Bedingte Narrowing-Verfahren mit verzögerter Auswertung

von
Bernd Braßel
Matrikelnummer 197353

Diplomarbeit

im Fach Informatik

vorgelegt an der
Mathematisch-Naturwissenschaftlichen Fakultät der
Rheinisch-Westfälischen Technischen Hochschule Aachen
im Januar 1999

Angefertigt am
Lehr- und Forschungsgebiet für Informatik II
bei
Univ.-Prof. Dr. Michael Hanus

Ich danke meinen Eltern, ohne deren großartige und großzügige Unterstützung mein Studium und damit die vorliegende Arbeit unmöglich gewesen wäre.

Ich danke auch meinem Bruder für seine redaktionelle Hilfestellung sowie

Herrn Prof. Dr. Michael Hanus für die Themenstellung und Betreuung dieser Arbeit.

Ich versichere, die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwandt zu haben.

Aachen, den 20. Januar 1999

Inhaltsverzeichnis

Kapitel 1

Einleitung

Den größten Anteil der vorliegenden Arbeit nimmt das Thema „Transformationen bedingter Termersetzungssysteme in unbedingte“ ein. Auch wenn es auf den ersten Blick nicht so scheinen mag, haben derartige Transformationen eine direkte Verbindung zu bedingten Narrowing-Verfahren. Wie diese Verbindung aussieht, was (un)bedingte Termersetzungssysteme überhaupt sind, und was man mit ihnen machen kann, soll in diesem ersten Kapitel erklärt werden.

Dazu wird zunächst einmal angesprochen, wie das Gebiet der Termersetzungssysteme innerhalb der (noch jungen) Geschichte der Informatik eingeordnet werden kann (Abschnitt 1.1). Danach werden die grundlegenden Definitionen für unbedingte Termersetzungssysteme gegeben und an Beispielen erläutert (Abschnitt 1.2). Dann wird das Forschungsgebiet, zu dem diese Arbeit gehört, beschrieben (Abschnitt 1.3) und danach der Begriff des bedingten Termersetzungssystems eingeführt (Abschnitt 1.4). Abschließend wird dargestellt, welche Forderungen an eine Transformation bedingter Termersetzungssysteme in unbedingte zu stellen sind (Abschnitt 1.5).

Das zweite Kapitel ist der Vorstellung bereits veröffentlichter Transformationen gewidmet, wobei die am besten geeignete ausgewählt wird. Diese wird dann in den Kapiteln 3 und 4 ausführlich analysiert und verbessert. Dabei werden wichtige Eigenschaften bewiesen, und es wird auch ausgeführt, was eventuell aus dieser Arbeit auch für die Analyse und Konstruktion weiterer vergleichbarer Transformationen von Interesse ist (Abschnitt 3.5). Abschließend folgt eine Zusammenfassung der erreichten Ergebnisse.

1.1 Ein kurzer historischer Abriß

Anfang dieses Jahrhunderts ging ein Ruck durch die Welt (nicht nur) der mathematischen Wissenschaft, als Gödel seine Ergebnisse über die prinzipielle Unvollständigkeit mathematischer Systeme veröffentlichte. Diese Ergebnisse fanden einen reichen Widerhall in der mathematischen Grundlagenforschung. Mit ihnen war zum Beispiel das Hilbertprogramm als gescheitert anzusehen, welches die Überführung der Mathematik mit finiten Mitteln in einen Formalismus vorsah, vergleiche [Bro96].

Kurz gesagt, sahen viele Mathematiker ihre bis dahin heile Welt zerbröckeln, es soll sogar zu einigen Selbstmorden aufgrund der neuen Erkenntnisse gekommen sein. Doch entgegen den Unkenrufen, die eine Mathematik als Wissenschaft schon für gescheitert erklären wollten, dauerte es nicht lange, bis eine andere Art der Herangehensweise auch neue grundlegende Ergebnisse liefern konnte. In Konsequenz des neuen Verständnisses war man nun bestrebt, Modelle und Theorien dafür zu schaffen, was tatsächlich effektiv beweisbar, beziehungsweise berechenbar ist. Nur zwei Forscher seien in diesem Zusammenhang hervorgehoben, der englische Mathematiker Alan M. Turing (1912-1954) und der amerikanische Logiker Alonzo Church (1903-1995). Sie stehen für zwei Definitionen für den Begriff der Berechenbarkeit, die Turing-Maschine und den λ-Kalkül. Diese, wie auch viele weitere Definitionen haben zu derselben Klasse der berechenbaren Funktionen geführt, was Church zu seiner berühmten, 1936 formulierten These veranlaßte, siehe zum Beispiel [Cla88]:

> Jede im intuitiven Sinne berechenbare Funktion ist Turing-berechenbar.

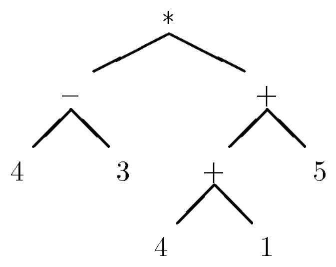
Weitere Definitionen mit demselben Ergebnis sind etwa die μ-rekursiven Funktionen und die *Termersetzungssysteme*.

1.2 Unbedingte Termersetzungssysteme

Was aber ist genau ein Termersetzungssystem, und warum ist es für die Informatik von Interesse?

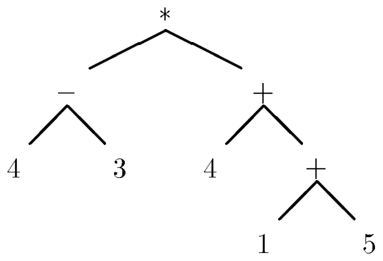
Dazu zunächst eine Beschreibung des Begriffs Term. Ein Term entsteht, so der Duden für Informatik, „indem Operanden mit Operatoren verknüpft werden“[Cla88]. Wichtig ist dabei vor allem die *Struktur*, die in jedem Term steckt. Diese läßt sich mit Hilfe von Bäumen veranschaulichen, in denen die Operatoren durch Knoten, die Operanden durch Blätter dargestellt werden.

Beispiel 1.1
Ein Term ist etwa $(4 - 3) * ((4 + 1) + 5)$, als Baum



Allgemein ist man gewohnt, einen Term wie in diesem Beispiel stark mit seinem „Ergebnis“ in Verbindung zu bringen. Dabei spielt die korrekte Klammerung oft keine wichtige Rolle, je nach Operation. Zum Beispiel kann man bei der üblichen mit den Zeichen $+$, $*$ und $-$ verbundenen Operationen auch anders klammern, wie etwa $(4 - 3) * (4 + (1 + 5))$, ohne das „Ergebnis“ zu verändern. Wenn es jedoch um Terme geht, so beschreibt jede Klammerung einen anderen Term, wie die Baumdarstellung zeigt.

Beispiel 1.2



Deshalb, und um Zweideutigkeiten durch versehentliches Weglassen von Klammern zu verhindern, werden Terme hier mit einer immer eindeutigen Notation versehen. Dies erreicht man durch Voranstellen der „Operationen" vor die „Operanden". Aus $(4-3)*((4+1)+5)$ wird in der sogenannten Präfixnotation $*(-(4,3),+(+(4,1),5))$. Ist nur die Stelligkeit einer jeden „Operation" bekannt, also die Anzahl der erwarteten Argumente, so bleibt die Notation auch ohne Klammern eindeutig, also $* \ - \ 4 \ 3 \ + \ + \ 4 \ 1 \ 5$. Aber auch, wenn im folgenden diese Stelligkeit immer als bekannt vorausgesetzt wird, bleiben die Klammern aus Gründen der besseren Übersichtlichkeit.

Die Begriffe „Ergebnis" und „Operation" wurden in Anführungszeichen geschrieben. Der Grund dafür ist, daß diese Begriffe hier noch nicht angebracht sind. Solange es nur um Terme und ihren Aufbau geht ist jedes „+" und jede „3" nur ein Symbol mit einer bestimmten Stelligkeit, in diesem Fall nämlich zwei und null. Die 3 ist also ebenfalls nur ein Symbol, vorerst ohne einen „Sinn" mit ihr zu verbinden. Solche nullstelligen Funktionssymbole werden als Konstantensymbole bezeichnet. Um diese „Sinnfreiheit" zu dokumentieren werden im weiteren „nichtssagende" Symbole wie f, g und h, zur (übersichtlichen aber nicht notwendigen) Unterscheidung für Konstanten auch a, b oder c verwandt. Ist f ein zweistelliges Funktionssymbol, so wird dies als $f^{(2)}$ notiert. Die gesamte Menge der jeweils verwandten Funktionssymbole wird als *Signatur* bezeichnet. Daher also die erste Definition:

Definition 1.2.1: (Signatur)

Eine Signatur Σ ist eine Menge von Funktionssymbolen, wobei jedem Funktionssymbol f eindeutig eine Stelligkeit $n \in \mathbb{N}$, im Zeichen $f^{(n)} \in \Sigma$, zugeordnet ist. $\diamond$

Beispiel 1.3
Mit $\Sigma = \{a^{(0)}, b^{(0)}, f^{(1)}, g^{(2)}\}$ ist $f(g(a, f(g(x, b))))$ ein *Term über* Σ.

An diesem Beispiel fällt das Zeichen x auf, welches nirgendwo in der Signatur erscheint. Es handelt sich hierbei nicht um ein Funktionssymbol, sondern um eine *Variable*. Variablen, bezeichnet mit x, y oder z, stellen sozusagen Löcher innerhalb eines Termes dar, die dann durch eine Belegung mit einem Term gefüllt werden können. Damit ist das grundlegende Prinzip für den Aufbau eines Terms angesprochen. Jedes Funktionssymbol hat entsprechend seiner Stelligkeit Argumente, die wiederum selbst Terme sein können oder aber Konstanten bzw. Variablen. Daher ist der Begriff des Terms definiert durch:

Definition 1.2.2: (Terme)

Die Menge $\mathcal{T}_\Sigma(X)$ der Terme über der Signatur Σ und der Variablenmenge X ist induktiv definiert als die kleinste Menge mit:

- Aus $x \in X$ folgt $x \in \mathcal{T}_\Sigma(X)$
- Aus $n \geq 0$ und $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(X)$ sowie $f^{(n)} \in \Sigma$ folgt $f(t_1, \ldots, t_n) \in \mathcal{T}_\Sigma(X)$.

◇

Im folgenden benutzte Zeichen für Terme sind s, t, u, v und w.

Beispiel 1.4
Mit $\Sigma = \{a^{(0)}, b^{(0)}, f^{(1)}, g^{(2)}\}$ und $X = \{x_1, x_2, x, y, z\}$ ist $f(g(x, b))$ ein Term aus $\mathcal{T}_\Sigma(X)$, weil

1. x eine Variable und damit ein Term ist,
2. b ein nullstelliges Funktionssymbol ist, daher alle (nämlich null) Argumente von b Terme sind und b also ein Term ist.
3. g ein zweistelliges Funktionssymbol ist, mit 1. und 2. daher auch $g(x, b)$.
4. f ein einstelliges Funktionssymbol ist, mit 3. daher auch $f(g(x, b))$.

Σ, X und $\mathcal{T}_\Sigma(X)$ aus diesem Beispiel werden auch für die folgenden Beispiele weiter verwandt.

Jeder Term t ist also wiederum aus Termen zusammengesetzt. Um diese *Teilterme* von t gezielt ansprechen zu können, braucht man Bezeichnungen wie „der zweite Argumentterm des ersten Argumentes von $f(g(x, b))$ ist b“. Allgemein wird dieser gezielte Zugriff mit dem Begriff der Position erreicht. Eine Position ist eine Folge von natürlichen Zahlen, also in diesem Fall $1 \cdot 2$. (Die Reihenfolge geht von oben im Term nach unten, daher umgekehrt als man eventuell erwarten würde.) Solche Positionen werden mit p oder q bezeichnet. Meint man den Term selbst, so ist die Folge der Zahlenangaben folglich leer. Dies wird mit ε bezeichnet. Diese Notation folgt der üblichen Schreibweise für die Worte A^* über einem Alphabet A, in diesem Fall das Alphabet der natürlichen Zahlen. Die Operation „$\cdot$“ ist dann das Mittel, aus einzelnen Buchstaben Worte zusammenzusetzen und wird als Wortkonkatenation bezeichnet.

Definition 1.2.3: (Position p, Teilterm an der Stelle p)
Sei Σ eine Signatur und X eine Menge von Variablen.
Dann ist eine Position p innerhalb eines Termes t aus $\mathcal{T}_\Sigma(X)$ ein Wort aus $\mathbb{N}^*$. Der Teilterm von t an der Stelle p, im Zeichen $t|_p$, ist definiert durch:

$$t|_p = \begin{cases} t & : \ p = \varepsilon \\ t_i|_q & : \ p = i \cdot q, \ i \in \{1, \ldots, n\}, \ t = f(t_1, \ldots, t_n) \end{cases}$$

$Pos(t)$ ist die Menge aller Positionen, für die $t|_p$ definiert ist. ◇

Beispiel 1.5

$$f(g(x, b))|_{1 \cdot 2} = g(x, b)|_2 = b|_\varepsilon = b$$

Nimmt man aus dem einbuchstabigen Wort „2“ den letzten Buchstaben heraus, so bleibt eben nichts als Leere. Daran sieht man den manchmal doch nützlichen Effekt eines wohlplazierten Nichts.

> Man formt Ton zu einem Gefäß, doch erst durch das Nichts im Innern kann man es benutzen [Lao].

$f(g(x,b))|_7$ zum Beispiel ist nicht definiert, 7 gehört also nicht zu der Menge der Positionen von $f(g(x,b))$.

Nun kommt der zweite Teil des Begriffes Termersetzungssystem, die *Ersetzung*. Noch ist unklar, wie mit Hilfe von Termen gerechnet werden kann, oder auch kurz, „wozu das alles gut sein soll". Wie man sich aber leicht vorstellen kann, ist der Schlüssel zu einer sinnvollen Anwendung die Ersetzung etwa von $(4+1)$ in dem Term $(4-3)*((4+1)+5)$ durch beispielsweise 5, sodaß also $(4-3)*(5+5)$ entsteht. Dazu die Definition für die Ersetzung eines Teilterms:

Definition 1.2.4: (Ersetzung eines Teilterms)
Sei Σ eine Signatur, X eine Menge von Variablen, s und t Terme aus $\mathcal{T}_\Sigma(X)$ sowie p eine Position in s.
Dann ist die Ersetzung des Teilterms $s|_p$ durch t, im Zeichen $s[p \longleftarrow t]$, definiert durch:

$$s[p \longleftarrow t] = \begin{cases} t & : \; p = \varepsilon \\ f(t_1, \ldots, t_i[q \longleftarrow t], \ldots, t_n) & : \; p = i \cdot q, \; i \in \{1, \ldots, n\}, \; t = f(t_1, \ldots, t_n) \end{cases}$$

$\diamond$

Beispiel 1.6
Mit dem oben angesprochenen Term (jetzt in Präfixnotation) ergibt sich also

$$\begin{aligned} & *(-(4,3), +(+(4,1),5))[2 \cdot 1 \longleftarrow 5] \\ = \; & *(-(4,3), +(+(4,1),5)[1 \longleftarrow 5]) \\ = \; & *(-(4,3), +(+(4,1)[\varepsilon \longleftarrow 5],5)) \\ = \; & *(-(4,3), +(5,5)) \end{aligned}$$

Bevor das bisherige Wissen zum Begriff des Termersetzungs*system* zusammengesetzt werden kann, ist nur noch die Belegung von Variablen eines Terms genauer zu beschreiben. Dazu zunächst die Abbildung, die „herausfindet", welche Variablen sich in einem Term befinden:

Definition 1.2.5: (Variablen eines Terms)
Sei Σ eine Signatur und X eine Menge von Variablen.
Dann ist die Abbildung $var : \mathcal{T}_\Sigma(X) \longrightarrow \mathcal{P}(X)$ definiert durch:

$$var(t) = \begin{cases} \{t\} & : \; t \in X \\ var(t_1) \cup \ldots \cup var(t_n) & : \; t = f(t_1, \ldots, t_n), \; n \geq 0 \end{cases}$$

Ein Term t, für den $var(t) = \emptyset$ gilt, heißt Grundterm. $\diamond$

Beispiel 1.7

$$\begin{aligned} var(f(g(x, f(g(x,y))))) &= var(g(x, f(g(x,y)))) \\ &= var(x) \cup var(f(g(x,y))) \\ &= \{x\} \cup var(g(x,y)) \\ &= \{x\} \cup var(x) \cup var(y) \\ &= \{x\} \cup \{x\} \cup \{y\} \\ &= \{x, y\} \end{aligned}$$

Auffallend an den letzten wie auch an den folgenden Definitionen ist die Ähnlichkeit in der Struktur. Zuerst wird definiert, was im Falle eines nullstelligen, sozusagen atomaren Termes zu tun ist, dann wird der komplexere Fall eines zusammengesetzten Terms auf seine Teilterme zurückgeführt. Diese Art von Definition nennt man rekursiv. Die einfache Weise, in der solche Definitionen für Terme gemacht werden können, ist einer der wesentlichen Gründe, Termersetzungssysteme als Grundlage für die Computerprogrammierung zu verwenden, wie später noch weiter ausgeführt werden wird.

Zunächst aber zu dem Begriff der Substitution, oder auch Belegung der Variablen eines Terms. Eine Substitution bildet einen Term auf einen anderen Term ab, wobei aber die einzigen Veränderungen da auftreten dürfen, wo der Ausgangsterm Variablen aufweist. Obwohl jede Substitution σ für alle möglichen Terme über einer bestimmten Menge $\mathcal{T}_\Sigma(X)$ definiert ist, reicht es aus anzugeben, auf was σ die Variablen abbildet. Dieser Umstand wird mit „Eindeutigkeit der homomorphen Erweiterung" bezeichnet. Eine Variablenbelegung kann nämlich nur auf eine einzige Weise zu einem Homomorphismus erweitert werden.

Ein Homomorphismus ist hier eine Abbildung $\phi : \mathcal{T}_\Sigma(X) \longrightarrow \mathcal{T}_\Sigma(X)$, für die gilt:

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

Der Begriff kommt daher, daß die Existenz einer solchen Abbildung gewisse Gleichförmigkeiten in der Struktur von Definitions- und Bildmenge von ϕ implizieren. Da es sich hier aber um ein und dieselbe Menge handelt, ist das nicht sonderlich bemerkenswert. Selbstähnlichkeit ist eine häufig feststellbare Eigenschaft.

Definition 1.2.6: (Substitution)

Sei Σ eine Signatur, X eine Menge von Variablen und σ_0 eine Abbildung $\sigma_0 : X \longrightarrow \mathcal{T}_\Sigma(X)$ derart, daß nur für endlich viele x aus X gilt: $\sigma_0(x) \neq x$.
Dann heißt die homomorphe Erweiterung $\sigma : \mathcal{T}_\Sigma(X) \longrightarrow \mathcal{T}_\Sigma(X)$ von σ_0, definiert durch

$$\sigma(t) = \begin{cases} \sigma_0(t) & : \ t \in X \\ f(\sigma(t_1), \ldots, \sigma(t_n)) & : \ t = f(t_1, \ldots, t_n), \ n \geq 0 \end{cases}$$

eine Substitution. ◇

Außer σ werden auch die Zeichen τ und π für Substitutionen stehen.

Beispiel 1.8

Hier wird deutlich, wie eine Substitution im folgenden notiert wird:
Mit $\sigma = \{x \mapsto f(a), y \mapsto g(x, f(b))\}$ ist

$$\begin{aligned} \sigma(f(g(x, f(g(z, y))))) &= f(\sigma(g(x, f(g(z, y))))) \\ &= f(g(\sigma(x), \sigma(f(g(z, y))))) \\ &= f(g(f(a), f(\sigma(g(z, y))))) \\ &= f(g(f(a), f(g(\sigma(z), \sigma(y))))) \\ &= f(g(f(a), f(g(z, g(x, f(b)))))) \end{aligned}$$

Jetzt ist das nötige Rüstzeug vorhanden, um den Begriff des Termersetzungssystems zu definieren und danach auch dessen Arbeitsweise. Ein solches System besteht neben der zum Termaufbau nötigen Signatur und Variablenmenge aus einer Menge von Regeln, die angeben, welche Terme ersetzt werden dürfen.

Zum Beispiel würde die Regel $(4 + 1) \longrightarrow 5$ die weiter oben aufgeführte Ersetzung erlauben. Diese Regel ist natürlich nicht sehr sinnvoll, da es wohl einer ganzen Menge derartiger Regeln bedürfte, um die Addition aller vierstelligen Zahlen zu ermöglichen. Genau das ist, wie sich noch zeigen wird, der Grund für die Verwendung von Variablen. Mit ihrer Hilfe lassen sich viele Berechnungen durch wenige Regeln definieren.

Definition 1.2.7: (Unbedingtes Termersetzungssystem)
Sei Σ eine Signatur, X eine abzählbare Menge von Variablen und P eine endliche Menge von Regeln der Form

$$\rho \ : \ l \longrightarrow r$$

mit $l, r \in \mathcal{T}_\Sigma(X)$ und $var(l) \supseteq var(r)$.
Dann ist $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein unbedingtes Termersetzungssystem. $\diamond$

Ein Zeichen für (un)bedingte Termersetzungssysteme ist $\mathcal{R}$. Unbedingte Termersetzungssysteme sind das, was allgemein unter einem Termersetzungssystem verstanden wird. „Unbedingt" soll hier nur die Unterscheidung zu den in dieser Arbeit hauptsächlich interessierenden bedingten Termersetzungssystemen bilden. Einzelne Regeln werden mit ρ bezeichnet. In Beispielen werden aus Gründen der Übersicht Termersetzungssysteme nur durch die Folge der Regeln aus P angegeben, da sich daraus Signatur wie Variablenmenge ergibt.

Die nächste Definition gibt an, wie die Regeln eines Termersetzungssystems zu verwenden sind. Bevor dann ein ausführliches Beispiel die Arbeitsweise erläutert.

Definition 1.2.8: (Unbedingte Reduktionsschritte)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein Termersetzungssystem, $s, t \in \mathcal{T}_\Sigma(X)$, p eine Position in s, $\rho : l \longrightarrow r$ eine Regel aus P und σ eine Substitution.
Die Relation der Reduktionsschritte $\longrightarrow_\mathcal{R} \subseteq \mathcal{T}_\Sigma(X) \times \mathcal{T}_\Sigma(X)$ ist definiert durch

$$s \longrightarrow_{\mathcal{R}[\rho,p,\sigma]} t \text{ genau dann, wenn } s|_p = \sigma(l) \text{ und } t = s[p \longleftarrow \sigma(r)]$$

$\diamond$

Die Angaben von $\mathcal{R}, \rho, p$ und σ können auch weggelassen werden, falls sie aus dem Kontext klar ersichtlich oder aber unerheblich sind.

Es folgt ein ausführliches Beispiel, um die Arbeitsweise eines Termersetzungssystems zu verdeutlichen.

Beispiel 1.9
Dabei ist $\Sigma = \{0^{(0)}, s^{(1)}, +^{(2)}\}$ und $X = \{x, y\}$.
Wenn man sich die natürlichen Zahlen mit Hilfe des s-Symbols (successor = Nachfolger) aufgebaut denkt - also 0 ist 0, der Nachfolger von 0 ($s(0)$) ist die Eins, $s(s(s(0)))$ entspricht der Drei usw. - dann läßt sich die Addition auf einfache Weise definieren als

$$\mathcal{R} = \begin{pmatrix} 1: & +(0,y) & \longrightarrow & y \\ 2: & +(s(x),y) & \longrightarrow & s(+(x,y)) \end{pmatrix}$$

In diesem System soll nun die hochkomplizierte Aufgabe $2+3$ berechnet, also der Term $+(s(s(0)), s(s(s(0))))$ reduziert werden, solange das möglich ist. Dazu ist zunächst eine Substitution σ zu suchen, so daß σ angewandt auf eine der beiden linken Regelseiten einen Teilterm von $t = +(s(s(0)), s(s(s(0))))$ ergibt. Das geschulte Auge erkennt sofort, daß es sich dabei nur um den Teilterm an der Position ε, also um den gesamten Term t selbst handeln kann, da nur dort das Symbol $+$ auftaucht. Ebenso schnell ist zu sehen, daß eine Anwendung der ersten Regel nicht in Frage kommen kann, da der erste Teilterm von t ungleich 0 ist.

Bleibt also nur die zweite Regel, und mit $\sigma = \{x \mapsto s(0), y \mapsto s(s(s(0)))\}$ ist die gesuchte (und in diesem Fall einzig mögliche) Substitution gefunden. Dieser Vorgang wird als Pattern Matching bezeichnet.

Also kann ersetzt werden wie folgt (die weiteren Schritte folgen demselben Prinzip):

$$\begin{array}{lll} +(s(s(0)), s(s(s(0)))) & \longrightarrow_{\mathcal{R}[2,\varepsilon,\{x\mapsto s(0), y\mapsto s(s(s(0)))\}]} & s(+(s(0), s(s(s(0))))) \\ & \longrightarrow_{\mathcal{R}[2,1,\{x\mapsto 0, y\mapsto s(s(s(0)))\}]} & s(s(+(0, s(s(s(0)))))) \\ & \longrightarrow_{\mathcal{R}[1,1\cdot 1,\{y\mapsto s(s(s(0)))\}]} & s(s(s(s(s(0))))) \end{array}$$

... und siehe da, zwei und drei ist fünf. Doch sollte trotz der Einfachheit des Beispiels nicht übersehen werden, daß mit den zwei gegebenen Regeln beliebige Zahlen in einer beliebigen Schachtelung von Additionen zuverlässig berechnet werden. (Wenn man nur genug Papier bzw. Speicher hat, um alle „s“ aufzunehmen.)

Doch bevor genauer darauf eingegangen wird, was unter „berechnet“ zu verstehen ist, werden noch einige Notationen erläutert.

1. Der transitive Abschluß von $\longrightarrow_{\mathcal{R}}$ wird mit $\xrightarrow{+}_{\mathcal{R}}$ notiert.
   Transitiv ist eine Relation $R \subseteq A \times A$ falls für alle $a, b, c \in A$ gilt, aus $R(a, b)$ und $R(b, c)$ folgt $R(a, c)$. Demnach gilt $s \xrightarrow{+}_{\mathcal{R}} t$, falls der Term s, ggf. über einige Zwischenterme, auf t reduzierbar ist.
   Im Beispiel gilt etwa $+(s(s(0)), s(s(s(0)))) \xrightarrow{+}_{\mathcal{R}} s(s(+(0, s(s(s(0))))))$.
2. Der reflexive und transitive Abschluß wird mit $\xrightarrow{*}_{\mathcal{R}}$ bezeichnet.
   Reflexiv ist eine Relation $R \subseteq A \times A$ falls für alle $a \in A$ gilt, $R(a, a)$.
   $s \xrightarrow{*}_{\mathcal{R}} t$ gilt also genau dann, wenn $s \xrightarrow{+}_{\mathcal{R}} t$ oder $s = t$.
3. Gilt $s \xrightarrow{+}_{\mathcal{R}} t$, so heißt t *Nachfolger* von s.
4. Hat ein Term t keine Nachfolger, so heißt t *Normalform.*

Als Ergebnis der Beispielsberechnung wurde $s(s(s(s(s(0)))))$ angesehen. Das liegt vor allem daran, daß mit diesem Term eine Normalform für $\mathcal{R}$ vorliegt. Damit sinnvoll von einem Ergebnis gesprochen werden kann, darf es für jeden Term nur höchstens *ein* solches Ergebnis geben. Es kann für eine „vernünftige“ Berechnung nicht angehen, daß etwa $+(s(+(0,0)), +(s(0), 0)))$ verschiedene Ergebnisse liefert, je nachdem, ob man zuerst den äußeren oder einen der inneren $+$-Terme ersetzt. Dies ist bei dem Beispielssystem nicht der Fall, es läßt sich jedoch mit wenig Aufwand ein derartig „sinnloses“ System konstruieren.

Beispiel 1.10

$$\mathcal{R}' = \left(\begin{array}{lll} 1: & +(0, y) & \longrightarrow & y \\ 2: & +(s(x), y) & \longrightarrow & s(+(x, y)) \\ 3: & +(x, y) & \longrightarrow & +(x, x) \end{array} \right)$$

Diese zusätzliche Regel ist „natürlich“ falsch. Sie entspricht nicht der herkömmlichen Addition. Das äußert sich zum Beispiel bei der Berechnung des Terms $+(0, s(0))$. Dieser hat nun *zwei* Normalformen

$$s(0) \longleftarrow +(0, s(0)) \longrightarrow +(0,0) \longrightarrow 0$$

Der entscheidende Begriff hierzu ist *Konfluenz.* Als „zusammenfließend“ wird ein Termersetzungssystem bezeichnet, wenn alle auseinanderführenden Ableitungen auch wieder zu einem

Zielterm zusammengeführt werden können. Allgemein sieht das so aus:

$$\begin{array}{ccccc} & & s & & \\ & \overset{*}{\swarrow} & & \overset{*}{\searrow} & \\ t_1 & & & & t_2 \\ & \overset{*}{\searrow} & & \overset{*}{\swarrow} & \\ & & u & & \end{array}$$

und die zugehörige Definition lautet:

Definition 1.2.9: (Konfluenz)

Ein Termersetzungssystem $\mathcal{R}$ heißt konfluent, wenn für alle $s, t_1, t_2 \in \mathcal{T}_\Sigma(X)$ gilt:
Falls $s \xrightarrow{*} t_1$ und $s \xrightarrow{*} t_2$, dann sind t_1 und t_2 zusammenführbar, im Zeichen $t_1 \downarrow t_2$, das heißt, es existiert ein $u \in \mathcal{T}_\Sigma(X)$ mit $t_1 \xrightarrow{*} u$ und $t_2 \xrightarrow{*} u$. ◇

Man kann sich leicht vor Augen führen, daß in einem konfluenten System jeder Term *höchstens eine* Normalform hat. Das System aus Beispiel 1.9 ist konfluent, das System aus Beispiel 1.10 ist es nicht.

Beispiel 1.11

$$\mathcal{R}' = \begin{pmatrix} 1: & f(a) & \longrightarrow & f(f(a)) \\ 2: & f(x) & \longrightarrow & f(f(f(x))) \end{pmatrix}$$

In diesem Beispiel sind zwar alle auseinander laufenden Terme wieder zusammenführbar, jedoch hat kein Term eine Normalform. Auch wenn es nach diesem Beispiel nicht so erscheinen mag, gibt es auch sinnvolle Funktionen, die auf eine ähnliche Art und Weise arbeiten. Zum Beispiel kann es nützlich sein, eine Liste aller natürlichen Zahlen auf diese Weise zu erzeugen.

Entsprechend der Konfluenz gibt es eine Eigenschaft, die *mindestens eine* Normalform für jeden Term gewährleistet. Das ist genau dann der Fall, wenn jede Ableitung irgendwann ein Ende findet. Die dazugehörige Eigenschaft des Termersetzungssystems heißt *terminierend*.

Definition 1.2.10: (Terminierung)

Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein Termersetzungssystem.
$\mathcal{R}$ heißt terminierend, wenn es keine unendliche Ableitung $t_0 \longrightarrow t_1 \longrightarrow \ldots$ mit $i \in I\!N$ und $t_i \in \mathcal{T}_\Sigma(X)$ gibt. ◇

Das System aus Beispiel 1.9 ist terminierend, das System aus Beispiel 1.11 ist es nicht. Ein Termersetzungssystem, das konfluent *und* terminierend ist, heißt *konvergent*. In einem konvergenten System hat jeder Term *genau eine* Normalform.

Beispiel 1.12

$$\mathcal{R}_2 = \begin{pmatrix} 1: & +(0, y) & \longrightarrow & y \\ 2: & +(s(x), y) & \longrightarrow & s(+(x, y)) \\ 3: & +(x, y) & \longrightarrow & +(y, x) \end{pmatrix}$$

Mit Hinzufügen der dritten Regel (Kommutativität) ist das System nicht mehr terminierend. Dennoch ist diese Regel sinnvoll, denn jetzt ist beispielsweise das Ergebnis des Terms $+(s(s(s(s(0)))), 0)$ mit einer kürzeren Ableitung erreichbar, nämlich

$$\begin{array}{lll} +(s(s(s(s(0)))), 0) & \longrightarrow_{[3,\varepsilon,\{x \mapsto s(s(s(s(0)))), y \mapsto 0\}]} & +(0, s(s(s(s(0))))) \\ & \longrightarrow_{[1,\varepsilon,\{x \mapsto 0, y \mapsto s(s(s(s(0))))\}]} & s(s(s(s(0)))) \end{array}$$

Trotz der Nichtterminierung hat jeder Term in diesem Beispiel genau eine Normalform. An Beispiel 1.12 ist ersichtlich, wie wichtig die Auswahl der richtigen Regel oder auch der richtigen Ersetzungsstelle zum richtigen Zeitpunkt sein kann. Wählt man im letzten Beispiel grundsätzlich die dritte Regel, was immer möglich ist, so kommt man nie zu der gewünschten Normalform. Eine Funktion, die bei gegebenem Term eine oder auch parallel mehrere der möglichen Reduktionen auswählt, nennt man eine *Strategie*. Wünschenswerte, aber nicht immer erreichbare, Eigenschaften einer solchen Strategie sind etwa Vollständigkeit - wenn eine Normalform existiert, wird sie auch erreicht - oder gar Optimalität - es wird immer der kürzeste Weg zur Berechnung der Normalform gewählt.

Die verschiedenen Möglichkeiten zur Strategiebildung spielen im Rahmen dieser Arbeit keine große Rolle, weshalb auch keine Definition gegeben wird. Siehe aber [Thi94].

Auch die Eigenschaft der Terminierung ist in dieser Arbeit weniger interessant als die der Konfluenz. Das liegt zum Teil daran, daß man mit nicht terminierenden Systemen, wie schon angedeutet, durchaus sinnvolle Funktionen definieren kann, zum anderen aber daran, daß einfach überprüfbare Kriterien dafür, daß ein gegebenes System terminierend ist, die Menge der möglichen Systeme zu stark einschränken würde. Es wird in einem späteren Abschnitt auf die Terminierung zurückzukommen sein, siehe 3.4.

