Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Unendliche Reduktionen in der kombinatorischen Logik

Diplomarbeit

Leipzig, im Juni 2002

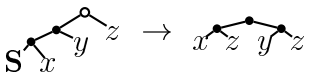
vorgelegt von

Dörges, Till
geb. am 18.06.1976

Studiengang Informatik

Einleitung

Gegenstand dieser Arbeit ist $\mathbf{CL}(\mathbf{S})$, das Termersetzungssystem mit der einzigen Regel



Sie ist ein Schritt auf dem Weg, an dessen Ziel man sich die Lösung des Wortproblems für $\mathbf{CL}(\mathbf{S})$ erhofft.

Waldmann zeigte in [Wal97], daß in $\mathbf{CL}(\mathbf{S})$ die Termination entscheidbar ist und daß jede unendliche Reduktion top-terminiert.

Daraus folgt, daß unendliche Normalformen existieren und das Wortproblem gelöst wäre, könnte man diese Normalformen effektiv finden und vergleichen.

Deswegen untersuche ich in der vorliegenden Arbeit unendliche Reduktionsketten in $\mathbf{CL}(\mathbf{S})$.

Es wird vermutet, daß sie im wesentlichen durch Muster wie z.B. $\mathsf{M}^{n+1} :=$  beschreibbar sind.

Eine durch Muster verursachte unendliche Reduktionskette mit Startterm t ist typischerweise nach folgendem (oder einem ähnlichen) Schema aufgebaut:

$$\begin{aligned} \exists j \geq 0 \quad &: \quad t \leadsto^c \mathsf{M}^0 K[\mathsf{M}^j] \\ \forall Z, \forall i > 0 \quad &: \quad \mathsf{M}^i Z \leadsto^c \mathsf{M}^{i-1} Z \\ \forall j \geq 0 \quad &: \quad \mathsf{M}^0 K[\mathsf{M}^j] \leadsto^c \mathsf{M}^j K[\mathsf{M}^{j+1}] \end{aligned}$$

Darüberhinaus wird vermutet, daß tatsächlich jeder Term aus $\mathbf{CL}(\mathbf{S})$ ohne endliche Normalform eine unendliche Reduktionskette mit einem Muster Muster aufweist.

Ich präsentiere vier Beweisschemata, mit deren Hilfe sich unendliche Reduktionsketten in allen bisher untersuchten Termen ohne Normalform beweisen lassen.

Zudem definiere ich das Konzept der allgemeinsten Muster, mit denen sich Muster klassifizieren lassen, und beweise, daß die Menge der allgemeinsten Muster unendlich ist.

Außerdem beschreibe ich, wie man Muster automatisch findet und verifiziert.

Schließlich stelle ich einen Beweisansatz vor, mit dem die zwingende Existenz von Mustern belegbar wäre. Als eine Folgerung aus diesem Beweis ergäbe sich weiterhin die Schleifenfreiheit von $\mathbf{CL}(\mathbf{S})$.

Grundsätzlich ist die Untersuchung unendlicher Reduktionsketten interessant, weil sie eng mit der Frage der *Termination* von Programmen verwandt ist. Das Wortproblem bezeichnet die Frage nach der *Äquivalenz* von Termen – also Programmen. Es ist im Rahmen von Programmtransformationen bedeutsam.

$\mathbf{CL}(\mathbf{S})$ im besonderen ist ein Termersetzungssystem, für das über die Äquivalenzfrage noch nicht ausreichend viele Informationen bekannt sind. Die Untersuchungen von Waldmann lassen vermuten, daß $\mathbf{CL}(\mathbf{S})$ „in der Nähe“ bisher bekannter Klassen von Termersetzungssystemen liegt.

Im allgemeinen gilt ein Termersetzungssystem R als beherrschbar, wenn sich die transitive Hülle der Ersetzungsrelation $\rightarrow_R$ effektiv beschreiben läßt. Dies bedeutet z. B. für eine Termmenge M: Ist M einfach beschreibbar, so ist es auch $\rightarrow_R^*(M)$. Beispiele für beherrschbare Klassen sind inverse monadische bzw. invers wachsende Termersetzungssysteme. Leider ist dies weder für $\mathbf{CL}(\mathbf{S})$ noch für das Termersetzungssystem $\mathbf{CL}(\mathbf{S})^{-1}$, das durch die Umkehrung der $\mathbf{S}$-Regel erzeugt wird, der Fall. Immerhin aber zeigte Waldmann, daß die Menge der Vorgänger aller Terme mit einer Normalform aus $\mathbf{CL}(\mathbf{S})$ regulär ist, weshalb auch die Frage der Termination für $\mathbf{CL}(\mathbf{S})$ entscheidbar ist.

Um das Wortproblem zu entscheiden, gibt es mehrere Ansätze. Neben der Möglichkeit, effektiv vergleichbare Normalformen zu finden, ist eine weitere Variante, für zwei Terme jeweils die Menge der Nachfolger zu erzeugen und zu prüfen, ob der Schnitt dieser Mengen leer ist.

Es wird vermutet, daß mindestens einer dieser Wege gangbar ist, auch wenn die Nachfolgermengen von Termen aus $\mathbf{CL}(\mathbf{S})$ nicht regulär sind.

Inhaltsverzeichnis

Teil I.

Grundlagen und Definitionen

Kapitel 1

Abstrakte Reduktionssysteme

1.1. Einleitung

Definition 1.1.1 *Abstrakte Reduktionssysteme* (ARS) sind Tupel $(A, \rightarrow)$, wobei A eine beliebige Menge und $\rightarrow$ eine binäre Relation auf A ist.

ARS „wissen“ nichts über die interne Struktur von A, sondern berücksichtigen lediglich die Relation $\rightarrow$.

Der Begriff „Reduktion“ könnte zu der Annahme verleiten, daß ein Element $x \in A$ im Laufe einer Reduktion, also der kontinuierlichen Anwendung von $\rightarrow$, immer kleiner wird. Dies muß aber nicht zwingend so sein, wie das Beispiel $0 \rightarrow 1 \rightarrow 2 \rightarrow \ldots$ sofort veranschaulicht.

Auch kann es in vielen Fällen sehr sinnvoll sein, nicht nur $\rightarrow$, sondern $\leftrightarrow^*$, d. h., den reflexiven, symmetrischen und transitiven Abschluß von $\rightarrow$ zu betrachten. Damit wäre eine Reduktionskette eines ARS nichts weiter als ein Pfad, dem bidirektional gefolgt werden kann.

Besonders interessiert ist man in der Regel an fundierten Relationen, also solchen, deren Reduktionsketten in endlicher Zeit terminieren.

Dieses Kapitel ist bewußt knapp gehalten und soll lediglich die nötigen Definitionen und Notationen einführen. Es lehnt sich stark an [Wal97] an. Wer an einer tiefergehenden Betrachtung von ARS interessiert ist, dem seien z.B. [Ave95] oder [BN98] empfohlen.

In Ermangelung einer geeigneteren Stelle möchte ich hier noch grundsätzlich anmerken:

- Das Vokabular in den Bereichen Termersetzungssysteme und kombinatorische Logik ist für die englische Sprache quasi standardisiert. Für das Deutsche ist dies bislang nicht der Fall. Zwar habe ich versucht, Begriffe aus der Literatur zu übernehmen; aber nicht immer war dies möglich.
- $\mathbb{N}$ bezeichnet die Menge der natürlichen Zahlen einschließlich 0.

1.2. Relationen

1.2.1. Konstruktion

Definition 1.2.1 Eine binäre *Relation* R auf einer Menge M ist eine Teilmenge von $M \times M$.

Mit $R := \to$ ist es oft bequemer, $a \to_R b$ oder noch einfacher $a \to b$ statt $(a, b) \in R$ zu schreiben.

Ebenfalls aus Gründen der Bequemlichkeit wird im weiteren Verlauf der Arbeit gelegentlich darauf verzichtet werden, die Menge M, auf der die Relation R (bzw. $\to$) definiert ist, explizit zu benennen. Dies allerdings nur, wenn M aus dem Kontext ersichtlich ist.

Definition 1.2.2 Die *Komposition* zweier Relationen $R \subseteq A \times B$ und $S \subseteq B \times C$ ist definiert durch

$$R \circ S := \{(x, z) \in A \times C \mid \exists y \in B : \ (x, y) \in R \wedge (y, z) \in S\}$$

Definition 1.2.3 (Identität) $\to^0 := \{(a, a) \mid x \in M\}$

Definition 1.2.4 (Inversion) $R^{-1} := \{(b, a) \mid (a, b) \in R\}$

Anstatt $\to^{-1}$ verwende ich auch $\leftarrow$.

Definition 1.2.5 ($i+1$-fache Komposition) $\to^{i+1} := \to \circ \to^i$ für $i \geq 0$

Definition 1.2.6 (Reflexiver Abschluß) $\to^= := \to \cup \to^0$

Definition 1.2.7 (Symmetrischer Abschluß) $\leftrightarrow := \to \cup \leftarrow$

Definition 1.2.8 (Transitiver Abschluß) $\to^+ := \bigcup_{k>0} \to^k$

Definition 1.2.9 (Reflexiver, transitiver Abschluß) $\to^* := \bigcup_{k\geq 0} \to^k$

Ebenfalls gebräuchlich ist $\twoheadrightarrow$ anstelle von $\to^*$.

1.2.2. Eigenschaften

Definition 1.2.10 (Transitivität) $\to$ ist *transitiv* gdw. $\to \circ \to \subseteq \to$.

Definition 1.2.11 (Symmetrie) $\to$ ist *symmetrisch* gdw. $\to = \leftarrow$.

Definition 1.2.12 (Antisymmetrie) $\to$ ist *antisymmetrisch* gdw. $\to \bigcap \leftarrow \subseteq \to^0$

Definition 1.2.13 (Asymmetrie) $\rightarrow$ ist *asymmetrisch* gdw. $\rightarrow \bigcap \leftarrow = \emptyset$

Gelegentlich findet man auch die Bezeichnungen antisymmetrisch anstatt asymmetrisch und schwach antisymmetrisch anstatt antisymmetrisch.

Definition 1.2.14 (Reflexivität) $\rightarrow$ ist *reflexiv* gdw. $\rightarrow^0 \subseteq \rightarrow$

Definition 1.2.15 (Irreflexivität) $\rightarrow$ ist *irreflexiv* gdw. $\rightarrow^0 \bigcap \rightarrow = \emptyset$

Definition 1.2.16 (Konnexivität) $\rightarrow$ ist *konnex* gdw. $\forall a, b \in M$ mit $a \neq b$ entweder $a \rightarrow b$ oder $b \rightarrow a$ gilt.

Satz 1.2.17 Ist eine Relation R über einer Menge M irreflexiv und transitiv, so ist R auch asymmetrisch.

Beweis: Angenommen, R wäre nicht asymmetrisch, es gäbe also $a, b \in M$ mit $(a, b) \in R$ sowie $(b, a) \in R$, gemäß der Transitivität von R müßte dann auch $(a, a) \in R$ gelten, was im Widerspruch zur Irreflexivität stünde. □

1.3. Ordnungen

Ordnungen werden ganz natürlich durch Relationen eingeführt.

Definition 1.3.1 (Quasiordnung) Eine Relation ist eine *Quasiordnung* gdw. sie reflexiv und transitiv ist.

Definition 1.3.2 (Partielle Ordnung) Eine Relation ist eine *partielle Ordnung* gdw. sie antisymmetrisch und eine Quasiordnung ist.

Definition 1.3.3 (Striktordnung) Eine Relation ist eine *Striktordnung* oder *Halbordnung* gdw. sie irreflexiv und transitiv ist.

Den *strikten* Teil jeder partiellen Ordnung R erhält man, indem man die Identitätsrelation abzieht, also $R \setminus R^0$.

Definition 1.3.4 (Äquivalenz) Eine Relation ist eine *Äquivalenz* gdw. sie reflexiv, symmetrisch und transitiv ist.

Definition 1.3.5 (Totale Ordnung) Eine Relation ist eine *totale Ordnung* (oder *lineare Ordnung*) gdw. sie partielle Ordnung und konnex ist.

Definition 1.3.6 (Minima) Sei R eine partielle Ordnung über M. Dann ist die Menge der *Minima* von M:

$$\min_R(M) := \{x \in M \mid \neg\exists y \in M, x \neq y : (y, x) \in R\}$$

Die $x \in \min_R(M)$ heißen *minimal bezüglich* R.

Definition 1.3.7 (Maxima) Sei R eine partielle Ordnung über M. Dann ist die Menge der *Maxima* von M:

$$\max_R(M) := \{x \in M \mid \neg\forall y \in M, x \neq y : (x, y) \in R\}$$

Die $x \in \max_R(M)$ heißen *maximal bezüglich* R.

Definition 1.3.8 (Minimum, Maximum) Gilt $\min_R(M) = \{x\}$, heißt x das *Minimum* bezüglich R. Analog für *Maximum*.

Bemerkung 1.3.9 Die Definitionen für Minima, Maxima, Minimum und Maximum lassen sich unverändert für Striktordnungen übernehmen.

Bemerkung 1.3.10 Ist R zusätzlich konnex und endlich, gibt es immer ein Minimum und ein Maxium. Es gilt also $|\min_R(M)| = 1$ bzw. $|\max_R(M)| = 1$.

Beispiel 1.3.11 $<$ auf $\mathbb{N}$ hat das Minimum 0, aber kein Maximum. $<$ auf $\mathbb{Z}$ hat weder ein Minimum noch ein Maximum.

1.4. Pfade

Relationen, die mittels der Definitionen aus Abschnitt 1.2.1 konstruiert werden, lassen sich sehr gut als Pfad oder Weg zwischen zwei Elementen veranschaulichen:

Definition 1.4.1 Seien $\to$ eine Relation auf M und $a, b \in M$.

- $a \to^n b$ heißt, daß es einen *Pfad* der Länge n von a nach b gibt.
- $a \to^* b$ bedeutet, daß es einen endlichen Pfad von a nach b gibt.
- $a \to^+ b$ ist analog zu $a \to^* b$, jedoch darf der Pfad nicht leer sein.

Definition 1.4.2 (Nachfolger, Vorgänger) Gilt $a \to^+ b$, so heißt b *Nachfolger* von a. Gilt sogar $a \to b$, heißt b *unmittelbarer Nachfolger* von a. Analog ist a der *Vorgänger* bzw. *unmittelbare Vorgänger* von b.

Definition 1.4.3 (Normalform) $x \in M$ ist genau dann in *Normalform*, wenn es keinen Nachfolger hat.

Definition 1.4.4 $x \in M$ *besitzt die Normalform* y gdw. es ein $y \in M$ gibt, so daß $x \to^* y$ und y in Normalform ist.

Definition 1.4.5 $x \in M$ *normalisiert* gdw. x eine Normalform besitzt.

Definition 1.4.6 $\to$ auf M ist *normalisierend* gdw. alle $x \in M$ normalisieren.

Definition 1.4.7 $x \in M$ *terminiert* gdw. es keinen unendlichen Pfad $x \to x_1 \to x_2 \to x_3 \to \ldots$ gibt.

Definition 1.4.8 $\to$ auf M ist *terminierend* gdw. alle $x \in M$ terminieren.

1.5. Fundierte Relationen

Im Englischen werden fundierte Relationen als „well-founded relations“ bezeichnet. Die „fundierte Induktion“ ist vielleicht auch bekannter als „Noether-Induktion“[1].

Definition 1.5.1 (Minimal) $x \in M$ ist *minimal für* $\to$ gdw. es keine Nachfolger hat, wenn x also in Normalform ist.

Eigentlich entspricht „minimal“ aus Definition 1.5.1 am ehesten dem Begriff „maximal“ gemäß Definition 1.3.7, was natürlich unglücklich ist. Im Gegensatz zu den Definitionen 1.3.6 und 1.3.7 werden bei 1.5.1 keine weiteren Eigenschaften für $\to$ verlangt, wie z.B., daß $\to$ eine partielle Ordnung sein müßte.

Definition 1.5.2 (Fundiert) Eine Relation $\to$ über M ist *fundiert* gdw. jede nichtleere Teilmenge $B \subseteq M$ ein minimales Element nach Definition 1.5.1 enthält.

Satz 1.5.3 Eine Relation ist terminierend gdw. sie fundiert ist.

Für den Beweis sei z.B. auf [BN98] verwiesen.

1.6. Konfluenz und Termination

Da sich Pfade auch als Beschreibung einer Berechnung ansehen lassen, ist von besonderem Interesse, unter welchen Voraussetzungen ein Pfad nur eine endliche Länge besitzt, also die Rechnung terminiert. Außerdem möchte man wissen, ob es eventuell Verzweigungen entlang des Pfades gibt, ob also unterschiedliche Rechnungen für dieselbe Eingabe ausgeführt werden können. Gibt es Verzweigungen, so ist die Frage nach der Eindeutigkeit der Ergebnisse von großer Wichtigkeit.

[1] Im Gedenken an die Mathematikerin Emmy Noether

Definition 1.6.1 (Diamant-Eigenschaft) Eine Relation $\rightarrow$ hat die *Diamant-Eigenschaft* gdw. $\leftarrow \circ \rightarrow \subseteq \rightarrow \circ \leftarrow$

Definition 1.6.2 Eine Relation $\rightarrow$ ist *lokal konfluent* gdw. $\leftarrow \circ \rightarrow \subseteq \rightarrow^* \circ \leftarrow^*$

Definition 1.6.3 (Konfluenz) $\rightarrow$ ist *konfluent* gdw. $\leftarrow^* \circ \rightarrow^* \subseteq \rightarrow^* \circ \leftarrow^*$

Definition 1.6.4 (Church–Rosser) Eine Relation $\rightarrow$ hat die *Church-Rosser-Eigenschaft* gdw. $\leftrightarrow^* \subseteq \rightarrow^* \circ \leftarrow^*$

Definition 1.6.5 (Konvergenz) Eine Relation $\rightarrow$ ist *konvergent* gdw. sie terminierend und konfluent ist.

Ohne Beweis zitiere ich die nächsten Sätze:

Satz 1.6.6 Wenn die Relation $\rightarrow$ die Diamant-Eigenschaft hat, dann hat auch $\rightarrow^*$ die Diamant-Eigenschaft.

Satz 1.6.7 Genau dann, wenn $\rightarrow$ die Church-Rosser-Eigenschaft hat, ist $\rightarrow$ auch konfluent.

Satz 1.6.8 (Newmann-Lemma) Ist $\rightarrow$ terminierend, dann fallen Konfluenz und lokale Konfluenz zusammen.

Satz 1.6.9 Ist $\rightarrow$ konfluent, dann hat jedes $x \in M$ höchstens eine Normalform.

Kapitel 2

Termersetzungssysteme

2.1. Einleitung

Termersetzungssysteme sind abstrakte Reduktionssysteme (s. Abschnitt 1.1), für die die Menge A explizit als Menge von Termen angegeben ist. D. h., zusätzlich zu der externen Struktur, die durch die Relation R eingeführt wird, verfügen die Terme über eine interne Struktur in Form der Subtermeigenschaft.

In diesem Kapitel führe ich wiederum nötige Definitionen sowie Notationen ein. Bei Letzterem orientiere ich mich im wesentlichen an [DJ91].

Was weiterführende Literatur anbetrifft, so werden in [Wec92] universelle Algebren sehr detailliert behandelt. Als gute Einführung zur Termersetzung sei [BN98] empfohlen. Dershowitz und Jouannaud schließlich geben in [DJ90] ebenfalls einen lesenswerten Überblick.

2.2. Termalgebra

2.2.1. Terme

Definition 2.2.1 (Signatur) Eine *Signatur* ist ein Tupel (a, Σ), wobei Σ eine Menge von *Funktionssymbolen* und a eine Abbildung ist, so daß $\forall f \in \Sigma : \ \mathrm{a}(f) \in \mathbb{N}$ gilt. $\mathrm{a}(f)$ ist die *Stelligkeit* oder *Arität* der Funktion f. Die f mit $\mathrm{a}(f) = 0$ heißen *Konstanten*.

Mittels Signaturen ist es nun möglich, ganz allgemein Mengen von *Termen* zu definieren.

Definition 2.2.2 Gegeben seien eine Signatur $\mathcal{S} := (\mathrm{a}, \Sigma)$ und eine Menge von *Variablen* $\mathcal{V}$. Es gelte ferner $\Sigma \cap \mathcal{V} = \emptyset$, was sich leicht durch Umbenennung erreichen läßt. Dann wird die *Termmenge* $\mathcal{T}(\mathcal{S}, \mathcal{V})$ über $\mathcal{S}$ und $\mathcal{V}$ induktiv definiert:

$$\begin{aligned} \forall v \in \mathcal{V} \quad &: \quad v \in \mathcal{T}(\mathcal{S}, \mathcal{V}) \\ \forall f \in \Sigma,\ n = \mathrm{a}(f),\ \forall t_1, \ldots, t_n \in \mathcal{T}(\mathcal{S}, \mathcal{V}) \quad &: \quad f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{S}, \mathcal{V}) \end{aligned}$$

D. h., jede Variable ist automatisch ein Term.

Definition 2.2.3 Die *Menge der Variablen* eines Terms t wird mit $\mathrm{Var}(t)$ bezeichnet.

Definition 2.2.4 (Grundterme) Die Menge der *Grundterme* ist gegeben durch $\mathcal{T}(\mathcal{S}, \emptyset)$. Mit anderen Worten, Grundterme enthalten keine Variablen. Sie bestehen ausschließlich aus Konstanten.

Definition 2.2.5 (Größe) Die *Größe* eines Terms $t \in \mathcal{T}(\mathcal{S}, \mathcal{V})$ ist gegeben durch:

$$\begin{aligned} \forall v \in \mathcal{V} &: \mathrm{size}(v) := 1 \\ \forall f \in \Sigma \text{ mit } \mathrm{a}(f) = 0 &: \mathrm{size}(f) := 1 \\ \forall t_i \in \mathcal{T}(\mathcal{S}, \mathcal{V}) &: \mathrm{size}(f(t_1, \ldots, t_n)) := 1 + \mathrm{size}(t_1) + \ldots + \mathrm{size}(t_n) \end{aligned}$$

Ebenfalls induktiv eingeführt wird die *Tiefe* eines Terms:

Definition 2.2.6 (Tiefe)

$$\begin{aligned} \forall v \in \mathcal{V} &: \mathrm{depth}(v) := 0 \\ \forall f \in \Sigma \text{ mit } \mathrm{a}(f) = 0 &: \mathrm{depth}(f) := 0 \\ \forall t_i \in \mathcal{T}(\mathcal{S}, \mathcal{V}) &: \mathrm{depth}(f(t_1, \ldots, t_n)) := 1 + \max(\mathrm{depth}(t_1), \ldots, \mathrm{depth}(t_n)) \end{aligned}$$

2.2.2. Positionen

Positionen sind sehr stark mit dem Listenkonzept funktionaler Programmiersprachen wie etwa Haskell oder Lisp verwandt, weswegen ich nun ganz allgemein Listen definieren werde:

Definition 2.2.7 Eine *Liste* wird mittels der *Konstruktoren* : (*cons*) und [] (*leere Liste*) definiert. Eine Liste mit Elementen des Typs a ist als

$$[a] := \begin{array}{ll} [] & \text{oder} \\ a : [a] & \end{array}$$

definiert. Zwei Listen sind identisch, wenn sie elementweise gleich sind.

Es führte an dieser Stelle zu weit, tiefer in die Typtheorie einzusteigen. Insbesondere Listen sollten in jedem grundlegenden Werk zu funktionaler Programmierung behandelt werden. Für eine Einführung in die funktionale Programmierung mit Haskell ist [Hud00] lesenswert.

Beispiel 2.2.8 Die Liste mit den natürlichen Zahlen 1,2 und 3 wäre Definition 2.2.7 entsprechend $1 : (2 : (3 : []))$. Vereinfachend schreibt man dies als $[1, 2, 3]$.

Definition 2.2.9 Eine Liste $a = [a_1, \ldots, a_l]$ ist *echtes Präfix* einer Liste $b = [b_1, \ldots, b_l, \ldots, b_n]$, geschrieben $a <_P b$, gdw.

$$\exists l : \forall i \in \{1, \ldots, l\} : a_i = b_i \ \wedge \ n > l$$

Die leere Liste [] ist echtes Präfix jeder Liste, außer von sich selbst.

Satz 2.2.10 $<_P$ auf einer Menge von Listen ist eine Striktordnung.

Beweis:

- Transitivität: Für alle Listen a, b, c mit $a <_P b$ und $b <_P c$ sei o.B.d.A. a die kürzeste und c die längste Liste, da die Listen laut Definition verschieden lang sein müssen. Ist $a = []$, folgt die Transitivität unmittelbar aus der Definition. Sonst gibt es $i < j < k$, so daß

  $$\begin{aligned} a &= [a_1, \ldots, a_i] \\ b &= [b_1, \ldots, b_i, \ldots, b_j] \\ c &= [c_1, \ldots, c_i, \ldots, c_j, \ldots, c_k] \end{aligned}$$

  Und es gilt nach Definition

  $$\forall l \in \{1, \ldots, i\} : \qquad a_l = b_l = c_l$$

- Irreflexivität: Für alle Listen a, b mit $a <_P b$ gilt $a \neq b$, da b gemäß Definition länger als a ist. □

Definition 2.2.11 Zwei Listen a und b haben das *gemeinsame echte Präfix* c gdw. $c <_P a$ und $c <_P b$.

Nach diesem kurzen Exkurs in die Welt der Listen kehre ich zum eigentlichen Thema des Abschnitts zurück.

Definition 2.2.12 Gegeben sei $\mathcal{T}(\mathcal{S}, \mathcal{V})$, dann ist die Menge aller *Positionen* eines Terms:

$$\begin{aligned} \forall v \in \mathcal{V} \ &: \ \mathrm{Pos}(v) := \{()\} \\ \forall t_i \in \mathcal{T}(\mathcal{S}, \mathcal{V}) \ &: \ \mathrm{Pos}(f(t_1, \ldots, t_n)) := \{()\} \ \cup \bigcup_{i \in \{1,\ldots,n\}, s \in \mathrm{Pos}(t_i)} (i : s) \end{aligned}$$

() heißt *Wurzelposition*. Sie entspricht der leeren Liste.

Eine Position ist also nichts weiter als eine Liste natürlicher Zahlen, wobei ich mir aus satztechnischen Gründen erlaubt habe, ‚[' bzw. ‚]' durch ‚(' bzw. ‚)' zu ersetzen. Mit dem Positionskonzept ist es möglich, bestimmte Teile eines Terms eindeutig zu identifizieren.

Definition 2.2.13 Seien $p_a = (a_1, \ldots, a_l, \ldots, a_m)$ und $p_b = (b_1, \ldots, b_l, \ldots, b_n)$ Positionen, dann ist p_a *links* von p_b, geschrieben $p_a <_L p_b$, gdw.

$$\exists l : \forall i \in \{1, \ldots, l\} : a_i = b_i \ \wedge \ a_{l+1} < b_{l+1}$$

Das echte gemeinsame Präfix kann auch leer sein.

Satz 2.2.14 $<_L$ auf einer Menge von Positionen ist eine Striktordnung.

Beweis:

- Transitivität: Gegeben seien drei Positionen $p_a = (a_1, \ldots, a_i)$, $p_b = (b_1, \ldots, b_j)$ und $p_c = (c_1, \ldots, c_k)$ mit $p_a <_L p_b$ sowie $p_b <_L p_c$. Es gilt:

$$\begin{aligned} \exists l \in \{1, \ldots, \min(i,j)\} &: a_l < b_l \ \wedge \ \neg\exists x < l \ : \ a_x \neq b_x \\ \exists m \in \{1, \ldots, \min(j,k)\} &: b_m < c_m \ \wedge \ \neg\exists y < m \ : \ b_y \neq c_y \end{aligned}$$

  Also gilt auch:

$$\forall x \in \{1, \ldots, \min(l,m) - 1\} : \qquad a_x = b_x = c_x$$

  Ist $m > l$, folgt $b_l = c_l$. Andernfalls, also $m \leq l$, müssen laut Definition $a_m = b_m$ und $b_m < c_m$ gelten. Daraus folgt, $p_a <_L p_c$.

- Irreflexivität: Qua Definition unterscheiden sich zwei Positionen, die zur Relation gehören, an mindestens einer Stelle. □

Definition 2.2.15 Die Position p_a ist *oberhalb* der Position p_b, geschrieben $p_a <_O p_b$, gdw. $p_a <_P p_b$. p_a ist *unterhalb* von p_b gdw. $p_b <_O p_a$.

Definition 2.2.16 Sei $t \in \mathcal{T}(\mathcal{S}, \mathcal{V})$ ein Term und $p \in \mathrm{Pos}(t)$ eine Position. p heißt

- *außen* gdw. $p \in \min\limits_{<_O}(\mathrm{Pos}(t))$.
- *innen* gdw. $p \in \max\limits_{<_O}(\mathrm{Pos}(t))$.
- *links* gdw. $p \in \min\limits_{<_L}(\mathrm{Pos}(t))$.
- *rechts* gdw. $p \in \max\limits_{<_L}(\mathrm{Pos}(t))$.

Bemerkung 2.2.17 Ein Term kann jeweils mehrere äußere, innere, linke oder rechte Positionen enthalten.

Bemerkung 2.2.18 Die Konzepte innen/außen und links/rechts sind voneinander unabhängig.

