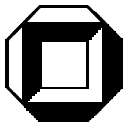
Normalisierung, Unifikation und Antiunifikation in freien Monoiden

Armin Biere
Diplomarbeit



Universität Karlsruhe (TH)
Fakultät für Informatik

Betreuer:

Prof. Dr. Stephan Jähnichen
Dipl. Inf. Birgit Heinz

September 1993

Zusammenfassung

Die Arbeit befaßt sich mit der Untersuchung der Termordnung in freien Monoiden. Oder anders ausgedrückt um Wörter über einem bestimmten Alphabet angereichert mit Variablen. Es werden drei Ergebnisse über Normalisierung, Matching, Generalisierung und Unifikation in dieser Algebra vorgestellt. Aber es stehen die Themen Normalisierung und Generalisierung im Vordergrund.

Der vorgestellte Unifikationsalgorithmus ist eine Variation des Verfahrens von Plotkin zur Unifikation unter Gültigkeit des Assoziativgesetzes ([Plot72]) und bleibt so im Grunde unkonstruktiv. Er dient aber auch nur als Hilfsmittel und Motivation für die konstrukiven Algorithmen zur Generalisierung und dem Matching, die im Laufe dieser Arbeit entwickelt wurden.

Die hier erörterten Normalisierungsverfahren gehen über eine Klammernormalform hinaus und erlauben die Entscheidung der Gleichheit von Wörtern mit Variablen, indem sie eine bis auf Variablenumbenennung eindeutige Normalform minimaler Länge erzeugen.

Soweit es sich bei den vorgestellten Sätzen um konstruktive Existenzaussagen handelt, wird auch auf entsprechende Algorithmen und deren Komplexität eingegangen.

Ich versichere, die Arbeit selbständig ohne fremde Hilfe und nur unter Bezugnahme der aufgeführten Quellen angefertigt zu haben.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Arbeit beschäftigt sich mit einem dem Informatiker und Mathematiker wohlvertrauten Gebiet; mit Wörtern über einem festen Alphabet. Diese fundamentale Datenstruktur der freien Monoide spielt eine wichtige Rolle in weiten Gebieten der theoretischen Informatik und der Mathematik. Insbesondere ist sie die Grundlage für die Definition einer Sprache im Gebiet der formalen Sprachen und schließlich ist jeder Text, syntaktisch, gesehen ein Wort über einem endlichen Alphabet.

Was uns hier besonders interessiert, sind aber nicht nur die reinen Wörter, von denen man in anderem Zusammenhang auch als Strings oder Zeichenketten spricht, sondern es soll zusätzlich auch erlaubt sein, bei der Bildung von Wörtern Variablen zu benutzen. Dabei bedeutet z.B. „xax“: „Am Anfang und am Ende steht dasselbe und in der Mitte ein ‚a‘“.

Auf solchermaßen ausgestattete Wörter lassen sich Variablensubstitutionen anwenden, und man erhält eine Art Subsumptionsordnung. Die Suche nach Infima und Suprema bezüglich dieser Ordnung ist gleichbedeutend mit Anti-Unifikation und Unifikation.

Diese Untersuchungen haben „*per se*“ eine direkte Anwendung in Termersetzungssystemen, bei denen man die Assoziativität und die Existenz eines neutralen Elementes nicht als Regeln formulieren will, sondern sie aus dem Ersetzungsmechanismus herausnimmt und stattdessen A1–Unifikation und A1–Matching braucht.[1] Analoges gilt für Beweiser, die in einer assoziativen Theorie mit neutralem Element rechnen.

Im anderen Teil der Diplomarbeit geht es um die semantische Struktur der Wörter selbst. So sind z.B. die Wörter „$xyxy$“ und „zz“ semantisch gesehen gleich.[2] Die Aufgabe einer Normalisierung ist es nun, unter jeweils gleichen Wörtern eines stellvertretend für alle auszuwählen. Meistens, so auch hier, besteht der Normalisierungsvorgang nicht aus einem Schritt. Vielmehr gelangt man durch schrittweise Vereinfachung vom Ausgangswort zu immer einfacheren Wörtern, um schließlich bei der Normalform anzugelangen. Derartige Simplifikationskalküle werden für Wörter mit Variablen betrachtet und führen zu einem Entscheidungsverfahren für die Gleichheit von Wörtern.

Am Anfang der Diplomarbeit stand die Suche nach einer Existenzaussage oder einem Algorithmus für Generalisierungen von Termen modulo der speziellen Theorie der freien Monoide.

Das Pendant zu diesem Problem, die Unifikation in Halbgruppen, erwies sich als äußerst komplex, und der bekannte Lösungsansatz (siehe [Maka77] und [Jaff90]) von Makanin ließ sich nicht ohne weiteres auf dieses Problem übertragen. Auf der Suche nach einem einfacheren Zugang zur Unifikation erwies sich die Beschäftigung mit dem Problem der größten gemeinsamen Zeichenfolge zweier

[1] A1 ist eine Abkürzung für die Gültigkeit des Assoziativitätsgesetzes und der Existenz eines neutralen Elementes. Siehe zum Beispiel [Siek89] für eine allgemeine Nomenklatur.

[2] Um diese Aussage zu verifizieren betrachte man die Substitution $\sigma := \{x/\epsilon,\ y/z,\ z/xy\}$ („ϵ“ bezeichne das leere Wort), die wechselseitig das eine Wort in das andere überführt.

Wörter als sehr fruchtbar. Eine Übertragung des Lösungsverfahrens auf Wörter mit Variablen lieferte dann ein Semi-Entscheidungsverfahren für die Unifikation. Im Nachhinein stellte sich dieser Ansatz als eine Variation des Algorithmus von Plotkin zur Unifikation mit einem assoziativen Funktor heraus (siehe [Plot72]). Demgegenüber arbeitet das hier vorgestellte Verfahren nur mit einem binären assoziativen Funktor, bezieht aber zusätzlich die Existenz eines neutralen Elementes mit ein.

Durch eine Spezialisierung des Unifikationsalgorithmus, indem das Matching zweier Terme als einseitige Unifikation betrachtet wird, ließ sich dann eine Entscheidungsprozedur für das Matchingproblem angeben. Damit war der Grundstein gelegt für einen Generalisierungsalgorithmus. Es fehlte nur noch ein Verfahren, alle relevanten Wörter, von denen ein gegebenes Wort eine Instanz ist, aufzählen zu können. Ein einfaches kombinatorisches Lemma lieferte dann das fehlende Argument.

Während der Beschäftigung mit Unifikation und Generalisierung entstand die Frage nach einer Normalform, denn syntaktisch unterschiedliche Wörter können dieselbe semantische Bedeutung haben.

Der Weg führte dann von einer weniger konstruktiven, mehr auf semantischer Ebene arbeitenden Normalisierung über die Charakterisierung von Fixpunkten von Variablensubstitutionen zu einer rein syntaktischen Vereinfachungsvorschrift. Beide Simplifikationsalgorithmen stimmen insofern überein, daß sie dieselbe Normalform erzeugen.

1.1 Übersicht

Das erste Kapitel liefert eine exakte Definition von Wörtern mit Variablen und darauf aufbauend einige kleinere Sätze. Das nächste Kapitel ist der Normalisierung gewidmet und ist relativ unabhängig von den folgenden. Danach folgen zuerst einige Betrachtungen zu Wörtern ohne Variablen und der längsten gemeinsamen Zeichenfolge, worauf aufeinander aufbauend die Unifikation, das Matching und schließlich die Generalisierung behandelt werden.

Wörter mit oder ohne Variablen werden mit l, r, s, u, v, w gekennzeichnet. $a, b, c, \ldots$ stehen normalerweise für Konstanten, wogegen x, y, z Variablen bezeichnen und Variablensubstitutionen mit kleinen griechische Buchstaben λ, δ, σ geschrieben werden.

Kapitel 2

Grundlegende Definitionen

Als erstes muß wie allgemein üblich der Rahmen für die weiteren Untersuchungen abgesteckt werden. Es folgt eine Beschreibung des Diskursbereichs, dem sich einige später benötigte unmittelbare Folgerungen anschließen.

In algebraischer Sprechweise bilden „Wörter“ und die Konkatenation ein freies Monoid[1].

Dies ist die kleinste Algebra über einem festen Alphabet mit einem binaren Operator „$\cdot$“ und einer ausgezeichneten Konstanten ϵ, die folgenden Gleichungen genügt:

$$\begin{array}{lrcll} \forall u,v,w: & (u\cdot v)\cdot w & = & u\cdot(v\cdot w) & (\quad \text{Assoziativität} \quad) \\ \forall u: & u\cdot\epsilon & = & \epsilon\cdot u & \\ & & = & u & (\quad \text{Neutrales Element} \quad) \end{array}$$

Um die Entscheidbarkeit dieser Gleichheit zu zeigen, wird ein Termersetzungssystem [2] eingesetzt:

Definition 2.1

$\mathsf{Mon} = \mathsf{Mon}(\mathcal{V},\mathcal{S})$ sei die Menge der Terme, aufgebaut aus dem einzigen Funktionssymbol „$\cdot$“ (zweistellig), den Konstanten $\{\epsilon\}\cup\mathcal{S}$ und den Variablen $\mathcal{V}$, wobei $\mathcal{V}$ als mindestens abzählbar unendlich vorausgesetzt wird. $\mathsf{T}_{\mathsf{Mon}} = \mathsf{T}_{\mathsf{Mon}(\mathcal{V},\mathcal{S})}$ sei das Termersetzungssystem über Mon mit folgenden Ersetzungsregeln:

$$(x\cdot y)\cdot z \quad ::= \quad x\cdot(y\cdot z) \tag{2.1}$$
$$x\cdot\epsilon \quad ::= \quad x \tag{2.2}$$
$$\epsilon\cdot x \quad ::= \quad x \tag{2.3}$$

wobei $x,y,z\in\mathcal{V}$.

Wie gleich gezeigt wird, ist $\mathsf{T}_{\mathsf{Mon}}$ vollständig. Dadurch wird eine Normalform definiert, die aber nur eine Klammernormalform mit Weglassen von „ϵ“ darstellt. Deshalb sei hier ausdrücklich darauf hingewiesen, daß diese Normalform noch nicht ausreicht, um die Gleichheit von Wörtern mit Variablen zu entscheiden.[3] Sie sollte deshalb auch nicht mit der im nächsten Kapitel definierten Normalform verwechselt werden.

[1] Eine algebraische Definition eines freien Monoides findet man zum Beispiel in [Jac185] auf Seite 68. Dort wird auch die Eindeutigkeit bei festem Alphabet — bei Jacobson sind das die *generators* — gezeigt.

[2] Bezüglich Termersetzungsysteme sei auf [DeJo90] verwiesen.

[3] Es muß natürlich noch definiert werden, was unter Gleichheit von Wörten zu verstehen ist. Aber mit der bisherigen Definition einer Normalform sind die Wörter x, y und xy alle paarweise verschieden, obwohl sie eigentlich alle dasselbe Wort beschreiben.

Lemma 2.2 $\mathsf{T}_{\mathsf{Mon}}$ ist vollständig.

Beweis: Mit der Abbildung n von der Termalgebra in die natürlichen Zahlen und der dadurch induzierten Reduktionsordnung erweist sich das System als noethersch:

$$\begin{array}{rcll} n(a) & := & 1 & \text{für alle Konstanten(inklusive } \epsilon) \\ n(u \cdot v) & := & 2 \cdot n(u) + n(v) & \text{für alle Terme } u, v \in \mathsf{Mon} \end{array}$$

Denn für alle $x, y, z \in \mathsf{Mon}$ gilt:

1. $n(x) > 0$.
2. $n((x \cdot y) \cdot z) \overset{\text{Def}}{=} 2n(x \cdot y) + n(z) \overset{\text{Def}}{=} 4n(x) + 2n(y) + n(z)$
   $\overset{\text{mit 1.}}{>} 2n(x) + 2n(y) + n(z) \overset{\text{Def}}{=} 2n(x) + n(y \cdot z)$
   $\overset{\text{Def}}{=} n(x \cdot (y \cdot z))$.
3. $n(x \cdot \epsilon) \overset{\text{Def}}{=} 2n(x) + n(\epsilon) \overset{\text{Def}}{=} 2n(x) + 1 > n(x)$.
4. $n(\epsilon \cdot x) \overset{\text{Def}}{=} 2n(\epsilon) + n(x) \overset{\text{Def}}{=} 2 + n(x) > n(x)$.

Nun ist noch die Konfluenz zu zeigen, wobei hier wegen der Noetherschheit die lokale Konfluenz genügt. Die Betrachtung der kritischen Paare führt zu:

(1) mit (1):

$$((a \cdot b) \cdot c) \cdot d \left\{ \begin{array}{c} \overset{(1)\ \text{innen}}{::=} \\ \overset{(1)\ \text{außen}}{::=} \end{array} \right. \begin{array}{c} (a \cdot (b \cdot c)) \cdot d \\ \end{array} \begin{array}{l} \overset{(1)}{::=} \\ \end{array} \begin{array}{c} a \cdot ((b \cdot c) \cdot d) \overset{(1)}{::=} \\ (a \cdot b) \cdot (c \cdot d) \overset{(1)}{::=} \end{array} \left. \right\} a \cdot (b \cdot (c \cdot d))$$

(1) mit (2):

$$(a \cdot b) \cdot \epsilon \left\{ \begin{array}{ccc} & \overset{(2)}{::=} & \\ \overset{(1)}{::=} & a \cdot (b \cdot \epsilon) & \overset{(2)}{::=} \end{array} \right\} a \cdot b$$

(1) mit (2):

$$(a \cdot \epsilon) \cdot c \left\{ \begin{array}{ccc} & \overset{(2)}{::=} & \\ \overset{(1)}{::=} & a \cdot (\epsilon \cdot c) & \overset{(3)}{::=} \end{array} \right\} a \cdot c$$

(1) mit (3):

$$(\epsilon \cdot b) \cdot c \left\{ \begin{array}{ccc} & \overset{(3)}{::=} & \\ \overset{(1)}{::=} & \epsilon \cdot (b \cdot c) & \overset{(3)}{::=} \end{array} \right\} b \cdot c$$

(2) mit (3):

$$\epsilon \cdot \epsilon \overset{(2)\ \text{oder}\ (3)}{::=} \epsilon$$

Weitere nichttriviale Überlagerungen oder kritische Paare gibt es nicht. ■

Definition 2.3

Die durch $\mathsf{T}_{\mathsf{Mon}(\mathcal{V},\mathcal{S})}$ erzeugte Gleichheit sei mit $\equiv$ bezeichnet.

Um den Notationsaufwand nicht zu sehr anwachsen zu lassen, wird die Notation aus dem Bereich der regulären Ausdrücke übernommen, deren Syntax sich im Anhang A.1 findet. Insbesondere wird die Konkatenation zweier Wörter nicht mehr mit „·“ bezeichnet. Der Einfachheit halber werden dann die Wörter hintereinander geschrieben. Also uv statt $u \cdot v$.

Definition 2.4

Zu einer beliebigen Menge von Konstantenzeichen $\mathcal{S}$ und einer unendlichen Menge von Variablen $\mathcal{V}$, mit $\mathcal{V} \cap \mathcal{S} = \emptyset$, definiere $\Lambda = \Lambda(\mathcal{S}, V) := (\mathcal{S} \cup \mathcal{V})^*$

Mit dem vorigen erhält man also $\Lambda = \mathsf{Mon}(\mathcal{V}, \mathcal{S})/\equiv$. Für diese Wörter werden nun einige einfache Funktionen und Relationen definiert:

Definition 2.5

1. Für alle $u, v \in \Lambda$:
   (a) $|u| :=$ „*Länge von* u“.
   (b) $u \prec v \ :\Leftrightarrow\ |u| \leq |v|$.
   (c) $u \sqsubseteq v \ :\Leftrightarrow\ \exists\, l, r \in \Lambda\colon\ lur \equiv v$.
2. Für alle $u \in \Lambda$ sei
   (a) $\mathcal{V}_u := \{x \in \mathcal{V} \mid \exists\, l, r \in \Lambda\colon\ lxr \equiv u\}$.
   (b) $\mathcal{S}_u := \{s \in \mathcal{S} \mid \exists\, l, r \in \Lambda\colon\ lsr \equiv u\}$.
3. Die Anzahl der Vorkommen $\#(z, w)$ eines Alphabetzeichens $z \in \mathcal{V} \cup \mathcal{S}$ in einem Wort $w \in \Lambda$, mit $\#\colon (\mathcal{V} \cup \mathcal{S}) \times \Lambda \to \mathbb{N}$, wird rekursiv definiert als:
   (a) $\#(z, \epsilon) \ := \ 0$.
   (b) $\#(z, yv) := \ \#(z, v)$, für $y \in \mathcal{V} \cup \mathcal{S} \backslash \{z\}$.
   (c) $\#(z, zv) := \ \#(z, v) + 1$.

Besteht zwischen zwei Wörtern $u, v \in \Lambda$ der Zusammenhang $u \sqsubseteq v$, so bedeutet dies, daß u ein *Teilwort* von v ist. Eine andere Bezeichnung dafür ist *Suffix*. Dieser Begriff ist wohl zu unterscheiden vom Begriff der *Teilfolge*, der in Kapitel 4 eingeführt wird. Ist ein Wort u Teilwort eines Wortes v, ensteht u aus v, indem man ein beliebiges Anfangs- und Endstück von v wegstreicht. Demgegenüber ist u eine Teilfolge von v genau dann, wenn u durch Streichung beliebig vieler (auch von keinem) Buchstaben aus v hervorgeht. Damit wird der Begriff der Teilfolge genau wie in der Analysis verwendet.

Bis jetzt haben die Variablen noch keine besondere Rolle gespielt. Auf dem Weg, ihre Bedeutung näher zu fassen, stößt man auf folgende informale Definition:[4]

Variablen repräsentieren eine unbekannte Größe.

Oder anders formuliert, man kann für eine Variable ein beliebiges Objekt des gegebenen Diskursbereiches einsetzen. Dieser Einsetzungsmechanismus entspricht auf formaler Seite dem Anwenden

[4] In der Analysis wird man eher auf die Defintion treffen, daß eine Variable eine Veränderliche ist.

von Substitutionen. Dies ist zum Beispiel auch der Blickwinkel, aus dem der Lambdakalkül Variablen sieht, und man wird in der Mathematik viele weitere Beispiele antreffen, die diese Sichtweise unterstützen.

In allen Gebieten, die Variablen so betrachten, besteht dann ein erster Schritt zur Klärung der Semantik von Variablen in der Einführung von Variablensubstitutionen, man denke z.B. an die Regel zur β-Reduktion im Lambdakalkül. Deren Einführung geht eine exakte Definition von Variablensubstitutionen voraus.

Definition 2.6 (Substitutionen und Subsumptionsordnung)

1. Für alle $\sigma : \mathcal{V} \to \Lambda$, definiere die Fortsetzung von σ auf Λ auf natürliche Weise durch:

   (a) $\sigma(\epsilon) := \epsilon$.
   (b) $\sigma(au) := a\sigma(u)$ für $a \in \mathcal{S}$.
   (c) $\sigma(au) := \sigma(a)\sigma(u)$ für $a \in \mathcal{V}$.

2. $\forall u, v \in \Lambda : u \leq v \ :\Leftrightarrow\ \exists\, \sigma : \mathcal{V} \to \Lambda$ mit $\sigma(u) \equiv v$.

3. $\forall u, v \in \Lambda : u = v \ :\Leftrightarrow\ \exists\, \sigma, \tau : \mathcal{V} \to \Lambda$ mit $\sigma(u) \equiv v$ und $\tau(v) \equiv u$.

Dieser zuletzt definierten Relation „$=$“ gilt das Hauptinteresse dieser Arbeit. Sie stellt quasi die semantische Gleichheit von Wörtern mit Variablen dar, der die mehr syntaktischere Gleichheit „$\equiv$“ gegenübersteht. Auf der anderen Seite ist „$=$“ die durch die nicht antisymmetrische Präordnung „$\leq$“ induzierte Äquivalenzrelation, wie Lemma 2.8 zeigt. So wird sich ein weiterer Schwerpunkt der Arbeit mit dem Studium von „$\leq$“ beschäftigen.