Für die Konfluenz hingegen gibt es einfache hinreichende Bedingungen, die gewährleisten, daß jede auseinanderführende Ableitung auch wieder zusammengeführt werden kann. Um die Arbeitsweise dieser Kriterien zu verstehen, muß zuerst ergründet werden, welche Art von Regeln die Konfluenz überhaupt gefährden können. „Konfluenzgefährdende" Regeln heißen *überlappend*, weil ihre linken Regelseiten so geformt sind, daß ihr Anwendungsbereich in ein und demselben Term eventuell gemeinsame Positionen aufweisen könnte.

Definition 1.2.11: (Überlappend)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem und σ eine Substitution.
Zwei Regeln aus P heißen überlappend mit Unifikator σ genau dann, wenn für ihre linken Regelseiten l_1 und l_2 gilt: $\sigma(l_1) = \sigma(l_2)$. $\diamond$

Bei der Unifikation der linken Regelseiten sind vorher alle Variablen so umzubenennen, daß jede Regel ihre eigenen Variablen besitzt.

In Beispiel 1.10 überlappen die Regeln

$$\begin{array}{rlcl} 2: & +(s(x), y) & \longrightarrow & s(+(x, y)) \\ 3: & +(x', y') & \longrightarrow & +(x', x') \end{array}$$

mit Unifikator $\{x' \mapsto s(x), y \mapsto y'\}$.

Für einen Algorithmus zur Unifikation von Termen siehe zum Beispiel [BA95].

Mit Hilfe des Begriffes der überlappenden Regeln läßt sich nun eine Eigenschaft formulieren, die hinreichend für die Konfluenz eines Termersetzungssystems ist, die *Orthogonalität*. Sie gewährleistet, daß ein gegebener Teilterm nur in die Anwendung einer einzigen Regel des Systems einbezogen werden kann. Die schwache Orthogonalität stellt zumindestens sicher, daß im Fall von Überlappungen das Ergebnis beider Regelanwendungen dasselbe ist.

Definition 1.2.12: ((Schwache) Orthogonalität)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein unbedingtes Termersetzungssystem.

- $\mathcal{R}$ heißt linear genau dann, wenn
  keine Variable in einer linken Regelseite mehr als einmal vorkommt.

- $\mathcal{R}$ heißt orthogonal genau dann, wenn
  $\mathcal{R}$ linear ist und es keine überlappenden Regeln in P gibt.
- $\mathcal{R}$ heißt schwach orthogonal genau dann, wenn
  $\mathcal{R}$ linear ist und für alle überlappenden Regelpaare $l_1 \longrightarrow r_1$ und $l_2 \longrightarrow r_2$ aus P mit Unifikator σ gilt: $\sigma(r_1) = \sigma(r_2)$.

$\diamond$

Alle bisher in den Beispielen verwendeten Systeme waren linear. Das System aus Beispiel 1.9 ist orthogonal, das System in Beispiel 1.11 ist weder orthogonal noch schwach orthogonal, aber trotzdem konfluent. Ein Beispiel für ein schwach orthogonales System ist

$$\begin{array}{lll} 1: & or(x, true) & \longrightarrow & true \\ 2: & or(true, y) & \longrightarrow & true \\ 3: & or(false, y) & \longrightarrow & y \\ 4: & or(x, false) & \longrightarrow & x \end{array}$$

Überlappungen sind

$$\begin{array}{lllll} 1: & \text{und} & 2: & \text{mit Unifikator} & \{x \mapsto true, & y \mapsto true\} \\ 1: & \text{und} & 3: & \text{mit Unifikator} & \{x \mapsto false, & y \mapsto true\} \\ 2: & \text{und} & 4: & \text{mit Unifikator} & \{x \mapsto true, & y \mapsto false\} \\ 3: & \text{und} & 4: & \text{mit Unifikator} & \{x \mapsto false, & y \mapsto false\} \end{array}$$

Wie man sich aber überzeugen kann, sind alle rechten Regelseiten unter dem angegebenen Unifikator jeweils gleich; in den ersten drei Fällen $true$, im letzten $false$.

Eine weitere sehr schöne Eigenschaft von Systemen ist die Konstruktorbasiertheit. Hinter diesem Begriff verbirgt sich eine Unterscheidung zwischen Funktionssymbolen, welche zum Aufbau von Datenstrukturen verwendet werden (Konstruktoren) und solchen, welche die so entstandenen Daten manipulieren (definierte Funktionen).

Definition 1.2.13: (Konstruktorbasiertes Termersetzungssystem)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein Termersetzungssystem.
Ist die Signatur zerlegbar in $\Sigma = \mathcal{D} \uplus \mathcal{C}$ und gilt für alle Regeln $\rho : f(l_1, \ldots, l_n) \longrightarrow r$ aus P: $f \in \mathcal{D}$ und $l_i \in \mathcal{T_C}(X)$ für $i \in \{1, \ldots, n\}$, so heißt $\mathcal{R}$ konstruktorbasiert.
Ein Term t, der ausschließlich aus Konstruktoren und Variablen zusammengesetzt ist, also $t \in \mathcal{T_C}(X)$, heißt *Muster*; ein Term t, der nur aus Konstruktoren besteht ($t \in \mathcal{T_C}(\emptyset)$), heißt *Konstruktorterm.* $\diamond$

Neben anderen Eigenschaften können in konstruktorbasierten Systemen Überlappungen nur unter Regeln ein und desselben definierten Funktionssymboles auftreten. Für diese Klasse existieren auch vollständige Strategien (siehe oben). Auch in dieser Arbeit wird sich unter Abschnitt 3.3 erweisen, daß die Klasse der Systeme mit dieser Eigenschaft viel einfacher zu handhaben ist.

Alle bisher betrachteten Systeme waren konstruktorbasiert, $s^{(1)}$ und $0^{(0)}$ sind typische Beispiele für Konstruktorsymbole, $+^{(2)}$ ein ebenso typisches definiertes Funktionssymbol. Überhaupt stammt die bisher verwendete Sprechweise, in der immer von Funktionen die Rede war, aus dem Bereich der konstruktorbasierten Systeme. Typisch für diesen Bereich ist es ebenfalls, nicht jede Normalform als Ergebnis einer Berechnung anzusehen, sondern nur solche, die ausschließlich aus Konstruktortermen bestehen. Bleibt ein definiertes Funktionssymbol nicht reduzierbar, so wird dies als Fehler in der Funktionsdefinition empfunden.

Außerhalb dieses Bereiches ist man eher daran interessiert, abstrakte *Eigenschaften* von Funktionen zu spezifizieren, als die tatsächlichen Funktionen selbst zu definieren. Anwendungen derartiger Systeme finden sich etwa in der Spezifikation von abstrakten Datentypen oder in der Gleichungstheorie. In beiden ist es auch oft sinnvoll, statt der Reduktion $\longrightarrow$ die Gleichheit $=$, also eine Ersetzungsmöglichkeit in beide Richtungen, zu betrachten.

Beispiel 1.13
$\Sigma = \{\varepsilon^{(0)}, pop^{(1)}, top^{(1)}, push^{(2)}\}$ und $X = \{x, s\}$

$$\mathcal{R}_{Stack} = \begin{pmatrix} 1: & pop(push(s,x)) & = & s \\ 2: & push(pop(s), top(s)) & = & s \\ 3: & top(push(s,x)) & = & x \end{pmatrix}$$

Beispiel 1.14

$$\mathcal{R}_3 = \begin{pmatrix} 1: & +(0,y) & = & y \\ 2: & +(s(x),y) & = & s(+(x,y)) \\ 3: & +(x,y) & = & +(y,x) \\ 4: & +(+(x,y),z) & = & +(x,+(y,z)) \end{pmatrix}$$

Die vierte Regel (Assoziativität) ist nicht konstruktorbasiert.

In dieser Arbeit werden hauptsächlich konstruktorbasierte Systeme betrachtet, weil es in der Praxis eine inzwischen weit verbreitete Anwendung für schwach orthogonale, konstruktorbasierte Termersetzungssysteme gibt: *Die Funktionale Programmierung.*

Funktionale Programmiersprachen sind ein fortschrittliches Konzept, das dem Bereich der deklarativen Programmierung zugerechnet wird.

1.3 Deklarative Programmierung

In den letzten Jahren hat die Bedeutung der deklarativen Programmierung immer weiter zugenommen. Der wichtige Grundgedanke dieser Art von Programmierung ist es, ein Problem zu lösen, indem man es in einer präzisen Sprache formuliert. Dieser Ansatz steht im Gegensatz zu der sogenannten imperativen Programmierung, in der eine Lösung immer die Form einer Folge von Befehlen annimmt.

> Ein Problem ist so gut wie gelöst, kann man es nur genau genug formulieren.

Ohne hier genauer darauf einzugehen, hat sich in einigen Anwendungsgebieten gezeigt, daß der deklarativen Programmierung durchaus eine Existenzberechtigung zuzusprechen ist. Sie ist dem Gedanken näher, erlaubt daher oft eine schnellere Umsetzung und Überprüfung von Ideen, als es herkömmliche Programmiersprachen bieten. Auf der anderen Seite sind die imperativen Programmiersprachen, wenn nicht dem Gedanken, so doch der Maschine näher, so daß eine imperative Lösung, ist sie einmal halbwegs fehlerfrei erstellt, oft effizienter mit den Maschinenresourcen hauszuhalten weiß.

Bekanntlich teilt sich das Feld der deklarativen Programmierung in die beiden Hauptrichtungen der logischen und der funktionalen Programmierung. Während der getrennten Erforschung dieser beiden Ansätze wurde immer wieder die Meinung vertreten, daß es möglich sein müsse, diese beiden Ansätze zusammenzubringen, um so die Vorteile beider zu vereinigen. Normalerweise ist die Verbindung zweier verschiedener Paradigmen mit gewissen Einbußen verbunden; ein Teil der Vorteile beider Prinzipien geht verloren. Im Bereich der funktional-logischen Programmierung besteht allerdings berechtigte Hoffnung, diesem Schicksal zu entgehen. Dies liegt in der Hauptsache daran, daß die beiden zu verbindenden Ansätze von Beginn an als verwandt bezeichnet werden können. Sie stellen jeweils einen Teilaspekt eines zugrundeliegenden Paradigmas dar, nämlich das der deklarativen Programmierung (siehe dazu [Llo94]). In letzter Zeit wurden immer weitere Bestrebungen unternommen, diese Paradigmen in einem einzigen zu vereinheitlichen, dem Paradigma einer deklarativen Programmierung (siehe [Han94]).

In einem solchen vereinheitlichen Paradigma werden statt reiner Ersetzungsstrategien Mechanismen gefordert, die zusätzlich in der Lage sind, Anfragen im Stil logischer Programmierung zu beantworten. Zu dieser Forderung sind mit *Residuation* und *Narrowing* zufriedenstellende Lösungen gefunden. Beide Verfahren lassen sich verbinden, wie in [Han97] dargestellt wird.

Will man einen logischen Programmierstil unterstützen, braucht es aber auch eine andere Klasse von Termersetzungssystemen. Ein logisches Programm basiert auf Hornklauseln:

$$p \Leftarrow q_1, \ldots, q_n$$

Diese lassen sich als Regeln eines *bedingten* Termersetzungssystems betrachten

$$p \longrightarrow true\ if\ q_1 \wedge \ldots \wedge q_n \stackrel{*}{\longrightarrow} true$$

Es existieren funktional-logische Sprachen, in denen derartige Regeln zulässig sind, zum Beispiel Babel (siehe [KLMNRA89]) und Curry ([Han97]). Problematisch ist aber, daß zum einen teilweise noch der theoretische Hintergrund fehlt, zum anderen die Effizienz gegenüber den unbedingten Programmteilen zu wünschen übrig läßt. Beides liegt daran, daß in der Vergangenheit die bedingten Systeme bei weitem weniger Aufmerksamkeit erhalten haben.

Von einer weiteren Erforschung, wie sie in dieser Arbeit geschieht, sind demnach weitere Fortschritte für derartige Sprachen, wie auch für den gesamten Bereich der bedingten Termersetzungssysteme überhaupt, zu erwarten.

1.4 Bedingte Termersetzungssysteme

Das folgende Beispiel soll, einführend in die bedingten Termersetzungssysteme, die größte Zahl aus einer Folge von natürlichen Zahlen ermitteln. Dazu wird die Funktion max definiert.

Beispiel 1.15

$$\mathcal{R} = \left(\begin{array}{lll} 1: & \geq(y,0) & \longrightarrow & true \\ 2: & \geq(s(x),s(y)) & \longrightarrow & \geq(x,y)) \\ 3: & \max(w) & \longrightarrow & m(w,0) \\ 4: & m(\varepsilon,x) & \longrightarrow & x \\ 5: & m(x\cdot w,y) & \longrightarrow & m(w,x)\ if\ \geq(x,y) \stackrel{*}{\longrightarrow} true \\ 6: & m(x\cdot w,y) & \longrightarrow & m(w,y)\ if\ \geq(y,x) \stackrel{*}{\longrightarrow} true \end{array} \right)$$

Die letzten beiden Regeln haben eine bisher unbekannte Form. Es handelt sich um *bedingte* Regeln.

Definition 1.4.1: (Bedingte Termersetzungssysteme)
Ein bedingtes Termersetzungssystem $\mathcal{R}$ ist ein Tripel $\langle \Sigma, X, \mathrm{P} \rangle$, wobei Σ eine Signatur, X eine abzählbare Menge von Variablen und P eine endliche Menge von Regeln der Form

$$\rho \ : l \longrightarrow r \ if \ u \xrightarrow{*} v$$

enthält mit $l, r, u, v \in \mathcal{T}_\Sigma(X)$ und $var(r) \cup var(u) \subseteq var(l)$ sowie v Grundterm und Normalform ist.
Außerdem kann P auch noch eine endliche Menge von unbedingten Termersetzungsregeln enthalten. ◇

Wie bei den unbedingten Systemen besteht die Definition der Arbeitsweise in der Erklärung der Reduktionsschritte. Ein bedingter Reduktionsschritt kann durchgeführt werden, wenn, wie bei unbedingten Schritten, die erforderliche Substitution existiert und *zusätzlich* die im *if*-Teil geforderte Ableitung.

Definition 1.4.2: (Bedingte Reduktionsschritte)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem, $s, t, u \in \mathcal{T}_\Sigma(X)$, p eine Position in s, $\rho : l \longrightarrow r \ if \ u \xrightarrow{*}_\mathcal{R} v$ eine Regel aus P und σ eine Substitution.
Die Relation der bedingten Reduktionsschritte $\longrightarrow_\mathcal{R} \subseteq \mathcal{T}_\Sigma(X) \times \mathcal{T}_\Sigma(X)$ ist definiert durch

$$s \longrightarrow_{\mathcal{R}[\rho,p,\sigma]} t \text{ genau dann, wenn } s|_p = \sigma(l) \text{ und } t = s[p \longleftarrow \sigma(r)] \text{ und } \sigma(u) \xrightarrow{*}_\mathcal{R} v$$

◇

Beispiel 1.16
Als Ableitung im System des oben angegebenen Beispieles soll die größte Zahl aus der Folge $2 \cdot 4 \cdot 0$ herausgefunden werden.

$$\begin{array}{lll}
max(s(s(0)) \cdot s(s(s(s(0)))) \cdot 0) & \longrightarrow_{[3]} & m(s(s(0)) \cdot s(s(s(s(0)))) \cdot 0, 0) \\
\text{Da } \geq (s(s(0)), 0) \longrightarrow_{[1]} true & \longrightarrow_{[5]} & m(s(s(s(s(0)))) \cdot 0, s(s(0))) \\
\text{Da } \geq (s(s(s(s(0)))), s(s(0))) \xrightarrow{*}_{[2,2,1]} true & \longrightarrow_{[5]} & m(0, s(s(s(s(0))))) \\
\text{Da } \geq (s(s(s(s(0)))), 0) \longrightarrow_{[1]} true & \longrightarrow_{[6]} & m(\varepsilon, s(s(s(s(0))))) \\
 & \longrightarrow_{[4]} & s(s(s(s(0))))
\end{array}$$

Es sind noch verschiedene Verallgemeinerungen der Definition für bedingte Termersetzungssysteme und bedingte Reduktionsschritte denkbar. So werden in der Literatur etwa Systeme betrachtet, die statt der $\xrightarrow{*}$-Relation im *if*-Teil der Regeln Äquivalenz (=), Zusammenführbarkeit zweier Terme ($u \downarrow v$) (siehe Definition 1.2.9) oder Nichtgrundterme als Ziel einer Ableitung ($u \xrightarrow{*} v$) haben. Diese verallgemeinerten Systeme haben andere Eigenschaften, als die in dieser Arbeit betrachteten [BK86].

Andere Verallgemeinerungen bewirken allerdings keinerlei zusätzliche Eigenschaften. So wird nur aus Gründen der einfacheren Darstellung von nur *einer* Bedingung $u \xrightarrow{*} v$ ausgegangen.

Die meisten der in Abschnitt 1.2 gemachten Definitionen sind auch für die bedingten Termersetzungssysteme anwendbar. Der Begriff der konstruktorbasierten Systeme wird hier eine Ergänzung dahingehend erfahren, daß der Zielterm v einer jeden Bedingung ein reiner Konstruktorterm sein muß. Die (schwache) Orthogonalität wird allerdings erst in Abschnitt 4.1 übertragen, da es dort speziell um diese Eigenschaft für bedingte Systeme geht.

Definition 1.4.3: (Konstruktorbasiertes Termersetzungssystem, bedingter Fall)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem.
Ist die Signatur zerlegbar in $\Sigma = \mathcal{D} \uplus \mathcal{C}$ und gilt für alle Regeln ρ aus P der Form
$\rho : f(l_1, \ldots, l_n) \longrightarrow r \ if \ u \stackrel{*}{\longrightarrow} v$:
$f \in \mathcal{D}$ und $l_i \in \mathcal{T}_{\mathcal{C}}(X)$ für $i \in \{1, \ldots, n\}$ sowie $v \in \mathcal{T}_{\mathcal{C}}(\emptyset)$, so heißt $\mathcal{R}$ konstruktorbasiert. $\diamond$

Das System aus Beispiel 1.15 war bereits konstruktorbasiert.

1.5 Korrektheit einer Transformation

Wie schon im vorangegangenen Abschnitt erwähnt, soll es in dieser Arbeit darum gehen, bedingte Termersetzungssysteme in unbedingte zu transformieren. In dem kurzen historischen Abriß 1.1 wurde bereits erwähnt, daß man mit unbedingten Systemen bereits „alles“ berechnen kann. Das ist auch der Grund, warum eine solche Transformation rein grundsätzlich möglich sein muß, denn mit bedingten Termersetzungssystemen läßt sich nicht mehr (aber auch nicht weniger) berechnen, siehe dazu [BT87]. In letzterer Veröffentlichung hat Bergstra jedoch bewiesen, daß bedingte Termersetzungssysteme allerdings ein mehr, nämlich eine „kompaktere Darstellung“, bieten. Kompakter bedeutet in diesem Zusammenhang, daß es keine Transformation bedingt zu unbedingt geben kann, bei der Ausgangs- und Zielsystem äquivalent sind, die Signatur aber unverändert bleiben kann. Jede Transformation bedingter Termersetzungssysteme in unbedingte muß die Signatur um weitere Zeichen erweitern oder zumindest die Stelligkeit der gegebenen Symbole erhöhen.

Daran kann man erkennen, daß mit einer solchen Transformation ein Preis zu entrichten ist, es geht dabei etwas an Ausdrucksstärke verloren. Dennoch ist die Auffindung einer geeigneten Transformation ein lohnendes Unterfangen. Erst wenn man eine solche Transformation gefunden hat, kann man sich fragen, *was genau* die Ausdrucksstärke des bedingten Systems ausmacht. Auch sind die Ergebnisse der Forschung, die sich bei weitem stärker mit unbedingten Systemen beschäftigte, leichter auf den Bereich der unbedingten Systeme übertragbar. Schließlich kann auch jede funktional-logische Sprache davon profitieren, daß die Mechanismen, die für unbedingte Systeme optimiert wurden, und meist mit einer einfachen Hilfskonstruktion für die unbedingten Erweiterungen versehen wurden, eventuell ohne Veränderungen auf das Zielsystem der Transformation angewandt werden können. Alle Effizienzvorteile, die im unbedingten Fall zum Beispiel durch Parallelisierungen entstanden sind, könnten mit einer geeigneten Transformation auch auf die bedingten Erweiterungen angewandt werden.

Wann aber kann man nun von einer Äquivalenz zwischen einem bedingten und einem unbedingten Termersetzungssystem sprechen? Wann ist eine gegebene Transformation korrekt?

Wie erwähnt, kann die Signatur der beiden Termersetzungssysteme nicht gleich sein. Damit ist natürlich auch die Menge der Terme bezüglich dieser Signaturen unterschiedlich. Eine Äquivalenz zwischen den beiden Termersetzungssystemen erfordert eine Zuordnung zwischen den Termmengen.

Definition 1.5.1: (Starttermzuordnung)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes, $\mathcal{R}' = \langle \Sigma', X, \mathrm{P}' \rangle$ ein unbedingtes Termersetzungssystem, wobei Σ' eine Erweiterung von Σ ist.
Eine Abbildung $H : \mathcal{T}_{\Sigma}(X) \longrightarrow \mathcal{T}_{\Sigma'}(X)$ heißt Starttermzuordnung zwischen $\mathcal{R}$ und $\mathcal{R}'$, wenn H ein injektiver Homomorphismus ist. $\diamond$

Da die Zielsignatur eine Erweiterung der Ausgangssignatur ist, ist das Term-Universum des Zielsystems notwendigerweise größer. Daher sind auch unter Umständen nicht alle Terme an der Wiedergabe des Ausgangssystems beteiligt. Die Menge aller möglichen Terme wird daher auf die von Starttermen aus erreichbaren eingeschränkt.

Definition 1.5.2: (Menge der sinntragenden Terme)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes, $\mathcal{R}' = \langle \Sigma', X, \mathrm{P}' \rangle$ ein unbedingtes Termersetzungssystem und H eine Starttermzuordnung zwischen $\mathcal{R}$ und $\mathcal{R}'$.
Dann ist Menge $T(\mathcal{R}, H) \subseteq \mathcal{T}_{\Sigma'}(X)$ definiert als kleinste Menge mit

- Aus $t \in \mathcal{T}_{\Sigma}(X)$ folgt $H(t) \in T(\mathcal{R}, H)$
- Aus $s \in T(\mathcal{R}, H)$ und $s \xrightarrow{*}_{\mathcal{R}'} t$ folgt $t \in T(\mathcal{R}, H)$

$\diamond$

Wären in der Definition 1.4.1 für bedingte Termersetzungssysteme auch Extravariablen zugelassen worden, so hätte bei der Ableitung im zweiten Punkt noch gefordert werden müssen, daß die Menge der für diese Variablen einsetzbaren Terme eine Teilmenge der sinntragenden Terme ist.

Auch in der Rückrichtung muß eine Zuordnung existieren, die mit der Starttermzuordnung in enger Beziehung steht.

Definition 1.5.3: (Endtermzuordnung)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes, $\mathcal{R}' = \langle \Sigma', X, \mathrm{P}' \rangle$ ein unbedingtes Termersetzungssystem sowie H eine Starttermzuordnung zwischen $\mathcal{R}$ und $\mathcal{R}'$.
Eine surjektive, eventuell partielle Abbildung $G : \mathcal{T}_{\Sigma'}(X) \dashrightarrow \mathcal{T}_{\Sigma}(X)$ heißt Endtermzuordnung, wenn $T(\mathcal{R}, H)$ Teil des Definitionsbereiches von G ist und für alle $t \in \mathcal{T}_{\Sigma}(X)$ gilt $G(H(t)) = t$. $\diamond$

Auch wenn aufgrund der notwendigen Signaturerweiterung die beiden Termzuordnungen sowie die Menge der sinntragenden Terme ebenfalls notwendig sind, werden sie in allen Veröffentlichungen bis auf eine nicht angegeben. Implizit ist aber zumindest die Starttermzuordnung immer vorhanden. In solchen Fällen wurde darauf geachtet, daß das Zielsystem der Transformation in sofern dieselbe Form bekommt, als die Starttermzuordnung auf der Schnittmenge der beiden Termuniversen die Identität ist. Falls das Ergebnissystem nicht korrekt ist, existiert gar keine Endtermzuordnung. Auch werden andere, nicht sinntragende Terme, dann „geleugnet", also gar nicht erst betrachtet.

Definition 1.5.4: (Vollständigkeit und Korrektheit einer Entsprechung)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes, $\mathcal{R}' = \langle \Sigma', X, \mathrm{P}' \rangle$ ein unbedingtes Termersetzungssystem, H eine Starttermzuordnung, G eine Endtermzuordnung.
Dann gilt:

- $\mathcal{R}'$ entspricht $\mathcal{R}$ vollständig, falls für alle $s, t \in \mathcal{T}_{\Sigma}(X)$ gilt:
  Aus $s \xrightarrow{*}_{\mathcal{R}} t$ folgt die Existenz einer Ableitung $H(s) \xrightarrow{*}_{\mathcal{R}'} H(t)$
- $\mathcal{R}'$ entspricht $\mathcal{R}$ korrekt, falls für alle $s \in \mathcal{T}_{\Sigma}(X)$ und alle $t \in \mathcal{T}_{\Sigma'}(X)$ gilt:
  Aus $H(s) \xrightarrow{*}_{\mathcal{R}'} t$ folgt die Existenz einer Ableitung $G(H(s)) \xrightarrow{*}_{\mathcal{R}} G(t)$

Der sinntragende Teil, das heißt $\xrightarrow{*}_{\mathcal{R}'}$ beschnitten auf $T(\mathcal{R}, H)$, heißt *Entsprechung von $\mathcal{R}$ in $\mathcal{R}'$*. $\diamond$

Die Bedingung der Vollständigkeit ist etwas härter als nötig (vergleiche Abschnitt 3.5). Es würde auch reichen

$$s \xrightarrow{*}_{\mathcal{R}} t \Rightarrow H(s) \xrightarrow{*}_{\mathcal{R}'} t' \text{ mit } G(t') = t$$

zu fordern.

Ziel einer Transformation bedingter Termersetzungssysteme in unbedingte muß es sein, vollständige und korrekte Entsprechung des Zielsystems mit dem Ausgangssystem zu gewährleisten. Die Begriffe lassen sich also direkt auf eine ganze Transformation übertragen.

Definition 1.5.5: (Vollständigkeit und Korrektheit einer Transformation)

Sei K eine Klasse von bedingten Termersetzungssystemen, U die Klasse der unbedingten Termersetzungssysteme.
Eine Transformation $T : K \longrightarrow U$ heißt

- Vollständig bezüglich K genau dann, wenn
  für jedes $\mathcal{R} \in K$ gilt, daß $T(\mathcal{R})$ dem System $\mathcal{R}$ vollständig entspricht.
- Korrekt bezüglich K genau dann, wenn
  für jedes $\mathcal{R} \in K$ gilt, daß $T(\mathcal{R})$ dem System $\mathcal{R}$ korrekt entspricht.

$\diamond$

Zusätzlich zu Vollständigkeit und Korrektheit ist aber auch von Interesse, ob, bei gegebener Konfluenz des Ausgangssystems, auch das Zielsystem konfluent ist. Gilt dies grundsätzlich für eine Transformation, so wird diese Eigenschaft mit Konfluenzerhaltung bezeichnet.

Definition 1.5.6: (Konfluenzerhaltend)

Sei K eine Klasse von bedingten Termersetzungssystemen, U die Klasse der unbedingten Termersetzungssysteme und T eine Transformation $T : K \longrightarrow U$, die korrekt und vollständig bezüglich K ist.
T heißt konfluenzerhaltend, wenn für alle konfluenten $\mathcal{R} \in K$ die Entsprechung von $\mathcal{R}$ in $T(\mathcal{R})$ ebenfalls konfluent ist. $\diamond$

Anzumerken ist, daß manche der definierten Eigenschaften nicht unbedingt im allgemeinen gelten müssen. Es wäre auch vorstellbar, daß sie nur im Zusammenwirken mit einer bestimmten Auswertungsstrategie vorhanden sind. In Abschnitt 3.4 wird dann auch auf die Eigenschaft der Terminierung eingegangen.

Eines der Ziele dieser Arbeit ist es, eine korrekte, konfluenzerhaltende Transformation für ein möglichst großes K zu finden. In Abschnitt 3.4 gelingt es tatsächlich, eine solche Transformation für die Klasse der gesamten bedingten Termersetzungssysteme anzugeben.

Kapitel 2

Verschiedene Verfahren

In der Vergangenheit sind bereits verschiedene Verfahren zur Transformation bedingter in unbedingte Termersetzungssysteme vorgeschlagen worden. Diese sollen im folgenden daraufhin untersucht werden, in bezug auf welche Klasse von Systemen sie korrekt, vollständig und konfluenzerhaltend sind. Am Ende wird sich dann ein „Verfahren der Wahl“ herauskristallisiert haben, welches dann im nächsten Kapitel unter verschiedenen Blickwinkeln tiefer untersucht und erweitert wird.

Die verschiedenen hier untersuchten Verfahren sind den angegebenen Veröffentlichungen entnommen. Sie werden mit einer einheitlichen Syntax versehen und auf ein bestimmtes Beispiel angewendet. Dieses Referenzbeispiel wird dann jeweils mit kleinen Veränderungen versehen, um auf die speziellen Eigenschaften des jeweiligen Ansatzes aufmerksam machen zu können. Ich denke, daß diese Vorgehensweise für das Verständnis der Arbeit sehr vorteilhaft ist. Auch wenn es nicht sehr abwechslungsreich erscheinen mag, so ist es doch wesentlich einfacher, ein bekanntes, einfaches Beispiel in verschiedenem Kontext zu sehen, als jedesmal neue Regeln zu verinnerlichen.

2.1 Bergstra und Klop

Das erste der vorgestellten Verfahren wurde von J. A. Bergstra und J. W. Klop in [BK86] veröffentlicht. Grundidee der Transformation ist es, die Bedingung mit einem neuen Funktionssymbol zu „ummanteln“, welches solange unersetzt bleibt, bis die geforderte Grundnormalform erreicht ist. Zu diesem Zweck wird die Signatur für jede bedingte Regel mit Nummer i um ein neues Symbol δ_i erweitert. Jedes dieser Symbole bekommt eine einzelne definierende Regel; zusätzlich gibt es noch ein weiteres Symbol I, für das ebenfalls eine Regel existiert.

Definition 2.1.1: (Transformation $\mathcal{BK}$ von Bergstra und Klop)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem und $\rho_1, \ldots, \rho_n$ die bedingten Regeln aus P.
Dann entsteht durch die Transformation von Bergstra und Klop das unbedingte Termersetzungssystem $\mathcal{R}_{\mathcal{BK}} = \langle \Sigma_{\mathcal{BK}}, X, \mathrm{P}_{\mathcal{BK}} \rangle$, das wie folgt definiert ist:

$$\begin{aligned} \Sigma_{\mathcal{BK}} &= \Sigma \cup \{\delta_1^{(1)}, \ldots, \delta_n^{(1)}\} \cup \{I^{(1)}\} \\ \mathrm{P}_{\mathcal{BK}} &= (\mathrm{P} \setminus \{\rho_1, \ldots, \rho_n\}) \cup \{\rho_1', \rho_1'', \ldots, \rho_n', \rho_n''\} \cup \{Ix \longrightarrow x\} \end{aligned}$$

Dabei wird jede bedingte Regel $\rho_i :\ l_i \longrightarrow r_i\ if\ u_i \stackrel{*}{\longrightarrow} v_i$ durch zwei unbedingte Regeln der Form

$$\rho_i' \quad : \quad l_i \longrightarrow \delta_i(u_i) r_i$$

$$\rho_i'' \quad : \quad \delta_i(v_i) \longrightarrow I$$

ersetzt. $\diamond$

Nun wird die Transformation zunächst einmal an einem Beispiel durchgeführt, welches, wie schon weiter oben erwähnt, mit leichten Modifikationen immer wieder Verwendung finden wird.

Beispiel 2.1

$$\mathcal{R} = \left(\begin{array}{lllll} head(x:xs) & \longrightarrow & x & nil([]) & \longrightarrow \; true \\ tail(x:xs) & \longrightarrow & xs & nil(x:xs) & \longrightarrow \; false \\ \multicolumn{5}{l}{append(xs,ys) \longrightarrow head(xs) : append(tail(xs), ys) \; if \; nil(xs) \xrightarrow{*} false} \end{array} \right)$$

Bei den in diesem Beispiel verwendeten Funktionen handelt es sich um „prominente" Vertreter aus der funktionalen Programmierung. Sie verarbeiten Listen unbestimmter Länge. Dabei ist zum Beispiel die vierelementige Liste $[1,3,5,4]$ als abkürzende Schreibweise für $1:(3:(5:(4:[])))$ zu betrachten. Der nullstellige Konstruktor $[]$ ist also eine Liste, und falls xs eine Liste ist, so auch $x:xs$.

Dann ermittelt $head([1,3,5,4])$ das erste Element einer Liste, also in diesem Fall 1.
$tail([1,3,5,4])$ reduziert entsprechend auf den Rest der Liste ohne Kopf, hier $[3,5,4]$.
$nil([1,3,5,4])$ ist der Test, ob eine gegebene Liste leer ist; in diesem Beispiel $false$
$append([1,3,5,4],[7,6])$ soll schließlich zwei Listen aneinanderhängen, hier also $[1,3,5,4,7,6]$ erzeugen.

Verwundern mag allerdings, daß *append* in diesem Beispiel nur eine Regel aufweist, und damit im beabsichtigten Sinn unvollständig ist. Im weiteren Verlauf wird eine weitere Regel ähnlich $append([],ys) \longrightarrow ys$ hinzugefügt werden, um das Beispiel sinnvoll zu vervollständigen. Diese zusätzliche Regel würde aber in diesem Fall wegen der Musterüberlappung aus dem Bereich der „erlaubten" Termersetzungssysteme herausführen, wie weiter unten ausgeführt wird.

In $\mathcal{R}$ gibt es nun die folgende Ableitung, die ebenfalls im weiteren als Referenz dienen wird:

$$\begin{array}{rcl} \underline{append([a],[])} & \longrightarrow & \underline{head([a])} : append(tail([a]),[]) \\ & \longrightarrow & a : append(\underline{tail([a])},[]) \\ & \longrightarrow & a : append([],[]) \end{array}$$

Auf Angabe der Regelnummern und Substitutionen wird aus Gründen der Übersicht verzichtet. Die jeweilige Ersetzungsposition wird aus den Unterstreichungen des jeweiligen Teilterms ersichtlich. Wichtig ist, daß zwar die Funktionen *head* und *tail* in anderer Reihenfolge ausgewertet werden könnten, ansonsten aber keinerlei Variationsmöglichkeit besteht. Insbesondere terminiert jede Ableitung, die von $append([a],[])$ ausgeht.

