_{m'}}.u_i'$ haben, wobei die Matrix von u_i' den selben Typ hat wie die Matrizen von u und v. In diesem Fall kann der Kopf a von u eine Funktionskonstante, eine freie Variable oder eine gebundene Variable (d.h. eines der x_i) sein, und damit haben wir eine Reduktionsfolge der Form

$$\theta(u) = \theta(\lambda\overline{x_k}.[\lambda\overline{z_n}.z_i(\overline{r_{m'}})\overline{u_n}]) \to_\beta \theta(\lambda\overline{x_k}.[(\lambda\overline{y_{m'}}.u_i')\overline{r'_{m'}}])$$

$$\to^*_\beta \theta(\lambda\overline{x_k}.a'(\overline{t_p})) \xleftrightarrow{*}_\beta \theta(\lambda\overline{x_k}.a(\overline{v_m})),$$

wo $r_i' = \{z_1 \mapsto u_1, \ldots, z_n \mapsto u_n\}r_i$ für $1 \leq i \leq m'$, $\lambda\overline{x_k}.a'(\overline{t_p}) = (\lambda\overline{x_k}.[(\lambda\overline{y_{m'}}.u_i')\overline{r'_{m'}}])\downarrow_\beta$ und entweder $a' = a$ oder a eine freie Variable aus $Dom(\theta)$ ist.

(E) In den Köpfen beider Terme treten Substitutionen auf:
Dann sei $u = \lambda\overline{x_k}.F(\overline{u_n})$ und $v = \lambda\overline{x_k}.G(\overline{u_m})$ mit $F, G \in Dom(\theta)$. Hier müssen wir eventuell die beiden Köpfe F und G matchen, aber dafür haben wir viele Möglichkeiten. Um unsere Analyse zu vereinfachen, versuchen wir, diesen Fall (falls möglich) auf den vorangegangenen Fall zu reduzieren. O.B.d.A. konzentrieren wir uns auf die Bindung, die für die Variable F gemacht wurde. Dann gibt es zwei Unterfälle.

(i) θ ersetzt F durch einen Nicht-Projektionsterm, z.B. $\theta(F) = \lambda\overline{z_n}.a(\overline{s_p})$, wo $a \neq G$ keine gebundene Variable ist (und wegen Idempotenz nicht in $Dom(\theta)$ ist), und verursacht dann (eventuell) eine β-Reduktion, wonach wir das Ergebnis gemäß Fall (D) untersuchen können.

(ii) θ ersetzt F durch einen Projektionsterm (der den oben diskutierten Typerhaltungsbedingungen genügt), z.B. $\theta(F) = \lambda\overline{z_n}.z_j(\overline{t_q})$. Wenn dann nach Reduktion in Normalform das Kopfsymbol entweder eine Funktionskonstante, eine gebundene Variable oder eine Variable, die nicht in $Dom(\theta)$ liegt, ist, können wir das Ergebnis mit Fall (D) untersuchen. Ist der Kopf eine Variable in $Dom(\theta)$, können wir (rekursiv) Fall (E) auf diese neuen Terme anwenden.

Durch rekursives Anwenden dieser Analyse auf die erzeugten Unterprobleme können wir jede Bindung und jede β-Reduktion der Originalsequenz bestimmen. Dies bildet die Grundlage für die folgende Menge von Transformationsregeln, welche Unifikatoren durch den „inkrementellen" Aufbau von Bindungen findet, wobei sie partielle Bindungen benutzt. In obigem Fall (D) bedeutet dies, daß es nur eine endliche Anzahl von Wahlmöglichkeiten für eine partielle Bindung gibt, weil es nur eine mögliche Imitation und nur eine endliche Anzahl möglicher Projektionen gibt. In Fall (E) ist dies leider falsch. In [Hue76] wird gezeigt, daß zwei flexible Terme nicht notwendig einen endlichen CSU besitzen müssen, und tatsächlich kann es eine unendliche Menge von unabhängigen Unifikatoren geben, welche flexible Terme als Bindungen enthalten, so daß es selbst, wenn wir nur versuchen, das oberste Funktionssymbol der Bindung zu finden, eine unendliche Anzahl von Wahlmöglichkeiten geben kann, da es für jeden Typ immer eine unendliche Menge von Funktionsvariablen gibt. Sogar wenn es nur eine endliche Menge von Funktionskonstanten in der Sprache gibt, ist es nicht möglich, den Nichtdeterminismus dieses Falles allgemein auf eine endliche Zahl von Wahlmöglichkeiten für partielle Bindungen zu reduzieren, also muß der Suchbaum unendlich verzweigen.

Eine vollständige Unifikations-Prozedur muß zu einem beliebigen System S von Termen aus $\mathcal{L}_{exp}$ und einer normalisierten Substitution $\theta \in U(S)$ stets eine Substitution σ finden, so daß $\sigma \in U(S)$ und $\sigma \leq_\beta \theta[FV(S)]$. Die grundlegende Idee der Transformationsmethode ist, zu gegebenem $\theta \in U(S)$, „Stücke" von θ zu finden, indem wir gelöste Paare $\langle x, t\rangle$ finden, für die $\theta(x) \overset{*}{\longleftrightarrow}_\beta \theta(t)$ gilt; in diesem Fall können wir mit einer Argumentation, die der aus Lemma 2.3.20 ähnelt, schließen, daß $\theta =_\beta \{x \mapsto t\} \circ \theta$, und wenn wir ausreichend viele solcher Paare finden, erreichen wir schließlich $\sigma =_\beta \{x_1 \mapsto t_1\} \circ \cdots \circ \{x_n \mapsto t_n\}$, wobei σ ein Unifikator für S ist, der allgemeiner als (oder äquivalent zu) θ ist. In anderen Worten: wir approximieren θ sukzessive, bis die Substitution genügend aufgebaut ist, so daß sie S unifiziert. Dies tun wir durch „Auflösen" von Variablen (wie im Fall (C)) oder durch Benutzen von Approximationen für individuelle Bindungen, die wir *partielle Bindungen* nennen (Methode von *Huet*):

Definition 2.3.21 (partielle Bindung, allgemeiner flexibler Term)
Eine *partielle Bindung* vom Typ $\alpha_1, \ldots, \alpha_n \to \beta$ (für einen Grundtyp β) ist ein Term der Form

$$\lambda\overline{y_n}.a\Big(\lambda\overline{z^1_{p_1}}.H_1(\overline{y_n}, \overline{z^1_{p_1}}),\ \ldots\ , \lambda\overline{z^m_{p_m}}.H_m(\overline{y_n}, \overline{z^m_{p_m}})\Big)$$

für ein Atom a, wobei

(1) $\tau(y_i) = \alpha_i$ für $1 \leq i \leq n$,
(2) $\tau(a) = \gamma_1, \ldots, \gamma_m \to \beta$, mit $\gamma_i = \varphi^i_1, \ldots, \varphi^i_{p_i} \to \gamma'_i$ für $1 \leq i \leq m$,

(3) $\tau(z^i_j) = \varphi^i_j$ für $1 \leq i \leq m$ und $1 \leq j \leq p_i$,

(4) $\tau(H_i) = \alpha_1, \ldots, \alpha_n, \varphi^i_1, \ldots, \varphi^i_{p_i} \to \gamma'_i$ für $1 \leq i \leq m$.

Dabei sind $\gamma'_1, \ldots, \gamma'_m$ Grundtypen. Die unmittelbaren Unterterme der partiellen Bindung (d.h. die Argumente des Atoms a) heißen *allgemeine flexible Terme*. □

Man beachte, daß diese partiellen Bindungen durch ihren Typ und ihr Kopfsymbol a eindeutig (bis auf Umbenennung der freien Variablen) festgelegt sind.

Definition 2.3.22 (Imitationsbindung, Projektionsbindung, Variante, passend) Eine partielle Bindung gemäß Definition 4.8 heißt *Imitationsbindung* für a, falls a eine Funktionskonstante oder eine freie Variable ist. Sie heißt eine i-te *Projektionsbindung*, falls a eine gebundene Variable y_i für ein $i : 1 \leq i \leq n$ ist. Eine *Variante* einer partiellen Bindung t ist ein Term $\rho(t)\downarrow_\beta$, wobei ρ eine Umbenennung der Menge $H_1, \ldots, H_m$ der freien Variablen in den Köpfen der allgemeinen flexiblen Terme in t ist, die weg von allen Variablen im Kontext, in dem t benutzt wird, ist. Für eine beliebige Variable F heißt eine partielle Bindung t *passend zu* F, falls $\tau(t) = \tau(F)$. Eine Imitationsbindung ist *passend zu* $\lambda\overline{x_k}.F(\overline{u_n})$, falls sie passend zu F ist. □

Um die auftretenden Ausdrücke klein zu halten, erweitern wir unsere vektorartige Schreibweise auf partielle Bindungen in der Form

$$\begin{aligned}&\lambda\overline{y_n}.a(\overline{\lambda\overline{z_{p_m}}.H_m(\overline{y_n},\overline{z_{p_m}})})\\ &\quad := \lambda\overline{y_n}.a\Big(\lambda\overline{z^1_{p_1}}.H_1(\overline{y_n},\overline{z^1_{p_1}}),\ \ldots\ ,\lambda\overline{z^m_{p_m}}.H_m(\overline{y_n},\overline{z^m_{p_m}})\Big).\end{aligned}$$

Entsprechend unserer oben gegebenen Analyse der Unifikation höherer Ordnung haben wir die folgende Menge von Transformationen:

Definition 2.3.23 (System der Transformationsregeln $\mathcal{HT}$)
Sei S ein System von λ-Termen (evtl. leer). Dann definieren wir das *Transformationssystem* $\mathcal{HT}$ als Menge der folgenden Transformationen:

- Weglassen unifizierter Paare:
$$\{\ \langle u, u\rangle\ \} \cup S \Longrightarrow\ S \tag{1}$$

- *Termzerlegung*:
$$\begin{aligned}&\{\ \langle \lambda\overline{x_k}.a(\overline{u_n}), \lambda\overline{x_k}.a(\overline{v_n})\rangle\ \} \cup S\\ &\quad\Longrightarrow \bigcup_{1\leq i\leq n}\Big\{\langle \lambda\overline{x_k}.u_i, \lambda\overline{x_k}.v_i\rangle\Big\} \cup S,\end{aligned} \tag{2}$$
wo a ein beliebiges Atom ist.

- *Variablenelimination*:
Falls $u = \lambda\overline{x_k}.F(\overline{x_k})$ und $v = \lambda\overline{x_k}.v'$ für ein k, eine Variable F und einen Term v' mit $F \notin FV(v)$, dann gilt

$$\{\ \langle u, v\rangle\ \} \cup S \Longrightarrow \{\ \langle F, \lambda\overline{x_k}.v'\rangle\ \} \cup \sigma(S)\downarrow_\beta, \tag{3}$$

wobei $\sigma = \{F \mapsto \lambda\overline{x_k}.v'\}$.

Diese drei Transformationen sind analog zum System $\mathcal{ST}$. Um auch Funktionsvariablen verarbeiten zu können, brauchen wir eine weitere Transformation, die in drei Fälle unterteilt ist:

- *Imitation*:

$$\{\ \langle\lambda\overline{x_k}.F(\overline{u_n}), \lambda\overline{x_k}.a(\overline{v_m})\rangle\ \} \cup S$$
$$\Longrightarrow \{\ \langle F, t\rangle, \langle\lambda\overline{x_k}.F(\overline{u_n}), \lambda\overline{x_k}.a(\overline{v_m})\rangle\ \} \cup S, \tag{4a}$$

  wenn a eine Funktionskonstante oder eine freie Variable $\neq F$ ist und t eine Variante einer Imitationsbindung für a ist, die passend zu F ist, z.B. $t = \lambda\overline{y_n}.a(\overline{\lambda\overline{z_{pm}}.H_m(\overline{y_n}, \overline{z_{pm}})})$.

- *Projektion*:

$$\{\ \langle\lambda\overline{x_k}.F(\overline{u_n}), \lambda\overline{x_k}.a(\overline{v_m})\rangle\ \} \cup S$$
$$\Longrightarrow \{\ \langle F, t\rangle, \langle\lambda\overline{x_k}.F(\overline{u_n}), \lambda\overline{x_k}.a(\overline{v_m})\rangle\ \} \cup S, \tag{4b}$$

  wenn a ein beliebiges Atom (eventuell gebunden) ist und t eine Variante einer i-ten Projektionsbindung (für ein $i : 1 \leq i \leq n$) ist, die passend zum Term $\lambda\overline{x_k}.F(\overline{u_n})$ ist, d.h. $t = \lambda\overline{y_n}.y_i(\overline{\lambda\overline{z_{p_q}}.H_q(\overline{y_n}, \overline{z_{p_q}})})$, so daß, falls $Head(u_i)$ eine Funktionskonstante ist, gilt: $Head(u_i) = a$.

- *partielle Bindung*:

$$\{\ \langle\lambda\overline{x_k}.F(\overline{u_n}), \lambda\overline{x_k}.G(\overline{v_m})\rangle\ \} \cup S$$
$$\Longrightarrow \{\ \langle F, t\rangle, \langle\lambda\overline{x_k}.F(\overline{u_n}), \lambda\overline{x_k}.G(\overline{v_m})\rangle\ \} \cup S, \tag{4c}$$

  wenn $t = \lambda\overline{y_n}.a(\overline{\lambda\overline{z_{pm}}.H_m(\overline{y_n}, \overline{z_{pm}})})$ eine Variante einer beliebigen partiellen Bindung ist, die passend zum Term $\lambda\overline{x_k}.F(\overline{u_n})$ ist, so daß $a \neq F$ und $a \neq G$. □

Als Teil der Transformationen (4a)-(4c) wenden wir unmittelbar Transformation (3) auf das neue Paar $\langle F, t\rangle$ an, was sich als Anwendung der Substitution $\{F \mapsto t\}$ auf den Rest des Systems auswirkt. Wie beim System $\mathcal{ST}$ sind auch hier die Vereinigungen wieder Vereinigungen von *Multimengen*, nicht von Mengen!

Im folgenden sagen wir $\theta \in Unify(S)$ genau dann, wenn es eine Folge von Transformationen $S \Longrightarrow^* S_n$ gibt, so daß S_n in gelöster Form ist und $\theta = \sigma_{S_n}|_{FV(S)}$.

Beispiel 2.3.24 Die folgende Transformationsfolge führt zu einem System in gelöster Form. (Um die Wirkung des β-Reduktionsschrittes zu zeigen, der auf die Substitution in Regel (3) folgt, benutzen wir die Schreibweise $\frac{\theta(e)}{\theta(e)\Downarrow_\beta}$. Im „Zähler" steht also das Ergebnis nach der Substitution und im „Nenner" das Endergebnis nach β-Reduktion.)
$\langle F(f(a)), f(F(a))\rangle$
$\Longrightarrow_{4a} \langle F, \lambda x.f(Y(x))\rangle, \langle \frac{\lambda x.f(Y(x)))f(a)}{f(Y(f(a)))}, f(\frac{\lambda x.f(Y(x)))a}{f(Y(a))})\rangle$
$\Longrightarrow_2 \langle F, \lambda x.f(Y(x))\rangle, \langle Y(f(a)), f(Y(a))\rangle$
$\Longrightarrow_{4b} \langle F, \lambda x.f(\frac{(\lambda x.x)x}{x})\rangle, \langle Y, \lambda x.x\rangle, \langle \frac{(\lambda x.x)f(a)}{f(a)}, f(\frac{(\lambda x.x)a}{a})\rangle$
$\Longrightarrow_1 \langle F, \lambda x.f(x)\rangle, \langle Y, \lambda x.x\rangle$
Also ist $\{F \mapsto \lambda x.f(x)\} \in Unify(F(f(a)), f(F(a)))$.

2.3.4 Korrektheit und Vollständigkeit

Satz 2.3.25 (Korrektheit, [SG89])
Es sei $S \Longrightarrow^* S'$ mit S' in gelöster Form. Dann gilt $\sigma_{S'}|_{FV(S)} \in U(S)$. □

Im folgenden definieren wir ein System von Transformationen auf Paaren (θ, S), welches zeigt, wie der Aufbau einer Substitution θ eine geeignete Folge von Transformationen festlegt.

Definition 2.3.26 (Transformationssystem $\mathcal{CT}$) Sei θ eine normalisierte Substitution und S ein beliebiges System. Die ersten drei Transformationen entsprechen denen aus $\mathcal{HT}$:

$$\theta, S \Longrightarrow_i \theta, S'$$

für $i = 1, 2, 3$, falls $S \Longrightarrow_i S'$ in $\mathcal{HT}$, mit der Einschränkung, daß Regel (2) nur auf ein Paar $\langle u, v\rangle$ angewandt werden darf, bei dem das vorderste Funktionssymbol in u und v *keine* freie Variable in $Dom(\theta)$ ist. Außerdem haben wir die Regel

$$\{F \mapsto s\} \cup \theta, \{\langle \lambda\overline{x_k}.F(\overline{u_n}), \lambda\overline{x_k}.v\rangle\} \cup S$$
$$\Longrightarrow_4 \{F \mapsto s\} \cup \eta \cup \theta, \{\langle F, t\rangle, \langle \lambda\overline{x_k}.F(\overline{u_n}), \lambda\overline{x_k}.v\rangle\} \cup S,$$

wo F auf der linken Seite noch nicht gelöst ist, s ein Term der Form $\lambda\overline{y_n}.a(\overline{s_m})$ ist,

$$t = \lambda\overline{y_n}.a(\overline{\lambda\overline{z_{p_m}}.H_m(\overline{y_n}, \overline{z_{p_m}})})$$

eine zu F passende partielle Bindung mit dem (bis auf α-Konversion) gleichen Kopf wie s ist, und

$$\eta = \{H_1 \mapsto \lambda\overline{y_n}.s_1, \ldots, H_m \mapsto \lambda\overline{y_n}.s_m\}.$$

(Falls $m = 0$, wird η weggelassen.) Wie in $\mathcal{HT}$ wird im Anschluß an Regel (4) unmittelbar Regel (3) auf das neue Paar $\langle F, t\rangle$ angewandt. □

Beispiel 2.3.27 Wir ergänzen das Paar in Beispiel 4.11 um die Substitution $\theta = \{F \mapsto \lambda x.f(x)\}$. Dann ergibt sich diese $\mathcal{CT}$-Transformationsfolge:

$$\{F \mapsto \lambda x.f(x)\}, \{\langle F(f(a)), f(F(a))\rangle\}$$
$$\Longrightarrow_4 \{F \mapsto \lambda x.f(x), Y \mapsto \lambda x.x\},$$
$$\{\langle F, \lambda x.f(Y(x))\rangle, \langle \tfrac{\lambda x.f(Y(x)))f(a)}{f(Y(f(a)))}, f(\tfrac{\lambda x.f(Y(x)))a}{f(Y(a))})\rangle\}$$
$$\Longrightarrow_2 \{F \mapsto \lambda x.f(x), Y \mapsto \lambda x.x\}, \{\langle F, \lambda x.f(Y(x))\rangle, \langle Y(f(a)), f(Y(a))\rangle\}$$
$$\Longrightarrow_4 \{F \mapsto \lambda x.f(x), Y \mapsto \lambda x.x\},$$
$$\{\langle F, \lambda x.f(\tfrac{(\lambda x.x)x}{x})\rangle, \langle Y, \lambda x.x\rangle, \langle \tfrac{(\lambda x.x)f(a)}{f(a)}, f(\tfrac{(\lambda x.x)a}{a})\rangle\}$$
$$\Longrightarrow_1 \{F \mapsto \lambda x.f(x), Y \mapsto \lambda x.x\}, \{\langle F, \lambda x.f(x)\rangle, \langle Y, \lambda x.x\rangle\}$$

Satz 2.3.28 (Vollständigkeit von $\mathcal{HT}$, [SG89])
Es sei S ein System und $\theta \in U(S)$. Dann gibt es eine Transformationsfolge

$$S = S_0 \Longrightarrow S_1 \Longrightarrow S_2 \Longrightarrow \ldots \Longrightarrow S_n,$$

wo S_n in gelöster Form ist und $\sigma_{S_n} \leq_\beta \theta[FV(S)]$. □

Man beachte, daß diese Aussage sich von der Vollständigkeitsaussage für das System $\mathcal{ST}$ dadurch unterscheidet, daß wir hier nur die Existenz *einer* terminierenden Transformationsfolge erhalten, wogegen im Fall erster Ordnung *alle* Transformationsfolgen terminierten.

Die Aussagen über Korrektheit und Vollständigkeit fassen wir im folgenden, letzten Satz dieses Abschnittes zusammen:

Satz 2.3.29 (Satz über das Transformationssystem $\mathcal{HT}$, [SG89])
Für ein beliebiges System S ist die Menge

$$\{\sigma_{S'}|_{FV(S)} : S \Longrightarrow^*_{\mathcal{HT}} S' \text{ und } S' \text{ in gelöster Form}\}$$

ein $CSU(S)$. □

2.3.5 Zusammenfassung

Ausgehend von Unifikationsproblemen erster Ordnung haben wir ein System von Transformationen entwickelt, unter denen die Lösungen des Problems invariant bleiben: wir haben Korrektheit und Vollständigkeit nachgewiesen und außerdem bemerkt, daß das Verfahren terminierend ist. Diese Transformationen waren im wesentlichen Termzerlegung und Variablenelimination. Anschließend haben wir unsere Transformationen verallgemeinert und so das Verfahren auf λ-Terme höherer Ordnung ausgedehnt: hierzu mußten wir unser Transformationssystem um Regeln zur Imitation und Projektion ergänzen, was die Komplexität stark vergrößerte: während jede der Transformationen im Fall erster Ordnung die Terme „verkleinerte“ (in einem von uns definierten Sinn) und somit schließlich keine Transformation mehr anwendbar ist, gilt dies im Fall höherer Ordnung nicht mehr: Dadurch können Suchbäume ins Unendliche wachsen, und das Verfahren ist nicht mehr terminierend. Somit erhielten wir bei der Vollständigkeit die Einschränkung, daß das Verfahren nur nicht-deterministisch vollständig ist, d.h. es muß nicht jede Transformationsfolge terminieren, aber es gibt in jedem Fall eine, die terminiert.

Außerdem haben wir beim Übergang zum Fall höherer Ordnung die Einfachheit der Existenz eines allgemeinsten Unifikators (mgu) verloren; hier gibt es im allgemeinen nur noch eine vollständige Menge von Unifikatoren, die zu jedem Unifikator einen allgemeineren Unifikator enthält.

2.4 Narrowing

Narrowing stellt die operationale Grundlage funktional-logischer Programmiersprachen dar. Es vereint die operationalen Prinzipien der Resolution mit Unifikation (Logik-Programmierung) und der Reduktion (Funktionale Programmierung). Funktional-logische Programme erlauben die Verwendung von logischen Variablen und Funktionen höherer Ordnung. Wir geben kurz die Definitionen des traditionellen Narrowing aus [Han94b].

Definition 2.4.1 (Prädikat, Literal, Gleichung, Klausel, Variante) Sei $\mathcal{P}$ eine Menge von *Prädikatsymbolen* und $= \in \mathcal{P}$. Ein *Literal* $p(t_1, \ldots, t_n)$ besteht aus einem n-stelligen Prädikatsymbol p, daß auf n Argumentterme $\overline{t_n}$ angewandt wird. Eine *Gleichung* ist ein Literal, dessen Prädikatsymbol $=$ ist. Für Gleichungen verwenden wir die Infixnotation $t_1 = t_2$.

Eine *Klausel* hat die Form $L_0 : -L_1, \ldots, L_n$. $(n \geq 0)$, wobei $L_0, \ldots, L_n$ Literale sind. Eine Klausel heißt (*bedingte*) *Gleichung*, falls L_0 eine Gleichung ist und *unbedingte Gleichung*, falls zusätzlich $n = 0$. Da Gleichungen nur von links nach rechts gelesen werden sollen, nennen wir sie *Termersetzungsregeln*.

Eine Klausel heißt *Variante* einer anderen Klausel, falls sie aus dieser durch bijektives Ersetzen von Variablen durch andere Variablen entsteht. □

Definition 2.4.2 (funktionallogisches Programm) Ein *funktionallogisches Programm* (oder *gleichungslogisches Programm*) ist eine endliche Menge P von Klauseln. □

Definition 2.4.3 (Termersetzungsschritt) Es sei P ein funktionallogisches Programm. Ein *Termersetzungsschritt* kann auf einen Term t angewendet werden, falls eine Position p in t, eine unbedingte Regel $l = r \in P$ und eine Substitution σ existieren, so daß

$$t|_p = \sigma(l) \text{ und } s = t[\sigma(r)]_p.$$

Durch diesen Termersetzungsschritt wird t zu s. Schreibweise: $t \rightarrow_P s$.

In diesem Fall heißt t *reduzibel* (an der Position p). Ein Term t heißt *irreduzibel* oder *in Normalform*, falls es keinen Term t' mit $t \rightarrow_P t'$ gibt. □

Enthält ein Funktionsaufruf logische Variablen, ist es i.a. nötig, diese Variablen mit geeigneten Termen zu instanziieren, bevor ein Termersetzungsschritt angewendet werden kann. Dies läßt sich realisieren, indem im Termersetzungsschritt das Matchen von l mit $t|_p$ durch Unifikation dieser beiden Terme ersetzt wird – die Bezeichnung für dieses Verfahren ist *Narrowing*.

Definition 2.4.4 (Narrowing-Schritt) Es sei P ein funktionallogisches Programm. Ein Term t wird durch einen *Narrowing-Schritt* in einen Term t' überführt, falls die folgenden Bedingungen erfüllt sind:

1. p ist eine Position in t, so daß $t|_p$ keine Variable ist,
2. $l = r$ ist eine neue Variante einer Regel aus P,
3. σ ist ein mgu von $t|_p$ und l,
4. $t' = \sigma(t[r]_p)$.

In diesem Fall schreiben wir: $t \rightsquigarrow_{[p,l=r,\sigma]} t'$ oder $t \rightsquigarrow_{[l=r,\sigma]} t'$ oder auch nur $t \rightsquigarrow_\sigma t'$.

Für eine Folge $t_0 \rightsquigarrow_{\sigma_1} t_1 \rightsquigarrow_{\sigma_2} t_2 \rightsquigarrow_{\sigma_3} \ldots \rightsquigarrow_{\sigma_n} t_n$ schreiben wir kurz $t \rightsquigarrow_\sigma^* t_n$, wobei $\sigma = \sigma_1 \circ \sigma_2 \circ \cdots \circ \sigma_n$.[3] □

Bemerkung 2.4.5 Aus der Verwendung von mgu's ist klar, daß dieses Verfahren nur auf Terme erster Ordnung angewendet werden kann.

Um alle Lösungen für eine Anfrage an P zu erhalten, müssen parallel alle Regeln auf alle nicht-variablen Positionen angewendet werden. Dadurch entsteht ein großer (und oft unendlicher) Suchraum, so daß Optimierungen durch den Einsatz einer *Narrowing-Strategie* sinnvoll sind. Eine dieser Strategien ist das *Needed Narrowing* [AEH94, HP96], das im folgenden Kapitel vorgestellt wird. Dieses wird dann auch auf den Fall höherer Ordnung verallgemeinert. □

[3] Aus Konsistenzgründen mit dem Abschnitt über Unifikation definieren wir den Kompositionsoperator $\circ$, abweichend von [Han94b], durch $(\sigma_1 \circ \sigma_2)(t) := \sigma_2(\sigma_1(t))$.

Kapitel 3

Definierende Bäume

Termersetzungssysteme legen fest, an welchen Stellen Ersetzungen vorgenommen werden können. Wenn ein Term mehrere Redexe besitzt, so wird durch das Termersetzungssystem nicht vorgegeben, welcher oder welche dieser Redexe ersetzt werden sollen – dies ist Aufgabe einer Ersetzungsstrategie (wie etwa „Leftmost-Innermost“), die für jeden Term den zu ersetzenden Redex (bzw. für parallele Strategien die Redexe) bestimmt.

In [Ant92] wird der Begriff des *definierenden Baumes* eingeführt: mit Hilfe dieser Struktur kann die Strategie auf syntaktischer Ebene in das Termersetzungssystem hineingeholt werden. Die mit Hilfe definierender Bäume beschreibbaren Termersetzungssysteme heißen *induktiv sequentiell*.