Der Begriff Gleichheit ist im Falle der semantischen Gleichheit mit Vorsicht zu genießen, weil sie die Kongruenzeigenschaft nicht erfüllt (siehe 2.8). Als Äquivalenzrelation stellt sie eine Erweiterung von „$\equiv$“ dar, und dies sollte sich auch in einer ähnlichen Namensgebung niederschlagen.[5]

Definition 2.7 (Semantische und Syntaktische Gleichheit)

Zwei Wörter u, v aus Λ heißen *syntaktisch gleich* genau dann, wenn $u \equiv v$. Gilt nur $u = v$ so heißen sie *semantisch gleich*.

Als Parallele hierzu betrachte man die klassische Definition des Termverbandes über einer festen Signatur Σ. Die syntaktische Gleichheit entspricht dort der Gleichheit in der freien Termalgebra über Σ. Die damit verträgliche semantische Gleichheit wird nun auch über Substitutionen definiert, entsprechend wie es hier geschehen ist. Man erhält auf genau die gleiche Weise folgende Aussage:

Lemma 2.8

$=$ ist eine Äquivalenz- aber keine Kongruenzrelation über Λ.

[5] Es mag verwunderlich sein, daß eine Äquivalenzrelation, die die Erweiterung (im Sinne einer Mengeninklusion) einer Kongruenzrelation darstellt, keine Kongruenzrelation zu sein braucht. Dies liegt an der logischen Struktur der Bedingung für die Kongruenzeigenschaft, nämlich $\forall u, v, r, s \in \Lambda : u =_{KR} v,\ r =_{KR} s \ \Rightarrow\ u \cdot r =_{KR} v \cdot s$. Im Falle einer Erweiterung kann nun die Prämisse öfters zutreffen und dabei die Konklusio falsch sein.

Beweis:

Reflexivität: Wähle $\tau := \sigma := \mathsf{id}$.

Symmetrie: Vertausche τ und σ.

Transitivität: Sei $u = v$ und $v = w$, dann gibt es Substitutionen $\sigma_1, \sigma_2, \tau_1, \tau_2 : \mathcal{V} \to \Lambda$, so daß

$$\sigma_1(u) \equiv v,\ \tau_1(v) \equiv u,\ \sigma_2(v) \equiv w \text{ und } \tau_2(w) \equiv v$$

Nun definiere $\sigma :\equiv \sigma_2 \circ \sigma_1$ und $\tau :\equiv \tau_1 \circ \tau_2$. Dann gilt $\sigma(u) \equiv \sigma_2(\sigma_1(u)) \equiv \sigma_2(v) \equiv w$ und $\tau(w) \equiv \tau_1(\tau_2(w)) \equiv \tau_1(v) \equiv u$, woraus sich $u = w$ ableiten läßt.

Keine Kongruenz: Es gilt $x = y$, aber nicht $x \cdot y = x \cdot x$.

■

Damit macht es Sinn, $\Lambda/\!=$ zu schreiben, und $\Lambda/=$ ist schließlich die mathematische Präzisierung dessen, was hier unter Wörtern mit Variablen verstanden werden soll:

Definition 2.9 Die Menge der *Wörter mit Variablen* sei definiert als $\Lambda/=$.

Mit obiger Bemerkung, daß „$=$“ die durch „$\leq$“ induzierte Äquivalenzrelation ist, hat man mit $\Lambda/=$ einen Bereich gefunden, auf dem „$\leq$“ zur partiellen Ordnung wird.

Hier sind noch einige vergleichende Worte zur leeren Theorie (oben klassischer Termverband genannt) angebracht. Hat man eine Gleichungstheorie E vorliegen (vergl. [Ba91] oder [Pott89]), so wird die Subsumptionsordnung auf Termen modulo E wie folgt definiert:

$$s \leq_E t \ :\Leftrightarrow\ \exists \sigma : \sigma(s) \equiv_E t \tag{2.4}$$

Dabei bedeute „$\equiv_E$“ die durch E induzierte Gleichheit über den Termen. Bei der klassischen Definition des Termverbandes spricht man von der leeren Theorie und setzt $E := \emptyset$.

Aus ordnungstheoretischer Sicht haben alle diese Subsumptionsordnungen den Nachteil, keine richtige partielle Ordnung, sondern nur Präordnungen zu sein. Sie sind nämlich nicht antisymmetrisch, denn schon im Falle der leeren Theorie gilt zwar $x \leq_\emptyset y$ und $y \leq_\emptyset x$ aber nicht $x \equiv_\emptyset y$. Um dieses Defizit zu umgehen, betrachtet man eine Erweiterung von „$\equiv_E$“, nämlich die durch „$\leq_E$“ induzierte Äquivalenzrelation:

$$s =_E t \ :\Leftrightarrow\ s \leq_E t,\ t \leq_E s \tag{2.5}$$

Dies bedeutet auf der Termseite einen Übergang zur Faktorstruktur modulo „$=_E$“. Mit anderen Worten, man betrachtet statt reinen Termen nun Äquivalenzklassen.

In der leeren Theorie besteht bekanntermaßen eine Äquivalenzklasse von „$=_\emptyset$“ aus allen Termen, die durch Variablenumbenennung auseinander hervorgehen.[6] Bei einer nicht leeren Theorie kommen zu diesen mindestens noch die mittels „$\equiv_E$“ vergleichbaren Terme hinzu. Aber leider ist das noch nicht alles. Es können zusätzlich noch mehr Wörter gleich sein, wie das Beispiel mit den Wörtern $xyxy$ und zz von Seite 5 zeigt.

In der leeren Theorie ist die Lösung von Unifikations- oder Generalisierungsproblemen immer nur Eindeutigkeit bis auf Variablenumbenennung. [7] Bei nicht leerem E möchte man also auch eine

[6] siehe [Schm92].

[7] Ohne diese Einschränkung wäre die Lösungsmenge eines Unifikationsproblems sogar im Falle der leeren Theorie im allgemeinen nicht endlich.

Normalform über den Termen haben, so daß die Lösungsmenge eines E-Unifikationsproblems bzw. E-Generalisierungsproblems nur aus normierten, verschiedenen Termen besteht (Zwei Terme s und t heißen verschieden, wenn $s \neq_E t$).

Für ein beliebiges E und insbesondere auch in der speziellen Theorie A1 kann eine solche Normalisierung, wie oben gezeigt, nur bis auf Variablenumbenennung eindeutig sein. Für E =A1 ist interessanterweise diese Bedingung zusammen mit der Forderung von minimaler Länge hinreichend für die Normiertheit eines Wortes mit Variablen. Dieses Resultat und Algorithmen für die Berechnung einer Normalform werden im nächsten Kapitel behandelt. Dafür sind aber erst ein paar Vorbereitungen notwendig:

Das erste Lemma stellt klar, daß die Fortsetzung einer Substitution von den Variablen auf ganz Λ homomorph bezüglich der Konkatenation ist.

Lemma 2.10 $\forall u, v \in \Lambda,\ \sigma : \mathcal{V} \to \Lambda :\ \sigma(uv) \equiv \sigma(u)\sigma(v).$

Beweis: Induktion über $n := |u|$.

$\underline{n = 0:}$ $|u| = 0$ ergibt $u \equiv \epsilon$, woraus folgt:

$$\begin{array}{lllll} \sigma(uv) & \equiv & \sigma(v) & \equiv & \epsilon\sigma(v) \\ & \equiv & \sigma(\epsilon)\sigma(v) & \equiv & \sigma(u)\sigma(v) \end{array}$$

$\underline{n \rightsquigarrow n+1:}$ $|u| = n + 1$. Dann ist $u \equiv zu'$ mit $z \in \mathcal{V} \cup \mathcal{S}$ und:

$$\begin{array}{lllll} \sigma(uv) & \equiv & \sigma(zu'v) & \overset{\text{Def.}}{\equiv} & \sigma(z)\sigma(u'v) \\ & \overset{\text{Ind.-Beh.}}{\equiv} & \sigma(z)\sigma(u')\sigma(v) & \overset{\text{Def.}}{\equiv} & \sigma(zu')\sigma(v) \\ & \equiv & \sigma(u)\sigma(v) & & \end{array}$$

■

Als nächstes steht die Teilwortbeziehung im Mittelpunkt. Später wird deren Antisymmetrie und die Eigenschaft, unter Anwendung von Substitutionen erhalten zu bleiben, gebraucht.

Lemma 2.11

1. $\sqsubseteq$ ist eine partielle Ordnung.
2. $\forall u, v \in \Lambda,\ \sigma : \mathcal{V} \to \Lambda :\ u \sqsubseteq v \implies \sigma(u) \sqsubseteq \sigma(v).$

Beweis von 2.11.1:

Reflexivität: wähle $l :\equiv r :\equiv \epsilon$.

Transitivität: Sei $u \sqsubseteq v$ und $v \sqsubseteq w$. Dann gibt es $l, l', r, r' \in \Lambda$, so daß $v \equiv lur$ und $w \equiv l'vr'$. Damit folgt sofort $w \equiv l'lurr'$, was $u \sqsubseteq w$ ergibt.

Antisymmetrie: Sei $u \sqsubseteq v$ und $v \sqsubseteq u$. Dann gibt es $l, l', r, r' \in \Lambda$, so daß $v \equiv lur$ und $u \equiv l'vr'$. Einsetzen ergibt $u \equiv l'lurr' \equiv \epsilon$, woraus $l \equiv l' \equiv r \equiv r'$ folgt. Somit ist $u \equiv v$. ■

Beweis von 2.11.2: Sei also $lur \equiv v$, dann folgt mit Lemma 2.10, daß $\sigma(v) \equiv \sigma(lur) \equiv \sigma(l)\sigma(u)\sigma(r)$, somit $\sigma(u) \sqsubseteq \sigma(v)$. ■

Für die Häufigkeit des Vorkommens eines Buchstabens in einem Wort gilt auch eine Homomorphieeigenschaft. Dann macht man noch die Beobachtung, daß nur Variablen durch die Anwendung einer Substitution aus einem Wort verschwinden können, sich also die Häufigkeit des Vorkommens einer Konstanten durch Anwendung einer Substitution nicht verringert.

Lemma 2.12

1. $\forall u, v \in \Lambda,\ z \in \mathcal{V} \cup \mathcal{S}$: $\#(z, uv) = \#(z, u) + \#(z, v)$
2. $\forall u, v \in \Lambda,\ z \in \mathcal{V} \cup \mathcal{S}$: $\#(z, v) \leq \#(z, uv)$
3. $\forall u \in \Lambda,\ \sigma: \mathcal{V} \to \Lambda,\ z \in \mathcal{S}$: $\#(z, u) \leq \#(z, \sigma(u))$

Beweis von von 2.12.1: Ersetze im Beweis von 2.10 „$\sigma($“ durch „$\#(z,$“ und die Konkatenation durch „$+$“. ■

Beweis von von 2.12.2: Folgt unmittelbar aus 2.12.1. ■

Beweis von von 2.12.3: Induktion über $n := |u|$.

$\underline{n = 0:}$ $\#(z, \underbrace{u}_{\equiv \epsilon}) \equiv \#(z, \epsilon) \equiv \#(z, \underbrace{\sigma(u)}_{\equiv \epsilon})$.

$\underline{n \rightsquigarrow n+1:}$ $|u| = n + 1$

(A) $u \equiv zu'$, so ist $\sigma(u) \equiv z\sigma(u')$ wegen $z \in \mathcal{S}$, und es gilt:

$$\begin{array}{lcll} \#(z, \sigma(u)) & = & \#(z, z\sigma(u')) & \overset{\text{Def.}}{=} \#(z, \sigma(u')) + 1 \\ & \overset{\text{Ind.-Beh.}}{\geq} & \#(z, w) + 1 & = \#(z, zu') \end{array}$$

(B) $u \equiv u_0 u',\ u_0 \in (\mathcal{V} \cup \mathcal{S}) \backslash \{z\}$, so folgt:

$$\begin{array}{lcll} \#(z, u) & = & \#(z, u_0 u') & \overset{u_0 \not\equiv z}{=} \#(z, u') \\ & \overset{\text{Ind.-Beh.}}{\leq} & \#(z, \sigma(u')) & \overset{2.12.2}{\leq} \#(z, \sigma(u_0)\sigma(u')) \\ & = & \#(z, \sigma(u_0 u')) & = \#(z, \sigma(u)) \end{array}$$

■

Durch den nächsten Satz wird die Beziehung von Wörtern, die semantisch gleich sind, ein wenig erhellt. Die Substitutionen, die das eine Wort auf das andere abbilden, können nämlich Variablen nur durch konstantenfreie Wörter substituieren.

Satz 2.13

Seien $u, v \in \Lambda,\ \sigma, \tau: \mathcal{V} \to \Lambda$ mit $\sigma(u) \equiv v$ und $\tau(v) \equiv u$, so gilt $\sigma, \tau : \mathcal{V} \to \mathcal{V}^*$.

Beweis: (indirekt) Angenommen, es gibt $l, r, s, t \in \Lambda,\ a \in \mathcal{S},\ x \in \mathcal{V}_u$, so daß $u \equiv lxr,\ \sigma(x) \equiv sat$. Setzt man nun $\delta := \tau \circ \sigma$, dann ist lxr ein Fixpunkt von δ ($\delta(lxr) \equiv lxr$) und es folgt

$$
\begin{array}{lcl}
\#(a, lxr) & \overset{x \not\equiv a}{=} & \#(a, lr) \\
& \overset{2.10}{=} & \#(a, \delta(l)\delta(r)) \\
& < & \#(a, \delta(l)) + 1 + \#(a, \delta(r)) \\
& = & \#(a, \delta(l)) + \#(a, a) + \#(a, \delta(r)) \\
& \overset{2.12.1}{=} & \#(a, \delta(l)) + \#(a, \tau(s)a\tau(t)) + \#(a, \delta(r)) \\
& \overset{2.10}{=} & \#(a, \delta(l)) + \#(a, \tau(sat)) + \#(a, \delta(r)) \\
& \overset{2.12.1}{=} & \#(a, \delta(l)\tau(sat)\delta(r)) \\
& = & \#(a, \delta(l)\delta(x)\delta(r)) \\
& = & \#(a, lxr),
\end{array}
\qquad
\begin{array}{lcl}
& \overset{2.12.3}{\leq} & \#(a, \delta(lr)) \\
& \overset{2.12.1}{=} & \#(a, \delta(l)) + \#(a, \delta(r)) \\
\\ \\ \\ \\ \\
& = & \#(a, \delta(l)\tau(\sigma(x))\delta(r)) \\
& \overset{2.12.1}{=} & \#(a, \delta(lxr))
\end{array}
$$

was offensichtlich zu einem Widerspruch führt. ■

Die erste Aussage des nächsten Satzes lautet, daß eine Substitution, die keine Variable eines bestimmten Wortes auf ϵ abbildet, die Länge des Wortes auch nicht verkürzt. Daran schließt sich sofort der dritte Teil an, der besagt, daß für zwei semantisch gleiche Wörter, von denen eines kürzer ist als das andere, jede Substitution, die das längere auf das kürzere abbildet, mindestens eine Variable durch ϵ ersetzen muß. Dieses Ergebnis wird im nächsten Kapitel eine erste einfache Normalisierungsvorschrift ergeben.

Die mittlere Aussage betrifft die Anzahl Variablen vor und nach der Anwendung einer Substitution. Wird nämlich keine Variable durch ein Wort, indem nicht mehr als eine Variable vorkommt, ersetzt, so kann auch die Anzahl Variablen im Bildwort nicht größer sein als die des ursprünglichen Wortes. Dies ist ein Hilfsmittel, um die Injektivität einer Variablensubstitution, die eine Variable wieder auf eine Variable abbildet, zu zeigen, bildet also ein Hauptargument, wenn es um Variablenumbenennungen geht.

Satz 2.14

1. $\forall u \in \Lambda,\ \sigma: \mathcal{V} \to \Lambda$ gilt: falls $\forall x \in \mathcal{V}_u : |\sigma(x)| \not\equiv \epsilon$ gilt, dann ist $|\sigma(u)| \geq |u|$.
2. $\forall u \in \Lambda,\ \sigma: \mathcal{V} \to \Lambda$ gilt: falls $\forall x \in \mathcal{V}_u : \left|\mathcal{V}_{\sigma(x)}\right| \leq 1$ gilt, dann ist $\left|\mathcal{V}_{\sigma(u)}\right| \leq |\mathcal{V}_u|$.
3. Sei $u, v \in \Lambda$ mit $u = v$, $|v| < |u|$ und $\sigma(u) \equiv v$, so ist $\sigma(x) \equiv \epsilon$ für ein $x \in \mathcal{V}_u$.

Beweis von von 2.14.1: Induktion über n:=$|u|$.

<u>$n = 0$:</u> $u \equiv \epsilon$, womit $|\sigma(u)| = |\sigma(\epsilon)| = |\epsilon| = |u|$.

<u>$n \rightsquigarrow n+1$:</u> $|u| = n + 1,\ u \equiv zu',\ c \in \mathcal{V} \cup \mathcal{S}$:

$$
\begin{array}{lcll}
|\sigma(u)| & = & |\sigma(zu')| & \overset{\text{Def.}}{=} |\sigma(z)\sigma(u')| \\
& = & |\sigma(z)| + |\sigma(u')| & \overset{\text{Vor.}}{\geq} 1 + |\sigma(u')| \\
& \overset{\text{Ind.-Beh.}}{\geq} & 1 + |u'| & = |zu'| \\
& = & |u| &
\end{array}
$$

■

Beweis von von 2.14.2: Der Beweis wird geführt über die Länge von u. Sei also $n := |u|$.

$\underline{n = 0:}$ $u \equiv \epsilon$, $\mathcal{V}_u = \mathcal{V}_{\sigma(u)} = \emptyset$.

$\underline{n \rightsquigarrow n+1:}$ Sei also nun $u \equiv cu'$, mit $u' \in \Lambda$, $c \in \mathcal{V} \cup \mathcal{S}$. Falls nun $z \in \mathcal{S} \cup \mathcal{V}_{u'}$ gilt, dann ist $\mathcal{V}_u = \mathcal{V}_{u'}$ und $\mathcal{V}_{\sigma(u)} = \mathcal{V}_{\sigma(u')}$. Die Induktionsbehauptung auf u' angewendet liefert nun die Behauptung. Im anderen Fall liegt c in $\mathcal{V} \backslash \mathcal{V}_{u'}$ und $\mathcal{V}_u = \{c\} \dot{\cup} \mathcal{V}_{u'}$. [8] Somit:

$$\begin{aligned} \left|\mathcal{V}_{\sigma(u)}\right| &= \left|\mathcal{V}_{\sigma(cu')}\right| &&= \left|\mathcal{V}_{\sigma(c)} \cup \mathcal{V}_{\sigma(u')}\right| \\ &\leq \underbrace{\left|\mathcal{V}_{\sigma(c)}\right|}_{\geq 1} + \left|\mathcal{V}_{\sigma(u')}\right| &&\overset{\text{Ind.-Beh.}}{\leq} 1 + |\mathcal{V}_{u'}| \\ &= |\{c\}| + |\mathcal{V}_{u'}| &&\overset{\text{Vor.}}{=} |\mathcal{V}_u| \,. \end{aligned}$$

■

Beweis von von 2.14.3: Gilt für alle $x \in \mathcal{V}_u$, daß $\sigma(x) \not\equiv \epsilon$ ist, so ist nach 2.14.1 auch $|v| = |\sigma(u)| \geq |u|$, im Widerspruch zur Voraussetzung. ■ 

[8] $\dot{\cup}$ ist die disjunkte Vereinigung zweier Mengen.