Das nach Bergstra und Klop transformierte System hat die Form

$$\mathcal{R}_{\mathcal{BK}} = \left(\begin{array}{lllll} head(x:xs) & \longrightarrow & x & nil([]) & \longrightarrow \; true \\ tail(x:xs) & \longrightarrow & xs & nil(x:xs) & \longrightarrow \; false \\ & & & & \\ append(xs,ys) & \longrightarrow & \multicolumn{3}{l}{\delta_1(nil(xs))head(xs) : append(tail(xs),ys)} \\ \delta_1(false) & \longrightarrow & I & & \\ Ix & \longrightarrow & x & & \end{array} \right)$$

Die entsprechende Ableitung in $\mathcal{R}_{\mathcal{BK}}$ sieht nun so aus:

$$\begin{array}{rcl}
\underline{append([a],[])} & \longrightarrow & \delta_1(\underline{nil([a])})head([a]) : append(tail([a]),[]) \\
& \longrightarrow & \underline{\delta_1(false)}head([a]) : append(tail([a]),[]) \\
& \longrightarrow & \underline{I\ head([a]) : append(tail([a]),[])} \\
& \longrightarrow & \underline{head([a])} : append(tail([a]),[]) \\
& \longrightarrow & a : append(\underline{tail([a])},[]) \\
& \longrightarrow & a : append([],[])
\end{array}$$

Es sind, ausgehend von $append([a],[])$, noch weitere Ableitungen möglich, worauf aber erst weiter unten eingegangen wird.

Dieses Verfahren ist schon laut der Veröffentlichung [BK86] nur auf einer sehr kleinen Klasse von Termersetzungssystemen vollständig. So dürfen die Muster des unbedingten Teils des Ausgangssystems $\mathcal{R}$ nicht überlappen und müssen linear sein.

Satz 2.1: (Vollständigkeit der Transformation $\mathcal{BK}$)

Mit der Identität als Starttermzuordnung ist $\mathcal{BK}$ vollständig bezüglich der Klasse der bedingten Termersetzungssysteme, deren unbedingter Teil orthogonal ist. ◇

Unter „unbedingtem Teil“ des bedingten Termersetzungssystems ist hierbei der $l \longrightarrow r$-Anteil aller Regeln zu verstehen. Zu den Begriffen linear und orthogonal siehe Definition 1.2.12 Diese Bedingungen treffen auf das Beispiel zu.

In [BK86] wird nur die Vollständigkeit, also die Entsprechung einer Ableitung in $\mathcal{R}_{\mathcal{BK}}$ für jede Ableitung im Ausgangssystem $\mathcal{R}$, gezeigt. Es wird jedoch darauf verwiesen, daß Korrektheit ebenfalls vorliege. *Dies ist aber falsch,* wie Dershowitz und Okada in [DO90] zeigten. Ein Gegenbeispiel läßt sich auch schon an einer weiteren Ableitung des Referenzbeispieles zeigen.

Abgesehen von der oben gezeigten gibt es noch die Ableitung

$$\begin{array}{ll}
& \underline{append([a],[])} \\
\longrightarrow & \delta_1(nil([a]))head([a]) : \underline{append(tail([a]),[])} \\
\longrightarrow & \delta_1(nil([a]))head([a]) : \delta_1(nil(tail([a])))head(tail([a])) : \underline{append(tail(tail([a])),[])} \\
\longrightarrow & \delta_1(nil([a]))head([a]) : \delta_1(nil(tail([a])))head(tail([a])) : \delta_1(nil(tail(tail([a])))) \\
& head(tail(tail([a]))) : \underline{append(tail(tail(tail([a]))),[])} \\
& \vdots
\end{array}$$

Diese Ableitung ist offensichtlich problematisch. Nicht nur, daß sie nicht terminiert, die Folge von verschachtelten $tail(x)$-Termen kann nicht im Sinne der Funktionsdefinition sein. Im Ausgangssystem $\mathcal{R}$ existiert keine Entsprechung für diese Ableitung, woraus folgt, daß $\mathcal{BK}$ auch auf der sehr restriktiven Klasse der orthogonalen (und sogar terminierenden) Termersetzungssysteme nicht korrekt ist.

Das Problem liegt in der Form der neu eingeführten Regeln $\rho' : l \longrightarrow \delta_i(u)r$. In diesen wird die rechte Regelseite r bereits mit eingeführt, bevor die Bedingung u ausgewertet wurde. In einem bedingten Termersetzungssystem erscheint der Term r erst nach der entsprechenden Auswertung und kann somit auch nicht vorzeitig reduziert werden. Anscheinend schwebte Bergstra und Klop eine Art von links-rechts-Auswertung vor, mit der eine Korrektheit möglich wäre.

Um Korrektheit oder gar Konfluenzerhaltung zu erzielen, wenn möglich auf einer größeren Klasse von Systemen, bedarf es also anderer Verfahren.

2.2 Giovanetti und Moiso

Das Problem der Transformation von Bergstra und Klop wird umgangen, indem die rechte Regelseite einer bedingten Regel erst nach Auswertung der Bedingung eingeführt wird. Auf Basis dieser Grundidee ist das Verfahren von Giovanetti und Moiso [GM87] verständlich. Es handelt sich bei diesem um eine Weiterentwicklung des Verfahrens von Dershowitz und Plaisted [DP86]. In letzterem werden nur Bedingungen mit $if - then - else$ Fallunterscheidungen transformiert; das Verfahren von Giovanetti und Moiso arbeitet auf einer größeren Klasse von Bedingungen.

Zunächst werden die Regeln in verschiedene Klassen unterteilt, denen jeweils ein neues Funktionssymbol if (für induced function symbol) zugeordnet wird. Das transformierte System enthält dann statt der bedingten Regeln unbedingte Regeln für die if-Symbole, und zwar eine mehr als die entsprechende Klasse Regeln enthielt.

Definition 2.2.1: (Transformation $\mathcal{GM}$ von Giovanetti und Moiso)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem und $\rho_1, \ldots, \rho_n$ die bedingten Regeln aus P, wobei die einzelnen ρ_i von der Form $\rho_i : \; l_i \longrightarrow r_i \; if \; u_i \xrightarrow{*} v_i$ sind und $var(l_i) = \{x_{i_1}, \ldots, x_{i_j}\}$ ist.

$\rho_1, \ldots, \rho_n$ werden in möglichst wenige Mengen $C_1, \ldots, C_m$ unterteilt, so daß für alle $\rho_i, \rho_j \in C_k$ gilt: $(l_i, u_i) \cong (l_j, u_j)$, wobei $\cong$ syntaktische Gleichheit bis auf Variablenumbenennung bedeutet. Diese eventuelle Umbenennung wird in der gesamten Klasse C_i einheitlich durchgeführt.

Dann entsteht durch die Transformation von Giovanetti und Moiso das unbedingte Termersetzungssystem $\mathcal{R}_{\mathcal{GM}} = \langle \Sigma_{\mathcal{GM}}, X, \mathrm{P}_{\mathcal{GM}} \rangle$, das wie folgt definiert ist:

$$\begin{aligned} \Sigma_{\mathcal{GM}} &= \Sigma \cup \{if_1, \ldots, if_m\} \\ \mathrm{P}_{\mathcal{GM}} &= (\mathrm{P} \setminus \{\rho_1, \ldots, \rho_n\}) \cup \{\rho'_1, \ldots, \rho'_m\} \cup \{\rho''_1, \ldots, \rho''_n\} \end{aligned}$$

Ist C_k die Menge, welche die Regel ρ_i enthält, so ist if_k ein neues Funktionssymbol, welches die Stelligkeit $|\{x_{i_1}, \ldots, x_{i_j}\}| + 1$ hat. Die unbedingten Regeln ρ'_k und ρ''_i haben dann die Form

$$\begin{aligned} \rho'_k &: \; l_i \longrightarrow if_k(x_{i_1}, \ldots, x_{i_j}, u_i) \\ \rho''_i &: \; if_k(x_{i_1}, \ldots, x_{i_j}, v_i) \longrightarrow r_i \end{aligned}$$

◇

Die Transformation wird an dem jetzt vervollständigten Referenzbeispiel verdeutlicht:

Beispiel 2.2

$$\mathcal{R} = \left(\begin{array}{lcllcl} head(x:xs) & \longrightarrow & x & nil([]) & \longrightarrow & true \\ tail(x:xs) & \longrightarrow & xs & nil(x:xs) & \longrightarrow & false \\ append(xs,ys) & \longrightarrow & \multicolumn{4}{l}{ys \; if \; nil(xs) \xrightarrow{*} true} \\ append(xs,ys) & \longrightarrow & \multicolumn{4}{l}{head(xs) : append(tail(xs), ys) \; if \; nil(xs) \xrightarrow{*} false} \end{array} \right)$$

Die Referenzableitung wird entsprechend verlängert zu

$$\begin{aligned} \underline{append([a],[])} &\longrightarrow \underline{head([a])} : append(tail([a]),[]) \\ &\longrightarrow a : append(\underline{tail([a])},[]) \\ &\longrightarrow a : \underline{append([],[])} \\ &\longrightarrow [a] \end{aligned}$$

Entsprechend der Definition können die beiden bedingten Regeln in einer (der einzigen) Klasse C_1 zusammengefaßt werden, da sie beide die linke Seite $append(xs, ys)$ und die auszuwertende Bedingung $nil(xs)$ beinhalten. Demnach ist nur ein neues Funktionssymbol $if_1^{(3)}$ hinzuzufügen, sowie die beiden bedingten Regeln durch insgesamt drei neue zu ersetzen. Das Ergebnis der Transformation nach Giovanetti und Moiso sieht also wie folgt aus:

$$\mathcal{R}_{\mathcal{GM}} = \left(\begin{array}{lllll} head(x:xs) & \longrightarrow & x & nil([]) \longrightarrow & true \\ tail(x:xs) & \longrightarrow & xs & nil(x:xs) \longrightarrow & false \\ & & & & \\ append(xs, ys) & \longrightarrow & \multicolumn{3}{l}{if_1(xs, ys, nil(xs))} \\ if_1(xs, ys, true) & \longrightarrow & \multicolumn{3}{l}{ys} \\ if_1(xs, ys, false) & \longrightarrow & \multicolumn{3}{l}{head(xs) : append(tail(xs), ys)} \end{array} \right)$$

Eine mögliche, entsprechende Ableitung im gewonnenen System sieht nun so aus, wobei sich die anderen Möglichkeiten von dieser nicht wesentlich unterscheiden.

$$\begin{array}{lll} \underline{append([a], [])} & \longrightarrow & if_1([a], [], \underline{nil([a])}) \\ & \longrightarrow & \underline{if_1([a], [], false)} \\ & \longrightarrow & \underline{head([a])} : append(tail([a]), []) \\ & \longrightarrow & a : append(\underline{tail([a])}, []) \\ & \longrightarrow & a : \underline{append([], [])} \\ & \longrightarrow & a : if_1([], [], \underline{nil([])}) \\ & \longrightarrow & a : \underline{if_1([], [], true)} \\ & \longrightarrow & [a] \end{array}$$

[GM87] enthält den Satz für die Vollständigkeit, Konfluenzerhaltung und Terminierungserhaltung (siehe Satz 3.7) der Transformation in bezug auf die Klasse der *easily safely transformable* Termersetzungssysteme. Starttermzuordnung ist auch hier die Identität, allerdings wird schon in der Veröffentlichung erkannt, daß eine Endtermzuordnung nicht existiert, das Verfahren also entsprechend nicht korrekt ist.

Die Klasse der *easily safely transformable* Termersetzungssysteme fällt aus dem hier verwandten Begriff der bedingten Termersetzungssysteme heraus, da für v in einer Regel $l \longrightarrow r\ if\ u \xrightarrow{*} v$ auch Terme zugelassen sind, die weder Normalformen noch Grundterme sind, sowie Extravariablen enthalten können.

Die Schnittmenge der *easily safely transformable* Systeme mit den hier definierten, bedingten Termersetzungssystemen ist definiert durch:

Definition 2.2.2: (Easily Safely Transformable)

In der Klasse der *easily safely transformable* bedingten Termersetzungssysteme gilt für jedes enthaltene $\mathcal{R}$:

- $\mathcal{R}$ ist konfluent und terminierend
- $\mathcal{R}$ ist reduzierend (siehe Definition 3.4.8 weiter unten)
- Der unbedingte Teil von $\mathcal{R}$ (vergleiche Satz 2.1) ist orthogonal.

$\diamond$

Satz 2.2: (Vollständigkeit der Transformation $\mathcal{GM}$)
Mit der Identität als Starttermzuordnung ist $\mathcal{BK}$ vollständig, konfluenzerhaltend und terminierungserhaltend (Satz 3.7) bezüglich der Klasse der *easily safely transformable* Termersetzungssysteme. ◇

Da ja bereits Konfluenz und Terminierung vorausgesetzt sind, bedeutet „-erhaltend“ im Zusammenhang dieses Satzes nichts anderes, als daß bei gegebenem $\mathcal{R}$ das Ergebnis $\mathcal{R}_{\mathcal{GM}}$ ebenfalls konfluent und terminierend ist.

Das folgende Beispiel wird zeigen, daß die Konfluenzerhaltung sofort verloren ist, läßt man Regeln zu, die zwar überlappen, jedoch aufgrund ihrer Bedingungen nicht auf denselben Term anwendbar sind. Dies entspräche immer noch dem Grundgedanken der Orthogonalität (vergleiche 4.1).

Beispiel 2.3

$$\mathcal{R} = \left(\begin{array}{lcllcl} head(x:xs) & \longrightarrow & x & nil([]) & \longrightarrow & true \\ tail(x:xs) & \longrightarrow & xs & nil(x:xs) & \longrightarrow & false \\ append([],ys) & \longrightarrow & ys & & & \\ append(xs,ys) & \longrightarrow & \multicolumn{4}{l}{head(xs):append(tail(xs),ys)\ if\ nil(xs) \stackrel{*}{\longrightarrow} false} \end{array} \right)$$

Auch in dieser Variation ist es unmöglich, die beiden *append*-Regeln überlappend anzuwenden; sie definieren im wesentlichen noch dasselbe wie zuvor. Dennoch liefert die Transformation jetzt ein System mit anderen Eigenschaften, da die erste (unbedingte) *append*-Regel nicht mehr in die Klasse C_1 eingeordnet wird und somit nicht durch eine neue Regel ersetzt wird.

$$\mathcal{R}_{\mathcal{GM}} = \left(\begin{array}{lcllcl} head(x:xs) & \longrightarrow & x & nil([]) & \longrightarrow & true \\ tail(x:xs) & \longrightarrow & xs & nil(x:xs) & \longrightarrow & false \\ & & & & & \\ append([],ys) & \longrightarrow & ys & & & \\ append(xs,ys) & \longrightarrow & \multicolumn{4}{l}{if_1(xs,ys,nil(xs))} \\ if_1(xs,ys,false) & \longrightarrow & \multicolumn{4}{l}{head(xs):append(tail(xs),ys)} \end{array} \right)$$

Nun zeigt eine einfache Ableitung, daß $\mathcal{GM}$ in diesem Fall nicht mehr konfluent ist.

$$\begin{array}{lccc} & & \underline{append([],[])} & \\ if_1([],[],\underline{nil([])}) & \longleftarrow & & \longrightarrow \ [] \\ if_1([],[],true) & \longleftarrow & & \end{array}$$

Auch wenn die hier betrachtete Transformation schon einen vielversprechenderen Eindruck macht als die vorhergehende, so ist doch der Bereich, auf dem sie vollständig und konfluenzerhaltend ist, noch zu klein. Hier liegt das Problem vor allem darin, daß *verschiedene* Funktionssymbole zur Bedingungsauswertung *einer* Funktion benutzt werden. Dieses Problem wird erst mit dem in Abschnitt 2.4 betrachteten Verfahren abgestellt.

2.3 Hintermeier

Während es bei den bisherigen Ansätzen darum ging, die gewünschte Elimination von Bedingungen durch geschickte Umformung des Termersetzungssystems zu erreichen, verfolgt Hintermeier in [Hin95] ein anderes Ziel. Die bedingten Teile des Ausgangssystems werden durch hinzugefügte Regeln explizit ausgewertet, wobei unter anderem ein Matchingalgorithmus sowie Regeln für die Anwendung von Ausdrücken angefügt und eine Regelreihenfolge gefordert wird, um eine „Bottom-Up“-Auswertung zu erhalten.

Mit diesen Bestandteilen wird das unbedingte Ergebnissystem stark vergrößert und enthält Teile, die normalerweise einer Auswertungsstrategie zuzuschreiben sind. Gerade die Auswertungsreihenfolge spielt eine große Rolle, wenn es darum geht, auch im nichtterminierenden Fall Vollständigkeit zu erzielen.

Was Hintermeier leistet, ist ein Interpreter auf Basis eines Termersetzungssystems, der auch nur unter bestimmten, für die Zielsetzung dieser Arbeit wiederum zu restriktiven, Voraussetzungen vollständig ist. So fordert Hintermeier die Terminierung, was, wie bereits weiter oben erwähnt, in den Bereich der funktional-logischen Programmierung nicht sinnvoll übertragbar ist. Es ist aber, gerade wegen der miteingebauten Bottom-Up-Auswertung, leicht einzusehen, daß die Forderung der Terminierung für Hintermeiers Ansatz eine wesentliche Rolle spielt, also nicht, ohne Einbuße der Vollständigkeit, weggelassen werden kann.

Leider ist Hintermeiers Verfahren aus den genannten Gründen zu komplex, als daß eine Darstellung in einer angepaßten Syntax und eine Anwendung auf das einfache Referenzbeispiel gewinnbringend durchgeführt werden könnte.

2.4 Viry

In [AGM90] skizzierten Aida, Goguen und Meseguer ein Transformationsverfahren, welches dann von Viry in [Vir95] formalisiert und untersucht wurde. Dieses Verfahren hat, wie sich im folgenden zeigen wird, einige interessante Eigenschaften. Es stellt die Basis weiterer Modifikationen in den folgenden Kapiteln dar. Zunächst wird hier gezeigt, daß die Transformation von Viry alle Stärken der bisher untersuchten Ansätze besitzt und darüber noch hinausgeht. Dennoch enthält sie noch eine Schwachstelle, die dann im weiteren Verlauf der Arbeit durch eine gezielte Modifikation beseitigt werden kann (Abschnitt 3.4). Schließlich werden in Kapitel 4 noch spezielle Ergebnisse mit Bezugnahme auf die funktional-logische Programmierung erzielt.

Grundidee des Verfahrens:
Bei der Transformation von Viry wird jedes Funktionssymbol um eine Stelle pro bedingter Regel für diese Funktion erweitert. An diesen Stellen soll die Auswertung der jeweiligen Bedingung erfolgen. Hat die Auswertung die gewünschte Normalform erreicht, wird der zugehörige Ersetzungsschritt durchgeführt. Ein zusätzliches Symbol $\perp$ wird eingeführt, um eine noch nicht angestoßene Bedingungsauswertung zu symbolisieren. Neu an dieser Idee ist, daß mit ihr sowohl bedingte als auch unbedingte Regeln für ein und dasselbe Funktionssymbol erfaßt werden können.

Definition 2.4.1: (Transformation $\mathcal{V}$ von Viry)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem und für jedes Funktionssymbol $f^{(n)}$ aus Σ seien $\rho_{f,1}, \ldots, \rho_{f,m}$ die bedingten und $\rho_{f,m+1}, \ldots, \rho_{f,m+k}$ die unbedingten Regeln aus P.

Dann entsteht durch die Transformation von Viry das unbedingte Termersetzungssystem $\mathcal{R}_{\mathcal{V}} = \langle \Sigma_{\mathcal{V}}, X, \mathrm{P}_{\mathcal{V}} \rangle$, welches wie folgt definiert ist:

$$\mathrm{P}_{\mathcal{V}} \quad = \quad (\mathrm{P} \setminus \{\rho_{f,1}, \ldots, \rho_{f,m+k}\}) \cup \{\rho'_{f,1}, \ldots, \rho'_{f,m}\} \cup \{\rho''_{f,1}, \ldots, \rho''_{f,m+k}\}$$

$\Sigma_{\mathcal{V}}$ ist gleich $\Sigma \cup \perp^{(0)}$. Außerdem haben die Funktionssymbole $f^{(n)} \in \Sigma$ nun die Stelligkeit $(n+m)$.

Jeder bedingten Regel der Form $\rho_{f,i} \,:\, f(t_1,\ldots,t_n) \longrightarrow r_{f,i}\ if\ u_{f,i} \xrightarrow{*} v_{f,i}$ mit $var(r_{f,i}) = \{x_1,\ldots,x_j\}$ werden zwei unbedingte Regeln der Form

$$\begin{aligned}
\rho'_{f,i} \;&:\; f(t_1^X,\ldots,t_n^X \mid y_1,\ldots,\bot_i,\ldots,y_m) \longrightarrow f(t_1^X,\ldots,t_n^X \mid y_1,\ldots,[u_{f,i}^{\bot},(\vec{x})]_i,\ldots,y_m)\\
\rho''_{f,i} \;&:\; f(z_1,\ldots,z_n \mid y_1,\ldots,[v_{f,i}^X,(\vec{x})]_i,\ldots,y_m) \longrightarrow r_{f,i}^{\bot}
\end{aligned}$$

zugeordnet, wobei $\vec{x} = (x_1,\ldots,x_j)$ ist und es sich bei den $y_1,\ldots,y_m$ und $z_1,\ldots,z_n$ um frische Variablen handelt.

Außerdem wird jeder unbedingten Regel $\rho_{f,i} \,:\, f(t_1,\ldots,t_n) \longrightarrow r_{f,i}$ eine Regel der Form

$$\rho''_{f,i} \;:\; f(t_1^X,\ldots,t_n^X \mid y_1,\ldots,y_m) \longrightarrow r_{f,i}^{\bot}$$

zugeordnet. ◇

Das Zeichen | trägt nur zur Übersichtlichkeit bei. Es trennt die ursprünglichen von den neuen Stellen im Term. Die ursprünglichen Stellen werden im weiteren als *unbedingt*, die neuen als *bedingt* bezeichnet. Die Schreibweise $[\ldots]_i$ und $\bot_i$ sollen lediglich andeuten, daß die entsprechenden Symbole an der i-ten bedingten Stelle des Termes stehen. t^X und $t^{\bot}$ sind zwei einfache Abbildungen, welche die bedingten Stellen mit frischen Variablen bzw. $\bot$-Symbolen füllen und wie folgt definiert sind.

Definition 2.4.2: (Abbildung $t^X : \mathcal{T}_{\Sigma}(X) \longrightarrow \mathcal{T}_{\Sigma_{\mathcal{V}}}(X)$)
Es sei X eine Variablenmenge, mit $X = \{y_1,\ldots,y_m\} \uplus X_1 \uplus \ldots \uplus X_n$ und $var(t) \cap X = \emptyset$.

$$t^X = \begin{cases} t & : \ t \in X \\ f(t_1^{X_1},\ldots,t_n^{X_n} \mid y_1,\ldots,y_m) & : \ t = f(t_1,\ldots,t_n),\ n,m \geq 0 \end{cases}$$

◇

Definition 2.4.3: (Abbildung $t^{\bot} : \mathcal{T}_{\Sigma}(X) \longrightarrow \mathcal{T}_{\Sigma_{\mathcal{V}}}(X)$)

$$t^{\bot} = \begin{cases} t & : \ t \in X \\ f(t_1^{\bot},\ldots,t_n^{\bot} \mid \underbrace{\bot,\ldots,\bot}_{m}) & : \ t = f(t_1,\ldots,t_n),\ n,m \geq 0 \end{cases}$$

◇

Diese Abbildung ist auch die in Definition 1.5.1 geforderte Starttermzuordnung. Sie erfüllt nach Konstruktion die nötige Homomorphieeigenschaft. Die Endtermzuordnung in der entgegengesetzten Richtung ist gegeben durch

Definition 2.4.4: (Abbildung $t \mapsto \overline{t} : \mathcal{T}_{\Sigma_{\mathcal{V}}}(X) \longrightarrow \mathcal{T}_{\Sigma}(X)$)

$$\overline{t} = \begin{cases} t & : \ t \in X \\ f(\overline{t_1},\ldots,\overline{t_n}) & : \ t = f(t_1,\ldots,t_n \mid y_1,\ldots,y_m),\ n,m \geq 0, f \neq \bot \end{cases}$$

◇

Da $t \mapsto \overline{t}$ lediglich für Terme undefiniert ist, bei denen $\bot$ an einer unbedingten Stelle steht, gehört die gesamte Menge der sinntragenden Terme zum Definitionsbereich der Abbildung. Da außerdem für alle Terme t aus $\mathcal{T}_{\Sigma}(X)$ $\overline{t^{\bot}} = t$ gilt, erfüllt die Abbildung die Forderungen für eine Endtermzuordnung (Definition 1.5.3).

Wie immer wird die Arbeitsweise der Transformation von Viry am Referenzbeispiel verdeutlicht.

Beispiel 2.4

$$\mathcal{R} = \left(\begin{array}{lllll} head(x:xs) & \longrightarrow & x & nil([]) \longrightarrow & true \\ tail(x:xs) & \longrightarrow & xs & nil(x:xs) \longrightarrow & false \\ append(xs,ys) & \longrightarrow & \multicolumn{3}{l}{ys \; if \; nil(xs) \xrightarrow{*} true} \\ append(xs,ys) & \longrightarrow & \multicolumn{3}{l}{head(xs) : append(tail(xs),ys) \; if \; nil(xs) \xrightarrow{*} false} \end{array} \right)$$

Statt der beiden bedingten Regeln für *append* werden vier neue Regeln eingesetzt; das Funktionssymbol *append* erhält zwei zusätzliche Stellen hinter dem |-Symbol. In den ersten beiden neuen Regeln, den ρ'-Regeln, werden diese Stellen, falls sie mit $\perp$ belegt sind, mit der entsprechenden auszuwertenden Bedingung, in beiden Fällen $nil(xs)$, initialisiert. Die anderen Regeln sorgen dafür, daß bei erfolgreicher Auswertung, $true$ bzw. $false$, durch die entsprechende rechte Regelseite, bei der alle auftretenden bedingten Stellen mit $\perp$ belegt wurden, ersetzt wird.

$$\mathcal{R}_{\mathcal{V}} = \left(\begin{array}{lll} head(x:xs) \longrightarrow x & & nil([]) \longrightarrow true \\ tail(x:xs) \longrightarrow xs & & nil(x:xs) \longrightarrow false \\ & & \\ append(xs,ys \mid \perp, y_2) & \longrightarrow & append(xs,ys \mid [nil(xs),(ys)],y_2) \\ append(xs,ys \mid y_1, \perp) & \longrightarrow & append(xs,ys \mid y_1,[nil(xs),(xs,ys)]) \\ append(z_1,z_2 \mid [true,(ys)],y_2) & \longrightarrow & ys \\ append(z_1,z_2 \mid y_1,[false,(xs,ys)]) & \longrightarrow & head(xs) : append(tail(xs),ys \mid \perp,\perp) \end{array} \right)$$

Damit nun die Referenzableitung (siehe Beispiel 2.1) des Ausdrucks $append([a],[])$ durchgeführt werden kann, muß zunächst die Starttermzuordnung (siehe Definition 1.5.1) auf diesen angewandt werden.

$$append([a],[])^{\perp} = append([a],[] \mid \perp,\perp)$$

Dann sieht eine der möglichen Ableitungen wie folgt aus:

$$\begin{array}{rll} & & \underline{append([a],[] \mid \perp,\perp)} \\ (1) & \longrightarrow & append([a],[] \mid [\underline{nil([a])},([])],\perp) \\ & \longrightarrow & \underline{append([a],[] \mid [false,([])],\perp)} \\ (2) & \longrightarrow & append([a],[] \mid [false,([])],[\underline{nil([a])},([a],[])]) \\ & \longrightarrow & \underline{append([a],[] \mid [false,([])],[false,([a],[])])} \\ (3) & \longrightarrow & \underline{head([a])} : append(tail([a]),[] \mid \perp,\perp) \\ & \longrightarrow & a : append(\underline{tail([a])},[] \mid \perp,\perp) \\ & \longrightarrow & a : \underline{append([],[] \mid \perp,\perp)} \\ (4) & \longrightarrow & a : append([],[] \mid [\underline{nil([])},([])],\perp) \\ & \longrightarrow & a : \underline{append([],[] \mid [true,([])],\perp)} \\ (5) & \longrightarrow & [a] \end{array}$$

Anmerkungen:

1. Egal welche der beiden Bedingungen zuerst ausgewertet wird; das Ergebnis bleibt dasselbe. Um das zu verdeutlichen, wird hier zunächst die „falsche" Bedingung geprüft, auch wenn eine andere Regelanwendung zu einer kürzeren Ableitung führen würde.

2. Nachdem die Auswertung der Bedingung mit $false$ das falsche Ergebnis liefert, bleibt nun nichts anderes übrig, als die zweite Bedingung zu initialisieren.

3. Als nächstes werden, um das Beispiel übersichtlich zu halten, die unbedingten Funktionen ersetzt.

4. Auch hier können beide Bedingungen gewählt werden, ohne das Ergebnis zu beeinflussen.

5. Nun ist noch die Endtermzuordung anzuwenden, in diesem Fall $\overline{[a]} = [a]$.

Virys Ansatz leidet nicht an dem Problem von Bergstra und Klop (siehe Abschnitt 2.1), da die rechten Regelseiten erst nach Auswertung der Bedingungen auftreten. Auch ist es möglich, Regeln mit Überlappungen zu behandeln; für Korrektheit und Vollständigkeit besteht keinerlei Orthogonalitätsforderung, wie weiter unten noch ersichtlich wird. Daher ist dieser Ansatz auch dem von Giovanetti und Moiso (siehe Abschnitt 2.3) vorzuziehen. Das liegt daran, daß keine Auswahl völlig verschiedener Funktionssymbole if_i stattfindet, sondern alle Bedingungen parallel nebeneinander stehen. Da die Transformation auch mit Funktionssymbolen umgehen kann, für die es sowohl bedingte als auch unbedingte Regeln gibt, läßt sich auch die Veränderung in Beispiel 2.3 ohne Problem behandeln. Aus $append([], ys) \longrightarrow ys$ wird dann die ρ''-Regel $append([], ys \mid y_1) \longrightarrow ys$, da es dann nur noch eine bedingte Regeln für $append$ gibt.

Insgesamt gilt das von Viry in [Vir95] bewiesene Ergebnis:

Satz 2.3: (Vollständigkeit und Korrektheit der Transformation $\mathcal{V}$)
Mit Starttermzuordnung $t \mapsto t^{\perp}$ und Endtermzuordnung $t \mapsto \bar{t}$ ist $\mathcal{V}$ vollständig und korrekt bezüglich der gesamten Klasse der bedingten Termersetzungssysteme. $\diamond$

Dieses erfreuliche Ergebnis wird nur durch die Erkenntnis getrübt, daß $\mathcal{V}$ nicht konfluenzerhaltend ist, wie das nächste Beispiel weiter unten zeigen wird.

Doch zunächst soll anhand von $\mathcal{V}$ die Menge der sinntragenden Terme aus Definition 1.5.2 aufgezeigt werden. Ein Beispiel für einen Term außerhalb der Menge $T(\mathcal{R}, t \mapsto t^{\perp})$ mit dem $\mathcal{R}$ aus Beispiel 2.4 ist der Term $append([a], [] \mid [true, ([])], [false, ([a], [])])$. Inhaltlich besagt dieser Term, daß der Test $nil([a])$ *sowohl positiv als auch negativ* ausgefallen ist, und kann entsprechend sowohl zu $[]$ als auch zu $head([a]) : append(tail([a]), [])$ und damit letztlich zu $[a]$ reduziert werden. Das kann nicht im Sinne des Ausgangssystems sein. Gemäß der Definition für $T(\mathcal{R}, t \mapsto t^{\perp})$ und dem Satz 2.3 kann ein solcher Term niemals aus einem Startterm hergeleitet worden sein. Entsprechend kann es bei der Frage der Konfluenzerhaltung, wie auch in der Definition 1.5.6 erfaßt, nur um die Menge der sinntragenden Terme gehen.

Nun zu dem angesprochenen Gegenbeispiel zur Konfluenzerhaltung von $\mathcal{V}$. Das folgende Beispiel wurde von Viry selbst in [Vir95] angegeben.

Beispiel 2.5

$$\mathcal{R} = \left(\begin{array}{llll} \rho_{f,1}: & f(g(x)) & \longrightarrow & h(x) \quad if\ c(x) \xrightarrow{*} true \\ \rho_{c,1}: & c(a) & \longrightarrow & true \\ \rho_{g,1}: & g(g(x)) & \longrightarrow & g(x) \end{array} \right)$$

$\mathcal{R}$ ist konfluent und besitzt nur eine bedingte Regel. Die Ableitung, für die es im transformierten System eine problematische Entsprechung gibt, ist sehr einfach.

$$f(g(g(a))) \xrightarrow{\rho_{g,1}} f(g(a)) \xrightarrow{\rho_{g,1}} h(a)$$

Das zu $\mathcal{R}$ gehörige unbedingte System hat nur eine Regel mehr als $\mathcal{R}$; die unbedingten Regeln werden hier lediglich gemäß der Definition umbenannt, gewissermaßen durch sich selbst ersetzt.

$$\mathcal{R}_{\mathcal{V}} = \left(\begin{array}{lllll} \rho'_{f,1}: & f(g(x) \mid \bot) & \longrightarrow & f(g(x) \mid [c(x), (x)]) \\ \rho''_{f,1}: & f(z_1 \mid [true, (x)]) & \longrightarrow & h(x) \\ \rho''_{c,1}: & c(a) & \longrightarrow & true \\ \rho''_{g,1}: & g(g(x)) & \longrightarrow & g(x) \end{array} \right)$$

In $\mathcal{R}_{\mathcal{V}}$ gibt es zwei Ableitungen von $f(g(g(a)))^{\bot}$ aus.

$$\begin{array}{rcccl} & & \underline{f(\underline{g(g(a))} \mid \bot)} & & \\ \underline{f(g(a) \mid \bot)} & \xleftarrow{\rho''_{g,1}} & & \xrightarrow{\rho'_{f,1}} & f(\underline{g(g(a))} \mid [c(g(a)), (g(a))]) \\ f(g(a) \mid [\underline{c(a)}, (a)]) & \xleftarrow{\rho'_{f,1}} & & \xrightarrow{\rho''_{g,1}} & f(g(a) \mid [c(g(a)), (g(a))]) \\ \underline{f(g(a) \mid [true, (a)])} & \xleftarrow{\rho''_{c,1}} & & & \\ h(a) & \xleftarrow{\rho''_{f,1}} & & & \end{array}$$

Die links aufgeführte Ableitung ist die laut Satz 2.3 existierende Entsprechung der oben angegebenen Ableitung im Ausgangssystem. Die zweite Reduktionsfolge hat zwar ebenfalls eine Entsprechung in $\mathcal{R}$, eben $f(g(g(a))) \longrightarrow f(g(a))$, sie läßt sich jedoch nicht weiter zu $h(a)$ reduzieren. Trotz der Konfluenz von $\mathcal{R}$ ist $\mathcal{R}_{\mathcal{V}}$ nicht konfluent, $\mathcal{V}$ also nicht konfluenzerhaltend. Das gilt zumindest für die Klasse *aller* bedingten Termersetzungssysteme. In Abschnitt 3.3 wird gezeigt, daß $\mathcal{V}$ sehr wohl konfluenzerhaltend bezüglich der Klasse der konstruktorbasierten Termersetzungssysteme ist.