3.1 Grundlagen

Wir geben im folgenden nicht die ursprüngliche Definition aus [Ant92] an, sondern verwenden direkt die später benötigte, leicht veränderte Definition aus [HP96]:

Definition 3.1.1 (Muster höherer Ordnung) Ein Term t in β-Normalform heißt *Muster höherer Ordnung* (engl.: higher-order pattern), falls jedes freie Auftreten einer Variable F in einem Unterterm $F(\overline{u_n})$ von t geschieht, so daß die $\overline{u_n}$ zu einer Liste verschiedener gebundener Variablen η-äquivalent sind. □

Definition 3.1.2 (Termersetzungssystem höherer Ordnung)
Eine *Termersetzungsregel höherer Ordnung* ist ein Paar $l \to r$, für das die folgenden Bedingungen erfüllt sind:

- l ist ein Muster höherer Ordnung aber keine freie Variable,
- l und r sind lange $\beta\eta$-Normalformen vom gleichen Grundtyp, und

- $FV(l) \supseteq FV(r)$.

Ein *Termersetzungssystem höherer Ordnung* ist eine Menge $\mathcal{R}$ von Termersetzungsregeln höherer Ordnung. Ein Term ist in *$\mathcal{R}$-Normalform*, wenn keine Regel aus $\mathcal{R}$ anwendbar ist, und eine Substitution θ ist *$\mathcal{R}$-normalisiert*, wenn alle Terme im Bild von θ in $\mathcal{R}$-Normalform sind.

Ein Funktionssymbol f heißt *definiert*, wenn es eine Regel $f(\cdots) \to t$ in $\mathcal{R}$ gibt, anderenfalls heißt f *Konstruktorsymbol*. □

Definition 3.1.3 (definierende Bäume) Es sei $\mathcal{R}$ ein Termersetzungssystem höherer Ordnung. $\mathcal{T}$ heißt *definierender Baum* mit *Muster* (engl.: pattern) π, falls $\mathcal{T}$ endliche Tiefe und eine der folgenden zwei Formen hat:

- $\mathcal{T} = \mathit{rule}(l \to r)$, wobei $l \to r$ eine Variante einer Regel aus $\mathcal{R}$ ist, so daß $l = \pi$;

- $\mathcal{T} = \mathit{branch}(\pi, o, \overline{\mathcal{T}_k})$, wobei
  - o ein Auftreten (engl.: occurrence) einer Variablen in π ist,
  - $\overline{c_k}$ verschiedene Konstruktoren des gleichen Typs wie $\pi|_o$ $(k > 0)$,
  - und für jedes $i \in \{1, \ldots, k\}$ $\mathcal{T}_i$ ein definierender Baum mit Muster $\pi[c_i(\overline{X_{n_i}})]_o$ ist; dabei ist n_i die Stelligkeit von c_i, und die X_{n_i} sind frische, unterschiedliche Variablen.

$\mathit{pat}(\mathcal{T})$ ist das Muster des definierenden Baumes $\mathcal{T}$. □

Bemerkung 3.1.4 In der ursprünglichen Fassung [Ant92] gibt es zusätzlich die Möglichkeit

- $\mathcal{T} = \mathit{exempt}(\pi)$, wobei π ein Muster ist.

Die *exempt*- (Ausnahme-) Knoten werden für unvollständig definierte Funktionen verwendet. □

Definition 3.1.5 (definierender Baum einer Funktion)
Für eine n-stellige Funktion f heißt ein definierender Baum mit Muster $f(\overline{X_n})$ *ein definierender Baum von f*, falls es zu jeder Regel $l \to r$ in $\mathcal{R}$ mit $l = f(\overline{t_n})$ einen Knoten $\mathit{rule}(l' \to r')$ in $\mathcal{T}$ gibt, so daß l eine Variante von l' ist. □

Beispiel 3.1.6 (definierender Baum für *diff*) Ein definierender Baum für *diff* ist

$$
\begin{aligned}
&\mathit{branch}(\mathit{diff}(F, X), 1,\\
&\qquad \mathit{rule}(\mathit{diff}(\lambda y.y, X) \to 1),\\
&\qquad \mathit{rule}(\mathit{diff}(\lambda y.\sin(F'(y)), X) \to \cos(F'(X)) * \mathit{diff}(\lambda y.F'(y), X)\ ),\\
&\qquad \mathit{rule}(\mathit{diff}(\lambda y.\ln(F'(y)), X) \to \mathit{diff}(\lambda y.F'(y), X)\ )/F'(X)\ )\ ).
\end{aligned}
$$

Definition 3.1.7 (induktiv sequentiell) Eine definierte Funktion f in einem Termersetzungssystem $\mathcal{R}$ heißt *induktiv sequentiell*, falls es einen definierenden Baum $\mathcal{T}$ gibt, so daß die Menge der Blätter des Baumes (bis auf Varianten) identisch mit der Menge der f definierenden Regeln aus $\mathcal{R}$ ist.
Ein Termersetzungssystem $\mathcal{R}$ heißt *induktiv sequentiell*, falls jede der in $\mathcal{R}$ definierten Funktionen induktiv sequentiell ist. □

3.2 Der Fall erster Ordnung

Definition 3.2.1 (Needed Narrowing)
Die folgenden Regeln beschreiben die *Needed Narrowing*-Strategie:

Initial

$$t \Rightarrow^{\varnothing} \mathcal{E}val(t, \mathcal{T})$$
falls $t = f(\overline{t_n})$ und
$\mathcal{T}$ ein definierender Baum von f ist

Apply

$$\mathcal{E}val(t, rule(l \to r)), G \Rightarrow^{\varnothing} \sigma(r), G$$
falls $\sigma(l) = t$

Select

$$\mathcal{E}val(t, branch(\pi, o, \overline{\mathcal{T}_k})), G \Rightarrow^{\varnothing} \mathcal{E}val(t, \mathcal{T}_i), G$$
falls $t|_o = c(\overline{t_n})$ und $pat(\mathcal{T}_i)|_o = c(\overline{X_n})$

Instantiate

$$\mathcal{E}val(t, branch(\pi, o, \overline{\mathcal{T}_k})), G \Rightarrow^{\sigma} \sigma(\mathcal{E}val(t, \mathcal{T}_i), G)$$
falls $t|_o = X$ (Variable) und
$\sigma = \{X \mapsto pat(\mathcal{T}_i)|_o\}$

Eval Subterm

$$\mathcal{E}val(t, branch(\pi, o, \overline{\mathcal{T}_k})), G \Rightarrow^{\varnothing} \mathcal{E}val(t|_o, \mathcal{T}), \mathcal{E}val(t, branch(pi, o, \overline{\mathcal{T}_k})), G$$
falls $t|_o = f(\overline{t_n})$ und
$\mathcal{T}$ ein definierender Baum von f ist

Replace Subterm

$$t', \mathcal{E}val(t, branch(\pi, o, \overline{\mathcal{T}_k})), G \Rightarrow^{\varnothing} \mathcal{E}val(t[t']_o, branch(\pi, o, \overline{\mathcal{T}_k})), G$$
falls $t' \neq \mathcal{E}val(\ldots, \ldots)$ □

Bemerkung 3.2.2 Obige Definition ist nicht die ursprüngliche Darstellung aus dem Artikel [AEH94], in dem der Begriff des Needed Narrowings eingeführt wurde. □

3.3 Case-Ausdrücke

Zur Vorbereitung auf den Fall höherer Ordnung verwenden wir nun eine andere Notation für induktiv-sequentielle Termersetzungssysteme und führen *Case-Ausdrücke* ein.

Definition 3.3.1 (Case-Ausdruck) Ein *Case-Ausdruck* hat die Form

$$\mathit{case}\ X\ \mathit{of}\ c_1(\overline{X_{n_1}}) : \mathcal{X}_1, \ldots, c_k(\overline{X_{n_k}}) : \mathcal{X}_k,$$

wobei X eine Variable, $c_1, \ldots, c_k$ verschiedene Konstruktoren des selben Typs wie X und $\mathcal{X}_1, \ldots, \mathcal{X}_k$ Terme sind, die selbst auch wieder Case-Ausdrücke enthalten können. Die Variablen $\overline{\mathcal{X}_{n_i}}$ heißen *Muster-Variablen* und tauchen ausschließlich in den entsprechenden Untertermen $\mathcal{X}_i$ auf. □

Wir können nun definierende Bäume wie folgt in Case-Ausdrücke übersetzen:

Definition 3.3.2 (Übersetzung in Case-Ausdrücke) Sei $\mathcal{T}$ ein definierender Baum. Dann ist die Übersetzung $\mathcal{C}ase(t)$ wie folgt definiert:

$$\mathcal{C}ase(rule(l \to r)) := r,$$

$$\mathcal{C}ase(branch(\pi, o, \overline{\mathcal{T}_k}))$$

$$:= \mathit{case}\ \pi|_o\ \mathit{of}\ \ pat(\mathcal{T}_1)|_o : \mathcal{C}ase(\mathcal{T}_1), \ldots, pat(\mathcal{T}_k)|_o : \mathcal{C}ase(\mathcal{T}_k).$$

Für einen definierenden Baum $\mathcal{T}$ der n-stelligen Funktion f mit Muster $f(\overline{X_n})$ ist

$$f(\overline{X_n}) \to \mathcal{C}ase(\mathcal{T})$$

die neue Termersetzungsregel für f. □

Nun müssen wir noch eine Übersetzung für Anfragen angeben:

Definition 3.3.3 (Übersetzung der Anfragen in Case-Terme)
Die Funktion $\mathcal{EC}$ übersetzt $\mathcal{E}val$-Anfragen in Anfragen mit Case-Ausdrücken:

$$\mathcal{EC}(t) := t$$

$$\mathcal{EC}(\mathcal{E}val(t, \mathcal{T})) := \sigma(\mathcal{C}ase(\mathcal{T})), \text{ wobei } \sigma(pat(\mathcal{T})) = t$$

$$\mathcal{EC}(G, \mathcal{E}val(t, branch(\pi, o, \overline{\mathcal{T}_k})) := \mathit{case}\ \ \mathcal{EC}(G)\ \ \mathit{of}\ \ \overline{p_k : \mathcal{X}_k},$$
$$\text{wobei } \mathcal{EC}(\mathcal{E}val(t, branch(\pi, o, \overline{\mathcal{T}_k})) = \mathit{case}\ s\ \mathit{of}\ \ \overline{p_k : \mathcal{X}_k}.$$ □

3.4 Der Fall höherer Ordnung

Für die Verallgemeinerung definierender Bäume auf den Fall höherer Ordnung verwenden wir die Notation mit Case-Ausdrücken.

Definition 3.4.1 (Shallow-Pattern, defin. Baum höherer Ordnung)
Ein *Shallow-Pattern*[1] ist ein linearer Term der Form $\lambda\overline{x_n}.v(\overline{H_m(\overline{x_n})})$.

$\mathcal{T}$ ist ein *definierender Baum höherer Ordnung* (*HDT*: *higher-order definitional tree*), falls $\mathcal{T}$ endliche Tiefe hat und einer der folgenden Fälle vorliegt:

- $\mathcal{T} = p : \text{case } X \text{ of } \overline{\mathcal{T}_n}$,
- $\mathcal{T} = p : rhs$,

wo p Shallow-Patterns mit frischen Variablen, X eine freie Variable und $\overline{\mathcal{T}_n}$ HDTs sind bzw. rhs (right hand side) ein Term ist, der eine rechte Regelseite repräsentiert. Die Shallow-Patterns der HDTs $\overline{\mathcal{T}_n}$ müssen paarweise nicht-unifizierbar sein.

Die Termersetzungsregel $f(\overline{X_n}) \to \mathcal{X}$ wird mit dem HDT $f(\overline{X_n}) : \mathcal{X}$ identifiziert. □

Definition 3.4.2 ($\overline{x_k}$-Lifter) Sei $t \in \mathcal{L}(\mathcal{T})$ und W eine Menge von Variablen. Dann ist ein *$\overline{x_k}$-Lifter von t, weg von W* eine Substitution

$$\sigma = \{F \to (\rho F)(\overline{x_k}) \mid F \in FV(t)\},$$

wobei ρ eine Umbenennung mit $Dom(\rho) = FV(t), Rng(\rho) \cap W = \varnothing$ und $\rho F : \tau_1 \to \cdots \to \tau_k \to \tau$ ist, falls $F : \tau$ und $x_i : \tau_i$ für alle $i = 1, \ldots, k$.

Eine Regel $l \to r$ heißt *$\overline{x_k}$-gehoben*, wenn l und r durch Anwendung eines $\overline{x_k}$-Lifters entstanden sind. □

In [HP96] wurde folgendes Regel-System zum Narrowing im Fall höherer Ordnung vorgestellt:

Definition 3.4.3 (System LNT im Fall höherer Ordnung)
Das System LNT besteht aus den folgenden Regeln:

Bind

$e \to^? Z, G \quad \Rightarrow^{\varnothing} \quad \sigma(G)$
mit $\sigma = \{Z \mapsto e\}$, falls e kein case-Ausdruck

Case Select

$\lambda\overline{x_k}.\text{case } \lambda\overline{y_l}.v(\overline{t_m}) \text{ of } \overline{p_n : \mathcal{X}_n} \to^? Z, G \quad \Rightarrow^{\varnothing} \quad \lambda\overline{x_k}.\sigma(\mathcal{X}_i) \to^? Z, G$
falls $p_i = \lambda\overline{y_l}.v(\overline{X_m(\overline{x_k}, \overline{y_l})})$ und $\sigma = \{\overline{X_m \mapsto \lambda\overline{x_k}, \overline{y_l}.t_m}\}$

Imitation

$\lambda\overline{x_k}.\text{case } \lambda\overline{y_l}.X(\overline{t_m}) \text{ of } \overline{p_n : \mathcal{X}_n} \to^? Z, G \quad \Rightarrow^{\sigma}$
$\sigma(\lambda\overline{x_k}.\text{case } \lambda\overline{y_l}.X(\overline{t_m}) \text{ of } \overline{p_n : \mathcal{X}_n} \to^? Z, G)$

[1]Wir verwenden die englische Bezeichnung, da weder die Übersetzung „seichtes Muster“ noch die Halbübersetzung „Shallow-Muster“ überzeugen konnten.

falls $p_i = \lambda\overline{y_l}.c(\overline{X_o(\overline{x_k}, \overline{y_l})})$ (c Konstruktor)
und $\sigma = \{X \mapsto \lambda\overline{x_m}.c(\overline{H_o(\overline{x_m})})\}$

Function Guess

$\lambda\overline{x_k}.\text{case } \lambda\overline{y_l}.X(\overline{t_m}) \textit{ of } \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G \qquad \Rightarrow^\sigma$
$\sigma(\lambda\overline{x_k}.\text{case } \lambda\overline{y_l}.X(\overline{t_m}) \textit{ of } \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G)$
falls $\lambda\overline{x_k}, \overline{y_l}.X(\overline{t_m})$ kein Higher-Order-Pattern ist und
$\sigma = \{X \mapsto \lambda\overline{x_m}.f(\overline{H_o(\overline{x_m})})\}$ (f definierte Funktion)

Projection

$\lambda\overline{x_k}.\text{case } \lambda\overline{y_l}.X(\overline{t_m}) \textit{ of } \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G \qquad \Rightarrow^\sigma$
$\sigma(\lambda\overline{x_k}.\text{case } \lambda\overline{y_l}.X(\overline{t_m}) \textit{ of } \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G)$
mit $\sigma = \{X \mapsto \lambda\overline{x_m}.x_i(\overline{H_o(\overline{x_m})})\}$

Case Eval

$\lambda\overline{x_k}.\text{case } \lambda\overline{y_l}.f(\overline{t_m}) \textit{ of } \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G \qquad \Rightarrow^\varnothing$
$\lambda\overline{x_k}, \overline{y_l}.\sigma(\mathcal{X}) \rightarrow^? X, \quad \lambda\overline{x_k}.\text{case } \lambda\overline{y_l}.X(\overline{x_k}, \overline{y_l}) \textit{ of } \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G$
mit $\sigma = \{\overline{X_m \mapsto \lambda\overline{x_k}, \overline{y_l}.t_m}\}$,
$f(\overline{X_m(\overline{x_k}, \overline{y_l})}) \rightarrow \mathcal{X}$ ist eine $\overline{x_k}, \overline{y_l}$-gehobene Regel.

Durch diese Regeln wird eine Relation $\underset{LNT}{\Longrightarrow}^\sigma$ definiert. Eine Folge

$$t_0 \underset{LNT}{\Longrightarrow}^{\theta_1} t_1 \underset{LNT}{\Longrightarrow}^{\theta_2} t_2 \underset{LNT}{\Longrightarrow}^{\theta_3} \cdots \underset{LNT}{\Longrightarrow}^{\theta_n} t_n$$

schreiben wir auch kurz als $t_0 \overset{*}{\underset{LNT}{\Longrightarrow}}{}^\theta t_n$ mit $\theta := \theta_1 \circ \cdots \circ \theta_n$. □

Eine Anfrage der Form $t \rightarrow^? c$ (wobei c keine definierten Symbole enthält) transformieren wir in die Anfrage $f(t) \rightarrow^? \text{true}$, wobei f ein neues Funktionssymbol ist, für das wir die Regel $f(c) \rightarrow \text{true}$ zur Regelmenge $\mathcal{R}$ hinzufügen.

Da das Regelsystem LNT so aufgebaut ist, daß Ableitungen immer mit einer Anfrage der Form *case* t *of* $\text{true} : \text{true} \rightarrow^? X$ beginnen, definieren wir für jede Anfrage $t \rightarrow^? \text{true}$ eine *initiale Anfrage*:

Definition 3.4.4 (initiale Anfrage) Für einen Term t sei

$$\mathcal{I}(t) := \textit{case } t \textit{ of } \text{true} : \text{true} \rightarrow^? X_1$$

die *initiale Anfrage* zu t bzw. zur Anfrage $t \rightarrow^? \text{true}$. □

Satz 3.4.5 (Korrektheit von LNT, [HP96])
Falls $\mathcal{I}(t) \overset{*}{\underset{LNT}{\Longrightarrow}}{}^\theta \{\}$ *für einen Term* t*, dann gilt* $\theta(t) \overset{*}{\longrightarrow} \text{true}$. □

Satz 3.4.6 (Vollständigkeit von LNT, [HP96])
Falls $\theta(t) \overset{*}{\longrightarrow} \text{true}$ *und* θ *in* $\mathcal{R}$*-Normalform ist, dann gibt es eine Substitution* θ'*, so daß* $\mathcal{I}(t) \overset{*}{\underset{LNT}{\Longrightarrow}}{}^{\theta'} \{\}$ *mit* $\theta' \leq_{FV(t)} \theta$. □

Satz 3.4.7 (Optimalität von LNT, [HP96])
Sind $\mathcal{I}(t) \overset{*}{\underset{LNT}{\Longrightarrow}} {}^{\theta}\{\}$ *und* $\mathcal{I}(t) \overset{*}{\underset{LNT}{\Longrightarrow}} {}^{\theta'}\{\}$ *zwei verschiedene Ableitungen, dann sind* θ *und* θ' *nicht vergleichbar, d.h.* $\theta \not\leq_{FV(t)} \theta'$ *und* $\theta' \not\leq_{FV(t)} \theta$. □

Beispiel 3.4.8 Die Funktion *diff* ist durch folgendes Termersetzungssystem $\mathcal{R}$ definiert:

$$\begin{aligned}
&\mathit{diff}(\lambda y.y, X) \to 1\\
&\mathit{diff}(\lambda y.\sin(F(y)), X) \to \cos(F(X)) * \mathit{diff}(\lambda y.F(y), X)\\
&\mathit{diff}(\lambda y.\ln(F(y)), X) \to \mathit{diff}(\lambda y.F(y), X)/F(X).
\end{aligned}$$

Der zugehörige definierende Baum ist

$$\begin{aligned}
\mathit{diff}(F, X) \to \mathit{case}\ F\ \mathit{of}\ \ &\lambda y.y &&: 1,\\
&\lambda y.\sin(F'(y)) &&: \cos(F'(X)) * \mathit{diff}(\lambda y.F'(y), X),\\
&\lambda y.\ln(F'(y)) &&: \mathit{diff}(\lambda y.F'(y), X)/F'(X).
\end{aligned}$$

Wir wollen eine Lösung für die Anfrage $\lambda x.\mathit{diff}(\lambda y.\sin(F(x, y)), x) \to^? \lambda x.\cos(x)$ berechnen; um diese Anfrage auf die Form $\ldots \to^?$ true zu transformieren, ergänzen wir die Regel

$$f(\lambda x.\cos(x)) \to \mathrm{true};$$

der entsprechende Case-Ausdruck ist

$$f(G) \to \mathit{case}\ G\ \mathit{of}\ \lambda x.\cos(x) : \mathrm{true},$$

und wir berechnen eine Lösung für $t := f(\lambda x.\mathit{diff}(\lambda y.\sin(F(x, y)), x)) \to^? \mathrm{true}$.

$\mathcal{I}(t) = \mathit{case}\ f(\lambda x.\mathit{diff}(\lambda y.\sin(F(x, y)), x))\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$
  $\Downarrow$ **Case Eval** mit Regel für f; neues X_2
$\mathit{case}\ \lambda x.\mathit{diff}(\lambda y.\sin(F(x, y)), x)\ \mathit{of}\ \lambda x.\cos x : \mathrm{true} \to^? X_2,$
$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$
  $\Downarrow$ **Case Eval** mit Regel für *diff*; neues X_3 [2]
$\lambda x.\mathit{case}\ \lambda y.\sin F(x, y)\ \mathit{of}$
  $\lambda y.y : 1,$
  $\lambda y.\sin G(x, y) : \cos G(x, (\lambda y.y)(x)) * \mathit{diff}(\lambda y.G(x, y), (\lambda y.y)x),$
  $\lambda y.\ln G(x, y) : \mathit{diff}(\lambda y.G(x, y), (\lambda y.y)(x))/G(x, (\lambda y.y)(x)) \to^? X_3,$
$\mathit{case}\ \lambda x.X_3(x)\ \mathit{of}\ \lambda x.\cos x : \mathrm{true} \to^? X_2,$
$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$
  $\Downarrow$ **Case Select**, $\sigma = \{G \mapsto \lambda x.\lambda y.F(x, y)\}$
$\lambda x.(\cos F(x, x) * \mathit{diff}(\lambda y.F(x, y), x)) \to^? X_3,$
$\mathit{case}\ \lambda x.X_3(x)\ \mathit{of}\ \lambda x.\cos x : \mathrm{true} \to^? X_2,$

[2] Die x-gehobene Regel für *diff* ist

$$\begin{aligned}
\mathit{diff}(F(x), X(x)) \to \mathit{case}\ F(x)\ \mathit{of}\ &\lambda y.y : 1,\\
&\lambda y.\sin(F'(x)(y)) : \cos(F'(x)(X(x))) * \mathit{diff}(\lambda y.F'(x)(y), X(x)),\\
&\lambda y.\ln(F'(x)(y)) \ \ : \mathit{diff}(\lambda y.F'(x)(y), X(x))/F'(x)(X(x)).
\end{aligned}$$

$$
\begin{array}{l}
\text{case } X_2 \textit{ of } \text{ true} : \text{true} \to^? X_1 \\
\qquad \Downarrow \quad \textbf{Bind} \\
\text{case } \lambda x.\big((\cos F(x,x) * \textit{diff}\,(\lambda y.(F(x,y)), x))\big) \textit{ of} \\
\qquad \lambda x. \cos x : \text{true} \to^? X_2, \\
\text{case } X_2 \textit{ of } \text{ true} : \text{true} \to^? X_1 \\
\qquad \Downarrow \quad \textbf{Case Eval} \text{ mit Regel für } *, \text{ neues } X_3, \\
\qquad\qquad \sigma = \{X \mapsto \lambda x. \cos F(x,x), Y \mapsto \lambda x.\textit{diff}\,(\lambda y.(F(x,y)), x)\}^{3} \\
\lambda x.\text{case } \textit{diff}\,(\lambda y.F(x,y), x) \textit{ of} \\
\qquad 1 : \cos F(x,x), \\
\qquad s(Y'(x)) : \cos F(x,x) + \cos F(x,x) * Y'(x) \to^? X_3, \\
\text{case } \lambda x.X_3(x) \textit{ of } \lambda x. \cos x : \text{true} \to^? X_2, \\
\text{case } X_2 \textit{ of } \text{ true} : \text{true} \to^? X_1 \\
\qquad \Downarrow \quad \textbf{Case Eval} \text{ mit Regel für } \textit{diff}, \text{ neues } X_4 \\
\qquad\qquad \sigma = \{H \mapsto \lambda x.(\lambda y.F(x,y)), X \mapsto \lambda x.x\}^{4} \\
\lambda x.\text{case } \lambda y.F(x,y) \textit{ of } \lambda y.y : 1, \ldots \to^? X_4, \\
\lambda x.\text{case } X_4(x) \textit{ of} \\
\qquad 1 : \cos F(x,x), \\
\qquad s(Y'(x)) : \cos F(x,x) + \cos F(x,x) * Y'(x) \to^? X_3, \\
\text{case } \lambda x.X_3(x) \textit{ of } \lambda x. \cos x : \text{true} \to^? X_2, \\
\text{case } X_2 \textit{ of } \text{ true} : \text{true} \to^? X_1 \\
\qquad \Downarrow_\sigma \quad \textbf{Projection} \text{ mit } \sigma = \{F \mapsto \lambda x.\lambda y.y\} \\
\lambda x.\text{case } \lambda y.y \textit{ of } \lambda y.y : 1, \ldots \to^? X_4, \\
\lambda x.\text{case } X_4(x) \textit{ of} \\
\qquad 1 : \cos x, \\
\qquad s(Y'(x)) : \cos x + \cos x * Y'(x) \to^? X_3, \\
\text{case } \lambda x.X_3(x) \textit{ of } \lambda x. \cos x : \text{true} \to^? X_2, \\
\text{case } X_2 \textit{ of } \text{ true} : \text{true} \to^? X_1 \\
\qquad \Downarrow \quad \textbf{Case Select}, \sigma = \varnothing \\
\lambda x.1 \to^? X_4, \\
\lambda x.\text{case } X_4(x) \textit{ of} \\
\qquad 1 : \cos x, \\
\qquad s(Y'(x)) : \cos x + \cos x * Y'(x) \to^? X_3, \\
\text{case } \lambda x.X_3(x) \textit{ of } \lambda x. \cos x : \text{true} \to^? X_2, \\
\text{case } X_2 \textit{ of } \text{ true} : \text{true} \to^? X_1 \\
\qquad \Downarrow \quad \textbf{Bind}, \sigma = \langle X_4 / \lambda x.1 \rangle
\end{array}
$$

[3] Die x-gehobene Regel für $*$ ist:

$$X(\underline{1}) * Y(\underline{1}) \to \textit{case } Y(x) \textit{ of } 1 : X(x), s(Y'(x)) : X(x) + X(x) * Y'(x)$$

[4] Die x-gehobene Regel für *diff* ist

$$
\begin{array}{ll}
\textit{diff}\,(F(x), X(x)) \to \textit{case } F(x) \textit{ of } & \lambda y.y : 1, \\
& \lambda y. \sin(F'(x)(y)) : \cos(F'(x)(X(x))) * \textit{diff}\,(\lambda y.F'(x)(y), X(x)), \\
& \lambda y. \ln(F'(x)(y)) \ : \textit{diff}\,(\lambda y.F'(x)(y), X(x)) / F'(x)(X(x)).
\end{array}
$$

$\lambda x.case\ 1\ of$
$\quad 1 : \cos x,$
$\quad s(Y'(x)) : \cos x + \cos x * Y'(x) \to^? X_3,$
$case\ \lambda x.X_3(x)\ of\ \lambda x.\cos x : \mathrm{true} \to^? X_2,$
$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \to^? X_1$
$\quad \Downarrow \quad$ **Case Select**, $\sigma = \varnothing$
$\lambda x.\cos x \to^? X_3,$
$case\ \lambda x.X_3(x)\ of\ \lambda x.\cos x : \mathrm{true} \to^? X_2,$
$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \to^? X_1$
$\quad \Downarrow \quad$ **Bind**, $\sigma = \{X_3 \mapsto \lambda x.\cos x\}$
$case\ \lambda x.\cos x\ of\ \lambda x.\cos x : \mathrm{true} \to^? X_2,$
$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \to^? X_1$
$\quad \Downarrow \quad$ **Case Select**, $\sigma = \varnothing$
$\mathrm{true} \to^? X_2,$
$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \to^? X_1$
$\quad \Downarrow \quad$ **Bind**, $\sigma = \{X_2 \mapsto \mathrm{true}\}$
$case\ \ \mathrm{true}\ \ of\ \ \mathrm{true} : \mathrm{true} \to^? X_1$
$\quad \Downarrow \quad$ **Case Select**, $\sigma = \varnothing$
$\mathrm{true} \to^? X_1$
$\quad \Downarrow \quad$ **Bind**, $\sigma = \{X_1 \mapsto \mathrm{true}\}$
$\varnothing.$

Damit ist die berechnete Lösung $\{F \mapsto \lambda x, y.y\}$.