Beispiel 2.2.19 Seien $P := \{(), (1,2), (1,2,3), (1,2,4)\}$ und $Q := \{(2,3), (3,4)\}$ Mengen von Positionen. Dann gibt es in P drei linke Positionen, nämlich $\{(), (1,2), (1,2,3)\}$ und in Q zwei äußere sowie zwei innere Positionen, nämlich jeweils $\{(2,3), (3,4)\}$. Die linke äußere Position in Q ist $(2,3)$.

Satz 2.2.20 Für jede nicht-leere Menge P von Positionen gilt

$$|\min_{<_L}(\min_{<_O}(P))| = 1 \tag{2.1}$$
$$|\max_{<_L}(\max_{<_O}(P))| = 1 \tag{2.2}$$
$$|\max_{<_L}(\min_{<_O}(P))| = 1 \tag{2.3}$$
$$|\min_{<_L}(\max_{<_O}(P))| = 1 \tag{2.4}$$

Beweis: Die erste Aussage läßt sich folgendermaßen beweisen:

- Gilt $() \in P$, dann folgt $\min_{<_O}(P) = \{()\}$, da die leere Position () Präfix jeder Position außer von sich selbst ist.
- Gilt $() \notin P$, dann gilt für alle $p_a, p_b \in \min_{<_O}(P)$ entweder $p_a <_L p_b$ oder $p_b <_L p_a$, woraus folgt, daß $|\min_{<_L}(\min_{<_O}(P))| = 1$.

Die restlichen Beweise funktionieren analog. □

2.2.3. Subterme

Definition 2.2.21 (Subterm) Seien s und t Terme aus $\mathcal{T}(\mathcal{S}, \mathcal{V})$. Der *Subterm* s von t an Position p, geschrieben $s = t[p]$, ist:

$$\begin{aligned} t[()] &:= t \\ p_1 \in \{1,\ldots,n\}: \; f(t_1,\ldots,t_n)[(p_1,\ldots,p_j)] &:= t_{p_1}[(p_2,\ldots,p_j)] \end{aligned}$$

Definition 2.2.22 Die Menge aller Positionen eines bestimmten Subterms s in t ist definiert durch: $\mathrm{Pos}(s,t) := \{p \in \mathrm{Pos}(t) \mid t[p] = s\}$

Definition 2.2.23 (Ersetzung) $t[p \leftarrow t']$ ist der Term, der durch die *Ersetzung* des Terms t' in t an der Position p entsteht:

$$\begin{aligned} t[() \leftarrow t'] &:= t' \\ f(t_1,\ldots,t_n)[(p_1,\ldots,p_j) \leftarrow t'] &:= f(\ldots, t_{p_1}[(p_2,\ldots,p_j) \leftarrow t'], \ldots) \end{aligned}$$

Definition 2.2.24 (Subtermrelation) s ist Subterm von t, geschrieben $s \trianglelefteq t$, gdw. es $p \in \mathrm{Pos}(t)$ gibt, so daß $t[p] = s$.

2.3. Substitutionen

Definition 2.3.1 (Substitution) Für eine Menge von Termen $\mathcal{T}(\mathcal{S},\mathcal{V})$ ist eine *Substitution* eine Abbildung $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{S},\mathcal{V})$, wobei es nur endlich viele $v \in \mathcal{V}$ geben darf, die nicht auf sich selbst abgebildet werden. Um σ auf beliebige Terme zu erweitern, wird σ' eingeführt – wie immer induktiv:

$$\begin{aligned} \forall v \in \mathcal{V} &: \sigma'(v) := \sigma(v) \\ \forall t_i \in \mathcal{T}(\mathcal{S},\mathcal{V}) &: \sigma'(f(t_1,\ldots,t_n)) := f(\sigma'(t_1),\ldots,\sigma'(t_n)) \end{aligned}$$

Der Einfachheit halber wird σ' als σ geschrieben. Ebenfalls sehr gebräuchlich ist es, anstatt $\sigma(X)$ die Postfixnotation $X\sigma$ zu verwenden.

Definition 2.3.2 (Unifikator) Ein *Unifikator* zweier Terme t_1 und t_2 ist eine Substitution σ, so daß $t_1\sigma = t_2\sigma$. t_1 und t_2 heißen *unifizierbar*. Für Mengen M von Termen mit $|M| > 2$ wird diese Definition kanonisch erweitert.

Definition 2.3.3 Ein Unifikator σ einer Termmenge M ist der *allgemeinste Unifikator* gdw. es für alle Unifikatoren σ' von M eine Substitution θ gibt, so daß $\sigma' = \sigma\theta$.

Sind die Terme einer Menge unifizierbar, ist der allgemeinste Unifikator berechenbar[1]. Für diesbezügliche Details sei auf [Sch95] verwiesen. Unifikation wird im folgenden lediglich in Abschnitt 2.5 benötigt.

Definition 2.3.4 (Kontext) Sei $\mathcal{T}(\mathcal{S},\mathcal{V})$ eine Termmenge entsprechend Definition 2.2.2. Dann ist ein *Kontext* ein Term aus $\mathcal{T}(\mathcal{S}',\mathcal{V})$, mit $\mathcal{S}' = (\Sigma \cup \{[]\}, \mathrm{a})$ und $\mathrm{a}([]) = 0$, in dem [] mindestens einmal vorkommt.

Ist K ein Kontext, so bezeichnet $K[X]$ das Anwenden der Substitution $\sigma([]) = X$ auf K.

Anstatt [] kann man sich auch eine ausgezeichnete Variable innerhalb des Terms vorstellen.

2.4. Termersetzungssysteme

Definition 2.4.1 (Ersetzungsregel) Eine *Ersetzungsregel* ist ein Paar $(l,r) \in \mathcal{T}(\mathcal{S},\mathcal{V})^2$, mit $\mathrm{Var}(l) \supseteq \mathrm{Var}(r)$ und $l \notin \mathcal{V}$. Meistens wird (l,r) als $l \to r$ geschrieben.

[1] Dies ist z. B. das Herzstück jedes Prolog-Interpreters.

Definition 2.4.2 Ein *Termersetzungssystem* (TES oder TRS[2]) ist eine Menge von Ersetzungsregeln.

Definition 2.4.3 Ein TES R definiert für die Termmenge $\mathcal{T}(\mathcal{S},\mathcal{V})$ die sogenannte *Ersetzungsrelation*:

$$\rightarrow_R := \{(t_1,t_2) \in \mathcal{T}(\mathcal{S},\mathcal{V})^2 \mid \exists (l,r) \in R : \exists p \in \mathrm{Pos}(t_1) : \exists \sigma : t_1[p] = l\sigma \wedge t_2 = t_1[p \leftarrow r\sigma]\}$$

$l\sigma$ heißt *Redex*[3] und $r\sigma$ *Kontraktum*. p ist die *Redexposition*.

$t_1 \rightarrow_R t_2$ bedeutet also das Anwenden einer Regel aus R auf den Term t_1, so daß t_2 daraus entsteht. Dies nennt man auch *reduzieren*.

$t_1 \rightarrow_R t_2 \rightarrow_R \ldots$ oder kürzer $t_1 \rightarrow t_2 \rightarrow \ldots$ heißt *Reduktionskette* oder *Reduktionspfad*.

Definition 2.4.4 Für ein TES R mit Ersetzungsrelation $\rightarrow$ heißt ein Term q von einem Term s *erreichbar*, geschrieben $s \rightsquigarrow q$, gdw. es einen Term p gibt, so daß $s \rightarrow^* p$ und $q \trianglelefteq p$.

q ist von s in *höchstens n Schritten erreichbar*, geschrieben $s \rightsquigarrow^n q$, gdw. es einen Term p gibt, so daß $s \rightarrow^i p$ mit $i \leq n$ und $q \trianglelefteq p$.

Bemerkung 2.4.5 Eine alternative Formulierung ist: $s \rightsquigarrow q$ gdw. es einen Kontext K gibt, so daß $s \rightarrow^* K[q]$. Analog für $s \rightsquigarrow^n q$.

Im folgenden werde ich kurz einige wichtige Eigenschaften von Ersetzungsregeln auflisten:

Definition 2.4.6 Ein Term $t \in \mathcal{T}(\mathcal{S},\mathcal{V})$ ist *linear*, gdw. $\forall v \in \mathcal{V} : |\mathrm{Pos}(v,t)| \leq 1$.

Definition 2.4.7 Eine Ersetzungsregel (l,r) eines TES R heißt *linkslinear*, wenn l linear ist. Analog für *rechtslinear*. Die Regel (l,r) heißt *linear*, wenn l und r linear sind und $\mathrm{Var}(l) = \mathrm{Var}(r)$.

Definition 2.4.8 Eine Regel (l,r) heißt *Linksgrundregel* gdw. l ein Grundterm ist. *Rechtsgrundregel* sowie *Grundregel* analog.

Definition 2.4.9 Eine Regel (l,r) heißt *expandierend* gdw. $\exists v \in \mathcal{V} : 1 \leq |\mathrm{Pos}(v,l)| < |\mathrm{Pos}(v,r)|$.

Definition 2.4.10 *Löschend* ist eine Regel (l,r) gdw. $\exists v \in \mathcal{V} : |\mathrm{Pos}(v,l)| > |\mathrm{Pos}(v,r)| = 0$.

[2] vom englischen „Term Rewrite System“

[3] vom englischen „reducible expression“ (reduzierbarer Ausdruck)

Definition 2.4.11 Eine Regel (l, r) heißt *kollabierend* gdw. $r \in \mathcal{V}$.

Diese Definitionen für einzelne Regeln lassen sich in folgender Weise auf das jeweils ganze TES ausdehnen.

Definition 2.4.12 Ein TES heißt linkslinear gdw. **alle** seine Regeln linkslinear sind. Dies gilt analog für rechtslinear, linear, Linksgrund-, Rechtsgrund- und Grund-TES.

Definition 2.4.13 Ein TES heißt verdoppelnd (bzw. löschend, kontrahierend) gdw. **wenigstens eine** seiner Regeln verdoppelnd (bzw. löschend, kontrahierend) ist.

Definition 2.4.14 Ein TES heißt *terminierend* gdw. die Ersetzungsrelation $\rightarrow_R$ terminierend ist.

2.5. Konfluenz

Der Grund für das große Interesse an Konfluenz im Zusammenhang mit TES, ist die Tatsache, daß bei konfluenten TES die Wahl des Pfades beliebig ist. Es ist also egal, welchen Redex man jeweils kontrahiert, da am Ende der Berechnung immer derselbe Term steht; natürlich nur, wenn das TES terminierend ist.

Nicht-konfluente TES entstehen z.B. aus *überlappenden* Regeln, so daß die Anwendung einer Regel das Verschwinden eines anderen Redexes zur Folge hat. Dies darf aber auf keinen Fall mit löschend verwechselt werden.

Definition 2.5.1 (Kritisches Paar) Gegeben seien ein TES R sowie

- $(l_1, r_1), (l_2, r_2) \in R$ mit $\mathrm{Var}(l_1) \cap \mathrm{Var}(l_2) = \emptyset$, was sich durch Umbenennung erreichen läßt.
- $p \in \mathrm{Pos}(l_1)$ mit $l_1[p] \notin \mathcal{V}$. $p \neq ()$ für den Fall, daß die Regeln identisch sind.
- $l_1[p]$ und l_2 sind unifizierbar mit allgemeinstem Unifikator σ.

$(l_1[p \leftarrow r_2]\sigma, r_1\sigma)$ heißt *kritisches Paar* von R.

Ein kritisches Paar heißt *konfluent* gdw. es einen gemeinsamen Nachfolger für $l_1[p \leftarrow r_2]\sigma$ und $r_1\sigma$ gibt. Ein anderer Ausdruck dafür ist: Das kritische Paar ist *zusammenführbar*.

Sind nicht alle kritischen Paare aus R konfluent, so ist R als Ganzes nicht lokal konfluent. Für den Fall, daß das ursprüngliche TES terminierend war, kann man durch sog. *Vervollständigung*, also die Hinzunahme weiterer Ersetzungsregeln, versuchen,

- die nicht-konfluenten kritischen Paare konfluent zu machen (*zusammenzuführen*) und
- $\leftrightarrow_R^*$ nicht zu verändern.

Gelingt dies, erhält man ein konvergentes TES, dessen Wortproblem durch Vergleich der Normalformen entscheidbar und äquivalent zum Wortproblem im Ursprungs-TES ist, weil $\leftrightarrow_R^*$ nicht verändert wurde.

Da allerdings weder $\mathbf{CL}(\mathbf{S})$ noch ein anderes kombinatorisches TES kritische Paare hat, werde ich nicht näher darauf eingehen. Mehr dazu findet sich in [BN98].

Definition 2.5.2 Ein TES ist *nicht-überlappend* gdw. es keine kritischen Paare hat.

Definition 2.5.3 Ein TES heißt *orthogonal* gdw. es nicht-überlappend und linkslinear ist.

Satz 2.5.4 Orthogonale TES sind konfluent.

Für den Beweis siehe wiederum [BN98].

2.6. Reduktionsstrategien

Definition 2.6.1 Die Menge aller *Redexpositionen* eines Terms t sei $\mathrm{RPos}(t) := \{p \in \mathrm{Pos}(t) \mid t[p] \text{ ist ein Redex}\}$.

Definition 2.6.2 Sei R ein TES und $t \in \mathcal{T}(\mathcal{S}, \mathcal{V})$ ein Term. $p \in \mathrm{RPos}(t)$ sei eine Redexposition. Der Redex $t[p]$ heißt *äußerer* Redex gdw. p außen ist. Analog für *innen*, *links* und *rechts*.

Definition 2.6.3 Sei R ein TES. Eine *Reduktionsstrategie* für R ist eine Abbildung s, die jedem $t \in \mathcal{T}(\mathcal{S}, \mathcal{V})$ mit $\mathrm{RPos}(t) \neq \emptyset$ ein $p \in \mathrm{RPos}(t)$ zuordnet.

Bemerkung 2.6.4 Die Reduktionsstrategie s erzeugt zu gegebenem t_1 eine eindeutige Reduktionskette $t_1 \to t_2 \to \ldots$, indem jedes t_i an der Position $\mathrm{s}(\mathrm{RPos}(t_i))$ reduziert wird.

Bemerkung 2.6.5 Der Begriff Reduktionsstrategie gemäß Definition 2.6.3 entspricht einer „one step reduction strategy“ laut [Bar84].

Definition 2.6.6 Sei R ein konfluentes TES. Eine Reduktionsstrategie s heißt *normalisierend* gdw. für alle $t \in \mathcal{T}(\mathcal{S}, \mathcal{V})$, die eine Normalform $u \in \mathcal{T}(\mathcal{S}, \mathcal{V})$ besitzen, $t \to_R^* u$ unter s gilt.

Definition 2.6.7 Die Reduktionsstrategie, die jedem Term $t \in \mathcal{T}(\mathcal{S}, \mathcal{V})$ mit $\mathrm{RPos}(t) \neq \emptyset$

- den linken äußeren Redex zuordnet, heißt $\mathrm{rs}_{\mathrm{LA}}(t) := \min_{<_L}(\min_{<_O}(\mathrm{RPos}(t)))$.

- den rechten inneren Redex zuordnet, heißt $\mathrm{rs}_{\mathrm{RI}}(t) := \max_{<_L}(\max_{<_O}(\mathrm{RPos}(t)))$.
- den rechten äußeren Redex zuordnet, heißt $\mathrm{rs}_{\mathrm{RA}}(t) := \max_{<_L}(\min_{<_O}(\mathrm{RPos}(t)))$.
- den linken inneren Redex zuordnet, heißt $\mathrm{rs}_{\mathrm{LI}}(t) := \min_{<_L}(\max_{<_O}(\mathrm{RPos}(t)))$.

Für weitergehende Informationen zu Reduktionsstrategien sind [Bar84] und [Gra92] gute Startpunkte.

2.7. Grundtermersetzungssysteme

Einerseits sind Grund-TES sehr viel weniger mächtig als TES mit Variablen; dafür werden jedoch u.U. einige Entscheidungsprobleme wie z.B. das Wortproblem handhabbarer.

Satz 2.7.1 Termination für Grund-TES ist entscheidbar.

Satz 2.7.2 Konfluenz für Grund-TES ist entscheidbar.

Beweise finden sich in [Oya87].

2.8. Entscheidungsprobleme

Definition 2.8.1 (Wortproblem) Für ein TES R mit $\mathcal{T}(\mathcal{S},\emptyset)$ und Ersetzungsrelation $\rightarrow_R$ wird als *Wortproblem* bezeichnet:

Eingabe: $t_1, t_2 \in \mathcal{T}(\mathcal{S},\emptyset)$

Ausgabe: `Ja`, falls $t_1 \leftrightarrow_R^* t_2$. `Nein` sonst.

Bemerkung 2.8.2 Ist ein TES terminierend und konfluent, dann ist das Wortproblem wg. Satz 1.6.9 trivialerweise entscheidbar, nämlich durch Ausrechnen und Vergleichen der jeweiligen Normalformen.

Definition 2.8.3 (Halteproblem) Für ein TES R mit $\mathcal{T}(\mathcal{S},\emptyset)$ und Ersetzungsrelation $\rightarrow_R$ wird als *Halteproblem* bezeichnet:

Eingabe: $t \in \mathcal{T}(\mathcal{S},\emptyset)$

Ausgabe: `Ja`, falls alle Reduktionsketten $t \rightarrow_R \ldots$ endlich sind. `Nein` sonst.

Kapitel 3
Kombinatorische Logik

3.1. Einleitung

Eingeführt wurde die kombinatorische Logik vor knapp 80 Jahren von Moses Schönfinkel in [Sch24] und später von vielen anderen, wie z.B. Haskell B. Curry in [Cur30] oder [CF58], beständig erweitert.

Ursprünglich verfolgte Schönfinkel zwei Ziele, die er mit der kombinatorischen Logik zu erreichen beabsichtigte. Zum einen wollte er durch eine variablenfreie Notation die Probleme mit dem Skopus gebundener und freier Variablen aus der Welt schaffen (daher „kombinatorisch“) und zum anderen eine entscheidbare Theorie konstruieren (daher „Logik“).

„Entscheidbar“ für sich ist natürlich ein unpräziser Begriff. In Bezug auf Logiken sind damit traditionellerweise die Fragen nach Erfüllbarkeit und Allgemeingültigkeit von Formeln gemeint. Bei Termersetzungssystemen werden unter diesem Begriff i.a. Wort- und Halteproblem verstanden.

Im Laufe der Jahre hat sich die Perspektive, aus der die kombinatorische Logik betrachtet wird, geändert. Heute wird sie vor allem als eine Menge von Termersetzungssystemen angesehen.

Sein erstes Ziel hat Schönfinkel, wenn auch um den Preis einer nicht ganz übersichtlichen Notation, erreicht. Der Erfolg des zweiten Ansinnens ist dagegen weniger einfach zu beurteilen. Wäre er vollständig gewesen, gäbe es beispielsweise diese Diplomarbeit nicht.

3.2. Kombinatorische Logik

Definition 3.2.1 Sei $\mathcal{S} := (\mathrm{a}, \Sigma)$ eine Signatur mit $\Sigma := (\cdot, C_1, \ldots, C_n)$ sowie $\mathrm{a}(\cdot) := 2$ und $\forall i: \ \mathrm{a}(C_i) := 0$. Sei ferner die Termmenge $\mathcal{T}(\mathcal{S}, \mathcal{V})$.

- – Die C_i nennt man *Kombinatoren*.
  – $\cdot$ heißt *Applikation*.
  – $\cdot$ wird ausschließlich in Infixnotation verwendet.

  - $\cdot$ ist *linksassoziativ*, d.h. $(\ldots(C_1 \cdot C_2) \cdot \ldots \cdot C_n)$ darf geschrieben werden als $C_1 \cdot C_2 \cdot \ldots \cdot C_n$.
  - $\cdot$ wird i.a. nicht aufgeschrieben, also $C_1 \cdot C_2 \cdot \ldots \cdot C_n = C_1 C_2 \ldots C_n$.
- Ein Term t heißt *Rückgrat*, wenn er von der Form $M_1 \ldots M_n$ ist. M_1 heißt *Kopf*. Die übrigen M_i heißen *Argumente*.
- Ein *purer Term* enthält lediglich $\cdot$ und Variablen, aber keine Kombinatoren.
- Die Regel (l, r) eines TES ist *kombinatorisch* gdw. l linear, ein Rückgrat, der Kopf von l ein Kombinator, die Argumente Variablen sind und r ein purer Term ist.
- Eine Menge $\{(l_1, r_1), \ldots, (l_n, r_n)\}$ von kombinatorischen Ersetzungsregeln mit paarweise verschiedenen Köpfen $\{C_1, \ldots, C_n\}$ der jeweiligen l_i nennt man *kombinatorische Logik* über der *Basis* $\{C_1, \ldots, C_n\}$, geschrieben $\mathbf{CL}(C_1, \ldots, C_n)$.
- $t \in \mathbf{CL}(C_1, \ldots, C_n)$ ist eine Abkürzung für: „Sei $\mathbf{CL}(C_1, \ldots, C_n)$ eine kombinatorische Logik mit Termmenge $\mathcal{T}(\mathcal{S}, \mathcal{V})$ und $t \in \mathcal{T}(\mathcal{S}, \mathcal{V})$."
- **CL** schließlich ist die Klasse aller kombinatorischen Logiken.

Schönfinkel, Curry et al. haben u.a. folgende Kombinatoren eingeführt:

I	x	$\to x$	(Identität)
K	xy	$\to x$	(Konstante)
S	xyz	$\to xz(yz)$	(Substitution)
C	xyz	$\to xzy$	(Komposition)
B	xyz	$\to x(yz)$	
J	$xyza$	$\to xy(xaz)$	
L	xy	$\to x(yy)$	

Inzwischen gibt es auch einen ornithologischen Ansatz zur Namensetymologie der Kombinatoren ([Smu85] und [Kee96]).

Satz 3.2.2 Jede kombinatorische Logik $\mathbf{CL}(C_1, \ldots, C_n)$ ist konfluent.

Beweis: Jedes kombinatorische TES, also jede kombinatorische Logik, ist orthogonal (vgl. Definition 2.5.3). Zusammen mit Satz 2.5.4 folgt daraus die Behauptung. □

Bemerkung 3.2.3 Kombinatorische TES sind i.a. nicht-terminierend.

3.2.1. Länge

In Anlehnung an Definition 2.2.5 führe ich ein:

Definition 3.2.4 Gegeben sei eine kombinatorische Logik $\mathbf{CL}(C_1, \ldots, C_n)$. Die *Länge* eines Terms $t \in \mathcal{T}(\mathcal{S}, \mathcal{V})$ ist:

$$\begin{aligned} \forall v \in \mathcal{V} &: \text{length}(v) := 1 \\ \forall f \in \Sigma \text{ mit } \text{a}(f) = 0 &: \text{length}(f) := 1 \\ \forall t_i \in \mathcal{T}(\mathcal{S}, \mathcal{V}) &: \text{length}(f(t_1, \ldots, t_n)) := \text{length}(t_1) + \ldots + \text{length}(t_n) \end{aligned}$$

length(t) ist nichts anderes als die Anzahl der Konstanten und Variablen in t. Der entscheidende Unterschied zu Definition 2.2.5 ist, daß Funktionssymbole mit einer Arität größer 0 nicht mitgezählt werden. Interessant wird dieser Unterschied bei einer Signatur, in der einstellige Funktionssymbole enthalten sind. Im Gegensatz zur Größe würde die Länge eines Terms durch beliebig oft wiederholte Anwendung eines einstelligen Funktionssymbols nicht verändert.

Bemerkung 3.2.5 Für jeden Grundterm t einer kombinatorischen Logik aus **CL** ergibt length(t) genau die Anzahl der Kombinatoren.

Bemerkung 3.2.6 Für eine beliebige kombinatorische Logik aus **CL** gibt es nur endlich viele Grundterme einer bestimmten Länge.

Lemma 3.2.7 Sei $R \in \mathbf{CL}$ eine kombinatorische Logik. Die Häufigkeit des Vorkommens von $\cdot$ innerhalb eines Terms t bezeichne ich mit app(t). Dann gilt:

$$\forall t \in \mathcal{T}(\mathcal{S}, \mathcal{V}) : \ \text{app}(t) + 1 = \text{length}(t)$$

Beweis: Per Induktion über die Termlänge:

- Der Basisfall folgt direkt aus der Definition von length():

$$\forall t \in \mathcal{T}(\mathcal{S}, \mathcal{V}), \text{length}(t) = 1 : \ \text{app}(t) = 0, \text{length}(t) = 1$$

- Sind s und t Terme mit app(s) $= m - 1$ und length(s) $= m$ bzw. app(t) $= n - 1$ und length(t) $= n$, dann gilt für die zusammengesetzten Terme $s \cdot t$ sowie $t \cdot s$

$$\begin{aligned} \text{app}(st) = \text{app}(ts) &= m + n - 2 + 1 \\ \text{length}(st) = \text{length}(ts) &= m + n \end{aligned}$$

□

Satz 3.2.8 Sei $R \in \mathbf{CL}$ eine kombinatorische Logik mit Termmenge $\mathcal{T}(\mathcal{S}, \mathcal{V})$. Dann gilt:

$$\forall t \in \mathcal{T}(\mathcal{S}, \mathcal{V}) : \ \text{size}(t) = 2\,\text{length}(t) - 1$$

Beweis: Sei app(t) wieder Häufigkeit des Vorkommens von $\cdot$ innerhalb eines Terms t. Es gilt size(t) = app(t) + length(t). Zusammen mit Lemma 3.2.7 ergibt sich daraus die Behauptung. □

3.2.2. Positionen

In **CL** gibt es lediglich ein Funktionssymbol mit einer von Null verschiedenen Stelligkeit: nämlich $\cdot$ mit $a(\cdot) = 2$. Hat ein Term also direkte Subterme, sind es stets zwei. Im folgenden werde ich die entsprechenden Positionen anstatt mit 1 und 2 mit L und R bezeichnen. Anstelle von (L, L, R, L) schreibe ich auch $LLRL$.

3.2.3. Redexe

Definition 3.2.9 Sei $R := \mathbf{CL}(C_1, \ldots, C_n)$, $t \in \mathcal{T}(\mathcal{S}, \mathcal{V})$, $p \in \mathrm{Pos}(t)$ und $t[p]$ ein Redex. Ist $p \in \{L\}^*$, besteht p also nur aus L, heißt $t[p]$ *Kopfredex*. Hat ein Term keinen solchen Redex, ist er in *Kopfnormalform*.

Bemerkung 3.2.10 Enthält t einen Kopfredex, hat t stets die Form $C_i M_1 \ldots M_n$.

3.2.4. Reduktionsstrategien

Da die folgenden Resultate allgemein bekannt sind, aber umfassende Beweise erfordern, zitiere ich sie an dieser Stelle lediglich. Für eine genauere Auseinandersetzung siehe wiederum [Bar84] und [Gra92].

Satz 3.2.11 Für jede kombinatorische Logik $R := \mathbf{CL}(C_1, \ldots, C_n)$ ist die Reduktionsstrategie $\mathrm{rs}_{\mathrm{LA}}()$ normalisierend.

Satz 3.2.12 Für jede nicht-löschende kombinatorische Logik $R := \mathbf{CL}(C_1, \ldots, C_n)$ ist jede Reduktionsstrategie s normalisierend.

Bemerkung 3.2.13 Für jede löschende kombinatorische Logik $R := \mathbf{CL}(C_1, \ldots, C_n)$ ist die Reduktionsstrategie $\mathrm{rs}_{\mathrm{RI}}()$ nicht-normalisierend. Sei z. B. $R := \mathbf{CL}(\mathbf{S}, \mathbf{K}, \mathbf{I})$ und $t := \mathbf{KI}(\mathbf{SSS}(\mathbf{SSS})(\mathbf{SSS}))$. Die Normalform von t ist $\mathbf{I}$. Diese wird mit $\mathrm{rs}_{\mathrm{RI}}()$ jedoch nicht erreicht, da $\mathbf{SSS}(\mathbf{SSS})(\mathbf{SSS})$ keine Normalform besitzt. Siehe auch Bemerkung 4.2.2.

3.3. Vollständigkeit

Definition 3.3.1 Ein kombinatorisches TES heißt *kombinatorisch vollständig* gdw. es für alle $n > 0$ und für alle puren Terme M mit $\mathrm{Var}(M) \subseteq \{x_1, \ldots, x_n\}$ einen Grundterm N gibt, so daß $N x_1 \ldots x_n \ \rightarrow^* \ M$.

Theorem 3.3.2 $\mathbf{CL}(\mathbf{S}, \mathbf{K}, \mathbf{I})$ ist kombinatorisch vollständig.