Kapitel 3

Normalisierung

3.1 Die Wahl einer Normalform

Bei der Beschäftigung mit einem abstrakten Datentyp bringt es oft Vorteile mit sich, eine Normalform zu verwenden.[1] Je nach Definition der Normalform erfahren diese Vorteile eine andere Ausprägung. Hier wird ein Standpunkt eingenommen, wie er ähnlich auch in [BuLi82] vertreten wird. Man erwartet, daß ein normalisiertes Objekt *einfacher* als das Ausgangsobjekt ist. Die Beziehung *einfacher* muß natürlich mathematisch präzisiert werden und wird durch eine passende partielle Ordnung auf der Menge der Objekte ersetzt. Damit läßt sich nun der Prozeß der Normalisierung darstellen als eine schrittweise Vereinfachung des Ausgangsobjektes unter Beibehaltung der semantischen Gleichheit. Von einer Normalform wird nun weiter gefordert, daß sie eindeutig ist und für jedes Objekt existiert. Hinreichend für diese beiden Forderungen sind Noetherschheit und Konfluenz des Simplifikationsverfahrens.

Für Wörter mit Variablen ist eine entsprechende partielle Ordnung leicht zu finden. Man nehme die durch den Betrag, das heißt durch die Länge eines Wortes, induzierte Relation „$\prec$“ aus Definition 2.6.1a. Die semantische Gleichheit soll das „$=$“ aus 2.6.3 sein. Die Noetherschheit der unten beschriebenen Verfahren ergibt sich dann aus der Eigenschaft der Vereinfachungsregeln, die Wörter immer nur zu verkürzen. Demgegenüber wird sich die Konfluenz nicht direkt nachweisen lassen, sondern es ist nur eine abgeschwächte Form modulo Variablenumbenennung gültig. So bekommt man aber trotzdem eine Entscheidungsprozedur für die semantische Gleichheit an die Hand. Der Test dafür, ob zwei Wörter mit Variablen semantisch gleich sind, besteht dann darin, für die beiden Wörter eine Normalform zu berechnen, und dann zu überprüfen, ob diese durch Variablenumbenennung auseinander hervorgehen.

Normalisierung ist ein ähnlicher Vorgang wie das Rechnen in einem Kalkül, zum Beispiel dem Prädikatenkalkül. An solch ein Kalkül werden immer zwei Forderungen gestellt, nämlich die der *Vollständigkeit* und *Korrektheit*. Dabei soll erstere bedeuten, daß alles, was sich in einer Metaebene folgern läßt, auch mit dem Kalkül berechnet werden kann, wogegen die Korrektheit des Kalküls verlangt, daß er keine falschen Ableitungen zuläßt. Diese Nomenklatur soll auf obigen Simplifikationskalkül übertragen das folgende bedeuten:

Korrektheit einer Vereinfachungsvorschrift: Ein durch die Vorschrift abgeleitetes Wort ist erstens tatsächlich einfacher als das ursprüngliche Wort und zweitens sind die beiden Wörter semantisch gleich.

[1] Vergleiche mit den Ausführungen über Lösungsmengen von Unifikations- bzw. Generalisierungsalgorithmen auf Seite 12.

Vollständigkeit einer Vereinfachungsvorschrift: Jedes Wort, das semantisch gleich zu einem Ausgangswort ist und nicht mehr zu vereinfachen, läßt sich auch durch (mehrmalige) Anwendung der Vereinfachungsvorschrift aus letzterem herleiten.

Wie bei der Konfluenz gilt auch hier, daß sich die Vollständigkeit nur modulo Variablenumbenennung zeigen läßt.

3.2 Variablenumbenennung

Als Vorbereitung für das folgende sind einige Resultate über Variablenumbenennungen nötig:

Definition 3.1

Für ein beliebiges $u \in \Lambda$ definiere:

1. $\pi : \mathcal{V} \to \mathcal{V}$ heiße *Variablenumbenennung* genau dann, wenn π injektiv ist.
2. $\pi : \mathcal{V} \to \mathcal{V}$ heiße *Variablenumbenennung* auf $\mathcal{U} \subseteq \mathcal{V}$ genau dann, wenn die Einschränkung $\pi|_{\mathcal{U}}$ von π auf $\mathcal{U}$ injektiv ist.
3. $\forall u, v \in \Lambda : u \subseteq v \iff u \prec v$ und $u = v$.
4. $\mathsf{min}_{\subseteq}(u) := \{v \in \Lambda \mid v \subseteq u,\ \forall v' \in \Lambda \text{ mit } v' \subseteq v \text{ gilt } v \subseteq v'\}$.
5. $\mathsf{D}_u := \{x \in \mathcal{V}_u \mid \exists v \in \Lambda,\ \exists \sigma, \tau : \mathcal{V} \to \Lambda,\ \sigma(u) \equiv v,\ \tau(v) \equiv u,\ \sigma(x) \equiv \epsilon\}$.

Der Begriff der Variablenumbenennung ist völlig analog wie im klassischen Fall der leeren Theorie und bedarf wohl keiner näheren Erläuterung. D_u soll ein Maß darstellen, ob ein Wort u noch weiter vereinfachbar ist und $u \subseteq v$ modelliert die Tatsache, daß u mindestens so einfach ist wie v. Mit der Festlegung, daß ein Wort vereinfachbar ist genau dann, wenn es ein kleineres semantisch gleiches Wort gibt, bekommt man eine einfache Charakterisierung für die Vereinfachbarkeit eines Wortes:

$$u \in \Lambda \text{ ist vereinfachbar} \iff \mathsf{D}_u \neq \emptyset.$$

Die Richtung $\Rightarrow$ liefert Satz 2.14.3. Die andere Richtung, von rechts nach links, wird durch 3.2.2 bewiesen. Dies erläutert auch die Bedeutung von $\mathsf{min}_{\subseteq}(u)$ näher. $\mathsf{min}_{\subseteq}(u)$ ist also die Menge der Minima der semantisch gleichen Wörter zu u bezüglich der Relation $\subseteq$. Statt der oben genannten Definition für $\mathsf{min}_{\subseteq}(u)$ hätte man auch die dazu äquivalente Formulierung aus 3.2.1 nehmen können. Dieser sieht man aber den Minimumscharakter nicht so sehr an.

Satz 3.2

1. $\mathsf{min}_{\subseteq}(u) = \{v \in \Lambda \mid v = u,\ \forall w \in \Lambda\colon\ w = u \Rightarrow |v| \leq |w|\}$.
2. $\mathsf{D}_u = \emptyset \iff u \in \mathsf{min}_{\subseteq}(u)$.
3. $\forall w \in \Lambda,\ u, v \in \mathsf{min}_{\subseteq}(w)$ gibt es $\sigma : \mathcal{V} \to \mathcal{V}$ mit $\sigma(u) \equiv v$ und σ ist eine Variablenumbenennung auf $\mathcal{V}_u$.

Beweis von 3.2.1:

„$\subseteq$“: $v \subseteq u$ ergibt insbesondere $v = u$. Sei also nun $w \in \Lambda$ mit $w = u$ so folgt mit der Transitivität von „$=$“ auch $w = v$. Falls nun schon $|v| \leq |w|$ gilt, so ist man fertig. Im anderen Fall gilt $|w| \leq |v|$ und damit auch $w \subseteq v$. Die Voraussetzung liefert dann $v \subseteq w$, woraus insbesondere $|v| \leq |w|$ folgt.

„$\supseteq$“: Aus $u = v$ folgt unmittelbar auch $v \subseteq u$, denn sonst wäre $|u| < |v|$, was direkt der Voraussetzung widerspricht. Sei nun weiter ein v' mit $v' \subseteq v$ gegeben. Mit der Transitivität von „$=$“ gilt nun auch $v' = u$, was mit der Voraussetzung $|v'| \leq |u|$ ergibt. Die letzten beiden Gleichungen zusammengenommen liefern dann $v \subseteq v'$.

■

Beweis von 3.2.2: Der Beweis wird jeweils über die Kontraposition der zwei Implikationen geführt:

„$\Rightarrow$“: Sei $u \notin \mathsf{min}_{\subseteq}(u)$. Da „$=$“ reflexiv ist, hat man natürlich $u = u$. Deshalb muß es nach 3.2.1 ein w geben mit $u = w$ aber $|w| < |u|$. Aus der Gleichheit bekommt man ein $\sigma : \mathcal{V} \to \Lambda$, für das $\sigma(u) \equiv w$ gilt. Somit ist $|\sigma(u)| < |u|$, also $\sigma(u) \prec u$, was mit 2.14.3 ein $x \in \mathcal{V}_u$ liefert mit $\sigma(x) \equiv \epsilon$. Dieses x liegt damit in D_u, womit sich diese Menge als nicht leer erweist.

„$\Leftarrow$“: Sei nun x aus D_u beliebig. Zu diesem x gibt es dann Substitutionen $\sigma, \tau : \mathcal{V} \to \Lambda$ und ein Wort $v \in \Lambda$, so daß die Gleichungen $\sigma(u) \equiv v$, $\tau(v) \equiv u$ und $\sigma(x) \equiv \epsilon$ erfüllt sind.

Nun definiere $u' :\equiv \{x/\epsilon\}u$, womit sich $\sigma(u') \equiv (\sigma \circ \{x/\epsilon\})(u) \equiv \sigma(u) \equiv v$ nachrechnen läßt. Weiter gilt: $(\tau \circ \sigma)(u') \equiv \tau(\sigma(u')) \equiv \tau(v) \equiv u$, und damit $u = u'$ und $|u'| < |u|$, da $x \in \mathcal{V}_u$. Ein Blick auf die Definition von $\mathsf{min}_{\subseteq}(u)$ ergibt schließlich $u \notin \mathsf{min}_{\subseteq}(u)$.

■

Beweis von 3.2.3: Aus $u, v \in \mathsf{min}_{\subseteq}(w)$ folgt insbesondere, daß es $\tau, \sigma : \mathcal{V} \to \Lambda$ gibt, für die $\sigma(u) \equiv v$, $\tau(v) \equiv u$ gilt. Für alle τ und σ, die diese Gleichungen erfüllen, wird nun nacheinander gezeigt:

1. $\sigma, \tau : \mathcal{V} \to \mathcal{V}'$: gilt nach 2.13.
2. $\forall x \in \mathcal{V}_u : \sigma(x) \not\equiv \epsilon$: Sonst wäre $\mathsf{D}_u \neq \emptyset$ und damit nach 3.2.2 $u \notin \mathsf{min}_{\subseteq}(u) = \mathsf{min}_{\subseteq}(w)$.
3. $\forall y \in \mathcal{V}_v : \sigma(y) \not\equiv \epsilon$: analog zu 2.
4. $\forall x \in \mathcal{V}_u, y \in \mathcal{V}_v : \sigma(x), \tau(y) \in \mathcal{V}$: Definiere $\delta := \tau \circ \sigma$, so ist $\delta(u) \equiv u$ (u ist Fixpunkt von δ). Angenommen es gäbe ein $x \in \mathcal{V}_u$ mit $|\sigma(x)| \geq 2$, dann gibt es $l, r \in \Lambda$, so daß $u \equiv lxr$. Da τ keine Variable auf ϵ abbildet, gilt auch $|\delta(x)| \geq 2$ und es folgt:

$$\begin{array}{rrcl} & |\delta(lxr)| & = & |lxr| \\ \Longrightarrow & |\delta(lr)| + \underbrace{|\delta(x)|}_{\geq 2} & = & |lr| + 1 \\ \Longrightarrow & |\delta(lr)| & < & |lr|\,. \end{array}$$

   Nach 2.14.1 muß es ein $x \in \mathcal{V}_{lr} \subseteq \mathcal{V}_u$ geben, so daß $\delta(x) \equiv \epsilon$. Dies ist führt zum Widerspruch zu Punkt 2, da die Voraussetzungen des Satzes auch auf δ zutreffen.

5. σ, τ sind injektiv auf $\mathcal{V}_u$: Zeige σ injektiv: angenommen es gibt $x, y \in \mathcal{V}_u$ mit $x \not\equiv y$ aber $\sigma(x) \equiv \sigma(y)$. Definiere $W := \mathcal{V}_u \backslash \{x, y\}$. Nach dem vorangehenden ist sowohl $\sigma(x)$ als auch $\sigma(y)$ eine Variable, und man rechnet nach:

$$\begin{array}{rcll} |\mathcal{V}_v| & = & \left|\mathcal{V}_{\sigma(u)}\right| & = & |\sigma(\{x, y\}) \cup \sigma(W)| \\ & = & |\sigma(\{x, y\})| + |W| & = & 1 + |W| \\ & < & 2 + |W| & = & |V_u| \end{array}$$

Ein Blick auf Satz 2.14.2 liefert dann ein $z \in \mathcal{V}_v$ mit $|\mathcal{V}_\tau(z)| > 1$ im Widerspruch zu 3. Der Beweis für die Injektivität von τ geht völlig analog.

■

Korollar 3.3

$\forall w \in \Lambda,\ u, v \in \mathsf{min}_{\subseteq}(w)$ gibt es eine Variablenumbenennung $\sigma : \mathcal{V} \to \mathcal{V}$ mit $\sigma(u) \equiv v$.

Beweis: Nach 3.2.3 bekommt man eine Variablenumbenennung π' auf $\mathcal{V}_u$ mit der gewünschten Eigenschaft: $\pi'(u) \equiv v$. Nach Satz A.5 gibt es dann eine injektive Erweiterung π von π' auf ganz $\mathcal{V}$. ■

Mit anderen Worten bedeutet dies, daß sich semantisch gleiche Wörter, die nicht mehr zu vereinfachen sind, nur bis auf eine Variablenumbenennung unterscheiden.

3.3 Die Simplifikationsvorschrift $\vdash$

Bei der Suche nach einer passenden Simplifikationsvorschrift hilft Satz 2.14.3 weiter und führt zur folgenden Definition einer ersten Vereinfachungsvorschrift:

Definition 3.4

Sei $u, v \in \Lambda$, so schreibe $u \vdash_0 v \ :\Longleftrightarrow\ \exists x \in \Lambda,\ \exists\, \sigma : \mathcal{V} \to \Lambda$, so daß $\{x/\epsilon\}u \equiv v$ und $\sigma(v) \equiv u$. Damit sei nun $\vdash\ := \mathsf{Tr}(\vdash_0)$.

Tr ist der in Abschnitt A.2 definierte transitive Abschlußoperator. Hierbei beachte man, daß dieser Operator im allgemeinen keine reflexive Relation erzeugt. Es gilt also nicht $u \vdash u$.

Dieser abstrakten Definition soll ein konkretes Beispiel folgen. Es wird das Wort $xyxy$ betrachtet. Ersetzt man darin z.B. x durch ϵ, so erhält man $\{x/\epsilon\}(xyxy) \equiv yy$. Mit $\{y/xy\}(yy) \equiv xyxy$ gilt deswegen: $xyxy \vdash yy$. Aus Symmetriegründen, indem man y durch ϵ ersetzt, folgt analog $xyxy \vdash xx$. Weiter ist klar, daß sich weder xx noch yy vereinfachen lassen. Man ist also schon bei Normalformen bezüglich „ $\vdash$ “ angelangt. Nun ist zwar $xx \not\equiv yy$, aber sie sind bis auf Variablenumbenennung gleich.

Ein weiteres Beispiel stellen alle Wörter dar, die nur aus Variablen bestehen, und in denen eine Variable nur einmal vorkommt:

$$\Lambda_1 :\equiv \{u \in \mathcal{V}^+ \mid \exists x \in \mathcal{V}_u : \#(x, u) = 1\}$$

Für ein solches $u \in \Lambda_1$ sei x_u eine Variable, die obige Bedingung erfüllt, dann lassen sich Substitutionen $\sigma_u, \tau_u : \mathcal{V} \to \Lambda$ definieren durch:

$$\tau_u(y) :\equiv \{x_u/u\},\ \sigma_u(y) :\equiv \begin{cases} x_u & \text{falls } y \equiv x_u \\ \epsilon & \text{sonst} \end{cases},\ \text{für alle } y \in \mathcal{V}$$

Damit ergibt sich für alle $u \in \Lambda_1$: $\sigma(u) \equiv x_u$ und $\tau(x_u) \equiv u$ also $u = x_u$.

Dann ist klar, daß die Anwendung eines σ_u auf ein u durch einzelne „ $\vdash_0$ “-Schritte nachgebildet werden kann. Somit gilt $u \vdash x_u$ und $\Lambda_1/_= \ = \ [z]_=$ mit einem beliebigen, aber festen $z \in \mathcal{V}$.

Satz 3.5 (Korrektheit von $\vdash$)

$\forall u, v \in \mathcal{V}: u \vdash v \;\Rightarrow\; u = v,\ |v| < |u|$

Beweis: Die Voraussetzung ergibt die Existenz eines $n \in \mathbb{N},\ n > 0$ und von Wörtern $u_0, \ldots, u_n \in \Lambda$, so daß

$$u \equiv u_0 \vdash_0 u_1 \vdash_0 \cdots \vdash_0 u_{n-1} \vdash_0 u_n \equiv v$$

Seien jetzt weiter $x_1, \ldots, x_n$ die Variablen in den einzelnen Ableitungsschritten, die durch ϵ ersetzt worden sind, und $\sigma_1, \ldots, \sigma_n$ die entsprechenden Substitutionen mit

$$\sigma_i(u_i) \equiv u_{i-1},\ \{x_i/\epsilon\}(u_{i-1}) \equiv u_i \text{ für } i = 1 \ldots n$$

Dann definiere man die Substitutionen $\sigma, \tau: \mathcal{V} \to \Lambda$ durch

$$\sigma :\equiv \sigma_n \circ \cdots \circ \sigma_1 \text{ und } \tau :\equiv \{x_1/\epsilon, \ldots, x_n/\epsilon\}$$

Insgesamt gilt dann $\sigma(u) \equiv v$ und $\tau(u) \equiv v$. Damit ist einmal $u = v$, und, da $n > 0$ ist, gilt auch $|v| < |u|$. ■

Wie aus dem ersten Beispiel ersichtlich ist, läßt sich keine symmetrische Formulierung für die Vollständigkeit finden, da „$\vdash$“ höchstens bis auf Variablenumbenennung vollständig ist. Aber es gibt noch ein weiteres Problem. Das Kriterium der Längenverkürzung reicht leider nicht aus, um die Ableitbarkeit eines Wortes zu gewährleisten. Man betrachte folgendes Beispiel:

$$u :\equiv xyxyxyzz \text{ und } v :\equiv xxxzyzy$$

Dann folgt $\{y/\epsilon, z/zy\}(u) \equiv v$ und $\{x/xy, y/\epsilon\}(v) \equiv u$, also $u = v$ und $|v| < |u|$, aber $u \vdash v$ gilt nicht. Es fehlt also noch eine weitere Einschränkung an die Wörter u und v.

Im Normalisierungsvorgang interessiert man sich eigentlich nur für das Ergebnis, die Normalform. Die Zwischenschritte sind nur insofern wichtig, daß sie die Korrektheit der Vorgehensweise sichern. Also genügt es vollkommen, zu zeigen, daß alle nicht weiter vereinfachbaren Wörter, die semantisch gleich zu einem Ausgangswort sind, aus diesem mittels „$\vdash$“ ableitbar sind. Wie oben gezeigt wurde, unterscheiden sich nicht mehr vereinfachbare Wörter, die semantisch gleich sind, nur durch eine Variablenumbenennung. Damit liegt die neue Definition von Vollständigkeit ganz im Sinne des Konfluenzbegriffes, bei dem Zwischenergebnisse ruhig auseinander liegen können, wenn nur gesichert ist, daß das Endergebnis gleich ausfällt, sich in diesem Fall also nur bis auf eine Variablenumbenennung unterscheidet.

Satz 3.6 (Vollständigkeit von $\vdash$)

$\forall u, v \in \Lambda: v \in \min_{\subseteq}(u),\ |v| < |u| \;\Rightarrow\; \exists \pi: \mathcal{V} \to \mathcal{V},\ \pi$ Variablenumbenennung, $u \vdash \pi(v)$.