Bemerkung 3.4.9 Die im obigen Beispiel gezeigten Ableitungsschritte enthalten implizite $\alpha\beta\eta$-Konversionen. In Kapitel 5 werden wir ein von LNT abgeleitetes Verfahren vorstellen, das *explizite Substitutionen* verwendet. Dadurch erfolgen nach einigen Ableitungsschritten explizite β- und η-Reduktionen auf der syntaktischen Ebene. α-Konversion entfällt, da wir die Darstellung von λ-Termen in *de Bruijn-Notation* verwenden, in der die Namen gebundener Variablen durch den „Abstand“ zum jeweiligen λ-Binder ersetzt werden. $\square$

Kapitel 4

Explizite Substitutionen

In diesem Kapitel motivieren wir die Verwendung von Kalkülen mit expliziten Substitutionen und beschreiben zwei aktuelle Repräsentanten, zum einen den $\lambda\sigma$-Kalkül [ACCL91, DHK95] und zum anderen den $\lambda\upsilon$-Kalkül [Les94, BBLRD95]. Diese verwenden beide die *de Bruijn-Notation* [dB72] zur Darstellung von λ-Termen, um α-Konversion (also die Umbennung gebundener Variablen) zu vermeiden.[1]

Interessant ist die unterschiedliche Behandlung der η-Reduktion in den beiden Kalkülen – im $\lambda\sigma$-Kalkül wird sie als bedingte Ersetzungsregel implementiert, wobei die Bedingung eine Gleichung modulo einer Gleichheitstheorie darstellt. Die **Eta**-Regel des $\lambda\upsilon$-Kalküls hingegen ist eine unbedingte Termersetzungsregel erster Ordnung, weswegen wir diesem Kalkül im nächsten Kapitel den Vorzug geben, wenn wir das System LNT in einen Kalkül mit expliziten Substitutionen transformieren.

4.1 Der $\lambda\sigma$-Kalkül

Wir präsentieren den $\lambda\sigma$-Kalkül im Zusammenhang mit Unifikationsproblemen.

4.1.1 Verschiedene Variablenarten

Bei Unifikationsproblemen suchen wir Substitutionen, unter denen zwei gegebene Terme gleich werden. In λ-Termen können wir verschiedene Arten[2] von Variablen unterscheiden:

[1] Ein Kalkül mit expliziten Substitutionen, der *nicht* de Bruijn-Indizes verwendet sondern mit der gewohnten Darstellung von λ-Termen arbeitet, wird in [Ros96] vorgestellt und u.a. mit $\lambda\sigma$ und $\lambda\upsilon$ verglichen. Siehe auch Anhang A.3.

[2] Wir sprechen nicht von verschiedenen Variablen-„Typen“, um Verwirrung mit Typen τ zu vermeiden.

- *Gebundene Variablen* sind von der Unifikation nicht betroffen – es werden nur die freien Variablen betrachtet.
- *Konstanten*, die beim Unifikationsprozeß nicht substituiert werden dürfen und schließlich
- echte *Unifikationsvariablen*, über die das Unifikationsproblem definiert wird.

Im folgenden werden wir die dritte Gruppe von den übrigen durch Bildung zweier syntaktischer Kategorien trennen: $\mathcal{V}$ enthält die gebundenen Variablen und Konstanten, und $\mathcal{X}$ enthält die Unfikationsvariablen (Meta-Variablen außerhalb des β-Reduktionsprozeß).

Definition 4.1.1 (offene λ-Terme) Sei $\mathcal{X}$ eine Menge von Variablen (den Meta-Variablen $X, Y, \ldots$) und $\mathcal{V}$ eine Menge von Konstanten $(x, y, \ldots)$. Wir betrachten eine Algebra über $\mathcal{X}$ und einer Menge von Operatoren, die $\mathcal{V}$ und eine über $\mathcal{V}$ indizierte Menge von Abstraktoren $(\lambda x.)$ enthält:
Die Menge der *offenen* λ*-Terme*, $\Lambda(\mathcal{V}, \mathcal{X})$, wird induktiv definiert durch

$$\textbf{Terme} \qquad a ::= x \mid X \mid (a\ a) \mid \lambda v.a \qquad (x, v \in \mathcal{V},\ X \in \mathcal{X}). \qquad \Box$$

Nun haben wir zwei Arten von Substitutionen:

- solche, die auf $\mathcal{V}$ definiert sind und für die β-Reduktion benötigt werden, und
- solche, die auf $\mathcal{X}$ definiert sind und der Unfikation dienen.

Die $\mathcal{V}$-Substitutionen werden wie bisher definiert – mit der zusätzlichen Regel $\theta(X) := X$ für $X \in \mathcal{X}$.

Bei den Substitutionen von Meta-Variablen (aus $\mathcal{X}$) stellt sich die Frage, ob sich diese auf $\mathcal{X}$-*Graftings* (d.h. Substitutionen erster Ordnung) reduzieren lassen. Das erste der beiden betrachteten Probleme bei Graftings ist gelöst: Variablen, die durch einen λ-Abstraktor gebunden sind, gehören zu $\mathcal{V}$ und sind daher invariant unter $\mathcal{X}$-Substitutionen. Aber die Gefahr des „variable capture" besteht immer noch: wird eine Variable durch einen Term substituiert, der Konstanten enthält, so können eventuell einige von ihnen durch Abstraktoren „eingefangen" werden, wie z.B. in $\{X \mapsto x\}(\lambda x.X) = \lambda x.x$. Deswegen führen wir nun den Begriff der Substitution mit Umbenennung gebundener Variablen ein:

Definition 4.1.2 (Substitution mit Umbenennung gebundener Variablen) Sei $\theta : \mathcal{X} \to \Lambda(\mathcal{V}, \mathcal{X})$. Dann heißt die Fortsetzung $\overline{\theta}$ von θ, die durch

1. $\overline{\theta}(X) := \theta(X)$ für $X \in \mathcal{X}$,

2. $\overline{\theta}(v) := v$ für $v \in \mathcal{V}$,

3. $\overline{\theta}(\ (a\ b)\ ) := (\overline{\theta}(a)\ \overline{\theta}(b))$,

4. $\overline{\theta}(\lambda y.a) := \lambda z.\overline{\theta}(\{y \mapsto z\}\alpha^{\{y\}\cup Var\theta}(a))$, wobei z eine neue Variable ist.

definiert wird, die zu θ gehörende *Substitution mit Umbenennung gebundener Variablen.*

Dabei ist die Funktion α^V, die alle gebundenen Variablen aus V in solche umbenennt, welche nicht zu V gehören, definiert durch

1. $\alpha^V(x) := x$,

2. $\alpha^V(\ (a\ b)\ ) := (\alpha^V(a)\ \alpha^V(b))$,

3. $\alpha^V(\lambda x.a) := \lambda x.\alpha^V(a)$, falls $x \notin V$,

4. $\alpha^V(\lambda x.a) := \lambda y.\{x \mapsto y\}\alpha^V(a)$,
   falls $x \in V$, y eine neue Variable (die nicht in a oder V auftaucht). □

Bemerkung 4.1.3 Grafting und Reduktion kommutieren nicht. Beispiel: $a = ((\lambda x.X)y)$, $\theta = \{X \mapsto x\}$. Dann gilt $a \rightarrow_\beta X$, aber $\theta(a) = ((\lambda x.x)y) \not\rightarrow_\beta \theta(X) = x$.
Entsprechend gilt: $\lambda x.(X\ x) \rightarrow_\eta X$, aber $\theta(\lambda x.(X\ x)) = \lambda x.(x\ x) \not\rightarrow_\eta \theta(X) = x$. □

Satz 4.1.4 ([DHK95])
Substitution (nach Definition 4.1.2) und Reduktion sind kommutativ. □

4.1.2 λ-Kalkül in de Bruijn-Notation

Im λ-Kalkül verwenden wir Namen für die durch Abstraktoren gebundenen Variablen; diese sind jedoch sowohl für Berechnungen als auch für theoretische Überlegungen irrelevant. Für Substitutionen ist oft eine Umbenennung dieser Variablen erforderlich (α-Konversion). Tatsächlich sind also Substitutionen nicht auf den λ-Termen selbst sondern auf α-Äquivalenzklassen definiert.
Wir können auf die Namen der gebundenen Variablen verzichten, denn um für eine gebundene Variable festzustellen, zu welchem λ-Binder sie gehört, brauchen wir nur zu zählen, wieviele λs zwischen der Position dieser Variable und dem zugehörigen Binder stehen. Diese Zahl heißt *Bindungshöhe* der Position. In de Bruijn-Notation [dB72] lassen wir nun im Binder den Variablennamen weg und ersetzen jede (gebundene) Position einer Variablen durch die entsprechende Bindungshöhe.

Definition 4.1.5 (λ-Terme in de Bruijn-Notation) Sei $\mathcal{X}$ eine Menge von Variablen. Dann ist die Menge $\Lambda_{DB}(\mathcal{X})$ der *λ-Terme in de Bruijn-Notation* induktiv definiert durch

$$\textbf{Terme} \qquad a ::= \underline{\mathtt{n}} \mid X \mid \lambda\ a \mid (a\ a) \qquad (n \in \mathbb{N}, X \in \mathcal{X}).$$

Die $\underline{\mathtt{n}}$, $n \in \mathbb{N}$, heißen *de Bruijn-Indizes.*[3] □

Man beachte, daß nur gebundene Variablen und Konstanten durch Indizes ersetzt werden; die Metavariablen (aus $\mathcal{X}$) behalten ihre Namen.

Definition 4.1.6 (λ-Höhe) Die *λ-Höhe* einer Position u in einem Term a ist die Anzahl der λs an Präfix-Positionen von u, geschrieben $|u|$. Sei u eine Position eines de Bruijn-Index $\underline{\mathtt{p}}$ in einem gegebenen Term a. Falls $p \leq |u|$, dann heißt $\underline{\mathtt{p}}$ *gebunden* in a, ansonsten ist $\underline{\mathtt{p}}$ ein *freier* Index von a. □

Bei der Übersetzung eines gewöhnlichen λ-Terms in de Bruijn-Notation stellt sich die Frage, wie freie $\mathcal{V}$-Variablen behandelt werden sollen. Dazu führt man eine Liste der freien Variablen ein und ersetzt im Term die i-te Variable aus dieser Liste durch den Index $\underline{\mathtt{i}}$:

Definition 4.1.7 (Referential) Ein *Referential* $\mathcal{R}$ ist eine Liste $(x_0, \ldots, x_n)$ von Variablen. □

Wir werden bei der Übersetzung von λ-Termen nur die gebundenen und freien Variablen aus $\mathcal{V}$ in das Referential aufnehmen – die Unifikationsvariablen aus $\mathcal{X}$ werden nicht übersetzt.

Sind $x_0, \ldots, x_n$ die freien $\mathcal{V}$-Variablen von a, dann wird a als Unterterm von $\lambda x_0 \ldots \lambda x_n.a$ übersetzt.

Definition 4.1.8 (de Bruijn-Übersetzung) Sei $\mathcal{R}$ ein Referential und $a \in \Lambda(\mathcal{V}, \mathcal{X})$, so daß alle freien $\mathcal{V}$-Variablen von a in $\mathcal{R}$ vorkommen. Dann ist die *de Bruijn-Übersetzung* von a bezüglich $\mathcal{R}$, $tr(a, \mathcal{R})$, definiert durch

1. $tr(x, \mathcal{R}) = \underline{\mathtt{j}}$, wobei j die erste Position von x in $\mathcal{R}$ ist,
2. $tr(X, \mathcal{R}) = X \qquad$ für $X \in \mathcal{X}$,
3. $tr((a\ b), \mathcal{R}) = (tr(a, \mathcal{R})\ tr(b, \mathcal{R}))$,
4. $tr(\lambda\ x.a, \mathcal{R}) = \lambda(tr(a, x.\mathcal{R}))$. □

Während der Übersetzung wird also für jedes λ das Referential vergrößert: die durch das λ gebundene Variable wird aufgenommen. Geschlossene λ-Terme können bzgl. des leeren Referentials [ ] übersetzt werden.

[3] De Bruijn-Indizes werden in der Literatur auch $\mathbf{n}$, $\underline{n}$ oder $V(n)$ geschrieben. Wir verwenden die Schreibweise $\underline{\mathtt{n}}$, um Verwechslungen mit natürlichen Zahlen $n \in \mathbb{N}$ zu vermeiden.
Zur Erinnerung: $\mathbb{N} = \{1, 2, 3, 4, \ldots\}$, also $0 \notin \mathbb{N}$.

Man beachte, daß bei dieser Übersetzung verschiedene Indizes derselben (gebundenen) Variable zugeordnet werden können, wie z.B. in

$$\lambda\underline{x}\lambda y.(x\ (\lambda z.(z\ x))\ y)\ \mapsto_{tr}\ \lambda(\lambda(\underline{2}\ \lambda(1\ \underline{3})1)).$$

Sei u Position eines de Bruijn-Index $\underline{\mathtt{p}}$ in einem Term $tr(a,\mathcal{R})$.

- Falls $p \leq |u|$, dann steht $\underline{\mathtt{p}}$ für eine in a gebundene $\mathcal{V}$-Variable,
- falls $p > |u|$, dann steht $\underline{\mathtt{p}}$ für die freie Variable, die im Referential an Position $p - |u|$ steht.

Wir wollen nun β-Reduktion in der Form $((\lambda\ a)\ b) \to \{\underline{\mathtt{1}} \mapsto b\}a$ einführen – dazu müssen wir zunächst die Substitution $\{\underline{\mathtt{1}} \mapsto b\}$ definieren.

Der im folgenden definierte Lift-Operator $^+$ erhöht alle freien Indizes eines Terms um 1:

Definition 4.1.9 (Lift-Operator $^+$) Für $a \in \Lambda_{DB}(\mathcal{X})$ definieren wir a^+, den *Lift* von a, durch $a^+ := \mathrm{lft}(a,0)$, wobei $\mathrm{lft}(a,i)$ induktiv definiert ist durch

1. $\mathrm{lft}((a_1\ a_2),i) = (\mathrm{lft}(a_1,i)\ \mathrm{lft}(a_2,i))$,
2. $\mathrm{lft}(\lambda a,i) = \lambda(\mathrm{lft}(a,i+1))$,
3. $\mathrm{lft}(X,i) = X$,
4. $\mathrm{lft}(\underline{\mathtt{m}},i) = \underline{\mathtt{m+1}}$, falls $m > i$, sonst $\underline{\mathtt{m}}$. □

Es gilt dann folgendes

Lemma 4.1.10 ([DHK95])
Für $a \in \Lambda(\mathcal{V},\mathcal{X})$, ein passendes Referential $\mathcal{R}$ und eine Variable $v \in \mathcal{V} \setminus \mathcal{R}$ gilt $tr(a,\mathcal{R})^+ = tr(a,v.\mathcal{R})$. □

Definition 4.1.11 (Analogon zur $\mathcal{V}$-Substitution) $\{\underline{\mathtt{n}} \mapsto b\}a$, die Substitution durch b bei λ-Höhe $(n-1)$ in a, ist definiert durch:

$$\begin{array}{lll}
\{\underline{\mathtt{n}} \mapsto b\}(a_1\ a_2) & := (\{\underline{\mathtt{n}} \mapsto b\}a_1\ \{\underline{\mathtt{n}} \mapsto b\}a_2) & \\
\{\underline{\mathtt{n}} \mapsto b\}X & := X & \\
\{\underline{\mathtt{n}} \mapsto b\}\lambda a & := \lambda(\{\underline{\mathtt{n+1}} \mapsto b^+\}a) & \\
\{\underline{\mathtt{n}} \mapsto b\}\underline{\mathtt{m}} & := \underline{\mathtt{m-1}},\ \text{falls } m > n & (\underline{\mathtt{m}} \in FV(a)) \\
 & \quad\ b,\quad \text{falls } m = n & (\underline{\mathtt{m}} \text{ durch } \lambda \text{ des } \beta\text{-Redex gebunden}) \\
 & \quad\ \underline{\mathtt{m}},\quad \text{falls } m < n & (\underline{\mathtt{m}} \in BV(a))
\end{array}$$
□

Definition 4.1.12 (β-Reduktion) Auf $\Lambda_{DB}(\mathcal{X})$ ist die β-*Reduktionsregel* definiert durch

$$((\lambda\, a)\; b) \rightarrow^{\beta} \{\underline{1} \mapsto b\}a. \qquad \square$$

Beispiel 4.1.13

$$\begin{array}{ccc}
\lambda x.((\lambda y.(x\; y))\; x) & \xrightarrow{tr} & \lambda((\lambda(\underline{2}\,\underline{1}))\;\underline{1}) \\
\beta \big\downarrow \text{in } \Lambda(\mathcal{V},\mathcal{X}) & & \beta \big\downarrow \text{in } \Lambda_{DB}(\mathcal{X}) \\
\lambda x.(x\; x) & \xrightarrow[tr]{} & \lambda(\{\underline{1} \mapsto \underline{1}\}(\underline{2}\;\underline{1})) = \lambda(\underline{1}\,\underline{1})
\end{array}$$

Nun definieren wir das Λ_{DB}-Analogon zur η-Reduktionsregel $\lambda x.(a\; x) \rightarrow a$ (für $x \notin FV(a)$): Wir betrachten ein Referential $\mathcal{R}$, das die Konstanten von a enthält. Wenn a durch $tr(a,\mathcal{R})$ kodiert wird, dann wird $\lambda x.(a\; x)$ durch $tr(\lambda x.(a\; x),\mathcal{R}) = \lambda(tr(a,x.\mathcal{R})\;\underline{1}) = \lambda(tr(a,\mathcal{R})^{+}\;\underline{1})$ kodiert. Daraus erhalten wir:

Definition 4.1.14 (η-Reduktion) Auf $\Lambda_{DB}(\mathcal{X})$ ist die η-*Reduktionsregel* definiert durch

$$\lambda(a\;\underline{1}) \rightarrow^{\eta} b \quad \text{falls } \exists b \in \Lambda_{DB}(\mathcal{X}):\; a = b^{+}. \qquad \square$$

Satz 4.1.15 ([DHK95])
Für $a \in \Lambda_{DB}(\mathcal{X})$ gilt:

$\exists b:\; a = b^{+} \iff$

Für jedes Vorkommen u eines Index $\underline{\mathtt{p}}$ in a gilt: $p \neq |u| + 1$. $\square$

Definition 4.1.16 ($\mathcal{X}$-Substitution) Sei $\theta : \mathcal{X} \rightarrow \Lambda_{DB}(\mathcal{X})$, dann wird die zugehörige Substitution $\overline{\theta}$ definiert durch:

1. $\overline{\theta}(X) := \theta(X)$,
2. $\overline{\theta}(\underline{\mathtt{n}}) := \underline{\mathtt{n}}$,
3. $\overline{\theta}(a_1\, a_2) := (\overline{\theta}(a_1)\,\overline{\theta}(a_2))$,
4. $\overline{\theta}(\lambda a) := \lambda(\overline{\theta}^{+}(a))$,

wobei $\theta^{+} = \{X_1 \mapsto a_1^{+}, \ldots, X_n \mapsto a_n^{+}\}$ für $\theta = \{X_1 \mapsto a_1, \ldots, X_n \mapsto a_n\}$. $\square$

Satz 4.1.17 ([DHK95])
Für $a \in \Lambda(\mathcal{V},\mathcal{X})$ und eine Substitution $\theta = \{X_1 \mapsto a_1, \ldots, X_n \mapsto a_n\}$ im λ-Kalkül sowie $\theta' := \{X_1 \mapsto tr(a_1,\mathcal{R}), \ldots, X_n \mapsto tr(a_n,\mathcal{R})\}$ gilt $tr(\theta(a),\mathcal{R}) = \theta'(tr(a,\mathcal{R}))$. $\square$

4.1.3 Definition des $\lambda\sigma$-Kalkül

$\lambda\sigma$-Kalküle sind Termersetzungssysteme erster Ordnung, die eine explizite Behandlung von durch β-Reduktionen ausgelösten Substitutionen ermöglichen. Sie enthalten den λ-Kalkül (in de Bruijn-Notation) als echtes Teilsystem, wobei die β- und η-Reduktionen als spezielle Anwendungen der $\lambda\sigma$-Regeln erhalten werden.

Internalisierung von Operationen als Operatoren

Wir betrachten eine Algebra $\mathcal{A}$ mit Wertebereich A und eine berechenbare Operation $F : A \to A$, z.B. $F = \mathit{double}$ auf der Menge der natürlichen Zahlen, ausgedrückt durch die Konstruktoren 0 und S. Also $F(0) = 0, F(S(0)) = S(S(0))$, etc. Wir erweitern die Menge der Ausdrücke durch Hinzufügen eines Operators f, der die Operation F internalisieren soll; dazu nehmen wir dann folgende Ersetzungsregeln für den Operator f hinzu:

$$f(0) \to 0, \quad f(S(X))) \to S(S(f(X))).$$

Der λ-Kalkül ist eine solche Internalisierung der Applikationsoperation auf funktionalen Ausdrücken. Wir betrachten als Beispiel die beiden λ-Terme $e_1 = \lambda x.x$ und $e_2 = 0$. Die Applikationsoperation ist so definiert, daß e_1, angewandt auf e_2, als Ergebnis 0 liefert. Anstatt diese Operation direkt zu definieren ($\mathit{apply}(e_1, e_2) := \ldots$), erweitert man die Menge der λ-Ausdrücke, so daß $(e_1 e_2)$ auch ein Ausdruck ist, und man fügt Ersetzungsregeln hinzu (β-Reduktion), so daß $(\lambda x.x\ 0) \to 0$.
Der λ-Kalkül internalisiert aber nicht alle Operationen: zwar wird die Applikation internalisiert, aber Substitution und Variablenumbenennung (Lifting) gibt es nur als externe Operationen. Der $\lambda\sigma$-Kalkül geht hier einen Schritt weiter und internalisiert auch die anderen beiden Operationen.

Definition 4.1.18 (simultane Substitution) Mit id bezeichnen wir die leere Liste, interpretiert als identische Substitution. Die *simultane Substitution* $a_1.a_2 \ldots a_n.id$ ersetzt $\underline{1}$ durch a_1, $\underline{2}$ durch a_2, ..., $\underline{\mathtt{n}}$ durch a_n und verringert alle anderen (freien) Indizes im Term um n. □

Dann kann der Operator $\uparrow$, der die Lifting-Operation $^+$ internalisiert, als unendliche simultane Substitution $\mathtt{2.3.4.5.6.}\ldots$ interpretiert werden. Wir fügen noch einen Kompositionsoperator $\circ$ ein und schreiben den Index $\underline{\mathtt{n+1}}$ als $\underline{1}[\uparrow \circ \cdots \circ \uparrow] = \underline{1}[\uparrow^n]$. Außerdem setzen wir $\uparrow^0 = id$.

Definition 4.1.19 (Terme des $\lambda\sigma$-Kalküls)
Sei $\mathcal{X}$ eine Menge von Ausdrucks-Metavariablen und $\mathcal{Y}$ eine Menge von Substitutions-Metavariablen. Die Menge $\mathcal{T}_{\lambda\sigma}(\mathcal{X}, \mathcal{Y})$ von *Termen* und *expliziten Substitutionen* ist induktiv definiert durch

Terme	$a ::= \underline{1} \mid X \mid (a\ a) \mid \lambda a \mid a[s]$
Substitutionen	$s ::= Y \mid id \mid \uparrow \mid a.s \mid s \circ s$

wobei $X \in \mathcal{X}$ und $Y \in \mathcal{Y}$. □

Definition 4.1.20 (Regeln des $\boldsymbol{\lambda\sigma}$-Kalküls) Der *$\lambda\sigma$-Kalkül* wird als Termersetzungssystem $\boldsymbol{\lambda\sigma}$ mit folgenden Regeln definiert:

Beta	$(\lambda a)b$	$\to a[b.id]$	
App	$(a\ b)[s]$	$\to (a[s]\ b[s])$	
VarCons	$\underline{1}[a.s]$	$\to a$	
Id	$a[id]$	$\to a$	
Abs	$(\lambda a)[s]$	$\to \lambda(a[\underline{1}.(s \circ \uparrow)])$	
Clos	$(a[s])[t]$	$\to a[s \circ t]$	
IdL	$id \circ s$	$\to s$	
ShiftCons	$\uparrow \circ (a.s)$	$\to s$	
AssEnv	$(s_1 \circ s_2) \circ s_3$	$\to s_1 \circ (s_2 \circ s_3)$	
MapEnv	$(a.s) \circ t$	$\to a[t].(s \circ t)$	
IdR	$s \circ id$	$\to s$	
VarShift	$\underline{1}.\uparrow$	$\to id$	
SCons	$\underline{1}[s].(\uparrow \circ s)$	$\to s$	
Eta	$\lambda(a\ \underline{1})$	$\to b$	falls $a =_\sigma b[\uparrow]$

Die Ersetzungsregeln $\boldsymbol{\lambda\sigma}$ definieren eine Gleichheitstheorie, deren Kongruenz als $=_{\lambda\sigma}$ geschrieben wird. Läßt man die Regeln **Beta** und **Eta** weg, erhält man das Termersetzungssystem $\boldsymbol{\sigma}$, welches die Anwendung von Substitutionen berechnet. Die Kongruenzrelation der zugehörigen Gleichheitstheorie schreiben wir als $=_\sigma$. □

Bemerkung 4.1.21 Die in [DHK95] eingeführte η-Reduktionsregel ist eine bedingte Termersetzungsregel – um die Bedingung zu überprüfen, muß eine nicht-triviale Gleichung $\exists b:\ a =_\sigma b[\uparrow]$ gelöst werden.

Die natürlichere Wahl einer Regel der Form $\lambda(a[\uparrow]\underline{1}) \to a$ ist nicht möglich, da diese eine unendliche Menge kritischer Paare hervorrufen würde.

Dadurch wird das Ziel, implizite Vorgänge explizit zu machen, nur für die β-Reduktion erreicht. Der im nächsten Abschnitt eingeführte $\lambda\upsilon$-Kalkül verzichtet zunächst ganz auf η-Reduktion, wird aber in Abschnitt 4.2.3 um eine explizite Definition der η-Reduktion ergänzt. Briaud weist in seinem Artikel [Bri95] darauf hin, daß diese explizite η-Regel auch in andere Kalküle mit expliziten

Substitutionen übernommen werden kann: für den $\lambda\sigma$-Kalkül wäre dies eine Regel der Form

Eta $\lambda(a\ \underline{1}) \to a[\perp.id]$

für ein neu einzuführendes Symbol $\perp$ (näheres über die Verwendung von $\perp$ in Abschnitt 4.2.3). □

Satz 4.1.22 ([ACCL91, Río93, CHL92])

1. *Das Termersetzungssystem* $\boldsymbol{\lambda\sigma}$ *ist lokal konfluent auf jedem (offenen oder geschlossenen)* $\lambda\sigma$*-Term.*

2. $\boldsymbol{\lambda\sigma}$ *ist konfluent auf substitutions-abgeschlossenen Termen (d.h. Termen ohne Substitutions-Variablen).*

3. $\boldsymbol{\lambda\sigma}$ *ist* nicht *konfluent auf offenen Termen (d.h. Termen mit Ausdrucks- und Substitutionsvariablen).* □

Satz 4.1.23 (Normalformen des $\boldsymbol{\lambda\sigma}$-Kalküls, [Río93])
Jeder $\lambda\sigma$*-Term in Normalform bzgl.* $\boldsymbol{\lambda\sigma}$ *hat eine der folgenden Formen:*

1. λa,

2. $(a\, b_1 \ldots b_n)$, *wobei* a *entweder* $\underline{1}$, $\underline{1}[\uparrow^n]$, X *oder* $X[s]$ *für einen Substitutions-Term* $s \neq id$ *in Normalform ist,*

3. $a_1 \ldots a_p \cdot \uparrow^n$, *wobei* $a_1, \ldots, a_p$ *Terme in Normalform sind und* $a_p \neq \underline{\mathtt{n}}$. □

Bemerkung 4.1.24 Der $\lambda\sigma$-Kalkül enthält keine Regel der Form $X[s] \to X$. Dadurch ist $X[s]$ zunächst eine Normalform – die Auswertung dieser Substitution wird also verzögert, bis X ein Wert zugewiesen wird. Dies ist nicht die Art, wie wir die zu lösenden Variablen X im LNT-Verfahren behandeln wollen: Wir werden später (im Kapitel 5) eine solche Regel $X[s] \to X$ einführen, da wir nach jeder durch β-Reduktion angestoßenen Substitution einen *reinen* Term (d.h. einen Term ohne Substitutionsanteil $[s]$) erhalten wollen. Wir werden dies aber gefahrlos machen können, da die Variablen X immer durch Terme ersetzt werden, die bis auf freie Vorkommen von Variablen Y der selben Art wie X geschlossen sind.