Der zugehörige Beweis ist klassisch:

Beweis: Sei R ein TES mit Termmenge $\mathcal{T}(\mathcal{S},\mathcal{V})$ und $\lfloor\cdot\rfloor_x : \mathcal{T}(\mathcal{S},\mathcal{V}) \to \mathcal{T}(\mathcal{S},\mathcal{V})$, $x \in \mathcal{T}(\mathcal{S},\mathcal{V})$ die nachfolgend definierte Abbildung. $\lfloor M\rfloor_x$ erzeugt einen Term, in dem x nicht mehr vorkommt, und für den $\lfloor M\rfloor_x\, x \to^* M$ gilt.

$$\begin{array}{rrcl} & \lfloor x\rfloor_x & := & \mathbf{I} \\ \forall x \notin M & \lfloor M\rfloor_x & := & \mathbf{K}M \\ & \lfloor MN\rfloor_x & := & \mathbf{S}\lfloor M\rfloor_x\lfloor N\rfloor_x \end{array}$$

Jetzt sei aus Definition 3.3.1 $N := \lfloor \ldots \lfloor\lfloor M\rfloor_{x_n}\rfloor_{x_{n-1}} \ldots \rfloor_{x_1}$. □

Beispiel 3.3.3 Sei $M = ba$. Dann sind

$$\begin{array}{rcl} \lfloor M\rfloor_b & = & \mathbf{SI}(\mathbf{K}a) \\ \lfloor\lfloor M\rfloor_b\rfloor_a & = & \mathbf{S}(\mathbf{K}(\mathbf{SI}))(\mathbf{S}(\mathbf{KK})\mathbf{I}) \end{array}$$

Korollar 3.3.4 $\mathbf{CL}(\mathbf{S},\mathbf{K})$ ist kombinatorisch vollständig.

Beweis: $\mathbf{SKK}\,x \to^2 x$ simuliert $\mathbf{I}\,x \to x$. □

Bemerkung 3.3.5 Mittels einer geeigneten Abbildung $[\cdot]$ lassen sich die natürlichen Zahlen $\mathbb{N}$ als Terme in $\mathbf{CL}(\mathbf{S},\mathbf{K})$ definieren. Sei $F \in \mathcal{T}(\mathcal{S},\mathcal{V})$ ein Term in $\mathbf{CL}(\mathbf{S},\mathbf{K})$. Dann sagt man, F *berechnet* die n-stellige Funktion f gdw.:

$$\forall x_i \in \mathbb{N}:\ F[x_1]\ldots[x_n] \to^* [f(x_1,\ldots,x_n)]$$

Theorem 3.3.6 Die in $\mathbf{CL}(\mathbf{S},\mathbf{K})$ berechenbaren Funktionen sind genau die partiell rekursiven Funktionen.

Um dieses zu zeigen, muß man eine bidirektionale Abbildung zwischen $\mathbf{CL}(\mathbf{S},\mathbf{K})$ sowie dem λ-Kalkül konstruieren und anschließend die Behauptung für das λ-Kalkül beweisen. Ersteres findet sich z.B. in [HS86] und letzteres in [Chu36].

Direkte Folgerungen aus Theorem 3.3.6 sind:

Satz 3.3.7 Das Wortproblem für $\mathbf{CL}(\mathbf{S},\mathbf{K})$ ist unentscheidbar.

Satz 3.3.8 Das Halteproblem für $\mathbf{CL}(\mathbf{S},\mathbf{K})$ ist unentscheidbar.

Dennoch gibt es kombinatorische TES, für die obige Fragen entscheidbar sind. Diese TES sind dann jedoch nicht vollständig. In einigen Fällen, z.B. für $\mathbf{CL}(\mathbf{I})$, sind die Beweise sogar trivial.

$\mathbf{CL}(\mathbf{J})$ sieht auf den ersten Blick komplizierter aus, da es expandierend ist. Probst und Studer beweisen jedoch in [PS01], daß $\mathbf{CL}(\mathbf{J})$ terminiert. Daraus folgt wegen Konfluenz unmittelbar auch die Entscheidbarkeit des Wortproblems.

Ein weiteres Beispiel ist $\mathbf{CL}(\mathbf{L})$, das nicht-terminierende Reduktionsketten enthält. Sprenger und Wymann-Böni zeigen aber in [SWB93] ein konkretes Entscheidungsverfahren für das Wortproblem.[1] Die Grundidee ist, das System $\mathbf{CL}(\mathbf{L})^{-1}$ mit der Inversen der Ersetzungsregel $\mathbf{L}$ zu betrachten. $\mathbf{CL}(\mathbf{L})^{-1}$ ist nicht konfluent, und der Standardvervollständiungsalgorithmus terminiert nicht. Trotzdem ist Vervollständigung möglich. Damit gelingt es, zu jedem Term aus $\mathbf{CL}(\mathbf{L})$ den „Startterm“ in $\mathbf{CL}(\mathbf{L})^{-1}$ zu finden und effektiv mit anderen zu vergleichen. Dieses Verfahren ist äquivalent zum Vergleichen von Normalformen, die nicht den Anfang, sondern das Ende der Reduktionskette bilden.

Das Halteproblem für $\mathbf{CL}(\mathbf{S})$ ist entscheidbar (vgl. [Wal98]). Für das Wortproblem scheint die Methode, das inverse System $\mathbf{CL}(\mathbf{S})^{-1}$ zu betrachten, leider nicht trivial zu sein, da $\mathbf{CL}(\mathbf{S})^{-1}$ zwar terminiert, aber nicht konfluent ist und auch keine Vervollständigung „auf der Hand“ liegt.

[1] Statman bewies konstruktiv in [Sta89] schon früher, daß das Wortproblem entscheidbar ist.

Teil II.

CL(S)

Kapitel 4
CL(S)

4.1. Einleitung

Wie im letzten Kapitel bereits erwähnt, ist schon das TES $\mathbf{CL}(\mathbf{S},\mathbf{K})$ Turing-vollständig und somit unentscheidbar. Dennoch gibt es aber unvollständige TES in **CL**, für die die wichtigsten Entscheidungsprobleme, nämlich Halte- und Wortproblem, entscheidbar sind. In $\mathbf{CL}(\mathbf{K})$ beispielsweise sind Termination und Wortproblem trivialerweise entscheidbar.

Gemäß Definition 2.4.13 bzw. 2.4.10 ist $\mathbf{CL}(\mathbf{K})$ löschend, und zusammen mit Satz 3.2.2 folgt, daß $\mathbf{CL}(\mathbf{K})$ terminierend ist. D. h., die Antwort auf das Halteproblem für $\mathbf{CL}(\mathbf{K})$ ist unabhängig von der Eingabe `Ja`.

Da $\mathbf{CL}(\mathbf{K})$ terminierend ist, muß eine Normalform existieren und wegen der Konfluenz ist sie sogar eindeutig. Damit lassen sich Normalformen effektiv berechnen und vergleichen. Dieses Entscheidungsverfahren ist im übrigen für alle terminierenden kombinatorischen Logiken, z. B. $\mathbf{CL}(\mathbf{I})$ oder $\mathbf{CL}(\mathbf{J})$, anwendbar.

Im weiteren werde ich die „andere Hälfte“ von $\mathbf{CL}(\mathbf{S},\mathbf{K})$, nämlich $\mathbf{CL}(\mathbf{S})$, untersuchen. Auch, wenn man zu der Vermutung gelangen könnte, daß $\mathbf{CL}(\mathbf{S})$ für die Komplexität in $\mathbf{CL}(\mathbf{S},\mathbf{K})$ verantwortlich ist, da $\mathbf{CL}(\mathbf{K})$ vergleichsweise überschaubar ist, erscheint es dennoch nicht völlig unmöglich, bestimmte Fragen für $\mathbf{CL}(\mathbf{S})$ entscheiden zu können. Insbesondere Waldmann hat in [Wal97] einiges über $\mathbf{CL}(\mathbf{S})$ zusammengetragen.

In diesem Kapitel werde ich – neben allgemeineren Betrachtungen über $\mathbf{CL}(\mathbf{S})$ – Muster einführen, mit deren Hilfe das Verhalten aller bisher untersuchten Terme aus $\mathbf{CL}(\mathbf{S})$ beschreibbar ist.

4.2. Notation

Um Schreiben (und hoffentlich auch Lesen) zu erleichtern, möchte ich kurz einige Konventionen fixieren:

Kombinatoren werden als Majuskeln und fett gesetzt: ($\mathbf{S}$,$\mathbf{K}$, ...)

Variablen, z. B. aus Ersetzungsregeln, werden klein und kursiv gesetzt: (x,y,z, ...)

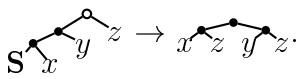
Metavariablen, die für beliebige Terme stehen können, werden groß und kursiv gesetzt: (X,Y,Z, ...)

Abkürzungen für Grundterme werden groß und ohne Serifen gesetzt: (A,T, ...)

Definition 4.2.1 Es seien $\mathsf{T} := \mathbf{SS}$ und $\mathsf{A} := \mathbf{SSS}$.

Bemerkung 4.2.2 Die Abkürzung A geht auf Barendregt zurück, der 1975 eine Wette mit Klop, Bergstra und Volken abgeschlossen hatte, wer als erster einen Term aus **CL**(**S**) ohne Normalform fände. Barendregt gewann mit AAA.

Da die Applikation $\cdot$ in **CL** als einzige nicht-konstante Funktion zweistellig ist, bietet sich zur Repräsentation eines Terms ein binärer Baum an.

Beispiel 4.2.3 $\mathbf{S}\,xyz \to xz(yz)$ ist gleichbedeutend mit 

.

Ist die Applikation statt eines ausgefüllten durch einen leeren Kreis repräsentiert, so handelt es sich bei dem entsprechenden Term um einen Redex.

4.3. CL(S)

Satz 4.3.1 Für alle $t \in \mathbf{CL}(\mathbf{S})$ gilt: t normalisiert gdw. t terminiert.

Beweis: Gemäß Satz 3.2.2 ist **CL**(**S**) konfluent und nach Definition 2.4.13 nicht löschend, woraus die Behauptung folgt (vgl. auch Satz 3.2.12). □

Innerhalb von **CL**(**S**) kann man also jede beliebige Reduktionsstrategie verwenden. Besitzt ein Term eine Normalform, so ist sie eindeutig und wird erreicht.

Definition 4.3.2 Die Menge aller Terme aus **CL**(**S**) mit Normalform sei $\mathfrak{N}$. Das Komplement, die Menge der Terme ohne Normalform, sei $\mathfrak{U}$.

Bemerkung 4.3.3 Es gilt $\mathfrak{N} \cap \mathfrak{U} = \emptyset$ und $\mathfrak{N} \cup \mathfrak{U} = \mathcal{T}(\mathcal{S}, \mathcal{V})$.

Diese Unterscheidung der Terme ist sinnvoll, da **CL**(**S**) zwar nicht terminierend, das Halteproblem aber dank der Resultate von Waldmann in [Wal98] entscheidbar ist (s. auch Tabelle 4.2).

Da **CL**(**S**) zudem top-terminierend (s. zum wiederholten Male [Wal97]) ist, „wandern“ die Redexe immer tiefer von der Wurzel weg. Es gibt also für jede nicht-terminierende Reduktionskette und eine Tiefe d eine Zahl $n \in \mathbb{N}$, so daß alle Terme der Reduktionskette, die nach n oder mehr Schritten erreicht werden, bis zur Tiefe d gleich sind.

i	0	1	2	3	4	5	6	7	8	9	10	11
length()	-	1	1	2	5	14	42	132	429	1430	4862	16796
depth()	1	1	3	21	651	457653	$2,1 \cdot 10^{11}$	...	...	...	...	...

Tabelle 4.1.: Anzahl aller Grundterme aus $\mathbf{CL(S)}$ für die jeweilige Länge bzw. Tiefe

i	1	2	3	4	5	6	7	8	9	10	11
$\lvert\{t \in \mathfrak{U} \mid \text{length}(t) = i\}\rvert$	0	0	0	0	0	0	2	41	276	1481	6829

Tabelle 4.2.: Grundterme aus $\mathbf{CL(S)}$ ohne endliche Normalform

In $\mathbf{CL(S)}$ ist die Anzahl der Grundterme ohne Normalform unbeschränkt, aber wenigstens abzählbar, da auch die Grundterme als solche abzählbar sind. Als Abzählbarkeitskriterien bieten sich die Funktionen length() oder depth() an (vgl. Tabelle 4.1).

Im weiteren werde ich die effektive Konstruktion einer Menge von Termen aus $\mathbf{CL(S)}$ betrachten, die genau alle Terme einer gegebenen Länge oder Tiefe enthält. Dies ist zum einen interessant, um die Terme aus $\mathbf{CL(S)}$ tatsächlich aufzählen zu können; zum anderen ist es für Abschätzungen des Wachstums der Kardinalität dieser Mengen wichtig.

Länge: Die folgenden Überlegungen finden sich in ähnlicher Form ebenfalls bei [Zac78].

Definition 4.3.4 Es sei

$$\forall n > 0 : \qquad L_n \;:=\; \{t \in \mathbf{CL(S)} \mid \text{length}(t) = n\}$$

die Menge aller Terme einer festen Länge n aus $\mathbf{CL(S)}$.

Satz 4.3.5 Die Mengen L_n lassen sich wie folgt konstruieren:

$$\begin{aligned} L_1 &= \{\mathbf{S}\} \\ L_n &= \bigcup_{i \in \{1,\ldots,n-1\}} \{l \cdot r \mid l \in L_i, r \in L_{n-i}\} \end{aligned}$$

Beweis: Per Induktion. □

Definition 4.3.6 Die Mächtigkeit einer Menge aller Terme einer festen Länge n aus $\mathbf{CL(S)}$ sei: $l_n := |L_n|$.

In Anlehnung an Satz 4.3.5 ergibt sich jedoch auch eine von der direkten Mengenkonstruktion unabhängige Berechnungsvorschrift:

Satz 4.3.7

$$\begin{aligned} l_1 &= 1 \\ l_n &= \sum_{i=1}^{n-1} l_i \cdot l_{n-i} \end{aligned}$$

Beweis: Folgt direkt aus Definition 4.3.6 und Satz 4.3.5. □

Für die Konstruktion einer Menge L_n wird sich der rekursive Ansatz nicht umgehen lassen. Geht es aber nur darum, die Mächtigkeit einer bestimmten Menge L_n zu bestimmen, ist dies mit einer geschlossenen Formel für l_n direkt möglich, da die l_n die Catalanzahlen sind:

$$l_n = \frac{1}{2n-1}\binom{2n-1}{n} \tag{4.1}$$

Eine eingehendere Betrachtung von (4.1) findet sich in [CLR91] wie auch diese Wachstumsabschätzung:

$$l_n = \Omega\left(\frac{4^n}{n^{3/2}}\right) \tag{4.2}$$

Tiefe: In Anlehnung an die Definitionen 4.3.4 und 4.3.6 bzw. die Sätze 4.3.5 und 4.3.7 gehe ich analog bezüglich der Tiefe vor:

Definition 4.3.8 Es sei

$$\forall n \geq 0: \qquad T_n := \{t \in \mathbf{CL}(\mathbf{S}) \mid \mathrm{depth}(t) = n\}$$

die Menge aller Terme einer festen Tiefe n aus $\mathbf{CL}(\mathbf{S})$.

Satz 4.3.9

$$\begin{aligned} T_0 &= \{\mathbf{S}\} \\ T_n^l &= \bigcup_{i \in \{0,\ldots,n-1\}} \{l \cdot r \mid l \in T_{n-1}, r \in T_i\} \\ T_n^r &= \bigcup_{i \in \{0,\ldots,n-2\}} \{l \cdot r \mid r \in T_{n-1}, l \in T_i\} \\ T_n &= T_n^l \cup T_n^r \end{aligned}$$

Beweis: Per Induktion. □

Definition 4.3.10 Die Mächtigkeit einer Menge aller Terme einer festen Tiefe n aus $\mathbf{CL}(\mathbf{S})$ sei: $t_n := |T_n|$.

Hierfür gibt es ebenfalls eine von T_n unabhängige Formel:

Satz 4.3.11

$$\begin{aligned} t_0 &= 1 \\ t_n &= t_{n-1} \cdot \left(\sum_{i=0}^{n-1} t_i \right) + t_{n-1} \cdot \left(\sum_{i=0}^{n-2} t_i \right) \end{aligned}$$

Beweis: Folgt direkt aus Definition 4.3.10 und Satz 4.3.9. □

4.4. Muster

Definition 4.4.1 (Induktives Muster) Sei $R \in \mathbf{CL}$ ein kombinatorisches TES mit Termmenge $\mathcal{T}(\mathcal{S}, \mathcal{V})$. Ein *induktives Muster* $\mathsf{M} := (\mathsf{K}, \mathsf{B})$ besteht aus einer *Basis* B und einem *Körper* K:

$$\begin{aligned} \mathsf{B} &\in \mathcal{T}(\mathcal{S}, \mathcal{V}) \\ \mathsf{K} &\in \mathcal{T}(\mathcal{S}', \mathcal{V}) \qquad \text{mit } \mathcal{S}' := (\mathrm{a}, \Sigma \cup \{\mathsf{Prev}\}),\ \mathrm{a}(\mathsf{Prev}) := 0 \end{aligned}$$

B ist also ein Grundterm und K ein Kontext, wobei Prev lediglich eine angemessenere Schreibweise des Kontextoperators [] darstellt. Es darf nämlich kein beliebiger Term in den Körper eingesetzt werden, sondern nur K oder B.

Sind sowohl K und B Grundterme, bezeichne ich M als *(induktives) Grundmuster*.

Ist in Zukunft von einem „Muster“ die Rede, so ist stets ein induktives Muster gemeint.

Definition 4.4.2 Der Begriff der Substitution gemäß Definition 2.3.1 wird für Muster dahingehend erweitert, daß die Substitution getrennt auf Körper und Basis anzuwenden ist.

Definition 4.4.3 Ein Muster $\mathsf{M} := (\mathsf{K}, \mathsf{B})$ ist in Normalform gdw. K und B jeweils in Normalform sind. Prev wird hierbei als normale Variable behandelt.

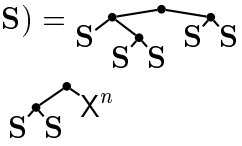
Definition 4.4.4 (Instanz) Die *Instanz* eines Musters $\mathsf{M} := (\mathsf{K}, \mathsf{B})$ wird durch einen nicht-negativen Exponenten $n \in \mathbb{N}$ beschrieben. Induktiv wird Instanz wie folgt definiert:

$$\begin{aligned} \mathsf{M}^0 &:= \mathsf{B} \\ \mathsf{M}^{n+1} &:= \mathsf{K}[\mathsf{M}^n] \end{aligned}$$

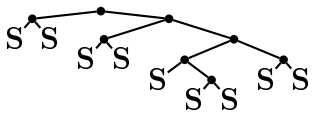
Wobei natürlich für alle n $\mathsf{M}^n \in \mathcal{T}(\mathcal{S}, \mathcal{V})$ gilt.

Beispiel 4.4.5 Sei $\mathsf{X} := (\mathbf{SS}\mathsf{Prev}, \mathbf{S}\mathsf{TT})$ ein Muster in $\mathbf{CL}(\mathbf{S})$, dann gilt für die Instanzen:

$$\mathsf{X}^0 = \mathbf{S}(\mathbf{SS})(\mathbf{SS}) = \quad \mathsf{X}^{n+1} = \mathbf{SS}\,\mathsf{X}^n =$$



Die 2. Instanz von X ist: $\mathsf{X}^2 = \mathbf{SS}(\mathbf{SS}(\mathbf{S}(\mathbf{SS})(\mathbf{SS}))) =$

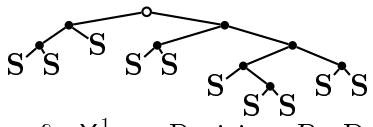


Die Instanz eines induktiven Grundmusters ist also ein Grundterm.

Die Idee der Muster ist schon vergleichsweise alt. Beispielsweise findet sich in [Klo80] eine Definition, die obiger sehr nahe kommt. In allen Beweisen von Zachos in [Zac78] finden sich ebenfalls Muster, auch wenn das Konzept nur implizit benutzt wurde.

Definition 4.4.6 Ein Muster M *paßt*[1] an der Position p in einen Term t gdw. es einen Exponenten $k \in \mathbb{N}$ und eine Substitution σ gibt, so daß $M^k\sigma = t[p]$.

Beispiel 4.4.7 Sei $t :=$



ein Term. Sei ferner X das Muster aus Beispiel 4.4.5, dann paßt X^1 an Position R. Damit ist implizit ebenfalls klar, daß X^0 an RR paßt.

Satz 4.4.8 Für alle Muster $\mathsf{M} := (\mathsf{K}, \mathsf{B})$ ist die Funktion $\mathrm{depth}(\mathsf{M}^k)$ linear in k:

$$\mathrm{depth}(\mathsf{M}^k) = \Theta(k)$$

Beweis: Dies folgt direkt aus der Musterdefinition 4.4.1, da K eine feste Tiefe hat. □

Eine Erläuterung der Θ-Definition findet sich z.B. in [Gru97].

[1] Das englische Verb lautet „to match“. Jedoch kann es sowohl im Sinne der Definition verwendet werden wie auch, um den Vorgang des Suchens nach einem passenden Muster zu beschreiben.

Kapitel 5

Unendliche Reduktionsketten

5.1. Einleitung

Daß etwas eine „Normalform“ hat, bedeutet noch nicht zwingend, daß diese endlich darstellbar sein muß. Beispielsweise hat jede rationale Zahl eine Normalform. Allerdings lassen sich Dezimalbrüche nur für den nicht-periodischen Fall als endliche Dezimalzahl aufschreiben. Um die Normalform periodischer Dezimalbrüche endlich notieren zu können, muß ein Hilfsmittel eingeführt werden, wie z.B. bei $0,\overline{142857} = 0,142857142857\ldots$

Die Situation für $\mathbf{CL}(\mathbf{S})$ ist eine ähnliche: Jeder Term hat eine Normalform, da $\mathbf{CL}(\mathbf{S})$ topterminierend ist. Näherungsweise kann man sich darunter den Grenzwert einer unendlichen Reduktionskette vorstellen. Nähere Betrachtungen sprengten jedoch den Rahmen dieser Arbeit.

Leider läßt sich die Normalform eines Terms in der Regel aber nicht endlich im Rahmen von $\mathbf{CL}(\mathbf{S})$ darstellen. Trotzdem besteht Interesse daran, die unendliche Normalform solcher Terme endlich repräsentieren zu können, etwa durch eine Grammatik o.ä. Gelänge dies, rückte die Entscheidbarkeit des Wortproblems für $\mathbf{CL}(\mathbf{S})$ vermutlich in greifbare Nähe, da mit einer endlichen Notation in diesem Fall Vergleiche effektiv durchgeführt werden könnten.

Voraussetzung für unendliche Normalformen sind unendliche Reduktionsketten, mit denen ich mich im folgenden näher befassen werde.

Die diesbezügliche Pionierarbeit hat Zachos durch die manuelle Untersuchung etlicher nicht-normalisierender Terme geleistet (s. [Zac78]).

5.2. Unendliche Reduktionsketten

Im Zusammenhang mit nicht-terminierenden TES ist es interessant zu wissen, warum Reduktionspfade unendlich lang werden können. Der einfachste Grund wären Zyklen oder Schleifen, die dafür sorgen, daß bestimmte Terme immer wieder auftauchen.

Definition 5.2.1 Ein *Zyklus* ist eine Reduktionskette $X \to^+ X$.

Definition 5.2.2 Seien K ein Kontext, σ eine Substitution und X ein Term, der Variablen enthalten darf. Eine *Schleife* ist eine Reduktionskette $X \to^+ K[X\sigma]$.

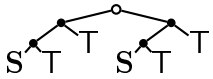
Ist X ein Grundterm, kann σ aus der Definition entfallen, und man spricht von einer *Grundschleife*.

Es gibt aber auch nicht-terminierende Ersetzungssysteme, die schleifenfrei sind, wie sie in der Arbeit von Geser und Zantema untersucht werden. Das Wortersetzungssystem $R := \{bc \to dc, bd \to db, ad \to abb\}$ führt z.B. zu der unendlichen, schleifenfreien Reduktionskette $abc \to^+ ab^2c \to^+ ab^3c \to^+ \ldots$[1] Eine ausführliche Betrachtung findet sich in [ZG96].

Für $\mathbf{CL}(\mathbf{S})$ ist bekannt, daß es sowohl zyklen- wie auch grundschleifenfrei ist (s. die Arbeiten von Bergstra und Klop [BK79] sowie Waldmann [Wal97]). Es wird allerdings lediglich vermutet, daß $\mathbf{CL}(\mathbf{S})$ sogar schleifenfrei ist. Dies folgte im übrigen auch direkt aus meiner Vermutung zur Sublinearität (vgl. Vermutung 5.7.7).

Exemplarisch untersuche ich einige Terme aus $\mathbf{CL}(\mathbf{S})$, die schleifenfrei sind, aber trotzdem eine unendliche Reduktionskette besitzen. Die verwendete Beweismethode orientiert sich stark an [Zac78] und [Wal97], wo sich bereits einige der folgenden Beweise finden.

5.2.1. Schema A

Satz 5.2.3 $t := \mathbf{S}(\mathbf{SS})(\mathbf{SS})(\mathbf{S}(\mathbf{SS})(\mathbf{SS}))$ =  hat keine Normalform.

Beweis: Sei $\mathsf{X} := (\mathbf{SS}\mathsf{Prev}, \mathbf{S}(\mathbf{SS})(\mathbf{SS}))$ ein Muster und $c \in \mathbb{N}$. Zusätzlich benötige ich folgende Hilfsaussagen:

$$t \rightsquigarrow^c \mathsf{X}^0\mathsf{X}^0 \tag{5.1}$$

$$\forall Z, \forall i > 0 \quad : \quad \mathsf{X}^i Z \rightsquigarrow^c \mathsf{X}^{i-1} Z \tag{5.2}$$

$$\forall i \geq 0 \quad : \quad \mathsf{X}^0\mathsf{X}^i \rightsquigarrow^c \mathsf{X}^{i+1}\mathsf{X}^{i+1} \tag{5.3}$$

Die erste Hilfsaussage ist trivialerweise wahr, da $t = \mathsf{X}^0\mathsf{X}^0$. Um die zweite zu zeigen, genügt ein Reduktionsschritt: $\mathbf{SS}\mathsf{X}^{i-1}Z \to \mathbf{S}Z(\mathsf{X}^{i-1}Z)$. Für die dritte reicht ebenfalls ein Schritt: $\mathbf{STT}\mathsf{X}^i \to \mathsf{T}\mathsf{X}^i(\mathsf{T}\mathsf{X}^i) = \mathsf{X}^{i+1}\mathsf{X}^{i+1}$.

Wiederholte Anwendung von (5.2) sorgt dafür, daß (5.3) angewendet werden kann. Nach der Anwendung von (5.3) läßt sich wiederum mit (5.2) beginnen. (5.1) schließlich stellt sicher, daß die unendliche Reduktionskette überhaupt beginnen kann. □

[1] Interessanterweise gibt es eine Ähnlichkeit zwischen der Art und Weise, wie hier die höheren Potenzen von b erreicht werden, und den immer höheren Musterinstanzen in den von mir untersuchten Reduktionsketten.

Ein erster Versuch für ein formales Beweisschema, mit Hilfe dessen die Existenz einer unendlichen Reduktionskette zu einem Startterm aus $\mathfrak{U}$ bewiesen werden kann, könnte also so aussehen:

Definition 5.2.4 (Beweisschema A) Ein Beweis nach *Schema A* für eine unendliche Reduktionskette in einem Term $t \in \mathfrak{U}$ ist ein Tupel (M, c), wobei M ein Grundmuster und c eine natürliche Zahl ist, so daß gilt:

$$\exists i, j \geq 0 \quad : \quad t \leadsto^c \mathsf{M}^i\mathsf{M}^j \tag{5.4}$$

$$\forall Z, \forall i > 0 \quad : \quad \mathsf{M}^i Z \leadsto^c \mathsf{M}^{i-1} Z \tag{5.5}$$

$$\forall i \geq 0 \quad : \quad \mathsf{M}^0\mathsf{M}^i \leadsto^c \mathsf{M}^i\mathsf{M}^{i+1} \tag{5.6}$$

Leider ist $\mathbf{CL}(\mathbf{S})$ nicht so „leicht“, daß Schema A bereits ausreicht. Dies zeigt schon das nächste Beispiel.

5.2.2. Schema B

Satz 5.2.5 $t := \mathsf{ATSS} = \mathbf{SSS}(\mathbf{SS})\mathbf{SS}$ hat keine Normalform.

Beweis: Seien

$$\begin{aligned} \mathsf{D} &:= \mathbf{S}(\mathbf{S}(\mathbf{S}\mathsf{T}(\mathbf{S}(\mathbf{S}\mathsf{T}\mathbf{S})\mathbf{S})))) \\ \mathsf{M}_L^n &:= \mathbf{S}\mathsf{Prev}(\mathsf{D}(\mathbf{S}\mathsf{Prev})) \\ \mathsf{M}_L^0 &:= \mathsf{DD} \end{aligned}$$

wobei D ein Grundterm und $\mathsf{M}_L := (\mathsf{M}_L^n, \mathsf{M}_L^0)$ ein induktives Grundmuster sind. Die Reduktionsstrategie sei weiterhin $\mathrm{rs}_{\mathrm{LA}}()$ sowie $c \in \mathbb{N}$.