Beweis: Der Beweis wird geführt durch Induktion über $|\mathsf{D}_u| =: n$. Der Fall $n = 0$ kann nicht auftreten, denn sonst ist $\mathsf{D}_u = \emptyset$, womit nach Satz 3.2.2 $v \in \min_{\subseteq}(u) = \min_{\subseteq}(v)$ gilt. Dies ergibt zusammen mit 3.2.1 $|u| = |v|$, im Widerspruch zur Voraussetzung. ↯

$n \rightsquigarrow n+1$ $(n > 1)$**:** Nach Definition von D_u bekommt man $x \in \mathcal{V}_u$, Substitutionen $\sigma, \tau: \mathcal{V} \to \Lambda$ und ein Wort $w \in \Lambda$, so daß $\sigma(u) \equiv w,\ \tau(w) \equiv u,\ \sigma(x) \equiv \epsilon$. Definiere nun $w' :\equiv \{x/\epsilon\}u$, so rechnet man zuerst $u \equiv (\tau \circ \sigma)(w')$ nach, womit $u = w' = v$ folgt. Damit ist auch $v \in \min_{\subseteq}(w')$ und, da $|\mathsf{D}_{w'}| < |\mathsf{D}_u|$ ist, liefert die Induktionsvoraussetzung eine Variablenumbenennung $\sigma: \mathcal{V} \to \mathcal{V}$, so daß $w' \vdash \sigma(v)$. Nach Definition von w' gilt $u \vdash w'$, was mit der Transitivität von „$\vdash$“ zu $u \vdash \sigma(v)$ führt.

$\underline{n = 1:}$ Der Beweis folgt dem des Induktionsschrittes bis zu dem Zeitpunkt, wo die Induktionsvoraussetzung benutzt wird. An dieser Stelle hat man jetzt zusätzlich $\mathsf{D}_{w'} = \emptyset$. Mit 3.2.2 gilt also $w' \in \min_{\subseteq} u$. Das Variablenumbenennungskorollar 3.3 liefert eine Variablenumbenennung $\sigma : \mathcal{V} \to \mathcal{V}$ mit $\sigma(v) = w'$. Wie im Induktionsschritt gilt auch hier $u \vdash w'$, was mit der letzten Gleichung den Beweis beendet.

■

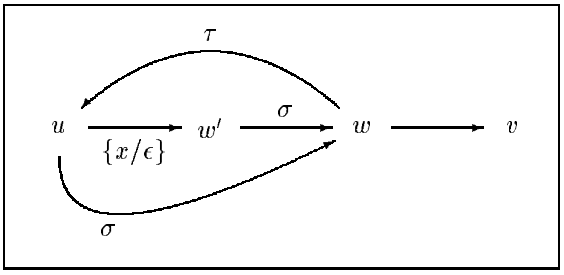


Abbildung 3.1: Veranschaulichung der Situation im Induktionsschritt beim Beweis der Vollständigkeit von $\vdash$. Es gilt $\sigma(x) \equiv \epsilon$, $x \in \mathsf{D}_u$ und $\sigma, \tau : \mathcal{V} \to \Lambda$ sind die entsprechenden Substitutionen in der Definition von D_u.

Die in Definition 3.4 definierte Simplifikationsvorschrift hat leider einen gewaltigen Nachteil. Um ein Wort u vereinfachen zu können, müssen zunächst zwei Dinge gefunden werden. Einmal eine Variable $x \in \mathcal{V}_u$ und eine Substitution σ, mit $\sigma(\{x/\epsilon\}u) \equiv u$. Ein *brute-force* Algorithmus würde also erstmal alle Variablen aus $\mathcal{V}_u$ durchgehen und dabei jedesmal die Existenz eines solchen σ testen. Letzteres läßt sich zurückführen auf das Filter- bzw. Matchingproblem, was in Kapitel 6 behandelt wird. Wie dort gezeigt wird, ist ein solcher Algorithmus im „*worst-case*“ mindestens in der Komplexität von $\mathbf{O}(|u|^2)$. Mit der Abschätzung[2] $|\mathcal{V}_u| \leq |u|$ kommt man also mindestens auf einen $\mathbf{O}(|u|^3)$ Algorithmus für das Finden eines Vereinfachungsschrittes. Abbildung 3.2 zeigt einen solchen Algorithums in Pseudocode[3]. Es wäre also wünschenswert, einen schnelleren und direkteren Weg zu finden.

Der hier beschriebene Algorithmus kann nur auf Vereinfachbarkeit testen. Aber durch eine kleine Modifikation läßt sich auch eine Vereinfachung ausgeben, falls eine existiert. Dazu muß man nur die Rückgabeparameter der beiden Prozeduren entsprechend ändern.

3.4 Die Simplifikationsvorschrift ▷

Für eine einfachere Normalisierungsvorschrift ist der Zusammenhang zwischen Fixpunkten von Substitutionen und der semantischen Gleichheit wichtig. Ist nämlich $u = v$, dann gibt es nach Definition 2.6.3 zwei Substitutionen $\sigma, \tau : \mathcal{V} \to \Lambda$, so daß $\sigma(u) \equiv v$ und $\tau(v) \equiv u$. Setzt man $\delta :\equiv \tau \circ \sigma$, so ist u ein Fixpunkt von δ, das soll heißen, es gilt $\delta(u) \equiv u$.

Zuerst zeigt Abbildung 3.3 ein Beispiel dafür, daß Substitutionen mit einem Fixpunkt schon eine gewisse strukturelle Komplexität besitzen können. Jede Zeile soll das gleiche, nur aus Variablen

[2] Man beachte die unterschiedliche Bedeutung der Betragszeichen: Der erste Betrag ist die Anzahl der Elemente in einer Menge, während der zweite Betrag die Länge eines Wortes bezeichnet.

[3] Der hier verwendete Pseudocode ist stark an C ausgerichtet. Für einen mehr mit Pascal oder ähnlichen Programmiersprachen vertrauten Leser sei auf folgende Unterschiede hingewiesen: Der Vergleichsoperator in C lautet statt „:=“ in Pascal nur „=“. Deshalb wurde in C der Vergleichsoperator zu „==“ umdefiniert und schließlich wird eine Blockstruktur nicht mit einem Paar aus „**begin**“ und „**end**“ gebildet, sondern der Block wird mit geschweiften Klammern („{“, „}“) umschlossen.

```
Boolean ist_zu_vereinfachen(u ∈ Λ)
{
    v ∈ Λ;

    forall x in V_u do {
        v = {x/ε}u;
        if ( match(u, v) )return True;
    };
    return False;
}
```

Abbildung 3.2: Test auf Vereinfachbarkeit mittels $\vdash_0$. Als Hilfsprozedur wird hier ein Filteralgorithmus benötigt, wie man ihn zum Beispiel in Kapitel 6 findet. Falls „match(u,v)“ zutrifft, dann soll das $u \leq v$ bedeuten, also v eine Instanz von u sein.

bestehende Wort $u :\equiv x_0zx_0zx_1yx_1zx_2x_0z$ darstellen. Als Substitution mit Fixpunkt u wurde $\delta :\equiv \{x_0/\epsilon, x_1/x_0z, x_2/x_1yx_1zx_2x_0z, y/\epsilon, z/\epsilon\}$ gewählt. In einer solchen Darstellung eines Wortes u, das fix unter einer bestimmten Substitution σ ist, sollen die Pfeile folgendes bedeuten. Wenn zwei Pfeile von einer Variablen $x \in \mathcal{V}$ ausgehen, und die Spitzen dieser Pfeile die rechte bzw. linke Grenze eines Wortes w markieren, so gilt $\sigma(x) \equiv w$. Treffen also insbesondere die beiden Spitzen zusammen, so wird die entsprechende Variable durch ϵ ersetzt, wie das hier z.B. für z und y der Fall ist.

Die Kernidee besteht in der Beobachtung, daß jede Variable in einem solchen Fixpunkt u zu einer Substitution δ in der Variablenmenge $\mathcal{V}_{\delta(x)}$ für eine Variable $x \in \mathcal{V}_u$ vorkommen muß, oder nicht so präzise: Jede Variable aus u kommt im Bild einer Variablen aus u vor. Ketten bezüglich der

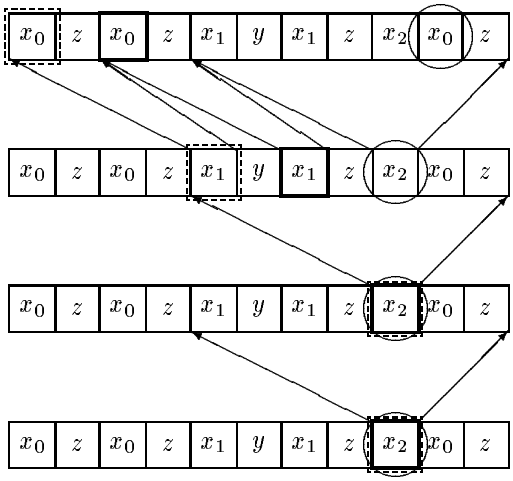


Abbildung 3.3: Konkretes Beispiel zur Definition der S^k.

Relation des Vorkommens im Bild einer Variablen kann man auch als Folgen über der Menge der Variablen $\mathcal{V}$ auffassen. Für solche Folgen von Variablen $x_0, x_1, \ldots$ wird später gezeigt, daß sie ab einem Index k_0 konstant werden, also $x_{k_0} \in \mathcal{V}_{\delta(x_{k_0})}$ gilt. Für die aus so einem Prozeß resultierende Substitution δ^{k_0} ist u immer noch ein Fixpunkt, sie hat aber *schönere* Eigenschaften als nur δ, was

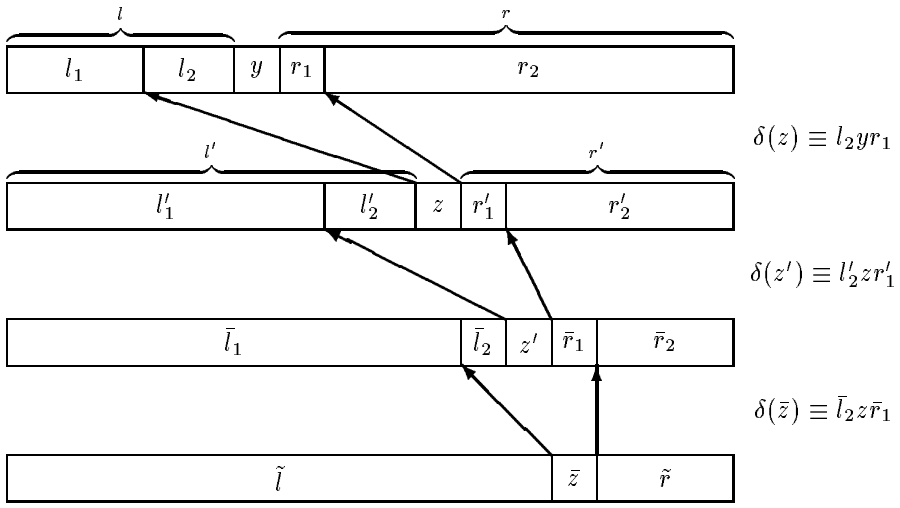


Abbildung 3.4: Abstraktes Beispiel zur Definition der S^k.

zu einem Strukturlemma über Fixpunkte führen wird.

Die Relation des Vorkommens ist leider noch keine Funktion, wie genau obiges Beispiel zeigt: x_0 und z kommen sowohl im Bild von x_1 als auch von x_2 vor. Dies führt dazu, daß auch die Konstruktion der Folge (x_k) hinsichtlich der Wahl des nächsten Folgegliedes nicht eindeutig ist. Je nachdem, ob man in Abbildung 3.3 vom linken, mittleren oder rechten Vorkommen der Variablen x_0 ausgeht, erhält man entweder die gestrichelte, die dick umrandete oder eingekreiste Folge von Variablen.

Eine präzise Definition der Relation des Vorkommens einer Variable muß also den Kontext bzw. den Ort des Vorkommens einer Variablen miteinbeziehen. Dies führt zu folgender Definition:

Definition 3.7

$\mathsf{S} = \mathsf{S}_\delta \subseteq (\Lambda \times \mathcal{V} \times \Lambda) \times (\Lambda \times \mathcal{V} \times \Lambda)$ wird definiert für alle $l, r, l', r' \in \Lambda,\ y, z \in \mathcal{V},\ \delta: \mathcal{V} \to \Lambda$, mit $\delta(lyr) \equiv lyr$ und $lyr \equiv l'zr'$, durch:

$$((l,y,r) , (l',z,r')) \in \mathsf{S}$$
$$\textbf{gdw}$$
$$\exists\, l_1, l_2, r_1, r_2 \in \Lambda :$$
$$l_1 l_2 \equiv l,\ r_1 r_2 \equiv r,\ \delta(z) \equiv l_2 y r_1,\ \delta(l') \equiv l_1,\ \delta(r') \equiv r_2.$$

Die ersten beiden Zeilen der Abbildung 3.4 stellen den Sachverhalt obiger Definition anschaulicher dar. Das Ziel bei der Definition von S war ja, eine Funktion zu erhalten, damit die zugehörige Folge der Variablen eindeutig ist. Umgesetzt auf die in Abbildung 3.3 verwendete Symbolik besagt das nächste Lemma, daß sich die Pfeile, die von zwei Variablen ausgehen, nicht überkreuzen.

Lemma 3.8 S ist eine Funktion, also $\mathsf{S}: \Lambda \times \mathcal{V} \times \Lambda \to \Lambda \times \mathcal{V} \times \Lambda$

Beweis: Gelte also

$$((l,y,r) , (l',z,r')) \in \mathsf{S},\ ((l,y,r) , (\lambda,\zeta,\rho)) \in \mathsf{S} \text{ und } \sigma(lyr) \equiv lyr \equiv l'zr' \equiv \lambda\zeta\rho,$$

wobei $l, l', \lambda, r, r', \rho \in \Lambda,\ y, z, \zeta \in \mathcal{V}$, so gibt es laut Definition 3.7 $l_1, l_2, \lambda_1, \lambda_2, r_1, r_2, \rho_1, \rho_2 \in \Lambda$, so daß:

$$l_1 l_2 \equiv \lambda_1 \lambda_2 \equiv l,\ r_1 r_2 \equiv \rho_1 \rho_2 \equiv r$$

$$\begin{array}{lllllllll} \delta(z) & \equiv & l_2 y r_1, & \delta(l') & \equiv & l_1, & \delta(r') & \equiv & r_2 \\ \delta(\zeta) & \equiv & \lambda_2 y \rho_1, & \delta(\lambda) & \equiv & \lambda_1, & \delta(\rho) & \equiv & \rho_2 \end{array}$$

Aus $l' \equiv \lambda$ und $r' \equiv \rho$ folgt auch $z \equiv \zeta$, womit man fertig ist. Sei also nun o.B.d.A.[4] $l' \equiv \lambda \rho \lambda'$, mit $\lambda' \in \Lambda$. Damit rechnet man folgende Gleichungen nach:

$$\begin{array}{lllll} \delta(\lambda\zeta)\delta(\rho) & \equiv & \delta(\lambda\zeta\rho) & \equiv & \delta(l' z r') \\ & \equiv & \delta(\lambda\zeta\lambda' z r') & \equiv & \delta(\lambda\zeta)\delta(\lambda')\delta(z)\delta(r') \\ & \equiv & \delta(\lambda\zeta)\delta(\lambda')l_2 y r_1 r_2 & \equiv & \delta(\lambda\zeta)\delta(\lambda')l_2 y r \end{array}$$

Dies ergibt, $\rho_2 \equiv \delta(\rho) \equiv \delta(\lambda')l_2 y r$ und der Übergang zu Beträgen liefert dann den gewünschten Widerspruch:

$$|\delta(\lambda')l_2 y r| = |\rho_2| \leq |\rho_1 \rho_2| = |r| \overset{!}{<} \delta(\lambda')l_2 y r$$

■

Diese Funktion wird nun wiederholt auf eine Ausgangssituation (l, y, r) mit einem festen δ angewendet. Man betrachtet also die Folge der Variablenkontexte $\mathsf{S}^k_\delta(l, y, r)$. In Abbildung 3.3 gelten zum Beispiel die folgenden Gleichungen, wobei

$$\delta :\equiv \{x_0/\epsilon, x_1/x_0 z, x_2/x_1 y x_1 z x_2 x_0 z, y/\epsilon, z/\epsilon\}$$

fest gewählt wird.

$$\begin{array}{lllllll} \mathsf{S}(\epsilon, \underline{x_0}, z x_0 z x_1 y x_1 z x_2 x_0 z) & \equiv & \mathsf{S}(x_0, \underline{z}, x_0 z x_1 y x_1 z x_2 x_0 z) & \equiv & (x_0 z x_0 z, \underline{x_1}, y x_1 z x_2 x_0 z) \\ \mathsf{S}(x_0 z, \underline{x_0}, z x_1 y x_1 z x_2 x_0 z) & \equiv & \mathsf{S}(x_0 z x_0, \underline{z}, x_1 y x_1 z x_2 x_0 z) & \equiv & (x_0 z x_0 z x_1 y, \underline{x_1}, z x_2 x_0 z) \\ \mathsf{S}(x_0 z x_0 z, \underline{x_1}, y x_1 z x_2 x_0 z) & \equiv & \mathsf{S}(x_0 z x_0 z x_1, \underline{y}, x_1 z x_2 x_0 z) & \equiv & \cdots \\ & \equiv & \mathsf{S}(x_0 z x_0 z x_1 y x_1 z x_2 x_0, \underline{z}, \epsilon) & \equiv & (x_0 z x_0 z x_1 y x_1 z, \underline{x_2}, x_0 z) \end{array}$$

Daraus ergibt sich zum Beispiel für die Vorkommen der drei x_0:

$$\begin{array}{lll} \mathsf{S}(\epsilon, \underline{x_0}, z x_0 z x_1 y x_1 z x_2 x_0 z) & \not\equiv & \mathsf{S}(x_0 z, \underline{x_0}, z x_1 y x_1 z x_2 x_0 z) \\ \mathsf{S}(\epsilon, \underline{x_0}, z x_0 z x_1 y x_1 z x_2 x_0 z) & \not\equiv & \mathsf{S}(x_0 z x_0 z x_1 y x_1 z x_2, \underline{x_0}, z) \\ \mathsf{S}(x_0 z x_0 z x_1 y x_1 z x_2, \underline{x_0}, z) & \not\equiv & \mathsf{S}(x_0 z, \underline{x_0}, z x_1 y x_1 z x_2 x_0 z) \end{array}$$

aber für alle $l, r \in \Lambda,\ x \in \mathcal{V}$, mit $lxr \equiv x_0 z x_0 z x_1 y x_1 z x_2 x_0 z$ gilt:

$$\mathsf{S}^2(l, x, r) \equiv (x_0 z x_0 z x_1 y x_1 z, \underline{x_2}, x_0 z)$$

Das nun folgende Lemma besagt, daß die Relation S eine gewisse Art von Transitivitätsgesetz erfüllt[5].

Lemma 3.9 (Pseudotransitivität von S)

Für alle $\delta : \mathcal{V} \to \Lambda,\ l, r, l', r' \in \Lambda,\ y, z \in \mathcal{V},\ k \in \mathbb{N}$, mit $\delta(lyr) \equiv lyr$ gilt:

$$\begin{array}{c} \mathsf{S}^k_\delta(l, y, r) \equiv (l', z, r') \\ \Longrightarrow \\ \exists\, l_1, l_2, r_1, r_2 \in \Lambda : \\ l_1 l_2 \equiv l,\ r_1 r_2 \equiv r,\ \delta^k(z) \equiv l_2 y r_1,\ \delta^k(l') \equiv l_1,\ \delta^k(r') \equiv r_2. \end{array}$$

[4] Der Beweis für die Verschiedenheit der rechten Kontexte wird symmetrisch geführt, und die freie Wahl der Bezeichner erlaubt es, l' länger als λ zu wählen.