Satz 4.1.25 ([DHK95])
$\mathcal{X}$-Grafting (Substitutionen erster Ordnung) und $\lambda\sigma$*-Reduktion sind kommutativ.* □

4.2 Der $\lambda\upsilon$-Kalkül

4.2.1 Die Regeln des $\lambda\upsilon$-Kalküls

Ausgehend vom $\lambda\sigma$-Kalkül hat Lescanne in [Les94] den $\lambda\upsilon$-Kalkül vorgestellt. Dies ist ein weiterer λ-Kalkül mit expliziten Substitutionen, in dem – anders als bei $\lambda\sigma$ – keine Komposition von Substitutionen möglich ist. Wir stellen den $\lambda\upsilon$-Kalkül in diesem Abschnitt kurz vor und präsentieren in Abschnitt 4.2.3 noch eine Erweiterung um eine explizite η-Regel ([Bri95]).

Wir werden später den $\lambda\upsilon$-Kalkül verwenden, um in dem Narrowing-Verfahren LNT aus [HP96] implizite Substitutionen, die durch β- oder η-Reduktionen ausgelöst werden, durch explizite Substitutionen im Stile des $\lambda\upsilon$-Kalküls zu ersetzen.

Lescannes Ansatz war es, einen Kalkül mit expliziten Substitutionen anzugeben, der mit einer minimalen Anzahl von Operatoren auskommt. Diese Überlegung folgt aus der Beobachtung, daß der $\lambda\sigma$-Kalkül überflüssige Operatoren ($\circ$ und .) enthält. Da der neue Kalkül Normalformen ohne Substitutionsanteil erzeugen soll, muß für jede auftretende explizite Substitution eine Reduktionsregel angegeben werden.

Im folgenden beschreiben wir die Entwicklung von $\lambda\sigma$ hin zu $\lambda\upsilon$, wie sie in [Les94] dargestellt wird.

Da zwar der Kalkül $\S = \boldsymbol{\lambda\sigma} \setminus \{\mathbf{Beta}\}$ konfluent und stark normalisierend ist, $\boldsymbol{\lambda\sigma}$ jedoch nicht auf offenen λ-Termen sondern nur auf reinen, geschlossenen Termen konfluent ist, haben Hardin und Lévy einen konfluenten Kalkül $\lambda\sigma_{\Uparrow}$ [HL89, CHL92] vorgestellt, der einen neuen *Lift-Operator* $\Uparrow$ einführt. Dabei ist $\Uparrow(s)$ die Substitution, die $\underline{\mathtt{1}}$ auf $\underline{\mathtt{1}}$ und $\underline{\mathtt{n+1}}$ auf $s(\underline{\mathtt{n}})[\uparrow]$ „abbildet". Damit läßt sich die Regel

$$\textbf{Abs} \quad (\lambda a)[s] \to \lambda(a[\underline{\mathtt{1}}.(s \circ \uparrow)])$$

des $\lambda\sigma$-Kalküls als

$$\textbf{Lambda} \quad (\lambda a)[s] \to \lambda(a[\Uparrow(s)])$$

schreiben. $\Uparrow(s)$ kann also als Abkürzung für $\underline{\mathtt{1}}.(s \circ \uparrow)$ aufgefaßt werden.

Im neuen Kalkül wird auf die Komposition $\circ$ von Substitutionen verzichtet; in Ausdrücken der Form $t[s_1][s_2]\ldots[s_n]$ wird immer nur $t[s_1]$ reduziert. Damit ist die **Clos**-Regel $(a[s])[t] \to a[s \circ t]$ überflüssig, und der Operator $\circ$ fällt weg. Es müssen nun Regeln der Form $\underline{\mathtt{n}}[\Uparrow(s)] \to ?$ angegeben werden. Auch der Cons-Operator . kann wegfallen, da Substitutionen nicht mehr in der Form $a_1.a_2.\ldots.a_n.id$ (simultane Substitution) angegeben werden, sondern nur noch die Ersetzung einer Variable betrachtet wird. Dazu werden Substitutionen der Form $b/$ eingeführt, die dem alten $b.id$ entsprechen, d.h.: $b/$ hat die intuitive Bedeutung $\underline{\mathtt{1}} \mapsto b, \underline{\mathtt{n+1}} \mapsto \underline{\mathtt{n}}$.

Wir erhalten die Sorten

Terme	$a ::= \underline{\mathtt{n}} \mid (a\ a) \mid \lambda a \mid a[s]$
Substitutionen	$s ::= a/ \mid \Uparrow(s) \mid \uparrow$

Definition 4.2.1 (Regeln des $\boldsymbol{\lambda\upsilon}$-Kalküls) Der $\lambda\upsilon$-*Kalkül* wird als Termersetzungssystem $\boldsymbol{\lambda\upsilon}$ mit folgenden Regeln definiert:

Beta	$(\lambda a)b$	$\to a[b/]$
App	$(a\ b)[s]$	$\to (a[s]\ b[s])$
Lambda	$(\lambda a)[s]$	$\to \lambda(a[\Uparrow(s)])$
FVar	$\underline{\mathtt{1}}[a/]$	$\to a$
RVar	$\underline{\mathtt{n+1}}[a/]$	$\to \underline{\mathtt{n}}$
FVarLift	$\underline{\mathtt{1}}[\Uparrow(s)]$	$\to \underline{\mathtt{1}}$
RVarLift	$\underline{\mathtt{n+1}}[\Uparrow(s)]$	$\to \underline{\mathtt{n}}[s][\uparrow]$
VarShift	$\underline{\mathtt{n}}[\uparrow]$	$\to \underline{\mathtt{n+1}}$

Wir setzen ferner $\boldsymbol{\upsilon} := \boldsymbol{\lambda\upsilon} \setminus \{\mathbf{Beta}\}$ und definieren $\upsilon(t)$ als $\boldsymbol{\upsilon}$ -Normalform von t, sofern diese eindeutig existiert (d.h. $\exists\ \upsilon(t)$ mit $t \xrightarrow[\upsilon]{*} \upsilon(t)$, $\upsilon(t)$ enthält keinen $\boldsymbol{\upsilon}$ -Redex, und für jedes t' mit $t \xrightarrow[\upsilon]{*} t'$, so daß t' keinen $\boldsymbol{\upsilon}$ -Redex enthält, gilt $t' = \upsilon(t)$). □

Lemma 4.2.2 ([LRD94])
$\boldsymbol{\upsilon}$ ist terminierend, d.h. es gibt keine unendlichen Reduktionsfolgen. □

Satz 4.2.3 (Korrektheit von Beta, [BBLRD95])
$\boldsymbol{\lambda\upsilon}$ implementiert die β-Reduktion korrekt, d.h.: für alle a, b gilt:

$$a \to_\beta b \iff a \xrightarrow[\mathbf{Beta}]{} b' \text{ und } b = \upsilon(b').$$ □

Zur Konfluenz von $\boldsymbol{\lambda\upsilon}$: Durch Überschneidung der Regeln **Beta** und **App** gibt es folgendes (einziges) kritisches Paar:

$$((\lambda a)b)[s] \xrightarrow[\mathbf{Beta}]{} a[b/][s],$$
$$((\lambda a)b)[s] \xrightarrow[\upsilon]{*} (\lambda a[\Uparrow(s)])b[s] \xrightarrow[\mathbf{Beta}]{} a[\Uparrow(s)][b[s]/],$$

aber es gilt folgendes

Lemma 4.2.4 (Substitutions-Lemma, [BBLRD95])
Für alle Terme a, b und expliziten Substitutionen s gilt:

$$a[b/][s] \xleftrightarrow[\upsilon]{*} a[\Uparrow(s)][b[s]/].$$ □

Darüberhinaus besteht zwischen zwei Termen a, b und ihren υ-Normalformen $\upsilon(a), \upsilon(b)$ der folgende Zusammenhang:

Lemma 4.2.5 (Projektions-Lemma, [BBLRD95])
Für alle Terme a, b gilt:

$$a \underset{\mathbf{Beta}}{\longrightarrow} b \Longrightarrow \; \upsilon(a) \overset{*}{\underset{\beta}{\longrightarrow}} \upsilon(b).$$

Dieselbe Aussage gilt auch für alle expliziten Substitutionen s, t:

$$s \underset{\mathbf{Beta}}{\longrightarrow} t \Longrightarrow \; \upsilon(s) \overset{*}{\underset{\beta}{\longrightarrow}} \upsilon(t). \qquad \square$$

Daraus ergibt sich schließlich die Konfluenz des $\boldsymbol{\lambda\upsilon}$-Kalküls:

Satz 4.2.6 (Konfluenz von $\boldsymbol{\lambda\upsilon}$, [BBLRD95])
$\boldsymbol{\lambda\upsilon}$ *ist konfluent auf der Menge der* **Terme**. □

4.2.2 Starke Normalisierung

Mellies [Mel95] hat nachgewiesen, daß der $\lambda\sigma$-Kalkül starke Normalisierung nicht bewahrt. Das heißt, es gibt einen Term, der stark β-normalisierend, aber nicht stark $\lambda\sigma$-normalisierend ist. Der $\lambda\upsilon$-Kalkül hingegen bewahrt starke Normalisierung, wie in [BBLRD95] gezeigt wird.

Beispiel 4.2.7 (Das Gegenbeispiel, [Mel95]) Sei M der geschlossene, einfach getypte λ-Term

$$\lambda v.(\lambda x.(\lambda y.y)((\lambda z.z)x))((\lambda w.w)v),$$

der im de Bruijn-Kalkül die Form

$$\lambda(\;(\lambda(\lambda\underline{1})((\lambda\underline{1})\underline{1}))\;((\lambda\underline{1})\underline{1})\;)$$

erhält. Dann gibt es eine Folge von $\lambda\sigma$-Reduktionen, die mit M beginnt und nicht terminiert.

Unabhängig davon gibt es aber auch $\lambda\sigma$-Reduktionen, die M zu seiner Normalform $\lambda\underline{1}$ reduzieren.

Für den $\lambda\upsilon$-Kalkül gilt hingegen der folgende Satz:

Satz 4.2.8 (Bewahrung der Normalisierung in $\boldsymbol{\lambda\upsilon}$, [BBLRD95])
Es sei t ein reiner Term. Wenn t stark β-normalisierend ist, dann ist t auch stark $\lambda\upsilon$-normalisierend. □

4.2.3 Eine explizite η-Regel

Wir kommen nun auf die in Bemerkung 4.1.21 versprochene explizite Variante einer η-Regel zurück: wie bereits festgestellt, führt der $\lambda\sigma$-Kalkül zwar eine explizite β-Regel **Beta** ein, wodurch β-Reduktion als Termersetzungssystem erster Ordnung dargestellt werden kann. Für die η-Regel wird dieser Ansatz allerdings nicht eingehalten, da es sich bei der Regel **Eta** des $\lambda\sigma$-Kalküls um eine bedingte Regel handelt. Um die Bedingung zu überprüfen, ist die Gleichung $\exists\, b:\ a =_\sigma b[\uparrow]$ zu lösen, d.h. es muß ein b gefunden werden, das die Gleichung $a = b[\uparrow]$ modulo der durch §definierten Gleichheitstheorie $=_\sigma$ löst.

Definition 4.2.9 (explizite Eta-Regel) Wir erweitern die Syntax der Terme um ein neues Symbol $\perp$ zu

Terme $\quad a ::= \underline{n} \mid (a\ a) \mid \lambda a \mid a[s] \mid \perp$

und definieren die *explizite Eta-Regel* durch

Eta $\quad \lambda(a\ \underline{1}) \to a[\perp/]$.

Da $\perp$ eine neue Konstante ist, fügen wir außerdem die Regel

Const $\quad \perp[s] \to \perp$

hinzu. □

Wir geben ein Beispiel für die Berechnung der $\beta\eta$-Reduktion im $\lambda\upsilon$-Kalkül mit **Eta**-Regel.

Beispiel 4.2.10 Sei $t = (\lambda x.\lambda y.xy)z$. Dann ist $t' = tr(t, \{z\}) = (\lambda(\lambda \underline{2}\ \underline{1}))\underline{1}$[4]. Im λ-Kalkül gilt $t \to_\beta \lambda y.z\ y \to_\eta z$. Im $\lambda\upsilon$-Kalkül haben wir

$$\begin{array}{lll}
(\lambda(\lambda\underline{2}\ \underline{1}))\underline{1} & \xrightarrow[\textbf{Beta}]{} (\lambda\underline{2}\ \underline{1})[\underline{1}/] & \\
& \xrightarrow[\upsilon]{} \lambda((\underline{2}\ \underline{1})[\Uparrow(\underline{1}/)]) & \text{(Lambda)} \\
& \xrightarrow[\upsilon]{} \lambda(\underline{2}[\Uparrow(\underline{1}/)]\underline{1}[\Uparrow(\underline{1}/)]) & \text{(App)} \\
& \xrightarrow[\upsilon]{}^{+} \lambda(\underline{1}[\underline{1}/][\uparrow])\underline{1}) & \text{(RVarLift, FVarLift)} \\
& \xrightarrow[\upsilon]{} \lambda(\underline{1}[\uparrow])\underline{1} & \text{(FVar)} \\
& \xrightarrow[\upsilon]{} \lambda(\underline{2}\ \underline{1}) & \text{(VarShift)} \\
& \xrightarrow[\textbf{Eta}]{} \underline{2}[\perp/] & \\
& \xrightarrow[\upsilon]{} \underline{1} & \text{(RVar)}
\end{array}$$

Definition 4.2.11 (η'-Reduktion) Sei $a \in \Lambda$. Dann definieren wir die η'-*Reduktionsregel* durch:

$$\lambda(a\ 1) \to_{\eta'} b \quad :\Longleftrightarrow \quad \lambda(a\ 1) \xrightarrow[\textbf{Eta}]{} a[\perp/] \text{ und } \upsilon(a[\perp/]) = b \in \Lambda.$$ □

Satz 4.2.12 (Korrektheit von Eta, [Bri95])
Für alle a, b gilt:

$$a \to_\eta b \iff a \to_{\eta'} b.$$ □

[4] Wir verwenden das Referential $\mathcal{R} = \{z\}$ für die freie Variable z.

4.2.4 Der getypte $\lambda\upsilon$-Kalkül

Es gibt den $\lambda\upsilon$-Kalkül auch in einer getypten Variante.

Definition 4.2.13 (getypte $\lambda\upsilon$-Terme) Die Menge der *getypten $\lambda\upsilon$-Terme*, $\Lambda\Upsilon$, über der Menge der Typen $\mathcal{T}$ hat folgende Grammatik:

Typen	$A ::= \alpha$	mit $\alpha \in \mathcal{T}$
Terme	$a ::= \underline{\mathtt{n}} \mid (a\ a) \mid \lambda A.a \mid a[s]$	
Substitutionen	$s ::= a/ \mid \Uparrow(s) \mid \uparrow$	
Kontexte	$\Gamma ::= [\,] \mid A \cdot \Gamma$	

Die Regeln für korrekt getypte Terme und Substitutionen sind:

Terme:

$$\frac{\Gamma \vdash a : A \to B \quad \Gamma \vdash b : A}{\Gamma \vdash (a\ b) : B} \qquad \frac{A \cdot \Gamma \vdash a : B}{\Gamma \vdash \lambda A.a : A \to B}$$

$$\frac{\Gamma \vdash a : A \quad \Delta \vdash s : \Gamma}{\Delta \vdash a[s] : A} \qquad \frac{}{A \cdot \Gamma \vdash \underline{\mathtt{1}} : A} \qquad \frac{\Gamma \vdash \underline{\mathtt{n}} : A}{B \cdot \Gamma \vdash \underline{\mathtt{n+1}} : A}$$

Substitutionen:

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash a/ : A \cdot \Gamma} \qquad \frac{}{A \cdot \Gamma \vdash \uparrow : \Gamma} \qquad \frac{\Gamma \vdash s : \Delta}{A \cdot \Gamma \vdash \Uparrow(s) : A \cdot \Delta}$$

□

Satz 4.2.14 ([BBLRD95])
Die $\lambda\upsilon$-Regeln sind typ-erhaltend. □

Satz 4.2.15 ([BBLRD95])
Jeder reine, korrekt getypte Term aus $\Lambda\Upsilon$ ist stark normalisierbar. □

4.2.5 Zusätzliche Variablen und Funktionen

Wir erweitern die (ungetypten) $\lambda\upsilon$-Terme nun um zusätzliche Variablen; dies sind wie im $\lambda\sigma$-Kalkül die $X \in \mathcal{X}$. Außerdem fügen wir Funktionssymbole f hinzu. Diese sollen durch explizite Substitutionen nicht verändert werden; wir werden dem Kalkül also neue Regeln **FreeVar** $X[s] \to X$ und **Function** $f[s] \to f$ hinzufügen. Außerdem verwenden wir die explizite **Eta**-Regel aus Abschnitt 4.2.3. Damit erhalten wir den folgenden Kalkül:

Definition 4.2.16 (Der $\lambda\upsilon\eta F$-Kalkül) Die Terme des *$\lambda\upsilon\eta F$-Kalküls* werden durch folgende Grammatik definiert:

Terme	$a ::= \underline{\mathtt{n}} \mid X \mid f \mid (a\ a) \mid \lambda a \mid a[s] \mid \bot$
Substitutionen	$s ::= a/ \mid \Uparrow(s) \mid \uparrow$

Das Termersetzungssystem $\boldsymbol{\lambda\upsilon\eta F}$ besteht aus folgenden Regeln:

Beta	$(\lambda a)b$	$\to a[b/]$
Eta	$\lambda(a\ \underline{\mathtt{1}})$	$\to a[\bot/]$
App	$(a\ b)[s]$	$\to (a[s]\ b[s])$
Lambda	$(\lambda a)[s]$	$\to \lambda(a[\Uparrow(s)])$
FVar	$\underline{\mathtt{1}}[a/]$	$\to a$
RVar	$\underline{\mathtt{n+1}}[a/]$	$\to \underline{\mathtt{n}}$
FVarLift	$\underline{\mathtt{1}}[\Uparrow(s)]$	$\to \underline{\mathtt{1}}$
RVarLift	$\underline{\mathtt{n+1}}[\Uparrow(s)]$	$\to \underline{\mathtt{n}}[s][\uparrow]$
VarShift	$\underline{\mathtt{n}}[\uparrow]$	$\to \underline{\mathtt{n+1}}$
FreeVar	$X[s]$	$\to X$
Function	$f[s]$	$\to f$
Const	$\bot[s]$	$\to \bot$

Mit $\boldsymbol{\upsilon F}$ bezeichnen wir $\boldsymbol{\lambda\upsilon\eta F}\backslash\{\textbf{Beta}, \textbf{Eta}\}$. □

Die Regeln **FreeVar**, **Function** und **Const** können auch in einer Regel zusammengefaßt werden, wenn wir $\bot$, $\mathcal{X}$ und die Menge der Funktionssymbole zu einer Menge von Konstanten zusammenfassen.

Briaud hat in einer demnächst erscheinenden Arbeit die folgenden Aussagen gezeigt:

Satz 4.2.17 (Hauptsatz über $\boldsymbol{\lambda\upsilon\eta F}$, [Bri97a, Bri97b])
Für obiges System $\boldsymbol{\lambda\upsilon\eta F}$ gelten die folgenden Aussagen:

1. $\boldsymbol{\lambda\upsilon\eta F}$ *implementiert korrekt β- und η-Reduktion, d.h.*
   - $a \to_\beta b \iff a \xrightarrow[\textbf{Beta}]{} b'$ *und* $b = \upsilon(b')$, *und*
   - $a \to_\eta b \iff a \to_{\eta'} b$,
     wobei η' definiert ist durch
     $$\lambda(a\ 1) \to_{\eta'} b :\iff \lambda(a\ 1) \xrightarrow[\textbf{Eta}]{} a[\bot/],\ \upsilon(a[\bot/]) = b,$$
     $$\text{und } \bot \text{ kommt nicht in } b \text{ vor.}$$
2. $\boldsymbol{\lambda\upsilon\eta F}' := \boldsymbol{\lambda\upsilon\eta F} \setminus \{\textbf{Eta}\} \cup \{\eta'\}$ *ist konfluent auf der Menge der* **Terme**.
3. $\boldsymbol{\lambda\upsilon\eta F}$ *erhält starke Normalisierung, d.h.: wenn für einen Term alle $\beta\eta$-Reduktionsfolgen terminieren, dann terminieren auch alle $\boldsymbol{\lambda\upsilon\eta F}$-Reduktionsfolgen.*

Die Aussagen bleiben gültig, wenn getypte Terme verwendet werden. □

Bemerkung 4.2.18 Wir gehen in dieser Arbeit grundsätzlich davon aus, daß alle Terme korrekt getypt sind. Daraus ergibt sich unter anderem, daß jeder Term eine $\beta\eta$-Normalform hat, da der getypte λ-Kalkül stark normalisierend (d.h. terminierend) ist.

Bei der Diskussion des LNT-Kalküls verzichten wir auf die Angabe der Typen, da die zusätzliche Information die Darstellung nur unübersichtlich macht. Aus obigem Resultat folgt also, daß alle $\lambda\upsilon\eta F$-Reduktionsfolgen, die wir betrachten, terminieren. □

Kapitel 5

Definierende Bäume mit Expliziten Substitutionen

5.1 Analyse der LNT-Regeln

In Kapitel 3 haben wir eine Erweiterung des Needed-Narrowing auf Funktionen höherer Ordnung vorgestellt: die neue Strategie LNT benutzt definierende Bäume (höherer Ordnung) [HP96] zur Beschreibung induktiv sequentieller Termersetzungssysteme.

Wir stellen im folgenden die Frage, inwiefern die Substitutionen σ in den Regeln des Systems LNT Substitutionen höherer Ordnung sind.

Eine Substitution höherer Ordnung unterscheidet sich von einer Substitution erster Ordnung durch die Möglichkeit des Auftretens der beiden folgenden Probleme (siehe auch Abschnitt 4.1.1):

1. „Variable capture“: $\{y \mapsto x\}(\lambda x.y) \neq \lambda x.x$,
2. Gebundene Variablen dürfen nicht ersetzt werden: $\{x \mapsto a\}(\lambda x.x) \neq \lambda x.a$,

falls dies Substitutionen höherer Ordnung sind. Im Fall erster Ordnung wären die Ungleichheitszeichen durch Gleichheitszeichen zu ersetzen (aber der λ-Kalkül ist eben *kein* System erster Ordnung).

Diese Probleme treten nicht bei jeder Substitution auf, und wir definieren nun den Begriff der *FO-Zulässigkeit* für Paare (t, σ), um zu unterscheiden, welche Ersetzungsschritte problematisch sind und welche nicht.

Definition 5.1.1 (FO-zulässige Substitution) Sei t ein beliebiger Term und $\sigma = \{\overline{X_k \mapsto s_k}\}$ eine Substitution. Dann heißt (t, σ) *FO-zulässig*, falls die beiden folgenden Bedingungen erfüllt sind:

1. Falls $X \in FV(s_i)$ für ein i, dann enthält t keinen Unterterm $\lambda X.t'$, so daß X_i in t' vorkommt, und

2. $X_1, \ldots, X_k \notin BV(t)$

Außerdem heißt σ *FO-zulässig für eine Anfrage* $G = t_1 \rightarrow^? Z_1, \ldots, t_m \rightarrow^? Z_m$, falls alle (t_i, σ) FO-zulässig sind. Wir schreiben dann auch: (G, σ) FO-zulässig. □

Die Bedingungen verhindern das Auftreten der beiden oben angegebenen Probleme – FO-zulässige Substitutionen sind also Substitutionen höherer Ordnung, bei denen der Einsatz eines Substitutionsmechanismus erster Ordnung möglich ist, ohne falsche Ergebnisse zu produzieren.

Beispiel 5.1.2 Die Paare $(\lambda x.y, \{y \mapsto x\})$ und $(\lambda x.x, \{x \mapsto a\})$ sind *nicht* FO-zulässig. (Im ersten Fall ist $x \in FV(s)$ (mit $s = x$), und $t = \lambda x.y$ enthält einen Unterterm $\lambda x.t'$ (nämlich sich selbst), in dem y vorkommt. Im zweiten Fall ist $x \in BV(t)$, wobei $t = \lambda x.x$.)

Das Paar $(\lambda \overline{x_k}.F(\overline{x_k}), \{F \mapsto \lambda \overline{x_k}.H(\overline{x_k})\})$ ist FO-zulässig.

Wir untersuchen nun in den sechs Regeln des Systems LNT, ob die auftretenden Substitutionen FO-zulässig sind.

Satz 5.1.3
Alle in LNT auftauchenden Substitutionen sind FO-zulässig.

Beweis. Die zweite Bedingung ist offensichtlich für jede der Regeln erfüllt: keines der Z, X, X_i taucht jemals gebunden auf. Wir überprüfen also noch die erste Bedingung.

1. Die **Bind**-Regel: $e \rightarrow^? Z, G \Rightarrow^{\varnothing} \sigma(G)$ mit $\sigma = \{Z \mapsto e\}$, falls e kein case-Ausdruck.
   Falls $G = \varnothing$, dann ist $e \rightarrow^? Z$ die vollständige Anfrage, und durch die Bindung $\sigma = \{Z \mapsto e\}$ wird das Verfahren beendet. In diesem Fall ist nichts zu zeigen.
   Ansonsten ist $G = t \rightarrow^? Z', G'$, und die Variable Z kommt nur einmal in t vor, da Z durch einen **Case Eval**-Schritt als frische Variable eingeführt wurde. Die erste Bedingung ist erfüllt, denn: wenn X eine freie Variable in e ist, dann ist X entweder eine der gesuchten Variablen oder wurde durch eine der *Guess-Regeln*[1] eingeführt – in jedem Fall enthält t keinen λ-Binder λX. Damit ist (t, σ) FO-zulässig, und wegen des Nichtauftretens von Z in G' auch (G, σ).

[1] Die Guess-Regeln sind **Imitation**, **Function Guess** und **Projection**.

2. Die **Case Select**-Regel: $\lambda\overline{x_k}.case\ \lambda\overline{y_l}.v(\overline{t_m})\ of\ \overline{p_n : \mathcal{X}_n} \to^? Z, G \Rightarrow^{\varnothing} \lambda\overline{x_k}.\sigma(\mathcal{X}_i) \to^? Z, G$, falls $p_i = \lambda\overline{y_l}.v(\overline{X_m(\overline{x_k}, \overline{y_l})})$ und $\sigma = \{\overline{X_m \mapsto \lambda\overline{x_k}, \overline{y_l}.t_m}\}$.
   Die Substitution $\sigma = \{\overline{X_m \mapsto \lambda\overline{x_k}, \overline{y_l}.t_m}\}$ wird nur auf $\mathcal{X}_i$ angewendet. Sei $X \in FV(\lambda\overline{x_k}, \overline{y_l}.t_j)$ für ein $j \in \{1, \ldots, m\}$. Zu zeigen ist: $\mathcal{X}_i$ enthält keinen Unterterm $\lambda X.t'$, so daß X_j in t' vorkommt. Das ist aber klar, da überhaupt kein λ-Binder λX vorkommt. (Da $X \notin \{\overline{x_k}, \overline{y_l}\}$, ist X eine der nirgends gebundenen Variablen: entweder eine gesuchte oder eine durch eine Guess-Regel eingeführte Variable.) Also ist $(G, \{X_j \mapsto \lambda\overline{x_k}, \overline{y_l}.t_j\})$ FO-zulässig für jedes j, d.h. (G, σ) ist FO-zulässig.