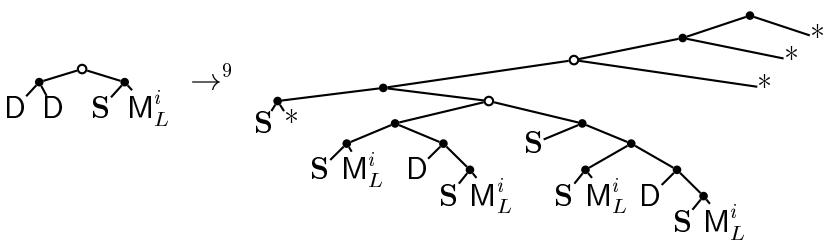
Die drei benötigten Hilfsaussagen sind von der Form:

$$t \leadsto^c \mathsf{M}_L^0(\mathbf{S}\mathsf{M}_L^0) \tag{5.7}$$

$$\forall Z, \forall i > 0 \quad : \quad \mathsf{M}_L^i Z \leadsto^c \mathsf{M}_L^{i-1} Z \tag{5.8}$$

$$\forall i \geq 0 \quad : \quad \mathsf{M}_L^0(\mathbf{S}\mathsf{M}_L^i) \leadsto^c \mathsf{M}_L^{i+1}(\mathbf{S}\mathsf{M}_L^{i+1}) \tag{5.9}$$

Um 5.7 zu zeigen, benötige ich 75 Reduktionen, also $t \to^{75} t'$ und $\mathsf{M}_L^0(\mathbf{S}\mathsf{M}_L^0) \trianglelefteq t'$ an Position $RLLLLLLLLLLLLLLLLLLLLLLLLR$. Aussage 5.8 benötigt wieder nur einen Schritt, und 5.9 schließlich läßt sich nach neun Schritten zeigen:



$*$ steht hier jeweils für einen beliebigen Term, der nicht weiter von Interesse ist. □

Offenbar muß die rechte Instanz des Musters nicht alleinstehend sein, was also zum nächsten Schema führt:

Definition 5.2.6 (Beweisschema B) Ein Beweis nach *Schema B* für eine unendliche Reduktionskette in einem Term $t \in \mathfrak{U}$ ist ein Tupel (M, c, K), wobei M ein Grundmuster, c eine natürliche Zahl und K ein Kontext ist, so daß gilt:

$$\exists i, j \geq 0 \quad : \quad t \leadsto^c \mathsf{M}^i K[\mathsf{M}^j] \tag{5.10}$$

$$\forall Z, \forall i > 0 \quad : \quad \mathsf{M}^i Z \leadsto^c \mathsf{M}^{i-1} Z \tag{5.11}$$

$$\forall i \geq 0 \quad : \quad \mathsf{M}^0 K[\mathsf{M}^i] \leadsto^c \mathsf{M}^i K[\mathsf{M}^{i+1}] \tag{5.12}$$

Doch auch dieser Ansatz ist immer noch nicht allgemein genug.

5.2.3. Schema C

Die unendliche Reduktionskette, die sich für $t := \mathsf{ATSS}$ durch den Beweis von Satz 5.2.5 ergibt, bezeichne ich im weiteren mit k.

k ist *nicht* die Reduktionskette, die entsteht, wenn t unendlich oft unter $\mathrm{rs}_{\mathrm{LA}}()$ reduziert wird. Zwar wurden die Aussagen (5.7)–(5.9) tatsächlich alle unter Verwendung der Strategie $\mathrm{rs}_{\mathrm{LA}}()$ nachgewiesen, der entscheidende Unterschied ist aber, daß ich – bzw. mein Programm `CLS` – mich für (5.9) damit begnügt habe zu zeigen, daß $\mathsf{M}_L^{i+1}(\mathsf{SM}_L^{i+1})$ erreicht wird, und zwar nach neun Schritten an Position $LLLR$. Reduziert man an dieser Position weiter, so erhält man k.

Allerdings befindet sich der laut $\mathrm{rs}_{\mathrm{LA}}()$ vorgeschriebene Redex an Position LL. Die Reduktion dort fortzuführen, resultiert in einer weiteren unendlichen Reduktionskette k', die sich ebenfalls durch das Grundmuster M_L beschreiben läßt, die sich aber signifikant von k unterscheidet.

Satz 5.2.7 $t := \mathsf{ATSS}$ hat eine weitere unendliche Reduktionskette k' mit dem Muster M_L.

Beweis: Seien wieder

$$\begin{aligned} \mathsf{D} &:= \mathsf{S}(\mathsf{S}(\mathsf{S}(\mathsf{ST}(\mathsf{S}(\mathsf{STS})\mathsf{S})))) \\ \mathsf{M}_L^n &:= \mathsf{SPrev}(\mathsf{D}(\mathsf{SPrev})) \\ \mathsf{M}_L^0 &:= \mathsf{DD} \end{aligned}$$

Grundterm und induktives Grundmuster. Die Reduktionsstrategie sei ebenfalls $\mathrm{rs}_{\mathrm{LA}}()$ sowie $c \in \mathbb{N}$.

Die benötigten Hilfsaussagen sind von der Form:

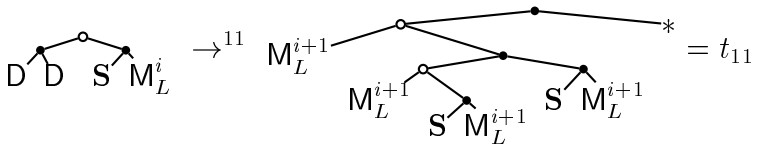
$$t \rightsquigarrow^c \mathsf{M}_L^0(\mathbf{S}\mathsf{M}_L^0) \tag{5.13}$$

$$\forall Z, \forall i > 0 \;:\; \mathsf{M}_L^i Z \rightsquigarrow^c \mathsf{M}_L^{i-1} Z \tag{5.14}$$

$$\forall i \geq 0 \;:\; \mathsf{M}_L^0(\mathbf{S}\mathsf{M}_L^i) \rightsquigarrow^c \mathsf{M}_L^{i+1}(\mathsf{M}_L^{i+1}(\mathbf{S}\mathsf{M}_L^{i+1})(\mathbf{S}\mathsf{M}_L^{i+1})) \tag{5.15}$$
$$\mathsf{M}_L^0(\mathsf{M}_L^{i+1}(\mathbf{S}\mathsf{M}_L^{i+1})(\mathbf{S}\mathsf{M}_L^{i+1})) \rightsquigarrow^c \mathsf{M}_L^{i+1}(\mathbf{S}\mathsf{M}_L^{i+1})$$

Die Aussagen (5.13) und (5.14) sind identisch mit (5.7) bzw. (5.8).

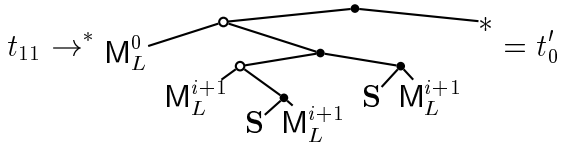
Den ersten Teil von (5.15) erhält man, indem man zunächst an den Beweis von (5.9) noch zwei weitere Reduktionsschritte anhängt:



(5.16)

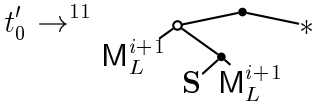
Wobei $*$ wieder bedeutet, daß der entsprechende Subterm für weitere Betrachtungen unerheblich ist.

Dank (5.14) gilt



(5.17)

woraus mit weiteren elf Schritten



(5.18)

folgt. □

Diese Überlegungen beschreiben die unendliche Reduktionskette k', die genau der Reduktionskette entspricht, die entsteht, wenn ATSS unter $\mathrm{rs}_{\mathrm{LA}}()$ reduziert wird.

Aus den Sätzen 5.2.5 und 5.2.7 folgt unmittelbar:

Satz 5.2.8 Ein induktives Grundmuster kann mehrere unendliche Reduktionsketten in Termen aus $\mathfrak{U}$ beschreiben.

Um die aus diesem Beweis gewonnenen Erkenntnisse wieder zu verallgemeinern, definiere ich:

Definition 5.2.9 (Beweisschema C) Ein Beweis nach *Schema C* für eine unendliche Reduktionskette in einem Term $t \in \mathfrak{U}$ ist ein Tupel $(\mathsf{M}, c, (K_1, \ldots, K_n))$, wobei M ein

Grundmuster, c eine natürliche Zahl, K_1 ein *Hauptkontext* und $(K_2, \ldots, K_n)$ eine Folge von *Nebenkontexten* ist, so daß gilt:

$$\exists i, j \geq 0 \quad : \quad t \leadsto^c \mathsf{M}^i K_1[\mathsf{M}^j] \tag{5.19}$$

$$\forall Z, \forall i > 0 \quad : \quad \mathsf{M}^i Z \leadsto^c \mathsf{M}^{i-1} Z \tag{5.20}$$

$$\begin{aligned} \forall j \geq 0 \quad : \quad & 1 \leq i < n : \mathsf{M}^0 K_i[\mathsf{M}^j] \leadsto^c \mathsf{M}^j K_{i+1}[\mathsf{M}^j] \\ & \mathsf{M}^0 K_n[\mathsf{M}^j] \leadsto^c \mathsf{M}^j K_1[\mathsf{M}^{j+1}] \end{aligned} \tag{5.21}$$

Bemerkung 5.2.10 Andere Hauptkontexte als den trivialen Kontext [] oder $\mathbf{S}[]$ konnte ich bis jetzt nicht beobachten. Ist $n = 1$, bedeutet dies, daß Teil 1 von (5.21) überhaupt nicht zum Tragen kommt und der Beweis Schema B entspricht. Jedoch scheint es Terme zu geben, die ausschließlich unendliche Reduktionsketten mit $n > 1$ besitzen. In einem solchen Fall wird (5.20) benötigt, um die Brücke zwischen allen Teilen von (5.21) zu schlagen.

Definition 5.2.11 (Nulldurchgang) Jedes Auftreten von M^0 im linken Teil eines beteiligten Redexes, das nicht im Startterm von (5.21) enthalten ist, nenne ich *Nulldurchgang* von M.

5.2.4. Schema D

Schema C beinhaltet Schema A wie B und erlaubt bereits, einen Großteil der Beweise für unendliche Reduktionsketten in Termen aus $\mathfrak{U}$ zu führen. Dennoch gibt es eine weitere Klasse von Beweisen, die nicht mit den bisherigen Methoden funktionieren.

Satz 5.2.12 $t := \mathsf{ATST}$ hat keine Normalform.

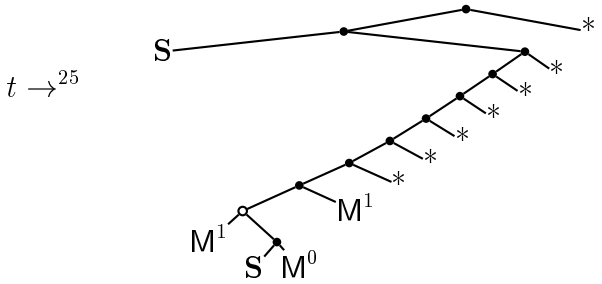
Beweis: Sei $D := \mathbf{S}(\mathbf{STS})\mathsf{T}$ ein Grundterm, $\mathsf{M} := (\mathsf{TPrev}, \mathbf{S}(\mathsf{T}(\mathbf{ST})\mathsf{D}))$ ein induktives Muster und $c \in \mathbb{N}$. Ferner gilt:

$$t \leadsto^c \mathsf{M}^1(S\mathsf{M}^0)\mathsf{M}^1 \tag{5.22}$$

$$\forall Z_{\{1,2\}}, \forall i > 0 \quad : \quad \mathsf{M}^{i+1} Z_1 Z_2 \leadsto^c \mathsf{M}^i Z_1 Z_2 \tag{5.23}$$

$$\forall i \geq 0 \quad : \quad \mathsf{M}^0(S\mathsf{M}^i)\mathsf{M}^{i+1} \leadsto^c \mathsf{M}^{i+2}(S\mathsf{M}^{i+1})\mathsf{M}^{i+2} \tag{5.24}$$

Gleichung (5.22) zu zeigen, benötigt 25 Schritte, die ich nicht alle aufführen möchte:



Auch hier steht $*$ für einen beliebigen Term, der nicht weiter von Interesse ist.

Für (5.23) reichen 2 Schritte, und (5.24) läßt sich nach 5 Schritten zeigen:

$$
\begin{aligned}
& \mathsf{M}^0(\mathbf{S}\mathsf{M}^i)\mathsf{M}^{i+1} \\
\to\ & \mathsf{T}(\mathbf{S}\mathsf{T})\mathsf{D}\mathsf{M}^{i+1}(\mathbf{S}\mathsf{M}^i\mathsf{M}^{i+1}) \\
\to\ & \mathbf{S}\mathsf{D}(\mathbf{S}\mathsf{T}\mathsf{D})\mathsf{M}^{i+1}(\mathbf{S}\mathsf{M}^i\mathsf{M}^{i+1}) \\
\to\ & \mathsf{D}\mathsf{M}^{i+1}(\mathbf{S}\mathsf{T}\mathsf{D}\mathsf{M}^{i+1})(\mathbf{S}\mathsf{M}^i\mathsf{M}^{i+1}) \\
\to\ & \mathbf{S}\mathsf{T}\mathbf{S}\mathsf{M}^{i+1}\mathsf{M}^{i+2}(\mathbf{S}\mathsf{T}\mathsf{D}\mathsf{M}^{i+1})(\mathbf{S}\mathsf{M}^i\mathsf{M}^{i+1}) \\
\to\ & \mathsf{M}^{i+2}(\mathbf{S}\mathsf{M}^{i+1})\mathsf{M}^{i+2}(\mathbf{S}\mathsf{T}\mathsf{D}\mathsf{M}^{i+1})(\mathbf{S}\mathsf{M}^i\mathsf{M}^{i+1}) \qquad \square
\end{aligned}
$$

Aus diesem Beweis ergibt sich die Definition für das letzte Schema:

Definition 5.2.13 (Beweisschema D) Ein Beweis nach *Schema D* für eine unendliche Reduktionskette in einem Term $t \in \mathfrak{U}$ ist ein Tupel $(\mathsf{M}, c, (K_1, \ldots, K_n))$, wobei M ein Grundmuster, c eine natürliche Zahl und $(K_1, \ldots, K_n)$ eine Folge von Kontexten ist, so daß gilt:

$$\exists i, j \geq 0 \quad : \quad t \leadsto^c \mathsf{M}^i K_1[\mathsf{M}^j] \ldots K_n[\mathsf{M}^j] \tag{5.25}$$

$$\forall Z_l, \forall i > 0 \quad : \quad \mathsf{M}^i Z_1 \ldots Z_n \leadsto^c \mathsf{M}^{i-1} Z_1 \ldots Z_n \tag{5.26}$$

$$\forall i \geq 0 \quad : \quad \mathsf{M}^0 K_1[\mathsf{M}^i] \ldots K_n[\mathsf{M}^i] \leadsto^c \mathsf{M}^i K_1[\mathsf{M}^{i+1}] \ldots K_n[\mathsf{M}^{i+1}] \tag{5.27}$$

Alle bisherigen Beweise haben nicht nur die Existenz einer Normalform für den jeweiligen Term widerlegt, sondern ebenfalls gezeigt, wie man die zugehörige unendliche Reduktionskette erhält.

Die Beziehungen der einzelnen Beweismuster lassen sich sehr schön graphisch darstellen (s. Abbildung 5.1). Dabei ist B eine Verallgemeinerung von A, und C sowie D sind zueinander orthogonale Verallgemeinerungen von B. Es wäre möglich, C und D ebenfalls zu kombinieren; allerdings habe ich bis jetzt noch keinen Fall beobachten können, für den dies nötig gewesen wäre, und das resultierende Schema wäre zudem recht unübersichtlich.

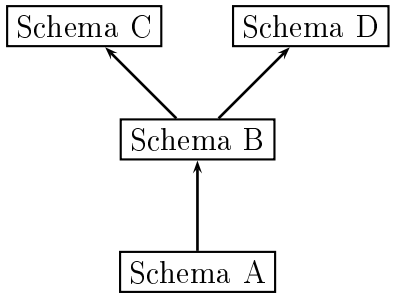


Abbildung 5.1.: „Stammbaum“ der Beweisschemata

Bis jetzt ist es nicht gelungen, eine andere Ursache für die Existenz unendlicher Reduktionsketten zu finden als induktive Grundmuster. Bei mehreren 1000 untersuchten Grundtermen ohne Normalform aus $\mathbf{CL}(\mathbf{S})$ war es in fast allen Fällen möglich zu beweisen, daß ein Grundmuster für eine unendliche Reduktionskette verantwortlich ist. Für eine detaillierte Auswertung s. Abschnitt 5.8.

Die Bearbeitung einer solchen Menge von Termen war nur durch eine automatisierte Untersuchung mittels der von mir entwickelten Software `CLS` möglich. `CLS` übernimmt u. a. das Finden der Muster und Beweise nach Schema C.

Die o. a. Ergebnisse veranlassen mich zu der Vermutung, daß tatsächlich immer induktive Grundmuster dafür verantwortlich sind, daß ein Grundterm aus $\mathbf{CL}(\mathbf{S})$ keine Normalform hat.

Vermutung 5.2.14 Für alle Terme aus $\mathfrak{U}$ gibt es mindestens einen Beweis für die Existenz einer unendlichen Reduktionskette nach einem der Schemata A-D. Dies bedeutet auch, daß zu jedem Term aus $\mathfrak{U}$ ein induktives Grundmuster existiert.

Bemerkung 5.2.15 Vermutung 5.2.14 besagt, daß für jeden Term aus $\mathfrak{U}$ *mindestens eine* unendliche Reduktionskette den angegebenen Kriterien entspricht. Daß sich die Vermutung für *jede* unendliche Reduktionskette halten läßt, erscheint schwierig, da jede beliebige Reduktionsstrategie zulässig ist.

Eine vorsichtige Verschärfung der Vermutung könnte in die Richtung gehen, daß für jede unendliche Reduktionskette $k := t_1 \to t_2 \to \ldots$ ein Muster M existiert, so daß immer höhere Instanzen von M auftreten:

$$\forall t \in \mathfrak{U} : \forall k : \exists \mathsf{M} : \forall i \in \mathbb{N} : \exists j \in \mathbb{N} : \mathsf{M}^i \trianglelefteq t_j \tag{5.28}$$

5.3. Reduktionsstrategien

Ein Term $t \in \mathfrak{U}$ kann je nach verwendeter Reduktionsstrategie mehrere unendliche Reduktionsketten haben. Dazu muß es zu einem gegebenen t_i aus einer Reduktionskette von t mehr als einen Redex geben, also $|\mathrm{RPos}(t_i)| > 1$. Verwendet man verschiedene, unfaire[2] Reduktionsstrategien bei der Analyse eines solchen Terms t, wird man in der Regel unterschiedliche induktive Muster finden.

Satz 5.3.1 Für $t := \mathsf{ATSS}$ gibt es mehr als ein induktives Grundmuster, das eine unendliche Reduktionskette beschreibt.

[2] Eine genaue Definition einer fairen Reduktionsstrategie sprengte den Rahmen dieser Arbeit. Anschaulich ist mit fair gemeint, daß kein Redex unendlich oft nicht reduziert wird.

Beweis:

- Mit der Reduktionsstrategie $\mathrm{rs}_{\mathrm{LA}}()$ findet sich gemäß Satz 5.2.5 die erste unendliche Reduktionskette. Das hierfür verantwortliche induktive Muster ist M_L.
- Das nächste Muster wurde mittels meiner Software `CLS` gefunden und verifiziert, wobei ich den automatisch generierten Beweis etwas lesbarer gemacht habe.

  Seien $\mathsf{D} := \mathbf{S}(\mathbf{S}(\mathbf{S}(\mathbf{ST}(\mathbf{S}(\mathbf{STS})\mathbf{S}))))$ und $\mathsf{M}_R := (\mathbf{S}(\mathsf{DD})\mathsf{Prev}, \mathsf{D}(\mathbf{S}(\mathsf{DD})))$ ein Grundterm bzw. ein induktives Muster und $c \in \mathbb{N}$. Wieder benötige ich drei Hilfsaussagen:

$$t \rightsquigarrow^c \mathsf{M}_R^0\mathsf{M}_R^1 \tag{5.29}$$

$$\forall Z, \forall i > 0 \quad : \quad \mathsf{M}_R^i Z \rightsquigarrow^c \mathsf{M}_R^{i-1} Z \tag{5.30}$$

$$\forall i \geq 0 \quad : \quad \mathsf{M}_R^0\mathsf{M}_R^i \rightsquigarrow^c \mathsf{M}_R^i\mathsf{M}_R^{i+1} \tag{5.31}$$

  Die Herleitung von (5.29) erfordert etwas mehr Aufwand. Aus Gründen der Übersichtlichkeit verzichte ich darauf, die einzelnen Reduktionsschritte explizit aufzulisten. Die verwendete Reduktionsstrategie ist $\mathrm{rs}_{\mathrm{RI}}()$:

$$t \rightarrow^{31} t'$$

  $\mathsf{M}_R^0\mathsf{M}_R^1 \trianglelefteq t'$ und zwar an Position $RLRRRLRRLRLRR$.

  Aussage (5.30) ist wieder einfach zu zeigen:

$$\mathbf{S}(\mathsf{DD})\mathsf{M}_R^{i-1} Z \rightarrow \mathsf{DD}Z(\mathsf{M}_R^{i-1} Z)$$

  Für (5.31) ersetze ich in M_R^0 den unwichtigen Teil durch $*$:

$$\mathbf{S}(\mathbf{S}*)(\mathbf{S}(\mathsf{DD}))\mathsf{M}_R^i \rightarrow \mathbf{S} * \mathsf{M}_R^i(\mathbf{S}(\mathsf{DD})\mathsf{M}_R^i) \rightarrow *(\mathbf{S}(\mathsf{DD})\mathsf{M}_R^i)(\mathsf{M}_R^i(\mathbf{S}(\mathsf{DD})\mathsf{M}_R^i))$$

□

Bemerkung 5.3.2 $\mathsf{M}_L^1 = \mathsf{M}_R^1 = \mathbf{S}(\mathsf{DD})(\mathsf{D}(\mathbf{S}(\mathsf{DD})))$. Dann jedoch divergieren die Musterinstanzen, da sich die eine Strategie nach rechts und die andere nach links orientiert.

5.4. Anzahl aller Grundmuster

Voraussetzung für einen Beweis in der Art der Sätze 5.2.3 oder 5.2.5 ist natürlich, daß man das entsprechende Muster kennt. Beim ersten Beweis war das Muster recht leicht zu finden, aber je größer die Muster werden, um so unübersichtlicher und schwieriger wird es. Zudem gibt es unendlich viele Grundmuster, so daß der Suchraum unendlich mächtig ist.

Satz 5.4.1 Es gibt unendlich viele Grundmuster, die unendliche Reduktionsketten in Grundtermen in $\mathbf{CL}(\mathbf{S})$ verursachen.

Beweis: Seien in Anlehnung an den zweiten Teil des Beweises von Satz 5.3.1

$$\begin{aligned} & & \mathsf{D} &:= \mathbf{S}(\mathbf{S}(\mathbf{S}(\mathbf{ST}(\mathbf{S}(\mathbf{STS})\mathbf{S})))) \\ \forall X \in \mathcal{T}(\mathcal{S},\emptyset): & & \mathsf{M}_X^n &:= \mathbf{S}(\mathsf{DD})\mathsf{Prev} \\ & & \mathsf{M}_X^0 &:= \mathbf{S}(\mathbf{S}X)(\mathbf{S}(\mathsf{DD})) \\ & & t_X &:= \mathsf{M}_X^0\mathsf{M}_X^1 \end{aligned}$$

Dann gibt es für alle X einen Term t_X mit zugehörigem Muster $\mathsf{M}_X := \left(\mathsf{M}_X^n, \mathsf{M}_X^0\right)$, der eine unendliche Reduktionskette hat, was sich jeweils analog zu Satz 5.3.1 beweisen läßt.

Da die Menge der Grundterme aus $\mathbf{CL}(\mathbf{S})$ nicht beschränkt ist, trifft dies auch für die Menge der Muster zu, die unendliche Reduktionsketten von Grundtermen in $\mathbf{CL}(\mathbf{S})$ verursachen. □

Bemerkung 5.4.2 Satz 5.4.1 ließe sich auch beweisen, indem man den Subterm DD für M_X^n und M_X^0 entsprechend anpaßte.

5.5. Grundmuster und Normalformen

Da die induktiven Grundmuster für $\mathbf{CL}(\mathbf{S})$ benutzt werden, um unendliche Reduktionsketten zu beschreiben, könnte man davon ausgehen, daß Basis und Körper des jeweiligen Musters eine Normalform haben, daß also das Muster in Normalform ist. Wählt man aber beispielsweise im Beweis von Satz 5.4.1 $X \in \mathfrak{U}$, dann ist M_X^0 nicht mehr in Normalform, der entsprechende Beweis funktioniert trotzdem unmodifiziert. Einzig die Reduktionsstrategie wäre nicht mehr $\mathrm{rs}_{\mathrm{RI}}()$, da diese nämlich in X hinabstiege und so ein anderes Muster fände.

Bemerkung 5.5.1 t_X mit $X \in \mathfrak{U}$ aus Satz 5.4.1 hat sogar unendlich viele unendliche Reduktionsketten, weil für die beschriebene Reduktionskette beliebig viele X als Subterme erzeugt werden, die für die Fortführung der eigentlichen Reduktionskette bedeutungslos sind, ihrerseits aber selbst als Startterm einer unendlichen Reduktionskette fungieren.

Bei inneren Reduktionsstrategien ist es mir nicht gelungen, ein Muster ohne Normalform zu finden, das unter Beibehalten der jeweiligen Reduktionsstrategie geeignet wäre, die entstehende unendliche Reduktionskette zu beschreiben. Dies liegt daran, daß bei den Beweisen der letzten beiden Hilfsaussagen aller Beweisschemata jeweils ein äußerer Redex reduziert werden muß. Eine innere Reduktionsstrategie stiege aber stets zuerst

in die Subterme des Körpers oder der Basis des Musters, die nicht in Normalform sind, hinab.

Im Gegensatz dazu ist es bei äußeren Strategien wie z.B. $\mathrm{rs}_{\mathrm{LA}}()$ und $\mathrm{rs}_{\mathrm{RA}}()$ möglich, Muster ohne Normalform zu finden.

Beispiel 5.5.2 Es seien

$$\begin{aligned}
\forall X_{LA} \in \mathbf{CL}(\mathbf{S}): \qquad \mathsf{M}_{LA} &:= (\mathbf{S}\mathsf{Prev}(X_{LA}(\mathbf{S}\mathsf{Prev})),\ \mathsf{D}X_{LA}) \\
t_{LA} &:= \mathsf{M}^0_{LA}(\mathbf{S}\mathsf{M}^0_{LA}) \\
\forall X_{RA} \in \mathbf{CL}(\mathbf{S}): \qquad \mathsf{M}_{RA} &:= (\mathbf{S}(\mathsf{DD})\mathsf{Prev},\ \mathbf{S}(\mathbf{S}X_{RA})(\mathbf{S}(\mathsf{DD}))) \\
t_{RA} &:= \mathsf{M}^0_{RA}\mathsf{M}^1_{RA}
\end{aligned}$$

M_{LA} ist das Muster, welches unter Verwendung der Strategie $\mathrm{rs}_{\mathrm{LA}}()$ in t_{LA} die unendliche Reduktionskette k_{LA} beschreibt (vgl. Satz 5.2.7).

M_{RA} ist das Muster, das unter $\mathrm{rs}_{\mathrm{RA}}()$ die unendliche Reduktionskette k_{RA} in t_{RA} beschreibt. Deutlich wird das sofort aus dem zweiten Teil des Beweises von Satz 5.3.1, da sich in diesem Fall die beiden Reduktionsstrategien $\mathrm{rs}_{\mathrm{RA}}()$ und $\mathrm{rs}_{\mathrm{RI}}()$ identisch verhalten.

Für X_{LA} und X_{RA} lassen sich jeweils beliebige Terme einsetzen, aber unter Beibehaltung der festgelegten Reduktionsstrategien werden genau die zugehörigen unendlichen Reduktionsketten k_{LA} und k_{RA} von den Starttermen aus generiert.

Vermutung 5.5.3 Bei der Verwendung innerer Reduktionsstrategien für die Suche nach einer unendlichen Reduktionskette gemäß Vermutung 5.2.14 sind die gefundenen Grundmuster ausschließlich in Normalform.

5.6. Allgemeinste Muster

Aus dem Beweis von Satz 5.4.1 geht hervor, daß es unendlich viele unendliche Reduktionsketten in Termen aus $\mathfrak{U}$ gibt, für die jeweils ein unterschiedliches induktives Grundmuster verantwortlich ist. Offensichtlich verhält es sich aber so, daß zur Beschreibung einer unendlichen Reduktionskette in einem Term nicht unbedingt alle Teile eines Musters wichtig sind (vgl. z.B. $*$ in Satz 5.2.5), so daß sich die Frage stellt, ob es möglich ist, die relevanten Teile eines Musters zu bestimmen. Hierbei sind mit „relevante Teile“ diejenigen Subterme in Körper und Basis gemeint, die für einen Beweis nach Vermutung 5.2.14 benötigt werden.

Allerdings werde ich mich im folgenden auf Beweise nach Schema C beschränken, was vor allem durch die begrenzte Zeit, die mir zur Verfügung steht, motiviert ist.

Definition 5.6.1 (Umbenennung) Sei $\mathcal{T}(\mathcal{S},\mathcal{V})$ eine Termmenge und σ eine Substitution. σ ist eine *Umbenennung*, wenn für die zugehörige Bildmenge B gilt: $B \subseteq \mathcal{V}$, wenn also Variablen ausschließlich auf Variablen abgebildet werden.

Definition 5.6.2 Seien M_a und M_b induktive Muster und σ eine Substitution, die keine Umbenennung ist. M_a ist *allgemeiner als* M_b, geschrieben $\mathsf{M}_a \supseteq \mathsf{M}_b$, gdw. $\mathsf{M}_a\sigma = \mathsf{M}_b$.