Viry macht für das Problem der Konfluenzerhaltung einen Lösungsversuch. Wie schon im Abschnitt 1.5 angedacht, kann Konfluenzerhaltung unter Umständen auch nur in Verbindung mit einer bestimmten Ersetzungsstrategie gewährleistet sein. Zu diesem Zweck definiert Viry eine Ableitungsklasse, die er *conditional eager* nennt.

Definition 2.4.5: (Conditional Eager Ableitung)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem.
Eine Ableitung α in $\mathcal{R}_{\mathcal{V}}$ heißt *conditional eager*, falls für jede Anwendung einer ρ'-Regel

$$\alpha = s \xrightarrow{*}_{\mathcal{R}_{\mathcal{V}}} t \longrightarrow_{\mathcal{R}_{\mathcal{V}}[\rho',p]} \underbrace{u \xrightarrow{*}_{\mathcal{R}_{\mathcal{V}}} v}_{\beta}$$

gilt, daß in der gesamten Restableitung β keinerlei ρ''-Regeln an unbedingten Stellen von $t|_p$ und dessen Nachfolgern angewendet werden. $\diamond$

Dies soll das Konfluenzproblem lösen, da auf diese Weise gewährleistet werden soll, daß die Argumente vor der Anwendung der Bedingungsinitialisierungen ρ' bereits „genügend ausgewertet“ wurden. Nach einer solchen Initialisierung sind die zugehörigen unbedingten Stellen also unangetastet zu belassen, um die bedingten Stellen nicht in Gefahr zu bringen, „aus dem Kontext zu rutschen“. Schaut man noch einmal auf die kritische Ableitung in Beispiel 2.5, so ergibt sich, daß die rechte Ableitung $f(g(g(a)) \mid \perp) \xrightarrow{*} f(g(a) \mid [c(g(a)), (g(a))])$ in der Tat nicht *conditional eager* ist.

Um hundertprozentig sicher zu gehen, daß die unbedingten Stellen weit genug ausgewertet sind, müssen sie bis zur Normalform reduziert werden. Aufgrund dieser strikten Grundidee für die Auswertung trägt der Begriff wohl auch die Bezeichnung *eager*. Eines der Probleme dieser „Lösung“ ist, daß solche strikte Auswertung bekanntermaßen in Gegenwart von Nichtterminierung zur Unvollständigkeit führt [Thi94]. Und auch wenn Viry in seiner Veröffentlichung [Vir95] beweist, daß zu jeder erfolgreichen Ableitung auch eine *conditional eager* Ableitung existiert, so ist dieses Ergebnis dennoch in der Praxis nicht umsetzbar, da es nicht entscheidbar ist, wann die bedingten Stellen „genügend ausgewertet“ sind. In [Vir95] vermutet Viry:

> Intuitively, this property [conditional eagerness] ensures that conditions are evaluated in the “right” context, ie. the critical peak of the first example above [Beispiel 2.5] would not be possible. In fact, that example suggests that any sound interpreter for conditional rewriting should implement a similiar strategy in order to avoid backtracking on the evaluation of conditions.

Dies erweist sich insofern als falsch, als ein derartiger Interpreter nicht konstruiert werden kann. Ein derartiges Programm müßte entscheiden können, wann die unbedingten Stellen ausreichend reduziert sind, um die Bedingungen erfolgreich auswerten zu können. Diese Entscheidung aber ist, ohne die Bedingungen selbst auszuwerten nicht zu treffen, denn die grundsätzliche Auswertung zur Normalform scheidet wegen Nichtterminierung aus, und ob ein Teilterm strikt auswertbar ist, oder nicht, ist genauso unentscheidbar wie die Terminierung [Thi94]. Vielmehr muß ein vollständiger Interpreter in der Lage sein, den Kontext zu wechseln, falls es nötig wird (genaueres dazu in 3.4).

Sein Lösungsansatz erweist sich aber auch noch in einer anderen grundlegenden Hinsicht als falsch. Die Einschränkung auf *conditional eager* Ableitung reicht nämlich, auch wenn Viry das in [Vir95] zu beweisen glaubt, nicht aus, um eine Form von Konfluenzerhaltung zu gewährleisten. Um dies darzulegen wird zunächst angegeben, was Viry genau für die Klasse der *conditional eager* Ableitungen behauptet. Es ist daran zu erinnern, daß entsprechend der Definition 1.5.2 ein sinntragender Term aus einem mit der Starttermzuordnung gebildeten Term entstanden sein muß.

Definition 2.4.6: ($\perp$-Konfluenz)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem.
$\mathcal{R}_\mathcal{V}$ heißt $\perp$-konfluent, falls für alle $s_1, s_2 \in \mathcal{T}_{\Sigma_\mathcal{V}}(X)$, die aus einem $\perp$-Term mittels einer *conditional eager* Ableitung entstanden sind, $t_1, t_2 \in \mathcal{T}_{\Sigma_\mathcal{V}}(X)$ existieren mit *conditional eager* Ableitungen $s_1 \xrightarrow{*} t_1$ und $s_2 \xrightarrow{*} t_2$ sowie $\overline{t_1} = \overline{t_2}$. ◇

Anschaulich sieht das ungefähr so aus:

$$\begin{array}{ccccc} & & s_1 & \xrightarrow{*} & t_1 \\ & {}^{*}\nearrow & & & \vdots \\ \overline{u}^{\perp} & & & & \overline{t_1} = \overline{t_2} \\ & \searrow^{*} & & & \vdots \\ & & s_2 & \xrightarrow{*} & t_2 \end{array}$$

Es schließt sich in [Vir95] der Satz an:

Satz 2.4: (Zweiter Satz von Viry)
Sei $\mathcal{R}$ ein konfluentes, bedingtes Termersetzungssystem.
Dann ist $\mathcal{R}_\mathcal{V}$ $\perp$-konfluent. $\diamond$

Dieser Satz ist falsch. Ein Gegenbeispiel ist in dem von Viry selbst gegebenen Beispiel 2.5 enthalten. Zu betrachten ist dazu nur ein Teil der kritischen Ableitung.

$$\begin{array}{ccc} & & f(g(g(a)) \mid [c(g(a)), (g(a))]) \\ & \overset{*}{\nearrow} & \\ f(g(g(a)) \mid \perp) & & \\ & \overset{*}{\searrow} & \\ & & h(a) \end{array}$$

Beide Ableitungen sind laut Definition *conditional eager*. Dennoch kann keine Zusammenführung der beiden Redukte erfolgen.

Doch trotz dieses negativen Ergebnisses ist die auf der Idee von Aida, Goguen und Meseguer beruhende Transformation brauchbar. Wie sich noch in Abschnitt 3.3 zeigen wird, tritt die Problematik in einer durchaus relevanten Klasse von bedingten Termersetzungssystemen gar nicht erst auf. Auch läßt sich die Idee so ausbauen, daß Konfluenzerhaltung für alle bedingten Termersetzungssysteme gewährleistet ist, siehe Abschnitt 3.4.

Kapitel 3

Tiefenanalyse der Transformation von Viry

In Kapitel 2 wurde herausgearbeitet, daß von den untersuchten Ansätzen zur Elimination von Bedingungen der von Viry in [Vir95] vorgestellte der am besten geeignete ist. In diesem Kapitel wird der Ansatz analysiert, wichtige Eigenschaften werden bewiesen, um das Verfahren in Kapitel 4 in Richtung auf den Bereich der funktional-logischen Programmierung spezialisieren zu können. Dazu werden in Abschnitt 3.1 einige grundlegende Begriffe eingeführt. Diese werden in Abschnitt 3.2 zur Analyse der Transformation anhand einer Modifikation genutzt. Diese Analyse stellt die Grundlagen bereit, mit denen dann in Abschnitt 3.3 die Konfluenzerhaltung auf einer eingeschränkten Klasse von Termersetzungssystemen bewiesen werden kann. Danach wird im Abschnitt 3.4 ein allgemeines konfluenzerhaltendes Verfahren vorgestellt. Das Kapitel schließt mit einem Exkurs über allgemeine Transformationsmethodik 3.5.

3.1 Grundlegende Arbeitsweise des Ergebnissystems

Der Schwerpunkt dieses Kapitels ist die Untersuchung der Konfluenzerhaltung der Transformation von Viry. Daß dies im allgemeinen, das heißt für die gesamte Klasse der bedingten Termersetzungssysteme, nicht gilt, hat sich bereits in Abschnitt 2.4 gezeigt. Es soll daher eine möglichst große Klasse gefunden werden, auf der $\mathcal{V}$ das gewünschte Ergebnis liefert. Dazu muß zunächst die Arbeitsweise des Ergebnissystems analysiert, mögliche Ansatzpunkte aufgezeigt werden.

Wie schon in der Vorstellung der Transformation und in Abschnitt 1.5 angesprochen, gibt es in jedem unbedingten Zielsystem immer einen besonderen Teil, der allein für die Konfluenz interessant ist. Dies ist die Menge der sinntragenden Terme. Wie in Definition 1.5.1 enthalten, ist die induktive Basis dieser Menge das Bild der Starttermzuordnung $t \mapsto t^{\perp}$. Ein Element dieser Basismenge weist an allen bedingten Stellen das Funktionssymbol $\perp$ auf. Deshalb werden derartige Terme als $\perp$-Terme bezeichnet. Da diesen Termen eine herausragende Rolle innerhalb von $\mathcal{R}_{\mathcal{V}}$ zukommt, werden sie hier eigens definiert.

Definition 3.1.1: ($\perp$-Term)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem und $u \in \mathcal{T}_{\Sigma_{\mathcal{V}}}(X)$.
Gilt $u = \overline{u}^{\perp}$, so heißt u $\perp$-Term. $\diamond$

Des weiteren bezieht sich die Definition für die Menge der sinntragenden Terme auf solche, die aus $\perp$-Termen ableitbar sind. Viry bezeichnet eine solche Ableitung in [Vir95] als *konservativ*.

Eine konservative Ableitung enthält aber zudem die Forderung, Extravariablen nur mit $\perp$-Termen zu belegen. Da aber in den Definitionen 1.2.7 und 1.4.1 Extravariablen ausgeschlossen wurden, braucht dieses Detail hier nicht weiter betrachtet zu werden, siehe 5.3. Ohne dies jedesmal besonders zu erwähnen, wird im folgenden davon ausgegangen, daß alle Terme t der Menge der sinntragenden Terme angehören.

Anzumerken ist noch in diesem Zusammenhang, daß, wie bei der Einführung der sinntragenden Terme erwähnt, auch eine Anpassung des Konfluenzbegriffes stattfinden muß. Es kann mit einer Einschränkung des Termuniversums nicht mehr um die Konfluenz des Gesamtsystems, sondern nur um Konfluenz des sinntragenden Teils, der Entsprechung von $\mathcal{R}$ in $\mathcal{R}_\mathcal{V}$, gehen. Dies wird hier nochmals ausdrücklich definiert.

Definition 3.1.2: (Konfluenz des sinntragenden Teils)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem.
Dann heißt $\mathcal{R}_\mathcal{V}$ konfluent im sinntragenden Teil, wenn für alle $\perp$-Terme aus $u \in \mathcal{T}_{\Sigma_\mathcal{V}}(X)$ gilt, daß alle Nachfolger von u zusammenführbar sind. $\diamond$

In Abschnitt 2.4 wurde mit dem Satz von Viry zum Ausdruck gebracht, daß die Transformation $\mathcal{V}$ korrekt und vollständig für alle Termersetzungssysteme ist. Es entspricht also jeder Ableitung $s \longrightarrow_\mathcal{R} t$ im Ausgangssystem eine Ableitung $s^\perp \longrightarrow_{\mathcal{R}_\mathcal{V}} t^\perp$ im Zielsystem. Dabei entspricht einem einzelnen bedingten Reduktionsschritt in $\mathcal{R}$ eine ganze Reihe von Schritten in $\mathcal{R}_\mathcal{V}$. Allgemein kann man zu jeder Ableitung in $\mathcal{R}$ eine *minimale Entsprechung* in $\mathcal{R}_\mathcal{V}$ finden, für deren Form gilt:

Lemma 3.1: (Minimale Entsprechung)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem.
Jede Ableitung α der Form $s_0 \longrightarrow_\mathcal{R} s_1 \longrightarrow_\mathcal{R} \ldots \longrightarrow_\mathcal{R} s_k$ hat eine Entsprechung $\alpha_\mathcal{V}$ in $\mathcal{R}_\mathcal{V}$ mit der Form

$$s_0^\perp \longrightarrow_{\mathcal{R}_\mathcal{V}} s_{11} \longrightarrow_{\mathcal{R}_\mathcal{V}} \ldots \longrightarrow_{\mathcal{R}_\mathcal{V}} s_{1j_1} \longrightarrow_{\mathcal{R}_\mathcal{V}} s_1^\perp \longrightarrow_{\mathcal{R}_\mathcal{V}} \ldots \longrightarrow_{\mathcal{R}_\mathcal{V}} s_{kj_k} \longrightarrow_{\mathcal{R}_\mathcal{V}} s_k^\perp$$

wobei die Anzahl j_i der Zwischenschritte jeweils minimal ist, so daß also keine kürzere Ableitung von $s_{i-1}^\perp$ nach $s_i^\perp$ in $\mathcal{R}_\mathcal{V}$ existiert.
Dann hat jeder beliebige Entsprechungsabschnitt aus $\alpha_\mathcal{V}$ die Form

$$s_{i-1}^\perp \longrightarrow_{\mathcal{R}_\mathcal{V}[\rho'_{f,j},p,\sigma]} s_{i1} \longrightarrow_{\mathcal{R}_\mathcal{V}} \ldots \longrightarrow_{\mathcal{R}_\mathcal{V}} s_{ij_i} \longrightarrow_{\mathcal{R}_\mathcal{V}[\rho''_{f,j},p,\tau]} s_i^\perp$$

wobei gilt, daß alle Reduktionen in „…“ an Positionen unterhalb von p stattfinden. Außerdem haben die entscheidenden Teilterme die Form:

$$\begin{aligned}
s_{i-1}^\perp|_p &= f(w_1, \ldots, w_n \mid \perp, \ldots, \perp_j, \ldots, \perp) \\
s_{i1}|_p &= f(w_1, \ldots, w_n \mid \perp, \ldots, [\sigma(u_{f,j}^\perp), \sigma(\vec{x})]_j, \ldots, \perp) \\
s_{ij_i}|_p &= f(w_1, \ldots, w_n \mid \perp, \ldots, [v_{f,j}^\perp, \sigma(\vec{x})]_j, \ldots, \perp) \\
s_i^\perp|_p &= \tau(r_{f,j}^\perp)(\sigma(\vec{x}))
\end{aligned}$$

$\diamond$

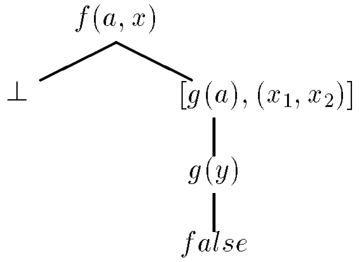
Dies ist nur ein Detail aus dem Beweis für den Satz von Viry und muß daher hier nicht eigens bewiesen werden. Es wird aber in späteren Beweisen in dieser Arbeit verwandt. Die Arbeitsweise dieser minimalen Ableitung sollte leicht nachvollziehbar sein, denn sie entspricht genau der Art, in der Reduktionsschritte im bedingten System angewandt werden: Zuerst wird die anzuwendende Regel gewählt, dann die zugehörige Bedingung bewiesen und schließlich der entsprechende Ersetzungsschritt durchgeführt.

Anzumerken sind einige Punkte:

- In jedem der minimalen Entsprechungsabschnitte wird eine bedingte Stelle $p{\cdot}j$ zunächst durch die zugehörige ρ'-Regel initialisiert und dann zu der entsprechenden Normalform ausgewertet (durch die Schritte in „...“). Dies ist dem nichtdeterministischen Verhalten des Ausgangssystems nachempfunden; es funktioniert nur mit dem Wissen, welche Bedingung die richtige ist. Mit der gewonnenen Normalform an der Stelle $p \cdot j$ kann dann der Schritt zu $s_i^{\perp}$ durchgeführt werden.
- Da die Zwischenschritte immer unterhalb der entsprechenden bedingten Stelle stattfinden, können die Entsprechungsabschnitte untereinander nicht interferieren.
- Einige Details der Teiltermformen, wie etwa $v_{f,j}^{\perp}$ oder der unverändert bleibende Vektor $\sigma(\vec{x})$, hängen direkt mit der Minimalität der Ableitungen zusammen. Es werden keine überflüssigen Reduktionen unternommen.

Als letztes Thema dieses Abschnittes wird eine im folgenden angewandte Beweistechnik angesprochen. Sie beruht auf der grundlegenden Einsicht, daß man sich die Terme des Zielsystems als induktiv aus $\perp$-Termen aufgebaut vorstellen kann. Dazu ein Beispiel: Der Term $f(a, x \mid \perp, [g(a \mid g(y \mid false)), (x_1, x_2)])$ sieht als „$\perp$-Term-Term“ wie folgt aus:

Beispiel 3.1



Um Eigenschaften der Transformation zu beweisen, wird über die *Tiefe dieser Baumstruktur*, im folgenden als *Bedingungstiefe* bezeichnet, induziert. Dabei sollen $\perp$-Blätter ignoriert werden, die Bedingungstiefe eines $\perp$-Termes also genau dann um eins erhöht werden, wenn eine der bedingten Stellen, mit welchem Term auch immer, initialisiert wird. Die entsprechende Definition lautet:

Definition 3.1.3: (Bedingungstiefe)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem.
Die Bedingungstiefe ist eine Abbildung $b : \mathcal{T}_{\Sigma_{\mathcal{V}}}(X) \longrightarrow I\!N$ und ist definiert durch

$$b(t) = \begin{cases} 0 & , t = \perp \\ 1 & , (t \neq \perp \wedge t^{(0)} \epsilon \Sigma_{\mathcal{V}}) \vee t \epsilon X \\ \max\{b(t_1), \ldots, b(t_n), 1 + b(s_1), \ldots, 1 + b(s_m)\} & , t = f(t_1, \ldots, t_n \mid s_1, \ldots, s_m) \\ \max\{b(s), b(x_1), \ldots, b(x_n)\} & , t = [s, (x_1, \ldots, x_n)] \end{cases}$$

$\diamond$

Es ist ersichtlich, daß mit dieser Definition die Bedingungstiefe eines Terms t genau dann gleich Eins ist, wenn t ein $\perp$-Term ist.

Beispiel 3.2
Für den Beispielterm gilt:

$$\begin{aligned}
& b(f(a, x \mid \perp, [g(a \mid g(y \mid false)), (x_1, x_2)])) \\
= \;& \max\{b(a), b(x), 1 + b(\perp), 1 + b([g(a \mid g(y \mid false)), (x_1, x_2)])\} \\
= \;& \max\{1, 1, 1 + 0, 1 + \max\{b(g(a \mid g(y \mid false))), b(x_1), b(x_2)\}\} \\
= \;& 1 + \max\{\max\{b(a), 1 + b(g(y \mid false))\}, 1, 1\} \\
= \;& 1 + 1 + \max\{b(y), 1 + b(false)\} \\
= \;& 1 + 1 + \max\{1, 1 + 1\} \\
= \;& 1 + 1 + 2 \\
= \;& 4
\end{aligned}$$

3.2 Analysierende Beobachtungen

In diesem Abschnitt wird eine modifizierte Transformation vorgestellt, welche für die Klasse aller bedingten Termersetzungssysteme konfluenzerhaltend ist. Allerdings weist sie einen anderen Nachteil auf, weshalb sie als Grundlage für die weitere Untersuchung weniger geeignet ist. Interessant an der Modifikation ist aber der Konfluenzbeweis selbst. Aus diesem lassen sich einige Schlüsse für die weiteren Beweise ziehen.

Definition 3.2.1: (Eine modifizierte Transformation $\mathcal{V}'$)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem.
Dann ist das unbedingte Termersetzungssystem $\mathcal{R}_{\mathcal{V}'} = \langle \Sigma_{\mathcal{V}}, X, \mathrm{P}_{\mathcal{V}'} \rangle$ definiert durch Hinzufügung einer neuen Regel für jedes Funktionssymbol wie folgt:

$$\mathrm{P}_{\mathcal{V}'} = \mathrm{P}_{\mathcal{V}} \cup \{f(x_1, \ldots, x_n \mid y_1, \ldots y_m) \longrightarrow f(x_1, \ldots, x_n \mid \perp, \ldots, \perp)\}$$

für alle $f^{(n+m)} \;\epsilon\; \Sigma_{\mathcal{V}}$. ◇

Die neu hinzugefügte Regel verändert nur die bedingten Stellen eines Terms. Da bei Anwendung der Regel nur bereits initialisierte Stellen wieder auf $\perp$ gesetzt werden, bleibt Korrektheit wie Vollständigkeit der Transformation erhalten. Zusätzlich ist die Entsprechung des Ausgangssystem im Zielsystem konfluent, wenn dies auf das Ausgangssystem ebenfalls zutrifft.

Satz 3.1: $\mathcal{V}'$ ist konfluenzerhaltend. ◇

Beweis:

Zu zeigen ist, daß für jedes konfluente bedingte Termersetzungssystem $\mathcal{R}$ und alle u, t_1, t_2 aus $T(\mathcal{R}, t \mapsto t^{\perp})$ mit $t_1 \overset{*}{\longleftarrow} u \overset{*}{\longrightarrow} t_2$ gilt: $t_1 \downarrow t_2$.

Da mit Satz 2.3 zu jeder Ableitung in $\mathcal{R}_{\mathcal{V}'}$ auch eine Ableitung in $\mathcal{R}$ existiert, gibt es Ableitungen

$$\overline{t_1} \overset{*}{\longleftarrow}_{\mathcal{R}} \overline{u} \overset{*}{\longrightarrow}_{\mathcal{R}} \overline{t_2}$$

Da $\mathcal{R}$ laut Voraussetzung konfluent ist, existiert ein $v \;\epsilon\; \mathcal{T}_{\Sigma}(X)$ mit

$$\overline{t_1} \overset{*}{\longrightarrow}_{\mathcal{R}} v \overset{*}{\longleftarrow}_{\mathcal{R}} \overline{t_2}$$

Daraus folgt direkt die Existenz entsprechender Ableitungen in $\mathcal{R}_{\mathcal{V}'}$

$$\begin{tikzcd}[column sep=small]
 & \overline{t_1}^{\perp} \arrow[dr, "*"] & \\
u \arrow[ur, "*"] \arrow[dr, "*"'] & & v^{\perp} \\
 & \overline{t_2}^{\perp} \arrow[ur, "*"'] &
\end{tikzcd}$$

Nun gilt mit den neu in $P_{\mathcal{V}'}$ eingeführten Regeln $t_1 \xrightarrow{*}_{\mathcal{R}_{\mathcal{V}'}} \overline{t_1}^{\perp}$ und $t_2 \xrightarrow{*}_{\mathcal{R}_{\mathcal{V}'}} \overline{t_2}^{\perp}$ und somit auch $t_1 \xrightarrow{*}_{\mathcal{R}_{\mathcal{V}'}} v^{\perp}$ und $t_2 \xrightarrow{*}_{\mathcal{R}_{\mathcal{V}'}} v^{\perp}$. Es ergibt sich also $t_1 \downarrow t_2$.

Somit ist auch der sinnvolle Teil von $\mathcal{R}_{\mathcal{V}'}$ konfluent. □

Unwillkürlich stellt sich die Frage, warum mit der Geltung dieses Lemmas nicht die modifizierte Transformation statt der von Viry benutzt wird. Doch hat diese neue Transformation den Nachteil zusätzlicher Nichtterminierung, ganz ähnlich dem Problem der Transformation von Bergstra und Klop (Abschnitt 2.1). Offensichtlich kann jede der neuen Regeln beliebig oft nacheinander auf denselben Term angewandt werden. Als Arbeitsgrundlage ist die neue Transformation also weniger geeignet.

Interessant an diesem modifizierten System ist allerdings die Erkenntnis, daß das Konfluenzproblem in $\mathcal{R}_{\mathcal{V}}$ ausschließlich auf die fehlerhafte Initialisierung der bedingten Stellen zurückzuführen ist. Mit anderen Worten, sind die Inhalte der bedingten Stellen nicht mehr im Kontext mit dem Term, der durch die unbedingten Stellen vorgegeben ist. Gibt es, wie durch die Modifikation, die Möglichkeit, diese Stellen jederzeit neu zu initialisieren, so stellt sich die gewünschte Konfluenzeigenschaft ein.

Aber eben dieser Ansatz der *jederzeit* möglichen Neuinitialisierung ist auch die Schwäche dieses zweiten Ansatzes. Idealerweise sollte nur dann neu initialisiert werden, wenn die bedingten Stellen des Terms sich nicht mehr im Kontext befinden. Gefordert ist also nicht unbedingt eine beliebige Neuinitialisierung, sondern nur, daß der bedingte Teil eines Terms jederzeit kontextstimmig ist oder wieder in diesen Zustand gebracht werden kann. Wie aber kann man erkennen, daß ein Term sich im Kontext befindet? Nach Konstruktion ist ein $\perp$-Term immer im richtigen Kontext. Im nächsten Lemma wird gezeigt, daß es ausreicht zu zeigen, daß ein beliebiger Term t mit dem zugehörigen $\perp$-Term $\overline{t}^{\perp}$ zusammenführbar ist, um Kontextstimmigkeit zu gewährleisten. Im folgenden wird eine Transformation, die zusätzlich zu den Viry-Regeln nur solche enthält, die ein Überschreiben bedingter Stellen mit $\perp$ ermöglichen, als Viry-ähnlich bezeichnet.

Aus dieser Betrachtung des Systems läßt sich eine hinreichende Bedingung für die Konfluenz des Zielsystems gewinnen, welche die später durchgeführten Beweise für Konfluenzerhaltung in einer übersichtlichen Weise ermöglicht. Zur Vorbereitung des entsprechenden Satzes werden nun zwei Hilfssätze formuliert und bewiesen. Der erste gibt eine hinreichende Bedingung dafür an, wann aus einem Term t alles abgeleitet werden kann, was auch aus dem zugehörigen $\perp$-Term $\overline{t}^{\perp}$ ableitbar ist. Hat t diese Eigenschaft, kann er als ableitungsvollständig bezeichnet werden.

Lemma 3.2: (Ableitungsvollständigkeit)

Sei $\mathcal{R}$ ein konfluentes, bedingtes Termersetzungssystem, $t, u \in T(\mathcal{R}, x \mapsto x^{\perp})$ mit $\overline{t}^{\perp} \xrightarrow{*} \overline{u}^{\perp}$ und $\mathcal{R}_{\mathcal{V}}$ aus $\mathcal{R}$ durch eine Viry-ähnliche Transformation entstanden.
Gilt für alle $s \in T(\mathcal{R}, x \mapsto x^{\perp})$ mit Bedingungstiefe $b(s) \leq b(t)$ (siehe Definition 3.1.3) die Zusammenführbarkeit $s \downarrow \overline{s}^{\perp}$, wobei für alle in den beiden Ableitungen vorkommenden Terme s' gilt $\overline{s'} = \overline{s}$, so gilt auch die Zusammenführbarkeit $t \xrightarrow{*} v \xleftarrow{*} \overline{u}^{\perp}$ mit $\overline{u} = \overline{v}$. ◇

Beweis:
Mit dem Satz 2.3 von Viry folgt die Existenz einer Ableitung $\overline{t} \xrightarrow{*}_{\mathcal{R}} \overline{u}$. Diese Ableitung habe die Form

$$(\overline{t} =)t_0 \longrightarrow_{\mathcal{R}} t_1 \longrightarrow_{\mathcal{R}} \ldots \longrightarrow_{\mathcal{R}} t_n (= \overline{u})$$

Entsprechend kann diese Ableitung in $\mathcal{R}_{\mathcal{V}}$ nachgebildet werden. Diese Art der Entsprechung wurde auch im Beweis des Satzes von Viry verwandt (siehe [Vir95]), entstammt aber genau der Verwendung des bedingten Systems $\mathcal{R}_{\mathcal{V}}$. Um von t_i zu t_{i+1} übergehen zu können, muß die zugehörige Bedingung in die Normalform überführbar sein. Andere Bedingungen sind dabei unerheblich und werden daher nicht überprüft.

Die Entsprechung des Schrittes $t_i \longrightarrow_{\mathcal{R}[\rho_{f,j},p,\sigma]} t_{i+1}$ kann somit die Form

$$t_i^{\perp} \longrightarrow_{\mathcal{R}_{\mathcal{V}}[\rho'_{f,j},p,\sigma']} x_1 \longrightarrow_{\mathcal{R}_{\mathcal{V}}} \ldots \longrightarrow_{\mathcal{R}_{\mathcal{V}}} x_k \longrightarrow_{\mathcal{R}_{\mathcal{V}}[\rho''_{f,j},p,\tau]} t_{i+1}^{\perp}$$

annehmen, wobei in den Schritten von x_1 bis x_k die bedingte Stelle $p \cdot (n+j)$ zu der gewünschten Zielform ausgewertet wird, alle Reduktionen also an oder unterhalb von $p \cdot (n+j)$ erfolgen. Diese Art der Ableitung wird verkürzend als ρ^*-Schritt bezeichnet und mit $t_i^{\perp} \xrightarrow{*}_{\mathcal{R}_{\mathcal{V}}[\rho^*_{f,j},p]} t_{i+1}^{\perp}$ notiert.

Die entscheidenden Teilterme haben dabei die Form

$$\begin{aligned}
t_i^{\perp}|_p &= f(w_1, \ldots, w_n \mid \perp, \ldots, \perp_j, \ldots, \perp) \\
x_1|_p &= f(w_1, \ldots, w_n \mid \perp, \ldots, [\sigma'(u_{f,j}^{\perp}), \sigma'(\vec{x})]_j, \ldots, \perp) \\
x_k|_p &= f(w_1, \ldots, w_n \mid \perp, \ldots, [v_{f,j}^{\perp}, \sigma'(\vec{x})]_j, \ldots, \perp) \\
t_{i+1}^{\perp}|_p &= \tau(r_{f,j}^{\perp})
\end{aligned}$$

Zu zeigen ist, daß mit den gemachten Voraussetzungen eine solche Ableitung auch an t durchgeführt werden kann, so daß t zu dem gesuchten Term v' reduzierbar ist mit $\overline{v'} = \overline{u}$.

Sei t bereits zu einem Term t' mit $\overline{t'} = \overline{t_i}$ und $b(t') \leq b(t)$ reduziert. Angenommen, es sei unmöglich, t' weiter zu einem Term t'' mit $\overline{t''} = \overline{t_{i+1}}$ zu reduzieren. Dann muß in t' an der Stelle $p \cdot (n+j)$ ein Term stehen, der sich nicht zu einem Term reduzieren läßt, der auf das Muster $v_{f,j}^X$ paßt. Laut Voraussetzung sind t' und $\overline{t'}^{\perp} (= \overline{t_i}^{\perp})$ zusammenführbar. Bei dieser Zusammenführung finden Veränderungen nur an oder unterhalb von bedingten Stellen statt. Das kann in bezug auf die Stelle $p \cdot (n+j)$ auf zwei Fälle geschehen:

1. $t'|_{p\cdot(n+j)}$ kann durch $\perp$ ersetzt werden.
   (Die angewandte Transformation ist Viry-ähnlich.) Dann kann aber die Ableitung $t' \xrightarrow{*}_{\mathcal{R}_{\mathcal{V}}[\rho^*_{f,j},p]} t''$ angewendet werden.

2. $t'|_{p\cdot(n+j)}$ und $\sigma'(u_{f,j}^{\perp})$ sind in einem Term c zusammenführbar.
   Nach Konstruktion ist $\sigma'(u_{f,j}^{\perp})$ ein $\perp$-Term, und es gilt somit $\sigma'(u_{f,j}^{\perp}) \xrightarrow{*} \overline{c}^{\perp}$ und (mit der Konfluenz von $\mathcal{R}$ und da $v_{f,j}$ Normalform in $\mathcal{R}$ ist) auch $\overline{c}^{\perp} \xrightarrow{*} v_{f,j}^{\perp}$. Andererseits liefert der Satz von Viry aber damit auch die Ableitung $\overline{t'|_{p\cdot(n+j)}}^{\perp} \xrightarrow{*} v_{f,j}^{\perp}$ via $\overline{c}^{\perp}$. Daß t' nicht zu einem t'' reduzierbar ist, kann also nicht an den unbedingten Stellen von $t'|_{p\cdot(n+j)}$ liegen, sondern es muß noch eine bedingte Stelle unterhalb von $p\cdot(n+j)$ geben, die eine solche Ableitung verhindert. Auf diese Stelle läßt sich aber die bisherige Argumentation erneut anwenden, so daß wiederum auf eine tieferliegende, bedingte Stelle verwiesen werden muß. Als endlicher Term kann t' aber keine unendliche Schachtelung bedingter Stellen besitzen.