3. Die **Imitation**-Regel: $\lambda\overline{x_k}.case\ \lambda\overline{y_l}.X(\overline{t_m})\ of\ \overline{p_n : \mathcal{X}_n} \to^? Z, G \Rightarrow^{\sigma} \sigma(\lambda\overline{x_k}.case\ \lambda\overline{y_l}.X(\overline{t_m})\ of\ \overline{p_n : \mathcal{X}_n} \to^? Z, G)$, falls $p_i = \lambda\overline{y_l}.c(\overline{X_o(\overline{x_k}, \overline{y_l})})$ (c Konstruktor) und $\sigma = \{X \mapsto \lambda\overline{x_m}.c(\overline{H_o(\overline{x_m})})\}$.
   Die freien Variablen in $\lambda\overline{x_m}.c(\overline{H_o(\overline{x_m})})$ sind genau die $\overline{H_o}$. Kein Term enthält einen λ-Binder λH_j, $j = 1, \ldots, o$, da die $\overline{H_o}$ durch dieses σ neu eingeführt werden. Damit ist (G, σ) FO-zulässig.

4. Für die **Function Guess**- und **Projection**-Regeln können wir dasselbe Argument wie für die **Imitation**-Regel verwenden.

5. Die **Case Eval**-Regel: $\lambda\overline{x_k}.case\ \lambda\overline{y_l}.f(\overline{t_m})\ of\ \overline{p_n : \mathcal{X}_n} \to^? Z, G \Rightarrow^{\varnothing} \lambda\overline{x_k}, \overline{y_l}.\sigma(\mathcal{X}) \to^? X, \lambda\overline{x_k}.case\ \lambda\overline{y_l}.X(\overline{x_k}, \overline{y_l})\ of\ \overline{p_n : \mathcal{X}_n} \to^? Z, G$
   mit $\sigma = \{\overline{X_m \mapsto \lambda\overline{x_k}, \overline{y_l}.t_m}\}$, $f(\overline{X_m(\overline{x_k}, \overline{y_l})}) \to \mathcal{X}$ ist eine $\overline{x_k}, \overline{y_l}$-gehobene Regel.
   Die Substitution $\sigma = \{\overline{X_m \mapsto \lambda\overline{x_k}, \overline{y_l}.t_m}\}$ wird nur auf $\mathcal{X}$ angewendet. Wie bei der **Case Select**-Regel gibt es kein $X \in FV(\lambda\overline{x_k}, \overline{y_l}.t_j)$, für das ein λ-Binder λX existiert: (G, σ) ist FO-zulässig. ■

Damit können wir alle Substitutions-Schritte mit einem Ersetzungsalgorithmus erster Ordnung nach folgendem Schema durchführen:

Definition 5.1.4 (FO-Substitution für case-Terme) Die *FO-Substitution* $\sigma := \langle X/s \rangle$ wird induktiv definiert über:

- $\sigma(X) := s$,
- $\sigma(Y) := Y$, falls Y Variable, $Y \neq X$,
- $\sigma(c) := c$, falls c Konstante,
- $\sigma(\lambda y.t) := \lambda y.\sigma(t)$,
- $\sigma(t_1\ t_2) := (\sigma(t_1)\ \sigma(t_2))$,
- $\sigma(case\ t\ of\ \overline{p_n : \mathcal{X}_n}) := case\ \sigma(t)\ of\ \overline{p_n : \sigma(\mathcal{X}_n)}$. □

Wir können also alle in LNT auftretenden Substitutionen der Form $\{\overline{X_m \mapsto t_m}\}$ durch $\langle\overline{X_m/t_m}\rangle$ ersetzen.

Bemerkung 5.1.5 Wir verwenden die Notation $\langle X/s\rangle$ zur Unterscheidung von „herkömmlichen" Substitutionen $\{X \mapsto s\}$. Für den Fall, daß eine Substitution $\{X \mapsto s\}$ für einen Term t FO-zulässig ist, sind natürlich $\langle X/s\rangle t$ und $\{X \mapsto s\}t$ identisch. □

Das LNT-Verfahren enthält aber noch immer Substitutionen, die nicht auf diese Weise verarbeitet werden können: durch die β-Regel des λ-Kalküls kommen diese ins Spiel.

In Kapitel 4 haben wir gesehen, wie β- und η-Reduktion durch einen Kalkül mit expliziten Substitutionen berechnet werden können, der ein Termersetzungssystem erster Ordnung ist. Wir wollen nun den $\lambda\upsilon$-Kalkül auf das System LNT anwenden.

5.2 Definierende Bäume in de Bruijn-Notation

In diesem Abschnitt stellen wir ein einfaches Verfahren vor, mit dem sich λ-Terme, die *case*-Ausdrücke enthalten, in de Bruijn-Notation übersetzen lassen.

Zunächst klären wir, welche Variablen wir durch de Bruijn-Indizes ersetzen wollen. In Abschnitt 4.1.2 wurden bei der Übersetzung $\Lambda(\mathcal{V}, \mathcal{X}) \to \Lambda_{DB}(\mathcal{X})$ nur die Variablen $v \in \mathcal{V}$ ersetzt – die Unifikationsvariablen aus $\mathcal{X}$ blieben unverändert. Zur Erinnerung: die Unifikationsvariablen sind die Unbekannten im Unifikationsprozeß, für die eine Lösung (d.h. Substitution) gefunden werden soll.

Übertragen wir dies auf die Menge der möglichen Anfragen für das System LNT.

- Zunächst haben wir die Variablen $\overline{x_k}$ und $\overline{y_l}$, die stets gebunden sind – sie können also nie substituiert werden, und daher werden wir sie in de Bruijn-Indizes übersetzen.
- Dann gibt es die Variablen $F, G, \ldots$, die in der initialen Anfrage frei vorkommen und durch das Verfahren gelöst werden sollen. Diese werden wir nicht übersetzen, da sie die selbe Rolle haben wie die Unifikationsvariablen bei der $\lambda\sigma$-Unifikation.
- Durch die Guess-Regeln werden neue Variablen $\overline{H_o}$ eingeführt, für die der Kalkül ebenfalls Lösungen berechnen muß. Zwar sind dies nur Zwischenergebnisse, die nicht zurückgegeben werden, wir können sie aber wie den vorangegangenen Typ behandeln.

- Schließlich gibt es die Variablen X_m, die in den Mustern p_i als Argumente der definierten Funktionen f auftreten. Diese kommen nur in Argumenten von *case* vor und werden durch **Case Select**-Schritte entfernt. Auch diese Variablen werden wir nicht übersetzen.

Wir können die de Bruijn-Übersetzung (Definition 4.1.8 in Abschnitt 4.1.2) verwenden, wenn wir vorher *case*-Terme in $\Lambda(\mathcal{V}, \mathcal{X})$ einbetten:

Definition 5.2.1 (Einbettung von *case*-Termen) Es sei $\mathcal{V}$ eine Menge von Variablen mit $x, x_1, x_2, \ldots, y, y_1, y_2, \ldots \in \mathcal{V}$ und $\mathcal{X}$ eine Menge von Variablen und Konstanten, die alle Konstruktor- und Funktionssymbole c, f, alle $X, Y, Z, H, \ldots$ sowie neue $(2n+1)$-stellige Funktionssymbole $f^n_{\mathbf{case}}$ (für alle $n \in \mathbb{N}$) enthält. Dann ist durch die Abbildung

$$\mathit{case}\ t\ \mathit{of}\ \overline{p_n : \mathcal{X}_n} \quad \mapsto \quad f^n_{\mathbf{case}}(t, p_1, \mathcal{X}_1, \ldots, p_n, \mathcal{X}_n)$$

(über homomorphe Fortsetzung) eine Einbettung

$$\mathbf{i} : \mathit{Case} \to \Lambda(\mathcal{V}, \mathcal{X})$$

definiert. □

Auf eingebettete Terme $\mathbf{i}(t)$ kann dann die de Bruijn-Übersetzung tr (Definition 4.1.8) mit leerem Referential $\mathcal{R} = [\,]$ (alle x_i, y_j sind gebunden) angewendet werden, und wir erhalten $tr(\mathbf{i}(t), [\,])$ in de Bruijn-Notation.

Um die Notation der *case*-Ausdrücke zu erhalten, definieren wir noch eine Abbildung $\mathbf{e}$, die aus dem erhaltenen Term einen *case*-Ausdruck – allerdings in de Bruijn-Notation – macht:

Definition 5.2.2 Wir definieren $\mathbf{e}$ als homomorphe Fortsetzung der Abbildung, die durch

$$f^n_{\mathbf{case}}(t, p_1, \mathcal{X}_1, \ldots, p_n, \mathcal{X}_n) \quad \mapsto \quad \mathit{case}\ t\ \mathit{of}\ \overline{p_n : \mathcal{X}_n}$$

(für alle $n \in \mathbb{N}$) definiert wird. □

Definition 5.2.3 (de Bruijn-Übersetzung von *case*-Termen) Es sei t ein case-Term. Dann ist die *de Bruijn-Übersetzung* von t definiert als

$$DB(t) := \mathbf{e}(tr(\mathbf{i}(t), [\,])).$$

Anfragen G werden durch $DB(\varnothing) := \varnothing$ und $DB(t \to^? Z, G) := DB(t) \to^? Z, DB(G)$ übersetzt. □

Im folgenden wollen wir noch für Substitutionen eine Übersetzung definieren, sofern dies möglich ist:

Definition 5.2.4 (de Bruijn-Übersetzung einer FO-zul. Substitution) Es sei $\sigma = \{\overline{X_m \mapsto t_m}\}$ eine FO-zulässige Substitution (für eine vorgegebene Anfrage G) mit $FV(t_i) \cap \mathcal{V} = \varnothing$ für alle i. Dann ist $\sigma' := DB(\sigma) := \langle \overline{X_m / DB(t_m)} \rangle$ die *de Bruijn-Übersetzung* von σ. □

Satz 5.2.5
Es sei G eine beliebige Anfrage für das System LNT und σ eine Substitution, für die $DB(\sigma)$ definiert ist. Dann gilt $DB(\sigma(G)) = \sigma'(DB(G))$, d.h.: das folgende kommutative Diagramm ist gültig.

$$\begin{array}{ccc} G & \xrightarrow{\sigma} & \sigma(G) \\ {\scriptstyle DB}\downarrow & & \downarrow{\scriptstyle DB} \\ G' & \xrightarrow[\sigma'=DB(\sigma)]{} & \sigma'(G') \end{array}$$

Beweis. Sei G eine solche Anfrage. Dann gilt entweder $G = t \rightarrow^? X$ oder $G = t \rightarrow^? X, G_1$. Wegen $\sigma(t \rightarrow^? X, G_1) = \sigma(t \rightarrow^? X), \sigma(G_1)$ und $DB(t \rightarrow^? X, G_1) = DB(t \rightarrow^? X), DB(G_1)$ müssen wir nur den ersten Fall untersuchen, also $G = t \rightarrow^? X$.

Es sei $\sigma = \{\overline{X_m \mapsto t_m}\}$ eine Substitution, für die $DB(\sigma)$ definiert ist. Damit gilt nach Definition 5.2.4: $FV(t_i) \cap \mathcal{V} = \varnothing$ für alle i.

Zu zeigen ist also: $DB(\sigma(t)) = DB(\sigma)(DB(t))$. Um diese Aussage über strukturelle Induktion zeigen zu können, zeigen wir sogar für ein beliebiges Referential $\mathcal{R}$[2]: $tr(\sigma(t), \mathcal{R}) = DB(\sigma)(tr(t, \mathcal{R}))$.

- $t = X_i$ für ein $i \in \{1, \ldots, m\}$. Dann ist $\sigma(t) = \sigma(X_i) = t_i$, nach Definition von σ. Also gilt $tr(\sigma(t), \mathcal{R}) = tr(t_i, \mathcal{R})$. In der anderen Richtung haben wir $DB(\sigma)(tr(t), \mathcal{R}) = DB(\sigma)(tr(X_i, \mathcal{R})) = DB(\sigma)(X_i)$ (nach Definition von tr) $= tr(t_i, \mathcal{R})$ (nach Definition von $DB(\sigma)$).

- $t = Y, Y \notin \{X_1, \ldots, X_m\}$. Dann ist $tr(\sigma(t), \mathcal{R}) = tr(\sigma(Y), \mathcal{R}) = tr(Y, \mathcal{R})$ $= Y$ und $DB(\sigma)(tr(t, \mathcal{R})) = DB(\sigma)(tr(Y, \mathcal{R})) = DB(\sigma)(Y) = Y$.

- $t = c$, c Konstante. Dann ist $tr(\sigma(t), \mathcal{R}) = tr(\sigma(c), \mathcal{R}) = tr(c, \mathcal{R}) = c$ und $DB(\sigma)(tr(t, \mathcal{R})) = DB(\sigma)(tr(c, \mathcal{R})) = DB(\sigma)(c) = c$.

- $t = \lambda y.t_1$. Es gilt $y \notin \{X_1, \ldots, X_m\}$, da σ FO-zulässig für t ist. Daher ist $tr(\sigma(t), \mathcal{R}) = tr(\sigma(\lambda y.t_1), \mathcal{R}) = tr(\lambda y.\sigma(t_1), \mathcal{R}) = \lambda tr(\sigma(t_1), y.\mathcal{R}) =$ (Ind.) $\lambda DB(\sigma)(tr(t_1, y.\mathcal{R})) = DB(\sigma)(tr(\lambda y.t_1, \mathcal{R})) = DB(\sigma)(tr(t, \mathcal{R}))$.

- $t = (t_1\ t_2)$. Dann gilt $tr(\sigma(t), \mathcal{R}) = tr(\sigma((t_1\ t_2)), \mathcal{R}) = tr((\sigma(t_1)\ \sigma(t_2)), \mathcal{R})$ $= (tr(\sigma(t_1), \mathcal{R})\ \ tr(\sigma(t_2), \mathcal{R}))$ $=$ (Induktion) $(DB(\sigma)(tr(t_1, \mathcal{R}))$

[2] Zur Erinnerung: Per Definition ist $DB(t) = tr(t, [])$ mit leerem Referential [].

$DB(\sigma)(tr(t_2, \mathcal{R}))) = DB(\sigma)((tr(t_1, \mathcal{R})\ tr(t_2, \mathcal{R}))) = DB(\sigma)(tr((t_1\ t_2), \mathcal{R})) = DB(\sigma)(tr(t, \mathcal{R}))$.

- $t = case\ t_1\ of\ \overline{p_n : \mathcal{X}_n}$. Hier ist nichts zu zeigen, da t intern als

  $$f^n_{\mathbf{case}}(t_1, p_1, \mathcal{X}_1, \ldots, p_n, \mathcal{X}_n)$$

  verarbeitet wird. ■

Da wir oft mit Untertermen arbeiten, deren Variablen im ganzen Term gebunden sind, definieren wir eine Übersetzung im Kontext dieser λ-Binder:

Definition 5.2.6 (Übersetzung im Kontext von $\boldsymbol{\lambda}$-Bindern $\boldsymbol{\lambda\overline{x_k}}$)
Es sei t ein beliebiger Term. Für die de Bruijn-Übersetzung von $\lambda\overline{x_k}.t$ gilt: $DB(\lambda\overline{x_k}.t) = \lambda^k t'$ für einen Term t'. Dieses t' nennen wir die *de Bruijn-Übersetzung von t im Kontext der λ-Binder $\lambda\overline{x_k}$*. Bezeichnung: $t' = DB_{[\lambda\overline{x_k}]}(t)$. □

Definition 5.2.7 (Übersetzung gehobener Regeln) Es sei $f(X_1, \ldots, X_m) \to t$ eine Regel r aus $\mathcal{R}$, und $f(X'_1(\overline{x_k}), \ldots, X'_m(\overline{x_k})) \to t'$ sei die $\overline{x_k}$-gehobene Form dieser Regel gemäß Definition 3.4.2. Wir führen (ausschließlich zum Zweck der Übersetzung) ein neues Funktionssymbol $f_\to$ ein und fassen die Regel als Term $f_\to(f(X'_1(\overline{x_k}), \ldots, X'_m(\overline{x_k})), t')$ auf. Vor der Übersetzung müssen wir noch die $\overline{x_k}$ durch $\lambda\overline{x_k}$ binden. Wir definieren die linken und rechten Seiten L und R der *k-gehobenen Regel* $L \to R$ durch

$$DB_{[\lambda\overline{x_k}]}(f_\to(f(X'_1(\overline{x_k}), \ldots, X'_m(\overline{x_k})), t')) = f_\to(L, R).$$

Wir schreiben $DB^{\frown k}(r) = L \to R$. □

Bemerkung 5.2.8 Mit den Bezeichnungen der obigen Definition gilt offensichtlich:

$$DB_{[\lambda\overline{x_k}]}(f(X'_1(\overline{x_k}), \ldots, X'_m(\overline{x_k}))) = L = f(\overline{X'_m(\underline{\mathtt{k}}, \ldots, \underline{\mathtt{1}})}),$$
$$DB_{[\lambda\overline{x_k}]}(t') = R.$$ □

Bemerkung 5.2.9 Ein ursprünglicher Ansatz war, k-Lifter im Sinne der $\overline{x_k}$-Lifter durch $\sigma = \{F \mapsto (\rho F)(\underline{\mathtt{k}}, \ldots, \underline{\mathtt{1}}) \mid F \in FV(t)\}$ einzuführen und auf eine nicht gehobene Übersetzung der Regel anzuwenden. Dies funktioniert aber nicht, da sich dann die de Bruijn-Indizes $\underline{\mathtt{1}}, \ldots, \underline{\mathtt{k}}$ eventuell auf falsche Variablen beziehen. □

5.3 Das System LNT$\lambda\upsilon$

Nach der Transformation in den $\lambda\upsilon$-Kalkül erhalten die LNT-Regeln folgende Gestalt:

Definition 5.3.1 (System LNT$\boldsymbol{\lambda\upsilon}$) Das System LNT$\lambda\upsilon$ besteht aus den folgenden Regeln:

Bind
$e \rightarrow^? Z, G \Rightarrow^{\varnothing} \langle Z/e\rangle(G)$

Case Select
$\lambda^k case\ \lambda^l v(\overline{t_m})\ of\ \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G \Rightarrow^{\varnothing} \lambda^k\langle\overline{X_m/\lambda^{k+l}t_m}\rangle\mathcal{X}_i \rightarrow^? Z, G,$
falls $p_i = \lambda^l v(\overline{X_m(\underline{\texttt{k+l}}, \ldots, \underline{\texttt{1}})})$

Imitation
$\lambda^k case\ \lambda^l X(\overline{t_m})\ of\ \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G \Rightarrow^{\sigma} \sigma\big(\lambda^k case\ \lambda^l X(\overline{t_m})\ of\ \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G\big),$
falls $p_i = \lambda^l c(\overline{X_o(\underline{\texttt{k+l}}, \ldots, \underline{\texttt{1}})}),$
wobei $\sigma = \langle X/\lambda^m c(\overline{H_o(\underline{\texttt{m}}, \ldots, \underline{\texttt{1}})})\rangle$

Function Guess
$\lambda^k case\ \lambda^l X(\overline{t_m})\ of\ \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G \Rightarrow^{\sigma} \sigma\big(\lambda^k case\ \lambda^l X(\overline{t_m})\ of\ \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G\big),$
falls $\lambda^{k+l} X(\overline{t_m})$ kein Muster höherer Ordnung
und f eine definierte Funktion ist,
wobei $\sigma = \langle X/\lambda^m f(\overline{H_o(\underline{\texttt{m}}, \ldots, \underline{\texttt{1}})})\rangle$

Projection
$\lambda^k case\ \lambda^l X(\overline{t_m})\ of\ \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G \Rightarrow^{\sigma} \sigma\big(\lambda^k case\ \lambda^l X(\overline{t_m})\ of\ \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G\big),$
wobei $\sigma = \langle X/\lambda^m \underline{\texttt{i}}(\overline{H_o(\underline{\texttt{m}}, \ldots, \underline{\texttt{1}})})\rangle$

Case Eval
$\lambda^k case\ \lambda^l f(\overline{t_m})\ of\ \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G$
$\Rightarrow^{\varnothing} \lambda^{k+l}\langle\overline{X_m/\lambda^{k+l}t_m}\rangle(\mathcal{X}) \rightarrow^? X, \lambda^k case\ \lambda^l X(\underline{\texttt{k+l}}, \ldots, \underline{\texttt{1}})\ of\ \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G,$
falls $f(\overline{X_m(\underline{\texttt{k+l}}, \ldots, \underline{\texttt{1}})}) \rightarrow \mathcal{X}$ die Übersetzung einer $\overline{x_k}, \overline{y_l}$-gehobenen Regel ist. □

Definition 5.3.2 (AppCase-Regel) Für die Verarbeitung von Case-Ausdrücken fügen wir dem $\boldsymbol{\lambda\upsilon\eta F}$-Kalkül noch die folgende Regel hinzu:

AppCase $\quad (case\ t\ of\ \overline{p_n : \mathcal{X}_n})[s] \rightarrow case\ t[s]\ of\ \overline{p_n[s] : \mathcal{X}_n[s]}$ □

Diese Regel dient nur der Vereinfachung: da wir *case* t *of* $\overline{p_n : \mathcal{X}_n}$ als Kurzschreibweise für $f^n_{\mathbf{case}}(t, p_1, \mathcal{X}_1, \ldots, p_n, \mathcal{X}_n)$ interpretieren, erspart **AppCase** eine Folge von **App**- und **Function**-Schritten.

5.4 Die Reduktionsstrategie für LNT$\lambda\upsilon$

Definition 5.4.1 (Reduktionsstrategie für LNT$\boldsymbol{\lambda\upsilon}$)
Wir definieren folgende *Reduktionsstrategie* für LNT$\lambda\upsilon$: Jeder LNT$\lambda\upsilon$-Reduktionsschritt besteht aus der Anwendung einer LNT$\lambda\upsilon$-Regel mit anschließender $\boldsymbol{\lambda\upsilon\eta F}$-Normalisierung der durch die LNT-Regel veränderten Teilanfragen (d.h. $\boldsymbol{\lambda\upsilon\eta F}$-Redexe werden reduziert, bis der Term in $\boldsymbol{\lambda\upsilon\eta F}$-Normalform ist).

Für die Reduktion mit $\boldsymbol{\lambda\upsilon\eta F}$ fordern wir zudem, daß nach jedem **Eta**-Schritt $\lambda(a\,\underline{1}) \to a[\perp/]$ zunächst der Unterterm $a[\perp/]$ υ-normalisiert wird. Kommt nach dieser Normalisierung immer noch das $\perp$-Symbol vor, wird der **Eta**-Schritt verworfen.[3] Ebenfalls verwerfen wir einen **Eta**-Schritt, wenn dadurch ein LNT$\lambda\upsilon$-Redex zerstört wird. □

Beispiel 5.4.2 Auf ein Zwischenergebnis der Form *case* $\lambda\cos(\underline{1})$ *of* $\lambda\cos(\underline{1})$: ... wird kein **Eta**-Schritt für $\lambda\cos(\underline{1})$ angewendet, da dies den **Case Select**-Redex zerstören würde.

Definition 5.4.3 (LNT$\boldsymbol{\lambda\upsilon}$-Ableitung) Wir definieren die Relation $\xRightarrow[LNT\lambda\upsilon]{}$ wie folgt:

$$G \xRightarrow[LNT\lambda\upsilon]{}{}^{\sigma} G', \text{ falls } G \Rightarrow^{\sigma} G_1 \text{ und } G_1 \xRightarrow[\lambda\upsilon\eta F]{*} G',$$

wobei G_1 unter Beachtung der Reduktionsstrategie (Definition 5.4.1) zu G' normalisiert wird. Eine Folge

$$t_0 \xRightarrow[LNT\lambda\upsilon]{}{}^{\theta_1} t_1 \xRightarrow[LNT\lambda\upsilon]{}{}^{\theta_2} t_2 \xRightarrow[LNT\lambda\upsilon]{}{}^{\theta_3} \cdots \xRightarrow[LNT\lambda\upsilon]{}{}^{\theta_n} t_n$$

schreiben wir auch kurz als $t_0 \xRightarrow[LNT\lambda\upsilon]{*}{}^{\theta} t_n$ mit $\theta := \theta_1 \circ \cdots \circ \theta_n$. □

5.5 Äquivalenz von LNT und LNT$\lambda\upsilon$

Aufgabe des neuen Systems LNT$\lambda\upsilon$ ist es, das System LNT zu ersetzen. Wir fordern also eine Äquivalenz der beiden Systeme in dem Sinne, daß für jede mögliche Anfrage mit LNT$\lambda\upsilon$ dieselben Lösungen berechnet werden wie mit LNT. Was hat sich beim Übergang zu LNT$\lambda\upsilon$ überhaupt geändert? Wir sind von der üblichen Darstellung von λ-Termen zur de Bruijn-Notation übergegangen, und wir haben die impliziten β- und η-Regeln des λ-Kalküls durch die expliziten **Beta**- und **Eta**-Regeln des $\lambda\upsilon$-Kalküls ersetzt. Durch das Umformen der Regeln aus $\mathcal{R}$ fallen die $\lambda\overline{x_k}, \overline{y_l}$-Lifter weg. Wir müssen also insbesondere für die **Case Eval**-Regel überprüfen, ob sich dadurch etwas an den Ableitungen ändert.

Wir zeigen in diesem Abschnitt die Gültigkeit des folgenden kommutativen Diagrammes:

$$\begin{tikzcd} G \arrow[r, "\text{LNT}\ ;\ \beta\eta"] \arrow[d, "DB"'] & H \arrow[d, "DB"] \\ G' \arrow[r, "\text{LNT}\lambda\upsilon\ ;\ \lambda\upsilon\eta F"'] & H' \end{tikzcd}$$

[3] Das bedeutet praktisch die Anwendung der η'-Regel anstelle der **Eta**-Regel. Wir verwenden nicht direkt die η'-Regel, da diese keine elementare Ersetzungsregel ist.

Dazu ist zu zeigen, daß für jede Anfrage G und jede mögliche Anwendung einer Regel aus LNT auf G mit anschließender $\beta\eta$-Reduktion ($\rightarrow H$) gilt, daß auf die G entsprechende Anfrage $G' = DB(G)$ die zugehörige Regel aus LNT$\lambda\upsilon$ anwendbar ist, und nach Reduktion mit $\boldsymbol{\lambda\upsilon\eta F}$, bis kein υF-Redex mehr vorhanden ist (d.h., daß kein Substitutionsteil $[s]$ mehr vorkommt), das Ergebnis H' gerade die Übersetzung $DB(H)$ des Ergebnisses H ist.

Satz 5.5.1 (Äquivalenz von LNT und LNT$\boldsymbol{\lambda\upsilon}$)
Es sei G eine Anfrage für das System LNT, $G' = DB(G)$ sei die de Bruijn-Übersetzung von G. Ferner sei r eine der Regeln des Systems LNT und r' die entsprechende Regel aus LNT$\lambda\upsilon$. Dann gelten die folgenden Aussagen:

1. *Falls r auf G anwendbar ist, dann sei $r(G)$ das Ergebnis dieser Anwendung inklusive Ausführung der Substitution(en), aber ohne $\beta\eta$-Reduktion. Dann ist auch r' auf G' anwendbar, und für das Ergebnis $r'(G')$ dieser Anwendung inklusive Ausführung der Substitution(en), aber ohne $\lambda\upsilon\eta F$-Reduktion, gilt: $r'(G') \xrightarrow[\lambda\upsilon\eta F]{*} H'$ und $H' = DB(r(G){\downarrow_{\beta\eta}})$.*

2. *Die Umkehrung[4]: Falls r' auf G' anwendbar ist, dann sei $r'(G')$ das Ergebnis dieser Anwendung inklusive Ausführung der Substitution(en), aber ohne $\lambda\upsilon\eta F$-Reduktion. Dann ist auch r auf G anwendbar, und für das Ergebnis $r(G)$ dieser Anwendung inklusive Ausführung der Substitution(en), aber ohne $\beta\eta$-Reduktion, gilt (wieder): $r'(G') \xrightarrow[\lambda\upsilon\eta F]{*} H'$ und $H' = DB(r(G){\downarrow_{\beta\eta}})$.*

Beweis. Wir zeigen nur die erste Aussage. Der Beweis der zweiten Aussage verläuft analog.