M_a ist *echt allgemeiner als* M_b, geschrieben $\mathsf{M}_a \supset \mathsf{M}_b$, gdw. $\mathsf{M}_a\sigma = \mathsf{M}_b$ und σ nicht die Identität ist.

Bemerkung 5.6.3 Im Falle der Relation $\supseteq$ bzw. $\supset$ betrachte ich also strenggenommen Klassen von Mustern, die bis auf eine andere Benennung der Variablen identisch sind.

Satz 5.6.4 $\supseteq$ auf Mustern ist eine partielle Ordnung.

Beweis:

- Reflexivität: Folgt aus der Definition, da σ auch die Identität sein darf.
- Antisymmetrie: Es genügt zu zeigen, daß $\supset$ asymmetrisch ist. Seien also M_a und M_b zwei beliebige Muster, für die $\mathsf{M}_a \supset \mathsf{M}_b$ gilt, mit $\mathsf{M}_a\sigma = \mathsf{M}_b$. Seien ferner $\mathcal{T}(\mathcal{S},\mathcal{V})$ die Termmenge für $\mathbf{CL}(\mathbf{S})$ und $\mathcal{S}'$ die um Prev erweiterte Signatur.

  Da σ nicht die Identität ist, gilt außerdem:

$$\begin{aligned} \exists v \in \mathcal{V},\, \exists t \in \mathcal{T}(\mathcal{S}',\mathcal{V}) : \quad & v \neq t \\ \wedge \quad & \mathrm{Pos}(v, \mathsf{M}_a^0) = \mathrm{Pos}(t, \mathsf{M}_b^0) \neq \emptyset \\ \vee \quad & \mathrm{Pos}(v, \mathsf{M}_a^n) = \mathrm{Pos}(t, \mathsf{M}_b^n) \neq \emptyset \\ \wedge \quad & v\sigma = t \end{aligned}$$

  Des weiteren gibt es keine Substitution τ, so daß $t\tau = v$, da $t \notin \mathcal{V}$ gilt, weil für $\supset$ auf Mustern reine Umbenennungen unzulässig sind.

  Hieraus folgt: $\mathsf{M}_b \not\supset \mathsf{M}_a$.
- Transitivität: Seien M_a, M_b und M_c drei beliebige Muster, für die $\mathsf{M}_a \supseteq \mathsf{M}_b$ und $\mathsf{M}_b \supseteq \mathsf{M}_c$ gilt. Die zugehörigen Substitutionen seien σ und τ, mit $\mathsf{M}_a\sigma = \mathsf{M}_b$ bzw. $\mathsf{M}_b\tau = \mathsf{M}_c$.

  Damit existiert ebenfalls $\rho = \tau \circ \sigma$, also die Komposition von τ und σ, so daß $\mathsf{M}_a\rho = \mathsf{M}_c$. □

Satz 5.6.5 $\supset$ auf Mustern ist eine Striktordnung.

Beweis: Laut Definition ist $\supset$ irreflexiv. Die Transitivität folgt aus dem Beweis von Satz 5.6.4. □

Bemerkung 5.6.6 Im folgenden sind auch Muster, die Variablen enthalten, für einen Beweis nach Schema C zulässig. Aussage (5.19) muß dementsprechend um eine Substitution σ erweitert werden:

$$\exists i, j \geq 0 : \qquad t \rightsquigarrow \sigma(\mathsf{M}^i K_1[\mathsf{M}^j]) \tag{5.32}$$

An dieser Stelle sei angemerkt, daß Variablen – ohne die Zuhilfenahme einer Substitution – Reduktionsketten terminieren können. Z.B. enthält der Term $t = axyz \in \mathbf{CL}(\mathbf{S})$ keinen Redex. Wendet man aber σ mit $a\sigma = \mathbf{S}$ auf t an, so erhält man $t\sigma = \mathbf{S}xyz$, das natürlich einen Redex enthält.

Definition 5.6.7 (TBMK) Seien $t \in \mathfrak{U}$, M ein Muster, $K := (K_1, \ldots, K_n)$ ein Haupt- sowie $n-1$ Nebenkontexte und $B := \{B_1, B_2, B_3\}$ ein Beweis im Sinne von Definition 5.2.9 – modifiziert durch Bemerkung 5.6.6. Die B_i entsprechen den Hilfsaussagen (5.32), (5.20) und (5.21). Darüberhinaus sei k die unendliche Reduktionskette in t, die durch M, K und B beschrieben wird.

Das Tupel (t, B, M, K) nenne ich *TBMK bezüglich* k.

Definition 5.6.8 Seien $x := (t_x, B_x, \mathsf{M}_x, K_x)$ und $y := (t_y, B_y, \mathsf{M}_y, K_y)$ zwei TBMK bezüglich der Reduktionskette k. x ist *allgemeiner als* y, geschrieben $x \supseteq y$, gdw.

- $t_x = t_y$, $K_x = K_y$
- $\mathsf{M}_x \supseteq \mathsf{M}_y$ mit $\mathsf{M}_x\tau = \mathsf{M}_y$ und
- B_y auch mit M_x „funktioniert". D.h., wenn σ_x und σ_y die Substitutionen gemäß (5.32) aus Bemerkung 5.6.6 für $B_{x,1}$ bzw. $B_{y,1}$ sind, daß es die Substitution σ mit $\sigma_y \circ \sigma = \sigma_x$ gibt, womit $\sigma\left(\mathsf{M}_x^i K_{x,1}[\mathsf{M}_x^j]\right) = \mathsf{M}_y^i K_{y,1}[\mathsf{M}_y^j]$ gilt, und daß $B_{x,2}$ bzw. $B_{x,3}$ sich ohne Substitution mit den gleichen Reduktionen wie $B_{y,2}$ bzw. $B_{y,3}$ beweisen lassen.

Bemerkung 5.6.9 τ und σ aus Definition 5.6.8 sind identisch.

Definition 5.6.10 (Allgemeinstes TBMK) Sei X_k die Menge aller TBMK bezüglich der Reduktionskette k. Das *allgemeinste TBMK bezüglich* k ist $\min_{\supseteq}(X_k)$.

Satz 5.6.11 Das allgemeinste TBMK bezüglich einer Reduktionskette k ist bis auf Umbenennung eindeutig bestimmt (vgl. Bemerkung 5.6.3).

Beweis: Die Menge aller TBMK X_k bezüglich einer beliebigen, aber festen, unendlichen Reduktionskette k ist für $\supseteq$ konnex, da es gemäß Definition 5.6.8 für zwei $x, y \in X_k$ mit $x \neq y$ eine Substitution geben muß, damit entweder $x \supseteq y$ oder $y \supseteq x$. Zudem ist X_k endlich, da es nur eine endliche Anzahl Kombinationsmöglichkeiten von Subtermen aus Beweisen oder Mustern gibt, die sich durch Variablen ersetzen lassen. Mit Bemerkung 1.3.10 folgt die Behauptung. □

Definition 5.6.12 (Allgemeinstes Muster) Sei x ein allgemeinstes TBMK bezüglich irgendeiner unendlichen Reduktionskette mit zugehörigem Muster M. M ist ein *allgemeinstes Muster*.

Allgemeinste Muster, ausgehend von einem beliebigen TBMK, lassen sich ohne den „Umweg“ über das allgemeinste TBMK mit folgendem Algorithmus finden:

Algorithmus 5.6.13 Sei $x = (t, \{B_1, B_2, B_3\}, \mathsf{M}, K)$ ein TBMK. Das allgemeinste Muster M' zu x erhält man, indem man B_2 und B_3, also die Reduktionen entsprechend (5.20) bzw. (5.21) auf benötigte Terme hin untersucht. Dabei sind für einen Subterm s diese Fälle zu unterscheiden:

1. s wird in B_2 oder B_3 als Kombinator verwendet.
2. s wird im letzten Term von B_3 als Teil des Körpers von M benötigt, um die nächsthöhere Instanz zu erreichen.

   Die Gesamtheit der Subterme des Musterkörpers ohne Prev bezeichne ich von nun an als *Restkörper*.

   U. U. wird mehr als eine nächsthöhere Musterinstanz erzeugt, d. h., es kann also mehrere Restkörper geben, die möglicherweise von verschiedenen Anfangstermen abgeleitet wurden.
3. s wird nicht verwendet.

1 läßt sich überprüfen, indem jeder Subterm der Startterme von B_2 und B_3 individuell markiert wird. Neue Knoten, die im Laufe der Reduktionen entstehen, erhalten keine Markierung. Wird ein Term s im Laufe des Beweises benötigt, werden die entsprechende Markierung sowie die Markierungen aller Terme s' mit $s \trianglelefteq s'$ aus der Liste der unbenutzten Terme entfernt.

Nun müssen die Restkörper in den Endtermen von B_3 mit dem Restkörper im Startterm von B_2 verglichen werden. Terme, die man im Laufe von B_2 benötigt, müssen auch in B_3 erhalten bleiben, und umgekehrt.

Allerdings ist es möglich, daß einige Teilterme miteinander „synchronisiert“ werden müssen. Zum einen kann es am Ende von B_3 mehrere Musterinstanzen geben, und zum anderen tritt bei einem Nulldurchgang im Laufe von B_3 ein weiteres Vorkommnis der Basis des Musters auf. Wurde also ein Term eines Vorkommnisses im Laufe des Beweises benötigt, so muß auch der entsprechende Term des Partner-Vorkommnisses als unbrauchbar markiert werden.

Alle Subterme aus den jeweiligen Auftreten von Basis und Körper des Musters, die schließlich als unbenutzt markiert sind und deren Positionen sich außen befinden, kann man durch Variablen ersetzen, wobei auch hier gleiche Terme durch gleiche Variablen ersetzt werden müssen. □

Beispiel 5.6.14 In Anlehnung an Satz 5.2.3 seien:

$$\begin{aligned} t &:= \mathbf{S}(\mathbf{SS})(\mathbf{SS})(\mathbf{S}(\mathbf{SS})(\mathbf{SS})) \\ B &:= \{B_1 = (5.1), B_2 = (5.2), B_3 = (5.3)\} \\ \mathsf{M} &:= (\mathbf{SS}\mathsf{Prev}, \mathbf{S}(\mathbf{SS})(\mathbf{SS})) \\ K &:= [] \end{aligned}$$

Und somit sei $x := (t, B, \mathsf{M}, K)$ ein TBMK bezüglich der Reduktionskette k, die sich aus Satz 5.2.3 ergibt. Das allgemeinste TBMK bezüglich k sei weiterhin x'.

Das allgemeinste Muster M' erhalte ich durch die Analyse von B_2 und B_3 entsprechend Algorithmus 5.6.13 (vgl. Abbildung 5.2):

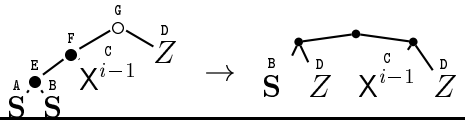
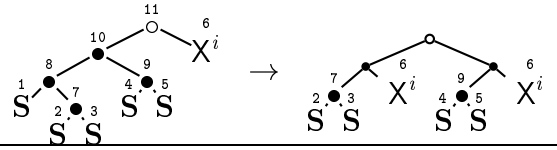
	Herleitung	unbenutzte Terme
B_2		$[\mathtt{B}, \mathtt{C}, \mathtt{D}]$
B_3		$[2, 3, 4, 5, 6, 7, 9]$

Abbildung 5.2.: Allgemeinstes Muster finden

Der Restkörper in B_2 wird durch die Terme $[\mathtt{A}, \mathtt{B}]$ gebildet. In B_3 sind es $[2, 3]$ bzw. $[4, 5]$. Für B_3 wird damit die Liste der unbenutzten Terme zu $[3, 5, 6]$.

Daraus folgt, $\mathsf{M}' = (\mathbf{S}Z\mathsf{Prev}, \mathbf{S}(\mathbf{S}Z)(\mathbf{S}Z))$.

Bemerkung 5.6.15 Obige Rechnungen waren eigentlich nicht in $\mathbf{CL}(\mathbf{S})$, da dessen Signatur keine Markierungen vorsieht. Jedoch läßt sich leicht eine eindeutige Abbildung zwischen markiertem und unmarkiertem $\mathbf{CL}(\mathbf{S})$ konstruieren.

Bemerkung 5.6.16 Eine markierte kombinatorische Logik dient dazu herauszufinden, ob Terme im Laufe einer Reduktion benutzt werden oder nicht. Bei Algorithmus 5.6.13 wird an jeder unbenutzten und bezüglich Präfixordnung minimalen Position (die Position ist also außen) eine Variable eingesetzt. Dies entspricht der Konstruktion einer „Overlap Closure“, die sich bei Zantema und Geser in [ZG96] finden.

Alle Begrifflichkeiten, die ich im Laufe dieses Abschnitts eingeführt habe, sollen die Arbeit mit unendlichen Reduktionsketten und den zugehörigen Mustern und Beweisen erleichtern. Die Grundidee, die hinter dem Konzept der allgemeinsten Muster steckt, ist,

„Bausteine“ zu identifizieren, mit denen sich die unendlichen Reduktionsketten endlich beschreiben lassen. Sehr schön wäre in diesem Zusammenhang die Aussage gewesen, daß die Menge der allgemeinsten Muster endlich ist. Denn dann ließe sich für jeden Term aus $\mathfrak{U}$ durch simples Ausprobieren aller allgemeinsten Muster eine unendliche Reduktionskette im Sinne von Vermutung 5.2.14 finden – vorausgesetzt natürlich, daß die Vermutung selbst stimmt.

Wie der folgende Beweis zeigt, ist die Menge der allgemeinsten Muster unendlich, und zwar deshalb, weil es Grundmuster gibt, die mit ihren allgemeinsten Mustern übereinstimmen. Dies bedeutet, daß im Laufe eines Beweises nach Schema A,B,C oder D tatsächlich alle Subterme des Musters zum Erreichen der jeweils nächsten Instanz benötigt werden.

Satz 5.6.17 Die Menge der allgemeinsten Muster ist unendlich.

Beweis: Es sei $\mathsf{C} := (\mathsf{TPrev}, \mathsf{STS})$ ein Grundmuster. Sei außerdem für alle $n \in \mathbb{N}$ $\mathsf{M}_n := (\mathsf{SPrev}(\mathsf{S}(\mathsf{SC}^n(\mathsf{SPrev}))), \mathsf{SC}^n(\mathsf{SC}^n))$ ein Grundmuster.

Es gilt:

$$\forall i > 0, \forall X, Y \quad : \quad \mathsf{C}^i XY \rightsquigarrow \mathsf{C}^{i-1} XY \tag{5.33}$$

$$\forall X, Y \quad : \quad \mathsf{C}^0 XY \rightsquigarrow Y(X(\mathbf{S}X)Y) \tag{5.34}$$

(5.33) läßt sich in zwei Schritten zeigen, und (5.34) gilt wegen

$$\begin{aligned} & \mathsf{C}^0 XY \to \mathsf{T}X(\mathbf{S}X)Y \to \mathbf{S}(\mathbf{S}X)(X(\mathbf{S}X))Y \\ \to \quad & \mathbf{S}XY(X(\mathbf{S}X)Y) \to X(X(\mathbf{S}X)Y)(\underline{Y(X(\mathbf{S}X)Y)}) \end{aligned}$$

Ferner zeige ich, daß für alle n für M_n die Teile zwei und drei eines Beweises nach Schema C gelten (vgl. Definition 5.2.9 bzw. (5.20) sowie (5.21)) und zwar mit Hauptkontext $\mathbf{S}[]$, Nebenkontext $\mathsf{SC}^n(\mathbf{S}[])$ und $c \in \mathbb{N}$:

$$\forall Z, \forall i > 0 \quad : \quad \mathsf{M}_n^i Z \rightsquigarrow^c \mathsf{M}_n^{i-1} Z \tag{5.35}$$

$$\begin{aligned} \forall i \geq 0 \quad : \quad & \mathsf{M}_n^0(\mathsf{SM}_n^i) \rightsquigarrow^c \mathsf{M}_n^i(\mathsf{SC}^n(\mathsf{SM}_n^i)) \\ & \mathsf{M}_n^0(\mathsf{SC}^n(\mathsf{SM}_n^i)) \rightsquigarrow^c \mathsf{M}_n^{i+1}(\mathsf{SM}_n^{i+1}) \end{aligned} \tag{5.36}$$

(5.35) zeigt man in einem Schritt. Für (5.36) ist folgende Aussage hilfreich:

$$\mathsf{M}_n^0 Z \to \mathsf{C}^n Z(\mathsf{SC}^n Z) \underset{(5.33)}{\rightsquigarrow} \mathsf{C}^0 Z(\mathsf{SC}^n Z) \underset{(5.34)}{\rightsquigarrow} \mathsf{SC}^n Z(\underline{Z(\mathbf{S}Z)(\mathsf{SC}^n Z)}) \tag{5.37}$$

Nun läßt sich (5.36) zeigen. Übergang Haupt-/Nebenkontext:

$$\mathsf{M}_n^0(\mathsf{SM}_n^i) \underset{(5.37)}{\rightsquigarrow} \mathsf{SM}_n^i(\mathsf{S}(\mathsf{SM}_n^i))(\mathsf{SC}^n(\mathsf{SM}_n^i)) \rightsquigarrow \mathsf{M}_n^i(\mathsf{SC}^n(\mathsf{SM}_n^i))$$

Übergang Neben-/Hauptkontext (nach wiederholter Anwendung von (5.35)):

$$
\begin{array}{rl}
 & \mathsf{M}_n^0(\mathsf{SC}^n(\mathsf{SM}_n^i)) \\
\underset{(5.37)}{\rightsquigarrow} & \mathsf{SC}^n(\mathsf{SM}_n^i)(\mathsf{S}(\mathsf{SC}^n(\mathsf{SM}_n^i)))(\mathsf{SC}^n(\mathsf{SC}^n(\mathsf{SM}_n^i))) \\
\rightsquigarrow & \mathsf{C}^n(\mathsf{S}(\mathsf{SC}^n(\mathsf{SM}_n^i)))(\mathsf{SM}_n^i(\mathsf{S}(\mathsf{SC}^n(\mathsf{SM}_n^i)))) \\
\underset{(5.33)}{\rightsquigarrow} & \mathsf{C}^0(\mathsf{S}(\mathsf{SC}^n(\mathsf{SM}_n^i)))(\mathsf{SM}_n^i(\mathsf{S}(\mathsf{SC}^n(\mathsf{SM}_n^i)))) \\
= & \mathsf{C}^0(\mathsf{S}(\mathsf{SC}^n(\mathsf{SM}_n^i)))\mathsf{M}_n^{i+1} \\
\underset{(5.34)}{\rightsquigarrow} & \mathsf{S}(\mathsf{SC}^n(\mathsf{SM}_n^i))(\mathsf{S}(\mathsf{S}(\mathsf{SC}^n(\mathsf{SM}_n^i))))\mathsf{M}_n^{i+1} \\
\rightsquigarrow & \mathsf{SC}^n(\mathsf{SM}_n^i)\mathsf{M}_n^{i+1} \\
\rightarrow & \mathsf{C}^n\mathsf{M}_n^{i+1}(\mathsf{C}^n(\mathsf{SM}_n^i)) \\
\underset{(5.33)}{\rightsquigarrow} & \mathsf{C}^0\mathsf{M}_n^{i+1}(\mathsf{C}^n(\mathsf{SM}_n^i)) \\
\underset{(5.34)}{\rightsquigarrow} & \mathsf{M}_n^{i+1}(\mathsf{SM}_n^{i+1})
\end{array}
$$

Wichtig ist festzustellen, daß im Laufe der obigen Reduktionen alle Subterme des Musters M_n benötigt wurden. Es lassen sich also zu den entsprechenden Starttermen mit den Gleichungen (5.35) und (5.36) unendlich viele unendliche Reduktionsketten beweisen, wobei sich die M_n jeweils nicht weiter verallgemeinern lassen. □

Mit einem allgemeinsten Muster ist es möglich, ganze Klassen von Termen gemäß Vermutung 5.2.14 zu beweisen. Offenbar ist die Struktur von $\mathbf{CL}(\mathbf{S})$ trotz ihrer Einfachheit aber so mächtig, daß im Laufe der Aufzählung von $\mathfrak{U}$ immer wieder Terme auftauchen, die sich nicht auf einen bereits untersuchten Term zurückführen lassen.

Der vorangegangene Beweis zeigt eine Richtung, in der weitergeforscht werden könnte, um mit Mustern unendliche Reduktionsketten doch zu beschreiben, nämlich „Muster in Mustern“.

5.7. Wachstum

Definition 5.7.1 Sei R ein kombinatorisches TES $\in \mathbf{CL}$ und $t := XY$ ein Term. Unter Rückgriff auf Definition 2.2.6 von depth() sei:

$$
\begin{array}{rcl}
\mathrm{depth_L}(t) & := & \mathrm{depth}(X) \\
\mathrm{depth_R}(t) & := & \mathrm{depth}(Y)
\end{array}
$$

In [Wal97] definiert Waldmann eine ähnliche Funktion für $\mathbf{CL}(\mathbf{S})$:

Definition 5.7.2 Seien X und Y Grundterme aus $\mathbf{CL}(\mathbf{S})$:

$$\begin{aligned} \mathrm{d_r}(\mathbf{S}) &:= 0 \\ \mathrm{d_r}(XY) &:= 1 + \mathrm{d_r}(Y) \end{aligned}$$

Mit Hilfe von $\mathrm{d_r}()$ beweist er folgenden Satz, den ich hier ohne Beweis zitiere:

Satz 5.7.3 Sei $t \underset{r_1}{\to} t_1 \underset{r_2}{\to} t_2 \underset{r_3}{\to} t_3 \underset{r_4}{\to} \ldots$ eine unendliche Reduktionskette in $\mathbf{CL}(\mathbf{S})$. Seien $r_1, r_2, r_3 \ldots$ die jeweils reduzierten Redexe. Dann gilt: $\limsup_{i\to\infty} (\mathrm{d_r}(r_i)) \to \infty$.

Aus Satz 5.7.3 folgen sofort:

Korollar 5.7.4 Für eine unendliche Reduktionskette in $\mathbf{CL}(\mathbf{S})$ mit Redexen $r_1, r_2, r_3, \ldots$ ist $\mathrm{depth}(r_i)$ nicht beschränkt.

Korollar 5.7.5 Für eine unendliche Reduktionskette in $\mathbf{CL}(\mathbf{S})$ mit Redexen $r_1, r_2, r_3, \ldots$ ist $\mathrm{size}(r_i)$ nicht beschränkt.

Korollar 5.7.6 Für eine unendliche Reduktionskette in $\mathbf{CL}(\mathbf{S})$ mit Redexen $r_1, r_2, r_3, \ldots$ ist $\mathrm{length}(r_i)$ nicht beschränkt.

In allen bisher untersuchten Termen aus $\mathfrak{U}$ besteht die unendliche Reduktionskette bei feststehender Reduktionsstrategie im wesentlichen aus paarweise auftretenden Instanzen eines Grundmusters M zusammen mit einem Kontext K_1 in der Form $\mathsf{M}^i \overset{\frown}{K_1}[\mathsf{M}^j]$. Dabei muß stets so lange reduziert werden, bis $i = 0$ ist, um $j+1$ zu erreichen. D. h., es scheint stets die Basis von M für das Erzeugen der nächsthöheren Musterinstanz verantwortlich zu sein.

Da $\mathsf{M}^i K_1[\mathsf{M}^j]$ – zwar nicht immer, aber häufig – einen Redex bildet, und somit in der Redexfolge einer unendlichen Reduktionskette immer wieder auftaucht, liegt es nahe, sich linken und rechten Teilterm in den Redexen getrennt anzusehen.

Das Erreichen der Musterbasis in der Redexfolge läßt sich für den Term $\mathbf{S}(\mathbf{SS})(\mathbf{SS})(\mathbf{S}(\mathbf{SS})(\mathbf{SS}))$ aus Satz 5.2.3 sehr schön graphisch darstellen, da es außer Termen mit zwei Musterinstanzen keine weiteren Redexe gibt. Abbildung 5.3 zeigt, wie sich die Tiefe des jeweils linken Teilterms der Redexe verhält: $\mathrm{depth_L}(\mathsf{M}^i K_1[\mathsf{M}^j])$ fällt immer wieder auf $\mathrm{depth}(\mathsf{M}^0)$ zurück.

Anders sieht es hingegen aus, wenn man die Tiefe des gesamten Redexes betrachtet, also $\mathrm{depth}(\mathsf{M}^i K_1[\mathsf{M}^j])$. Abbildung 5.4 ist zu entnehmen, daß die Tiefe des rechten Teilterms und somit des Gesamtredexes sogar monoton wächst (vgl. auch Abbildung B.1).

Reduktionsketten anderer Terme, wie z. B. AAA, bezeugen die Existenz mindestens eines Nulldurchgangs, wie aus Abbildung B.2 ersichtlich ist. Der hier nicht ganz so regelmäßige

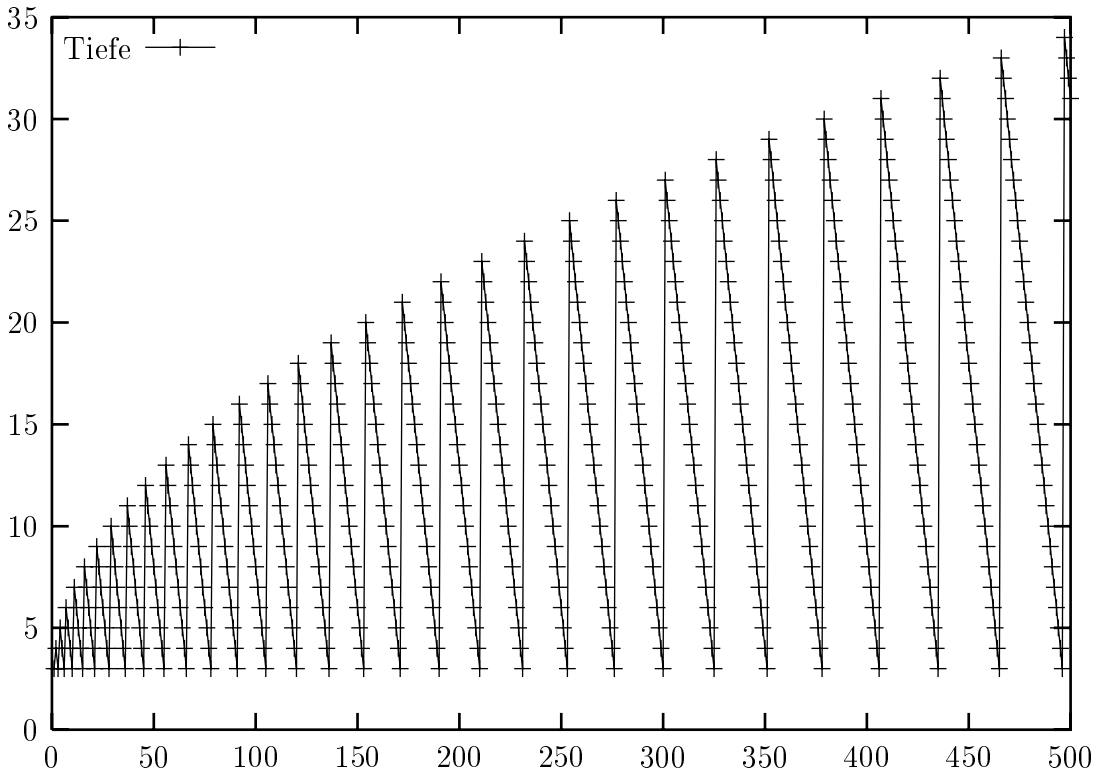


Abbildung 5.3.: STT(STT), $\mathrm{rs}_{\mathrm{LA}}()$, Schrittweite 1, $\mathrm{depth}_{\mathrm{L}}(r_i)$

Verlauf wie in Abbildung 5.3 ist der Tatsache geschuldet, daß auch noch andere Redexe als solche, die aus zwei Musterinstanzen nebst einem eventuellen Kontext bestehen, an der Reduktionskette beteiligt sind. Für weitere Wachstumsfunktionen s. Anhang B.

Insgesamt liegt die nächste Vermutung „auf der Hand“:

Vermutung 5.7.7 (Sublinearität) Sei $t \in \mathfrak{U}$ ein Term und $t \underset{r_1}{\to} t_1 \underset{r_2}{\to} t_2 \underset{r_3}{\to} t_3 \ldots$ die zugehörige unendliche Reduktionskette mit den Redexen r_1, r_2, r_3, ... unter einer festgelegten Reduktionsstrategie. Für die Funktion $\mathrm{depth}_{\mathrm{L}}(r_i)$ gilt:

$$\liminf_{i\to\infty} \mathrm{depth}_{\mathrm{L}}(r_i) < \infty \tag{5.38}$$

$$\limsup_{i\to\infty} \mathrm{depth}_{\mathrm{L}}(r_i) = \infty \tag{5.39}$$

$$\forall c > 0 : \exists i_0 : \forall i > i_0 \quad : \quad \mathrm{depth}_{\mathrm{L}}(r_i) < ci \tag{5.40}$$

Die ersten beiden Gleichungen bedeuten, daß immer wieder absolut kleine Werte vorkommen, die Funktion aber trotzdem nicht nach oben beschränkt ist. Die dritte Gleichung besagt, daß $\mathrm{depth}_{\mathrm{L}}(r_i)$ echt schwächer wächst als jede lineare Funktion.