[5] Würde in der Konklusio statt δ^k nur δ stehen, dann wäre das Lemma nur eine Umschreibung der Transitivität von S. S als Relation betrachtet kann nicht transitiv sein, denn sonst wäre S sogar idempotent, was folgendes Beispiel widerlegt: mit $u :\equiv xyz,\ \delta :\equiv \{x/\epsilon, y/x, z/yz$ gilt $\mathsf{S}(\epsilon, x, yz) \equiv (x, y, z) \not\equiv (xy, z, \epsilon) \equiv \mathsf{S}^2(\epsilon, x, yz)$.

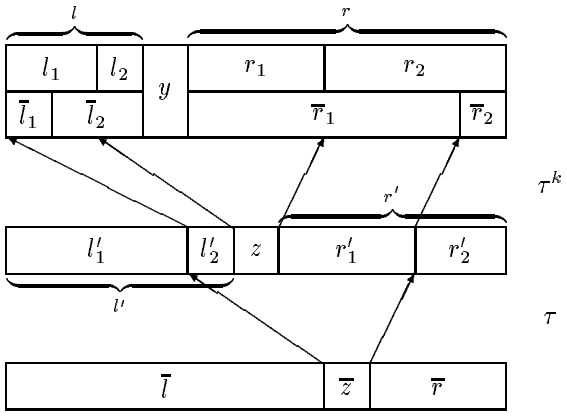


Abbildung 3.5: Beispiel zum Beweis.

Beweis: Der Beweis wird geführt durch Induktion über k. Die in diesem Beweis verwendeten Bezeichner erfüllen die in Abbildung 3.5 dargestellte Situation! Dagegen stellt Abbildung 3.4 die Bottom-Up Situation bei der Definition der S^k dar, und hier im Induktionsschritt beim Abspalten der letzten Anwendung von S eher eine Top-Down Sicht gegeben ist.

Der Induktionsanfang ist gerade Definition 3.7. Im Induktionsschritt sei

$$\mathsf{S}^{k+1}(l,y,r) \equiv (\overline{l},\overline{z},\overline{r}) \text{ und } \mathsf{S}^k(l,y,r) \equiv (l',z',r')$$

Die Induktionsvoraussetzung liefert nun die Existenz von $l_1, l_2, r_1, r_2 \in \Lambda$, so daß gilt

$$l_1 l_2 \equiv l,\ r_1 r_2 \equiv r,\ \delta^k(z) \equiv l_2 y r_1,\ \delta^k(l') \equiv l_1,\ \delta^k(r') \equiv r_2$$

Nach Definition von S im letzten Schritt von k auf $k+1$ bekommt man $l'_1, l'_2, r'_1, r'_2 \in \Lambda$, mit $\delta(\overline{z}) \equiv l'_2 y r'_1$, $\delta(\overline{l}) \equiv l'_1$, $\delta(\overline{r}) \equiv r'_2$. Nun setze man

$$\overline{l}_2 :\equiv \delta^k(l'_2) l_2,\ \overline{r}_1 :\equiv r_1 \delta^k(r'_1),\ \overline{l}_1 :\equiv \delta^k(l'_1),\ \overline{r}_2 :\equiv \delta^k(r'_2)$$

und rechnet nach:

$$\delta^{k+1}(\overline{z}) \equiv \delta^k(\delta(\overline{z})) \equiv \delta^k(l'_2 z r'_1) \equiv \delta^k(l'_2) l_2 y r_1 \delta^k(r'_1) \equiv \overline{l}_2 y \overline{r}_1$$

$$\delta^{k+1}(\overline{l}) \equiv \delta^k(\delta(\overline{l})) \equiv \delta^k(\delta(l'_1) \equiv \overline{l}_1$$

$$\delta^{k+1}(\overline{r}) \equiv \delta^k(\delta(\overline{r})) \equiv \delta^k(\delta(r'_2) \equiv \overline{r}_2$$

$$\overline{l}_1 \overline{l}_2 \equiv \delta^k(l'_1)\delta^k(l'_2) l_2 \equiv \delta^k(l'_1 l'_2) l_2 \equiv \delta^k(l') l_2 \equiv l_1 l_2 \equiv l$$

$$\overline{r}_1 \overline{r}_2 \equiv r_1 \delta^k(r'_1)\delta^k(r'_2) \equiv r_1 \delta^k(r'_1 r'_2) \equiv r_1 \delta^k(r') \equiv l_1 l_2 \equiv l$$

■

Zu einer durch die S^k definierten Folge von Variablen $z_0, z_1, \ldots$ zu einem Fixpunkt u für eine Substitution δ ist insbesonders die dazu gehörige Folge von Wörtern

$$w_0 :\equiv z_0,\ w_1 :\equiv \delta(z_1),\ w_2 :\equiv \delta^2(z_2), \ldots$$

interessant, und man stellt fest, daß sie bezüglich „$\sqsubseteq$“ eine aufsteigende Kette bilden mit der oberen Schranke u:

$$w_0 \sqsubseteq w_1 \sqsubseteq w_2 \sqsubseteq \cdots \sqsubseteq u$$

Dies ist die Aussage des nächsten Korollars:

Korollar 3.10

Zu $u, l_0, r_0 \in \Lambda,\ z_0 \in \mathcal{V},\ \delta : \mathcal{V} \to \Lambda$, mit $\delta(u) \equiv u,\ l_0 z_0 r_0 \equiv u$ und

$$(l_k, z_k, r_k) :\equiv \mathsf{S}^k(l_0, z_0, r_0),\ \ w_k :\equiv \delta^k(z_k)$$

gilt $\forall m, n \in \mathbb{N}$:

$$w_m \sqsubseteq w_{m+n} \sqsubseteq u.$$

Beweis: Die rechte Ungleichung folgt unmittelbar aus der Monotonie der Anwendung von Substitutionen bezüglich „$\sqsubseteq$“ (siehe 2.11.2). Für den Rest wende man Lemma 3.9 auf $(l, z, r) :\equiv (l_m, z_m, r_m)$, δ und $k := n$ an. Dann gibt es $r_1, r_2, l_1, l_2 \in \Lambda$, so daß $\delta^n(z_{m+n}) \equiv l_2 z_m r_1$ und

$$\begin{array}{lcll} w_{m+n} & \equiv & \delta^{m+n}(z_{m+n}) & \equiv \ \delta^m(\delta^n(z_{m+m})) \\ & equiv & \delta^m(l_2 z_m r_1) & \equiv \ \delta^m(l_2)\delta^m(z_m)\delta^m(r_1) \\ & \equiv & \delta^m(l_1) w_m \delta^m(r_2) & \end{array}$$

■

Als Veranschaulichung betrachte man wieder das Beispiel aus Abbildung 3.3 mit $l_0 :\equiv x_0,\ z_0 :\equiv z$ und $r_0 :\equiv x_0 z, x_1 y x_1 z x_2 x_0 z$ also $u :\equiv l_0 z_0 r_0$. Die dazugehörige Folge von Variablen ist dann

$$z_0 \equiv z,\ z_1 \equiv x_1,\ z_2 \equiv x_2 \equiv z_3 \equiv z_4 \equiv z_5 \equiv \cdots$$

Damit errechnet sich die Folge (w_i) wie folgt:

$$w_0 \equiv z \sqsubseteq w_1 \equiv \delta(z_1) \equiv \delta(x_1) \equiv x_0 z \sqsubseteq w_2 \equiv \delta^2(z_2) \equiv \delta^2(x_2) \equiv u \equiv w_3 \equiv \delta^3(z_3) \equiv \cdots$$

Lemma 3.11 (Strukturlemma für Fixpunkte)

Zu $u \in \Lambda,\ \delta : \mathcal{V} \to \Lambda$, mit $\delta(u) \equiv u$, gibt es $k, n \in \mathbb{N},\ y_1, \ldots, y_k \in \mathcal{V},\ u_0, \ldots, u_k \in \mathcal{S}^*$, so daß:

$$u \equiv u_0 v_1 u_1, \ldots u_{k-1} v_k u_k,\ \delta^n(v_i) \equiv v_i,\ \delta^n(u_i) \equiv u_i,\ v_i :\equiv \delta^n(y_i),\ y_i \in \mathcal{V}_{v_i}.$$

Beweis: Der Beweis wird geführt durch Induktion über $|u|$.

Im Induktionsanfang ist $|u| = 0$, somit $u \equiv \epsilon$. Wähle also $k = n = 0,\ u_0 :\equiv \epsilon$.

Im Induktionsschritt unterscheide man zwei Fälle. Zunächst sei $u \equiv su'$, für ein $s \in \mathcal{S}$. Dann ist die Aussage schon richtig, da die Substitution $\delta^{n'}$ aus der Induktionsvoraussetzung angewendet auf u' schon die Bedingung erfüllt. Es bleibt noch der Fall $u \equiv zu'$, mit $z \in \mathcal{V}$. Betrachte nun das vorhergehende Korollar bezüglich (ϵ, z, u') und δ. Wegen der Antisymmetrie von $\sqsubseteq$ (vergl. 2.11.1), und, da es nur endlich viele Wörter unterhalb von u gibt, wird die dort definierte Folge (w_i) nach höchsten m Schritten konstant, für ein $m \in \mathbb{N}$.

Mit Lemma 3.9 erhält man also ein $v \in \mathcal{V}^*,\ y \in \mathcal{V}_v$, mit $\delta^m(y) \equiv \delta^m(v) \equiv v$ und $u \equiv v\tilde{u}$. Wie im ersten Teil des Induktionsschrittes liefert die Induktionsvoraussetzung angewendet auf $\tilde{u}$ ein $\tilde{n}$, mit dem die Aussage für $\tilde{u}$ gilt. Sei nun $n := \mathsf{max}\{\tilde{n}, m\}$, so gelten obige Aussagen auch dann, wenn man die Exponenten $\tilde{n}$ und m durch den möglicherweise größeren Exponenten n ersetzt. ■

Man betrachte zum Beispiel das Wort

$$u :\equiv u_1x_1x_2y_1y_2x_1u_1x_1x_1y_3y_2x_1u_1,$$

das einen Fixpunkt der folgenden Substitution darstellt

$$\delta :\equiv \{x_1/\epsilon, x_2/x1, y1/x_2y_1, y_2/y_2x_1, y_3/x_1x_1y_3\}.$$

Nun sei $v_1 :\equiv x_1x_2y_1,\ v_2 :\equiv y_2x1,\ v_3 :\equiv x_1x_1y_3$ womit gilt:

$$u \equiv u_1v_1v_2u_1v_3u_1, \text{ und } \delta^2(v_i) \equiv \delta^2(y_i) \equiv v_i,\ y_i \in \mathcal{V}_{v_i}$$

Hier sieht man auch, wie für verschiedene Variablen der Index, ab dem die entsprechende Folge (w_i) konstant wird, unterschiedlich ist. Für y_1 ist er zwei und für die anderen Variablen ist er eins. Der Gesamtindex für das Wort ist das Maximum, also zwei.

Der Zusammenhang zwischen dieser Fixpunktstruktur und der semantischen Gleichheit zweier Wörter wird dann durch folgende Definition einer Vereinfachungsregel erläutert.

Definition 3.12

Zu $u, v \in \Lambda$ schreibe $u \triangleright v$ genau dann, wenn es eine Zerlegung $\mathcal{U} \cup \mathcal{W} \cup \mathcal{Y} = \mathcal{V}$ von $\mathcal{V}$ in paarweise disjunkte Mengen $\mathcal{U}$,$\mathcal{W}$,$\mathcal{Y}$ gibt und ein $k \in \mathbb{N}$, so daß für $i, j = 1 \ldots k$:

$$y_1, \ldots, y_k \in \mathcal{Y},\ u_0, \ldots, u_k \in (\mathcal{U} \cup \mathcal{S})^*,\ w_i, w_i' \in \mathcal{W}^* y_i \mathcal{W}^*,$$
$$|w_i'| \leq |w_i|, \text{ und } |w_n'| < |w_n| \text{ für ein } n\ (1 \leq n \leq k),$$
$$w_i \equiv w_j \text{ und } w_i' \equiv w_j', \text{ falls } y_i \equiv y_j$$

$$\begin{aligned} u &\equiv u_0w_1u_1w_2u_2 \cdots u_{k-1}w_ku_k \\ v &\equiv u_0w_1'u_1w_2'u_2 \cdots u_{k-1}w_k'u_k \end{aligned}$$

Um dies zu veranschaulichen, wird wieder das letzte Beispiel herangezogen. Jetzt lassen sich mehrere Zerlegungen angeben, die die Forderungen der Definition erfüllen. Einmal kann man wie durch die Wahl der Bezeichner schon angedeutet $\mathcal{Y} :\equiv \{y_1, y_2, y_3\}$, $\mathcal{U} :\equiv \{u_1\}$ und $\mathcal{V} :\equiv \{x_1, x_2\}$ wählen, oder dasselbe mit y_1 und x_2 vertauscht. Diese beiden Zerlegungen sind aber nicht optimal. Sie führen nämlich nur zu den noch nicht normalisierten Wörtern $u_1y_1y_2u_1y_3u_1$ bzw. $u_1x_2y_2u_1y_3u_1$. Mit den Zerlegungen $\mathcal{Y} :\equiv \{y_1, y_3\}, \mathcal{U} :\equiv \{u_1\}, \mathcal{V} :\equiv \{x_1, x_2, y_2\}$ und derselben Zerlegung mit y_1 und x_2 gerade vertauscht, gelangt man durch einen Vereinfachungsschritt sofort zu den normalisierten Wörtern $u_1y_2u_1y_3u_1$ bzw. $u_1x_2u_1y_3u_1$.

Wie der Leser bemerkt haben wird, hat man in obigen Beispielen nach Angabe einer Zerlegung der Variablenmenge immer $w_i' :\equiv y_i$ gewählt, was ja in jedem Fall korrekt ist. Um beim Vereinfachen eines Wortes so effizient wie möglich vorzugehen, wird man auch immer diese Vorgehensweise wählen. Damit stellt sich aber die Frage, warum überhaupt die w_i' eingeführt worden sind. Nun, das liegt am Beweis der Vollständigkeit von „$\triangleright$“. Dort versucht man „$\vdash$“ zu simulieren, wozu man ja schon ein Vollständigkeitsresultat hat, um so die Vollständigkeit auf „$\triangleright$“ zu übertragen. Dieser Ansatz ist um ein vielfaches leichter als der Versuch eines direkten Beweises!

Zuerst folgt die Korrektheit, von deren Richtigkeit man sich leicht ohne Rückgriff auf andere Sätze überzeugt.

Satz 3.13 (Korrektheit von $\triangleright$)

$\forall u, v \in \Lambda :\ u \triangleright v \implies u = v, |v| < |u|.$

Beweis: Gelte also $u \triangleright v$ und die Bezeichner seien wie in 3.12. Mit

$$\delta :\equiv \left\{ \begin{array}{lcl} \mathcal{V} & \to & \Lambda \\ x & \mapsto & \left\{ \begin{array}{ll} \epsilon & , \; x \in \mathcal{W} \\ x & , \; x \in \mathcal{U} \cup \mathcal{Y} \end{array} \right. \end{array} \right. , \; \sigma :\equiv \bigcup_{i=1}^{k} \{y_i / w_i\}, \; \tau :\equiv \bigcup_{i=1}^{k} \{y_i / w_i'\}$$

und der Disjunktheitsforderung folgt $\sigma(u_i) \equiv \tau(u_i) \equiv \delta(u_i) \equiv u_i$ für $i = 1, \ldots, k$. Damit ist

$$\delta(u) \equiv \delta(v) \equiv u_0 y_1 u_1 y_2 u_2 \cdots u_{k-1} y_k u_k \equiv: w,$$

woraus sich $\sigma(\delta(v)) \equiv \sigma(w) \equiv u$ und $\tau(\delta(u)) \equiv \tau(w) \equiv v$ ergibt. Also gilt $u = v$ und $|v| < |u|$ folgt definitionsgemäß. ■

Satz 3.14 (Vollständigkeit von $\triangleright$)

$\forall u, v \in \Lambda : v \in \min_{\subseteq}(u), \; |v| < |u| \; \Rightarrow \; \exists \pi : \mathcal{V} \to \mathcal{V}, \; \pi$ Variablenumbenennung, $u \triangleright \pi(v)$.

Beweis: Statt direkt die Vollständigkeit zu beweisen, wird $\vdash_{\circ} \subseteq \triangleright$ gezeigt. Mit Lemma A.4 und der Vollständigkeit von $\vdash$ (Satz 3.6) folgt dann der Rest.

Zu $u, v \in \Lambda, \; x \in \mathcal{V}_u, \; \sigma : \mathcal{V} \to \Lambda$, mit $\{x/\epsilon\}u \equiv v, \; \sigma(v) \equiv u$, definiere man $\delta := \sigma \circ \{x/\epsilon\}$. Damit ist u ein Fixpunkt von δ, und man erhält wie in Lemma 3.11 y_i, v_i, u_j usw. mit den dort aufgeführten Eigenschaften.

Definiere nun $\mathcal{Y} := \{y_1, \ldots, y_k\}, \; \mathcal{W} := \mathcal{V}_{v_1 v_2 \cdots v_k} \backslash \mathcal{Y}, \; \mathcal{U} := \mathcal{V} \backslash (\mathcal{Y} \cup \mathcal{W})$ und $w_i := v_i, \; w' := \{x/\epsilon\}w_i$. Die drei Mengen bilden offensichtlich eine paarweise disjunkte Zerlegung von $\mathcal{V}$, und $|w_i'| \leq |w_i|$ gilt auch. Desweiteren gibt es ein $n \in \mathbb{N}$, so daß $x \in \mathcal{V}_{v_n}$ und deshalb $|w_n'| < |w_n|$. Ein Vergleich mit Definition 3.12 ergibt schließlich $u \triangleright v$. ■

Bei den bisherigen Beispielen konnte man durch geschickte Wahl der Zerlegung und der y_i immer erreichen, daß schon nach einem Vereinfachungsschritt eine Normalform gefunden wurde. Die Vermutung liegt nahe, daß dies immer der Fall ist, der Beweis dafür steht aber noch aus.

Ein weiteres ungelöstes Problem in diesem Zusammenhang ist ein effizienter Algorithmus, der eine *beste* Zerlegung berechnet. Nur an Hand eines solchen Algorithmus läßt sich eine Aussage darüber treffen, ob die Vereinfachungsvorschrift „$\triangleright$“ besser abschneidet als „$\vdash$“. Der Vorteil von „$\triangleright$“ liegt aber auf jeden Fall in seiner syntaktischen Definition. Es muß kein Bezug auf Matching oder ähnliches genommen werden und falls obige Vermutung zutrifft, sieht man einem Wort sofort seine Normalform an, indem man einfach eine *beste* Zerlegung betrachtet.

Kapitel 4

Exkurs über Größte Gemeinsame Teilfolgen

Auf der Suche nach einem Verfahren zur Berechnung der Unifikation und Generalisierung (Antiunifikation) von Wörtern mit Variablen wird ein analoges Problem im Bereich von Wörtern ohne Variablen betrachtet. Dieses Kapitel beschäftigt sich dabei auch unterem anderen mit dem *Longest Common Substring*, wozu eine neue Sichtweise dargestellt wird.

An die Stelle der partiellen Ordnung $\leq$ aus 2.6.2 tritt dabei:

Definition 4.1

$\forall u, v \in \mathcal{S}^* :\ v \mid u :\Leftrightarrow\ \exists k \in \mathbb{N},\ u_0, \ldots, u_k, v_1, \ldots, v_k \in \mathcal{S}^*$, mit $u \equiv u_0 v_1 u_1 \cdots u_{k-1} v_k u_k$ und $v \equiv v_1 \cdots v_k$. Statt $v \mid u$ schreibe auch: v ist eine *Teilfolge* von u.

Zu Unterscheidung der Begriffe Teil*folge* und Teil*wort* sei auf die entsprechenden Ausführungen auf Seite 9 hingewiesen.