- Die **Case Select**-Regel:
  Damit **Case Select** anwendbar ist, muß die Anfrage die Form

  $$G = \lambda\overline{x_k}.\text{case } \lambda\overline{y_l}.v(\overline{t_m}) \text{ of } \overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G_1$$

  haben. Außerdem muß für ein i gelten: $p_i = \lambda\overline{y_l}.v(\overline{X_m(\overline{x_k}, \overline{y_l})})$. Dann gilt für die übersetzte Anfrage $G' = DB(G)$:

  $$G' = \lambda^k\text{case } \lambda^l v(\overline{t'_m}) \text{ of } \overline{p'_n : \mathcal{X}'_n} \rightarrow^? Z, G'_1,$$

  mit $G'_1 = DB(G_1)$, wobei nach Konstruktion der Übersetzung für die

[4] Man beachte, daß wir keine Aussagen über beliebige Anfragen an LNT$\lambda\upsilon$ machen. Die betrachteten Anfragen G' sind stets DB-Übersetzungen $DB(G)$ der Anfragen G an LNT.

t_i', p_i' und $\mathcal{X}_i'$ gilt: [5]

$$\begin{aligned} DB(\lambda\overline{x_k}\lambda\overline{y_l}.t_i) &= \lambda^{k+l}t_i', \\ DB(\lambda\overline{x_k}.p_i) &= \lambda^k p_i', \\ DB(\lambda\overline{x_k}.\mathcal{X}_i) &= \lambda^k\mathcal{X}_i'. \end{aligned}$$

Es gilt also auch $p_i' = \lambda^l\upsilon(\overline{X_m(\underline{\mathtt{k+1}},\ldots,\underline{\mathtt{1}})})$, und damit ist die **Case Select**-Regel aus LNT$\lambda\upsilon$ anwendbar, d.h. die Anwendbarkeit von r' auf G' ist gezeigt.

Nach Anwendung der Regel erhalten wir in den beiden Kalkülen

$$\begin{aligned} &\lambda\overline{x_k}.\{\overline{X_m \mapsto \lambda\overline{x_k},\overline{y_l}.t_m}\}\mathcal{X}_i \to^? Z, G_1 \\ &\text{bzw. } \lambda^k\langle\overline{X_m/\lambda^{k+l}t_m'}\rangle\mathcal{X}_i' \to^? Z, G_1'. \end{aligned}$$

Wir nennen die beiden Substitutionen σ und σ'. Nachdem die $\overline{X_m}$ in $\mathcal{X}_i$ bzw. $\mathcal{X}_i'$ ersetzt wurden, gilt wegen $DB(\lambda\overline{x_k}\lambda\overline{y_l}.t_i) = \lambda^{k+l}t_i'$, daß

$$DB(\lambda\overline{x_k}.\sigma(\mathcal{X}_i)) = \lambda^k\sigma'(\mathcal{X}_i').$$

Das heißt: bis hier ist die neue Regel eine korrekte Übersetzung der alten.

Nun kommen wir zur $\beta\eta$-Reduktion bzw. **Beta/Eta**-Reduktion. Nach Satz 4.2.17 gilt

$$a \to_\beta b \iff a \to_{\mathbf{Beta}} b' \text{ und } b = \upsilon(b').$$

Daraus folgt, daß alle **Beta**-Reduktionen mit anschließender υ-Normalisierung denselben Effekt haben wie die ursprünglichen β-Reduktionen.
Nach Satz 4.2.17 gilt

$$a \to_\eta b \iff a \to_{\mathbf{Eta}} b' \text{ und } b = \upsilon^{**}(b'),$$

wobei sich die Normalisierung $\upsilon^{**}(b')$ auf den Redex $t[\perp/]$, der durch **Eta** erzeugt wird, beschränkt.

- Die **Case Eval**-Regel:
  Case Eval ist nur anwendbar, wenn die Anfrage die Form

  $$G = \lambda\overline{x_k}.\text{case } \lambda\overline{y_l}.f(\overline{t_m}) \textit{ of } \overline{p_n : \mathcal{X}_n} \to^? Z, G_1$$

  hat. Dann gilt für die übersetzte Anfrage $G' = DB(G)$:

  $$G' = \lambda^k\text{case } \lambda^l f(\overline{t_m'}) \textit{ of } \overline{p_n' : \mathcal{X}_n'} \to^? Z, G_1',$$

[5] Wir können hier nicht etwa $DB(t_i) = t_i'$ schreiben, da dann t_i und t_i' aus dem Kontext gerissen wären: ohne die zugehörigen λ-Binder lassen sich diese Terme nicht ineinander konvertieren.

mit $G'_1 = DB(G_1)$, wobei (wie oben) für die t'_i, p'_i und $\mathcal{X}'_i$ gilt:

$$\begin{aligned} DB(\lambda\overline{x_k}\lambda\overline{y_l}.t_i) &= \lambda^{k+l}t'_i, \\ DB(\lambda\overline{x_k}.p_i) &= \lambda^k p'_i, \\ DB(\lambda\overline{x_k}.\mathcal{X}_i) &= \lambda^k \mathcal{X}'_i. \end{aligned}$$

Die **Case Eval**-Regeln der beiden Systeme unterscheiden sich in der Art und Weise, wie der definierende Baum für die Funktion f angewendet wird. Im System LNT haben die Regeln aus $\mathcal{R}$ die Form $f(\overline{X_m}) \to \mathcal{X}$, und die $\overline{X_m}$ müssen zunächst mit $\overline{x_k}, \overline{y_l}$-Liftern gehoben werden (siehe Abschnitt 3.4).

Nach Anwendung der Regel erhalten wir im Kalkül LNT:

$$\lambda\overline{x_k}, \overline{y_l}.\sigma(\mathcal{X}) \to^? X, \quad \lambda\overline{x_k}.\text{case } \lambda\overline{y_l}.X(\overline{x_k}, \overline{y_l}) \text{ of } \overline{p_n : \mathcal{X}_n} \to^? Z, G_1$$

mit $\sigma = \{\overline{X_m \mapsto \lambda\overline{x_k}, \overline{y_l}.t_m}\}$, wobei $f(\overline{X_m(\overline{x_k}, \overline{y_l})}) \to \mathcal{X}$ eine $\overline{x_k}, \overline{y_l}$-gehobene Regel ist.

Wir betrachten den Prozeß des $\overline{x_k}, \overline{y_l}$-Hebens nun etwas genauer: $\mathcal{R}$ enthalte die (ungehobene) Regel $f(\overline{X_m}) \to r(\overline{X_m})$. Nun wird ein $\overline{x_k}, \overline{y_l}$-Lifter auf diese Regel angewendet, d.h. nach Definition 3.4.2, daß die $\overline{X_m}$ durch $\overline{X'_m(\overline{x_k}, \overline{y_l})}$ ersetzt werden. Die gehobene Regel hat also die Form $f(\overline{X_m(\overline{x_k}, \overline{y_l})}) \to r(\overline{X_m(\overline{x_k}, \overline{y_l})})$. (Wir verzichten dabei auf den Strich $'$.) Das Ergebnis der Regelanwendung ist also

$$\lambda\overline{x_k}, \overline{y_l}.\sigma\big(r(\overline{X_m(\overline{x_k}, \overline{y_l})})\big) \to^? X,$$

$$\lambda\overline{x_k}.\text{case } \lambda\overline{y_l}.X(\overline{x_k}, \overline{y_l}) \text{ of } \overline{p_n : \mathcal{X}_n} \to^? Z, G_1$$

mit $\sigma = \{\overline{X_m \mapsto \lambda\overline{x_k}, \overline{y_l}.t_m}\}$.
Es gilt $\lambda\overline{x_k}, \overline{y_l}.\sigma\big(r(\overline{X_m(\overline{x_k}, \overline{y_l})})\big) = \lambda\overline{x_k}, \overline{y_l}.r(\overline{(\lambda\overline{x_k}, \overline{y_l}.t_m)(\overline{x_k}, \overline{y_l})})$ $=_\beta \lambda\overline{x_k}, \overline{y_l}.r(\overline{t_m})$. Nach β-Reduktion erhalten wir also die Anfrage

$$\lambda\overline{x_k}, \overline{y_l}.r(\overline{t_m}) \to^? X, \lambda\overline{x_k}.\text{case } \lambda\overline{y_l}.X(\overline{x_k}, \overline{y_l}) \text{ of } \overline{p_n : \mathcal{X}_n} \to^? Z, G_1.$$

Nun verfolgen wir die Auswertung derselben Anfrage im System LNTλv: Wir verwenden die Übersetzung der $\overline{x_k}, \overline{y_l}$-gehobenen Regel, d.h. wir übersetzen $f_\to(f(\overline{X_m(\overline{x_k}, \overline{y_l})}), \mathcal{X})$ im Kontext der $\overline{x_k}$ und erhalten

$$DB_{[\lambda\overline{x_k}, \overline{y_l}]}(f_\to(f(\overline{X_m(\overline{x_k}, \overline{y_l})}), \mathcal{X})) = f_\to(f(\overline{X_m(\underline{\mathtt{k+l}}, \ldots, \underline{\mathtt{1}})}), \mathcal{X}'),$$

wobei $\mathcal{X}' = DB_{[\lambda\overline{x_k}, \overline{y_l}]}(\mathcal{X})$ ist. Die Regel, die wir anwenden müssen, hat also die Form

$$f(\overline{X_m(\underline{\mathtt{k+l}}, \ldots, \underline{\mathtt{1}})}) \to \mathcal{X}'.$$

Wir wenden nun die **Case Eval**-Regel an und erhalten die Anfrage

$$\lambda^{k+l}\langle\overline{X_m/\lambda^{k+l}t'_m}\rangle(\mathcal{X}') \rightarrow^? X,$$

$$\lambda^k\text{case }\lambda^l X(\underline{\texttt{k+1}},\ldots,\underline{\texttt{1}})\text{ of }\overline{p'_n : \mathcal{X}'_n} \rightarrow^? Z, G'_1.$$

Dabei ist die zweite Zeile gerade die DB-Übersetzung von

$$\lambda\overline{x_k}.\text{case }\lambda\overline{y_l}.X(\overline{x_k},\overline{y_l})\text{ of }\overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G_1,$$

so daß wir nur die erste Zeile betrachten müssen. Bei der Untersuchung des LNT-Schrittes hatten wir $\mathcal{X} = r(\overline{X_m(\overline{x_k},\overline{y_l})})$ gesetzt – mit der Bedeutung, daß die Ausdrücke $X_i(\overline{x_k},\overline{y_l})$ in r vorkommen können. Damit ist $\mathcal{X}' = DB_{[\lambda\overline{x_k},\overline{y_l}]}(\mathcal{X}) = DB_{[\lambda\overline{x_k},\overline{y_l}]}(r(\overline{X_m(\overline{x_k},\overline{y_l})}))$. Die erste Zeile der Anfrage lautet also

$$\lambda^{k+l}\langle\overline{X_m/\lambda^{k+l}t'_m}\rangle\big(DB_{[\lambda\overline{x_k},\overline{y_l}]}(r(\overline{X_m(\overline{x_k},\overline{y_l})}))\big) \rightarrow^? X,$$

und für deren linke Seite gilt:

$$\begin{aligned}
&\lambda^{k+l}\langle\overline{X_m/\lambda^{k+l}t'_m}\rangle\big(DB_{[\lambda\overline{x_k},\overline{y_l}]}(r(\overline{X_m(\overline{x_k},\overline{y_l})}))\big)\\
&= \langle\overline{X_m/\lambda^{k+l}t'_m}\rangle\big(\lambda^{k+l}DB_{[\lambda\overline{x_k},\overline{y_l}]}(r(\overline{X_m(\overline{x_k},\overline{y_l})}))\big)\\
&= \langle\overline{X_m/\lambda^{k+l}t'_m}\rangle\big(DB(\lambda\overline{x_k},\overline{y_l}.r(\overline{X_m(\overline{x_k},\overline{y_l})}))\big)\\
&= DB(\{\overline{X_m \mapsto \lambda\overline{x_k},\overline{y_l}.t_m}\}\big(\lambda\overline{x_k},\overline{y_l}.r(\overline{X_m(\overline{x_k},\overline{y_l})})\big)) \qquad \text{(Satz 5.2.5)}\\
&= DB(\lambda\overline{x_k},\overline{y_l}.r(\overline{\big(\lambda\overline{x_k},\overline{y_l}.t_m\big)(\overline{x_k},\overline{y_l})})).
\end{aligned}$$

Da das Argument von DB zu $\lambda\overline{x_k},\overline{y_l}.r(\overline{t_m})$ β-reduziert wird, gilt auch

$$DB(\lambda\overline{x_k},\overline{y_l}.r(\overline{\big(\lambda\overline{x_k},\overline{y_l}.t_m\big)(\overline{x_k},\overline{y_l})})) \xrightarrow[\beta]{*} DB(\lambda\overline{x_k},\overline{y_l}.r(\overline{t_m})),$$

und aus der Korrektheit von $\boldsymbol{\lambda\upsilon\eta F}$ (Satz 4.2.17) folgt

$$DB(\lambda\overline{x_k},\overline{y_l}.r(\overline{\big(\lambda\overline{x_k},\overline{y_l}.t_m\big)(\overline{x_k},\overline{y_l})})) \xrightarrow[\mathbf{Beta},\upsilon]{*} DB(\lambda\overline{x_k},\overline{y_l}.r(\overline{t_m})).$$

- Die **Projection**-Regel:
  Wir betrachten also Anfragen der Form

$$G = \lambda\overline{x_k}.\text{case }\lambda\overline{y_l}.X(\overline{t_m})\text{ of }\overline{p_n : \mathcal{X}_n} \rightarrow^? Z, G_1.$$

  Die übersetzte Anfrage $G' = DB(G)$ hat dann die Form

$$G' = \lambda^k\text{case }\lambda^l X(\overline{t'_m})\text{ of }\overline{p'_n : \mathcal{X}'_n} \rightarrow^? Z, G'_1,$$

  und es gelten wieder $G'_1 = DB(G_1)$ und

$$\begin{aligned}
DB(\lambda\overline{x_k}\lambda\overline{y_l}.t_i) &= \lambda^{k+l}t'_i,\\
DB(\lambda\overline{x_k}.p_i) &= \lambda^k p'_i,\\
DB(\lambda\overline{x_k}.\mathcal{X}_i) &= \lambda^k\mathcal{X}'_i
\end{aligned}$$

für die t_i', p_i' und $\mathcal{X}_i'$. Die **Projection**-Regel führt nun dazu, daß auf die gesamte Anfrage G bzw. G' die Substitution $\sigma = \{X \mapsto \lambda\overline{x_m}.x_i(\overline{H_o(\overline{x_m})})\}$ bzw. $\sigma' = \langle X/\lambda^m \underline{\mathtt{i}}(\overline{H_o(\underline{\mathtt{m}},\ldots,\underline{\mathtt{1}})})\rangle$ angewendet wird, wobei σ' die DB-Übersetzung $DB(\sigma)$ von σ (nach Definition 5.2.4) ist. Es gilt nach Satz 5.2.5:

$$DB(\sigma(G)) = \sigma'(DB(G)).$$

Wieder folgt mit Satz 4.2.17 die Behauptung.

- Die **Imitation**- und **Function Guess**-Regeln:
  Hier ist der Beweis identisch mit dem zur **Projection**-Regel.

- Die **Bind**-Regel:
  Hier können wir direkt Satz 5.2.5 anwenden: Für die Anfrage $G = e \rightarrow^? Z, G_1$ mit $G' = DB(G) = DB(e) \rightarrow^? Z, G_1'$ und $\sigma = \{Z \mapsto e\}$ gilt $\sigma' := DB(\sigma) = \langle Z/DB(e)\rangle$. Es folgt also $DB(\sigma(G)) = \sigma'(G')$, und die Behauptung gilt wegen Satz 4.2.17. ■

Damit ist insgesamt für die Anwendung *einer* Regel in den Systemen LNT und LNT$\lambda\upsilon$ die Äquivalenz gezeigt. Wir folgern eine entsprechende Aussage für eine Kette von Regelanwendungen.

Folgerung 5.5.2 *Es sei G eine Anfrage an das System LNT und $G' = DB(G)$. Für jede Anfrage H, die sich aus G mit den Regeln aus LNT und $\beta\eta$-Reduktion herleiten läßt, gibt es eine Anfrage H', die sich aus G' mit den Regeln aus LNT$\lambda\upsilon$ und $\lambda\upsilon\eta F$-Reduktion herleiten läßt, so daß $H' = DB(H)$.*

$$\begin{array}{ccccccccc}
G & \longrightarrow & G_1 & \longrightarrow & G_2 & \stackrel{*}{\longrightarrow} & G_{n-1} & \longrightarrow & G_n = H \\
{\scriptstyle DB}\big\downarrow & & {\scriptstyle DB}\big\downarrow & & {\scriptstyle DB}\big\downarrow & & {\scriptstyle DB}\big\downarrow & & \big\downarrow{\scriptstyle DB} \\
G' & \longrightarrow & G_1' & \longrightarrow & G_2' & \underset{*}{\longrightarrow} & G_{n-1}' & \longrightarrow & G_n' = H'
\end{array}$$

Beweis. Die Aussage läßt sich durch eine einfache Induktion über die Länge l der Ableitungsfolge zeigen:

- $l = 0$, d.h. $H = G$. Dann gilt natürlich auch $DB(H) = DB(G)$.

- $l > 0$. Sei r die LNT-Regel, die auf G angewendet wird. Nach Satz 5.5.1 ist dann die entsprechende LNT$\lambda\upsilon$-Regel r' auf G' anwendbar, und nach $\beta\eta$- bzw. $\lambda\upsilon\eta F$-Reduktion erhalten wir Anfragen G_1 bzw. G_1' mit $G_1' = DB(G_1)$. Für die Anfrage G_1 gilt die Behauptung nach Induktionsvoraussetzung. ■

Aus der Äquivalenz zu LNT folgt sofort, daß für LNT$\lambda\upsilon$ die gleichen Ergebnisse hinsichtlich der Korrektheit, Vollständigkeit und Optimalität gelten wie für LNT:

Folgerung 5.5.3 (Korrektheit von LNT$\boldsymbol{\lambda\upsilon}$)
Falls $\mathcal{I}(t) \xRightarrow[LNT\lambda\upsilon]{*} {}^{\theta}\{\}$ *für einen Term* t, *dann gilt* $\theta(t) \xrightarrow{*} \mathrm{true}$.

Beweis. Sofort aus Satz 3.4.5 und Folgerung 5.5.2. ■

Folgerung 5.5.4 (Vollständigkeit von LNT$\boldsymbol{\lambda\upsilon}$)
Falls $\theta(t) \xrightarrow{*} \mathrm{true}$ *und* θ *in* $\mathcal{R}$*-Normalform ist, dann gibt es eine Substitution* θ', *so daß* $\mathcal{I}(t) \xRightarrow[LNT\lambda\upsilon]{*} {}^{\theta'}\{\}$ *mit* $\theta' \leq_{FV(t)} \theta$.

Beweis. Sofort aus Satz 3.4.6 und Folgerung 5.5.2. ■

Folgerung 5.5.5 (Optimalität von LNT$\boldsymbol{\lambda\upsilon}$)
Sind $\mathcal{I}(t) \xRightarrow[LNT\lambda\upsilon]{*} {}^{\theta}\{\}$ *und* $\mathcal{I}(t) \xRightarrow[LNT\lambda\upsilon]{*} {}^{\theta'}\{\}$ *zwei verschiedene Ableitungen, dann sind* θ *und* θ' *nicht vergleichbar, d.h.* $\theta \not\leq_{FV(t)} \theta'$ *und* $\theta' \not\leq_{FV(t)} \theta$.

Beweis. Sofort aus Satz 3.4.7 und Folgerung 5.5.2. ■

5.6 Beispiel

Nachdem wir die Äquivalenz der beiden Kalküle nachgewiesen haben, wollen wir ein Beispiel betrachten. Durch die expliziten **Beta**- und **Eta**-Reduktionen wird jeder der LNT-Ableitungsschritte in eine Folge elementarer Schritte (je ein LNT$\lambda\upsilon$-Schritt, gefolgt von mehreren $\boldsymbol{\lambda\upsilon\eta F}$-Schritten) zerlegt. Das folgende Beispiel können wir nur ausschnittsweise betrachten, da eine vollständige Ableitung von der Anfrage bis zur Lösung mehrere Seiten füllen würde. Dennoch stellen wir zwei Teilfolgen, in denen explizite **Beta**- bzw. **Eta**-Reduktionen auftreten, vollständig dar.

Bei der Anwendung der Regeln müssen wir beachten, daß ein Ausdruck der Form $f(a,b)$ nur eine andere Schreibweise für $((fa)b)$ ist: so kann beispielsweise die **App**-Regel nicht in einem Schritt auf $f(a,b)$ angewandt werden, sondern wir erhalten die Ableitungen $f(a,b)[\![\sigma]\!] \to ((fa)[\![\sigma]\!]b[\![\sigma]\!]) \to ((f[\![\sigma]\!]a[\![\sigma]\!])b[\![\sigma]\!])$[6].

Prinzipiell könnten wir auch die Regeln **App** und **Function** zu einer neuen Regel

$$\textbf{AppFun} \qquad f(t_1,\ldots,t_n)[s] \to f(t_1[s],\ldots,t_n[s])$$

kombinieren; dies wäre aber nur eine Abkürzung für die Folge

[6] Abweichend von unserer ursprünglichen Notation verwenden wir für die Substitutionen nicht die eckigen Klammern [und] sondern die „Semantikklammern“ $[\![$ und $]\!]$, da so die Struktur der Ausdrücke besser erkennbar ist.

$$f(t_1,\ldots,t_n)[s] = (((f\ t_1)t_2)\ldots t_n)[s] \underset{\text{App}}{\longrightarrow} ((((f\ t_1)t_2)\ldots t_{n-1})[s]\ t_n[s])$$
$$\overset{*}{\underset{App}{\longrightarrow}} (((f[s]\ t_1[s])t_2[s])\ldots t_n[s]) \underset{\text{Function}}{\longrightarrow} (((f\ t_1[s])t_2[s])\ldots t_n[s])$$
$$= f(t_1[s],\ldots,t_n[s]).$$

Beispiel 5.6.1

$\mathit{case}\ f(\lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1}))\ \mathit{of}\ \ \mathrm{true}:\mathrm{true} \rightarrow^{?} X_1$

$= DB\big(\mathit{case}\ f(\lambda x.\mathit{diff}(F(x,y),x))\ \mathit{of}\ \ \mathrm{true}:\mathrm{true} \rightarrow^{?} X_1\big)$[7]

$\Downarrow$ **LNT$\lambda\upsilon$: Case Eval** mit Regel für f, neues X_2

$(\lambda \mathit{case}\ \underline{1}\ \mathit{of}\ \lambda\cos\underline{1}:\mathrm{true})(\lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})) \rightarrow^{?} X_2,$

$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true}:\mathrm{true} \rightarrow^{?} X_1$

$\Downarrow$ **Beta**

$(\mathit{case}\ \underline{1}\ \mathit{of}\ \lambda\cos\underline{1}:\mathrm{true})[\![\lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})/]\!] \rightarrow^{?} X_2,$

$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true}:\mathrm{true} \rightarrow^{?} X_1$

$\Downarrow$ **App Case**

$\mathit{case}\ \underline{1}[\![\lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})/]\!]\ \mathit{of}$

$(\lambda\cos\underline{1})[\![\lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})/]\!]:\mathrm{true}[\![\lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})/]\!] \rightarrow^{?} X_2,$

$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true}:\mathrm{true} \rightarrow^{?} X_1$

$\Downarrow$ **FVar**

$\mathit{case}\ \lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})\ \mathit{of}$

$(\lambda\cos\underline{1})[\![\lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})/]\!]:\mathrm{true}[\![\lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})/]\!] \rightarrow^{?} X_2,$

$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true}:\mathrm{true} \rightarrow^{?} X_1$

$\Downarrow$ **Lambda**

$\mathit{case}\ \lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})\ \mathit{of}$

$\lambda((\cos\underline{1})[\![\Uparrow(\lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})/)]\!]):\mathrm{true}[\![\lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})/]\!] \rightarrow^{?} X_2,$

$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true}:\mathrm{true} \rightarrow^{?} X_1$

$\Downarrow^2$ **App, Const**

$\mathit{case}\ \lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})\ \mathit{of}$

$\lambda(\cos\underline{1}[\![\Uparrow(\lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})/)]\!]):\mathrm{true}[\![\lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})/]\!] \rightarrow^{?} X_2,$

$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true}:\mathrm{true} \rightarrow^{?} X_1$

$\Downarrow$ **FVarLift**

$\mathit{case}\ \lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})\ \mathit{of}$

$\lambda\cos\underline{1}:\mathrm{true}[\![\lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})/]\!] \rightarrow^{?} X_2,$

$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true}:\mathrm{true} \rightarrow^{?} X_1$

$\Downarrow$ **Const**

$\mathit{case}\ \lambda \mathit{diff}(\lambda \sin F(\underline{2},\underline{1}),\underline{1})\ \mathit{of}\ \ \lambda\cos\underline{1}:\mathrm{true} \rightarrow^{?} X_2,$

$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true}:\mathrm{true} \rightarrow^{?} X_1$

$= DB\big(\mathit{case}\ \lambda x.\mathit{diff}(\lambda y.\sin(F(x,y)),x)\ \mathit{of}\ \lambda x.\cos x:\mathrm{true} \rightarrow^{?} X_2,$

$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true}:\mathrm{true} \rightarrow^{?} X_1\big)$

$\Downarrow$ **LNT$\lambda\upsilon$: Case Eval** mit Regel für diff, neues X_3,

[7] Wir geben vor jedem LNT$\lambda\upsilon$-Schritt die Darstellung im normalen λ-Kalkül an.

$$\sigma = \langle H/\lambda(\lambda \sin F(\underline{2},\underline{1})), X/\lambda\underline{1}\rangle$$ [8]

$\lambda(case\ (\lambda(\lambda \sin F(\underline{2},\underline{1})))(\underline{1})\ of$

$\quad\lambda\underline{1} : 1,$

$\quad\lambda \sin G(\underline{2},\underline{1}) : \cos G(\underline{1},(\lambda\underline{1})(\underline{1})) * diff(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})\underline{1}),$

$\quad\lambda \ln G(\underline{2},\underline{1}) : diff(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})(\underline{1}))/G(\underline{1},(\lambda\underline{1})(\underline{1}))\ ) \rightarrow^{?} X_3,$

$case\ \lambda X_3(\underline{1})\ of\ \lambda \cos(\underline{1}) : \mathrm{true} \rightarrow^{?} X_2,$

$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \rightarrow^{?} X_1$

$\quad\Downarrow$ **Beta**

$\lambda(case\ ((\lambda \sin F(\underline{2},\underline{1}))[\![\underline{1}/]\!])\ of$

$\quad\lambda\underline{1} : 1,$

$\quad\lambda \sin G(\underline{2},\underline{1}) : \cos G(\underline{1},(\lambda\underline{1})(\underline{1})) * diff(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})\underline{1}),$

$\quad\lambda \ln G(\underline{2},\underline{1}) : diff(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})(\underline{1}))/G(\underline{1},(\lambda\underline{1})(\underline{1}))\ ) \rightarrow^{?} X_3,$

$case\ \lambda X_3(\underline{1})\ of\ \lambda \cos(\underline{1}) : \mathrm{true} \rightarrow^{?} X_2,$

$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \rightarrow^{?} X_1$

$\quad\Downarrow$ **Lambda**

$\lambda(case\ \lambda((\sin F(\underline{2},\underline{1}))[\![\Uparrow(\underline{1}/)]\!])\ of$

$\quad\lambda\underline{1} : 1,$

$\quad\lambda \sin G(\underline{2},\underline{1}) : \cos G(\underline{1},(\lambda\underline{1})(\underline{1})) * diff(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})\underline{1}),$

$\quad\lambda \ln G(\underline{2},\underline{1}) : diff(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})(\underline{1}))/G(\underline{1},(\lambda\underline{1})(\underline{1}))\ ) \rightarrow^{?} X_3,$

$case\ \lambda X_3(\underline{1})\ of\ \lambda \cos(\underline{1}) : \mathrm{true} \rightarrow^{?} X_2,$