Am interessantesten sind vom Standpunkt dieser Arbeit aus sicherlich (5.38) und (5.39), denn daraus folgte mit einiger Sicherheit, daß für jeden Term aus $\mathfrak{U}$ mindestens ein

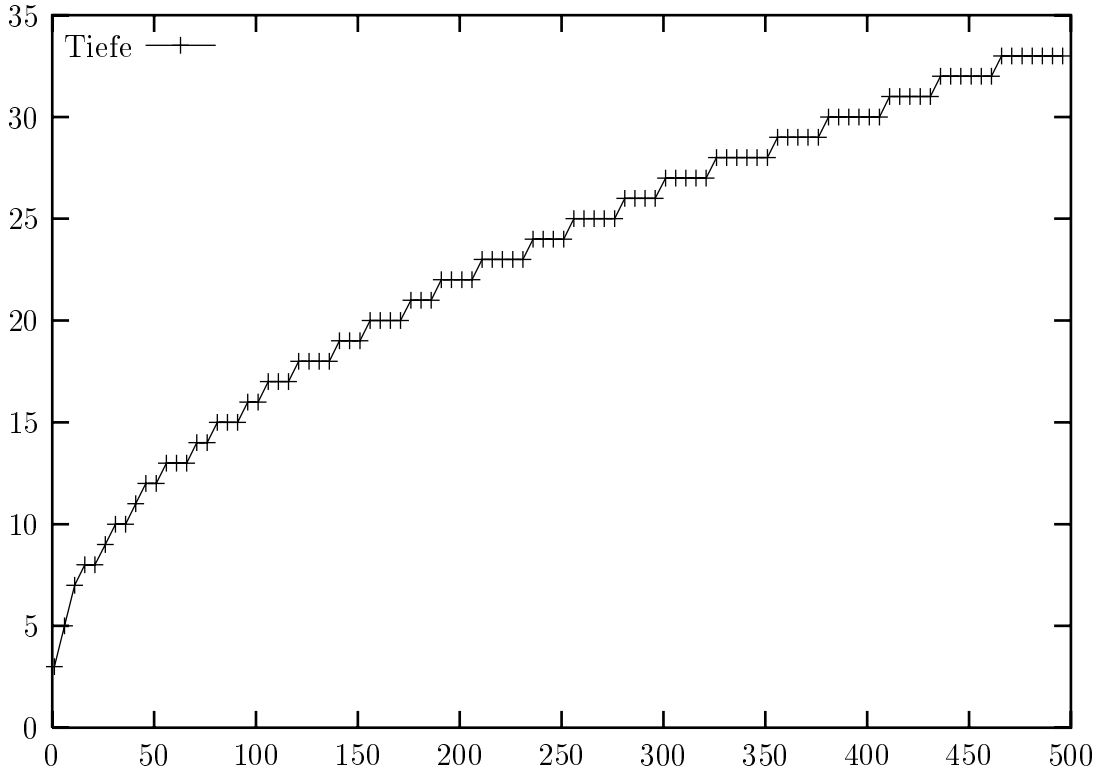


Abbildung 5.4.: $\mathsf{STT}(\mathbf{STT})$, $\mathrm{rs}_{\mathrm{LA}}()$, Schrittweite 5, $\mathrm{depth}_{\mathrm{R}}(r_i)$

induktives Muster existiert, das eine unendliche Reduktionskette verursacht. Außerdem stützen beide Gleichungen die Beobachtungen, daß $\mathrm{depth}_{\mathrm{L}}(r_i)$ zwischen einer konstanten unteren Schranke und einer einhüllenden Funktion, die wie $O(\sqrt{i})$ wächst, oszilliert.

Die Nichtexistenz von Schleifen in $\mathbf{CL}(\mathbf{S})$ ließe sich aus (5.40) folgern, denn eine mehrfach durchlaufene Schleife bedeutete ein lineares Wachstum der Funktion $\mathrm{depth}_{\mathrm{L}}(r_i)$ (vgl. auch Satz 4.4.8).

5.8. Auswertung

Da es sehr mühsam ist, eine größere Anzahl Terme aus $\mathfrak{U}$ manuell zu untersuchen, habe ich die Software `CLS` entwickelt, die das Finden von Mustern, Beweisen und allgemeinsten Mustern automatisiert.

Zu einem gegebenen Term $t \in \mathfrak{U}$ findet `CLS` ein induktives Grundmuster M und liefert Beweise nach Schema A, B oder C, daß M in t eine unendliche Reduktionskette beschreibt. Beweise nach Schema D konnte ich aus Zeitgründen leider nicht mehr implementieren.

In einem zweiten Bearbeitungsschritt berechnet `CLS` das allgemeinste Muster nach Algorithmus 5.6.13.

Für eine detaillierte Beschreibung der Interna von `CLS` sei auf Kapitel 6 und für die Benutzerdokumentation auf Anhang C verwiesen.

Die Dateien mit den errechneten Resultaten sind auf der `CLS`-Homepage (`http://www.doerges.net/cls`) zu finden.

5.8.1. Grundmuster und automatische Beweise

Mit Hilfe von `CLS` habe ich alle Terme der Längen 7-11 aus $\mathfrak{U}$ untersucht (insgesamt 8629 Terme).

Die Parameter (vgl. Kapitel 6) für die einzelnen Suchläufe von `cls` waren:

Suchstrategie: `TDDynamic`

Verifikationsstrategie: `And [Weak,Strong,Iter]`

Reduktionsstrategie: `LeftMostOuterMost` und ggf. `RightMostInnerMost`

Reduktionsschritte 50, wobei bei Nichterfolg auf 100 und anschließend auf 150 erhöht wurde

Gelegentlich habe ich Schrittzahl und verwendete Reduktionsstrategie manuell variieren müssen.

Insgesamt wurden in allen Fällen Muster gefunden, jedoch gelang nicht immer ein Beweis. Die entsprechenden Zahlen sind in Tabelle 5.1 aufgeführt.

length(t)	7	8	9	10	11
Gesamtzahl	2	41	276	1481	6829
verifiziert 1	2	33	255	1357	6366
verifiziert 2	0	4	10	51	224
verifiziert 3+	0	0	0	2	4
unvollständiger Beweis	0	4	11	71	235
kein Beweis	0	0	0	0	0
ohne Muster	0	0	0	0	0

Tabelle 5.1.: Unendliche Reduktionsketten für $t \in \mathfrak{U}$, untersucht mit `CLS`

Als kurze Erläuterung zu den einzelnen Tabellenzeilen:

Gesamtzahl: Untersuchte Terme der gegebenen Länge aus $\mathfrak{U}$; hier gleichzeitig die Menge aller Terme der gegebenen Länge aus $\mathfrak{U}$

verifiziert $1, 2, 3+$: Es wurde ein Beweis gefunden, daß ein, zwei, drei oder mehr Muster eine unendliche Reduktionskette verursachen.

unvollständiger Beweis: `cls` konnte zwar ein Muster, aber nur einen unvollständigen Beweis für eine unendliche Reduktionskette finden. „Unvollständig“ bedeutet, daß die Teile 1 und 2 gemäß Schema C gezeigt werden konnten und daß Teil 3 wenigstens soweit gezeigt werden konnte, daß immer höhere Instanzen des Musters auftreten. (vgl. Abschnitt 6.5)

In der Regel deutet dies auf einen Beweis nach Schema D sowie in Ausnahmefällen auf einen Beweis nach Schema C, der mehr als einen Nulldurchgang erfordert (was `CLS` leider ebenfalls nicht beherrscht).

kein Beweis: bedeutet, daß ein Muster, aber kein Beweis gefunden werden konnte.

Der überwiegende Teil der Terme, die in der Zeile „unvollständiger Beweis“ aufgeführt werden, sind Terme wie ATST (vgl. Satz 5.2.12). Für die Längen acht und neun sind die Terme explizit in Tabelle 5.2 bzw. 5.3 gelistet.

A T S T
S T S S S T
S (A T) S S
S (S T S S) S S

Tabelle 5.2.: Alle $t \in \mathfrak{U}$ der Länge 8, für die `CLS` keinen Beweis findet

S (A T S T)
S (S T S S S T)
S (S (A T) S S)
S (S (S T S S) S S)
A (S T S) T
S T (S T) S T
A T S T S
S T S S S T S
S (S T (S T)) S S
S (A T) S S S
S (S T S S) S S S

Tabelle 5.3.: Alle $t \in \mathfrak{U}$ der Länge 9, für die `CLS` keinen Beweis findet

Dabei fällt auf, daß sich die Terme der Länge acht in höchsten zwei Schritten auf ATST zurückführen lassen und daß in acht Termen der Länge neun jeweils ATST der für die unendliche Reduktionskette verantwortliche Subterm ist. Für die restlichen drei gilt:

$$\mathsf{S(ST(ST))SS} \to \mathsf{ST(ST)ST} \to \mathsf{A(STS)T}$$

In A(STS)T läßt sich mit demselben Muster wie schon für ATST eine unendliche Reduktionskette beweisen.

Besagtes Muster $\mathsf{M}_{\mathsf{ATST}} := \mathsf{TPrev}, \mathbf{S}(\mathsf{T}(\mathbf{ST})(\mathbf{S}(\mathbf{STS})\mathsf{T}))$ wurde in allen 15 Fällen durch `CLS` gefunden (s. auch Satz 5.2.12 bzw. [Zac78]).

Für die übrigen Terme ohne vollständigen Beweis der Längen zehn und elf findet `CLS` in 254 Fällen exakt das Muster $\mathsf{M}_{\mathsf{ATST}}$ und in 15 weiteren Fällen acht Muster, die vom selben Typ sind, da der Körper ebenfalls TPrev und die Basis $\mathbf{S}X$ mit $X \in \mathbf{CL}(\mathbf{S})$ ist (s. Tabelle A.2).

Für $t := \mathsf{T}XYZ$, mit $X, Y, Z \in \mathbf{CL}(\mathbf{S})$, gilt nämlich $t \to^2 YZ(XYZ)$, so daß (5.26) aus Schema D (Definition 5.2.13) auf jeden Fall funktioniert.

Was die 37 verbleibenden Terme (s. Tabelle A.3) betrifft, konnte `CLS` leider keinen Beweis für eine unendliche Reduktionskette finden, was aber damit zu begründen ist, daß alle Beweise gemäß Schema C jeweils zwei Nulldurchgänge benötigen und dieses in `CLS` (noch) nicht implementiert ist. Die gefundenen Muster sind ebenfalls korrekt. Manuelle Beweise sind auf jeden Fall machbar, führten hier jedoch zu weit.

Um die Untersuchungen abzurunden, habe ich große Zufallsterme durch `CLS` bearbeiten lassen (s. Tabelle 5.5). Da bereits ein vergleichsweise kleiner Teilterm dafür ausreicht, daß der gesamte Term keine Normalform besitzt, ist der „durchschnittliche“ Term aus $\mathbf{CL}(\mathbf{S})$ nicht in Normalform. Die Ergebnisse fügen sich nahtlos in die bisherigen ein, da es `CLS` sogar gelungen ist, in jedem Fall einen Beweis zu finden.

Durch meine Resultate sehe ich die Vermutung 5.2.14, daß nämlich die Reduktionsketten in Termen aus $\mathfrak{U}$ in der beschriebenen Art und Weise im wesentlichen auf induktiven Grundmustern beruhen, eindeutig erhärtet.

5.8.2. Allgemeinste Muster

Insgesamt wurden durch `CLS` 8611 Beweise gemäß der Schemata A, B und C gefunden, die sich durch 560 allgemeinste Muster beschreiben lassen (vgl. Tabellen 5.4 und A.4).

length$(t) \leq i$	Gesamt		mit Var.		ohne Var.	
	AM	Beweise	AM	Beweise	AM	Beweise
7	1	2	1	2	0	0
8	17	43	13	36	4	7
9	81	318	68	280	13	38
10	217	1783	193	1601	24	182
11	560	8611	491	7773	69	838

Tabelle 5.4.: Verhältnis allgemeinste Muster zu Beweisen nach Termlänge

Aus der Tabelle ist ersichtlich, daß vergleichsweise wenige allgemeinste Muster benötigt werden, um alle Beweise zu führen. In der Spalte „ohne Var.“ sind diejenigen Grundmuster zusammengefaßt, die gleichzeitig ihr allgemeinstes Muster sind, die sich also nicht weiter verallgemeinern lassen.

Es wird klar Satz 5.6.17 bestätigt, daß die Menge der allgemeinsten Muster unendlich ist. Hier bedeutet dies, daß der Quotient $\frac{\text{Zahl der allgemeinsten Muster}}{\text{Zahl der Beweise}}$ bei steigender Termgröße nicht gegen 0 strebt.

length(t)	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
	53	54	56	57	58	59	60	61	62	63	64	65	66	67	68	70	72	73	77	78
	79	81	83	84	86	87	88	89	90	91	92	93	94	95	96	98	104	107	108	111
	112	115	119	129	130	131	134	144	163	169	185	-	-	-	-	-	-	-	-	-
Gesamt	3	2	3	5	5	7	2	4	7	2	3	3	2	3	5	2	5	1	3	5
	5	3	8	3	9	6	3	1	4	3	3	2	6	3	5	1	2	4	2	3
	3	3	4	3	3	1	1	3	1	1	1	2	2	3	1	3	1	1	2	1
	2	1	2	2	1	4	1	1	1	1	1	1	2	3	1	1	1	2	1	1
	1	1	1	1	1	2	1	1	1	1	1	-	-	-	-	-	-	-	-	-
verif. 1	2	2	0	4	5	7	2	3	7	2	3	3	2	3	4	2	5	1	3	5
	5	2	6	3	6	5	3	1	4	3	2	1	5	3	3	1	2	3	2	2
	3	3	3	3	2	1	1	3	1	1	1	2	2	3	1	3	1	1	2	1
	2	1	1	2	1	2	1	1	1	1	0	0	2	3	1	1	1	2	1	1
	1	1	1	1	0	1	1	1	1	0	1	-	-	-	-	-	-	-	-	-
verif. 2	1	0	3	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
	0	1	2	0	3	1	0	0	0	0	1	1	1	0	1	0	0	1	0	1
	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	1	0	0	2	0	0	0	0	1	1	0	0	0	0	0	0	0	0
	0	0	0	0	1	1	0	0	0	1	0	-	-	-	-	-	-	-	-	-
verif. 3+	1	0	3	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
	0	1	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	-	-

Tabelle 5.5.: Unendliche Reduktionsketten für zufällige $t \in \mathfrak{U}$, untersucht mit `CLS`

Kapitel 6

Software CLS

6.1. Einleitung

Dieses Kapitel erläutert die technischen Seiten der Software CLS, die im Rahmen dieser Arbeit entstand. Ich möchte gleich an dieser Stelle betonen, daß große Teile von CLS – anders als der Name vielleicht vermuten läßt – für beliebige Basen aus **CL** einsetzbar sind. Ferner läßt sich CLS nicht nur in Form der beiden Programme `cls` und `ana` verwenden, sondern die einzelnen Module lassen sich sehr gut mit einem interaktiven Haskellinterpreter wie z.B. Hugs oder GHCi einsetzen (s. [JP99] oder [GHC01]). Ebenso ist es möglich, die von mir geschriebenen Module als Bausteine anderer Programme zu nutzen.

Der Schwerpunkt der folgenden Betrachtungen liegt auf Algorithmen, die spezifisch für **CL**(**S**) sind, und zwar

- dem automatischen Finden von Grundmustern, die unendliche Reduktionsketten in Termen aus **CL**(**S**) verursachen,
- der anschließenden Verifikation, daß diese Muster tatsächlich für eine unendliche Reduktionskette verantwortlich sind, und
- der Herleitung allgemeinster Muster.

Die übrigen Module stelle ich lediglich grob vor.

Für ein detailliertes Verständnis der Module von CLS ist eine Auseinandersetzung mit dem Quellcode sowieso unumgänglich. Immerhin habe ich alle Module sehr ausführlich mit Kommentaren versehen, so daß eine Einarbeitung möglich sein sollte.

Quasi als Gegenstück zu diesem Kapitel ist Anhang C als Benutzerdokumentation gedacht.

CLS wurde in der funktionalen Programmiersprache Haskell in der Sprachfassung 98 (s. [JH99]) geschrieben. Dafür gab es verschiedene Gründe; der Hauptgrund war jedoch die enge Verwandtschaft zwischem dem funktionalen Programmierparadigma und der kombinatorischen Logik. So war es oft möglich, Definitionen o.ä. direkt in ein Modul zu übernehmen.

6.2. Grundlegende Datenstrukturen und Funktionen

Die wichtigste Datenstruktur ist sicherlich `Expr`, die einen Term einer beliebigen kombinatorischen Logik darstellen kann. Die Definition ist in `Types.hs`:

```
data Expr = Var Integer
          | Com Combinator
          | App { left :: Expr
                , right :: Expr }
          | Col { pattern :: Pattern -- "collapsed" following this pat.
                , args    :: [Expr]  -- how to instanciate the vars
                , iteration:: Integer -- for 2nd order patterns only
                , hasredex :: Bool
                , reps    :: Integer -- no. of repetitions
                , spine   :: Direction -- into what dir. reps where found
                }                      -- L if reps == 1
          | PatPrev
```

Die meisten Felder dürften selbsterklärend sein. `PatPrev` darf nur zur Definition eines Musters eingesetzt werden. Es entspricht dem in Definition 4.4.1 eingeführten Prev. Variablen müssen fortlaufend – mit 1 beginnend – durchnummeriert werden.

Ein Kombinator besteht aus:

```
data Combinator = Combinator { cname :: String
                             , carity :: Integer
                             , cbody :: Expr   -- after reduction
                             , cinfo :: String }
```

Durch den Typ `Combinator` wird gleichzeitig auch die komplette Ersetzungsregel definiert. Für die linke Seite reichen dank der Linkslinearität aller kombinatorischen Logiken in **CL** (vgl. Definition 2.4.7) der Name des Kombinators und seine Arität. Die rechte Seite wird durch `cbody` angegeben.

Der wichtigste Kombinator in dieser Arbeit ist (s. Datei `Combinators.hs`):

```
s = Combinator
    { cname = "S", cinfo = "Starling"
    , carity = 3
    , cbody = App (App (Var 1) (Var 3)) (App (Var 2) (Var 3))
    } :: Combinator
```

Der letzte Typ, der sich ebenfalls in der Datei `Types.hs` befindet, dient der Repräsentation von Mustern:

```
data Pattern = Pattern { pname :: String
                       , parity :: Integer -- no. of variables
                       , porder :: Integer -- 0,1,2
                       , pbody :: Expr
                       , pbase :: Expr  } -- 2nd order patterns only
```

Das wichtigste Datenfeld hier ist `porder`, das Werte aus $\{0, 1, 2\}$ annehmen darf. Ein Pattern der Ordnung

- 0 entspricht einer Abkürzung für einen Grundterm (vgl. z.B. 4.2.1).
- 1 bedeutet, daß es für einen beliebigen Term, also einen Term mit Variablen, stehen kann.
- 2 schließlich steht für Muster gemäß Definition 4.4.1.

Das Feld `pbase` muß nur für Pattern 2. Ordnung ausgefüllt werden und `parity` muß gleich der höchsten Variable oder 0 sein.

Nachfolgend zwei Beispiele aus `Patterns.hs`, die sich an Definition 4.2.1 bzw. Satz 5.2.3 anlehnen:

Beispiel 6.2.1 A := **SSS**:

```
a = Pattern { pname = "A"
            , parity = 0
            , pbody = App (App (Com s) (Com s)) (Com s)
            , porder = 0
            , pbase = error "Pattern a: pbase for 2nd order patterns only!"
            } :: Pattern
```

Beispiel 6.2.2 X := (**SS**Prev, **S**TT):

```
x = Pattern { pname = "X"
            , parity = 0
            , pbody = App (pbody t) PatPrev
            , porder = 2
            , pbase = unLeftSpine [(Com s), (pbody t), (pbody t)]
            } :: Pattern
```

Die Module `Expr.hs` und `Pattern.hs` fassen jeweils die entsprechenden Typen sowie verschiedene Manipulations- und Informationsfunktionen auf den Typen zusammen, etwa `depth :: Expr → Integer` oder `subExpr :: Expr → Posn → Maybe Expr`.

In der Datei `Match.hs` finden sich die notwendigen Funktionen, um mit bekannten Mustern arbeiten zu können. Der prominenteste Typ dieser Datei ist

```
data Match = Match {mposn :: Posn      -- Position of the match
                   ,mreps :: Integer   -- No. of reps
                   ,mdir  :: Direction -- Dir. of reps
                   ,mlrpo :: Bool      -- Last repetition points opposite ?
                   ,margs :: [Expr]    -- Values of variables ( if any)
                   ,minst :: Integer}  -- Instance/ iteration for 2nd order pat
```

der im wesentlichen dazu dient, Position, Instanz und Substitution nach Definition 4.4.6 zu spezifizieren.

Die Funktion `match :: Expr → Pattern → [Match]` liefert für einen Term und ein Muster alle passenden Instanzen, wobei Instanzen, die Teile einer höheren Instanz sind, nicht extra zurückgegeben werden.

Alles, was der Auseinandersetzung mit Redexen, Reduktionsketten o.ä. dient, ist in `Reduce.hs` zu finden.

Um eine – möglicherweise unendliche – Reduktionskette zu einem Startterm zu erzeugen, ist `reduces :: Expr → ReductionStrategy → [Expr]` gedacht. Die gewünschte Reduktionsstrategie wird durch einen Parameter vom Typ

```
data ReductionStrategy = LeftMostOuterMost
                       | LeftMostInnerMost
                       | RightMostOuterMost
                       | RightMostInnerMost
                       | Head
```

spezifiziert.

Das Hauptmodul für das Programm `cls` findet sich in den Dateien `Main.hs` und `Option.hs` bzw. in den Dateien `Analyze.hs` und `Analyze_Option.hs` für das Hilfsprogramm `ana`.

6.3. Finden

Das Modul `Search` ist in der Datei `Search.hs` definiert. Da ich mehrere Ansätze zum Suchen von Grundmustern in unendlichen Reduktionsketten in **CL**(**S**) implementiert habe, gibt es mittels des Datentyps

```
data SearchStrategy = TDStatic         -- reduce once
                    | TDDynamic        -- search in reduction chain
```

```
  | BUDissectByDepth   -- try only subexps as possible bases
  | BUUserBases       -- user specified possible bases
```

eine entsprechende Auswahlmöglichkeit.

Grundsätzlich muß zwischen „top-down“- und „bottom-up“-Ansätzen unterschieden werden.

Die Idee, die beiden bottom-up-Ansätzen zu Grunde liegt, ist, zunächst die Basis eines Musters zu finden. Alle Subterme eines vorgegebenen Elements der Reduktionskette werden dazu gezählt; Subterme, die im Vergleich zu anderen häufiger auftreten, sind mit einer höheren Wahrscheinlichkeit wichtige Bestandteile eines Musters. Alle Subterme, deren Häufigkeit über einer bestimmten Schwelle liegt, werden als Basis eines Musters genommen. Nun werden die Kontexte jedes Auftretens – bis zu einer einstellbaren Tiefe – der angenommenen Basis miteinander verglichen. Die am häufigsten vorkommende Teilmenge dieser Kontexte wird als Körper angenommen.

Der Unterschied zwischen `BUDissectByDepth` und `BUUserBases` besteht lediglich darin, welche Subterme für das Histogramm zulässig sind, auf dessen Grundlage die Basis ausgewählt wird.

In der Praxis hat sich leider gezeigt, daß die bottom-up-Ansätze lediglich für Muster mit einem Prev einigermaßen brauchbare Ergebnisse liefern. Grundsätzlich werden nämlich viel zu viele falsche Muster erzeugt, die in nachfolgenden Verifikationsschritten wieder herausgefiltert werden müssen.

Der Ansatz `TDStatic` liefert keine wesentlich besseren Ergebnisse als die beiden vorherigen. Er versucht, erst den Körper eines Musters zu bestimmen und anschließend die Basis. „Static“ soll darauf hindeuten, daß wiederum nur ein vorgegebenes Element der Reduktionskette untersucht wird. Die zugehörige Funktion `findPatternsTDStatic` nimmt an, daß jedes (linke) Rückgrat, das einen Redex enthält, also Terme der Form $\mathbf{C}_i X_1 \ldots X_n$, eine Instanz eines Musters ist. Eine entscheidende Schwäche dieses Ansatzes ist aber, daß alle möglichen Kombinationen für die Position(en) von Prev durchprobiert werden müssen, um auf diese Weise den Musterkörper zu finden. Schon für die Tiefe drei gibt es schlimmstenfalls 15 (wenn man die Wurzelposition () wegläßt, noch 14) Positionen. Die Kardinalität der Potenzmenge einer Menge mit 14 Elementen ist $2^{14} = 16384$.

Die einzige Methode, die befriedigende Ergebnisse liefert, ist `TDDynamic` (vgl. Abbildung 6.1). Auch sie versucht, zunächst den Körper und dann die Basis zu finden. „Dynamic“ bedeutet, daß eine Folge von Redexen betrachtet wird. Und zwar werden dicht beieinander liegende Paare von Redexen, R_i, R_j – genauer (linke) Rückgrate, die einen Kopfredex haben – untersucht. Die entscheidende Annahme dabei ist, daß zwei korrespondierende Subterme, m_i und m_j, aus diesem Paar jeweils eine Instanz des gesuchten

Musters sind. m_j wird in m_i gesucht und für jeden Treffer wird Prev eingefügt. Auf diese Weise kann der Körper des Musters gefunden werden. Die Basis wird auf dieselbe Art und Weise bestimmt wie bei der Methode `TDStatic`, nämlich mit der Funktion `checkPatternBodyAndGetPatternBase :: Expr → Expr →` **Maybe** `Expr`.

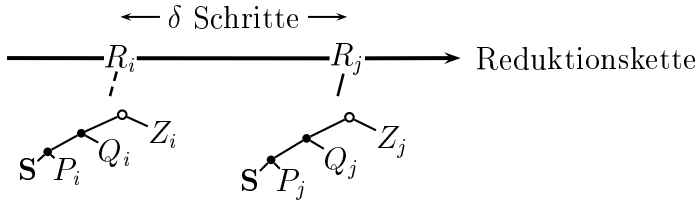


Abbildung 6.1.: Muster mit `TDDynamic` finden

Wichtig zu erwähnen ist, daß alle Suchmethoden lediglich mögliche Muster liefern.

Unabhängig von der eingesetzten Suchstrategie werden alle gefundenen Muster durch die Funktion `filterAndRankPatterns :: [(Pattern,a)] → [(Pattern,a)]` einer ersten Vorauswahl unterzogen. Beispielsweise werden Doubletten oder Muster mit definitiv falschen Körpern oder Basen aussortiert. Außerdem eliminiert `removeLessGeneralAndScore` diejenigen Muster, deren Instanzen durch ein allgemeineres Muster beschrieben werden.

6.4. Verifizieren

Nachdem ein potentieller Kandidat für ein Muster gefunden wurde, ist zu zeigen, daß mit diesem Muster tatsächlich eine unendliche Reduktionskette im dazugehörigen Term beschrieben werden kann. Die dafür nötigen Funktionen sind in `Verify.hs` implementiert, und zwar

- `verifyPatternIter`
- `verifyPatternTDDynamic`
- `verifyPatternStrong`

Um in übergeordneten Modulen eine Wahlmöglichkeit zur Verfügung zu stellen, gibt es den Datentyp

```
data VerificationStrategy = None -- don't verify at all
                          | Weak   -- verifyPatternTDDynamic
                          | Strong -- verifyPatternStrong
                          | Iter   -- verifyPatternIter
                          | And [VerificationStrategy]
```

wobei `And` dazu dient, verschiedene Tests miteinander zu kombinieren.

`verifyPatternIter` reduziert einen gegebenen Ausdruck und gibt in festen Abständen die höchste Musterinstanz aus. Allerdings wird nicht der komplette Term aus der Reduktionskette untersucht, sondern lediglich die Redexe, da die höchsten Musterinstanzen in der Regel an den Redexen beteiligt sind.

`verifyPatternTDDynamic` wurde speziell für die Zusammenarbeit mit `findPatternsTDDynamic` geschrieben. Es werden lediglich zwei Tests ausgeführt, die für den Fall eines positiven Ergebnisses ein starkes Indiz dafür sind, daß das Muster eine unendliche Reduktionskette für den zu untersuchenden Term beschreiben kann. Fällt der Test jedoch negativ aus, kann nicht ausgeschlossen werden, daß das Muster trotzdem eine unendliche Kette erzeugt.

Zum einen wird der „abwärts gerichtete“ Schritt nach Gleichung (5.20) geprüft und zum anderen wird untersucht, ob sich aus $\mathsf{M}^0 Z_i$ eine höhere Musterinstanz finden läßt. Z_i ist der Teil von R_i aus Abbildung 6.1, der zum Mustersuchen nicht herangezogen wird, von dem aber ebenfalls anzunehmen ist, daß er eine Musterinstanz enthält. Im Gegensatz zum ersten Test wird der zweite nicht symbolisch, sondern mit Ausdrücken aus der Reduktionskette durchgeführt.