Da alle möglichen Fälle betrachtet wurden, ist die Annahme, daß t' nicht zu t'' reduzierbar ist, zum Widerspruch geführt. Zu zeigen ist nur noch, daß $b(t'') \leq b(t')$ gilt. Auch wenn während der Ableitung $t' \xrightarrow{*} t''$ durchaus Terme entstehen können, die eine größere Bedingungstiefe aufweisen, so werden doch beim letzten Schritt $x'_k \longrightarrow_{[\rho'']} t''$ die bedingten Stellen des Terms $t'|_p$ gelöscht. Der dafür ersetzte Term hat nach Definition für $\mathcal{V}$ nur $\perp$-Symbole an neu entstehenden bedingten Stellen. Die Bedingungstiefe $b(t'')$ ist damit echt kleiner als $b(t')$ falls sich die maximale Bedingungstiefe von t' unterhalb einer bedingten Stelle von $t'|_p$ befand, ansonsten gilt $b(t'') = b(t')$.

Nun ergibt sich die Zusammenführbarkeit von $\overline{u}^\perp (= \overline{v}'^\perp)$ und v' in ein v und damit auch die gewünschte Ableitbarkeit von v aus t und $\overline{u}^\perp$ mit $\overline{v} = \overline{u}$. □

Der zweite Hilfssatz geht in seiner Aussage ein Stück weiter und gibt eine Bedingung an, wann zwei Terme, die durch Ableitungen aus einem $\perp$-Term entstanden sind, zusammenführbar sind.

Lemma 3.3: (Zusammenführbarkeit)

Sei $\mathcal{R}$ ein konfluentes, bedingtes Termersetzungssystem, $s, t, u \in T(\mathcal{R}, y \mapsto y^\perp)$ mit $s \xleftarrow{*} \overline{u}^\perp \xrightarrow{*} t$ und $\mathcal{R}_\mathcal{V}$ aus $\mathcal{R}$ durch eine Viry-ähnliche Transformation entstanden.
Gilt für alle $x \in T(\mathcal{R}, y \mapsto y^\perp)$ mit Bedingungstiefe $b(x) \leq \max\{b(s), b(t)\}$ die Zusammenführbarkeit $x \downarrow \overline{x}^\perp$, wobei für alle in den beiden Ableitungen vorkommenden Terme x' gilt $\overline{x}' = \overline{x}$, so gilt auch die Zusammenführbarkeit $s \downarrow t$. ◇

Beweis:
Mit Vollständigkeit und Korrektheit jeder Viry-ähnlichen Transformation und der Konfluenz von $\mathcal{R}$ ergibt sich die Existenz eines v mit den Ableitungen (siehe auch Beweis zu Satz 3.1):

$$\begin{tikzcd} & \overline{u}^\perp \arrow[dl, "*"'] \arrow[dr, "*"] & \\ \overline{s}^\perp \arrow[dr, "*"'] & & \overline{t}^\perp \arrow[dl, "*"] \\ & \overline{v}^\perp & \end{tikzcd}$$

Mit Lemma 3.2 ergibt sich die Existenz von Termen $v_s, v_t \in T(\mathcal{R}, x \mapsto x^\perp)$ mit $s \xrightarrow{*} v_s$ und $t \xrightarrow{*} v_t$ sowie $\overline{v_s} = \overline{v} = \overline{v_t}$.

Eine ähnliche Argumentation wie im Beweis zu Lemma 3.2 zeigt nun die Zusammenführbarkeit von v_s und v_t. Die beiden Terme können höchstens in den bedingten Stellen nicht übereinstimmen. Für diese Stellen gilt die Fallunterscheidung:

1. Beide Terme können so reduziert werden, daß sie an der fraglichen Stelle $\perp$ enthalten, ohne daß ihre unbedingten Stellen dabei verändert werden.

2. Nur einer der Unterterme an der fraglichen Stelle kann mit $\perp$ überschrieben werden. Nach diesem Überschreiben kann der Term mit der entsprechenden ρ'-Regel neu initialisiert werden, sonst könnte der andere Term nicht mit $\overline{v}^\perp$ zusammengeführt werden. Nach dieser Neuinitialisierung kann dann die Argumentation des nächsten Falls auf den neu entstandenen Teilterm angewandt werden, selbst wenn es wiederum möglich sein sollte, diesen Term mit $\perp$ zu überschreiben. Da er neu initialisiert wurde, muß er nämlich dieselben unbedingten Stelleninhalte aufweisen wie der nach Anwendung auf $\overline{v}^\perp$ dort stehende Teilterm und muß daher mit diesem zusammenführbar sein, und es folgt die restliche Argumentation von Fall drei.

3. Beide Terme enthalten an der fraglichen Stelle Unterterme v'_s und v'_t, die nicht mit $\perp$ überschrieben werden können.
   Beide sind laut Voraussetzung mit dem Term zusammenführbar, der bei Anwendung der entsprechenden ρ'-Regel auf den Term $\overline{v}^{\perp}$ an der fraglichen bedingten Stelle entsteht. Iteration der bisher angewendeten Argumentation ergibt, daß v'_s und v'_t in Terme x_s bzw. x_t mit $\overline{x_s} = \overline{x_t}$ reduziert werden können. Dabei gilt $b(x_s) \leq b(v'_s)$ und $b(x_t) \leq b(v'_t)$. Nun geht es nur noch um die bedingten Stellen von x_s und x_t. Auf diese kann nun die gesamte bisherige Argumentation erneut angewandt werden. Da die Bedingungstiefe für jede bedingte Stelle von x_s (x_t) *echt kleiner* als $b(x_s)$ ($b(x_t)$) ist, kommt die Argumentationsiteration notwendigerweise zu einem Ende. Die Terme v'_s und v'_t sind also zusammenführbar.

In allen drei Fällen sind also auch die bedingten Stellen von v_s und v_t zusammenführbar, so daß insgesamt die Ausgangsterme s und t zusammenführbar sind. □

Nachdem der letzte Hilfssatz eine hinreichende Bedingung für die Zusammenführbarkeit zweier Terme beinhaltete, kann nun zu einer weitergehenden Aussage geschritten werden. Der folgende Satz sagt aus, daß die Bedingung „für jeden Term s gilt $s \downarrow \overline{s}^{\perp}$“ hinreichend für die Konfluenz des Gesamtsystems ist. Damit steht ein Kriterium für den Beweis der Konfluenzerhaltung einer Transformation zur Verfügung. Zur Veranschaulichung kann man sich davon überzeugen, daß die Aussage für das Beispiel 2.5 und damit für die Transformation von Viry in bezug auf alle bedingten Termersetzungssysteme nicht gilt.

Diese Erkenntnis wird im weiteren dazu verwandt, die Konfluenzerhaltung der Transformation von Viry auf der Klasse der konstruktorbasierten bedingten Termersetzungssysteme (Abschnitt 3.3) und der Transformation $\mathcal{B}$ (Abschnitt 3.4) zu beweisen.

Satz 3.2: (Hinreichende Konfluenzbedingung)
Sei $\mathcal{R}$ ein bedingtes Termersetzungssystem und $\mathcal{R}_{\mathcal{V}}$ aus $\mathcal{R}$ durch eine Viry-ähnliche Transformation entstanden.
Gilt für alle Terme $t \in T(\mathcal{R}, x \mapsto x^{\perp})$ die Zusammenführbarkeit $t \downarrow \overline{t}^{\perp}$, wobei für alle in den beiden Ableitungen vorkommenden Terme s gilt $\overline{s} = \overline{t}$, so ist auch der sinntragende Teil von $\mathcal{R}_{\mathcal{V}}$ konfluent. ◇

Beweis:

Der Satz ist eine direkte Folgerung aus Lemma 3.3. □

Im letzten Abschnitt dieses Kapitels wird dargelegt, daß diese Erkenntnis möglicherweise für alle ähnlichen Transformationen „höherer“ Termersetzungssysteme in Standardsysteme angewendet werden kann.

3.3 Konfluenzerhaltung in konstruktorbasierten Systemen

Nachdem also nun eine hinreichende Bedingung für die Konfluenzerhaltung gefunden wurde, kann nunmehr ein Beweis für die Konfluenzerhaltung für $\mathcal{V}$ eingeschränkt auf eine bestimmte Klasse bedingter Termersetzungssysteme gewagt werden. Wie sich herausgestellt hat, reicht es aus, daß alle Terme mit dem zugehörigen $\perp$-Term zusammenführbar sind, um die Konfluenzerhaltung der Transformation zu gewährleisten.

In diesem Abschnitt wird bewiesen, daß $\mathcal{V}$ konfluenzerhaltend auf der Klasse der konstruktorbasierten Termersetzungssysteme (Definitionen 1.2.13 und 1.4.3) ist. Dazu wird bewiesen, daß das Ergebnis der Transformation eines bedingten, konstruktorbasierten Systems ebenfalls konstruktorbasiert ist.

Lemma 3.4: (Erhaltung der Konstruktorbasiertheit)
Sei $\mathcal{R}$ ein bedingtes, konstruktorbasiertes Termersetzungssystem.
Dann ist auch $\mathcal{R}_{\mathcal{V}}$ konstruktorbasiert. $\diamond$

Beweis:

Sei $\Sigma = \mathcal{D} \uplus \mathcal{C}$ die laut Definition 1.4.3 für konstruktorbasierte Termersetzungssysteme geforderte Signaturzerlegung. Dann läßt sich entsprechend Definition 2.4.1 $\Sigma_{\mathcal{V}}$ zerlegen in $\Sigma_{\mathcal{V}} = \mathcal{D}_{\mathcal{V}} \uplus \mathcal{C}_{\mathcal{V}}$, wobei sich $\mathcal{D}_{\mathcal{V}}$ von $\mathcal{D}$ nur durch die Stelligkeiten unterscheidet und $\mathcal{C}_{\mathcal{V}} = \mathcal{C} \cup \perp^{(0)}$ ist.

Anstatt der $m + k$ Regeln

$$\begin{array}{lcll} \rho_{f,i} & : & f(t_{f,i1}, \ldots, t_{f,in}) \longrightarrow r_{f,i} \; if \; u_{f,i} \xrightarrow{*} v_{f,i}, & 1 \leq i \leq m \\ \rho_{f,j} & : & f(t_{f,j1}, \ldots, t_{f,jn}) \longrightarrow r_{f,j}, & m < j \leq m + k \end{array}$$

aus P, die $f^{(n)} \in \Sigma$ definieren, enthält $P_{\mathcal{V}}$ ausschließlich Regeln der Form

$$\begin{array}{l} \rho'_{f,i} : f(t^X_{f,i1}, \ldots, t^X_{f,in} \mid y_1, \ldots, \perp_i, \ldots, y_m) \longrightarrow f(t^X_{f,i1}, \ldots, t^X_{f,in} \mid y_1, \ldots, [u^{\perp}_{f,i}, (\vec{x})]_i, \ldots, y_m) \\ \rho''_{f,i} : f(z_1, \ldots, z_n \mid y_1, \ldots, [v^X_{f,i}, (\vec{x})]_i, \ldots, y_m) \longrightarrow r^{\perp}_{f,i}, \quad 1 \leq i \leq m \\ \rho''_{f,j} : f(t^X_{f,j1}, \ldots, t^X_{f,jn} \mid y_1, \ldots, y_m) \longrightarrow r^{\perp}_{f,j}, \qquad\qquad m < j \leq m + k \end{array}$$

für $f^{(n+m)} \in \Sigma_{\mathcal{V}}$.

Da $\perp$ ein Konstruktorsymbol ist, gilt nach Definition 2.4.3 für Σ-Konstruktorterm t, daß $t^{\perp}$ ein $\Sigma_{\mathcal{V}}$-Konstruktorterm ist. Ebenso gilt für jedes Σ-Muster und jeden Σ-Konstruktorterm t, daß t^X ein $\Sigma_{\mathcal{V}}$-Muster ist (Definition 2.4.2). Laut Voraussetzung ist $\mathcal{R}$ konstruktorbasiert und damit sind alle $t_{f,ij}$ Muster und $v_{f,i}$ Konstruktorterme.

Für jede Regel $\rho : f(t_1, \ldots t_n) \longrightarrow r \in P_{\mathcal{V}}$ gilt daher: $f \in \mathcal{D}_{\mathcal{V}}$ und t_i ist Muster für alle $i \in \{1, \ldots, n\}$.

Somit ist $\mathcal{R}_{\mathcal{V}}$ konstruktorbasiert. $\square$

Nun schließt sich direkt der zentrale Satz an, der besagt, daß die Transformation konfluenzerhaltend ist.

Satz 3.3: (Konfluenzerhaltung für konstruktorbasierte Systeme)
$\mathcal{V}$ ist konfluenzerhaltend in bezug auf die Klasse der konstruktorbasierten, bedingten Termersetzungssysteme. $\diamond$

Beweis:
Nach Satz 3.2 ist zu zeigen, daß für alle Terme $t \in T(\mathcal{R}, x \mapsto x^{\perp})$ die Bedingung $t \downarrow \bar{t}^{\perp}$ gilt, wobei alle an den beiden Ableitungen beteiligten Terme dasselbe Bild bezüglich der Abbildung $\overline{x}$ haben.

Dies wird bewiesen durch vollständige Induktion über die Bedingungstiefe b.

Induktionsanker; die Behauptung gilt für alle $\perp$-Terme t, also für $b = 1$, da dann $t = \bar{t}^{\perp}$.

Induktionsschritt; gilt die Behauptung für alle sinntragenden Terme mit Bedingungstiefe $\leq b$, so auch für den Term $t \in T(\mathcal{R}, x \mapsto x^{\perp})$ mit $b(t) = b + 1$:

Sei $t|_p = f(t_1 \ldots t_n \mid \ldots, u, \ldots)$ und die Stelle von u sei q. Dann gilt für u einer der folgenden Fälle:

1. $u = \perp$
   u steht der Zusammenführung von t und $\overline{t}^{\perp}$ nicht im Wege.

2. $u \neq \perp$
   Da ohne Modifikation der Regeln nicht die Möglichkeit besteht, bedingte Stellen erneut mit $\perp$ zu überschreiben, ist zu zeigen, daß u mit dem Term u' zusammenführbar ist, der durch Anwendung der Regel $\rho'_{f,j}$ auf $\overline{t}^{\perp}$ an der Stelle q entsteht. Dazu ist zunächst zu bemerken, daß diese Regel anwendbar sein muß, denn das Muster auf der linken Seite der Regel muß auf einen Vorgänger von t anwendbar gewesen sein, sonst wäre t an der Stelle q gleich $\perp$. Danach aber müssen die Teile des Terms, die ein Passen auf das Muster ermöglichten, unverändert geblieben sein, da sie laut Definition 1.2.13 aus Konstruktorsymbolen bestehen.

Zu zeigen ist also $u \downarrow u'$. Dazu wird zunächst untersucht, wie es zu der Initialisierung der Stelle von u in der Ableitung von t aus einem $\perp$-Term gekommen ist. Eine solche Ableitung muß existieren, da t der Menge der sinntragenden Terme entstammt.

$$\overline{s}^{\perp} \xrightarrow{*} t' \text{ (mit } t'|_{q'} = \perp) \longrightarrow_{[\rho'_{f,j},\ldots,\sigma]} t'' \text{ (mit } t''|_{q'} = [\sigma(u_{n+j}), \sigma(\vec{x})]) \xrightarrow{*} t$$

wobei q' die Vorgängerposition von q ist, das heißt daß $t''|_{q'}$ genau der Term ist, der im weiteren Verlauf der Ableitung zu u reduziert wird.

Nun kommt eine wichtige Eigenschaft konstruktorbasierter Systeme zum Tragen. Da die Muster $l_1 \ldots l_n$ der Regel $\rho'_{f,j}$ nur aus Konstruktorsymbolen bestehen dürfen (Definition 1.2.13), bleiben diese Teile des Terms unverändert. Spätere Reduktionen an den unbedingten Stellen können nur unterhalb dieser Muster durchgeführt werden. Das bedeutet, daß derartige Reduktionen aber dann unterhalb von $[\sigma(u_{n+j}), \sigma(\vec{x})]$ *ebenfalls durchgeführt werden können.* Daher gilt, daß $[\sigma(u_{n+j}), \sigma(\vec{x})]$ nicht nur zu u, sondern auch zu dem Term v ableitbar ist, der an Stelle von u entsteht, wird $\rho'_{f,j}$ auf $t[q \longleftarrow \perp]$ angewendet, also die bedingte Stelle q neu initialisiert.

Es ergeben sich also die Ableitungen

$$\begin{array}{ccccc} & & [\sigma(u_{n+j}), \sigma(\vec{x})] & & \\ & \overset{*}{\swarrow} & & \overset{*}{\searrow} & \\ v & & & & u \end{array}$$

Mit der Korrektheit und Vollständigkeit von $\mathcal{V}$ sowie der Konfluenz von $\mathcal{R}$ existiert ein z mit den Ableitungen

$$\begin{array}{ccccc} & & [\overline{\sigma(u_{n+j})}^{\perp}, \overline{\sigma(\vec{x})}^{\perp}] & & \\ & \overset{*}{\swarrow} & & \overset{*}{\searrow} & \\ \overline{v}^{\perp} & & & & \overline{u}^{\perp} \\ & \overset{*}{\searrow} & & \overset{*}{\swarrow} & \\ & & \overline{z}^{\perp} & & \end{array}$$

Da es sich bei $\sigma(u_{n+j})$ und den $\sigma(\vec{x})$ um sinntragende Terme handelt, gibt es auch „Ursprungs"-$\perp$-Terme $[\overline{s_1}^{\perp}, \overline{(\vec{s})}^{\perp}]$, aus denen sie herleitbar sind. Daher ergibt sich das Gesamtbild von Ableitungen, welche die Voraussetzungen zur Anwendung von Lemma 3.2.3 bilden.

$$\begin{tikzcd}[column sep=small] & {[\overline{s_1}^{\perp}, \overline{(\vec{s})}^{\perp}]} \arrow[dl, "*"'] \arrow[dr, "*"] & \\ \overline{v}^{\perp} & & u \end{tikzcd}$$

Da für die Bedingungstiefe $b(\overline{v}^{\perp}) = 1 < b(u)$ gilt, folgt mit Lemma 3.2.3 das gewünschte Ergebnis $\overline{v}^{\perp} \downarrow u$ und insgesamt $t \downarrow \overline{t}^{\perp}$. □

Nun kann das Ergebnis dieses Satzes in zwei Richtungen interpretiert werden. Zum einen wurde gezeigt, daß das Zielsystem der Transformation $\mathcal{V}$ ohne Reinitialisierungen der bedingten Stellen mit $\perp$ auskommt. Das bedeutet nichts anderes, als daß bei der Auswertung der Bedingungen *kein Backtracking* benötigt wird. Sobald es in einem konstruktorbasierten System möglich ist, eine Bedingung zu initialisieren, ist es in der weiteren Ableitung nie wieder nötig, diese Entscheidung zu revidieren.

Es gibt noch eine zweite wichtige Erkenntnis aus dem Beweis. Da, wie gezeigt, alle Veränderungen unterhalb der unbedingten Stellen sich auch auf die bedingten Stellen übertragen lassen, ist es *unnötig, den Kontext in* $(\vec{x})$ *zwischenzuspeichern*. Die Transformation kann somit vereinfacht werden, um den speziellen Eigenschaften konstruktorbasierter Systeme Rechnung zu tragen. Weitere Anpassungen dieser Art werden noch in Kapitel 4 gemacht.

3.4 Konfluenzerhaltung im Allgemeinen

Nachdem im vorigen Abschnitt gezeigt wurde, daß die Transformation von Viry für konstruktorbasierte Termersetzungssysteme konfluenzerhaltend ist, wird in diesem Abschnitt das Verfahren so modifiziert, daß diese Eigenschaft auf allen Termersetzungssystemen erhalten bleibt.

Die Grundidee hierzu entspringt der weiter oben geleisteten Analysearbeit. Um Konfluenz zu gewährleisten, reicht es, daß alle Terme t mit dem zugehörigen $\perp$-Term $\overline{t}^{\perp}$ zusammenführbar sind. Dies ist, wie gezeigt, für den konstruktorbasierten Teil eines Termersetzungssystems unproblematisch, für andere Teile kann es erforderlich sein, bedingte Stellen erneut mit $\perp$ überschreiben zu können.

Da mit dem hier vorgestellten Verfahren die konstruktorbasierten Teile des zu transformierenden Termersetzungssystems berücksichtigt werden, handelt es sich um eine konservative Erweiterung des Ausgangsverfahrens - auf rein konstruktorbasierten Systemen entstehen keinerlei zusätzliche Ableitungsmöglichkeiten.

Die Definition geht wie folgt vor, zuerst wird die Signatur in definierende Funktionssymbole, Konstruktorsymbole und andere Symbole unterteilt. Dann werden die Entsprechungen zu den auch vorher schon vorhandenen Regeln definiert, danach die neu hinzukommenden Regeln angegeben.

Definition 3.4.1: (Transformation $\mathcal{B}$)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem und $\mathcal{R}_{\mathcal{V}} = \langle \Sigma_{\mathcal{V}}, X, \mathrm{P}_{\mathcal{V}} \rangle$ das aus $\mathcal{R}$ durch die Transformation $\mathcal{V}$ entstandene System.

Es gelte $\Sigma_{\mathcal{V}} = \mathcal{D} \uplus \mathcal{C} \uplus \mathcal{A}$, so daß $\mathcal{D}$ genau die Funktionssymbole aus Σ enthält, die in linken Regelseiten nur als Kopfsymbol auftauchen und $\mathcal{C}$ genau diejenigen Funktionssymbole aus Σ, die nie als Kopfsymbol in einer linken Regelseite auftauchen.

Dann entsteht durch die Transformation $\mathcal{B}$ das unbedingte Termersetzungssystem $\mathcal{R}_{\mathcal{B}} = \langle \Sigma_{\mathcal{B}}, X, \mathrm{P}_{\mathcal{B}} \rangle$, welches wie folgt definiert ist:

Für jede ρ'-Regel aus $\mathrm{P}_\mathcal{V}$ der Form

$$f(l_1, \ldots, l_n \mid \ldots, \perp, \ldots) \longrightarrow f(l_1, \ldots, l_n \mid \ldots, [c, (\vec{x})], \ldots)$$

enthält $\mathrm{P}_\mathcal{B}$ eine Regel der Form

$$f(k(l_1, Y_1), \ldots, k(l_n, Y_n) \mid \ldots, \perp, \ldots) \longrightarrow f(k(l_1, Y_1), \ldots, k(l_n, Y_n) \mid \ldots, \{[k(c, 0), (\vec{x})], 0\}, \ldots)$$

wobei $Y_i \in X^*$ mit $|Y_i|$ ist gleich der Anzahl der Funktionssymbole mit Ausnahme nullstelliger Konstruktorsymbole in l_i und für alle Variablen in $Y_1 \cdot \ldots \cdot Y_n$ gilt, daß diese paarweise unterschiedlich sind und in keinem der Terme $l_1, \ldots, l_n$ vorkommen.

Für jede ρ''-Regel $f(l_1, \ldots, l_n) \longrightarrow r$ aus $\mathrm{P}_\mathcal{V}$ enthält $\mathrm{P}_\mathcal{B}$ eine Regel

$$k(f(l_1, \ldots, l_n), x_1 \cdot \ldots \cdot x_j) \longrightarrow \begin{cases} k(r, 0) & : \ f \in \mathcal{D} \ und \ l_i \in \mathcal{T}_\mathcal{C}(X) \ mit \ i \in \{1, \ldots, n\} \\ k(r, 1) & : \ sonst \end{cases}$$

wobei j gleich der Anzahl der Funktionssymbole mit Ausnahme nullstelliger Konstruktorsymbole in $f(l_1, \ldots, l_n)$ ist und alle Variablen in $x_1 \cdot \ldots \cdot x_j$ paarweise unterschiedlich sind.

Für jedes $f^{(n+m)} \in \Sigma_\mathcal{V}$ mit n unbedingten, m bedingten Stellen enthält $\mathrm{P}_\mathcal{B}$ die n Regeln

$$\{f(\ldots\{x_i, 1\}\ldots \mid x_{n+1}, \ldots, x_{n+m}), y\} \quad \longrightarrow \quad \{f(\ldots\{x_i, 0\}\ldots \mid \perp, \ldots, \perp), 1\}$$

für $i \in \{1, \ldots, n\}$

Außerdem enthält $\mathrm{P}_\mathcal{B}$ die Regeln

$$\begin{array}{rcl} \langle\{x, 1\}\rangle & \longrightarrow & \langle\{x, 0\}\rangle \\ \{[\{x, 1\}, y], 0\} & \longrightarrow & \{[\{x, 0\}, y], 0\} \\ \{[x, (\ldots, \{y, 1\}, \ldots)], 0\} & \longrightarrow & \{[x, (\ldots, \{y, 0\}, \ldots)], 0\} \end{array}$$

In die neue Signatur sind die geschweiften und die Winkelklammern aufzunehmen. Diese haben nun den Charakter von Funktionssymbolen, da, genau betrachtet, alle zusätzlich hinzugefügten Regeln in $\mathrm{P}_\mathcal{B}$ Regeln für diese Symbole sind.

$$\Sigma_\mathcal{B} = \Sigma_\mathcal{V} \cup \{\langle\rangle^{(1)}, \{\}^{(2)}\}$$

$\diamond$

Definition 3.4.2: (Abbildung k)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P}\rangle$ ein bedingtes Termersetzungssystem und $\Sigma_\mathcal{V}$ wie in Definition 3.4.1 unterteilt in $\mathcal{D} \uplus \mathcal{C} \uplus \mathcal{A}$. Dann ist die Abbildung $k: \ \mathcal{T}_{\Sigma_\mathcal{V}}(X) \ \times \ (\{0,1\} \cup X^*) \longrightarrow \mathcal{T}_{\Sigma_\mathcal{B}}(X)$ definiert durch

$$k(t, x) = \begin{cases} t & : \ t \in X \ oder \ t^{(0)} \in \mathcal{C} \\ \{f(k(t_1, 0), \ldots, k(t_n, 0)), x\} & : \ x \in \{0, 1\} \ und \ t = f(t_1, \ldots, t_n) \\ \{f(k(t_1, Y_1), \ldots, k(t_n, Y_n)), y\} & : \ x = Y_1 \cdot \ldots \cdot Y_n \cdot y \ und \ t = f(t_1, \ldots, t_n) \end{cases}$$

wobei $|Y_i|$ gleich der Anzahl der Funktionssymbole mit Ausnahme nullstelliger Konstruktorsymbole in t_i ist. $\diamond$

Bevor nun die formalen Aspekte der Transformation $\mathcal{B}$ betrachtet werden, soll sie zunächst an dem Beispiel 2.5 durchgeführt werden, für welches die Transformation $\mathcal{V}$ die Konfluenz nicht erhalten konnte.

Beispiel 3.3

$$\mathcal{R} = \begin{pmatrix} 1: & f(g(x)) & \longrightarrow & h(x) & if\ c(x) \xrightarrow{*} true \\ 2: & c(a) & \longrightarrow & true & \\ 3: & g(g(x)) & \longrightarrow & g(x) & \end{pmatrix}$$

$\mathcal{R}$ ist konfluent.

Es ist von Vorteil, die Transformation des Systems ausführlich darzustellen. Daher zunächst das bereits weiter oben angegebene Ergebnissystem der Transformation $\mathcal{V}$:

$$\mathcal{R}_{\mathcal{V}} = \begin{pmatrix} \rho'_{f,1}: & f(g(x) \mid \bot) & \longrightarrow & f(g(x) \mid [c(x), (x)]) \\ \rho''_{f,1}: & f(z_1 \mid [true, (x)]) & \longrightarrow & h(x) \\ \rho''_{c,1}: & c(a) & \longrightarrow & true \\ \rho''_{g,1}: & g(g(x)) & \longrightarrow & g(x) \end{pmatrix}$$

Die Aufteilung der Signatur entsprechend der Definition 3.4.1 ist nun

$$\Sigma_{\mathcal{V}} = \{f^{(1+1)}, c^{(1)}\}(=: \mathcal{D}) \uplus \{a^{(0)}, true^{(0)}, h^{(1)}\}(=: \mathcal{C}) \uplus \{g^{(1)}\}(=: \mathcal{A})$$

Die Erstellung der neuen Regelmenge kann in drei Schritte unterteilt werden.

1. Transformation der Regeln aus $\mathcal{R}_{\mathcal{V}}$.
2. Erstellung der Regeln zur Reinitialisierung der bedingten Stellen.
3. Hinzufügen des konstanten Teils.

Der erste Schritt ergibt laut Definition die vier Regeln

$$\begin{array}{lllll} 1: & f(k(g(x), x_1) \mid \bot) & \longrightarrow & f(k(g(x), x_1) \mid [k(c(x), 0), (x)]) \\ 2: & k(f(z_1 \mid [true, (x)]), x_1) & \longrightarrow & k(h(x), 0) \\ 3: & k(c(a), x_1) & \longrightarrow & k(true, 0) \\ 4: & k(g(g(x)), x_1 \cdot x_2) & \longrightarrow & k(g(x), 1) \end{array}$$

Anmerkungen:

- Die rechten Regelseiten der Regeln zwei und drei enthalten im zweiten Argument für k eine Null. Das bedeutet, daß die linke Seite der entsprechenden Regel den Forderungen der Konstruktorbasiertheit entsprechen. Die vierte Regel hat hingegen eine Eins als zweites Argument, da das Funktionssymbol g aus $\mathcal{A}$ stammt.
- Nur in der vierten Regel treten mehrere Variablen für das zweite Argument von k auf, da nur hier eine Schachtelung mehrstelliger Funktionssymbole auf der linken Seite einer ρ'' Regel auftauchen.

Als Beispiel für die Berechnung der k-Funktion soll die linke Seite der vierten Regel hier ausführlich dargestellt werden.

$$k(g(g(x)), x_1 \cdot x_2) = \{g(k(g(x), x_1)), x_2\} = \{g(\{g(k(x, \varepsilon)), x_1\}), x_2\} = \{g(\{g(x), x_1\}), x_2\}$$

Das Ergebnis nach Anwendung der k-Funktion ist

$$\begin{array}{rlcl}
1: & f(\{g(x), x_1\} \mid \bot) & \longrightarrow & f(\{g(x), x_1\} \mid \{[\{c(x), 0\}, (x)], 0\}) \\
2: & \{f(z_1 \mid \{[true, (x)], x_1\}), x_2\} & \longrightarrow & \{h(x), 0\} \\
3: & \{c(a), x_1\} & \longrightarrow & true \\
4: & \{g(\{g(x), x_1\}), x_2\} & \longrightarrow & \{g(x), 1\}
\end{array}$$

Anmerkungen:

- Die y der $\{x, y\}$-Terme werden im folgenden als „Flags“ bezeichnet.
- Über die []- und ()-Klammern sowie über das |-Symbol wird in der Funktion k hinweggegangen, da es sich nur um anschauliche Mittel handelt (siehe Regel Nummer zwei).

Im zweiten Schritt werden nun die Regeln für das „Hochreichen“ der Einser-Flags und die damit gegebenenfalls verbundene Reinitialisierung der bedingten Stellen angegeben. Für jedes Funktionssymbol aus $\Sigma_{\mathcal{V}}$ gibt es so viele Regeln, wie es unbedingte Stellen besitzt. In diesem Fall sind das insgesamt vier derartige Regeln. Stünden für diese Aufgabe Funktionen höherer Ordnung oder auch die Curryfizierung der Argumente zur Verfügung, ließe sich diese Aufgabe mit einem wesentlich kleineren Anteil zusätzlicher Regeln erledigen, denn die Struktur ist bei allen Regeln sehr ähnlich.

$$\begin{array}{rlcl}
5: & \{f(\{x_1, 1\} \mid x_2), y\} & \longrightarrow & \{f(\{x_1, 0\} \mid \bot), 1\} \\
6: & \{c(\{x_1, 1\}), y\} & \longrightarrow & \{c(\{x_1, 0\}), 1\} \\
7: & \{g(\{x_1, 1\}), y\} & \longrightarrow & \{g(\{x_1, 0\}), 1\} \\
8: & \{h(\{x_1, 1\}), y\} & \longrightarrow & \{h(\{x_1, 0\}), 1\}
\end{array}$$

Mit dem Anfügen des konstanten Teils als dritten Schritt ergibt sich das folgende Gesamtsystem:

$$\mathcal{R}_{\mathcal{B}} = \left(\begin{array}{rlcl}
1: & f(\{g(x), x_1\} \mid \bot) & \longrightarrow & f(\{g(x), x_1\} \mid \{[\{c(x), 0\}, (x)], 0\}) \\
2: & \{f(z_1 \mid \{[true, (x)], x_1\}), x_2\} & \longrightarrow & \{h(x), 0\} \\
3: & \{c(a), x_1\} & \longrightarrow & true \\
4: & \{g(\{g(x), x_1\}), x_2\} & \longrightarrow & \{g(x), 1\} \\
 & & & \\
5: & \{f(\{x_1, 1\} \mid x_2), y\} & \longrightarrow & \{f(\{x_1, 0\} \mid \bot), 1\} \\
6: & \{c(\{x_1, 1\}), y\} & \longrightarrow & \{c(\{x_1, 0\}), 1\} \\
7: & \{g(\{x_1, 1\}), y\} & \longrightarrow & \{g(\{x_1, 0\}), 1\} \\
8: & \{h(\{x_1, 1\}), y\} & \longrightarrow & \{h(\{x_1, 0\}), 1\} \\
 & & & \\
9: & \langle\{x, 1\}\rangle & \longrightarrow & \langle\{x, 0\}\rangle \\
10: & \{[\{x, 1\}, y], 0\} & \longrightarrow & \{[\{x, 0\}, y], 0\} \\
11: & \{[x, (\{y, 1\})], 0\} & \longrightarrow & \{[x, (\{y, 0\})], 0\}
\end{array}\right)$$

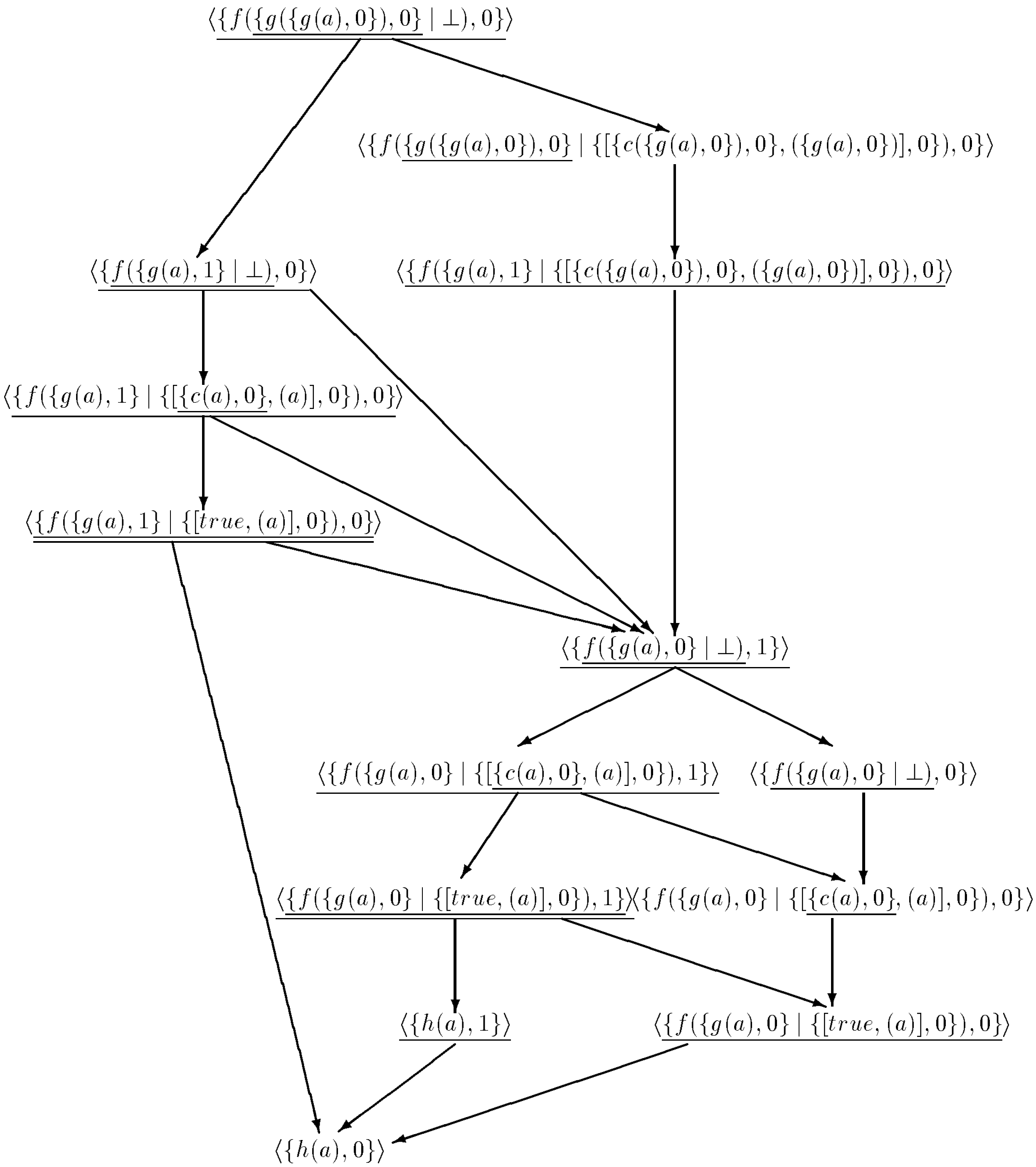
Anmerkungen zum konstanten Teil:

- Die Winkelklammern in Regel Nummer neun sind nur dazu da, das äußere Ende des Terms zu markieren, wie sich weiter unten noch deutlicher zeigen wird. Ist eine Eins bis „dort oben“ vorgedrungen, so kann sie gelöscht werden. Sie muß sogar gelöscht werden, um Konfluenz zu gewährleisten, siehe unten.