$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \rightarrow^{?} X_1$

$\quad\Downarrow^{*}$ **App, Function, FreeVar**

$\lambda(case\ \lambda(\sin F(\underline{2}[\![\Uparrow(\underline{1}/)]\!],\underline{1}[\![\Uparrow(\underline{1}/)]\!]))\ of$

$\quad\lambda\underline{1} : 1,$

$\quad\lambda \sin G(\underline{2},\underline{1}) : \cos G(\underline{1},(\lambda\underline{1})(\underline{1})) * diff(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})\underline{1}),$

$\quad\lambda \ln G(\underline{2},\underline{1}) : diff(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})(\underline{1}))/G(\underline{1},(\lambda\underline{1})(\underline{1}))\ ) \rightarrow^{?} X_3,$

$case\ \lambda X_3(\underline{1})\ of\ \lambda \cos(\underline{1}) : \mathrm{true} \rightarrow^{?} X_2,$

$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \rightarrow^{?} X_1$

$\quad\Downarrow^{*}$ **RVarLift, FVarLift**

$\lambda(case\ \lambda(\sin F(\underline{1}[\![\underline{1}/]\!][\![\uparrow]\!],\underline{1}))\ of$

$\quad\lambda\underline{1} : 1,$

$\quad\lambda \sin G(\underline{2},\underline{1}) : \cos G(\underline{1},(\lambda\underline{1})(\underline{1})) * diff(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})\underline{1}),$

$\quad\lambda \ln G(\underline{2},\underline{1}) : diff(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})(\underline{1}))/G(\underline{1},(\lambda\underline{1})(\underline{1}))\ ) \rightarrow^{?} X_3,$

$case\ \lambda X_3(\underline{1})\ of\ \lambda \cos(\underline{1}) : \mathrm{true} \rightarrow^{?} X_2,$

$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \rightarrow^{?} X_1$

$\quad\Downarrow$ **FVar, VarShift**

$\lambda(case\ \lambda \sin F(\underline{2},\underline{1})\ of$

---

[8] Die Übersetzung der x-gehobenen Regel für *diff* ist

$$\begin{aligned} diff(H(\underline{1}),X(\underline{1})) \rightarrow case\ H(\underline{1})\ of\ & \lambda\underline{1} : 1, \\ & \lambda \sin(G(\underline{2},\underline{1})) : \cos G(\underline{1},X(\underline{1})) * diff(\lambda G(\underline{2},\underline{1}),X(\underline{1})), \\ & \lambda \ln(G(\underline{2},\underline{1}))\ \ : diff(\lambda G(\underline{2},\underline{1}),X(\underline{1}))/G(\underline{1},X(\underline{1})). \end{aligned}$$

$$
\begin{array}{l}
\qquad \lambda\underline{1} : 1, \\
\qquad \lambda \sin G(\underline{2},\underline{1}) : \cos G(\underline{1},(\lambda\underline{1})(\underline{1})) * \mathit{diff}(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})\underline{1}), \\
\qquad \lambda \ln G(\underline{2},\underline{1}) : \mathit{diff}(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})(\underline{1}))/G(\underline{1},(\lambda\underline{1})(\underline{1}))\ ) \to^? X_3, \\
\mathit{case}\ \lambda X_3(\underline{1})\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2, \\
\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1 \\
\qquad\quad = DB\big(\lambda x.\mathit{case}\ \lambda y.\sin F(x,y)\ \mathit{of} \\
\qquad\qquad\quad \lambda y.y : 1, \\
\qquad\qquad\quad \lambda y.\sin G(x,y) : \cos G(x,(\lambda y.y)(x)) * \mathit{diff}(\lambda y.G(x,y),(\lambda y.y)x), \\
\qquad\qquad\quad \lambda y.\ln G(x,y) : \mathit{diff}(\lambda y.G(x,y),(\lambda y.y)(x))/G(x,(\lambda y.y)(x)) \to^? X_3, \\
\qquad\qquad \mathit{case}\ \lambda x.X_3(x)\ \mathit{of}\ \lambda x.\cos x : \mathrm{true} \to^? X_2, \\
\qquad\qquad \mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1\big) \\
\qquad \Downarrow \textbf{LNT}\boldsymbol{\lambda\upsilon}\textbf{: Case Select}, \sigma = \langle G/\lambda\lambda F(\underline{2},\underline{1})\rangle \\
\lambda(\cos(\lambda\lambda F(\underline{2},\underline{1}))(\underline{1},(\lambda\underline{1})(\underline{1})) * \mathit{diff}(\lambda(\lambda\lambda F(\underline{2},\underline{1}))(\underline{2},\underline{1}),(\lambda\underline{1})\underline{1})) \to^? X_3, \\
\mathit{case}\ \lambda X_3(\underline{1})\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2, \\
\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1 \\
\qquad \Downarrow^* \textbf{Beta},\ \upsilon F \\
\lambda(\cos(\lambda\lambda F(\underline{2},\underline{1}))(\underline{1},\underline{1}) * \mathit{diff}(\lambda(\lambda\lambda F(\underline{2},\underline{1}))(\underline{2},\underline{1}),\underline{1})) \to^? X_3, \\
\mathit{case}\ \lambda X_3(\underline{1})\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2, \\
\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1 \\
\qquad \Downarrow^* \textbf{Beta},\ \upsilon F \\
\lambda(\cos F(\underline{1},\underline{1}) * \mathit{diff}(\lambda F(\underline{2},\underline{1}),\underline{1})) \to^? X_3, \\
\mathit{case}\ \lambda X_3(\underline{1})\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2, \\
\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1 \\
\qquad\quad = DB\big(\lambda x.(\cos F(x,x) * \mathit{diff}(\lambda y.F(x,y),x)) \to^? X_3, \\
\qquad\qquad \mathit{case}\ \lambda x.X_3(x)\ \mathit{of}\ \lambda x.\cos x : \mathrm{true} \to^? X_2, \\
\qquad\qquad \mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1\big) \\
\qquad \Downarrow \textbf{LNT}\boldsymbol{\lambda\upsilon}\textbf{: Bind} \\
\mathit{case}\ \lambda(\lambda(\cos F(\underline{1},\underline{1}) * \mathit{diff}(\lambda F(\underline{2},\underline{1}),\underline{1})))(\underline{1})\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2, \\
\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1 \\
\qquad \Downarrow \textbf{Eta} \\
\mathit{case}\ (\lambda(\cos F(\underline{1},\underline{1}) * \mathit{diff}(\lambda F(\underline{2},\underline{1}),\underline{1})))[\![\perp/]\!]\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2, \\
\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1 \\
\qquad \Downarrow \textbf{Lambda} \\
\mathit{case}\ \lambda\big((\cos F(\underline{1},\underline{1}) * \mathit{diff}(\lambda F(\underline{2},\underline{1}),\underline{1}))[\![\Uparrow(\perp/)]\!]\big)\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2, \\
\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1 \\
\qquad \Downarrow^* \textbf{App, Function, FreeVar} \\
\mathit{case}\ \lambda\big((\cos F(\underline{1}[\![\Uparrow(\perp/)]\!],\underline{1}[\![\Uparrow(\perp/)]\!]) * \mathit{diff}((\lambda F(\underline{2},\underline{1}))[\![\Uparrow(\perp/)]\!],\underline{1}[\![\Uparrow(\perp/)]\!]))\big) \\
\qquad\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2, \\
\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1 \\
\qquad \Downarrow^* \textbf{FVarLift, Lambda} \\
\mathit{case}\ \lambda\big((\cos F(\underline{1},\underline{1}) * \mathit{diff}(\lambda(F(\underline{2},\underline{1})[\![\Uparrow(\Uparrow(\perp/))]\!]),\underline{1}))\big)\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2,
\end{array}
$$

$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \rightarrow^? X_1$

$\Downarrow^*$ **FVarLift, Lambda**

$case\ \lambda\big((\cos F(\underline{1},\underline{1}) * diff(\lambda(F(\underline{1}[\![\Uparrow(\bot/)]\!][\![\uparrow]\!],\underline{1})),\underline{1}))\big)\ of\ \lambda\cos(\underline{1}) : \mathrm{true} \rightarrow^? X_2,$

$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \rightarrow^? X_1$

$\Downarrow^*$ **FVarLift, VarShift**

$case\ \lambda\big((\cos F(\underline{1},\underline{1}) * diff(\lambda(F(\underline{2},\underline{1})),\underline{1}))\big)\ of\ \lambda\cos(\underline{1}) : \mathrm{true} \rightarrow^? X_2,$

$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \rightarrow^? X_1$

$$= DB\big(case\ \lambda x.\big((\cos F(x,x) * diff(\lambda y.(F(x,y)),x))\big)\ of\ \lambda x.\cos x : \mathrm{true} \rightarrow^? X_2,\ case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \rightarrow^? X_1\big)$$

$\Downarrow$ **LNTλv: Case Eval** mit Regel für $*$, neues X_3,

$\sigma = \langle X/\lambda\cos F(\underline{1},\underline{1}), Y/\lambda diff(\lambda(F(\underline{2},\underline{1})),\underline{1})\rangle$ [9]

$\lambda case\ (\lambda diff(\lambda(F(\underline{2},\underline{1})),\underline{1}))(\underline{1})\ of$

$1 : (\lambda\cos F(\underline{1},\underline{1}))(\underline{1}),$

$s(Y'(\underline{1})) : (\lambda\cos F(\underline{1},\underline{1}))(\underline{1}) + (\lambda\cos F(\underline{1},\underline{1}))(\underline{1}) * Y'(\underline{1}) \rightarrow^? X_3,$

$case\ \lambda X_3(\underline{1})\ of\ \lambda\cos(\underline{1}) : \mathrm{true} \rightarrow^? X_2,$

$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \rightarrow^? X_1$

$\Downarrow$ **Beta**

$\lambda case\ (diff(\lambda(F(\underline{2},\underline{1})),\underline{1})[\![\underline{1}/]\!])\ of$

$1 : (\lambda\cos F(\underline{1},\underline{1}))(\underline{1}),$

$s(Y'(\underline{1})) : (\lambda\cos F(\underline{1},\underline{1}))(\underline{1}) + (\lambda\cos F(\underline{1},\underline{1}))(\underline{1}) * Y'(\underline{1}) \rightarrow^? X_3,$

$case\ \lambda X_3(\underline{1})\ of\ \lambda\cos(\underline{1}) : \mathrm{true} \rightarrow^? X_2,$

$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \rightarrow^? X_1$

$\Downarrow^*$ **App, Function**

$\lambda case\ diff((\lambda(F(\underline{2},\underline{1})))[\![\underline{1}/]\!],\underline{1}[\![\underline{1}/]\!])\ of$

$1 : (\lambda\cos F(\underline{1},\underline{1}))(\underline{1}),$

$s(Y'(\underline{1})) : (\lambda\cos F(\underline{1},\underline{1}))(\underline{1}) + (\lambda\cos F(\underline{1},\underline{1}))(\underline{1}) * Y'(\underline{1}) \rightarrow^? X_3,$

$case\ \lambda X_3(\underline{1})\ of\ \lambda\cos(\underline{1}) : \mathrm{true} \rightarrow^? X_2,$

$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \rightarrow^? X_1$

$\Downarrow^*$ **Lambda, App, Const; FVar**

$\lambda case\ diff(\lambda(F(\underline{2}[\![\Uparrow(\underline{1}/)]\!],\underline{1}[\![\Uparrow(\underline{1}/)]\!])),\underline{1})\ of$

$1 : (\lambda\cos F(\underline{1},\underline{1}))(\underline{1}),$

$s(Y'(\underline{1})) : (\lambda\cos F(\underline{1},\underline{1}))(\underline{1}) + (\lambda\cos F(\underline{1},\underline{1}))(\underline{1}) * Y'(\underline{1}) \rightarrow^? X_3,$

$case\ \lambda X_3(\underline{1})\ of\ \lambda\cos(\underline{1}) : \mathrm{true} \rightarrow^? X_2,$

$case\ X_2\ of\ \ \mathrm{true} : \mathrm{true} \rightarrow^? X_1$

$\Downarrow^*$ **RVarLift, FVar, VarShift; FVarLift**

$\lambda case\ diff(\lambda(F(\underline{2},\underline{1})),\underline{1})\ of$

$1 : (\lambda\cos F(\underline{1},\underline{1}))(\underline{1}),$

---

[9] Die Übersetzung der y-gehobenen Regel für $*$ ist:

$X(\underline{1}) * Y(\underline{1}) \rightarrow case\ Y(\underline{1})\ of\ 1 : X(\underline{1}), s(Y'(\underline{1})) : X(\underline{1}) + X(\underline{1}) * Y'(\underline{1})$

$\quad s(Y'(\underline{1})) : (\lambda \cos F(\underline{1},\underline{1}))(\underline{1}) + (\lambda \cos F(\underline{1},\underline{1}))(\underline{1}) * Y'(\underline{1}) \to^? X_3,$
$\mathit{case}\ \lambda X_3(\underline{1})\ \mathit{of}\ \lambda \cos(\underline{1}) : \mathrm{true} \to^? X_2,$
$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$
$\quad \Downarrow^*\ \ \ldots$
$\lambda \mathit{case}\ \mathit{diff}(\lambda(F(\underline{2},\underline{1})),\underline{1})\ \mathit{of}$
$\quad 1 : \cos F(\underline{1},\underline{1}),$
$\quad s(Y'(\underline{1})) : \cos F(\underline{1},\underline{1}) + \cos F(\underline{1},\underline{1}) * Y'(\underline{1}) \to^? X_3,$
$\mathit{case}\ \lambda X_3(\underline{1})\ \mathit{of}\ \lambda \cos(\underline{1}) : \mathrm{true} \to^? X_2,$
$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$
$\quad = DB\big(\lambda x.\mathit{case}\ \mathit{diff}(\lambda y.F(x,y),x)\ \mathit{of}$
$\qquad 1 : \cos F(x,x),$
$\qquad s(Y'(x)) : \cos F(x,x) + \cos F(x,x) * Y'(x) \to^? X_3,$
$\qquad \mathit{case}\ \lambda x.X_3(x)\ \mathit{of}\ \lambda x.\cos x : \mathrm{true} \to^? X_2,$
$\qquad \mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1\big)$
$\quad \Downarrow$ **LNTλv: Case Eval** mit Regel für *diff*, neues X_4
$\qquad \sigma = \langle H/\lambda(\lambda F(\underline{2},\underline{1})), X/\lambda\underline{1}\rangle$ [10]
$\lambda(\mathit{case}\ (\lambda(\lambda F(\underline{2},\underline{1})))(\underline{1})\ \mathit{of}$
$\quad \lambda\underline{1} : 1,$
$\quad \lambda \sin G(\underline{2},\underline{1}) : \cos G(\underline{1},(\lambda\underline{1})(\underline{1})) * \mathit{diff}(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})\underline{1}),$
$\quad \lambda \ln G(\underline{2},\underline{1}) : \mathit{diff}(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})(\underline{1}))/G(\underline{1},(\lambda\underline{1})(\underline{1}))\ ) \to^? X_4,$
$\lambda \mathit{case}\ X_4(\underline{1})\ \mathit{of}$
$\quad 1 : \cos F(\underline{1},\underline{1}),$
$\quad s(Y'(\underline{1})) : \cos F(\underline{1},\underline{1}) + \cos F(\underline{1},\underline{1}) * Y'(\underline{1}) \to^? X_3,$
$\mathit{case}\ \lambda X_3(\underline{1})\ \mathit{of}\ \lambda \cos(\underline{1}) : \mathrm{true} \to^? X_2,$
$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$
$\quad \Downarrow^*$ wie schon für $\lambda(\mathit{case}\ (\lambda(\lambda \sin F(\underline{2},\underline{1})))(\underline{1})\ \mathit{of}\ \ \ldots)$
$\lambda(\mathit{case}\ \lambda \sin F(\underline{2},\underline{1})\ \mathit{of}$
$\quad \lambda\underline{1} : 1,$
$\quad \lambda \sin G(\underline{2},\underline{1}) : \cos G(\underline{1},(\lambda\underline{1})(\underline{1})) * \mathit{diff}(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})\underline{1}),$
$\quad \lambda \ln G(\underline{2},\underline{1}) : \mathit{diff}(\lambda G(\underline{2},\underline{1}),(\lambda\underline{1})(\underline{1}))/G(\underline{1},(\lambda\underline{1})(\underline{1}))\ ) \to^? X_4,$
$\lambda \mathit{case}\ X_4(\underline{1})\ \mathit{of}$
$\quad 1 : \cos F(\underline{1},\underline{1}),$
$\quad s(Y'(\underline{1})) : \cos F(\underline{1},\underline{1}) + \cos F(\underline{1},\underline{1}) * Y'(\underline{1}) \to^? X_3,$
$\mathit{case}\ \lambda X_3(\underline{1})\ \mathit{of}\ \lambda \cos(\underline{1}) : \mathrm{true} \to^? X_2,$
$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$
$\quad = DB\big(\lambda x.(\mathit{case}\ \lambda y.\sin F(x,y)\ \mathit{of}$
$\qquad \lambda y.y : 1,$

---

[10] Die Übersetzung der x-gehobenen Regel für *diff* ist

$$\begin{aligned}\mathit{diff}(H(\underline{1}),X(\underline{1})) \to \mathit{case}\ H(\underline{1})\ \mathit{of}\ & \lambda\underline{1} : 1,\\ & \lambda \sin(G(\underline{2},\underline{1})) : \cos G(\underline{1},X(\underline{1})) * \mathit{diff}(\lambda G(\underline{2},\underline{1}),X(\underline{1})),\\ & \lambda \ln(G(\underline{2},\underline{1}))\ \ : \mathit{diff}(\lambda G(\underline{2},\underline{1}),X(\underline{1}))/G(\underline{1},X(\underline{1})).\end{aligned}$$

$$\lambda y. \sin G(x,y) : \cos G(x, (\lambda y.y)x) * \mathit{diff}(\lambda y.G(x,y), (\lambda y.y)x),$$
$$\lambda y. \ln G(x,y) : \mathit{diff}(\lambda y.G(x,y), (\lambda y.y)x)/G(x, (\lambda y.y)x)\ ) \to^? X_4,$$
$$\lambda x.\mathit{case}\ X_4(x)\ \mathit{of}$$
$$1 : \cos F(x,x),$$
$$s(Y'(x)) : \cos F(x,x) + \cos F(x,x) * Y'(x) \to^? X_3,$$
$$\mathit{case}\ \lambda x.X_3(x)\ \mathit{of}\ \lambda x.\cos(x) : \mathrm{true} \to^? X_2,$$
$$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1)$$

$\Downarrow_\sigma$ **LNTλv: Projection** mit $\sigma = \langle F/\lambda\lambda\underline{1}\rangle$

$$\lambda(\mathit{case}\ \lambda(\lambda\lambda\underline{1})(\underline{2},\underline{1})\ \mathit{of}$$
$$\lambda\underline{1} : 1,$$
$$\lambda \sin G(\underline{2},\underline{1}) : \cos G(\underline{1}, (\lambda\underline{1})(\underline{1})) * \mathit{diff}(\lambda G(\underline{2},\underline{1}), (\lambda\underline{1})\underline{1}),$$
$$\lambda \ln G(\underline{2},\underline{1}) : \mathit{diff}(\lambda G(\underline{2},\underline{1}), (\lambda\underline{1})(\underline{1}))/G(\underline{1}, (\lambda\underline{1})(\underline{1}))\ ) \to^? X_4,$$
$$\lambda \mathit{case}\ X_4(\underline{1})\ \mathit{of}$$
$$1 : \cos(\lambda\lambda\underline{1})(\underline{1},\underline{1}),$$
$$s(Y'(\underline{1})) : \cos(\lambda\lambda\underline{1})(\underline{1},\underline{1}) + \cos(\lambda\lambda\underline{1})(\underline{1},\underline{1}) * Y'(\underline{1}) \to^? X_3,$$
$$\mathit{case}\ \lambda X_3(\underline{1})\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2,$$
$$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$$

$\Downarrow^*\ \ \ldots$

$$\lambda(\mathit{case}\ \lambda\underline{1}\ \mathit{of}$$
$$\lambda\underline{1} : 1,$$
$$\lambda \sin G(\underline{2},\underline{1}) : \cos G(\underline{1}, (\lambda\underline{1})(\underline{1})) * \mathit{diff}(\lambda G(\underline{2},\underline{1}), (\lambda\underline{1})\underline{1}),$$
$$\lambda \ln G(\underline{2},\underline{1}) : \mathit{diff}(\lambda G(\underline{2},\underline{1}), (\lambda\underline{1})(\underline{1}))/G(\underline{1}, (\lambda\underline{1})(\underline{1}))\ ) \to^? X_4,$$
$$\lambda \mathit{case}\ X_4(\underline{1})\ \mathit{of}$$
$$1 : \cos \underline{1},$$
$$s(Y'(\underline{1})) : \cos \underline{1} + \cos(\underline{1}) * Y'(\underline{1}) \to^? X_3,$$
$$\mathit{case}\ \lambda X_3(\underline{1})\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2,$$
$$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$$
$$= DB(\lambda x.(\mathit{case}\ \lambda y.y\ \mathit{of}$$
$$\lambda y.y : 1,$$
$$\lambda y. \sin G(x,y) : \cos G(x, (\lambda y.y)x) * \mathit{diff}(\lambda y.G(x,y), (\lambda y.y)x),$$
$$\lambda y. \ln G(x,y) : \mathit{diff}(\lambda y.G(x,y), (\lambda y.y)x)/G(x, (\lambda y.y)x)\ ) \to^? X_4,$$
$$\lambda x.\mathit{case}\ X_4(x)\ \mathit{of}$$
$$1 : \cos x,$$
$$s(Y'(x)) : \cos x + \cos x * Y'(x) \to^? X_3,$$
$$\mathit{case}\ \lambda x.X_3(x)\ \mathit{of}\ \lambda x.\cos(x) : \mathrm{true} \to^? X_2,$$
$$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1)$$

$\Downarrow$ **LNTλv: Case Select**, $\sigma = \varnothing$

$$\lambda 1 \to^? X_4,$$
$$\lambda \mathit{case}\ X_4(\underline{1})\ \mathit{of}$$
$$1 : \cos \underline{1},$$
$$s(Y'(\underline{1})) : \cos \underline{1} + \cos(\underline{1}) * Y'(\underline{1}) \to^? X_3,$$

$\mathit{case}\ \lambda X_3(\underline{1})\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2,$

$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$

$\quad\Downarrow \mathbf{LNT}\lambda\upsilon\mathbf{:\ Bind},\ \sigma = \langle X_4/\lambda 1\rangle$

$\lambda\mathit{case}\ (\lambda 1)(\underline{1})\ \mathit{of}$

$\quad 1 : \cos\underline{1},$

$\quad s(Y'(\underline{1})) : \cos\underline{1} + \cos(\underline{1}) * Y'(\underline{1}) \to^? X_3,$

$\mathit{case}\ \lambda X_3(\underline{1})\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2,$

$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$

$\quad\Downarrow^* \ \ldots$

$\lambda\mathit{case}\ 1\ \mathit{of}$

$\quad 1 : \cos\underline{1},$

$\quad s(Y'(\underline{1})) : \cos\underline{1} + \cos(\underline{1}) * Y'(\underline{1}) \to^? X_3,$

$\mathit{case}\ \lambda X_3(\underline{1})\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2,$

$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$

$\quad\quad = DB\big(\lambda x.\mathit{case}\ 1\ \mathit{of}$

$\quad\quad\quad 1 : \cos x,$

$\quad\quad\quad s(Y'(x)) : \cos x + \cos x * Y'(x) \to^? X_3,$

$\quad\quad\ \ \mathit{case}\ \lambda x.X_3(x)\ \mathit{of}\ \lambda x.\cos(x) : \mathrm{true} \to^? X_2,$

$\quad\quad\ \ \mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1\big)$

$\quad\Downarrow \mathbf{LNT}\lambda\upsilon\mathbf{:\ Case\ Select},\ \sigma = \varnothing$

$\lambda\cos\underline{1} \to^? X_3,$

$\mathit{case}\ \lambda X_3(\underline{1})\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2,$

$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$

$\quad\Downarrow \mathbf{LNT}\lambda\upsilon\mathbf{:\ Bind},\ \sigma = \langle X_3/\lambda\cos\underline{1}\rangle$

$\mathit{case}\ \lambda(\lambda\cos\underline{1})(\underline{1})\ \mathit{of}\ \lambda\cos(\underline{1}) : \mathrm{true} \to^? X_2,$

$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$

$\quad\Downarrow^* \ \ldots$

$\mathit{case}\ \lambda\cos\underline{1}\ \mathit{of}\ \lambda\cos\underline{1} : \mathrm{true} \to^? X_2,$

$\mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$

$\quad\quad = DB\big(\mathit{case}\ \lambda x.\cos x\ \mathit{of}\ \lambda x.\cos x : \mathrm{true} \to^? X_2,$

$\quad\quad\ \ \mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1\big)$

$\quad\Downarrow \mathbf{LNT}\lambda\upsilon\mathbf{:\ Case\ Select},\ \sigma = \varnothing$

$\mathrm{true} \to^? X_2, \mathit{case}\ X_2\ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$

$\quad\Downarrow \mathbf{LNT}\lambda\upsilon\mathbf{:\ Bind},\ \sigma = \langle X_2/\,\mathrm{true}\rangle$

$\mathit{case}\ \ \mathrm{true}\ \ \mathit{of}\ \ \mathrm{true} : \mathrm{true} \to^? X_1$

$\quad\Downarrow \mathbf{LNT}\lambda\upsilon\mathbf{:\ Case\ Select},\ \sigma = \varnothing$

$\mathrm{true} \to^? X_1$

$\quad\Downarrow \mathbf{LNT}\lambda\upsilon\mathbf{:\ Bind},\ \sigma = \langle X_1/\,\mathrm{true}\rangle$

$\varnothing.$

Damit ist die berechnete Lösung: $\langle F/\lambda\lambda\underline{1}\rangle$; das entspricht der Lösung $\{F \mapsto \lambda x, y.y\}$ für die ursprüngliche Anfrage an LNT.

Bemerkung 5.6.2 Wir sind in diesem Beispiel von der angegebenen Reduktionsstrategie abgewichen und haben immer nur die Ausdrücke $\lambda\upsilon\eta F$-reduziert, die für die weitere Berechnung erforderlich waren. Wir hätten stattdessen nach jedem LNT$\lambda\upsilon$-Schritt eine vollständige Reduktion auf $\lambda\upsilon\eta F$-Normalform angeben müssen. Dies hätte jedoch zu einer noch längeren Ableitungsfolge geführt – zum Nachvollziehen der Regeln reicht diese Darstellung jedoch aus. □

Kapitel 6

Zusammenfassung

Wir haben das System LNT [HP96], das Needed Narrowing mit definierenden Bäumen höherer Ordnung realisiert, in den λ-Kalkül in de Bruijn-Notation [dB72] transformiert. Dieser Kalkül läßt sich als „Faktorisierung" der α-Äquivalenz betrachten, da ein Λ_{DB}-Term $t' = DB(t)$ der Restklasse aller zu t α-äquivalenten Terme aus Λ entspricht.

Die implizite Anwendung von β- und η-Reduktionsschritten haben wir durch explizite **Beta**- und **Eta**-Reduktionen ersetzt, die mit Hilfe des $\lambda\upsilon$-Kalküls (mit der Erweiterung um eine explizite **Eta**-Regel) [LRD94, Bri95] über explizite Substitutionen in kleinen Schritten berechnet werden.

Das neue System LNT$\lambda\upsilon$ ist dadurch effizienter zu implementieren, da es nicht modulo $\alpha\beta\eta$-Konversion sondern nur mit syntaktischer Gleichheit arbeitet.

Die Korrektheits-, Vollständigkeits- und Optimalitätsresultate für LNT übertragen sich unmittelbar auf LNT$\lambda\upsilon$, da die beiden Systeme zueinander äquivalent sind.

Ausblick

In der jetzigen Fassung werden alle expliziten Substitutionen vollständig berechnet, bevor ein neuer LNT$\lambda\upsilon$-Schritt erfolgt. Dadurch werden auch unnötige Berechnungen ausgeführt, etwa in der rechten Seiten $\mathcal{X}_n$ eines Case-Ausdruckes, die *nicht* ausgewählt werden. Hier ist eine Optimierung möglich, indem nur die zur weiteren Berechnung benötigten Unterterme υ-normalisiert werden: in Case-Ausdrücken *case* t *of* $\overline{p_i : \mathcal{X}_i}$ also zunächst nur t und die p_i – erst nach Anwendung der **Case Select**-Regel ist eine Normalisierung des ausgewählten $\mathcal{X}_i$ nötig.