`verifyPatternStrong` geht analog zu Schema C nach Definition 5.2.9 vor, mit der Einschränkung, daß als Hauptkontext lediglich der triviale Kontext oder $\mathbf{S}[]$ erlaubt sind. Es wird zunächst versucht, eine paarweise Instanz des Musters zu finden (Gleichung (5.19)). Danach überprüft man, ob die abwärts gerichtete Reduktion durchgeführt werden kann (Gleichung (5.20)) und zum Schluß wird versucht, die aufwärts gerichtete zu zeigen (Gleichung (5.21)). Der erste Teil des Beweises arbeitet mit der tatsächlichen Reduktionskette, die letzten beiden werden symbolisch durchgeführt. Für den dritten Beweisteil gilt die Einschränkung, daß maximal ein Nulldurchgang erlaubt ist. Nicht unerwähnt bleiben soll der funktionsinterne Parameter `maxSymRed`, der die letzten beiden Teile des Beweises steuert und per Kommandozeilenoption kontrolliert werden kann.

Es sei hier nochmals darauf hingewiesen, daß die Beweise, die durch `verifyPatternStrong` geführt werden, zwar korrekt sind, es jedoch hinsichtlich der Reduktionsstrategie Unterschiede geben kann (vgl. z.B. Satz 5.2.7).

6.5. Allgemeinste Muster

Der eigentliche Algorithmus, nach dem allgemeinste Muster gefunden werden, wurde bereits in Kapitel 5 beschrieben (s. Algorithmus 5.6.13). Er ist im Modul `Result.hs` in der Funktion `resultToMostGeneralPattern` zu finden.

In `Result_Type.hs` findet sich neben den Datentypen `Proof` und `Result` der Typ `Found` mit

```
data Found = FOK        -- In order : Best  first .
           | FLikelyOK
           | FKO
           | FNone
    deriving (Eq,Ord)
```

Alle drei Typen dienen dazu, die Resultate von `cls` zu beschreiben. Enthält ein Objekt vom Typ `Result` den Wert

- `FOK`, dann bedeutet dies, daß für den entsprechenden Term mindestens ein Muster gefunden wurde, das beweisbar zu einer unendlichen Reduktionskette führt.
- `FLikelyOK` heißt, daß der Beweis vmtl. geführt werden könnte, und zwar nach Schema D oder nach Schema C mit mehr als einem Nulldurchgang. Genauer gesagt, müssen `verifyPatternTDDynamic` und die ersten beiden Beweise in `verifyPatternStrong` erfolgreich gewesen sein.

  Leider berücksichtigt der Parser für die Beweise (`proof` in `Result_Read.hs`) die Ausgabe von `verifyPatternTDDynamic` noch nicht, sondern nimmt per Default immer an, daß das entsprechende Ergebnis erfolgreich war. Dies ist deswegen keine besondere Einschränkung, da die Voreinstellung von `cls` eine Prüfung mit `verifyPatternTDDynamic` immer vorsieht und auch nur entsprechende Beweise in der Ausgabe auftauchen.
- `FKO` schließlich bedeutet, daß zwar ein Musterkandidat gefunden wurde, der Beweis aber nicht geführt werden kann (und daher der Musterkandidat ziemlich sicher nicht zu einer unendlichen Reduktionskette paßt).

Auf den Werten von `Found` ist eine Ordnung definiert mit `FOK` als bestem Wert. Dies ist hauptsächlich zum Filtern der Resultate (z.B. mit `ana`) interessant, da ein Filter vom Wert `FLikelyOK` so gedacht ist, daß für den jeweiligen Ausdruck mindestens ein Resultat mit `FLikelyOK`, aber kein besseres vorhanden sein darf.

Das Modul `Result_Read` definiert einen Parser, der die Ausgabe des Programms `cls` lesen kann, um sie anschließend mit `ana` weiterverarbeiten zu können.

`Label.hs` bietet Funktionen, die nötig sind, um mit markierter kombinatorischer Logik zu arbeiten, z.B.

- `label :: Expr → Expr`
- `reduceL :: Expr → ReductionStrategy →` **Maybe** `(Expr,Label)`.

6.6. Potential

Die Programme `cls` und `ana` haben im Rahmen dieser Arbeit zu meiner vollen Zufriedenheit funktioniert, jedoch gibt es in jedem Fall noch Spielraum für Verbesserungen.

Insbesondere sollte `cls` auch Beweise nach Schema C mit mehr als einem Nulldurchgang sowie Beweise nach Schema D führen können. Des weiteren wäre eine durchgängigere Benutzung der Typen aus `Result_Type.hs` insbesondere in den Modulen `Search` und `Verify` sehr anzustreben.

Vielleicht kann auch bei der Laufzeit noch einiges eingespart werden, da die Untersuchung einiger weniger Terme besonders lange dauert (vorzugsweise im Zusammenhang mit der Reduktionsstrategie `RightMostInnerMost`).

Eine genaue diesbezügliche Auflistung findet sich in den Dateien `TODO` und `KNOWNPROBLEMS`.

Zum Schluß möchte ich auf die Homepage von `CLS` verweisen:

`http://www.doerges.net/cls`.

Kapitel 7

Zusammenfassung

Ziel der Arbeit war es, unendliche Reduktionsketten zu untersuchen, um so ein mögliches Entscheidungsverfahren für das Wortproblem in $\mathbf{CL}(\mathbf{S})$ voranzubringen.

Das Verständnis unendlicher Reduktionsketten ist unabdingbar für die Beschreibung unendlicher Normalformen, mit deren Hilfe wiederum das Wortproblem entschieden werden könnte.

Haupthilfsmittel bei meinen Betrachtungen waren induktive Muster, mit denen sich zum einen die Existenz unendlicher Reduktionsketten in $\mathbf{CL}(\mathbf{S})$ beweisen läßt und mit denen sich zum anderen eben diese unendlichen Reduktionsketten charakterisieren lassen.

Neben vielen kleineren Aspekten, die ich untersucht habe, habe ich verschiedene Arten für die Beweise unendlicher Reduktionsketten klassifiziert und starke empirische Indizien dafür zusammengetragen, daß diese Klassifizierung vollständig ist (s. Vermutung 5.2.14).

Zudem ist die Grundidee stets dieselbe, so daß Struktur der Beweise sowie die Reduktionsketten selbst unabhängig vom gewählten Beweisschema eng miteinander verzahnt sind. Beide beruhen darauf, daß immer wieder paarweise Instanzen eines induktiven Musters (u.U. mit einem zusätzlichen Kontext) vorkommen, die gemeinsam für das Erreichen der nächsthöheren Instanz sorgen. Dazu oszilliert die linke Instanz im Laufe der Reduktionskette regelmäßig zwischen 0 und $O(\sqrt{n})$ (vgl. Vermutung 5.7.7).

Ließe sich tatsächlich beweisen, daß dies immer so ist, wäre übrigens auch gleich die Schleifenfreiheit von $\mathbf{CL}(\mathbf{S})$ gezeigt.

Als Hilfsmittel im Rahmen dieser Arbeit ist die Software `CLS` entstanden, die Finden von Mustern und Beweisen unendlicher Reduktionsketten automatisiert. Anders wären die vielen tausend Beweise gar nicht machbar gewesen. So ist es mir u.a. gelungen, die Arbeit von Zachos [Zac78] fast vollständig per Computer nachzuahmen (s. Tabelle A.1).

Über die Untersuchung unendlicher Reduktionsketten mittels Mustern hinaus habe ich versucht, den „Bausteinansatz“ durch das Konzept der allgemeinsten Muster fortzuführen. Allgemeinste Muster entstehen aus einem Beweis zu einer unendlichen Reduktionskette, in dem alle unnötigen Subterme des entsprechenden Musters durch Variablen ersetzt werden. So wird das Muster auch für andere Beweise verwendbar.

Allerdings ist die Klassifikation durch allgemeinste Muster noch nicht ausreichend, da ich mit Hilfe von „Mustern in Mustern“ zeigen konnte, daß die Menge der allgemeinsten Muster unendlich ist (vgl. Satz 5.6.17).

Dementsprechend wäre eine Richtung für weitere Untersuchungen, nach Möglichkeiten zu forschen, allgemeinste Muster weiter zusammenzufassen.

Unabhängig davon ist der Minimalautomat für das Halteproblem in $\mathbf{CL}(\mathbf{S})$ bekannt (s. [Wal97]) und somit auch das syntaktische Monoid der Sprache. Es wäre interessant zu untersuchen, ob zwischen den von mir identifizierten Beweisschemata und Klassen des Monoids Zusammenhänge bestehen.

Literaturverzeichnis

[AO94] Ansorge, Rainer ; Oberle, Hans J.: *Mathematik für Ingenieure.* Bd. 1: *Lineare Algebra und analytische Geometrie, Differential- und Integralrechnung einer Variablen.* Berlin : Akademie Verlag, 1994

[Ave95] Avenhaus, Jürgen: *Reduktionssysteme.* Springer, 1995. – ISBN 3–540–58559–1

[Bar84] Barendregt, Hendrik P.: *The Lambda Calculus, Its Syntax and Semantics.* Elsevier Science Publishers, 1984. – ISBN 0–444–86748–1

[BK79] Bergstra, Jan ; Klop, Jan W.: Church–Rosser strategies in the Lambda calculus. In: *Theoretical Computer Science* 9 (1979), S. 27–38

[BN98] Baader, Franz ; Nipkow, Tobias: *Term Rewriting and All That.* Cambridge University Press, 1998. – ISBN 0–521–45520–0

[CF58] Curry, Haskell B. ; Feys, Robert: *Combinatory Logic, Volume I.* Amsterdam : North-Holland, 1958 (Studies in Logic and the Foundations of Mathematics). – xvi+417 S

[Chu36] Church, Alonzo: A note on the Entscheidungsproblem. In: *Journal of Symbolic Logic* 1 (1936), S. 40–41, 101–102

[CLR91] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L.: *Introduction to algorithms.* 3. Auflage. Cambridge, Mass. [u.a.] : The MIT Press [u.a.], 1991 (The MIT electrical engineering and computer science series), S. 303 ff. – ISBN 0–262–03141–8

[Cur30] Curry, Haskell B.: Grundlagen der kombinatorischen Logik. In: *Amer. J. Math.* 52 (1930), S. 509–536, 789–834

[DJ90] Dershowitz, Nachum ; Jouannaud, Jean-Pierre: *Handbook of Theoretical Computer Science.* Bd. B: *Rewrite Systems*. Elsevier Science Publishers B.V., 1990, S. 243–319. – ISBN 0–262–72020–5

[DJ91] Dershowitz, Nachum ; Jouannaud, Jean-Pierre: Notations for Rewriting. In: *Bull. EATCS* 43 (1991), Februar, S. 162–172

[GHC01] The GHC Team: *The Glasgow Haskell Compiler User's Guide.* 2001. – `http://www.haskell.org/ghc/`

[GKP90] Graham, Ronald L. ; Knuth, Donald E. ; Patashnik, Oren: *Concrete Mathematics*. 6., verb. Auflage. Addison–Wesley Publishing Company, Oktober 1990. – ISBN 0–201–14236–8

[Gra92] Gramlich, Bernhard: Relating innermost, weak, uniform and modular termination of term rewriting systems. In: Voronkov, A. (Hrsg.): *LPAR'92*, Springer Verlag, 1992 (Lecture Notes in Artificial Intelligence 624), S. 285–296

[Gru97] Gruska, Jozef: *foundations of Computing*. International Thomson Computer Press, 1997. – ISBN 1–85032–243–0

[HS86] Hindley, J. R. ; Seldin, Jonathan P.: *Introduction to Combinators and λ-calculus*. Cambridge University Press, 1986. – ISBN 0–521–26896–6

[Hud00] Hudak, Paul: *The Haskell School of Expression*. Cambridge University Press, 2000. – ISBN 0–521–64408–9

[Jän96] Jänich, Klaus: *Lineare Algebra*. 6. Auflage. Springer, 1996 (Springer Lehrbuch). – ISBN 3–540–59223–7

[JH99] Jones, Simon P. ; Hughes, John: *Haskell 98: A Non-strict, Purely Functional Language*, 1999. – `http://www.haskell.org/definition/`

[JP99] Jones, Mark P. ; Peterson, John C.: *Hugs 98, A functional programming system based on Haskell 98, User Manual*, September 1999. – `http://cvs.haskell.org/Hugs/downloads/hugs.ps.gz`

[Kee96] Keenan, David C. *To Dissect a Mockingbird: A Graphical Notation for the Lambda Calculus with Animated Reduction*. `http://uq.net.au/~zzdkeena/Lambda/index.htm`. August 1996

[Klo80] Klop, Jan W.: *Reduction Cycles in Combinatory Logic*. Academic Press, 1980 (To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism), S. 193–214. – ISBN 0–12–349050–2

[Oya87] Oyamaguchi, M.: The Church-Rosser property for ground term rewriting systems is decidable. In: *Theoretical Computer Science* 49 (1987), S. 43–79

[PS01] Probst, Dieter ; Studer, Thomas: How To Normalize the Jay. In: *Theoretical Computer Science* 254 (2001), Nr. 1-2, S. 677–681

[PWZ96] Petkovsek, Marko ; Wilf, Herbert ; Zeilberger, Doron: $A = B$. A. K. Peters, 1996. – ISBN 1–56881–063–6

[RS97] Rozenberg, G. (Hrsg.) ; Salomaa, A. (Hrsg.): *Handbook of Formal Languages*. Springer, 1997 (Handbook of Formal Languages). – ISBN 3–540–61486–9

[Sch24] Schönfinkel, Moses: Über die Bausteine der mathematischen Logik. In: *Math. Annalen* 92 (1924), S. 305–316

[Sch95] Schöning, Uwe: *Logik für Informatiker.* 4., überarb. Auflage. Heidelberg : Spektrum, Akad. Verlag, 1995. – ISBN 3–86025–684–x

[Smu85] Smullyan, Raymond: *To Mock a Mockingbird: and other logic puzzles including an amazing adventure in combinatory logic.* Knopf, New York, 1985

[SP97] Schwarz, Stefan ; Potuček, Rudolf: *Das TEXikon : Referenzhandbuch für TEX und LATEX.* Bonn ; Reading [u.a.] : Addison–Wesley–Longman, 1997

[Sta89] Statman, Richard: The word problem for Smullyan's lark combinator is decidable. In: *J. Symbolic Comput.* 7 (1989), S. 103–112

[SWB93] Sprenger, M. ; Wymann-Böni, M.: How To Decide The Lark. In: *Theoretical Computer Science* 110 (1993), S. 419–432

[Wal97] Waldmann, Johannes: *The Combinator S*, Friedrich-Schiller-Universität Jena, Diss., 1997

[Wal98] Waldmann, Johannes: Normalization of S terms is decidable. In: Nipkow, Tobias (Hrsg.): *9th RTA*, 1998 (LNCS 1379), S. 138–150

[Wec92] Wechler, W. ; Brauer, W. (Hrsg.) ; Rozenberg, G. (Hrsg.) ; Salomaa, A. (Hrsg.): *Universal Algebra for Computer Scientists.* Berlin : Springer, 1992 (EATCS Monographs on Theoretical Computer Science). – ISBN 3–540–54280–9

[Zac78] Zachos, Efstathios: Kombinatorische Logik und S-Terme / Eidgenössische Technische Hochschule Zürich. 1978 (26). – Berichte des Instituts für Informatik

[ZG96] Zantema, H. ; Geser, A.: Non-Looping Rewriting / Utrecht University. 1996 (UU-CS-1996-03). – Forschungsbericht

Anhang A

Musterlisten

In diesem Abschnitt finden sich verschiedene Tabellen mit Resultaten meiner Software `CLS`.

Tabelle A.1 gibt die Ergebnisse der Untersuchung aller Terme aus $\mathfrak{U}$ der Längen sieben und acht mit `cls` wieder, um zu verdeutlichen, daß fast sämtliche Resultate von Zachos aus [Zac78] maschinell gefunden werden konnten.

Mit „Kontext“ ist der Hauptkontext im Sinne von Definition 5.2.9 gemeint. Nulldurchgänge und Nebenkontext(e) sind nicht extra angegeben. „Red.-Schritte“ bezeichnet die Obergrenze, bis zu der `cls` gesucht hat.

Die verwendeten Abkürzungen für Grundterme sind

$$\begin{aligned} \mathsf{B} &:= \mathbf{S}\,\mathsf{T} \\ \mathsf{C} &:= \mathbf{S}(\mathbf{S}(\mathbf{S}(\mathbf{S}\,\mathsf{T}(\mathbf{S}(\mathbf{S}\,\mathsf{T}\,\mathbf{S})\mathbf{S}))))\end{aligned}$$

Tabelle A.1.: Resultate von `CLS`

Term	Kontext	Red.-Strat.	Red.-Schritte	Ergebnis
A T S S	[]	RI	50	OK
	S (C C) Prev, C (S (C C))			
B S S S S	[]	RI	50	OK
	S (C C) Prev, C (S (C C))			
S (A T S S)	[]	RI	50	OK
	S (C C) Prev, C (S (C C))			
S (B S S S S)	[]	RI	50	OK
	S (C C) Prev, C (S (C C))			
S A S B	[]	LA	50	OK
	S B Prev, S B			
S A S A	[]	LA	50	OK
	S A Prev, S (S A) (S A (S A))			
A B T	[]	LA	50	OK
	T Prev, S B T			

Tabelle A.1.: Resultate von `CLS` (Fortsetzung)

Term	Kontext	Red.-Strat.	Red.-Schritte	Ergebnis
A A T	[]	LA	50	OK
	T Prev, B (S A T)			
B S T T	[]	LA	50	OK
	T Prev, T B T			
S (S A) S T	[]	LA	50	OK
	T B Prev, S B (T B)			
S (B S) S T	[]	LA	50	OK
	B Prev, T B B			
S (B S) S T	[]	LA	50	OK
	B B Prev, B (T B B)			
S T T S T	[]	LA	50	OK
	T Prev, B (S A T)			
A T S T	S[]	LA	50	vmtl. verif.
	T Prev, S (T B (S (B S) T))			
S B S S T	[]	LA	50	OK
	T Prev, T B T			
B S S S T	S[]	LA	50	vmtl. verif.
	T Prev, S (T B (S (B S) T))			
A (S B) S	[]	RI	50	OK
	S (S Prev) (S Prev (S (S Prev))), S (S B) S			
A (S A) S	[]	RI	50	OK
	S (S (S Prev)) (S Prev (S (S Prev))), S (S A) S			
A (B S) S	[]	RI	50	OK
	S (C C) Prev, C (S (C C))			
B S B S	S[]	RI	50	OK
	S Prev (S (S (S B S) (S Prev))), S (S B S) (S (S B S))			
B S A S	[]	RI	50	OK
	S (S (S Prev)) (S Prev (S (S Prev))), S (S A) S			
S (S A) T S	[]	LA	50	OK
	S A Prev, S (S A) (S A (S A))			
A T T S	[]	LA	50	OK
	T (S T T) S Prev, S (T (S T T) S) (S A (T (S T T) S))			
S B S T S	S[]	RI	100	OK
	S Prev (S (S (T (B S)) (S Prev))), S (T (B S)) (S (T (B S)))			
S A S T S	S[]	RI	100	OK

Tabelle A.1.: Resultate von `CLS` (Fortsetzung)

Term	Kontext	Red.-Strat.	Red.-Schritte	Ergebnis
	S Prev (S (S (B S) (S Prev))), S (S (B S) (S (B S))) (S (S (B S) (S (B S))))			
B S S T S	[]	LA	50	OK
	T (S T T) S Prev, S (T (S T T) S) (S A (T (S T T) S))			
S (S (S A)) S S	[]	LA	50	OK
	T B Prev, S B (T B)			
S (S (B S)) S S	[]	LA	50	OK
	B Prev, T B B			
S (S (B S)) S S	[]	LA	50	OK
	B B Prev, B (T B B)			
S (S T T) S S	[]	LA	50	OK
	T Prev, B (S A T)			
S (A T) S S	S[]	LA	50	vmtl. verif.
	T Prev, S (T B (S (B S) T))			
S (S B S) S S	[]	LA	50	OK
	T Prev, T B T			
S (B S S) S S	S[]	LA	50	vmtl. verif.
	T Prev, S (T B (S (B S) T))			
B B S S	[]	RI	50	OK
	S (C C) Prev, C (S (C C))			
A B S S	[]	RI	100	OK
	S (S (S Prev)) (S Prev (S (S Prev))), S (S (S (S B S)) S) (B (S (S (S B S)) S))			
A A S S	[]	RI	50	OK
	S Prev (T Prev), T (B (T T))			
S B T S S	[]	RI	50	OK
	S (S (S Prev)) (S Prev (S (S Prev))), S (S A) S			
B S T S S	[]	RI	100	OK
	S (S (S Prev)) (S Prev (S (S Prev))), B (S (S (T (B S))) S)			
S (S B) S S S	S[]	RI	100	OK
	S Prev (S (S (T (B S)) (S Prev))), S (T (B S)) (S (T (B S)))			
S (S A) S S S	S[]	RI	100	OK
	S Prev (S (S (B S) (S Prev))), S (S (B S) (S (B S))) (S (S (B S) (S (B S))))			

Tabelle A.1.: Resultate von CLS (Fortsetzung)

Term	Kontext	Red.-Strat.	Red.-Schritte	Ergebnis
S (B S) S S S	[]	LA	50	OK
	T (S T T) S Prev, S (T (S T T) S) (S A (T (S T T) S))			
S T T S S S	[]	RI	50	OK
	S Prev (T Prev), T (B (T T))			
A T S S S	[]	RI	50	OK
	S (C C) Prev, C (S (C C))			
S B S S S S	[]	RI	100	OK
	S (S (S Prev)) (S Prev (S (S Prev))), B (S (S (T (B S))) S)			
B S S S S S	[]	RI	50	OK
	S (C C) Prev, C (S (C C))			

In Tabelle A.2 sind alle 15 Terme der Längen zehn und elf, sortiert nach Mustern (insgesamt acht), für die die Beweise der unendlichen Reduktionsketten nach Schema D ablaufen und die als Musterbasis weder $\mathsf{M}^0_{\mathsf{ATST}} = \mathbf{S}(\mathsf{T}(\mathbf{S}\mathsf{T})(\mathbf{S}(\mathbf{S}\mathsf{T}\mathbf{S})\mathsf{T})$ noch die reduzierte Form $\mathbf{S}(\mathbf{S}(\mathbf{S}(\mathbf{B}\mathbf{S})\mathsf{T})(\mathsf{B}(\mathbf{S}(\mathbf{B}\mathbf{S})\mathsf{T})))$ haben.

Die vier Terme $\mathsf{A}(\mathbf{S}(\mathbf{B}\mathbf{S})\mathsf{T})\mathbf{S}$, $\mathsf{A}(\mathbf{S}\mathsf{B}\mathbf{S})\mathsf{T}$, $\mathsf{B}(\mathbf{S}\mathsf{B})\mathbf{S}\mathsf{T}$ und $\mathbf{S}(\mathsf{B}(\mathbf{S}\mathsf{B}))\mathbf{S}\mathbf{S}$ finden sich in der Tabelle doppelt, da es für sie (mindestens) zwei Muster gibt.

Last, but not least, sei angemerkt, daß die Basis des Musters $(\mathsf{TPrev}, \mathbf{S}(\mathsf{B}(\mathsf{TB})(\mathbf{S}(\mathbf{S}\mathsf{B}\mathbf{S})\mathsf{T})))$ keine Normalform hat (daher auch nur für äußere Strategien von CLS gefunden wurde) und daß sich die Basis von $(\mathsf{TPrev}, \mathbf{S}(\mathsf{T}(\mathbf{S}(\mathbf{B}\mathbf{S}))\mathsf{T}))$ in die Basis von $(\mathsf{TPrev}, \mathbf{S}(\mathsf{B}(\mathbf{S}(\mathbf{B}\mathbf{S})\mathsf{T})))$ überführen läßt.

Tabelle A.2.: $t \in \mathfrak{U}$ sortiert nach Mustern, Beweise vermutl. nach Schema D

M =(T Prev, S (T (S (B S) T)))	1
S B (S (B S)) T	
M =(T Prev, S (B (S (B S) T)))	1
S A (S (B S)) T	
M =(T Prev, S (S (B S) T))	1
B S (S (B S) T)	
M =(T Prev, S (S (B A) (S (S (B S) T) S)))	1
A (S (B S) T) S	
M =(T Prev, S (S (S (S B S) T) (B (S (S B S) T))))	3

Tabelle A.2.: $t \in \mathfrak{U}$ sortiert nach Mustern, Beweise vermutl. nach Schema D (Fortsetzung)

S (B (S B)) S S A (S B S) T B (S B) S T	
M =(T Prev, S (S (S (S (B S) T) S) (S (S (S (B S) T) S))))	1
A (S (B S) T) S	
M =(T Prev, S (T (S (B S)) T))	2
T T (S (B S)) T A S (S (B S)) T	
M =(T Prev, S (B (T B) (S (S B S) T)))	9
S (S (B (S B)) S S) S (A (S B S) T) S (B (S B) S T) S (B (S B)) S S A (S B S) T B (S B) S T S (B (S B)) S S S A (S B S) T S B (S B) S T S	

In der Tabelle A.3 sind die 37 Terme – sortiert nach vier Mustern, für die `CLS` keinen Beweis finden konnte. Alle betreffenden Beweise für die Existenz unendlicher Reduktionsketten nach Schema C erfordern nämlich zwei Nulldurchgänge, was `CLS` (leider) noch nicht beherrscht.

Tabelle A.3.: $t \in \mathfrak{U}$ sortiert nach Mustern, (noch) keine Beweise durch `CLS` gefunden

(S (S (S (S Prev) (T (S Prev)))) (S Prev (S (S (S Prev) (T (S Prev))))), S (S (B S) (T T)) (S (T T) (S (B S) (T T))))	23
S (S (A T) A S) S (S (B S S) A S) S (A T S (T T)) S (A T S (A S)) S (B S S S (T T)) S (B S S S (A S)) S (B B) A S S (A T) A S	

Tabelle A.3.: $t \in \mathfrak{U}$ sortiert nach Mustern, (noch) keine Beweise durch CLS gefunden (Fortsetzung)

S (B S S) A S A (B S) (T T) A (B S) (A S) B B S (T T) B B S (A S) S (A T) A S S S (B S S) A S S A T S (T T) A T S (A S) B S S S (T T) B S S S (A S) A T S (T T) S A T S (A S) S B S S S (T T) S B S S S (A S) S	
(S (S (S A (S Prev))) (S Prev (S (S A (S Prev)))), S (S A) (S (B S) (S A)))	8
S (A T S (S A)) S (B S S S (S A)) A (B S) (S A) B B S (S A) A T S (S A) B S S S (S A) A T S (S A) S B S S S (S A) S	
(S (S Prev (S (S (T T) (S Prev)))) (S (S (T T) (S Prev)) (S Prev (S (S (T T) (S Prev))))), S (S (T T)) (S (B S) (S (T T))))	4
A T S (S (T T)) A T S (S (A S)) B S S S (S (T T)) B S S S (S (A S))	
(S (S (S B) (S Prev)) (S Prev (S (S (S B) (S Prev)))), S (S (S B)) (S (B S) (S (S B))))	2
A T S (S (S B)) B S S S (S (S B))	

Tabelle A.4 enthält die häufigsten allgemeinsten Muster, sortiert nach der Anzahl der zusammengefaßten Beweise.

Tabelle A.4.: Häufigste allg. Muster mit Zahl der beschriebenen Beweise

Allgemeinstes Muster	Beweise
(S X Prev, S (S Y) (S X))	1066
(S X Prev, S Y (S Z (S (S U) (S X))))	263
(S Prev (X Prev), S Y (S (S Z) (T X)))	235
(T X S Prev, S (T Y S) (S (T Z) (T X S)))	210
(S X Prev, S Y (S (S Z) (S X)))	180
(S (S (S Prev)) (S Prev (S (S Prev))), S (S A) S)	175
(S X Prev, T (S (S X)) (S X))	174
(T X Prev, S (S Y) (T X))	164
(S X Prev, S (S Y) (S (T Z) (S X)))	153
(S X Prev, S (S (T Y)) (S (T Z) (S X)))	149
(S X Prev, S Y (S (T (S Z U)) (S X)))	137
(S Prev (S (S (B S) (S Prev))), S (S (B S) (S (B S))) X)	134
(S (S (S Prev)) (S Prev (S (S Prev))), S (S X) (S (S (T (B S))) S))	121
(S X Prev, S (S X) (S X))	112
(S X Prev, S (S Y) (S (T Z S) (S X)))	112
(B (T X) Prev, T Y (T B (T X)))	103
(S Prev (X Prev), S Y (S Z (S (S U) (T X))))	103
(S Prev (X Prev), S Y (S (S (S Z) (T X)) U))	101
(S X Prev, S Y (S Z (S (T U) (S X))))	101
(S (S X) Prev, T (S (S Y)) (S (S X)))	96

Anhang B
Redexwachstum

Die folgenden Graphiken wurden automatisch als Eingabedateien für Gnuplot erzeugt, das dann die Interpolation übernahm. Um eine gewisse Übersichtlichkeit zu wahren, sind die tatsächlichen Datenpunkte nicht immer extra markiert. Aus demselben Grund wurden teilweise auch eine geringere Anzahl an Redexen oder eine größere Schrittweite gewählt. „Schrittweite n“ bedeutet, daß aus der Reduktionskette der jeweils $n$te Redex betrachtet wurde.

Als Ergänzung zu den Abbildungen 5.3 und 5.4 visualisiere ich noch das Wachstum für die Tiefe des kompletten Redexes der unendlichen Reduktionskette von **STT**(**STT**) (s. Abbildung B.1).

Bei den weiteren Abbildungen beziehen sich jeweils drei auf dieselbe Reduktionskette.