Eine Generalisierung zweier Wörter mit Variablen ist gerade das Infimum bezüglich $\leq$[1]. Das Infimum bezüglich $\mid$ ist nun die größte gemeinsame Teilfolge(ggT) zweier Wörter, im Englischen *Longest Common Substring* (LCS) genannt. Demgegenüber steht die Unifikation als Supremum bezüglich $\leq$. Dieser entspricht hier die *kleinste gemeinsame Oberfolge* (kgO). Hier nun die exakte Definition:

Definition 4.2

Zu $s, t \in \mathcal{S}^*$ heiße g eine *gemeinsame Teilfolge* (gT) von s und t genau dann, wenn $g \mid s$ und $g \mid t$. Darüber hinaus ist g eine *größte gemeinsame Teilfolge* (ggT) von s und t, wenn g bezüglich seiner Länge unter allen gemeinsamen Teilfolgen maximal ist.
Zu $s, t \in \mathcal{S}^*$ heiße u eine *gemeinsame Oberfolge* (gO) von s und t genau dann, wenn $s \mid u$ und $t \mid u$. Darüber hinaus ist u eine *kleinste gemeinsame Oberfolge* (kgO) von s und t, wenn u bezüglich seiner Länge unter allen gemeinsamen Oberfolgen minimal ist.

[1] Siehe Kapitel 7.

Statt direkt auf die Lösung des Infimum- oder Supremumproblems zuzusteuern, steht zunächst das Filterproblem im Vordergrund, also die Entscheidung, ob ein Wort eine Teilfolge eines anderen ist, oder im Falle der Wörter mit Variablen, ob zu gegebenen Wörtern $u, v \in \Lambda$ v eine Instanz von v ist. Die Lösung dieses Problemes wird dann schnell zu einem Algorithmus für das Infimum und Supremum führen. Diese Vorgehensweise entspricht der von Pottier in [Pott89], wo auch das Matching[2] im Falle von kommutativen Halbgruppen (AC) als Instanz der Unifikation zuerst gelößt wird.

Eine andere Motivation für die Betrachtung von variablenfreien Wörtern liegt in der Beobachtung, daß für Wörter mit Variablen u und v, so daß v eine Instanz von u ist, es definitionsgemäß eine Substitution $\sigma : \mathcal{V} \to \Lambda$ gibt, mit $\sigma(u) \equiv v$. Weiterhin läßt die Anwendung einer Substitution Konstanten invariant, und so sind die Konstanten von u auch alle in v vorhanden. Dies legt folgende Definition nahe:

Definition 4.3 (ϵ-Homomorphismus)

Die Substitution $\epsilon : \mathcal{V} \to \Lambda$ mit $\epsilon|_{\mathcal{V}} \equiv \epsilon$ heiße ϵ-Homomorphismus.

Dieser ϵ-Homomorphismus streicht also aus einem Wort mit Variablen einfach alle Variablen heraus, indem er sie durch ϵ ersetzt und stellt somit eine Abbildung von Λ nach $\mathcal{S}^*$ dar. Dies liefert nun ein notwendiges Kriterium dafür, ob ein Wort eine Instanz eines anderen ist.

Lemma 4.4 $\quad \forall u, v \in \Lambda : u \leq v \;\Rightarrow\; \epsilon(u) \mid \epsilon(v).$

Beweis: Zuerst liefert die Voraussetzung die Existenz einer Substitution $\sigma : \mathcal{V} \to \Lambda$ mit $\sigma(u) \equiv v$. Sei dann $u \equiv t_0 s_1 t_1 \cdots s_n t_n$ mit $n \in \mathbb{N}$, $s_i, t_j \in \mathcal{S}^*$. So folgt einerseits $\epsilon(u) \equiv s_1 \cdots s_n$, und andererseits hat man

$$\epsilon(v) \equiv \epsilon(\sigma(u)) \equiv (\epsilon \circ \sigma)(u) \equiv (\epsilon \circ \sigma)(t_0) s_1 (\epsilon \circ \sigma)(t_1) \cdots s_n (\epsilon \circ \sigma)(t_n)$$

Definiere nun $u_j :\equiv (\epsilon \circ \sigma)(t_j)$ und $v_j :\equiv s_i$ und Definition 4.1 steht da. ■

Aus diesem Lemma ergibt sich unmittelbar ein notwendiges Kriterien für die Unifizierbarkeit zweier Wörter.

Korollar 4.5

Zu $u, v \in \Lambda$, $\sigma : \mathcal{V} \to \Lambda$, mit $\sigma(u) \equiv \sigma(v)$ gilt: Aus $u \in \mathcal{S}^*$ folgt $\epsilon(v) \mid u$.

Beweis: Es gilt $\sigma(v) \equiv \sigma(u) \equiv u \equiv \epsilon(u)$ und mit 4.4 folgt $\epsilon(v) \mid u$. ■

In diesen Zusammenhang gehört auch das folgende Lemma, das dieselbe Aufgabe erfüllt:

Lemma 4.6

Zu $u, v \in \Lambda$, $\sigma : \mathcal{V} \to \Lambda$, mit $\sigma(u) \equiv \sigma(v)$ gilt: Gibt es $c, d \in \mathcal{S}$, so daß $u \equiv cr$ $(u \equiv rc)$ und $v \equiv ds$ $(v \equiv sd)$, dann ist $c \equiv d$.

[2] Dort heißt es *Filtrage* (vergl. [Pott89] S. 13ff).

Beweis: Sei $u \equiv cr$ und $v \equiv ds$, dann ist

$$\sigma(u) \equiv \sigma(cr) \equiv c\sigma(r) \stackrel{!}{\equiv} d\sigma(s) \equiv \sigma(ds) \equiv \sigma(v)$$

Damit muß auch $c \equiv d$ gelten. ■

Diese Beobachtungen begründen das eigentliche Interesse an Wörtern *ohne* Variablen, und die Hoffnung bestand,

aus der Vorgehensweise für die Lösung des eingeschränkten Problems etwas für die Generalisierung von Wörtern *mit* Variablen zu lernen.

Die Problemstellung wird nun verallgemeinert. Ob ein Wort Teilfolge eines anderen ist (bzw. ob ein Wort mit Variablen ein anderes matcht), wird als Instanz des folgenden Problems betrachtet:

Zähle alle Wörter auf, die Teilfolgen eines gegebenen Wortes sind.

Hier hätte man auch die dazu duale Verallgemeinerung wählen können, nämlich die Aufgabe, alle Oberfolgen eines Wortes zu bestimmen. Dies wird später noch einmal aufgegriffen, um einen Algorithmus für kgO zu finden.

4.1 Endlicher Automat zur Akzeption der Menge der Teilfolgen eines Wortes

Das nächste Teilziel besteht darin, einen Algorithmus zu finden, der zu einem Wort ohne Variablen alle Teilfolgen bestimmt, und darüber hinaus noch übertragbar auf Wörter mit Variablen ist. Die Menge der Teilfolgen eines Wortes ist sehr groß.[3] Sie ist natürlich endlich, aber dennoch wäre es besonders in Hinblick auf Wörter mit Variablen interessant, ob es eine *kurze* Beschreibung gibt. Bei Wörtern *mit* Variablen hat man im Falle des analogen Problems des Matchings eine ähnliche Situation.

Tritt eine solche Fragestellung im Bereich der formalen Sprachen auf, versucht man zuerst die Einordnung in die Chomsky-Hierachie festzulegen. Für Wörter ohne Variablen ist wegen der Endlichkeit der Lösungsmenge das Filterproblem regulär. Es bleibt also die Frage nach einer uniformen Methode, aus einem gegebenen Wort einen endlichen Automat zu konstruieren, der alle Teilfolgen des gegebenen Wortes akzeptiert.

Ein solcher Automat ist ein Tupel $(Z, \triangle, A, S, F)$ aus Zuständen Z, dem Alphabet A, der Menge der Startzustände $S \subseteq Z$, der Menge der Finalzustände $F \subseteq Z$ und schließlich der Zustandsübergangsrelation $\triangle \subseteq Z \times S \times Z$. Um die formale Definition in Einklang mit der graphischen Darstellung eines Automaten zu bringen, wird statt $(z, s, z') \in \triangle$ meistens $z \stackrel{s}{\rightarrow} z'$ geschrieben.

Definition 4.7 (Automat zur Akzeption von Teilfolgen)

Zu $u \in \Lambda$ sei der indeterministische endliche Automat, der alle Teilfolgen des Wortes u akzeptiert, der erreichbare Teil des folgenden Automaten:

$$A :\equiv \mathcal{S},\ Z :\equiv \mathcal{S}^*,\ S :\equiv \mathcal{S} \cup \{\epsilon\},\ F :\equiv \{\epsilon\} \tag{4.1}$$

Die Übergangsrelation „$\rightarrow$“ sei dann die kleinste Relation mit folgenden Eigenschaften:

$$cr \stackrel{c}{\rightarrow} r,\ cr \stackrel{\epsilon}{\rightarrow} r,\ \text{mit } r \in \mathcal{S}^* \text{ und } c \in \mathcal{S} \tag{4.2}$$

[3] vergleiche die Abschätzung in Kapitel A.5.

Die Zustände des Automaten sind also alle Suffixe (inklusive dem leeren Wort) des gegebenen Wortes, und es gibt jeweils genau zwei Zuständsübergänge von einem Suffix zum um einen Buchstaben kürzeres. Der eine Übergang sei mit ϵ und der andere mit dem wegfallenden Buchstaben beschriftet. Der Startzustand ist das gegebene Wort selbst und der einzige Finalzustand das leere Wort. Abbildung 4.1 zeigt eine solche Konstruktion für ein gegebenes Wort $z_0 \cdots z_n$. Die dabei verwendete

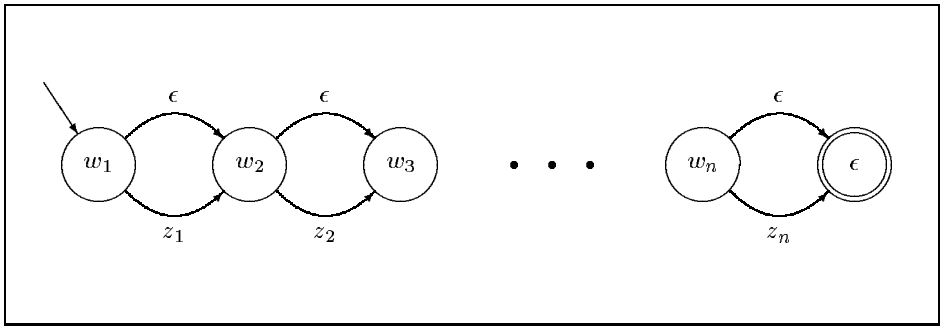


Abbildung 4.1: Dies ist ein endlicher Automat, der genau alle Teilfolgen des Wortes $z_1 \cdots z_n$ erkennt, wobei $n \in \mathbb{N}$, $z_i \in \mathcal{S}$. Dabei wird die Abkürzung $w_i :\equiv z_i \cdots z_n$ verwendet.

Symbolik ist die übliche:

- Zustände werden mit Kreisen dargestellt und ihre Namen werden in die Kreise geschrieben.
- Ein Pfeil von einem Kreis zu einem anderen, versehen mit einem Label l, soll bedeuten, daß es einen Zustandsübergang von den entsprechenden Zuständen unter dem Zeichen l gibt.
- Ein Startzustand wird mit einem Pfeil gekennzeichnet, der auf den entsprechenden Kreis zeigt und keinen Kreis als Ursprung hat.
- Die Finalzustände werden durch einen zweiten Kreis um den Namen des Zustandes gekennzeichnet.

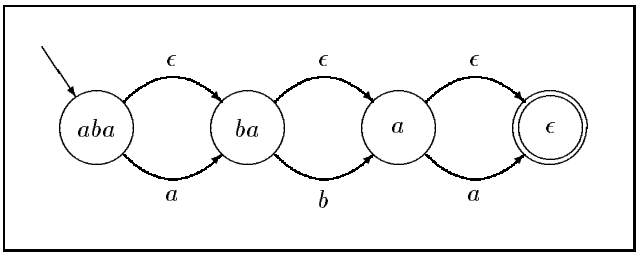


Abbildung 4.2: Dies ist ein *indeterministischer* endlicher Automat zur Akzeption aller Teilfolgen des Wortes aba.

Abbildung 4.2 liefert ein konkretes Beispiel. Dort ist ein indeterministischer Automat aufgezeigt, der alle Teilfolgen von aba akzeptiert. Für die weitere Betrachtung ist der Indeterminimus recht unhandlich. Eine Anwendung der Teilmengenkonstruktion liefert daraus den in Abbildung 4.3 dargestellten epsilonfreien deterministischen Automaten, der äquivalent zum ersten auch alle Teilfolgen von aba akzeptiert. Was bei der Betrachtung des Automaten auffällt, ist, daß die neuen Zustände, die ja nun Mengen von Zuständen des alten Automaten sind, immer aus einem Suffix von aba und

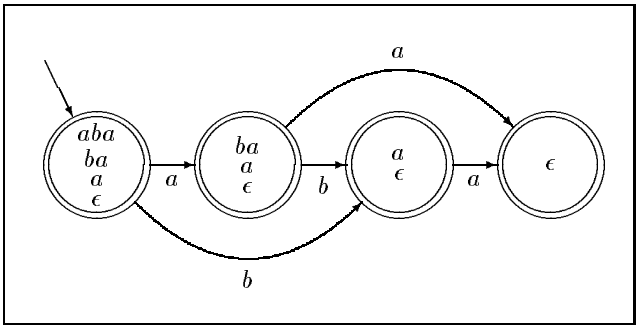


Abbildung 4.3: Dies ist ein *epsilonfreier deterministischer* endlicher Automat, der genau alle Teilfolgen des Wortes *aba* erkennt.

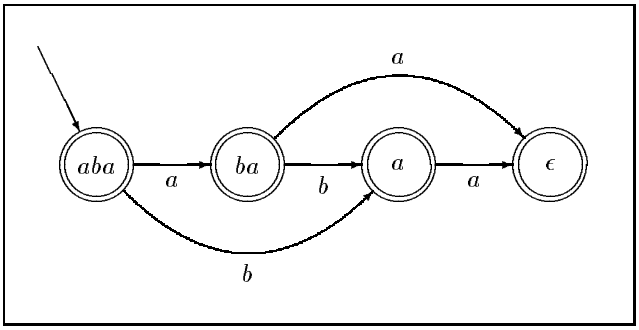


Abbildung 4.4: Dies ist ein *epsilonfreier deterministischer* endlicher Automat, der genau alle Teilfolgen des Wortes *aba* erkennt. Im Gegensatz zu dem vorherigen Automaten, der daselbe erfüllt, werden hier die Zustandsmengen nur durch ein Wort repräsentiert und zwar durch das längste.

allen dessen Suffixen bestehen. Man kann also die Beschriftung der Zustände reduzieren und anstatt alle Suffixe aufzuführen, wird wie in Abbildung 4.4 nur das längste Suffix aufgeführt. Damit wird Defintion 4.7 zu

Definition 4.8 (Automat zur Akzeption von Teilfolgen)

Die Gleichungen (4.1) werden bis auf die Finalmenge übernommen. Für diese soll nun gelten $F :\equiv Z$. Die Definition von „$\rightarrow$“ aus Gleichung (4.2) wird wie folgt modifiziert:

$$lzr \xrightarrow{z} r \text{ falls } l, r \in \mathcal{S}^*,\ z \in \mathcal{S} \backslash \mathcal{S}_l \tag{4.3}$$

Wenn ein Zustand mit einem Wort $w \in \mathcal{S}^*$ beschriftet ist, und es ein Suffix zr von w mit $z \in \mathcal{S}$ gibt, so daß für das Wort $l \in \mathcal{S}^*$ mit $lzr \equiv w$ gilt, daß z nicht in l vorkommt, dann gibt es einen Zustandübergang von w nach r unter der Akzeption des Zeichens z.

Die formale Beweisführung, daß dies tatsächlich eine Definition des deterministischen Automaten ist, der äquivalent zu dem aus Abbildung 4.1 ist, soll hier nicht geführt werden. Ebenso ist wohl klar, daß die von diesem Automaten akzeptierte Sprache genau die Menge der Teilfolgen des Ausgangswortes ist.

4.2 Endlicher Automat zur Akzeption der Menge der gemeinsamen Teilfolgen zweier Wörter

Jetzt kann man diese Teillösung dazu benutzen, alle gemeinsamen Teilfolgen zweier Wörter zu bestimmen. Es wird dabei ein endlicher Automat gesucht (auch diese Menge ist endlich), der alle gemeinsamen Teilfolgen zweier Wörter akzeptiert. Nun ist die Menge der gemeinsamen Teilfolgen genau die Schnittmenge der beiden Mengen der Teilfolgen der Wörter, und die Schnittmenge zweier Sprachen von endlichen Automaten wird wieder von einem endlichen Automaten akzeptiert. Zur Bestimmung des Schnittautomaten kann die bekannte Konstruktion aus dem Bereich der formalen Sprachen benützt werden. Die Schnittbildung auf Seite der Automaten soll durch das folgende Beispiel erläutert werden.

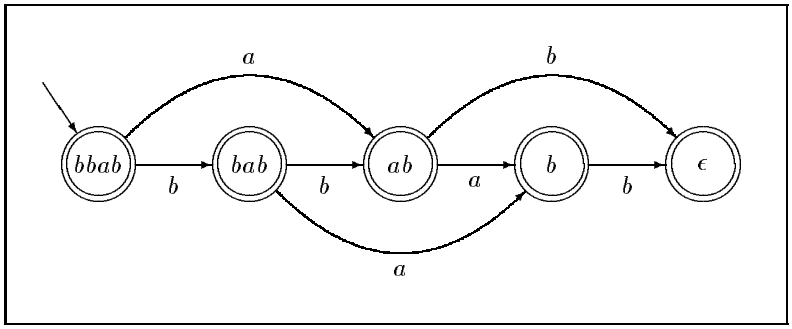


Abbildung 4.5: Dies ist ein epsilonfreier deterministischer endlicher Automat, der genau alle Teilfolgen des Wortes *bbab* akzeptiert. Man beachte, daß nun im Gegensatz zum indeterministischen Gegenstück dieses Automaten hier alle Zustände Finalzustände sind.

Abbildung 4.5 stellt einen zweiten Automaten zur Akzeption aller Teilfolgen des Wortes *bbab* dar, der nun mit dem ersten Automaten aus Abbildung 4.4 *geschnitten* werden soll. Dabei bestehen die neuen Zustände aus *ungeordneten* Paaren von Zuständen des alten Automaten. Die exakte Definition lautet nun wie folgt:

Definition 4.9 (Automat zur Akzeption gemeinsamer Teilfolgen)

Zu zwei Wörtern $u, v \in \mathcal{S}^*$ sei der deterministische endliche Automat, der alle gemeinsamen Teilfolgen von u und v akzeptiert, als der erreichbare Teil des folgenden Automaten definiert:

$$A :\equiv \mathcal{S},\ Z :\equiv \mathcal{S}^* \times \mathcal{S}^*,\ S :\equiv (u, v),\ F :\equiv \{(\epsilon, \epsilon)\}$$

mit folgender Zustandsrelation:

$$(l_1 z r_1, l_2 z r_2) \xrightarrow{z} (r_1, r_2) \text{ falls } l_1, l_2, r_1, r_2 \in \mathcal{S}^*,\ z \in \mathcal{S} \backslash (\mathcal{S}_{l_1} \cup \mathcal{S}_{l_2})$$

Da in dem ursprünglichen Automaten alle Zustände Finalzustände waren, sind auch hier alle Zustände Finalzustände.In unserem Beispiel bedeutet dies, daß die Wörtrer ϵ, a, b, ab und ba genau alle gemeinsamen Teilfolgen von aba und $bbab$ sind. Es folgen nun noch zwei weitere Beispiele:

Hat man einen solchen Automaten zur Akzeption aller gemeinsamen Teilfolgen gefunden, so liefert dieser auch die größten gemeinsamen Teilfolgen. Diese sind nämlich bijektiv abbildbar auf die längsten Pfade in dem Automaten.