Eine weitere Fragestellung in diesem Zusammenhang ist, welche Verbesserungen durch andere Reduktionsstrategien möglich sind. Faßt man zum Beispiel (als allgemeinste Variante) die LNT$\lambda\upsilon$-Regeln und die Regeln des Systems $\boldsymbol{\lambda\upsilon\eta F}$ zusammen, ohne die Reihenfolge der Anwendung vorzuschreiben, dann können auf beliebige nicht-reine Terme (also Terme mit Substitutionsanteil $[s]$) LNT$\lambda\upsilon$-Regeln angewendet werden. So entsteht ein neuer Kalkül, in dem mehr Ableitungen möglich sind.

Die Verwendung der de Bruijn-Notation vereinfacht zwar das „Rechnen“ mit λ-Termen, nicht aber ihre Lesbarkeit. Analog zu unserem Ansatz läßt sich eine LNT-Variante schaffen, die mit expliziten Substitutionen für gewöhnliche λ-Terme arbeitet.

Anhang A

Übersicht über λ-Kalküle mit expliziten Substitutionen

A.1 $\lambda\sigma$-Kalküle

$\lambda\sigma$-Kalküle wurden in [ACCL91] vorgestellt und in [DHK95, DHKP96] für die Unifikation höherer Ordnung verwendet.

A.1.1 Der ungetypte $\lambda\sigma$-Kalkül

Die folgenden Definitionen stammen aus [ACCL91]:

Terme	$a ::= \underline{1} \mid (a\ a) \mid \lambda a \mid a[s]$
Substitutionen	$s ::= id \mid \uparrow \mid a \cdot s \mid s \circ s$

Beta	$(\lambda a)b$	$\to a[b \cdot id]$
VarId	$\underline{1}[id]$	$\to \underline{1}$
VarCons	$\underline{1}[a.s]$	$\to a$
App	$(a\ b)[s]$	$\to (a[s]\ b[s])$
Abs	$(\lambda a)[s]$	$\to \lambda(a[\underline{1} \cdot (s \circ \uparrow)])$
Clos	$(a[s])[t]$	$\to a[s \circ t]$
IdL	$id \circ s$	$\to s$
ShiftId	$\uparrow \circ\, id$	$\to \uparrow$
ShiftCons	$\uparrow \circ\, (a \cdot s)$	$\to s$
Map	$(a \cdot s) \circ t$	$\to a[t] \cdot (s \circ t)$
Ass	$(s_1 \circ s_2) \circ s_3$	$\to s_1 \circ (s_2 \circ s_3)$

Für die Normalformen $\sigma(t)$ von t gilt dann: $a \underset{\mathbf{Beta}}{\longrightarrow} b \Rightarrow \sigma(a) \overset{*}{\underset{\beta}{\longrightarrow}} \sigma(b)$.

Wegen der Existenz kritischer Paare werden noch folgende Regeln hinzugenommen:

Id	$a[id]$	$\rightarrow a$
IdR	$s \circ id$	$\rightarrow s$
VarShift	$\underline{1} \cdot \uparrow$	$\rightarrow id$
SCons	$\underline{1}[s] \cdot (\uparrow \circ s)$	$\rightarrow s$

A.1.2 Der ungetypte $\lambda\sigma$-Kalkül mit Namen

In [ACCL91] wird auch eine Variante des $\lambda\sigma$-Kalküls vorgestellt, die statt de Bruijn-Indizes die normale Darstellung von λ-Termen mit Variablen verwendet:

Terme	$a ::= x \mid (a\ a) \mid \lambda x.a \mid a[s]$
Substitutionen	$s ::= id \mid \uparrow \mid (a/x) \cdot s \mid s \circ s$

Beta	$(\lambda x.a)b$	$\rightarrow a[(b/x) \cdot id]$
Var1	$x[(a/x) \cdot s]$	$\rightarrow a$
Var2	$x[(a/y) \cdot s]$	$\rightarrow x[s]$ für $x \neq y$
Var3	$x[id]$	$\rightarrow x$
App	$(a\ b)[s]$	$\rightarrow (a[s]\ b[s])$
Abs	$(\lambda x.a)[s]$	$\rightarrow \lambda x.(a[(x/x) \cdot (s \circ \uparrow)])$
Clos	$(a[s])[t]$	$\rightarrow a[s \circ t]$
IdL	$id \circ s$	$\rightarrow s$
ShiftId	$\uparrow \circ id$	$\rightarrow \uparrow$
ShiftCons	$\uparrow \circ (a \cdot s)$	$\rightarrow s$
Map	$(a \cdot s) \circ t$	$\rightarrow a[t] \cdot (s \circ t)$
Ass	$(s_1 \circ s_2) \circ s_3$	$\rightarrow s_1 \circ (s_2 \circ s_3)$

A.1.3 Der ungetypte $\lambda\sigma$-Kalkül mit Meta-Variablen

Wir geben für die ungetypte Version die Syntax der Terme und das Regelsystem aus [DHK95] an:

Terme	$a ::= \underline{1} \mid X \mid (a\ a) \mid \lambda a \mid a[s]$
Substitutionen	$s ::= Y \mid id \mid \uparrow \mid a.s \mid s \circ s$

wobei $X \in \mathcal{X}$ und $Y \in \mathcal{Y}$ Term- bzw. Substitutions-Metavariablen sind.

Beta	$(\lambda a)b$	$\to a[b.id]$
App	$(a\ b)[s]$	$\to (a[s]\ b[s])$
VarCons	$\underline{1}[a.s]$	$\to a$
Id	$a[id]$	$\to a$
Abs	$(\lambda a)[s]$	$\to \lambda(a[\underline{1}.(s\circ\uparrow)])$
Clos	$(a[s])[t]$	$\to a[s\circ t]$
IdL	$id\circ s$	$\to s$
ShiftCons	$\uparrow\circ(a.s)$	$\to s$
AssEnv	$(s_1\circ s_2)\circ s_3$	$\to s_1\circ(s_2\circ s_3)$
MapEnv	$(a.s)\circ t$	$\to a[t].(s\circ t)$
IdR	$s\circ id$	$\to s$
VarShift	$\underline{1}.\uparrow$	$\to id$
SCons	$\underline{1}[s].(\uparrow\circ s)$	$\to s$
Eta	$\lambda(a\ \underline{1})$	$\to b$ falls $a =_\sigma b[\uparrow]$

A.2 $\lambda\upsilon$-Kalkül

Der $\lambda\upsilon$-Kalkül wurde in [Les94] vorgestellt und hat folgende Syntax und Regeln:

Terme	$a ::= \underline{\mathtt{n}} \mid (a\ a) \mid \lambda a \mid a[s]$
Substitutionen	$s ::= a/ \mid \Uparrow(s) \mid \uparrow$

Beta	$(\lambda a)b$	$\to a[b/]$
App	$(a\ b)[s]$	$\to (a[s]\ b[s])$
Lambda	$(\lambda a)[s]$	$\to \lambda(a[\Uparrow(s)])$
FVar	$\underline{1}[a/]$	$\to a$
RVar	$\underline{\mathtt{n+1}}[a/]$	$\to \underline{\mathtt{n}}$
FVarLift	$\underline{1}[\Uparrow(s)]$	$\to \underline{1}$
RVarLift	$\underline{\mathtt{n+1}}[\Uparrow(s)]$	$\to \underline{\mathtt{n}}[s][\uparrow]$
VarShift	$\underline{\mathtt{n}}[\uparrow]$	$\to \underline{\mathtt{n+1}}$

A.3 λxgc-Kalkül

Im folgenden präsentieren wir einen Kalkül mit expliziten Substitutionen, der mit den üblichen λ-Termen – also nicht in de Bruijn-Notation – arbeitet. Der λxgc-Kalkül verwendet wie der $\lambda\upsilon$-Kalkül keine explizite Komposition von Substitutionen, und es gibt eine *explizite Garbage Collection.*

Die folgenden Definitionen stammen aus [Ros96] und beruhen auf [Ros92, BR95].

Definition A.3.1 (λx-Präterme) Die Menge der λx-*Präterme* ist definiert durch

$$t ::= x \mid \lambda x.t \mid (t\, t) \mid t[x := t]. \qquad \square$$

λx-Terme werden als Restklassen von λ-Termen modulo α-Konversion eingeführt:

Definition A.3.2 (λx-Terme)

1. Die Menge der *freien Variablen* eines λx-Präterms M, $FV(M)$, wird induktiv definiert durch: $FV(x) = \{x\}$, $FV(\lambda x.M) = FV(M) \setminus \{x\}$, $FV((M\,N)) = FV(M) \cup FV(N)$ und

   $$FV(M[x := N]) := (FV(M) \setminus \{x\}) \cup FV(N).$$

   Ein λx-Präterm M heißt *geschlossen*, falls $FV(M) = \varnothing$.

2. Die *Umbennenung* aller freien Auftreten von y in M nach z wird mit $\{y \mapsto z\}M$ bezeichnet und induktiv über M definiert durch

   - $\{y \mapsto z\}x := z$, falls $x = y$,
   - $\{y \mapsto z\}x := x$, falls $x \neq y$,
   - $\{y \mapsto z\}\lambda x.M := \{y \mapsto z\}(\{x \mapsto x'\}\lambda x'.M)$ für ein $x' \notin FV(\lambda x.M) \cup \{y, z\}$,
   - $\{y \mapsto z\}(M\,N) := (\{y \mapsto z\}M\, \{y \mapsto z\}N)$ und
   - $\{y \mapsto z\}(M[x := N]) := (\{y \mapsto z\}\{x \mapsto x'\}M)[x' := \{y \mapsto z\}N]$ für ein $x' \notin FV(\lambda x.M) \cup \{y, z\}$.

3. α-Konversion von λx-Prätermen wird induktiv definiert durch $x =_\alpha x$; $\lambda x.M =_\alpha \lambda y.N$, falls $\{x \mapsto z\}M =_\alpha \{y \mapsto z\}N$ für ein $z \notin FV(MN)$; $(M\,N) =_\alpha (P\,Q)$, falls $M =_\alpha P$ und $N =_\alpha Q$ – und $M[x := N] =_\alpha P[y := Q]$, falls $N =_\alpha Q$ und $\{x \mapsto z\}M =_\alpha \{y \mapsto z\}P$ für ein $z \notin FV(MP)$.

4. Die Menge der λx-*Terme*, Λx, ist die Menge der λx-Präterme modulo $=_\alpha$. Ein λx-Term heißt *geschlossen*, wenn seine Repräsentanten geschlossene λx-Präterme sind. $\square$

Definition A.3.3 (λxgc-Reduktion) Auf Λx werden die folgenden Reduktionen definiert:

1. *Substitutionserzeugung*: $\rightarrow_b$ ist die kleinste Relation, die mit

   $$(\lambda x.M)N \rightarrow M[x := N] \qquad \text{(b)}$$

   kompatibel[1] ist.

[1] Kompatibilität haben wir in Definition 2.2.15 erklärt.

2. *Explizite Substitution*: $\to_x$ ist die kleinste Relation, die mit

$$x[x := N] \to N, \tag{xv}$$
$$x[y := N] \to x, \text{ falls } x \neq_\alpha y, \tag{xvgc}$$
$$(\lambda x.M)[y := N] \to \lambda x.M[y := N] \text{ und} \tag{xab}$$
$$(M_1\, M_2)[y := N] \to (M_1[y := N]\, M_2[y := N]) \tag{xap}$$

   kompatibel ist.

3. *Garbage Collection*: $\to_{gc}$ ist die kleinste Relation, die mit

$$M[x := N] \to M, \text{ falls } x \notin FV(M) \tag{gc}$$

   kompatibel ist. Falls $x \notin FV(M)$, heißt N *Garbage*.

4. λxgc-*Reduktion* ist $\underset{bxgc}{\longrightarrow} := \to_b \cup \to_x \cup \to_{gc}$, λx-*Reduktion* ist $\underset{bx}{\longrightarrow} := \to_b \cup \to_x$, und $\underset{xgc}{\longrightarrow} := \to_x \cup \to_{gc}$. □

Beispiel A.3.4 Wir betrachten Reduktionen des Terms $(\lambda y.(\lambda z.z)y)x$ im λ-Kalkül und im λxgc-Kalkül:

$$(\lambda y.(\lambda z.z)y)x \to_\beta (\lambda z.z)x \to_\beta x,$$

$$(\lambda y.(\lambda z.z)y)x \to_b ((\lambda z.z)y)[y := x] \to_b z[z := y][y := x]$$
$$\to_{xv} y[y := x] \to_{xv} x,$$

$$(\lambda y.(\lambda z.z)y)x \to_b ((\lambda z.z)y)[y := x] \to_{xap} (\lambda z.z)[y := x](y[y := x])$$
$$\to_{xab} (\lambda z.z[y := x])(y[y := x]) \to_{xv} (\lambda z.z[y := x])x \to_{gc} (\lambda z.z)x$$
$$\to_b z[z := x] \to_{xv} x.$$

Satz A.3.5 ([Ros96])
Jeder (ungetypte) λ-Term, der stark β-normalisierend ist, ist auch stark $\underset{bxgc}{\longrightarrow}$-normalisierend. D.h. $\underset{bxgc}{\longrightarrow}$ bewahrt die starke Normalisierung von $\to_\beta$. □

Symbolverzeichnis

Symbol	Bemerkung	Seite
$\mathbb{N}, \mathbb{N}_0$	natürliche Zahlen (ohne und mit 0)	5
$\mathcal{T}_0$	Grundtypen	6
$\mathcal{T}$	Typen	6
$\rightarrow$	Typ-Konstruktor	6
Σ	Funktionssymbole	6
$\tau(f)$	Typ von f	6
V_α	Variablen vom Typ α	6
$V(\mathcal{T})$	Vereinigung der V_α	6
$\mathcal{A}(\mathcal{T})$	Atome	6
$\mathcal{L}(\mathcal{T})$	korrekt getypte λ-Terme	6
$(e_1\ e_2)$	Applikation	6
$(\lambda x.e)$	λ-Abstraktion	6
$e : \tau$	e ist vom Typ τ	6
$\otimes$	Produkt-Typen	7
$Sub(t)$	Unterterme von t	7
$\|e\|$	Termgröße	7
$BV(e)$	gebundene Variablen	7
$FV(e)$	freie Variablen	7
$Ord(\tau)$	Ordnung des Typs τ	7
$(A, \rightarrow)$	Reduktionssystem	8
$\succ_\alpha$	α-Konversion	9
$\succ_\beta$	β-Konversion	9
$\succ_\eta$	η-Konversion	9
$\rightarrow_\alpha$	α-Reduktion	9
$\rightarrow_\beta$	β-Reduktion	9
$\rightarrow_\eta$	η-Reduktion	9
$\rightarrow_{\beta\eta}$	$\rightarrow_\beta \cup \rightarrow_\eta$	9
$\rightarrow^+$	transitive Hülle von $\rightarrow$	9
$\stackrel{*}{\longleftrightarrow}$	symmetrische/reflexive/transitive Hülle von $\rightarrow$	9

$\overset{*}{\longleftrightarrow}_\beta$	β-Äquivalenz	9
$\overset{*}{\longleftrightarrow}_\eta$	η-Äquivalenz	9
$\overset{*}{\longleftrightarrow}_{\beta\eta}$	$\beta\eta$-Äquivalenz	9
$t\!\downarrow_\beta$	β-Normalform von t	11
$Head(t)$	Kopfsymbol von t	11
$t\!\uparrow^\eta$	η-expandierte Form	12
$t\!\updownarrow_\beta^\eta$	lange $\beta\eta$-Normalform	12
$\mathcal{L}_{exp}$	η-expandierte Terme	13
$\mathcal{L}_\eta$	Abschluß von $\mathcal{L}_{exp}$	13
$Dom(\sigma)$	Träger einer Substitution	14
$Rng(\sigma)$	eingeführte Variablen	14
Id	Identitätssubstitution	14
$\sigma\|_{W'}$	Einschränkung	14
$\widehat{\sigma}$	Fortsetzung von σ	14
$\sigma \cup \theta$	Vereinigung von σ und θ	15
$\sigma \circ \theta$	Komposition von σ und θ	15
$=\ [W]$	gleich über W	15
$=_\beta\ [W]$	β-gleich über W	15
$\leq\ [W]$	allgemeiner über W	15
$\leq_\beta\ [W]$	β-allgemeiner über W	15
$\leq_\eta\ [W]$	η-allgemeiner über W	15
$\leq_{\beta\eta}\ [W]$	$\beta\eta$-allgemeiner über W	15
$M_1 \cup M_2$	Vereinigung von Multimengen	16
$\langle s, t\rangle$	Termpaar	20
$U(S), U(s,t)$	Mengen der Unifikatoren	20, 24
$mgu(S)$	allgemeinster Unifikator von S	21
σ_S	allgemeinster Unifikator von S	21
$\mathcal{ST}$	Transformationsregeln (1. Ordnung)	21
$mgu(S)[W]$	mgu von S, weg von W	23
$CSU(S)[W]$	vollständige Unifikatormenge von S, weg von W	25
$\mathcal{HT}$	Transformationsregeln (höherer Ordnung)	30
$\mathcal{CT}$	Transformationsregeln (höherer Ordnung)	32
$\mathcal{P}$	Prädikatsymbole	34
$\rightarrow_P$	Termersetzungsschritt mit Regel aus P	34
$\rightsquigarrow_{[p,l=r,\sigma]}$	Narrowing-Schritt	35
π	Muster	38
rule	Regelknoten in definierendem Baum	38
branch	Verzweigungsknoten in definierendem Baum	38
$\mathcal{E}val$	Auswertung einer Anfrage	39
case X *of*	Case-Ausdruck	40

$\rightarrow_{\eta'}$	η'-Reduktion	59
Γ	Kontext (getypter $\lambda\upsilon$-Kalkül)	60
$\boldsymbol{\lambda\upsilon\eta F}$	$\lambda\upsilon\eta F$-Kalkül	60
$\boldsymbol{\upsilon F}$	$\boldsymbol{\lambda\upsilon\eta F}$ ohne **Beta** und **Eta**	60
$\langle X/s \rangle$	FO-Substitution	65
$f^n_{\mathbf{case}}$	Funktionssymbole für Case-Konstrukte	67
$\mathbf{i}$	Einbettung von *case*-Termen	67
$\mathbf{e}$	anschaulich: $\mathbf{i}^{-1}$ in de Bruijn-Notation	67
$DB(t)$	de Bruijn-Übersetzung von Case-Termen	67
$\sigma' = DB(\sigma)$	de Bruijn-Übersetzung von Substitutionen	68
$DB_{[\lambda\overline{x_k}]}(t)$	de Bruijn-Übersetzung im Kontext von Bindern	69
$f_{\rightarrow}$	Funktionssymbol, das $\rightarrow$ repräsentiert	69
LNT$\lambda\upsilon$	LNT im $\lambda\upsilon$-Kalkül	70
$\underset{LNT\lambda\upsilon}{\Longrightarrow}$	Reduktion mit LNT$\lambda\upsilon$-Regeln	71

Literaturverzeichnis

[ACCL91] M. Abadi, L. Cardelli, P.-L. Curien und J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991. Auch erschienen als: Rapport de Recherche N° 1176, INRIA, 1990.

[AEH94] S. Antoy, R. Echahed und M. Hanus. A needed narrowing strategy. In *Proc. 21st ACM Symposium on Principles of Programming Languages*, Seiten 268–279, Portland, 1994.

[Ant92] S. Antoy. Definitional trees. In *Proc. of the 3rd International Conference on Algebraic and Logic Programming*, Seiten 143–157. Springer LNCS 632, 1992.

[BA95] F. Baader und C. A. Albayrak. *Termersetzungssysteme, Skript zur Vorlesung an der RWTH Aachen.* Aachener Beiträge zur Informatik, Band 12. Verlag der Augustinus-Buchhandlung Aachen, 1995.

[Bar84] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* Studies in Logic and the Foundation of Mathematics. North-Holland, Amsterdam, 2. Auflage, 1984.

[BBLRD95] Z. E. A. Benaissa, D. Briaud, P. Lescanne und J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation, 1995.

[BGL+87] P.G. Bosco, E. Giovannetti, G. Levi, C. Moiso und C. Palamidessi. A complete semantic characterization of K-LEAF, a logic language with partial functions. In *Proc. 4th IEEE Internat. Symposium on Logic Programming*, Seiten 318–327, San Francisco, 1987.

[BR95] R. Bloo und K.H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN '95 – Computer Science in the Netherlands*, Seiten 62–72, 1995.

[Bri95] D. Briaud. An explicit *Eta* rewrite rule. In M. Dezani-Ciancaglini und G. Plotkin (Hrsg.), *Typed Lambda Calculi and Applications. Proc. of the Second International Conference on Typed Lambda Calculi and Applications, TLCA '95*, volume 902 of *Lecture Notes in Computer Science*, Seiten 94–108, Edinburgh, UK, April 1995. Springer. Auch erschienen als Rapport de Recherche 2417, INRIA, 1994.

[Bri96] D. Briaud. Higher order unification as a typed narrowing. In K. Schulz und S. Kepser (Hrsg.), *Proceedings of UNIF '96*, CIS-Bericht 96-91. Universität München, 1996.

[Bri97a] D. Briaud. Private Kommunikation via E-Mail über eine Erweiterung des $\lambda\upsilon$-Kalküls um Konstanten, August 1997.

[Bri97b] D. Briaud. Substitutions explicites et unification d'ordre supérieur. Thèse de l'Université Henri poincaré, Nancy I, September 1997.

[CHL92] P.-L. Curien, Th. Hardin und J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. Technical report, INRIA, 1992.

[Chu40] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[CR35] A. Church und J.B. Rosser. Some properties of conversion. *Transactions of the AMS*, 39:472–482, 1935.

[dB72] N. G. de Bruijn. Lambda calculus with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. In *Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen*, **75**(5), Seiten 381–392, 1972.

[DHK95] G. Dowek, Th. Hardin und C. Kirchner. Higher order unification via explicit substitutions. Rapport de Recherche No. 2709, INRIA Lorraine, 1995.

[DHKP96] G. Dowek, Th. Hardin, C. Kirchner und F. Pfenning. Unification via explicit substitutions: The case of higher-order patterns. In M. Maher (Hrsg.), *Proceedings of the Joint International Conference and Symposium on Logic Programming*, Bonn, Germany, September 1996. MIT Press. Wird erscheinen.

[Fri85] L. Fribourg. SLOG: A logic programming language interpreter based on clausal superposition and rewriting. In *Proc. IEEE Internat. Symposium on Logic Programming*, Seiten 172–184, Boston, 1985.

[GMS95] M. Goossens, F. Mittelbach und A. Samarin. *Der LATEX-Begleiter*. Addison-Wesley, 1995.

[Gol81] W. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science 13*, Seiten 225–230, 1981.

[Han87] M. Hanus. *Problemlösen mit Prolog*. Teubner, Stuttgart, 1987. 2. überarbeitete und erweiterte Auflage.

[Han94a] C. Hankin. *Lambda Calculi. A Guide for Computer Scientists*. Graduate Texts in Computer Science, 3. Oxford University Press, 1994.

[Han94b] M. Hanus. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 19&20:583–628, 1994.

[Han95] M. Hanus. Logikprogrammierung. Vorlesungsmitschrift (Sommersemester 1995), RWTH Aachen, 1995.

[Han97a] M. Hanus. Teaching functional and logic programming with a single computation model. In *Proc. Ninth International Symposium on Programming Languages, Implementations, Logics, and Programs (PLILP'97)*. Springer LNCS (to appear), 1997.

[Han97b] M. Hanus (Hrsg.). Curry: An integrated functional logic language. Erhältlich unter `http://www-i2.informatik.rwth-aachen.de-/~hanus/curry`, 1997.

[Her30] J. Herbrand. *Recherches sur la Théorie de la Démonstration*. Dissertation, Universität Paris, 1930.

[HKMN95] M. Hanus, H. Kuchen und J.J. Moreno-Navarro. Curry: A truly functional logic language. In *Proc. ILPS'95 Workshop on Visions for the Future of Logic Programming*, 1995.

[HL89] Th. Hardin und J.-J. Lévy. A confluent calculus of substitutions. In *France-Japan Artificial Intelligence and Computer Science Symposium*. Izu, 1989.

[Hog90] C. J. Hogger. *Essentials of Logic Programming*. Graduate Texts in Computer Science, 1. Oxford University Press, 1990.

[HP96] M. Hanus und C. Prehofer. Higher-order narrowing with definitional trees. Aachener Informatik-Berichte 96-2, Fachgruppe Informatik, RWTH Aachen, 1996. Auch als Kurzfassung erschienen in *Proc. Seventh International Conference on Rewriting Techniques and Applications (RTA '96)*, Springer LNCS 1103, S. 138–152, 1996.

[HS86] J. Hindley und J. Seldin. *Introduction to Combinators and Lambda Calculus*. Cambridge University Press, 1986.

[HS91] M. Hanus und A. Schwab. The implementation of the functional-logic language ALF. FB Informatik, Univ. Dortmund, 1991.

[Hue73] G. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1(1):27–57, 1973.

[Hue76] G. Huet. *Résolution d'équations dans langues d'ordre* $1, 2, \ldots, \omega$. Thèse d'Etat, Université de Paris VII, 1976.

[Ind96] K. Indermark. Funktionale Programmierung. Vorlesungsmitschrift (Wintersemester 1995/96), RWTH Aachen, 1996.

[Les94] P. Lescanne. From $\lambda\sigma$ to λv: A journey through calculi of explicit substitutions. In *Conference Record of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'94)*, Seiten 60–69, Portland, Oregon, January 17–21, 1994. ACM Press.

[LRD94] P. Lescanne und J. Rouyer-Degli. The calculus of explicit substitutions λv. Technical Report RR-2222, INRIA-Lorraine, January 1994.

[Mel95] P.-A. Mellies. Typed λ-calculi with explicit substitutions may not terminate. In M. Dezani-Ciancaglini und G. Plotkin (Hrsg.), *Typed Lambda Calculi and Applications. Proc. of the Second International Conference on Typed Lambda Calculi and Applications, TLCA '95*, volume 902 of *Lecture Notes in Computer Science*, Seiten 328–334, Edinburgh, UK, April 1995. Springer.

[MM82] A. Martelli und U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, April 1982.

[MR92] J.J. Moreno-Navarro und M. Rodríguez-Artalejo. Logic programming with functions and predicates: The language BABEL. *Journal of Logic Programming*, 12:191–223, 1992.

[New42] M.H.A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2), 1942.

[Pre95] C. Prehofer. *Solving Higher-Order Equations: From Logic to Programming.* Dissertation, TU München, 1995. Auch erschienen als Technischer Bericht I9508.

[Río93] A. Ríos. *Contributions à l'étude des λ-calculs avec des substitutions explicites.* Dissertation, Univ. Paris VII, 1993.

[Ros92] K.H. Rose. Explicit cyclic substitutions. In M. Rusinowitch und J.-L. Rémy (Hrsg.), *CTRS '92 – 3rd International Workshop on Conditional Term Rewriting Systems*, LNCS 656, Seiten 36–50. Springer-Verlag, 1992.

[Ros96] K. H. Rose. Explicit Substitution – Tutorial & Survey. BRICS Lecture Series LS-96-3, 1996.

[Sch24] M. Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305–316, 1924. Auch ins Englische übersetzt und erschienen als "On the building blocks of mathematical logic" in [vH67, S. 355–366].

[SG89] W. Snyder und J. Gallier. Higher-order unification revisited: Complete sets of transformations. *Journal of Symbolic Computations*, 8:101–140, 1989.

[Sny88] W. S. Snyder. *Complete sets of transformations for General Unification.* Dissertation, Department of Computer and Information Science, University of Pennsylvania, 1988.

[vH67] J. van Heijenoort (Hrsg.). *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931.* Harvard University Press, Cambridge, Massachusetts, 1967.

Stichwortverzeichnis

T

U

V

X

Zitate

Versicherung gemäß § 19 (6) DPO Mathematik

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

Aachen, 29. September 1997

....

Hans-Georg Eßer
RWTH Aachen, Matr.-Nr. 184383