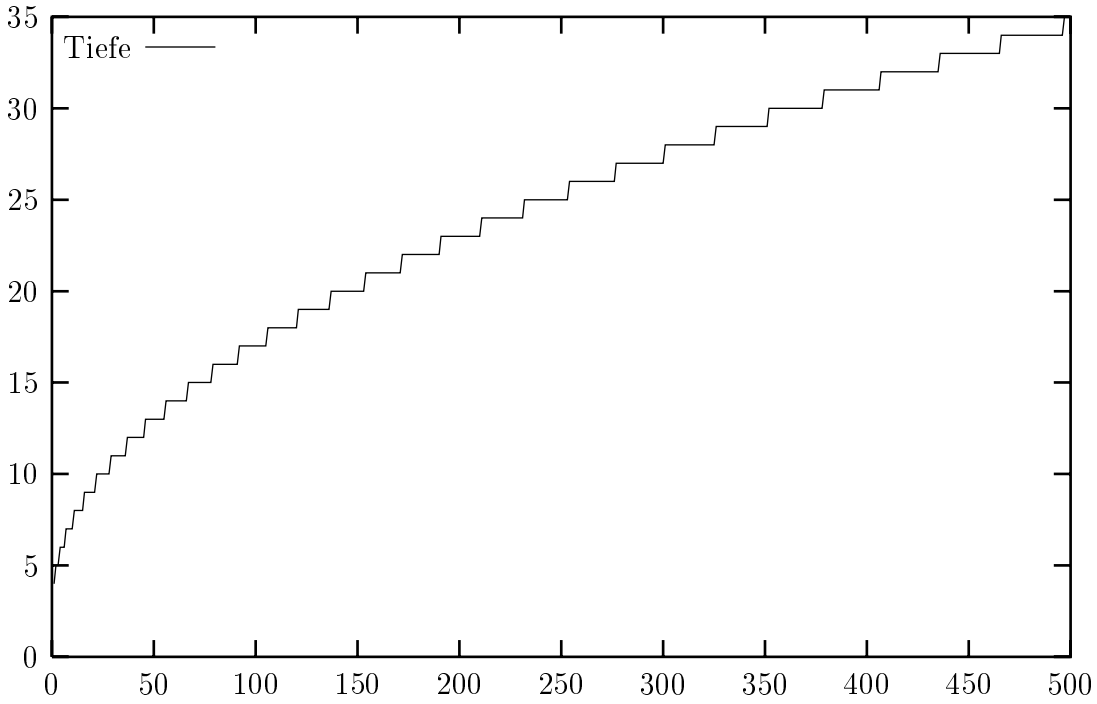


Abbildung B.1.: **STT**(**STT**), $\mathrm{rs}_{\mathrm{LA}}()$, Schrittweite 1, $\mathrm{depth}(r_i)$

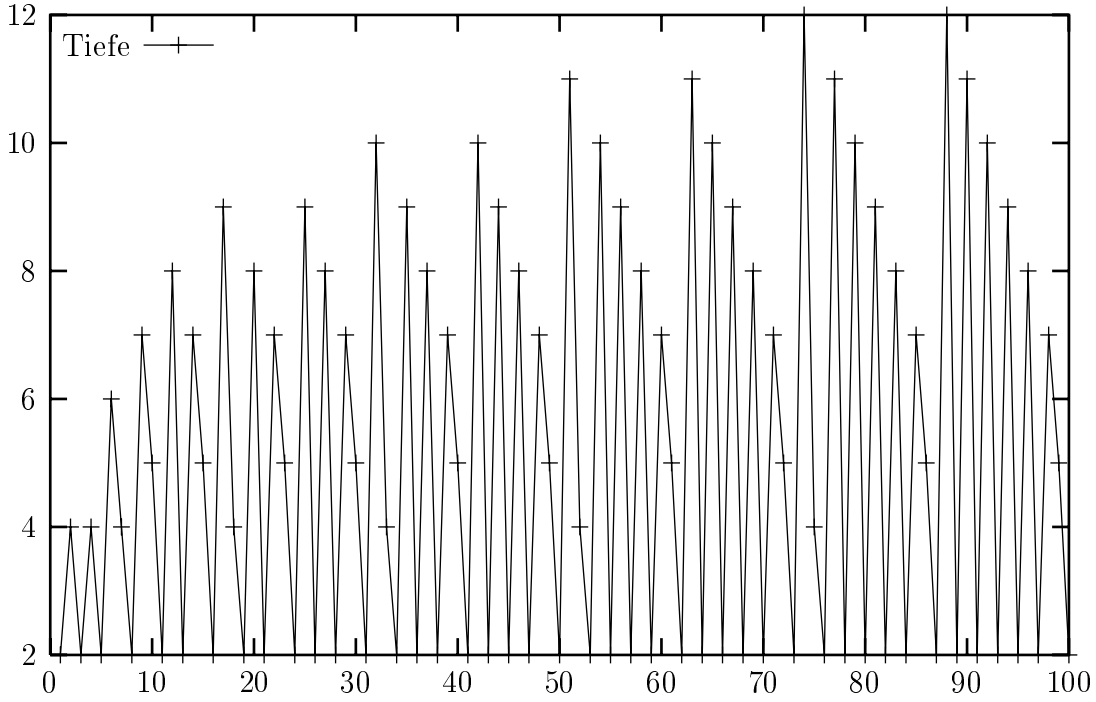


Abbildung B.2.: AAA, $\mathrm{rs}_{\mathrm{LA}}()$, Schrittweite 1, $\mathrm{depth}_{\mathrm{L}}(r_i)$

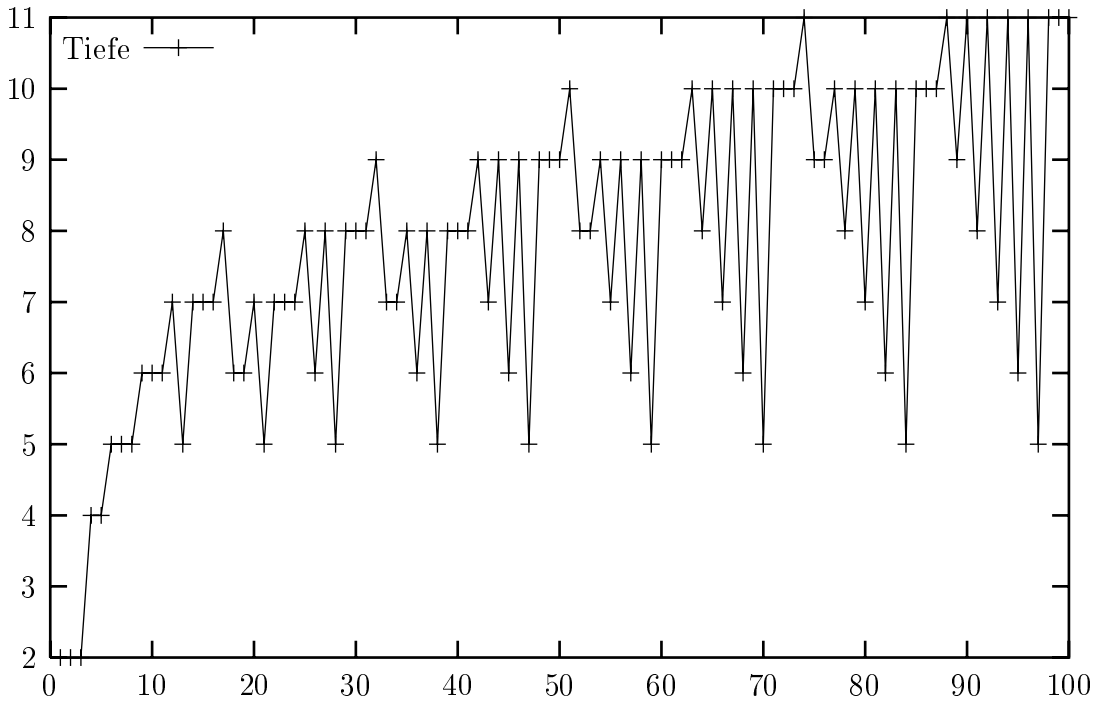


Abbildung B.3.: AAA, $\mathrm{rs}_{\mathrm{LA}}()$, Schrittweite 1, $\mathrm{depth}_{\mathrm{R}}(r_i)$

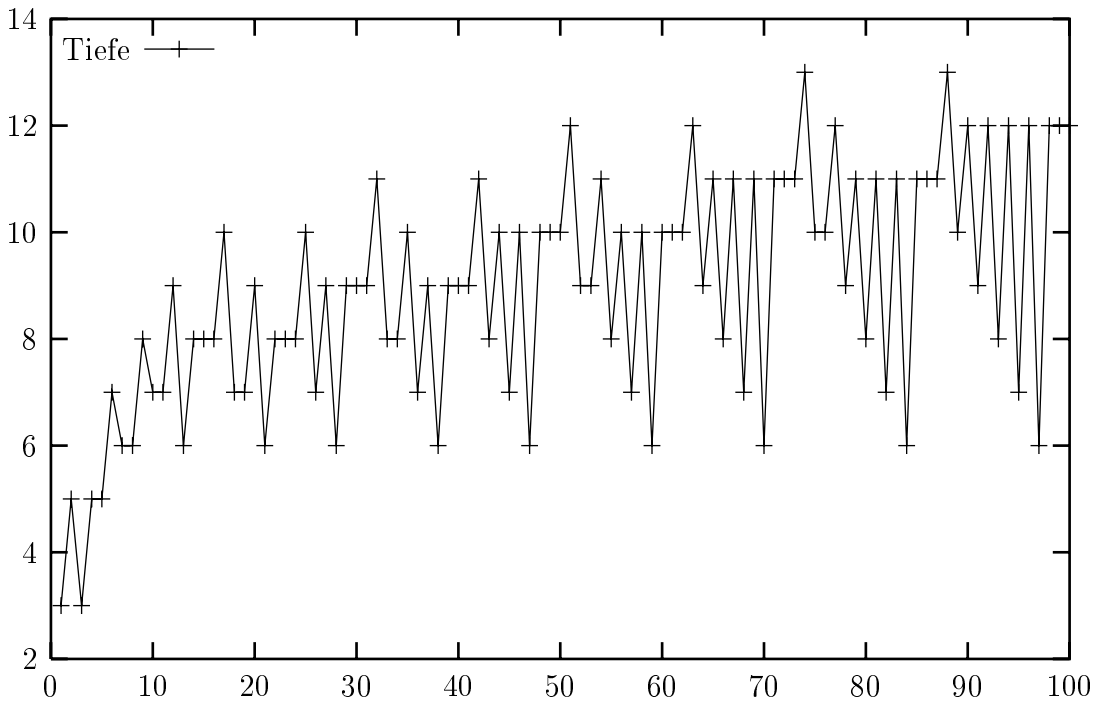


Abbildung B.4.: AAA, $\mathrm{rs}_{\mathrm{LA}}()$, Schrittweite 1, $\mathrm{depth}(r_i)$

Das zum Term AAA gehörige Muster ist $\mathsf{Y} := (\mathsf{APrev}, \mathsf{SA}(\mathsf{SAA}))$ und die Tiefe der Basis ist somit fünf.

In Abbildung B.2 wird allerdings der Wert zwei sehr oft erreicht. Dies liegt daran, daß die Redexfolge nicht nur aus paarweisen Musterinstanzen besteht, sondern immer wieder Zwischenschritte ausgeführt werden müssen. So ist jede Musterinstanz Y^i mit $i > 0$ nicht in Normalform. Der linke Teil des entsprechenden Redexes ist A, wodurch sich das häufige Auftreten der Zwei erklären läßt. Die absteigenden lokalen Maxima sind bei $\mathrm{depth}(\mathsf{SY}^i(\mathsf{SY}^i))$ und somit proportional zur Tiefe von Y^i.

Daß es jede Serie fallender Extrema zweimal zu geben scheint, liegt daran, daß für den Beweis der Existenz der zugehörigen unendlichen Reduktionskette ein Nulldurchgang benötigt wird. Außerdem unterscheidet sich die erste von der jeweils zweiten Serie durch eine zusätzliche vier.

Der unmittelbar vor vier liegende Wert wird übrigens durch einen Term erzeugt, der die nächsthöhere Instanz des Musters gar nicht beinhaltet. Erst die darauffolgenden zwei bzw. der nächste Tiefenwert beruhen auf der nächsthöheren Instanz.

Das Wachstum in Abbildung B.5 kann auf den ersten Blick linear aussehen. Erhöht man jedoch die Schrittzahl, zeigt sich, daß die einhüllende Funktion nicht linear, sondern schwächer wächst.

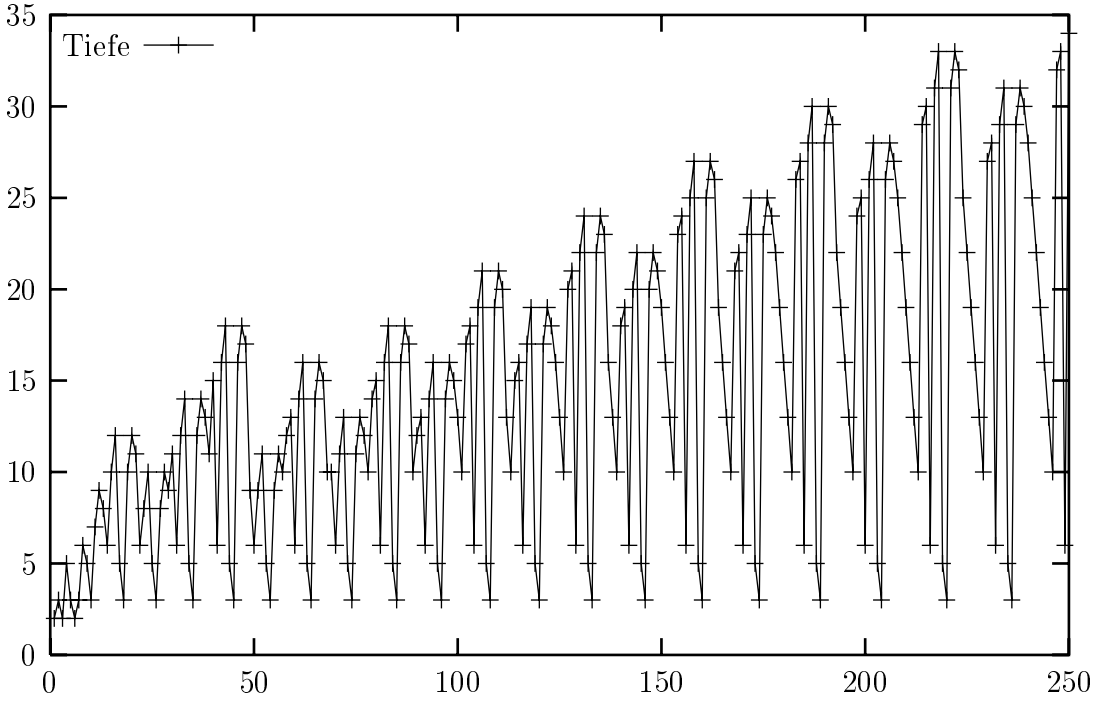


Abbildung B.5.: ATSS, $\mathrm{rs}_{\mathrm{LA}}()$, Schrittweite 1, $\mathrm{depth}_{\mathrm{L}}(r_i)$

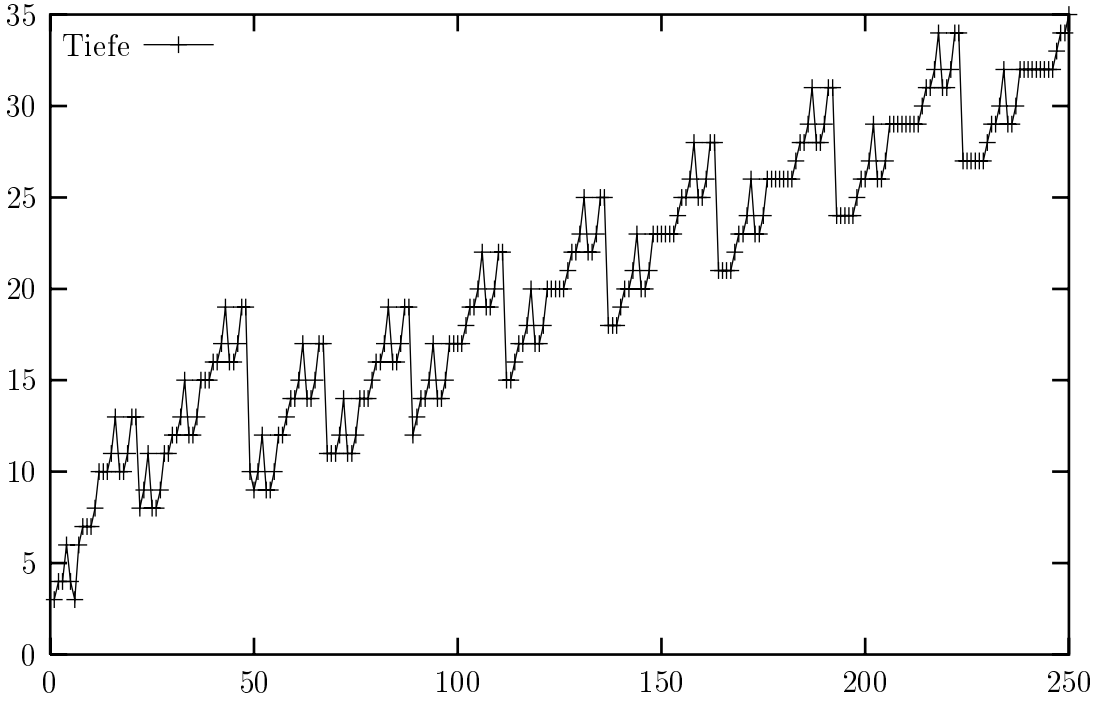


Abbildung B.6.: ATSS, $\mathrm{rs}_{\mathrm{LA}}()$, Schrittweite 1, $\mathrm{depth}_{\mathrm{R}}(r_i)$

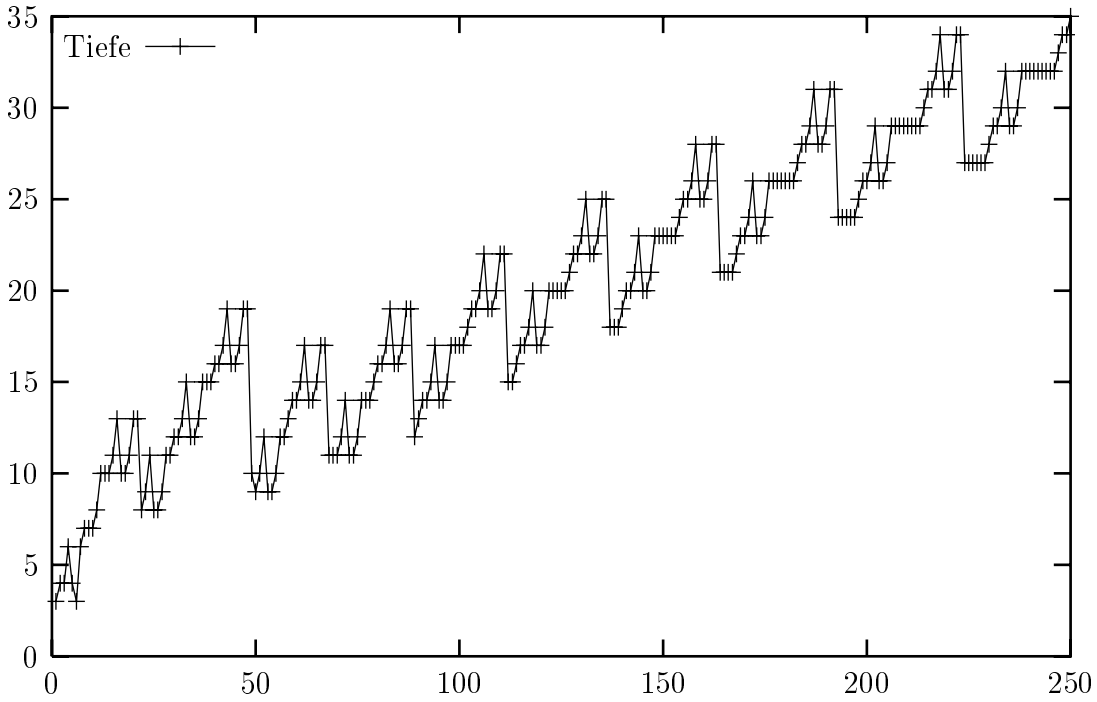


Abbildung B.7.: ATSS, $\mathrm{rs}_{\mathrm{LA}}()$, Schrittweite 1, $\mathrm{depth}(r_i)$

Anhang C

Dokumentation zu `CLS`

Das Programmpaket `CLS` besteht aus den zwei Programmen `cls` und `ana`. Nachfolgend findet sich eine Übersicht über die Einsatzmöglichkeiten beider Programme. Allerdings läßt dieses Kapitel sicherlich manches Detail vermissen, da `CLS` in den letzten Wochen verschiedene Metamorphosen durchlaufen mußte.

C.1. `cls`

`cls` ermöglicht den Umgang mit Termen, unendlichen Reduktionsketten, Mustern sowie den entsprechenden Beweisen. Es kennt vier Hauptmodi:

match: In jedem Eingabeterm wird nach allen angegebenen Mustern gesucht und entsprechende Listen mit Matches werden ausgegeben.

find: In jedem Eingabeterm wird nach Mustern gesucht, die eine unendliche Reduktionskette verursachen (können). Zudem wird anschließend – je nach Wert der Option `--veristrat` – für jedes gefundene Muster versucht zu beweisen, daß es zum gegebenen Startterm tatsächlich zu einer unendlichen Reduktionskette führt.

Da u.U. pro Programmstart mehrere Suchläufe möglich sind, zählt `cls` mit, für welchen Term bereits ein Muster nebst Beweis gefunden wurde. Diese Terme werden bei folgenden Suchläufen nicht weiter berücksichtigt, auch wenn die Muster zu keinem Beweis führen.

verify: Für jede Kombination aus Term und Muster wird ein Beweis probiert, ob das jeweilige Muster eine unendliche Reduktionskette verursacht.

hasnf: Dieser Teil wurde *nicht* von mir geschrieben, sondern von Johannes Waldmann im Rahmen seiner Dissertation [Wal97].

Gibt zu jedem Eingabeterm einen Wahrheitswert aus, ob der Term eine Normalform hat, oder nicht.

Daneben gibt es noch eine ganze Reihe von Optionen, mit deren Hilfe das Verhalten eines oder mehrerer Modi beeinflußt werden kann:

- Alle mit `--exp` beginnenden Schalter dienen dazu, die Ausdrücke anzugeben, die untersucht werden sollen. `--expFromFile` nimmt an, daß in der einzulesenden Datei pro Zeile ein Ausdruck steht. Leer– und mit '#' beginnende Zeilen werden ignoriert. `--expEnumByDepth` zählt alle Terme aus $\mathbf{CL}(\mathbf{S})$ auf, deren Tiefe im angegebenen Intervall liegt. `-expEnumZachos` zählt alle Terme aus $\mathbf{CL}(\mathbf{S})$ auf, deren Länge im angegebenen Intervall liegt, mit der Einschränkung, daß die Terme keine Normalform haben.

- Mit `--pat` beginnende Schalter dienen dazu, die Muster anzugeben, die untersucht werden sollen. Sie werden jedoch nur in den Modi `--match` und `--verify` berücksichtigt.

- Die Schalter `--bas`$*$ spezifieren die Basen für den Modus `--find`. Dies macht aber nur Sinn, wenn als Suchstrategie (mittels `--searchstrat`) `BUUserBases` gewählt wurde.

- `--maxred` gibt die Obergrenze vor, die für alle Reduktionen gilt. Größere Werte erhöhen natürlich die Rechenzeit, aber auch die Chance, Muster zu finden. Der Defaultwert ist tendenziell zu niedrig eingestellt.

- Wenn `--steps` > 1 ist, wird für den Modus `--find` die maximale Zahl von Reduktionen erst nach der angegebenen Zahl von Schritten erreicht. Z.B. werden mit `--maxred=150` und `steps=3` erst 50, dann 100 und zum Schluß 150 Schritte als Obergrenze festgelegt. Dies ist vor allem deshalb sinnvoll, weil sich die meisten Muster schon mit einer relativ kleinen Schrittzahl finden lassen und die Suchfunktionen immer bis zur maximalen Schrittzahl reduzieren.

- `--redstrat` und `--searchstrat` dienen dazu, die jeweilige Strategie festzulegen, was natürlich nicht für jeden Modus sinnvoll ist.

- `--veristrat` wählt ebenfalls die entsprechende Strategie. Werden ‚`w`'eak und ‚`s`'trong gleichzeitig angegeben, dann wird ein Beweis als gut bewertet, wenn eine der beiden Verifikationsmethoden (`verifyPatternTDDynamic` oder `verifyPatternStrong`) erfolgreich war.

- `--alsotryrmim` ist ebenfalls nur für den Modus `--find` relevant. Ist als Reduktionsstrategie `LeftMostOuterMost` gewählt, so bestimmt dieser Parameter, ob zusätzlich auch `RightMostInnerMost` versucht werden soll.

- `--maxcontextdepth` und `--maxdissectdepth` sind für den Modus `--find`, wenn als Suchstrategie eine bottom-up-Strategie gewählt wurde.

- `--individual` schließlich sorgt dafür, daß pro bearbeitetem Ausdruck eine eigene Datei angelegt wird, in die die Ergebnisse geschrieben werden. Der Dateiname wird aus dem Ausdruck gebildet.

C.2. ana

`ana` wurde im wesentlichen programmiert, um allgemeinste Muster nach Algorithmus 5.6.13 zu finden; quasi nebenbei erledigt es aber auch alle Auswertungs- und Statistikarbeiten.

In jedem Fall erwartet es als Eingabedatei(en) die Ausgabe von `cls`, die mittels `--output` oder `--individual` in eine Datei umgeleitet werden kann.

Wie auch bei `cls` wird die Arbeitsweise von `ana` durch Modi gesteuert:

statistic sortiert die Eingabeterme der Länge nach und gibt aus, wie viele Terme insgesamt bearbeitet wurden, für wie viele ein Muster mit gültigem Beweis, für wie viele zwei Muster mit gültigem Beweis bzw. für welche Terme kein Muster mit gültigem Beweis gefunden werden konnte etc. Für genauere Erläuterungen sei auf Abschnitt 5.8 verwiesen.

filter filtert die Resultate nach dem Typen `Found` und gibt jeweils die zugehörigen Terme aus (s. auch Abschnitt 6.5).

mgp sucht für jede Eingabe (für jedes TBMK) das allgemeinste Muster. Der Output erfolgt sortiert nach allgemeinsten Mustern mit einer Liste von Termen, für die das allgemeinste Muster hilft, eine unendliche Reduktionskette zu beweisen.

list gibt die Ergebnisse von `cls` formatiert aus. Allerdings werden nur die jeweils besten Resultate für jeden Ausdruck genommen. Für den Fall, daß es mehrere gleich gute Resultate gibt, werden diese alle ausgegeben (s. z.B. Tabelle A.1).

Neben den Modi gibt es auch für `ana` weitere Optionen:

- Mit Hilfe der Option `--format` kann Einfluß auf das Ausgabeformat (normal oder LaTeX) genommen werden, was insbesondere für Tabellen sehr nützlich ist.
- Alle mit `--f` beginnenden Optionen sind für die Beeinflussung des Filters gedacht (etwa, um nur Resultate von Termen einer bestimmten Länge zu berücksichtigen). Wird eine Filteroption mehrfach angegeben, so werden die entsprechenden Filter mit einem logischen Oder verknüft. Außerdem ist als erstes Zeichen eines Filters ein `!` zulässig, das den entsprechenden Filter invertiert.
- `--assumeweaktrue` nimmt an, daß die Verifikation des Beweises durch `verifyPatternTDDynamic` erfolgreich verlaufen ist, da der entsprechende Output z. Z. nicht gelesen wird. Die Voreinstellung von `cls` (einstellbar über `--veristrat`) ist entsprechend gewählt.

Anhang D
Danksagung

Danke, Danke, Danke an:

- meine Freundin Nicolle
- meinen Betreuer Johannes „Joe“ Waldmann, der mich perfekt betreut hat
- meine Mutter Sabine für gewissenhaftes Korrekturlesen
- meinen Vater Joachim fürs Stipendium
- die vielen Menschen, die die freie Software geschrieben haben, ohne die diese Arbeit gar nicht möglich gewesen wäre, z.B. Linus Torvalds und alle anderen Kernelhacker und -hackerinnen für ein prima Betriebssystem (`http://www.kernel.org`), das XEmacs-Entwicklerteam für einen tollen Editor[1] (`http://www.xemacs.org`), die Free Software Foundation (`http://www.fsf.org`) für viele der benutzten Programme, das GHC- und Hugs-Team für ihre Haskell-Interpreter bzw. Compiler (`http://www.haskell.org/[ghc|hugs]`) und natürlich Leslie Lamport für LaTeX und Donald E. Knuth für TeX
- Volker Ahlers und Stefan Kamphausen für das Dateigrundgerüst dieser Arbeit und nützliche Tipps zu XEmacs und AUCTeX (`http://www.skamphausen.de`)

[1] dem ich als Teetrinker gerne nachsehe, daß er immer noch nicht Kaffee kochen kann

Anhang E
Selbständigkeitserklärung

Ich versichere, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, den 10. Juni 2002

(Till Dörges)