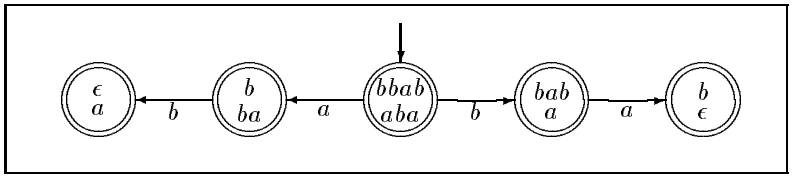


Abbildung 4.6: Dies ist ein Automat, der genau alle gemeinsamen Teilfolgen der Worte *aba* und *bbab* erkennt. Dies ist der Schnittautomat der Automaten aus Abbildung 9 und 10

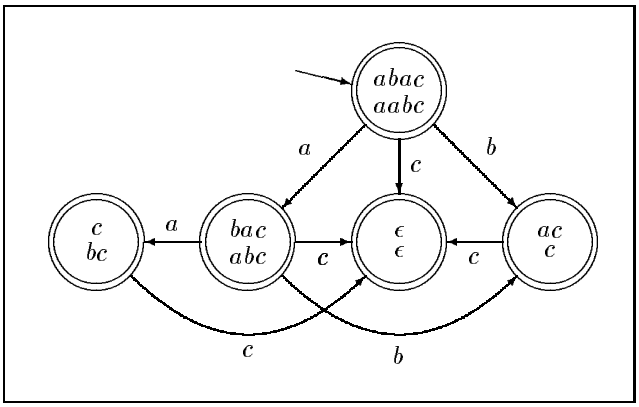


Abbildung 4.7: Dies ist ein Automat, der genau alle gemeinsamen Teilfolgen (GT) der Worte *abac* und *aabc* erkennt. Hier gibt es nur eine größte gemeinsame Teilfolge, nämlich *abc*.

Ist man nur an den längsten Pfaden interessiert, so kann man natürlich alle *Abkürzungen* aus dem Automaten streichen. Dies ist schon während der Konstruktion des Automaten möglich, wenn man die vorherige Definition 4.9 für den Schnittautomaten wie folgt modifiziert:

Definition 4.10 (Vermeidung von Abkürzungen)

Es sei alles so definiert, wie in Definition 4.9 mit folgender Ausnahme. Ein Zustandsübergang $(l_1 z r_1, l_2 z r_2) \xrightarrow{z} (r_1, r_2)$ wird **nicht** in die neue Zustandsübergangsfunktion übernommen, wenn $\mathcal{S}_{l_1} \cap \mathcal{S}_{l_2} \not\equiv \emptyset$.

In diesem Abschnitt wurde weitgehend auf formale Beweisführung verzichtet, da nur die Methodik des Vorgehens dargestellt werden sollte. Es fehlt auch eine Komplexitätsanalyse, insbesondere hinsichtlich der letzten Definition. Tiefergehende Algorithmen und Anwendungsbeispiele findet man zum Beispiel in [Tichy84].

4.3 Endlicher Automat zur Akzeption aller Oberfolgen eines Wortes

„Oberfolge“ ist der duale Begriff zu „Teilfolge“. So ist ein Wort $u \in \mathcal{S}^*$ genau dann eine Oberfolge eines Wortes $v \in \mathcal{S}^*$, wenn $v \mid u$ gilt. Wie im vorhergehenden Kapitel wird ausgehend von einem

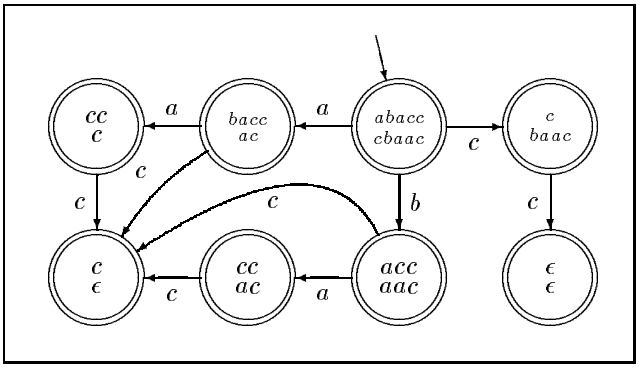


Abbildung 4.8: Dies ist ein Automat, der genau alle gemeinsamen Teilfolgen (GT) der Worte *abacc* und *cbaac* erkennt. Mit genau zwei maximalen Pfaden in diesem Automaten, haben die beiden Wörter *abacc* und *cbaac* auch genau zwei größte gemeinsame Teilfolgen, nämlich *aac* und *bac*.

Wort ein endlicher Automat konstruiert, der genau die Menge aller Oberfolgen dieses Wortes akzeptiert. Hier tritt nun zum ersten Mal der Fall auf, daß die Lösungsmenge eines Problems oberflächlich gesehen unendlich ist und es dennoch einen Algorithmus gibt, der sie berechnet. Statt die Lösungsmenge aufzuzählen, wird eine endliche Repräsentation erzeugt. Man könnte dies zum Beispiel mit der bekannten Tatsache aus der linearen Algebra vergleichen, wo ein Lineares Gleichungssystem, falls lösbar, im allgemeinen eine unendliche Lösungsmenge besitzt, die aber endlich repräsentierbar ist, z.B. durch Angabe von linear unabhängigen Lösungsvektoren.

Durch ähnliche Überlegugen wie im letzten Abschnitt gelangt man zu folgender Definition.

Definition 4.11 (Automat zur Akzeption von Oberfolgen)

Zu einem Wort $u \in \mathcal{S}^*$ sei der endliche Automat, der alle Oberfolgen von u akzeptiert, als der erreichbare Teil des folgenden Automaten definiert:

$$A :\equiv \mathcal{S},\ Z :\equiv \mathcal{S}^*,\ S :\equiv u,\ F :\equiv \{\epsilon\} \tag{4.4}$$

mit der Zustandsübergangsrelation

$$cr \xrightarrow{c} r,\ r \xrightarrow{c} r,\ \text{für alle } r \in \mathcal{S}^*,\ z \in \mathcal{S} \tag{4.5}$$

So ein Automat hat also im allgemeinen die in Abbildung 4.9 dargestellte Struktur. Hier wird das Symbol „$*$“ verwendet, um anzuzeigen, daß hier alle Labels erlaubt sind. Leider kann man hier nicht so einfach einen äquivalenten deterministischen Automaten angeben. Das liegt an der Verwendung des Metaabkürzungszeichens „$*$“. Dennoch kann man natürlich wie bei den gemeinsamen Teilfolgen auch hier den Schnittautomaten bilden, man sollte nur ein etwas kleineres Beispiel wählen, da die resultierenden Schnittautomaten recht umfangreich werden. Hierzu werden die Wörter *ab* und *ba* mit den dazu gehörenden Automaten aus Abbildung 4.10 betrachtet. Der Schnittautomat, der also alle gemeinsamen Oberfolgen von *ab* und *ba* akzeptiert, ist in Abbildung 4.11 dargestellt.

Hier entspricht die kleinste gemeinsame Oberfolge dem kürzesten Pfad vom Startzustand zum Finalzustand. In unserem Beispiel führt dies zu *aba* und *bab* als kleinste gemeinsame Oberfolgen von *ab* und *ba*.

Wie Abbildung 4.11 andeutet, ist im allgemeinen die Menge der erreichbaren Zustände in einem Automaten, der die gemeinsamen Oberfolgen zweier Wörter u und v akzeptiert, das Kreuzprodukt

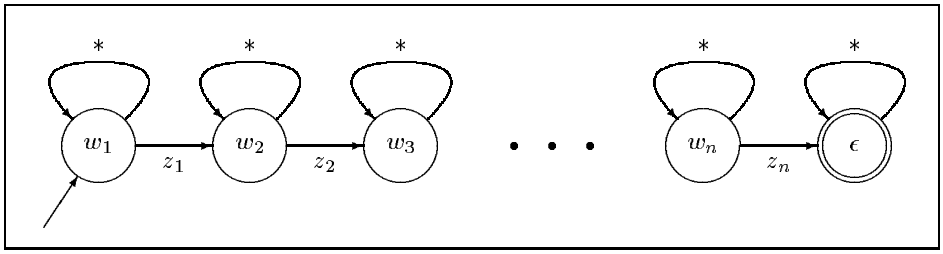


Abbildung 4.9: Dies ist ein Automat, der genau alle Oberfolgen eines allgemeinen Wortes $z_1 \cdots z_n$ erkennt, wobei $n \in \mathbb{N}$, $z_i \in \mathcal{S}$. Dabei wird die Abkürzung $w_i :\equiv z_i \cdots z_n$ verwendet.

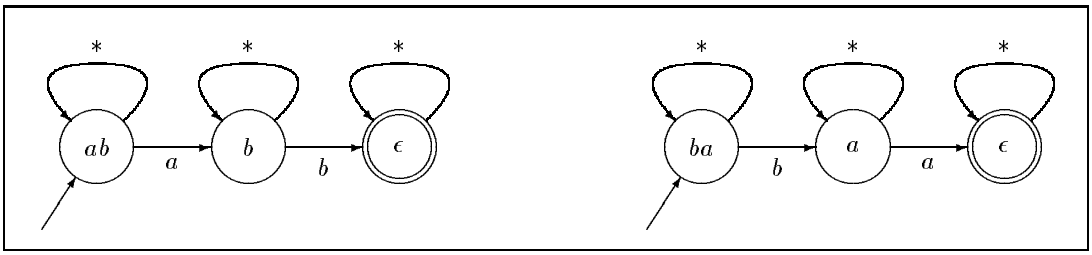


Abbildung 4.10: Dies sind zwei indeterministische endliche Automaten, die genau alle Oberfolgen eines Wortes erkennen. Der linke Automat erkennt alle Oberfolgen von ab und der rechte Automat erkennt alle Oberfolgen von ba.

zwischen den Suffixen von u und denen von v. Damit ist die Mächtigkeit der Zustandsmenge gleich $|u| \cdot |v|$.

Eine Verbesserung erreicht man nur dann, wenn man sich auf die kleinsten gemeinsamen Oberfolgen konzentriert. Dann können bei der Konstruktion des Automaten einige Zustandsübergänge von vorneherein übergangen werden, was unter Umständen auch erlaubt, einige Zustände wegzulassen.[4] Hier sind dies alle Umwege. Diese treten immer dann auf, wenn die ersten Buchstaben der beiden Wörter gleich sind, oder ein Zustandsübergang von einem Zustand zu sich selbst stattfindet. Im ersten Fall gibt es dann im optimierten Automaten zu diesem Zustand nur noch einen Zustandsübergang, nämlich von dem betrachteten Zustand zu dem Wortpaar, bei dem jeweils ein führender Buchstabe in jeder Komponente gestrichen wird. Die Übergänge von einem Zustand zu sich selbst werden einfach weggelassen. Wenn man dies auf den in Abbildung 4.11 dargestellten Automaten anwendet, dann werden die Zustände $(ba, \epsilon), (\epsilon, ab)$ und (a, b) überflüssig und man erhält den Automaten aus Abbildung 4.12.

[4] Bei dem Automaten zur Akzeption aller größten gemeinsamen Teil*folgen* im vorigen Abschnitt bleibt dagegen die Menge der Zustände nach der Optimierung gleich groß, es verringert sich nur eventuell die Anzahl der Zustandsübergänge.

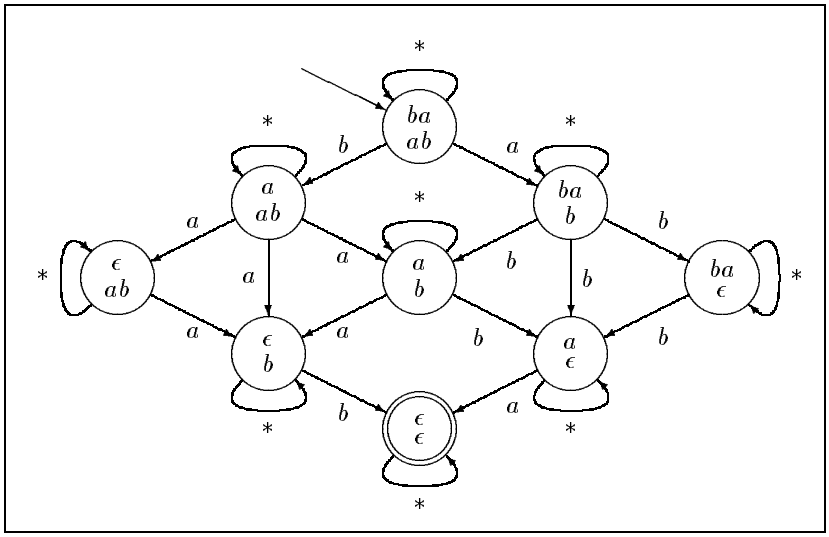


Abbildung 4.11: Dies ist ein endlicher Automat, der alle gemeinsamen Oberfolgen der Worte ab und ba erkennt. Betrachtet man die kürzesten Pfade vom Startzustand zum Finalzustand, so entsprechen diese den akzeptierten Wörtern aba und bab, die somit die kleinsten gemeinsamen Oberfolgen von ab und ba darstellen.

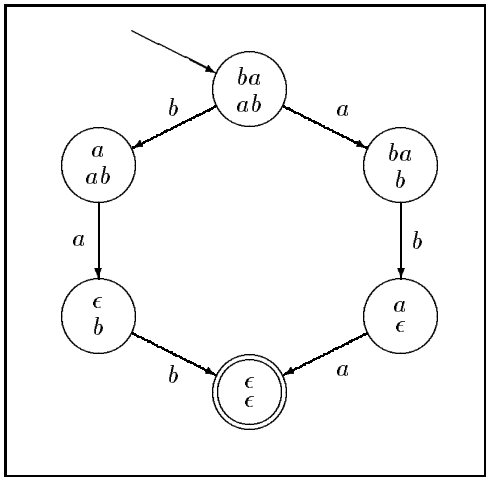


Abbildung 4.12: Dieser Automat erkennt alle kleinsten gemeinsamen Oberfolgen von ba und ab. Er entsteht aus dem letzten Automaten durch Streichen aller Umwege.

Kapitel 5

Unifikation

In diesem Kapitel wird die Unifikation von Wörtern mit Variablen betrachtet:

Definition 5.1 (Unifikation)

Zwei Wörter mit Variablen $u, v \in \Lambda$ heißen *unifizierbar* genau dann, wenn es eine Substitution $\sigma : \mathcal{V} \to \Lambda$ gibt, so daß $\sigma(u) \equiv \sigma(v)$. Ein solches σ, falls es existiert, heißt *Unifikator* von u und v und $\sigma(u)$ entsprechend das *Unifikat* von u und v.

Eigentlich betrachtet man Klassen von Wörtern aus $\Lambda/=$. So stellt sich die Frage, ob in obiger Definition statt „$\equiv$“ nicht nur „$=$“ gefordert werden sollte, was ja auf den ersten Blick eine schwächere Forderung wäre. Betrachtet man die Wörter ax und xb, so gibt es keine Lösung der Gleichung $ax \equiv xb$, wohl aber eine Lösung der Gleichung $ax = xb$. Diese Problematik führt offensichtlich in den Bereich der Unterscheidung zwischen *co-equalizer* und *equalizer*. Für weitere Referenzen in diesem Bereich sei auf [Ba91] verwiesen.

Analog zum klassischen Unifikator von zwei Termen entsteht auch hier die Frage nach allgemeinsten Unifikatoren:

Definition 5.2 (Allgemeinste Unifikatoren)

Sei σ ein Unifikator von $u, v \in \Lambda$, so heißt σ ein *allgemeinster Unifikator* von u und v genau dann, wenn für jeden weiteren Unifikator τ von u und v es eine Substiution $\lambda : \mathcal{V} \to \Lambda$ gibt, so daß $\tau \equiv \lambda \circ \sigma$.

In diesem Bereich sind schon viele Ergebnisse zusammengetragen worden. So findet man zum Beispiel in [Jaff90] einen Algorithmus zur Aufzählung aller allgemeinsten Unifikationen, der auf der Entscheidungsprozedur von Makanin basiert, die in [Maka77] dargestellt wird. Eine frühere Arbeit ist [Plot72], in der ein Verfahren zur Generierung eines Suchbaums beschrieben, der alle Lösungen eines Unifikationsproblems zweier Terme modulo der Assoziativität eines bestimmten Funktionssymbols aufzählt.

In dieser Arbeit wird ein ähnliches Verfahren, wie das von Plotkin betrachtet. Der Unterschied zu jener Arbeit besteht in der Beschränkung auf ein zweistelliges Funktionssymbol und die Einführung eines neutralen Elementes, wodurch weitere Übergangsregeln hinzugefügt werden müssen und der Vollständigkeitsbeweis schwieriger wird.

Eine weitere Neuerung ist die andere Sichtweise des Suchprozesses. Plotkin behandelt Such*bäume*, wogegen hier *zyklische* Strukturen betrachtet werden, was zu einem Automatenbegriff führt. Der Vorteil besteht darin, daß dadurch in manchen Fällen ein unendlicher Suchbaum auf einen endlichen Automaten abgebildet wird, und so z.B. ein Scheitern im Unendlichen bezüglich des Suchbaumes entscheidbar wird.

Das hier behandelte Semientscheidungsverfahren für die Unifikation führt durch eine kleine Abwandlung zu einem effizienten Algorithmus für das im nächsten Kapitel behandelte Matching- bzw. Filterproblem.

Ähnlich wie im Kapitel 4 dargestellt basiert das Verfahren auf Automaten, wie sie aus der Automatenheorie bekannt sind. Die Bezeichnungen seien wie dort gewählt.

Dabei wird üblicherweise die Endlichkeit von Z gefordert. Dies trifft auf die in diesem Abschnit betrachteten Automaten nicht zu, wobei aber bei den in Kapitel 7 dargestellten Automaten die Menge der aus dem Startzustand erreichbaren Zustände[1] endlich ist.

Die Menge der Zustände ist zunächst ganz Λ bzw. Λ^2. Als Startzustand wird immer das zu untersuchende Wort bzw. das Paar bestehend aus beiden zu untersuchenden Wörter gewählt. Es gibt immer nur einen Finalzustand, der wie im letzten Kapitel mit ϵ bzw. (ϵ, ϵ) beschriftet ist.

Jetzt wird die Methodik aus dem letzten Kapitel aufgegriffen und zunächst ein Automat angegeben, der alle Instanzen eines gegebenen Wortes aufzählt:[2]

Definition 5.3 (Automat zur Akzeption aller Instanzen)

Zu $r \in \Lambda,\ c \in \mathcal{S} \cup \mathcal{V} \cup \{\epsilon\},\ x \in \mathcal{V},\ \sigma : \mathcal{V} \to \Lambda$ definiere

$$\to \subseteq\ \Lambda \times (\mathcal{V} \cup \mathcal{S} \cup \{\epsilon\}) \times \Lambda^{\mathcal{V}} \times \Lambda$$

als die kleinste Relation, die folgenden Bedingungen genügt:

Vergleich:	cr	$\xrightarrow[c]{\sigma}$	r	falls $c \neq \epsilon,\ \sigma \equiv \{\}$.
ϵ-Substitution:	xr	$\xrightarrow[\epsilon]{\sigma}$	$\sigma(r)$	falls $\sigma \equiv \{x/\epsilon\}$.
Aufspaltung:	xr	$\xrightarrow[c]{\sigma}$	$x\sigma(r)$	falls $c \not\equiv x$, $\sigma \equiv \{x/cx\}$.