- Die Regeln zehn und elf behandeln die eckigen Klammern der bedingten Stellen, die hier nicht übergangen werden können, obwohl sie keine eigentliche Funktion haben.

Als nächstes wird gezeigt, wie die Ableitung, die nach der ursprünglichen Transformation des Systems zwei Normalformen hatte, nun ein eindeutiges Ergebnis liefert. Startterm ist statt $s \in \mathcal{T}_\Sigma(X)$ nun $\langle k(s^\perp, 0)\rangle$.

Beispiel 3.4



Nun geht es um die formalen Eigenschaften der Transformation. Bei der Betrachtung der Bezüge zwischen den Systemen sind, wie schon häufiger erwähnt und auch im nächsten

Abschnitt noch einmal angesprochen, zwei Abbildungen von besonderer Bedeutung: Die Start- und die Endtermzuordnung.

Starttermzuordnung ist, wie gesehen, $x \mapsto \langle k(x^\perp, 0)\rangle$. Entgegen der Forderung ist dies, wegen der Winkelklammern, *kein Homomorphismus*. Dennoch legen die bisherigen Überlegungen, und besonders die im nächsten Abschnitt, nahe, daß auch eine Abbildung mit dieser Eigenschaft existieren müßte. In der Tat wäre eine homomorphe Abbildung mit Gewährleistung der Konfluenzerhaltung eine solche, die, nach Anwendung von k, allen Flags den Wert Eins zuordnete. Dies würde aber zu vielen unnötigen Ableitungen führen, weshalb hier eine „Abkürzung“ mit Hilfe der Winkelklammern gewählt wurde. Auch müßte mit einer solchen Abbildung die, schon bei Einführung erwogene, schwächere Version einer Definition für die Vollständigkeit einer Transformation gewählt werden, so daß zu jeder Ableitung $s \xrightarrow{*} t$ im Ausgangssystem eine Ableitung $s' \xrightarrow{*} t'$ existiert, bei der zwar, wie gewohnt, $s' = H(s)$, aber statt $t' = H(t)$ die Bedingung $G(t') = t$ angegeben werden muß (siehe Definition 1.5.4).

Endtermzuordnung ist $x \mapsto \overline{e(x)}$, wobei e definiert ist durch

Definition 3.4.3: (Abbildung $e: \mathcal{T}_{\Sigma_\mathcal{B}}(X) \longrightarrow \mathcal{T}_{\Sigma_\mathcal{V}}(X)$)

$$e(t) = \begin{cases} t & : \ t \in X \text{ oder } t^{(0)} \in \Sigma_\mathcal{V} \\ f(e(t_1), \ldots, e(t_n)) & : \ t = \{f(t_1, \ldots, t_n), x\} \\ x & : \ t = \langle x \rangle \end{cases}$$

◇

Diese Abbildung erfüllt die gestellten Forderungen; alle sinntragenden Terme sind abbildbar und für alle Terme $t \in \mathcal{T}_\Sigma(X)$ gilt $\overline{e(\langle k(t^\perp, 0)\rangle)} = t$.

Nach Definition der Termzuordnungen kann nun der Satz für die Korrektheit und Vollständigkeit von $\mathcal{B}$ erfolgen.

Satz 3.4: (Korrektheit und Vollständigkeit von $\mathcal{B}$)
Die Transformation $\mathcal{B}$ mit Starttermzuordnung $x \mapsto \langle k(x^\perp, 0)\rangle$ und Endtermzuordnung $x \mapsto \overline{e(x)}$ ist korrekt und vollständig bezüglich der gesamten Klasse der bedingten Termersetzungssysteme. ◇

Beweis:
Zu zeigen ist, daß für jedes bedingte Termersetzungssystem $\mathcal{R}$ gilt

1. Aus $s \xrightarrow{*}_\mathcal{R} t$ folgt die Existenz einer Ableitung $\langle k(s^\perp, 0)\rangle \xrightarrow{*}_{\mathcal{R}_\mathcal{B}} \langle k(t^\perp, 0)\rangle$
2. Aus $\langle k(s^\perp, 0)\rangle \xrightarrow{*}_{\mathcal{R}_\mathcal{B}} t$ folgt die Existenz einer Ableitung $\overline{e(\langle k(s^\perp, 0)\rangle)} \xrightarrow{*}_\mathcal{R} \overline{e(t)}$

1.)
$\mathrm{P}_\mathcal{B}$ enthält für jede Regel $\rho_\mathcal{V} : l_\mathcal{V} \longrightarrow r_\mathcal{V} \in \mathrm{P}_\mathcal{V}$ eine Regel $\rho_\mathcal{B} : l_\mathcal{B} \longrightarrow r_\mathcal{B}$ mit $e(l_\mathcal{B}) = l_\mathcal{V}$ und $e(r_\mathcal{B}) = r_\mathcal{V}$, wobei in $l_\mathcal{B}$ nur Variablen an den Positionen der Flags stehen. Es folgt also aus $x \longrightarrow_{\mathcal{R}_\mathcal{V}[\rho_\mathcal{V}]} y$ auch für jedes x' mit $e(x') = x$ die Reduzierbarkeit $x' \longrightarrow_{\mathcal{R}_\mathcal{B}[\rho_\mathcal{B}]} y'$ mit $e(y') = y$.
Damit gilt insbesondere für jede Ableitung $s^\perp \xrightarrow{*}_\mathcal{V} t^\perp$, daß auch $\langle k(s^\perp, 0)\rangle \xrightarrow{*}_{\mathcal{R}_\mathcal{B}} \langle k(t^\perp, 0)\rangle$ gilt. Da $\mathcal{V}$ mit $x \mapsto x^\perp$ vollständig ist, folgt damit auch die Vollständigkeit von $\mathcal{B}$.

2.)
Die neu hinzugefügten Regeln in $\mathrm{P}_\mathcal{B}$ verändern nur die Flags und die bedingten Stellen eines Terms. Es gilt also für jeden sinntragenden Term s mit $s \longrightarrow_{\mathcal{B}[\rho]} t$ mit neuer Regel ρ, daß $\overline{e(s)} = \overline{e(t)}$ ist. Da außerdem für jede einer Regel $\rho_\mathcal{V}$ aus $\mathrm{P}_\mathcal{V}$ entsprechende Regel $\rho_\mathcal{B}$ gilt,

daß aus $s \longrightarrow_{\mathcal{B}[\rho_{\mathcal{B}}]} t$ auch $e(s) \longrightarrow_{\mathcal{V}[\rho_{\mathcal{V}}]} e(t)$ folgt und $\mathcal{R}_{\mathcal{V}}$ mit $x \mapsto \overline{x}$ korrekt ist, folgt damit insgesamt auch die Korrektheit von $\mathcal{B}$. □

Dieses Ergebnis war zu erwarten, handelt es sich doch bei $\mathcal{B}$ um eine Viry-ähnliche Transformation. Wesentlich interessanter ist nun aber die neu hinzugewonnene Eigenschaft der allgemeinen Konfluenzerhaltung. Das Beispiel 3.4 hat schon gezeigt, daß $\mathcal{B}$ zumindest für dieses System ein besseres Ergebnis als $\mathcal{V}$ erzielt. Der folgende Satz sagt aus, daß dies auch ganz allgemein gilt.

Satz 3.5: (Allgemeine Konfluenzerhaltung von $\mathcal{B}$)
$\mathcal{B}$ ist konfluenzerhaltend bezüglich der Klasse aller bedingten Termersetzungssysteme. ◇

Beweis:
Nach Satz 3.2 ist zu zeigen, daß für alle Terme $t \in T(\mathcal{R}, x \mapsto k(x^{\perp}, 0))$ die Bedingung $\langle k(\overline{e(t)}^{\perp}, 0)\rangle \downarrow_{\mathcal{R}_{\mathcal{B}}} \langle t\rangle$ gilt.

Dies wird bewiesen durch vollständige Induktion über die Bedingungstiefe.

Induktionsanker; die Behauptung gilt für alle t mit $b(e(t)) = 1$, also für die $e(t)$ ein $\perp$-Term ist, da die zusätzlichen Regeln in $\mathrm{P}_{\mathcal{B}}$ es erlauben, eventuell in t enthaltene Einsen zu eliminieren, so daß gilt:

$$\langle t\rangle \xrightarrow{*} \langle k(e(t), 0)\rangle$$

Induktionsschritt; gilt die Behauptung für alle sinntragenden Terme Bedingungstiefe $\leq b$, so auch für den Term $t \in T(\mathcal{R}, x \mapsto k(x^{\perp}, 0))$ mit $b(e(t)) = b + 1$:

Sei $t|_p = \{f(t_1, \ldots, t_n \mid \ldots, u, \ldots), 0\}$ und die Stelle von u sei q.

Dann gilt einer der folgenden Fälle:

1. Eines der Flags in den unbedingten Stellen von $t_1, \ldots, t_n$ enthält eine Eins.
   Dann ist u mit den zusätzlichen Regeln in $\mathrm{P}_{\mathcal{B}}$ durch $\perp$ ersetzbar. Da durch diese Regeln in $t_1, \ldots, t_n$ ebenfalls nur bedingte Stellen durch $\perp$ ersetzt werden, gilt die Behauptung für diese auch danach noch.
   Bei dieser Reinitialisierung kann eine Eins unter Umständen „ganz oben" ankommen. Diese kann durch die Regel $\langle\{x, 1\}\rangle \longrightarrow \langle\{x, 0\}\rangle$ gelöscht werden.

2. Alle Flags in den unbedingten Stellen von $t_1, \ldots, t_n$ enthalten Nullen.
   Seit Initialisierung des Vorgängers von q wurden also an unbedingten Stellen nur Regeln benutzt, deren linke Seite als Kopfsymbol ein f aus $\mathcal{D}$ hat und deren Argumente Muster sind. Damit existiert, analog zu dem Beweis von Satz 3.3, ein gemeinsamer Vorgänger von u und dem Term, der bei Initialisierung von q anstelle von u in $\overline{e(t)}^{\perp}$ entsteht. Mit der Induktionsvoraussetzung und Lemma 3.3 sind diese dann ebenfalls zusammenführbar. (Vergleiche Beweis zu Satz 3.3).

□

$\mathcal{B}$ ist also allgemein konfluenzerhaltend. Mit der modifizierten Transformation aus Definition 3.3.1 ist jedoch schon eine Transformation mit einer solchen Eigenschaft vorgestellt worden. Es ist bisher noch unklar, wo genau der Vorteil von $\mathcal{B}$ gegenüber dieser Modifikation von $\mathcal{V}$ ist.

Dazu wird abschließend für diesen Abschnitt eine Eigenschaft für $\mathcal{B}$ gezeigt, welche auch für $\mathcal{V}$ gilt aber bei der Modifikation verloren geht. Es geht dabei um die Eigenschaft der Terminierungserhaltung. Wie bereits Definition 1.2.10 besagt, ist ein Termersetzungssystem terminierend, wenn in ihm keine unendliche Ableitung existiert.

Ein wichtiger Satz im Zusammenhang mit der Terminierung unbedingter Termersetzungssysteme, siehe [BA95], benutzt den Begriff der Reduktionsordnung.

Definition 3.4.4: (Partielle Ordnung)
Sei A eine Menge und $\geq$ eine binäre Relation auf A.
Gilt

- Reflexivität: Für alle $a \in A$ gilt $a \geq a$.
- Antisymmetrie: Für alle $a, b \in A$ gilt aus $a \geq b$ folgt $b \not\geq a$.
- Transitivität: Für alle $a, b, c \in A$ gilt aus $a \geq b$ und $b \geq c$ folgt $a \geq c$.

dann heißt $\langle \geq, A \rangle$ partielle Ordnung und $\langle >, A \rangle$ mit $> := \{(a, b) \mid a \geq b \wedge a \neq b\}$ heißt strikte partielle Ordnung oder auch der strikte Teil von $\langle \geq, A \rangle$. ◇

Definition 3.4.5: (Reduktionsordnung)
Sei Σ eine Signatur und X eine Variablenmenge sowie $\langle >, \mathcal{T}_\Sigma(X) \rangle$ eine strikte partielle Ordnung. Gilt

- Wohlfundiertheit: Es gibt keine unendliche absteigende Kette $t_0 > t_1 > \ldots$ mit $t_i \in \mathcal{T}_\Sigma(X)$.
- Kontextstabilität: Für alle $s, t, u \in \mathcal{T}_\Sigma(X)$ und alle Positionen p in u gilt aus $s > t$ folgt $u[p \longleftarrow s] > u[p \longleftarrow t]$.
- Invarianz gegenüber Substitutionen: Für alle $s, t \in \mathcal{T}_\Sigma(X)$ und alle Substitutionen σ gilt aus $s > t$ folgt $\sigma(s) > \sigma(t)$.

dann heißt $\langle >, \mathcal{T}_\Sigma(X) \rangle$ Reduktionsordnung. ◇

Reduktionsordnungen werden gebraucht, um die Terminierung von unbedingten Termersetzungssystemen zu beweisen. Gibt es eine Reduktionsordnung, so daß jeder Ableitungsschritt ein kleineres Ergebnis hat, so muß das System terminieren, da es keine unendlichen, absteigenden Ketten geben kann. (Wegen des Hintergedankens der Anwendung einer Reduktionsordnung wird überhaupt erst von Reduktionsschritten, Reduzibilität und dergleichen gesprochen.)

Satz 3.6: (Terminierung und Reduktionsordnung)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein Termersetzungssystem.
$\mathcal{R}$ ist terminierend genau dann, wenn es eine Reduktionsordnung $\langle >, \mathcal{T}_\Sigma(X) \rangle$ gibt mit $l > r$ für alle $l \longrightarrow r \in \mathrm{P}$. ◇

Die Auffindung einer solchen Reduktionsordnung ist alles andere als trivial. Das kann allerdings nicht weiter verwundern, denn das Terminierungsproblem für Termersetzungssysteme ist unentscheidbar. Um hier einen Erfolg zu erzielen, muß nach Wegen gesucht werden, zwei Reduktionsordnungen zu einer neuen zu verbinden. Zwei solcher Methoden, die dann auch beide im nächsten Beweis angewendet werden, sollen hier definiert werden.

Definition 3.4.6: (Lexikographische Ordnung)
Seien $\langle A_1, \geq_1 \rangle \ldots \langle A_n, \geq_n \rangle$ partielle Ordnungen.
Dann ist $\langle A_1 \times \ldots \times A_n, \geq_{lex} \rangle$ definiert durch $(a_1, \ldots, a_n) \geq_{lex} (b_1, \ldots, b_n)$ genau dann, wenn

1. Es gibt ein i mit $a_1 = b_1 \wedge \ldots \wedge a_{i-1} = b_{i-1}$ und $a_i \geq_i b_i$ oder
2. $a_1 = b_1 \wedge \ldots \wedge a_n = b_n$

$\diamond$

Lemma 3.5: (Eigenschaften der lexikopraphischen Ordnung)
Seien $\langle A_1, \geq_1 \rangle \ldots \langle A_n, \geq_n \rangle$ partielle Ordnungen.
Dann gilt:

1. $\langle A_1 \times \ldots \times A_n, \geq_{lex} \rangle$ ist eine partielle Ordnung
2. Sind $\langle A_1, \geq_1 \rangle \ldots \langle A_n, \geq_n \rangle$ wohlfundiert, so auch $\langle A_1 \times \ldots \times A_n, \geq_{lex} \rangle$.

$\diamond$

Dieses und das folgende Lemma werden hier nicht eigens bewiesen, vergleiche hierzu [BA95].

Definition 3.4.7: (Lexikographische Pfadordnung)
Sei Σ eine Signatur mit mindestens einem Konstantensymbol und $>_\Sigma \subseteq \Sigma \times \Sigma$ eine strikte, partielle, wohlfundierte Ordnung auf Σ.
Dann ist die durch $>_\Sigma$ definierte Lexikographische Pfadordnung $>_{lpo} \subseteq \mathcal{T}_\Sigma(X) \times \mathcal{T}_\Sigma(X)$ definiert durch $s > t$ genau dann, wenn

1. $t \,\epsilon\, var(s)$ und $s \neq t$ oder
2. $s = f(s_1, \ldots, s_n)$ und $t = g(t_1, \ldots, t_n)$ mit
   (a) Es gibt ein $s' \,\epsilon\, \{s_1, \ldots, s_n\}$ mit $s' \geq_{lpo} t$ oder
   (b) $f >_\Sigma g$ und für alle $t' \,\epsilon\, \{t_1, \ldots, t_n\}$ gilt $s >_{lpo} t'$ oder
   (c) f=g und
      i. Für alle $t' \,\epsilon\, \{t_1, \ldots, t_n\}$ gilt $s >_{lpo} t'$ und
      ii. $(s_1, \ldots, s_n) \; (>_{lpo})_{lex} \; (t_1, \ldots, t_n)$

$\diamond$

Lemma 3.6: (Eigenschaft der lexikographischen Pfadordnung)
Sei Σ eine Signatur mit mindestens einem Konstantensymbol und $>_\Sigma \subseteq \Sigma \times \Sigma$ eine strikte, partielle, wohlfundierte Ordnung auf Σ.
Die durch $>_\Sigma$ definierte lexikographische Pfadordnung $>_{lpo}$ ist eine Reduktionsordnung. $\diamond$

Bei bedingten Termersetzungssystemen stellt sich der Sachverhalt anders dar. Wie das folgende Beispiel zeigt, kann mit der Transformation $\mathcal{V}$ nicht zu jedem terminierenden bedingten System auch ein terminierendes unbedingtes System gefunden werden.

Beispiel 3.5

$$\mathcal{R} = a \longrightarrow true, \; if \; a \xrightarrow{*} true$$

Die Ableitungsrelation für dieses Beispiel ist leer, jedoch hat das Gegenstück in $\mathcal{V}$ (und entsprechend auch das in $\mathcal{B}$) die Form

$$\mathcal{R}_{\mathcal{V}} = \begin{array}{lll} a(\bot) & \longrightarrow & a(a(\bot)) \\ a(true) & \longrightarrow & true \end{array}$$

Dieses System hat die unendliche Ableitung

$$a \longrightarrow a(a(\bot)) \longrightarrow a(a(a(\bot)))\ldots$$

Für die Terminierung von bedingten Systemen muß daher ein weitergehender Begriff eingeführt werden.

Definition 3.4.8: (Reduzierend (engl. Decreasing))
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem.
$\mathcal{R}$ heißt reduzierend, wenn es eine Reduktionsordnung $\langle >, \mathcal{T}_\Sigma(X) \rangle$ gibt mit $l > r$ und $l > u$ für alle $l \longrightarrow r\ if\ u \stackrel{*}{\longrightarrow} v \ \epsilon\ \mathrm{P}$. $\diamond$

Es kann keine Reduktionsordnung geben, welche die Regel aus Beispiel 3.5 so anordnet, daß das System reduzierend wäre.

Ziel ist es nun, zu beweisen, daß für jedes bedingte Ausgangssystem $\mathcal{R}$ mit der Eigenschaft reduzierend das zugehörige $\mathcal{R}_{\mathcal{V}}$ terminiert.

Dieser Satz ist auch in [Vir95] enthalten. Bei dem dazugehörigen Beweis hat Viry jedoch den Fehler gemacht, die Möglichkeit einer *beliebig tiefen* Schachtelung von Bedingungen nicht zu beachten. Die in seinem Beweis konstruierte Reduktionsordnung betrachtet nur den Fall bedingter Stellen auf dem Toplevel eines Terms. Da der Beweis nicht auf einfache Weise reparabel erscheint, muß hier ein neuer Weg beschritten werden.

Da das Ausgangssystem die Eigenschaft reduzierend aufweist, gibt es eine Reduktionsordnung $>_{\mathcal{R}}$ auf den Termen $\mathcal{T}_\Sigma(X)$. Die Transformation $\mathcal{V}$ erweitert die Stelligkeit aller Funktionssymbole mit bedingten Regeln. Wie schon weiter oben in Abschnitt 3.2 erwähnt, können die Terme aus $\mathcal{T}_{\Sigma_{\mathcal{V}}}(X)$ als aus $\bot$-Termen aufgebaut betrachtet werden. Diese Betrachtungsweise wird nun für den Beweis genutzt, indem der Termaufbau dahingehend uminterpretiert wird, daß *die Menge* $\mathcal{T}_\Sigma(X)$ *nunmehr als Signatur* angesehen wird. Stelligkeit der Funktionssymbole für diese Signatur sind dann die bedingten Stellen eines jeden Terms aus $\mathcal{T}_\Sigma(X)$, angeordnet in Preorder, was mit $\mathcal{T}_\Sigma(X)^{(pre)}$ notiert wird.

Beispiel 3.6

$$\begin{gathered} f(g(x \mid [c(y),(y)]) \mid h(y), h(x)) \\ \text{wird interpretiert als} \\ \underline{f(g(x))}(\underline{c(y)}, y, \underline{h(y)}, \underline{h(x)}) \end{gathered}$$

Die Unterstreichungen zeigen dabei an, wo ein größerer Term zu *einem* Funktionssymbol geworden ist. Insbesondere ist zu beachten, daß alle Variablen (im Beispiel also y) in der neuen Interpretation Konstantensymbole sind. Das konstruierte Universum besteht also ausschließlich aus Grundtermen.

Zusätzlich zu $\mathcal{T}_\Sigma(X)^{(pre)}$ enthält das konstruierte Termuniversum von jedem der neuen Funktionssymbole eine Konstanten-Version, bei der also die bedingten Stellen nicht betrachtet werden. Dies wird mit $\mathcal{T}_\Sigma(X)^{(0)}$ notiert.

Das so konstruierte Termuniversum $\mathcal{T}_{(\mathcal{T}_\Sigma(X)^{(pre)} \cup \mathcal{T}_\Sigma(X)^{(0)})}(\emptyset)$ wird mit $\mathcal{T}_\infty$ abgekürzt, da es sich nunmehr um ein Universum mit unendlicher Signatur handelt.

Die Ordnung $\langle >_\mathcal{R}, \mathcal{T}_\Sigma(X)\rangle$ wird erweitert zu $\langle >_{\mathcal{T}_\infty}, \mathcal{T}_\infty\rangle$ wie folgt:

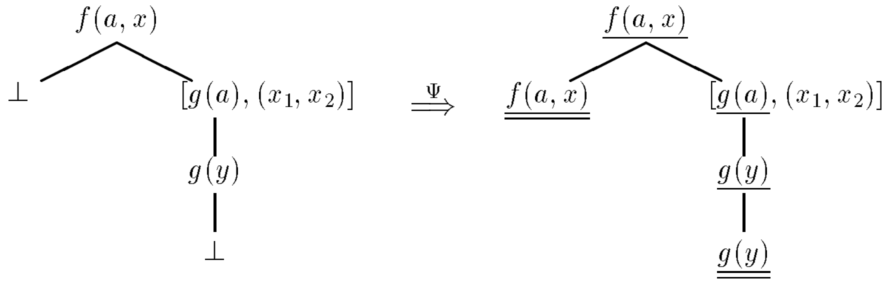
$$\begin{array}{lll} \text{Für alle} & f, g \in \mathcal{T}_\Sigma(X) & \text{gilt:} \quad f >_{\mathcal{T}_\infty} g, \text{ falls } f >_\mathcal{R} g, \text{ und} \\ \text{Für alle} & f \in \mathcal{T}_\Sigma(X) & \text{gilt:} \quad f^{(pre)} >_{\mathcal{T}_\infty} f^{(0)} \end{array}$$

Mit Hilfe dieser Anschauung wird es nun möglich, die von $>_{\mathcal{T}_\infty}$ definierte lexikographische Pfadordnung zu benutzen, um eine Reduktionsordnung auf $\mathcal{T}_{\Sigma_\mathcal{V}}(X)$ zu gewinnen. Dazu wird bei der Überführung Ψ der Terme aus $\mathcal{T}_{\Sigma_\mathcal{V}}(X)$ in das Universum $\mathcal{T}_\infty$ für jedes $\perp$ Symbol an bedingter Stelle die Konstanten-Version des zugehörigen unbedingten Terms eingesetzt.

Beispiel 3.7

$$\Psi(f(a, x \mid \perp, [g(a \mid g(y \mid \perp)), (x_1, x_2)]))$$
$$\Downarrow$$
$$\underline{f(a,x)}(\underline{\underline{f(a,x)}}, [\underline{g(a)}(\underline{g(y)}(\underline{\underline{g(y)}})), (x_1, x_2)])$$

Dabei soll die doppelte Unterstreichung die Konstanten-Version eines Termes bezeichnen. Anschaulich sieht die Übertragung wie folgt aus:



Satz 3.7: (Terminierungserhaltung von $\mathcal{V}$)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P}\rangle$ ein bedingtes Termersetzungssystem, welches reduzierend mit Reduktionsordnung $\langle >_\mathcal{R}, \mathcal{T}_\Sigma(X)\rangle$ ist.
Dann ist die Ordnung $\langle >_{\mathcal{R}_\mathcal{V}}, \mathcal{T}_{\Sigma_\mathcal{V}}(X)\rangle$ definiert durch

$$s >_{\mathcal{R}_\mathcal{V}} t \text{ genau dann, wenn } (\Psi(s)\ (>_{\mathcal{T}_\infty})_{lpo}\ \Psi(t))$$

und es gilt:
Für $s \xrightarrow{*}_{\mathcal{R}_\mathcal{V}} t$ folgt $s >_{\mathcal{R}_\mathcal{V}} t$, so daß also $\mathcal{R}_\mathcal{V}$ terminierend ist. $\diamond$

Beweis:

1. $\langle >_{\mathcal{R}_\mathcal{V}}, \mathcal{T}_{\Sigma_\mathcal{V}}(X)\rangle$ ist strikte partielle Ordnung:
   Da mit $>_\mathcal{R}$ auch $>_{\mathcal{T}_\infty}$ eine strikte partielle Ordnung ist, folgt diese Eigenschaft auch für $(>_{\mathcal{T}_\infty})_{lpo}$. Ψ ist außerdem injektiv.

2. $\langle >_{\mathcal{R}_\mathcal{V}}, \mathcal{T}_{\Sigma_\mathcal{V}}(X)\rangle$ ist wohlfundiert:
   Laut Konstruktion bildet Ψ endliche Terme auf endliche Terme ab, $(>_{\mathcal{T}_\infty})_{lpo}$ erbt die Wohfundiertheit von $>_\mathcal{R}$ über $>_{\mathcal{T}_\infty}$.

3. $\langle >_{\mathcal{R}_\mathcal{V}}, \mathcal{T}_{\Sigma_\mathcal{V}}(X)\rangle$ ist kontextstabil für die sinntragenden Terme:
   Kontextstabilität heißt, daß für alle $s, t, u \in \mathcal{T}_{\Sigma_\mathcal{V}}(X)$ und jede Stelle p in u mit $s >_{\mathcal{R}_\mathcal{V}} t$ auch folgt:
   $$u[p \longleftarrow s] >_{\mathcal{R}_\mathcal{V}} u[p \longleftarrow t].$$
   Für die Kontextstabilität sorgt die lexikographische Pfadordnung, es sei denn, nach der Ersetzung befindet sich ein $\perp$-Symbol an einer bedingten Stelle, welches sich zuvor an einer unbedingten befand. Da dieser Fall aber in der Menge der sinntragenden Terme nicht vorkommen kann, reicht $>_{\mathcal{R}_\mathcal{V}}$ hin, die Ableitungen in $\mathcal{R}_\mathcal{V}$ reduzierend anzuordnen.

4. $\langle >_{\mathcal{R}_\mathcal{V}}, \mathcal{T}_{\Sigma_\mathcal{V}}(X)\rangle$ ist invariant gegenüber Substitutionen für die sinntragenden Terme:
   Diese Eigenschaft erbt $>_{\mathcal{R}_\mathcal{V}}$ ebenfalls von $>_\mathcal{R}$ über $>_{\mathcal{T}_\infty}$, sieht man wiederum von $\perp$-Symbolen an unbedingten Stellen ab, siehe Punkt 3.

5. Für alle ρ'-Regeln $l \longrightarrow r \in \mathrm{P}_\mathcal{V}$ gilt $l >_{\mathcal{R}_\mathcal{V}} r$:
   $$\begin{array}{rcl} \Psi(f(\vec{l} \mid y_1 \ldots, \perp, \ldots, y_m)) & >_{\mathcal{R}_\mathcal{V}} & \Psi(f(\vec{l} \mid y_1 \ldots, [u^\perp_{f,i}, (\vec{x})], \ldots, y_m)) \\ \underline{f(\vec{l})}(\ldots, y_1, \ldots, \underline{\underline{f(\vec{l})}}, \ldots, y_m, \ldots) & >_{\mathcal{R}_\mathcal{V}} & \underline{f(\vec{l})}(\ldots, y_1, \ldots, [\Psi(u^\perp_{f,i}), \Psi(\vec{x})], \ldots, y_m, \ldots) \end{array}$$
   nach Definition Lexikographische Pfadordnung (3.4.7) genau dann, wenn
   $$\begin{array}{rcl} \underline{\underline{f(\vec{l})}} & >_{\mathcal{R}_\mathcal{V}} & \underline{u_{f,i}}(\underline{\underline{u_{f,i}}}, \ldots, \underline{\underline{u_{f,i}}}) \text{ und} \\ \underline{\underline{f(\vec{l})}} & >_{\mathcal{R}_\mathcal{V}} & x_i \text{ für alle } x_i \text{ aus } (\vec{x}) \end{array}$$
   Die erste Relation gilt, da das Ausgangssystem reduzierend ist. Die zweite Relation gilt, da jedes x_i aus $(\vec{x})$ in $f(\vec{l})$ enthalten ist.

6. Für alle ρ''-Regeln $l \longrightarrow r \in \mathrm{P}_\mathcal{V}$ gilt $l >_{\mathcal{R}_\mathcal{V}} r$:
   $$\begin{array}{rcl} \Psi(f(\vec{z} \mid y_1 \ldots, [v^X_{f,i}, (\vec{x})], \ldots, y_m)) & >_{\mathcal{R}_\mathcal{V}} & \Psi(r^\perp_{f,i}) \\ \underline{f(\vec{z})}(\ldots, y_1, \ldots, [\Psi(v^X_{f,i}), \Psi(\vec{x})], \ldots, y_m, \ldots) & >_{\mathcal{R}_\mathcal{V}} & \underline{r_{f,i}}(\underline{\underline{r_{f,i}}}, \ldots, \underline{\underline{r_{f,i}}}) \end{array}$$
   Diese Relation gilt, da $\mathcal{R}$ reduzierend ist und demnach $f(\vec{z}) >_\mathcal{R} r_{f,i}$ gilt. Damit gilt auch $\underline{f(\vec{z})} >_{\mathcal{T}_\infty} \underline{r_{f,i}}$ und $\underline{f(\vec{z})} >_{\mathcal{T}_\infty} \underline{\underline{r_{f,i}}}$.

Insgesamt ergibt sich, daß für $s \xrightarrow{*}_{\mathcal{R}_\mathcal{V}} t$ auch $s >_{\mathcal{R}_\mathcal{V}} t$ gilt und mit der Wohlfundiertheit von $\langle >_{\mathcal{R}_\mathcal{V}}, \mathcal{T}_{\Sigma_\mathcal{V}}(X)\rangle$ ist $\mathcal{R}_\mathcal{V}$ somit terminierend. □

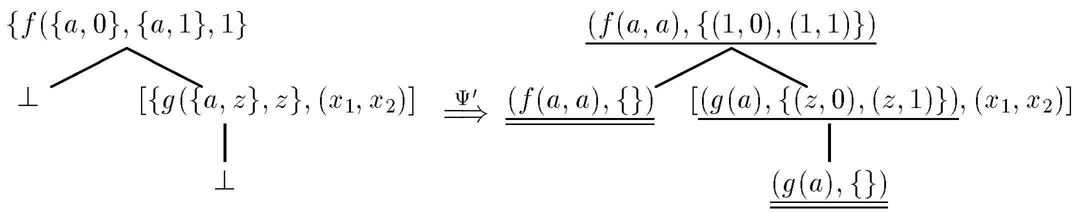
Als nächstes wird bewiesen, daß die Transformation $\mathcal{B}$ ebenfalls für jedes Ausgangssystem mit der Eigenschaft reduzierend ein terminierendes Zielsystem liefert. Dazu muß die Übertragung Ψ dahingehend modifiziert werden, daß auch die zusätzlichen Flags der Terme in $\mathcal{T}_{\Sigma_\mathcal{B}}(X)$ angeordnet werden können.

Daher werden bei der Überführung Ψ' der Terme aus $\mathcal{T}_{\Sigma_\mathcal{B}}(X)$ in $\mathcal{T}_\infty$ zusätzlich die Flags zusammen mit der Tiefe ihres Vorkommens gesammelt und dem Term beigefügt. Jedes Funktionssymbol f aus $\mathcal{T}_\infty$ wird also zu einem Paar (f, M), wobei die Multimenge M die Flags des Terms enthält. Diese Flags werden in Form von Paaren (a, n) wiedergegeben, wobei a entweder die 1 oder eine Variable aus X ist und n eine natürliche Zahl, welche der Tiefe des korrespondierenden Vorkommens von a entspricht. 0-Flags werden nicht betrachtet. Die Konstanten-Versionen der Funktionssymbole erhalten die leere Menge als zweite Tupelstelle.

Beispiel 3.8

$$\Psi'(\langle\{f(\{a,0\},\{a,1\} \mid \perp, [\{g(\{a,z\} \mid \perp), z\}, (x_1,x_2)]), 1\}\rangle)$$
$$=$$
$$\underline{(f(a,a),\{(1,0),(1,1)\})}(\underline{\underline{(f(a,a),\{\})}}, [\underline{(g(a),\{(z,0),(z,1)\})}(\underline{\underline{(g(a),\{\})}}), (x_1,x_2)])$$

Anschaulich sieht die Übertragung wie folgt aus:



Für Flagpaare $(a_1,n_1),(a_2,n_2)$ gilt:

$$(a_1,n_1) >_f (a_2,n_2) \text{ genau dann, wenn } a_1 = a_2 \text{ und } n_1 > n_2$$

Diese Ordnung induziert die Multimengenordnung $\gg$, mit der die Flagmengen eines jeden Funktionssymbols geordnet werden. Mit $\gg_{\mathcal{T}_\infty}$ wird die von $>_{\mathcal{T}_\infty}$ vor $\gg$ definierte lexikographische Ordnung bezeichnet.

Satz 3.8: (Terminierungserhaltung von $\mathcal{B}$)

Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P}\rangle$ ein bedingtes Termersetzungssystem, welches reduzierend mit Reduktionsordnung $\langle >_\mathcal{R}, \mathcal{T}_\Sigma(X)\rangle$ ist.
Dann ist die Ordnung $\langle >_{\mathcal{R}_\mathcal{B}}, \mathcal{T}_{\Sigma_\mathcal{B}}(X)\rangle$ definiert durch

$$s >_{\mathcal{R}_\mathcal{B}} t \text{ genau dann, wenn } (\Psi'(s)\ (\gg_{\mathcal{T}_\infty})_{lpo}\ \Psi'(t))$$

und es gilt:
Für $s \xrightarrow{*}_{\mathcal{R}_\mathcal{B}} t$ folgt $s >_{\mathcal{R}_\mathcal{B}} t$, so daß also $\mathcal{R}_\mathcal{B}$ terminierend ist. $\diamond$

Beweis:
Der neu hinzugefügten Multimengenanteil der Funktionssymbole kann für sich betrachtet werden. Hat dieser Anteil die Reduktionsordnungseigenschaften, so übertragen sich diese auf die gesamte Ordnung, so daß die in Beweis zu Satz 3.7 gemachten Aussagen (1-4) auch für $>_{\mathcal{R}_\mathcal{B}}$ zutreffen. Dazu sei $flags$ die Abbildung, die jedem Term aus $\mathcal{T}_{\Sigma_\mathcal{B}}(X)$ entsprechend der Arbeitsweise von Ψ die Multimenge seiner unbedingten Flags zuordnet. Dann gilt:

1. $\gg$ ist wohlfundiert, da $>_f$ wohlfundiert ist, siehe [BA95].

2. $\gg$ ist kontextstabil:
   für alle $s,t,u \in \mathcal{T}_{\Sigma_\mathcal{B}}(X)$ und jede Stelle p in u gilt, daß aus $flags(s) \gg flags(t)$ auch folgt:
   $$flags(u[p \longleftarrow s]) \gg flags(u[p \longleftarrow t]).$$

3. $\gg$ ist invariant gegenüber Substitution, wobei Flags nur mit 1 oder 0 belegt werden: Damit $flags(s) \gg flags(t)$ gilt, müssen alle Variablen x, die an Flagpositionen innerhalb von t vorkommen, auch an Flagpositionen in s stehen. In s steht jedes x dabei aber entweder tiefer oder kommt, bei gleicher Tiefststellung, häufiger in s vor.

   Ist $\sigma(x) = 0$, werden die Flags nicht mehr betrachtet, ist $\sigma(x) = 1$, so folgt ebenfalls $flags(\sigma(s)) \gg flags(\sigma(t))$, da die Multimenge der Flags von s auch in Bezug auf die Einsen schon größer als t ist.

Zu zeigen ist jetzt noch, daß sich alle Regeln aus $\mathrm{P}_{\mathcal{B}}$ mit $>_{\mathcal{R}_{\mathcal{B}}}$ anordnen lassen, zu der Form der Regeln vergleiche Definition 3.4.1.

1. Für alle ρ'-Regeln $l \longrightarrow r \in \mathrm{P}_{\mathcal{B}}$ gilt $l >_{\mathcal{R}_{\mathcal{B}}} r$:

   $$f(k(\vec{l}, Y) \mid \ldots, \perp, \ldots) \longrightarrow f(k(\vec{l}, Y) \mid \ldots, [k(c, 0), (\vec{x})], \ldots)$$

   Da der unbedingte Teil beider Regelseiten identisch ist, kommt es zu dem Vergleich der bedingten Stelle, der zugunsten der linken Seite ausfällt, ohne erst die Flagmenge zu betrachten, siehe Beweis zu Satz 3.7.

2. Für alle ρ''-Regeln $l \longrightarrow r \in \mathrm{P}_{\mathcal{B}}$ gilt $l >_{\mathcal{R}_{\mathcal{B}}} r$:

   $$k(l, Y) \longrightarrow k(r, 0) \text{ bzw. } k(r, 1)$$

   Schon nach dem Vergleich mit $>_{\mathcal{T}_\infty}$ gilt $\Psi'(k(l, Y)) >_{\mathcal{R}_{\mathcal{B}}} \Psi'(k(r, 0))$ bzw. $\Psi'(k(r, 1))$, siehe ebenfalls oben.

3. Für die für jedes Funktionssymbol hinzugefügten Regeln $l \longrightarrow r \in \mathrm{P}_{\mathcal{B}}$ gilt $l >_{\mathcal{R}_{\mathcal{B}}} r$:

   $$\{f(\ldots\{x_i, 1\}\ldots \mid x_{n+1}, \ldots, x_{n+m}), y\} \quad \longrightarrow \quad \{f(\ldots\{x_i, 0\}\ldots \mid \perp, \ldots, \perp), 1\}$$

   Bezüglich $>_{\mathcal{T}_\infty}$ ist der unbedingte Anteil beider Regelseiten gleich, für die unbedingten Flaganteile gilt:

   $$flags(l) = \{(1, 1), (y, 0)\} \gg flags(r) = \{(1, 0)\}$$

4. Für die restlichen Regeln $l \longrightarrow r \in \mathrm{P}_{\mathcal{B}}$ gilt $l >_{\mathcal{R}_{\mathcal{B}}} r$:

   (a) $\langle\{x, 1\}\rangle \longrightarrow \langle\{x, 0\}\rangle$ da $\{(1, 0)\} \gg \{\}$

   (b) $\{[\{x, 1\}, y], 0\} \longrightarrow \{[\{x, 0\}, y], 0\}$ da $\{(1, 1)\} \gg \{\}$

   (c) $\{[x, (\ldots, \{y, 1\}, \ldots)], 0\} \longrightarrow \{[x, (\ldots, \{y, 0\}, \ldots)], 0\}$ da für alle Multimengen X gilt $X \cup \{(1, 1)\} \gg X$.

Insgesamt ergibt sich, daß für $s \xrightarrow{*}_{\mathcal{B}} t$ auch $s >_{\mathcal{R}_{\mathcal{B}}} t$ gilt und mit der Wohlfundiertheit von $\langle >_{\mathcal{R}_{\mathcal{B}}}, \mathcal{T}_{\Sigma_{\mathcal{B}}}(X)\rangle$ ist $\mathcal{B}$ somit terminierend. □

3.5 Prolegomena zu einer jeden künftigen Transformation

Im nächsten Kapitel werden Ergebnisse erzielt, die in Richtung auf eine praktische Nutzung der hier gewonnenen theoretischen Ergebnisse im Rahmen der funktional-logischen Programmierung gehen. Bevor dies geschieht, ist es das Anliegen dieses Abschnitts, darzustellen, was aus dem hier erarbeiteten von grundsätzlicher Bedeutung für eine allgemeine Methodik zur Transformation ausdrucksstärkerer Termersetzungssysteme in Standardsysteme sein könnte. Die wesentliche Teile der in der vorliegenden Arbeit betrachteten Transformation werden in diesem Abschnitt ganz allgemein dargestellt, um eventuelle Charakteristika einer jeden derartigen Transformation hervorzuheben.

Die bisherige Arbeit untersuchte, inwieweit sich eine bestimmte höhere Art von Termersetzungssystemen (die bedingten) in äquivalente Standardtermersetzungssysteme (hier zur Unterscheidung mit unbedingt bezeichnet) transformieren läßt. Daß bedingte Termersetzungssysteme eine höhere Art der Ausdrucksmöglichkeit darstellen, bezieht sich hier zunächst nur auf den Umstand, daß ein äquivalentes, unbedingtes Termersetzungssystem, wie weiter oben schon erwähnt, immer eine größere Signatur bemühen muß [BT87]. Das heißt, daß mit den bedingten Systemen mindestens so etwas wie eine kompaktere Darstellungsmöglichkeit verbunden ist. Das liegt vor allem daran, daß in den Bedingungen ein Bezug auf die Ableitungsrelation $\xrightarrow{*}$ möglich ist, ein Bezug, der für ein Standardtermersetzungssystem in den Bereich der Metainformation gehört.

Dennoch sind bedingte Termersetzungssysteme nicht berechnungsstärker [BT87], was insofern nicht verwunderlich ist, als Standardsysteme berechnungsäquivalent zu den Turingmaschinen sind [BA95]. Das Motiv für eine Transformation in Standardsysteme besteht unter anderem darin, die wesentlich bessere Erforschung dieser Standardsysteme auch für die stärkeren Ausgangssysteme zu nutzen.

Daher macht es Sinn, auch andere Arten „ausdrucksstärkerer" Termersetzungssysteme in äquivalente Standardsysteme zu transferieren. Auch bei diesen ist zu erwarten, daß die Signatur des unbedingten Termersetzungssystem notwendigerweise größer wird. Unter gewissen Grundannahmen läßt sich ein Teil dieser Arbeit auf diesen allgemeinen Fall des Transformationsvorhabens übertragen.

Die nun folgende Aufzählung versucht, voraussichtliche Merkmale vergleichbarer Ansätze anzugeben. Sie enthält Beobachtungen über die hier untersuchte Transformation, die auf den ersten Blick für zufällige Eigenschaften gehalten werden könnten, die aber bei genauerer Untersuchung den Charakter der Notwendigkeit annehmen. Die Begründungszusammenhänge zwischen diesen Beobachtungen darzustellen, ist Anliegen dieses Exkurses. Es handelt sich bei dem Folgenden um Vermutungen, die in dieser Allgemeinheit wohl schwerlich beweisbar sind, die aber trotzdem Ansätze einer allgemeinen Methodik aufweisen.

1. Das Ausgangssystem hat eine kleinere Signatur als das äquivalente Standardsystem.
   Daraus ergibt sich die notwendige Folgerung:

2. Das Ausgangssystem hat ein kleineres Termuniversum.
   Aus diesem Umstand wiederum läßt sich folgern, daß das Ausgangssystem in das Zielsystem mit einer Art Einbettung eingebunden wird.

3. Erforderlich ist ein injektiver Homomorphismus H : Ausgangssystem $\longrightarrow$ Zielsystem.
   Im Fall der bedingten Termersetzungssysteme wurde die Abbildung $x \mapsto x^{\perp}$ mit Starttermzuordnung bezeichnet. Mit dem vergrößerten Termuniversum „schleichen" sich aber notwendigerweise auch Terme ein, die im Kontext einer Entsprechung des Ausgangssystems keinen Sinn haben.

4. Das Zielsystem muß daher auf eine Menge von sinntragenden Termen beschränkt werden.
   Die Definition des sinntragenden Teils 1.5.2 ist hinreichend, um auch in anderen Fällen die gewünschte Menge abzugrenzen. Die Vollständigkeit der Transformation läßt sich mit Hilfe der Abbildung H formulieren als:

5. Aus $s \longrightarrow_{\mathcal{R}} t$ folgt $H(s) \xrightarrow{*}_{\mathcal{R}'} H(t)$
   Die schwächere Version, siehe Abschnitt 1.5, wurde hier mit Absicht nicht benutzt. Ein einzelner Ersetzungsschritt im Ausgangssystem muß durch mehrere Schritte im Standardsystem simuliert werden. Dies ist ebenfalls charakteristisch für ausdrucksstärkere Systeme. Im Fall der bedingten Systeme muß erst die Initialisierung einer Bedingung und deren Auswertung zu der gewünschten Normalform erfolgen, bevor ein Schritt zu einem neuen Term im Sinne des Ausgangssystems erfolgen kann. Insbesondere gibt es meist auch eine ausgezeichnete Minimalentsprechung. Daraus folgt dann zwingend:

6. In der Ableitung $H(s) \xrightarrow{*}_{\mathcal{R}'} H(t)$ tauchen außer $H(s)$ und $H(t)$ Zwischenformen auf, die *nicht* zur Bildmenge von H gehören.
   Das kommt daher, daß es sich im Ausgangssystem nur um *einen* Schritt handelte. Zusätzlich zu H gibt es auch eine surjektive, meist partielle Rückabbildung G, mit deren Hilfe sich die Korrektheit der Transformation ausdrücken läßt als:

7. Aus $H(s) \longrightarrow_{\mathcal{R}'} t$ folgt $G(H(s)) \xrightarrow{*}_{\mathcal{R}'} G(t)$
   Der Zusammenhang der beiden Abbildungen läßt sich formulieren als $G(H(t)) = t$ für alle Terme des Ausgangsuniversums.
   Diese Zuordnung G sollte nun einen Term $H(t)$ *mitsamt der zugehörigen Zwischenformen* auf t abbilden. Warum ist dies eine sinnvolle Forderung? Zunächst sollte offensichtlich sein, daß die Zwischenformen auf keinen Fall auf irgendwelche anderen Terme im Universum von $\mathcal{R}$ abgebildet werden dürfen, soll denn Korrektheit gewährleistet sein. Allenfalls könnte G für diese Zwischenformen undefiniert sein. Dies würde Sinn machen, wenn garantiert werden kann, daß jede Zwischenform wieder zu einem Term aus der Bildmenge von H reduzierbar ist. Dies ist aber unmöglich zu garantieren, was mit einem weiteren charakteristischen Merkmal ausdrucksstärkerer Termersetzungssysteme zusammenhängt:

8. Die Menge der Normalformen des Ausgangssystems ist eventuell unentscheidbar.
   Dies ist zum Beispiel bei den bedingten Systemen der Fall. Bei Standardsystemen hingegen ist die Menge der Normalformen effektiv entscheidbar. Wie kann dann aber das Zielsystem äquivalent sein? Das ist nur dann möglich, wenn zum Bild einer Normalform z also $H(z)$ noch weitere Zwischenformen als Nachfolger existieren, so daß eben nicht entscheidbar ist, ob noch ein weiteres Element der Bildmenge von H ableitbar ist. Es ergibt sich insgesamt die sinnvolle Forderung an die Rückabbildung:

9. Der Definitionsbereich von G sollte die gesamte Menge der sinntragenden Terme enthalten.
   Nur so ist Kontinuität der Korrektheit für alle Ableitungen im Zielsystem zu gewährleisten. Die Abbildung G gibt dann auch Anlaß zu der weiteren Betrachtung:

10. Das Zielsystem ist also in eine Menge von Klassen unterteilbar, bei der jedes Element einer Klasse unter G auf denselben Term t im Ausgangsuniversum abgebildet wird.
    Jede Klasse hat insbesondere einen ausgezeichneten Repräsentanten, nämlich $H(t)$. Wie schon mehrfach ausgeführt, ist die Eigenschaft der Konfluenz für jede Art von Termersetzungssystemen von besonderem Interesse. Daher ist es wichtig, eine Methodik zu finden, wie die Konfluenzerhaltung einer Transformation gezeigt werden kann. Ist das Ausgangssystem konfluent und die Transformation korrekt und vollständig, so ist ein einfaches Kriterium für die Konfluenzerhaltung:

11. Für jede Klasse K und jeden Term $t \in K$ gilt $t \downarrow H(G(t))$
    Bei dieser Zusammenführung dürfen ausschließlich Terme derselben Klasse auftreten.

Dadurch wird jede Klasse zu einer Kongruenzklasse bezüglich einer Art von Teilrelation der Ableitungsrelation $\xrightarrow{*}_{\mathcal{R}'}$.
Es geht also nicht um die Kongruenzklassen der gesamten Ableitungsrelation, sondern nur der der Zwischenformen. Es ist vorstellbar, daß es bei wesentlich ausdrucksstärkeren Systemen eine Schachtelungs*hierarchie* von derartigen Kongruenzklassen gibt, daß also jede Klasse wiederum in eine Menge von Teilklassen einzuteilen ist, für die dann jeweils eine eigene Teilrelation kongruenzbildend wäre.

Möglicherweise reicht dieses Kriterium, je nach Transformation nicht hin, um Konfluenzerhaltung zu gewährleisten. Ursprünglich sollte für die hier erzielten Beweise eine stärkere Forderung Verwendung finden. Sie ist aber wegen dieser Stärke schwerer zu beweisen.

12. Für jede Klasse K und alle Terme $s, t \in K$ gilt $s \downarrow t$
Durch die Zweiteilung der Konfluenzerhaltung mit Hilfe eines derartigen hinreichenden Kriteriums werden die Beweise um ein vielfaches vereinfacht (wenn nicht sogar überhaupt erst menschenmöglich...)

Dieser kurze Überblick ist alles, was an dieser Stelle möglich ist. Mir bleibt die Hoffnung, daß die hier gemachten Ausführungen irgendwann einmal weiteren Überlegungen für wert befunden werden.

Kapitel 4

Transformation funktional-logischer Programme

Wie schon in der Einleitung dargelegt, ist eines der Hauptanwendungen von Termersetzungssystemen die funktionale Programmierung. Die funktionale und logische Programmierung bilden zusammen den Bereich der deklarativen Programmierung. In den letzten Jahren wurden einige Fortschritte dabei erzielt, die beiden Paradigmen zu vereinheitlichen. Bestehende funktional-logische Programmiersprachen verwenden als Grundlage zu dieser Vereinheitlichung die bedingten Termersetzungssysteme (siehe [KLMNRA89] und [Han97]). Aufgrund der wesentlich besseren Forschungslage auf seiten der unbedingten Systeme gibt es dabei, wie erwähnt, Probleme sowohl was den theoretischen Hintergrund, als auch was die operationale Effizienz betrifft. Effiziente Methoden, mit unbedingten Regeln umzugehen, sind wesentlich weiter entwickelt (vergleiche Abschnitt 1.3).

Daher soll es in diesem Kapitel darum gehen zu untersuchen, *welche Klasse von bedingten Termersetzungssystemen mit welcher Art von Transformation* als Ergebnis *in ein funktionales Programm überführt wird.*

4.1 Orthogonalität und Disjunktion

Die meist eingesetzten operationalen Mechanismen für funktional-logische Sprachen sind Residuation und Narrowing. Dabei besitzen Narrowingverfahren den Vorteil der Vollständigkeit, das heißt die Berechnung allgemeiner Formen für alle existierenden Normalformen. Residuation hingegen erlaubt eine einfache Einbindung externer Funktionen. Beide Ansätze lassen sich verbinden, siehe [Han97].

Um die Vollständigkeit der Berechnung zu garantieren, müssen dem betrachteten Termersetzungssystem bestimmte Beschränkungen auferlegt werden. So ist zum Beispiel ein funktionales Programm definiert als ein schwach orthogonales, konstruktorbasiertes Termersetzungssystem (Definitionen 1.2.12 und 1.2.13).

Konstruktorbasiertheit kann, wie in Definition 1.4.3 geschehen, ohne Probleme in den Bereich der bedingten Systeme übertragen werden. Für die Orthogonalität ist es aber ratsam, mehr Überlegung zu investieren.

Dem Begriff Orthogonalität liegt der Gedanke zugrunde, daß alle Regeln des Termersetzungssystems so aufgebaut sind, daß zu einem beliebigen Zeitpunkt keine zwei Regeln an überlappenden Teiltermen eines Terms angewandt werden können. Konstruktorbasiertheit schränkt derartige Überlappungsmöglichkeiten in zwei Punkten stark ein:

1. Zwei überlappende Regeln gehören zu demselben Funktionssymbol.
2. Können zwei Regeln in einem Term überlappend angewandt werden, so nur an ein und derselben Stelle.

In orthogonalen Systemen sind Überlappungen ganz verboten. Daher ist es gewährleistet, daß es *immer eindeutig* ist, welche Regel an einer Stelle eines Terms angewandt werden kann. Schwache Orthogonalität ist in sofern etwas lockerer, als überlappende Regelanwendungen erlaubt sind, aber *nur* unter der Bedingung, daß das Ergebnis aller Anwendungen exakt dasselbe ist. Die Eigenschaft der (schwachen) Orthogonalität ist hinreichend für die Konfluenz eines Termersetzungssystems.

Überlappen im *bedingten* Fall die linken Regelseiten zweier Regeln, so bedeutet das noch nicht, daß die beiden Regeln auch überlappend angewendet werden können. Bevor eine Regel angewendet werden kann, muß erst noch die zugehörige Bedingung erfolgreich auswertbar sein. Daher ist es sinnvoll, den Grundgedanken der Orthogonalität in der folgenden Form auf bedingte Systeme zu übertragen:

Definition 4.1.1: ((Schwache) Orthogonalität, bedingter Fall)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem.
Dann heißt $\mathcal{R}$

- linear genau dann, wenn gilt:
  Keine Variable kommt in einer linken Regelseite mehr als einmal vor.
- orthogonal genau dann, wenn $\mathcal{R}$ linear ist und gilt:
  Für alle überlappenden Regelpaare $l_1 \longrightarrow r_1 \; if \; u_1 \stackrel{*}{\longrightarrow} v_1$ und $l_2 \longrightarrow r_2 \; if \; u_2 \stackrel{*}{\longrightarrow} v_2$ mit Unifikator σ ist $(\sigma(u_1) \stackrel{*}{\longrightarrow} v_1) \wedge (\sigma(u_2) \stackrel{*}{\longrightarrow} v_2)$ unerfüllbar.
- schwach orthogonal genau dann, wenn $\mathcal{R}$ linear ist und gilt:
  Für alle überlappenden Regelpaare $l_1 \longrightarrow r_1 \; if \; u_1 \stackrel{*}{\longrightarrow} v_1$ und $l_2 \longrightarrow r_2 \; if \; u_2 \stackrel{*}{\longrightarrow} v_2$ mit Unifikator σ gilt
  1. $\sigma(r_1) = \sigma(r_2)$ oder
  2. $(\sigma(u_1) \stackrel{*}{\longrightarrow} v_1) \wedge (\sigma(u_2) \stackrel{*}{\longrightarrow} v_2)$ ist unerfüllbar.

◇

Mit dieser Übertragung ist es auch im bedingten Fall eindeutig, welche Regel angewandt werden kann, denn für alle in Frage kommenden Regeln schließen sich *die Bedingungen aus*.

Anzumerken ist, daß die Orthogonalität, wie sie hier definiert wurde, für unbedingte Termersetzungssysteme rein syntaktisch effektiv überprüfbar ist, während sie für bedingte Termersetzungssysteme unentscheidbar ist. In der Praxis müssen daher weitere, syntaktisch überprüfbare Einschränkungen für die Bedingungen gemacht werden, die gewährleisten, daß sie einander ausschließen, siehe dazu zum Beispiel [KLMNRA89].

Schön wäre es, eine Modifikation der Transformation von Viry zu finden, die ein konstruktorbasiertes, schwach orthogonales, bedingtes Termersetzungssystem in ein konstruktorbasiertes, schwach orthogonales, unbedingtes Termersetzungssystem transformiert, das die Kriterien in Definition 1.5.4 erfüllt. Dies ist jedoch im allgemeinen nicht erreichbar, denn eine solche Transformation würde ja entscheiden, wann sich die Bedingungen zweier Regeln ausschließen. Daher müssen weitere Einschränkung vorgenommen werden.

Zu diesem Zweck wird der Begriff des *disjunkten Termersetzungssystems* eingeführt. Es liegt dieselbe Idee wie bei der Orthogonalität zugrunde, mit der folgenden Änderung:
Angenommen, zwei Regeln überlappen in der linken Regelseite, die rechten Regelseiten sind dabei unterschiedlich. Das System ist damit nicht orthogonal. Kann man aber garantieren, daß ein Term, auf den beide Regeln an derselben Stelle angewandt werden könnten, *nicht ableitbar* ist, so ist die Wahl einer Regel *immer noch eindeutig*.

Es ist offensichtlich, daß diese Idee in einem normalen Termersetzungssystem sinnlos ist, da in einem solchen System alle Terme zu Ableitungen herangezogen werden dürfen. Hat man aber, wie im Kontext dieser Arbeit, eine Einschränkung des Termuniversums, so macht der Begriff eines disjunkten Termersetzungssystems Sinn. Diese Eigenschaft ist in derselben Weise wie die Orthogonalität hinreichend für die Konfluenz dieses eingeschränkten Teiles des Systems.

Definition 4.1.2: ((Schwach) Disjunktes Termersetzungssystem)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein konstruktorbasiertes und lineares unbedingtes Termersetzungssystem und $\mathcal{T} \subseteq \mathcal{T}_\Sigma(X)$ eine unter $\xrightarrow{*}$ abgeschlossene Menge von Termen.
Dann heißt $\mathcal{R}$

- disjunkt bezüglich $\mathcal{T}$, wenn für alle Regeln $\rho \neq \rho' \in \mathrm{P}$ gilt:
  Es existieren keine Terme $t, x, y \in \mathcal{T}$ mit Position p in t und
  $t \longrightarrow_{[\rho,p]} x$ sowie $t \longrightarrow_{[\rho',p]} y$.
- schwach disjunkt bezüglich $\mathcal{T}$, wenn für alle Regeln $\rho, \rho' \in \mathrm{P}$ gilt:
  Es existieren keine Terme $t, x \neq y \in \mathcal{T}$ mit Position p in t und
  $t \longrightarrow_{[\rho,p]} x$ sowie $t \longrightarrow_{[\rho',p]} y$.

$\diamond$

Anzumerken ist, daß für $\mathcal{T} = \mathcal{T}_\Sigma(X)$ disjunkt gleichbedeutend mit Orthogonalität ist. Auch können mit der Eigenschaft der Konstruktorbasiertheit Überlappungen nur an genau derselben Position auftreten.

Ziel des nächsten Abschnittes ist es, zu zeigen, daß ein schwach orthogonales Termersetzungssystem mit der Transformation $\mathcal{V}$ in ein schwach disjunktes System überführt werden kann.

4.2 Die Transformation $\mathcal{Z}$

Im Zielsystem von $\mathcal{V}$ gibt es zwei Quellen für Überlappungen, die ρ' und die ρ''-Regeln. Um ein disjunktes Zielsystem zu gewährleisten, muß zumindestens die erste Quelle ausgeschaltet werden.

Dazu wird eine vorgeschaltete Transformation innerhalb der bedingten Systeme angewandt werden, die alle Regeln auf eine einheitliche Form bringt, sie *konformiert*. Mit einheitlicher Form ist gemeint, daß alle überlappenden Regeln dieselbe linke Seite bekommen sollen. Es wird zunächst mit der Relation $\bowtie$ abgegrenzt, welche Regeln jeweils zu konformieren sind.

Definition 4.2.1: (Relation $\bowtie$)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem.
Die Relation $\bowtie \subset \mathrm{P} \times \mathrm{P}$ ist der transitive Abschluß der Relation überlappend (Definition 1.2.11). $\diamond$

Alle Regeln $\rho \bowtie \rho'$ werden auf dieselbe linke Seite gebracht. Diese Seite muß eine *Verallgemeinerung* der zu konformierenden Regelseiten sein.

Definition 4.2.2: (Verallgemeinerung linker Regelseiten)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein bedingtes Termersetzungssystem und seien

$$\rho_1 : l_1 \longrightarrow r_1 \ if \ u_1 \stackrel{*}{\longrightarrow} v_1, \ldots, \rho_n : l_n \longrightarrow r_n \ if \ u_n \stackrel{*}{\longrightarrow} v_n$$

Regeln aus P mit $\rho_i \bowtie \rho_j$ für $i, j \in \{1, \ldots, n\}$.
Der Term l heißt Verallgemeinerung der linken Regelseiten von $\rho_1 \ldots \rho_n$, wenn gilt:
l hat keine Variablen mit irgendeinem l_i gemeinsam und es gibt Substitutionen σ_i mit $\sigma_i(l) = l_i$ für alle i aus $\{1, \ldots, n\}$ und es gibt kein l' mit derselben Eigenschaft und einer Substitution τ mit $\tau(l) = l'$, die nicht nur eine Umbenennung der Variablen wäre. $\diamond$

Die Verallgemeinerung linker Regelseiten wird nun die neue linke Seite für alle überlappenden Regeln. Diese müssen daraufhin verändert werden, was mit Konformation bezeichnet wird.

Definition 4.2.3: (Konformation)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein konstruktorbasiertes, lineares und bedingtes Termersetzungssystem. Dann heißt $\mathcal{R}_\mathcal{K} = \langle \Sigma_\mathcal{K}, X, \mathrm{P}_\mathcal{K} \rangle$ Konformation von $\mathcal{R}$.

$$\Sigma_\mathcal{K} = \Sigma \cup \{true^{(0)}, *^{(0)}, \wedge^{(2)}, match^{(2)}, proj_1^{(1)}, \ldots, proj_s^{(1)}\},$$

wobei s die höchste Stelligkeit eines Konstruktorsymbols in Σ ist.

Sei $\mathrm{P}_1, \ldots, \mathrm{P}_n$ die Zerlegung von P in Äquivalenzklassen bezüglich $\bowtie$.

Für jedes P_i sei l_i eine Verallgemeinerung der linken Regelseiten. $(\mathrm{P}_i)_\mathcal{K}$ entsteht aus P_i, indem jede Regel $\rho : l \longrightarrow r \ if \ u \stackrel{*}{\longrightarrow} v$ aus P_i durch eine neue Regel ersetzt wird, entsprechend der Fallunterscheidung

1. l ist bis auf eine Variablenumbenennung σ gleich mit l_i
   Dann hat die neue Regel die Form $\sigma(l) \longrightarrow \sigma(r) \ if \ \sigma(u) \stackrel{*}{\longrightarrow} v$.

2. Ansonsten wird ρ ersetzt durch eine Regel der Form

$$\rho' : l_i \longrightarrow \sigma^{-1}(r) \ if \ (\sigma^{-1}(u), Match(\sigma)) \stackrel{*}{\longrightarrow} (v, true)$$

   wobei $\sigma(l_i) = l$ ist.

Für $Match(\sigma)$ wird ein Term wie folgt eingesetzt:

Sei $\{x_1, \ldots, x_m\}$ die Menge der Variablen aus l_i für die σ nicht nur eine Variablenumbenennung ist. Dann ist

$$Match(\sigma) = match(x_1, \tau(\sigma(x_1))) \wedge \ldots \wedge match(x_m, \tau(\sigma(x_m))) \stackrel{*}{\longrightarrow} true$$

wobei τ eine Substitution ist, die alle Variablen aus l durch das neue Konstruktorsymbol $*$ ersetzt.

σ^{-1} ist eine Substitution, die in gewisser Weise umgekehrt arbeitet als σ. Gilt $\sigma(x)|_{p_1 \cdot \ldots \cdot p_k} = y$, so ist $\sigma^{-1}(y) = proj_{p_k}(\ldots proj_{p_1}(x)\ldots)$. Anzumerken ist, daß wegen der Linearität der linken Regelseiten y nur in einem $\sigma(x)$ vorkommen kann, σ^{-1} also wohldefiniert ist.

Damit sind die Regeln in $\mathcal{R}_\mathcal{K}$ gegeben durch

$$\mathrm{P}_\mathcal{K} = (\mathrm{P}_1)_\mathcal{K} \cup \ldots \cup (\mathrm{P}_n)_\mathcal{K} \cup \{true \wedge x \longrightarrow x\} \cup Match \cup Proj$$

$Match$ enthält Regeln für jeden $match$-Ausdruck, der in einem $\mathrm{P}_{\mathcal{K}_i}$ vorkommt:

$$\begin{array}{lll} & match(x,*) & \longrightarrow \; true \\ c^{(0)} \in \Sigma: & match(c,c) & \longrightarrow \; true \\ c^{(s>0)} \in \Sigma: & match(c(x_1,\ldots,x_s), c(y_1,\ldots,y_s)) & \longrightarrow \; match(x_1,y_1) \wedge \ldots \wedge match(x_s,y_s) \end{array}$$

$Proj$ enthält Regeln entsprechend den in den $(\mathrm{P_i})_\mathcal{K}$ vorkommenden Ausdrücken:

$$proj_j(c(t_1,\ldots,t_j,\ldots,t_s)) \longrightarrow t_j$$

$\diamond$

Bevor deutlich gemacht wird, wozu diese Transformation dienen soll, wird sie zunächst auf eine weitere Variation des Referenzbeispiels angewandt.

Beispiel 4.1

$$\mathcal{R} = \begin{pmatrix} nil([]) & \longrightarrow & true \\ nil(x:xs) & \longrightarrow & false \\ append(xs,ys) & \longrightarrow & ys \; if \; nil(xs) \xrightarrow{*} true \\ append(x:xs,ys) & \longrightarrow & x : append(xs,ys) \end{pmatrix}$$

Nach der Transformation sieht das System wie folgt aus

$$\mathcal{R}_\mathcal{K} = \begin{pmatrix} nil([]) & \longrightarrow & true \\ nil(x:xs) & \longrightarrow & false \\ append(xs,ys) & \longrightarrow & ys \; if \; nil(xs) \xrightarrow{*} true \\ append(xs,ys) & \longrightarrow & proj_1(xs) : append(proj_2(xs), ys) \\ & & \quad if \; match(xs, *:*) \xrightarrow{*} true \\ true \wedge x & \longrightarrow & x \\ match(x_1 : x_2, y_1 : y_2) & \longrightarrow & match(x_1,y_1) \wedge match(x_2,y_2) \\ match(x,*) & \longrightarrow & true \\ proj_1(x_1 : x_2) & \longrightarrow & x_1 \\ proj_2(x_1 : x_2) & \longrightarrow & x_2 \end{pmatrix}$$

Es ist schnell ersichtlich, daß mit $\mathcal{R}$ auch $\mathcal{R}_\mathcal{K}$ konstruktorbasiert ist. Noch zu zeigen ist:

Satz 4.1: (Korrektheit und Vollständigkeit der Konformation)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein konstruktorbasiertes, bedingtes Termersetzungssystem.
Für beliebige Terme s,t aus $\mathcal{T}_\Sigma(X)$ gilt $s \xrightarrow{*}_\mathcal{R} t$ genau dann, wenn $s \xrightarrow{*}_{\mathcal{R}_\mathcal{K}} t$. $\diamond$

Beweis:
Die Behauptung ergibt sich aus zwei Beobachtungen für jede Regel $\rho : l \longrightarrow r \; if \; u \xrightarrow{*} v \in \mathrm{P}$ und die für diese eingesetzte Regel $\rho' : l' \longrightarrow r' \; if \; (u', m) \xrightarrow{*} (v, true) \in \mathrm{P}_\mathcal{K}$, wobei m der Anteil der Bedingung von ρ' ist, der die $match$-Ausdrücke enthält.

1. Das Muster in l paßt genau dann auf einen Term x, wenn für x der m-Ausdruck auf $true$ reduzierbar ist.
   Damit das Muster der linken Seite l der Regel ρ auf x paßt, muß es eine Substitution π geben mit $\pi(l) = x$. Nach Konstruktion überprüft der $Match$-Ausdruck in ρ' die Existenz einer solchen Substitution (genau so, wie es ein Interpreter auch tun könnte). Das Pattern-Matching ist lediglich in die Bedingung verlagert worden.

2. $\sigma'(r') \xrightarrow{*}_{\mathcal{R}_{\mathcal{K}}} \sigma(r)$ und $\sigma'(u') \xrightarrow{*}_{\mathcal{R}_{\mathcal{K}}} \sigma(u)$
   Diese Ableitungen erhält man, wenn man die $proj_i$-Anteile zuerst abarbeitet. Nach Konstruktion der $proj_i$-Regeln und der Terme r' und u' liefern diese $proj_i$-Anteile genau die Ergebnisse, die ansonsten durch die Zerlegung der Terme durch das Pattern-Matching geliefert worden wären.

Es ergibt sich, daß ρ genau dann auf einen Term $x \in \mathcal{T}_\Sigma(X)$ anwendbar ist, wenn auch ρ' auf x angewendet werden kann, und daß das Ergebnis beider Anwendungen äquivalent ist. □

Auch mit diesem Ergebnis ist fraglich, was mit der Konformation gewonnen wurde. Schließlich sind wegen der Transitivität von $\bowtie$ unter Umständen linke Seiten von Regeln unifizierbar, die zuvor noch nicht überlappten. Als Beispiel sollen die linken Regelseiten

$$\begin{array}{l} f(x, []) \\ f([], x : xs) \\ f(x, y) \end{array}$$

dienen. Eine allgemeinste linke Regelseite ist $f(x_1, x_2)$, so daß nach der Konformation alle drei Regeln überlappen, obwohl die ersten beiden Terme zuvor nicht unifizierbar waren.

Aber es hat auch einen wichtigen Vorteil, daß *alle überlappenden Regeln genau dieselbe linke Regelseite* haben. Der Nutzen dieser Eigenschaft wird im unbedingten Zielsystem $(\mathcal{R}_{\mathcal{K}})_{\mathcal{V}}$ deutlich.

Seien $\{i_1, \ldots, i_k\}$ die Indizes aller Regeln mit der linken Regelseite $f(l_1, \ldots, l_n)$ aus $\mathrm{P}_{\mathcal{K}}$. Dann können *alle zugehörigen* ρ' Regeln aus $(\mathrm{P}_{\mathcal{K}})_{\mathcal{V}}$

$$\begin{array}{rcl} \rho'_{f,i_1} : f(l_1 \ldots l_n \mid y_1, \ldots, \perp_{i_1}, \ldots, y_{i_k}, \ldots, y_m) & \longrightarrow & f(l_1 \ldots l_n \mid y_1, \ldots u_{i_1}, \ldots, y_{i_k}, \ldots, y_m) \\ & \vdots & \\ \rho'_{f,i_k} : f(l_1 \ldots l_n \mid y_1, \ldots, y_{i_1}, \ldots, \perp_{i_k}, \ldots, y_m) & \longrightarrow & f(l_1 \ldots l_n \mid y_1, \ldots, y_{i_1}, \ldots, u_{i_k}, \ldots, y_m) \end{array}$$

durch *eine einzige* Regel ersetzt werden:

$$f(l_1 \ldots l_n \mid y_1, \ldots, \perp_{i_1}, \ldots, \perp_{i_k}, \ldots, y_m) \quad \longrightarrow \quad f(l_1 \ldots l_n \mid y_1, \ldots u_{i_1}, \ldots, u_{i_k}, \ldots, y_m)$$

Ein System, bei dem die überlappenden ρ'-Regeln auf diese Weise zusammengefaßt wurden, heißt im folgenden ρ'-komprimiert.

Bei den letzten Regeln ist auf ein weiteres Detail aufmerksam zu machen, das vielleicht ansonsten übersehen wird. Wie bereits weiter oben, Abschnitt 3.3, angesprochen wurde, ist es bei konstruktorbasierten Systemen unnötig, den Kontext mit Hilfe eines Variablenvektors $(\vec{x})$ zwischenzuspeichern. Deshalb wurde dieser hier und auch im folgenden weggelassen, ohne besonders darauf hinzuweisen.

Definition 4.2.4: (Zusammengesetzte Transformation $\mathcal{Z}$)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein konstruktorbasiertes, lineares, bedingtes Termersetzungssystem. Dann entsteht $\mathcal{R}_{\mathcal{Z}} = \langle \Sigma_{\mathcal{Z}}, X, \mathrm{P}_{\mathcal{Z}} \rangle$ durch ρ'-Komprimierung und Weglassen der Variablenvektoren ($\vec{x}$) des Systems $(\mathcal{R}_{\mathcal{K}})_{\mathcal{V}}$. ◇

Das letzte Beispiel wird hier nun weitergeführt, die restlichen Schritte von $\mathcal{Z}$ dargestellt.

Beispiel 4.2

$$\mathcal{R}_{\mathcal{Z}} = \left(\begin{array}{lcl} nil([]) & \longrightarrow & true \\ nil(x : xs) & \longrightarrow & false \\ & & \\ append(xs, ys \mid \bot, \bot) & \longrightarrow & append(xs, ys \mid nil(xs), match(xs, * : *)) \\ append(xs, ys \mid true, y_2) & \longrightarrow & ys \\ append(xs, ys \mid y_1, true) & \longrightarrow & proj_1(xs) : append(proj_2(xs), ys \mid \bot, \bot) \\ & & \\ true \wedge x & \longrightarrow & x \\ match(x_1 : x_2, y_1 : y_2) & \longrightarrow & match(x_1, y_1) \wedge match(x_2, y_2) \\ match(x, *) & \longrightarrow & true \\ proj_1(x_1 : x_2) & \longrightarrow & x_1 \\ proj_2(x_1 : x_2) & \longrightarrow & x_2 \end{array} \right)$$

Der nun folgende Satz behauptet die wichtige Eigenschaft der Transformation $\mathcal{Z}$. Mit dieser kann ein schwach orthogonales System in ein disjunktes überführt werden.

Satz 4.2: (Transformation schwach orthogonaler Systeme)
Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein konstruktorbasiertes, schwach orthogonales, bedingtes Termersetzungssystem.
Dann ist $\mathcal{R}_{\mathcal{Z}}$ disjunkt bezüglich der Menge der sinntragenden Terme $T(\mathcal{R}, x \mapsto x^{\bot})$. ◇

Beweis:
Nach Konstruktion überlappen in $\mathrm{P}_{\mathcal{Z}}$ jetzt nur noch ρ''-Regeln. Seien $\{i_1, \ldots, i_k\}$ die Indizes dieser Regeln. Sie haben die Form

$$\begin{array}{lcl} \rho_{f,i_1} : f(l_1 \ldots l_n \mid y_1, \ldots, v_{i_1}, \ldots, y_{i_k}, \ldots, y_m) & \longrightarrow & r_{i_1} \\ \vdots & & \\ \rho_{f,i_k} : f(l_1 \ldots l_n \mid y_1, \ldots, y_{i_1}, \ldots, v_{i_k}, \ldots, y_m) & \longrightarrow & r_{i_k} \end{array}$$

Da aber laut Voraussetzung $\mathcal{R}$ schwach orthogonal ist, die Bedingungen sich also gegenseitig ausschließen, kann kein Term der Form

$$f(t_1 \ldots t_n \mid t_{n+1}, \ldots, v_{i_k}, \ldots, v_{i_k}, \ldots, t_{n+m})$$

aus einem $\bot$-Term abgeleitet werden. □

Damit ist das Ziel dieses Abschnittes erreicht. Im nächsten Abschnitt wird untersucht, unter welchen Umständen eine Transformation immer ein schwach orthogonales Zielsystem besitzen kann.

4.3 Schwache Orthogonalität des Zielsystems

Um schwache Orthogonalität des Zielsystems zu gewährleisten, muß auch noch die zweite Quelle für Überlappungen ausgeschlossen werden, die ρ''-Regeln. Dazu bietet sich intuitiv eine Möglichkeit an, die einer ähnlichen Idee folgt, wie die der ρ'-Komprimierung.

Unter gewissen Umständen ist es möglich, auch die bedingten Terme in den ρ''-Regeln und damit die bedingten Stellen insgesamt zusammenzufassen. Dies ist analog zu der Verfahrensweise der Transformation $\mathcal{GM}$, siehe Abschnitt 2.2.

Gilt nämlich für *alle überlappenden Regeln* eines Symbols f

$$\rho_1 : l_1 \longrightarrow r_1 \ (if \ u_1 \xrightarrow{*} v_1), \ldots, \rho_n : l_n \longrightarrow r_n \ (if \ u_n \xrightarrow{*} v_n)$$

die Bedingungen $l_i = l_j (= f(t_1, \ldots, t_n))$ und, falls vorhanden, $u_i = u_j (= u)$ sowie $v_i \neq v_j$ für alle $i, j \in \{1, \ldots, n\}$, so lassen sich die Regeln $n + m$ aus $\mathrm{P}_{\mathcal{V}}$ in die $n + 1$ Regeln

$$\begin{array}{rlcl}
\rho'_f : & f(t_1^X, \ldots, t_n^X \mid \ldots, \bot, \ldots) & \longrightarrow & f(t_1^X, \ldots, t_n^X \mid \ldots, u, \ldots) \\
\rho''_{f,1} : & f(t_1^X, \ldots, t_n^X \mid \ldots, v_1, \ldots) & \longrightarrow & r_1^\bot \\
& & \vdots & \\
\rho''_{f,i} : & f(t_1^X, \ldots, t_n^X \mid \ldots, x, \ldots) & \longrightarrow & r_i^\bot \\
& & \vdots & \\
\rho''_{f,n} : & f(t_1^X, \ldots, t_n^X \mid \ldots, v_n, \ldots) & \longrightarrow & r_n^\bot
\end{array}$$

zusammenfassen. $\rho''_{f,i}$ ist dabei ein Beispiel einer unbedingte Regel für f.

Ergebnis: Das Zielsystem ist schwach orthogonal.

Diese Einschränkung ist allerdings sehr restriktiv. Es läßt sich allerdings ein allgemeineres Kriterium für das bedingte System finden; die eben behandelte Möglichkeit erweist sich dann als Spezialfall dieses Kriteriums.

Dazu ist es zunächst erforderlich, die strikte Gleichheit in ein bedingtes Termersetzungssystem einzubauen.

Definition 4.3.1: (Strikte Gleichheit $\mathcal{R}_{==}$)
Sei $\mathcal{R} = \langle \mathcal{C} \cup \mathcal{D}, X, \mathrm{P} \rangle$ ein konstruktorbasiertes bedingtes Termersetzungssystem.
Dann ist $\mathcal{R}_{==} = \langle (\mathcal{C} \cup \{true^{(0)}, false^{(0)}\}) \cup (\mathcal{D} \cup \{==^{(2)}, \wedge^{(2)}\}), X, \mathrm{P}_{==} \rangle$, wobei $\mathrm{P}_{==}$ für jede Regel

$$l \longrightarrow r \ if \ u \xrightarrow{*} v$$

aus P eine Regel

$$l \longrightarrow r \ if \ (u == v) \xrightarrow{*} true$$

enthält, sowie die zusätzlichen Regeln

$$\begin{array}{llll}
c == c & \longrightarrow & true & c^{(0)} \in \mathcal{C} \\
c(x_1, \ldots, x_s) == c(y_1, \ldots, y_s) & \longrightarrow & (x_1 == y_1) \wedge \ldots \wedge (x_s == y_s) & c^{(s>0)} \in \mathcal{C} \\
b(x_1, \ldots, x_r) == c(y_1, \ldots, y_s) & \longrightarrow & false, & b \neq c, b^{(r \geq 0)}, c^{(s \geq 0)} \in \mathcal{C} \\
true \wedge x & \longrightarrow & x & \\
false \wedge x & \longrightarrow & false &
\end{array}$$

für jeden Konstruktor c, der in einem v-Term einer Regel aus P vorkommt und alle $b \in \mathcal{C}$ ◇

Analog zu dem Beweis zu Satz 4.1 gilt auch hier für alle Terme $s, t \in \mathcal{T}_\Sigma(X)$ $s \xrightarrow{*}_{\mathcal{R}} t$ genau dann, wenn $s \xrightarrow{*}_{\mathcal{R}_{==}} t$. Die Ableitungen haben auch dieselbe Länge, da die $==$-Ausdrücke nur innerhalb von Bedingungen auftreten.

Angewandt auf Beispiel 4.1 ergibt sich

$$(\mathcal{R}_{\mathcal{K}})_{==} = \left(\begin{array}{lll} nil([]) & \longrightarrow & true \\ nil(x:xs) & \longrightarrow & false \\ append(xs,ys) & \longrightarrow & ys\ if\ (nil(xs) == true) \xrightarrow{*} true \\ append(xs,ys) & \longrightarrow & proj_1(xs) : append(proj_2(xs), ys) \\ & & \qquad if\ (match(xs, * : *) == true) \xrightarrow{*} true \\ true \wedge x & \longrightarrow & x \\ false \wedge x & \longrightarrow & false \\ match(x_1 : x_2, y_1 : y_2) & \longrightarrow & match(x_1,y_1) \wedge match(x_2,y_2) \\ match(x,*) & \longrightarrow & true \\ proj_1(x_1 : x_2) & \longrightarrow & x_1 \\ proj_2(x_1 : x_2) & \longrightarrow & x_2 \\ (true == true) & \longrightarrow & true \\ (false == true) & \longrightarrow & false \\ (* == true) & \longrightarrow & false \\ ([] == true) & \longrightarrow & false \\ (x_1 : x_2 == true) & \longrightarrow & false \end{array}\right)$$

Anzumerken ist, daß es nur deshalb zu so vielen Regeln für == kommt, weil es keinerlei Typen gibt, die eine Gleichsetzung von etwa $true$ und [] ausschließen würden.

Definition 4.3.2: (Systeme mit endlichem Fehlschlag)

Sei $\mathcal{R} = \langle \Sigma, X, \mathrm{P} \rangle$ ein konstruktorbasiertes, schwach orthogonales, bedingtes Termersetzungssystem.
Dann heißt $\mathcal{R}$ schwach orthogonal und konstruktorbasiert mit endlichem Fehlschlag (kurz sokmeF), falls für alle Terme $t \in \mathcal{T}_\Sigma(X)$, alle Stellen p von t und alle Regeln $\rho : l \longrightarrow r\ if\ u \xrightarrow{*} v \in \mathrm{P}$ gilt:

Ist σ eine Substitution mit $\sigma(l) = t|_p$ aber v ist *nicht* aus $\sigma(u)$ ableitbar, dann folgt:
In $\mathcal{R}_{==}$ kann $\sigma(u) == v$ nach endlich vielen Schritten zu $false$ abgeleitet werden. ◇

Ist ein Termersetzungssystem sokmeF, so kann das Ergebnissystem eine schwach orthogonale Form annehmen. Dies geschieht mit einer Modifikation von $\mathcal{Z}$.

Definition 4.3.3: (Transformation $\mathcal{Z}'$)

Sei $\mathcal{R}$ ein sokmeF-Termersetzungssystem und das durch ρ'-Komprimierung und Weglassen der Variablenvektoren ($\vec{x}$) von $((\mathcal{R}_{\mathcal{K}})_{==})_{\mathcal{V}}$ entstandene System $\langle \Sigma, X, \mathrm{P} \rangle$.
Dann ist $\mathcal{R}_{\mathcal{Z}'} = \langle \Sigma, X, \mathrm{P}_{\mathcal{Z}'} \rangle$, wobei $\mathrm{P}_{\mathcal{Z}'}$ für alle überlappenden ρ''-Regeln der Form

$$\begin{array}{rcl} \rho''_{f,i_1} : f(l_1 \dots l_n \mid y_1, \dots, true_{i_1}, \dots, y_{i_k}, \dots, y_m) & \longrightarrow & r_{i_1} \\ \vdots & & \\ \rho''_{f,i_k} : f(l_1 \dots l_n \mid y_1, \dots, y_{i_1}, \dots, true_{i_k}, \dots, y_m) & \longrightarrow & r_{i_k} \end{array}$$

in P statt dessen Regeln der Form

$$\begin{array}{rcl} \rho''_{f,i_1} : f(l_1 \dots l_n \mid y_1, \dots, true_{i_1}, \dots, false_{i_k}, \dots, y_m) & \longrightarrow & r_{i_1} \\ \vdots & & \\ \rho''_{f,i_k} : f(l_1 \dots l_n \mid y_1, \dots, false_{i_1}, \dots, true_{i_k}, \dots, y_m) & \longrightarrow & r_{i_k} \end{array}$$

enthält, ansonsten aber gleich P ist. ◇

Da nunmehr die letzten Überlappungen entfernt wurden, ist das Ergebnissystem der Transformation $\mathcal{Z}'$ schwach orthogonal. Entsprechend der Konstruktion läßt sich auch jede Ableitung in $\mathcal{R}_{\mathcal{Z}}$ im System $\mathcal{R}_{\mathcal{Z}'}$ nachempfinden.

Beispiel für die Anwendung von $\mathcal{Z}'$:

Beispiel 4.3

$$\mathcal{R}'_{\mathcal{Z}} = \left(\begin{array}{lll}
nil([]) & \longrightarrow & true \\
nil(x : xs) & \longrightarrow & false \\
 & & \\
\multicolumn{3}{l}{append(xs, ys \mid \bot, \bot)} \\
\multicolumn{3}{l}{\qquad\qquad \longrightarrow\ append(xs, ys \mid nil(xs) == true, match(xs, * : *) == true)} \\
append(xs, ys \mid true, false) & \longrightarrow & ys \\
append(xs, ys \mid false, true) & \longrightarrow & proj_1(xs) : append(proj_2(xs), ys \mid \bot, \bot) \\
 & & \\
true \wedge x & \longrightarrow & x \\
false \wedge x & \longrightarrow & false \\
match(x_1 : x_2, y_1 : y_2) & \longrightarrow & match(x_1, y_1) \wedge match(x_2, y_2) \\
match(x, *) & \longrightarrow & true \\
proj_1(x_1 : x_2) & \longrightarrow & x_1 \\
proj_2(x_1 : x_2) & \longrightarrow & x_2 \\
(true == true) & \longrightarrow & true \\
(false == true) & \longrightarrow & false \\
(* == true) & \longrightarrow & false \\
([] == true) & \longrightarrow & false \\
(x_1 : x_2 == true) & \longrightarrow & false
\end{array}\right)$$

Die Forderung nach dem endlichen Fehlschlag ist sicherlich sehr hart. Eine andere, allgemeinere Möglichkeit, ein schwach orthogonales System als Ergebnis der Transformation zu erhalten, ist aber aufgrund der Unentscheidbarkeit für den Ausschluß der Bedingungen schwer vorstellbar.

Kapitel 5

Zusammenfassung

5.1 Ergebnisse

Nachdem in Kapitel 1 eine Einführung in den Problembereich dieser Arbeit gegeben wurde, enthielt Kapitel 2 eine Darstellung bereits veröffentlichter Verfahren. In den Kapiteln 3 und 4 wurden dann eigene Ergebnisse erzielt.

Die Ergebnisse dieser Arbeit auf einen Blick:

- Die Transformation $\mathcal{V}$, vorgestellt in Abschnitt 2.4, ist *konfluenzerhaltend* bezüglich der Klasse der *konstruktorbasierten* Termersetzungssysteme (Abschnitt 3.3).
- Der Beweis für dieses Ergebnis erlaubte die Möglichkeit, $\mathcal{V}$ für konstruktorbasierte Systeme anzupassen. So gilt:
  - In *konstruktorbasierten* Systemen ist *keine Kontextsicherung* in Variablenvektoren notwendig.
  - Die *Auswertung von Bedingungen für konstruktorbasierte Systeme* erfordert *kein Backtracking* in Form von Neuinitialisierung bedingter Stellen.
- $\mathcal{V}$ ist *terminierungserhaltend* bezüglich der Klasse der *reduzierenden* Termersetzungssysteme, Abschnitt 3.4. Der ursprünglich von Viry erbrachte Beweis erwies sich als fehlerhaft.
- In Abschnitt 3.4 wurde eine Transformation $\mathcal{B}$ vorgestellt, welche folgende Eigenschaften hat:
  - $\mathcal{B}$ ist *korrekt und vollständig* bezüglich der *gesamten* Klasse bedingter Termersetzungssysteme.
  - $\mathcal{B}$ ist *konfluenzerhaltend* bezüglich der *gesamten* Klasse bedingter Termersetzungssysteme.
  - $\mathcal{B}$ ist *terminierungserhaltend* bezüglich der Klasse der *reduzierenden* Termersetzungssysteme.
  - Für *konstruktorbasierte* Ausgangssysteme ergeben sich *keine zusätzlichen Ableitungsmöglichkeiten* im Zielsystem von $\mathcal{B}$.
- In Abschnitt 4.2 wurde eine Transformation $\mathcal{Z}$ vorgestellt, die *schwach orthogonale, konstruktorbasierte* Termersetzungssysteme in *disjunkte* Termersetzungssysteme überführt. Dieses Ergebnis dürfte für den Bereich der *funktional-logischen* Programmierung

von Interesse sein, stellt es doch einen großen Schritt in Richtung einer Implementierung der bedingten Teile mit der Effizienz funktionaler Programme dar.

- In Abschnitt 4.3 wurde eine Transformation $\mathcal{Z}'$ vorgestellt, welche die vermutlich größte Klasse bedingter Termersetzungssysteme, für die das gewährleistet werden kann, in *schwach orthogonale, konstruktorbasierte* Systeme überführt. Es handelt sich bei dieser Klasse um die *schwach orthogonalen, konstruktorbasierten Systeme mit endlichem Fehlschlag (sokmeF)*.

Außerdem wurde in Abschnitt 3.5 eine Abstraktion dessen versucht, was aus dieser Arbeit allgemein für das Aufgabenfeld der Transformation ausdrucksstärkerer Termersetzungssysteme in Standardsysteme von Interesse sein dürfte.

5.2 Transformationen

Es wurden im Rahmen dieser Arbeit eine ganze Reihe von Transformationen betrachtet. Diese werden in diesem Abschnitt zusammen mit ihren wichtigsten Eigenschaften tabellarisch erfaßt.

Allgemeine Transformationen bedingter Systeme in unbedingte				
Name	Korrekt	Vollständig	Konfluenzerhaltend	Terminierungserhaltend
$\mathcal{BK}$	-	Unbedingter Teil orthogonal	-	-
$\mathcal{GM}$	-	Easily safely transformable	Easily safely transformable	Easily safely transformable
$\mathcal{V}$	alle	alle	Konstruktorbasiert	Reduzierend
$\mathcal{V}'$	alle	alle	alle	nie
$\mathcal{B}$	alle	alle	alle	Reduzierend

„Allgemein“ heißt in diesem Zusammenhang der Umstand, daß mit der Transformation keine bestimmte Eigenschaft des Zielsystems, wie etwa die Orthogonalität, gewährleistet werden sollte.

Um eine bestimmte Art von Zielsystem zu gewährleisten, wurden in dieser Arbeit vorgeschaltete Transformationen verwendet, die jeweils eine Vereinheitlichung ansonsten problematischer Aspekte der Ausgangssysteme bewirken. Durch diese Technik ist es möglich, den Bereich der Ausgangssysteme zu vergrößern, vergleiche vor allem Abschnitt 4.3. Transformationen dieser Art sind die *Konformation* $\mathcal{K}$ und die strikte Gleichheit $==$.

Transformationen innerhalb der bedingten Systeme		
Name	Voraussetzungen	Zweck
$\mathcal{K}$	Konstruktorbasiert und linear	Vereinheitlichung überlappender linker Regelseiten
$==$	Konstruktorbasiert	Vereinheitlichung der Bedingungen

Mit Vorschaltung dieser Transformationen ist es möglich, bestimmte Eigenschaften des Zielsystems zu gewährleisten. Diese Eigenschaften werden von den Transformationen $\mathcal{Z}$ und $\mathcal{Z}'$ genutzt, um jeweils in eine bestimmte Klasse von unbedingten Systemen zu überführen.

Korrektheit, Vollständigkeit sowie Konfluenzerhaltung erben $\mathcal{Z}$ und $\mathcal{Z}'$ von der zugrundeliegenden Transformation $\mathcal{V}$.

Gezielte Transformationen bedingter Systeme in unbedingte		
Name	Voraussetzung an das Ausgangssystem	Eigenschaften Zielsystem
$\mathcal{Z}$	Konstruktorbasiert, schwach orthogonal	Schwach disjunkt (und konstruktorbasiert)
$\mathcal{Z}'$	Schwach orthogonal und konstruktorbasiert mit endlichem Fehlschlag (sokmeF)	Schwach orthogonal (und konstruktorbasiert)

5.3 Weitere Forschungsansätze

Wie wahrscheinlich jede wissenschaftliche Arbeit, schließt sich auch an diese eine Reihe von Fragestellungen an, deren Beantwortung eine sinnvolle Weiterführung der hier erzielten Ergebnisse bedeuten würde. Dazu seien hier einige erwähnt.

- Extravariablen
  Diese finden häufige Verwendung in logischen Programmen, und eine entsprechende Erweiterung der hier betrachteten Termersetzungssysteme wäre daher von Interesse. Unter Extravariablen versteht man dabei solche, die nicht in der linken Seite einer Regel $l \longrightarrow r \; if \; u \stackrel{*}{\longrightarrow} v$, wohl aber in den anderen Termen u oder gar r vorkommen. Extravariablen in u sind in der ursprünglichen Version von $\mathcal{V}$ [Vir95] erlaubt. Eine Arbeit über die Elimination von Extravariablen ist [Han95].

- Narrowing für disjunkte Systeme
  Als Ziel für die bedingte Version schwach orthogonaler Termersetzungssysteme wurde hier der Begriff des schwach disjunkten Systems eingeführt. Es dürfte für die weitere Nutzung der hier erzielten Ergebnisse von Interesse sein, zu untersuchen, ob und inwieweit existierende Narrowingverfahren, siehe [Han94], auf dieser Klasse von Termersetzungssystemen vollständig und optimal arbeiten.

- Regelanzahl und Ableitungslänge
  Mit Vollständigkeit und Korrektheit einer Transformation ist noch nichts darüber gesagt, ob das Zielsystem dem Ausgangssystem auch effizient entspricht. Wären für die Wiedergabe von n bedingten Regeln etwa 2^n unbedingte erforderlich gewesen, so wären Zweifel an einer Entsprechungseffizienz wohl angebracht. Eine andere Meßlatte stellt die (minimale, maximale, mittlere) Länge der Ableitungen im Zielsystem dar, die jeweils einer Ableitung im Ausgangssystem entsprechen. Mit Hilfe derartiger Untersuchungen kann der „Preis" abgeschätzt werden, der für die Transformation eines ausdrucksstärkeren Termersetzungssystems in ein Standardsystem zu entrichten ist.

- Größere Klassen bedingter Systeme
  Wie schon bei Einführung des Begriffes des bedingten Termersetzungssystems (Abschnitt 1.4) ausgeführt, sind einige Verallgemeinerungen des Begriffes denkbar. Ob und wie die hier erzielten Ergebnisse auf solchen Systemen zu relativieren sind, könnte von weiterem Interesse sein.

- Allgemeine Theorie der Transformation ausdrucksstärkerer Systeme
  In Abschnitt 3.5 wurde bereits angedacht, die hier verwendeten Techniken auf eine

abstrakte Form zu bringen. Dies könnte ein erster Schritt auf dem Weg zu einer allgemeinen Theorie für die Ausdrucksstärke verschiedener Klassen von Termersetzungssystemen sein.
Möglicherweise existiert eine „Hierarchie der Ausdrucksstärke“, welche dann Anlaß zur Präzision des Begriffes der „sprachlichen Ausdrucksstärke überhaupt“ führen könnte. Doch bis dahin ist es ein weiter Weg, denn es heißt nicht umsonst:

Auch die längste Reise beginnt mit dem ersten Schritt [Lao].

Literaturverzeichnis

[AGM90] Hitoshi Aida, Joseph Goguen, and José Meseguer. Compiling concurrent rewriting onto the rewrite rule machine. In Stéphane Kaplan and Mitsuhiro Okada, editors, *Conditional and Typed Rewriting Systems, 2nd International Workshop*, LNCS 516, pages 320–332, Montreal, Canada, June 11–14, 1990. Springer-Verlag.

[BA95] Franz Baader and Can Adam Albayrak. *Termersetzungssysteme*. Aachener Beitraege zur Informatik. Verlag der Augustinus Buchhandlung, Aachen, 1995.

[BK86] J. A. Bergstra and J. W. Klop. Conditional rewrite rules: Confluence and termination. *Journal of Computer and System Sciences*, 32(3):323–362, June 1986.

[Bro96] *Brockhaus Enzyklopaedie*. F.A. Brockhaus, Mannheim, 1986-1996. 19. Auflage.

[BT87] J. A. Bergstra and J. V. Tucker. Algebraic specifications of computable and semicomputable data types. *Theoretical Computer Science*, 50(2):137–181, 1987.

[Cla88] Volker Claus. *Duden Informtik*. Dudenverlag, Mannheim/Zuerich/Wien, 1988.

[DO90] N. Dershowitz and M. Okada. A rationale for conditional equational programming. *Theoretical Computer Science*, 75(1-2):111–138, September 1990.

[DP86] Nachum Dershowitz and David A. Plaisted. Equational programming. Technical Report 1265, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1986.

[GM87] Elio Giovannetti and Corrado Moiso. Notes on the elimination of conditions. In S. Kaplan and J.-P. Jouannaud, editors, *Proceedings of the First International Workshop on Conditional Term Rewriting Systems (Orsay, France)*, volume 308 of *Lecture Notes in Computer Science*, pages 91–97, Berlin, July 1987. Springer-Verlag.

[Han94] Michael Hanus. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 19,20:583–628, 1994.

[Han95] Michael Hanus. On extra variables in (equational) logic programming. In *Proc. Twelfth International Conference on Logic Programming*, pages 665–679. MIT Press, 1995.

[Han97] Michael Hanus. A unified computation model for functional and logic programming. In *Conference Record of POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 80–93, Paris, France, 15–17 January 1997.

[Hin95] C. Hintermeier. How to transform canonical decreasing HCTRs into equivalent canonical TRSs. *Lecture Notes in Computer Science*, 968:186–, 1995.

[KLMNRA89] Herbert Kuchen, Rita Loogen, Juan Jose Moreno-Navarro, and Mario Rodriguez-Artalejo. Graph-based Implementation of a Functional Logic Language. Technical Report 89-20, Aachener Informatik-Berichte, RWTH Aachen, Fachgruppe Informatik, 1989.

[Lao] Laotse. *Tao te king*. Eugen Diederichs Verlag, Muenchen.

[Llo94] J. W. Lloyd. Practical advantages of declarative programming. In *Joint Conference on Declarative Programming, GULP-PRODE'94*, 1994.

[Thi94] Peter Thiemann. *Grundlagen der funktionalen Programmierung*. Teubner, Stuttgart, 1994.

[Vir95] Patrick Viry. Elimination of conditions. Technical Report TR-95-19, Pisa University, Italy, December 1, 1995.