Definiert man nun, wie bei der Unifikation in Definition 5.5, die Sequenz akkumulierter Wörter und auch „$\Rightarrow$“, so erhält man für alle $u \in \Lambda,\ \sigma : \mathcal{V} \to \Lambda$ mit $\sigma\,|_{\mathcal{V} \setminus \mathcal{V}_u} \equiv \mathsf{id}$:

$$u \underset{\sigma(u)}{\overset{\sigma}{\Longrightarrow}} \epsilon$$

Der Beweis wird nicht geführt. Er läßt sich aber auf ähnliche Weise wie bei der Unifikation oder beim Matching bewerkstellen.

Aus diesem Automaten erhält man nun durch Kombination der einzelnen Regeln, genauso wie bei der Schnittbildung von endlichen Automanten, die folgende Zustandsrelation, die zu einem Automaten zur Akzeption aller (allgemeinsten) Unifikate zweier Wörter gehört.

[1] Im Englischen „*admissible part*“.
[2] Es wird nur die Zustandsrelation angegeben. Den Rest entnehme man dem letzten Absatz.

Definition 5.4 (Zustandübergangsrelation $\rightarrow$)

Zu $r, s \in \Lambda,\ c \in \mathcal{S} \cup \mathcal{V} \cup \{\epsilon\},\ x \in \mathcal{V},\ \sigma : \mathcal{V} \rightarrow \Lambda$ definiere

$$\rightarrow \subseteq\ \Lambda^2 \times (\mathcal{V} \cup \mathcal{S} \cup \{\epsilon\}) \times \Lambda^{\mathcal{V}} \times \Lambda^2$$

als die kleinste Relation, die folgenden Bedingungen genügt:

Vergleich:	(cr, cs)	$\xrightarrow[c]{\sigma}$	(r, s)	falls $c \not\equiv \epsilon,\ \sigma \equiv \{\}$.
ϵ-Substitution:	(xr, s)	$\xrightarrow[\epsilon]{\sigma}$	$(\sigma(r), \sigma(s))$	falls $\sigma \equiv \{x/\epsilon\}$.
Aufspaltung:	(xr, s)	$\xrightarrow[c]{\sigma}$	$(x\sigma(r), \sigma(s))$	falls $c \notin \{x, \epsilon\}, \sigma \equiv \{x/cx\}$.
Symmetrie:	(u, v)	$\xrightarrow[c]{\sigma}$	(r, s)	falls $(v, u) \xrightarrow[c]{\sigma} (r, s)$.

Um nicht zuviele Symbole einführen zu müssen, werden wieder dieselben Pfeile, wie in der letzten Definition benützt.

Zum Automatenbegriff gehört auch immer die Definition der Akzeptierten Sprache. Da hier zusätzlich zur Akzeption eines Unifikats auch die unifizierende Substitution generiert werden soll, ist dies hier etwas umständlicher.

Definition 5.5 (Sequenz akkumulierter Wörter)

Zu $n \in \mathbb{N},\ \sigma_1, \ldots, \sigma_n : \mathcal{V} \rightarrow \Lambda,\ c_1, \ldots, c_n \in (\mathcal{V} \cup \mathcal{S} \cup \{\epsilon\}),\ (u, v), q_0, q_1, \ldots, q_n, (r, s) \in \Lambda^2$, mit

$$(u, v) \equiv q_0 \xrightarrow[c_1]{\sigma_1} q_1 \xrightarrow[c_2]{\sigma_2} \cdots \xrightarrow[c_n]{\sigma_n} q_n \equiv (r, s),$$

definiere die dazugehörige *Sequenz akkumulierter Wörter* (**SaW**) $w_1, \ldots, w_n \in \Lambda$ rekursiv durch die Gleichungen $w_1 :\equiv c_1,\ w_i :\equiv \sigma_i(w_{i-1})c_i$, für $i = 2 \ldots n$, und schreibe mit $\tau :\equiv \sigma_n \circ \cdots \circ \sigma_1$:

$$(u, v) \xRightarrow[w_n]{\tau} (r, s)$$

Speziell für $n = 0$ also $(u, v) \xRightarrow[\epsilon]{\{\}} (u, v)$. Schreibe $n := |\Rightarrow|$.

Durch die rekursive Definition der w_i wird die zum Schluß neu generierte Teilsubstitution σ_i auf das schon akzeptierte Wort w_i angewendet, um das Wort w_{i+1} zu erhalten. Dies stellt einen wesentlichen Unterschied zur Definition der akzeptierten Sprache im Bereich der endlichen Automaten dar. Dort wird das schon akzeptierte Teilwort unverändert übernommen. Inwieweit sich das (bei einer endlichen Zustandsmenge) auf die Einordnung in die Chomsky-Hierachie auswirkt, steht noch offen.

Die Abbildung auf Seite 48 beschreibt die logische Fortführung der Beispiele aus Abbildung 5.2 und 5.3. Hier werden die Wörter $axbx$ und $yayb$ unifiziert. Dabei wird das im Englischen „*co-equalizer problem*“ genannte genannte Problem für das letzte Beispiel aus Abbildung 5.3 gelöst. Hierbei sind alle offensichtlichen Sackgassen, die sich aus dem Lemma 4.6 und dem Korollar 4.5 ergeben, nicht mehr weiter aufgeführt. Es gibt nur einen endlichen Pfad vom Startzustand zum Zielzustand, dem

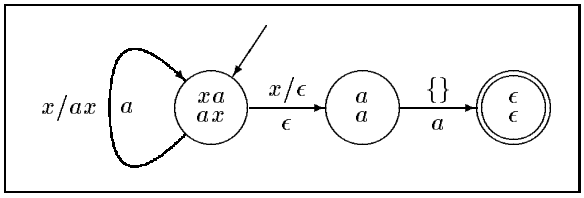


Abbildung 5.1: Dies ist ein Automat dessen SaW alle Unifikate von den Wörtern ax und xa ergeben. Die Unifikate sind also alle Wörter ungleich ϵ bestehend nur aus dem Buchstaben a. Dies ist also ein Beispiel dafür, daß das Verfahren eine endliche Repräsentation einer unendlichen Lösungsmenge liefern kann.

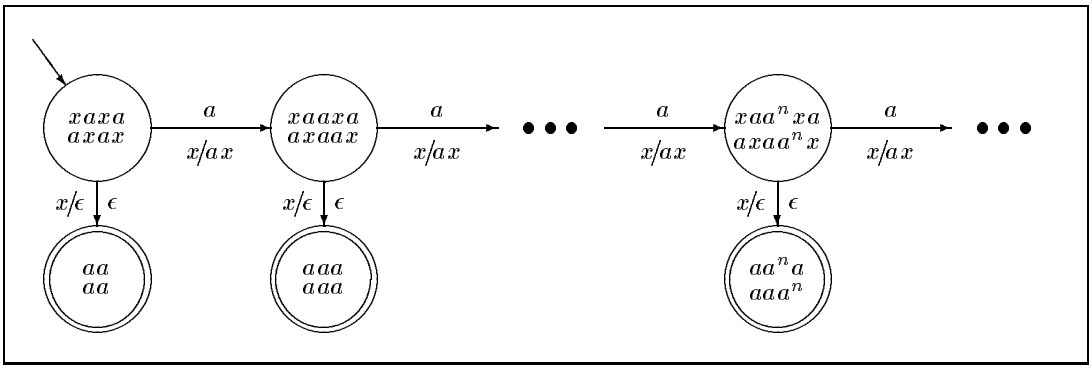


Abbildung 5.2: Hier wird ein Automat mit unendlich vielen Zuständen dargestellt, der die Unifikate der Wörter $axax$ und $xaxa$ akzeptiert. Trotz der unendlichen Anzahl von Zuständen läßt sich die Menge der Unifikate einfach ablesen. Es ist dies nämlich genau die Menge aller geradzahligen Wörter, die nur aus dem Buchstaben a bestehen — also $\{a^{2n} \mid n > 0\}$ —, wie durch ein einfaches Induktionsargument unter Zuhilfenahme des Vollständigkeits- und Korrektheitssatzes gezeigt werden kann.

das Unifikat ab entspricht. Alle andere Pfade sind nicht erfolgreich und somit ist das Wort ab die einzige Unifikation von $axba$ und $yayb$.

Das w_n aus obiger Definition wurde rekursiv von links nach rechts definiert, so daß man auf einfachste Weise beim Anhängen eines neuen Zustandüberganges am rechten Ende w_{n+1} bekommt. Soll der neue Zustandsübergang aber am linken Ende angehängt werden, dann ist das nicht so einfach, und folgendes Lemma, das später im Beweis der Vollständigkeit benötigt wird, hilft weiter.

Lemma 5.6 Aus $(u,v) \xrightarrow[c]{\sigma} (u',v') \overset{\tau}{\underset{w}{\Longrightarrow}} (r,s)$ folgt $(u,v) \overset{\tau \circ \sigma}{\underset{\tau(c)w}{\Longrightarrow}} (r,s)$.

Beweis: (Induktion über $n = |\Rightarrow|$)

$\underline{n = 0:}$ $\tau \equiv \mathsf{id} \equiv \{\}$, $w \equiv \epsilon$, $u' \equiv r$, $v' \equiv s$ und damit ist $(u,v) \overset{\sigma \circ \{\}}{\underset{c\epsilon}{\Longrightarrow}} (r,s)$.

$\underline{n \rightsquigarrow n+1:}$ Gelte

$$(u,v) \overset{\sigma}{\underset{c}{\Longrightarrow}} (u',v') \equiv q_0 \overset{\sigma_1}{\underset{c_1}{\Longrightarrow}} q_1 \longrightarrow \cdots \xrightarrow[c_n]{\sigma_n} q_n \xrightarrow[c_{n+1}]{\sigma_{n+1}} (r,s),\ \tau \equiv \sigma_n \circ \cdots \circ \sigma_1.$$

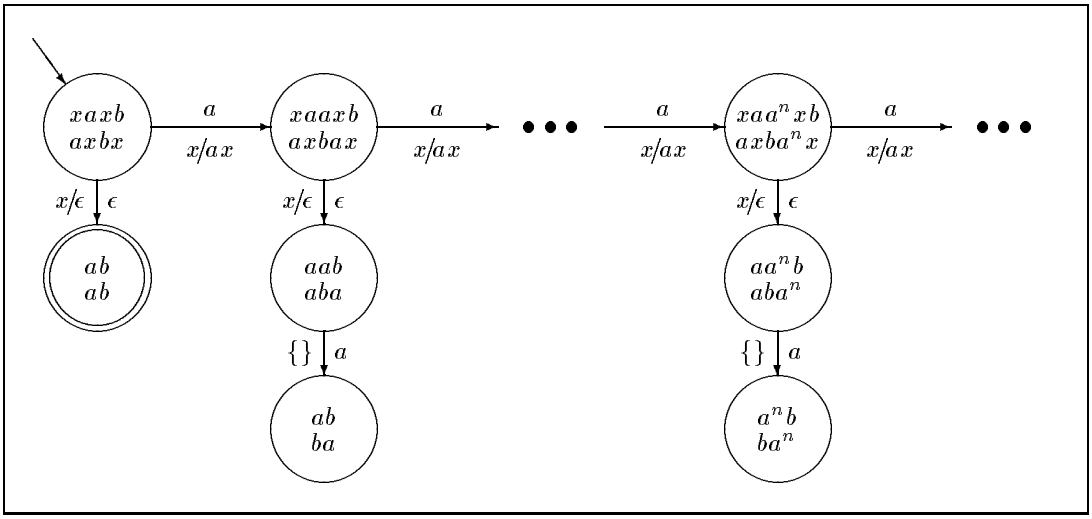


Abbildung 5.3: Dies ist ein gegenüber dem letzten Beispiel leicht modifizierter Automat. Hier werden die Unifikate von $axbx$ und $xaxb$ berechnet. Dabei findet man nur ein Unifikat, nämlich ab, wo die einzige Variable x durch ϵ ersetzt wird. Ansonsten landet man nur in einer Sackgasse $(a^n b, ba^n)$.

Mit (w_i) als **SaW** liefert die Induktionsbehauptung $(u,v) \overset{\tau\circ\sigma}{\underset{\tau(c)w_n}{\Longrightarrow}} q_n \overset{\sigma_{n+1}}{\underset{c_{n+1}}{\longrightarrow}} (r,s)$. Mit der Definition von $\Rightarrow$ ergibt das:

$$(u,v) \overset{(\sigma_{n+1}\circ\tau)\circ\sigma}{\underset{(\sigma_{n+1}\circ\tau)(c)\underbrace{\sigma_{n+1}(w_n)c_{n+1}}_{\equiv w_{n+1}}}{\Longrightarrow}} (r,s)$$

■

Im folgenden wird wieder von Vollständigkeit und Korrektheit gesprochen. Hier ist damit gemeint, daß ein Verfahren zur Lösung eines Problems einmal, falls überhaupt eine Lösung existiert, diese auch generiert und andererseits auch keine falsche Lösung ausgibt.

Satz 5.7 (Korrektheit der „⇒“-Regeln)

Sei $u, v \in \Lambda$, so gilt für alle $w \in \Lambda$, $\tau : \mathcal{V} \to \Lambda$:

$$\text{Aus } (u,v) \overset{\tau}{\underset{w}{\Longrightarrow}} (\epsilon,\epsilon) \text{ folgt } w \equiv \tau(u) \equiv \tau(v)$$

Beweis: Setze in Lemma 5.8 $(r,s) :\equiv (\epsilon,\epsilon)$. ■

Was die Vollständigkeit betrifft, beschränkt man sich auf zumindest allgemeinste Unifakionen, und erhält folgende Formulierung:

Lemma 5.8

Aus $(u,v) \xrightarrow[c_1]{\sigma_1} q_1 \to \cdots \to q_{n-1} \xrightarrow[c_n]{\sigma_n} (r,s)$ mit zugehöriger **SaW** $w_1, \ldots, w_n$ folgt:

$$\sigma(u) \equiv w_n r, \sigma(v) \equiv w_n s \text{ für } \sigma \equiv \sigma_n \circ \cdots \circ \sigma_1$$

Beweis: Der Beweis wird geführt durch Induktion über n: Im Induktionsanfang für $n = 0$ gilt $w_0 \equiv \epsilon,\ \sigma \equiv \mathsf{id},\ r \equiv u,\ s \equiv v$ und die zwei Gleichungen stimmen.

Im Induktionsschritt wird eine Fallunterscheidung nach dem letzten Zustandsübergang getroffen.

(A) Es sei also

$$(u,v) \xrightarrow[c_1]{\sigma_1} \cdots \xrightarrow[c_n]{\sigma_n} (cr, cs) \xrightarrow[c_{n+1}]{\sigma_{n+1}} (r,s)$$

und o.B.d.A. sei der $n+1$-Übergang ein Vergleich. Damit folgt $\sigma_{n+1} \equiv \{\}$ und $c \equiv c_{n+1} \not\equiv \epsilon$. Mit $\sigma :\equiv \sigma_n \circ \cdots \circ \sigma_1$ liefert die Induktionsbehauptung:

$$\begin{array}{lllllll}
(\sigma_{n+1} \circ \sigma)(u) & \equiv & \sigma(u) & \equiv & w_n cr & \equiv & \underbrace{\sigma_{n+1}(w_n)c}_{w_{n+1}}\, r \quad \equiv \quad w_{n+1} r \\
(\sigma_{n+1} \circ \sigma)(v) & \equiv & \sigma(v) & \equiv & w_n cs & \equiv & \underbrace{\sigma_{n+1}(w_n)c}_{w_{n+1}}\, s \quad \equiv \quad w_{n+1} s
\end{array}$$

(B) Jetzt gelte

$$(u,v) \xrightarrow[c_1]{\sigma_1} \cdots \xrightarrow[c_n]{\sigma_n} (xr, s) \xrightarrow[c_{n+1}]{\sigma_{n+1}} (\sigma_{n+1}(r), \sigma_{n+1}(s))$$

und o.B.d.A. ist der $n+1$-Übergang eine ϵ-Substitution. Dann ist $\sigma_{n+1} \equiv \{x/\epsilon\}$ und $c_{n+1} \equiv \epsilon$. Wieder definiere $\sigma :\equiv \sigma_n \circ \cdots \circ \sigma_1$ und mit der Induktionsbehauptung rechnet man nach:

$$\begin{array}{lllll}
(\sigma_{n+1} \circ \sigma)(u) & \equiv & \sigma_{n+1}(\sigma(u)) & \overset{\text{Ind.-Beh.}}{\equiv} & \sigma_{n+1}(w_n xr) \\
 & \equiv & \underbrace{\sigma_{n+1}(w_n)\epsilon}_{w_{n+1}}\, \sigma_{n+1}(r) & \equiv & w_{n+1}\sigma_{n+1}(r) \\
(\sigma_{n+1} \circ \sigma)(v) & \equiv & \sigma_{n+1}(\sigma(v)) & \overset{\text{Ind.-Beh.}}{\equiv} & \sigma_{n+1}(w_n s) \\
 & \equiv & \underbrace{\sigma_{n+1}(w_n)\epsilon}_{w_{n+1}}\, \sigma_{n+1}(s) & \equiv & w_{n+1}\sigma_{n+1}(s)
\end{array}$$

(C) Zum Schluß sei

$$(u,v) \xrightarrow[c_1]{\sigma_1} \cdots \xrightarrow[c_n]{\sigma_n} (xr, cs) \xrightarrow[c_{n+1}]{\sigma_{n+1}} (x\sigma_{n+1}(r), \sigma_{n+1}(s))$$

und o.B.d.A. sei der $n+1$-Übergang eine Aufspaltung. Dann ist $\sigma_{n+1} \equiv \{x/cx\}$ und $c_{n+1} \equiv c \not\equiv \epsilon$. Wieder definiere $\sigma :\equiv \sigma_n \circ \cdots \circ \sigma_1$ und mit der Induktionsbehauptung rechnet man nach:

$$\begin{array}{lllll}
(\sigma_{n+1} \circ \sigma)(u) & \equiv & \sigma_{n+1}(\sigma(u)) & \overset{\text{Ind.-Beh.}}{\equiv} & \sigma_{n+1}(w_n xr) \\
 & \equiv & \underbrace{\sigma_{n+1}(w_n)c}_{w_{n+1}}\, x\sigma_{n+1}(r) & \equiv & w_{n+1} x\sigma_{n+1}(r) \\
(\sigma_{n+1} \circ \sigma)(v) & \equiv & \sigma_{n+1}(\sigma(v)) & \overset{\text{Ind.-Beh.}}{\equiv} & \sigma_{n+1}(w_n cs) \\
 & \overset{c \not\equiv x}{\equiv} & \underbrace{\sigma_{n+1}(w_n)c}_{w_{n+1}}\, \sigma_{n+1}(s) & \equiv & w_{n+1}\sigma_{n+1}(s)
\end{array}$$

Der Fall für die Symmetrie ist offensichtlich erfüllt. ■

Satz 5.9 (Vollständigkeit der „⇒"-Regeln)

Sei $u, v \in \Lambda$, so gilt für alle $\sigma : \mathcal{V} \to \Lambda$:

$$\text{Aus } \sigma(u) \equiv \sigma(v) \text{ folgt } \exists\ \lambda, \tau : \mathcal{V} \to \Lambda : (u, v) \underset{\tau(u)}{\overset{\tau}{\Longrightarrow}} (\epsilon, \epsilon) \text{ und } \sigma \equiv \lambda \circ \tau$$

Beweis: Der Beweis wird geführt durch Induktion über zwei Parameter, wobei das Induktionsschema aus A.6 benützt wird mit $U := \{(u, v, \sigma) \mid u, v \in \Lambda,\ \sigma : \mathcal{V} \to \Lambda\}$, $\mathsf{f}(u, v, \sigma) := |\sigma(u)| =: n$ und $\mathsf{g}(u, v, \sigma) := |u| + |v| =: m$. Das Prädikat $\mathcal{A}$ über U sei definiert durch:

$$\mathcal{A}(u, v, \sigma) \ :\Longleftrightarrow\ (\ \sigma(u) \equiv \sigma(v)\ \Rightarrow\ \exists \lambda, \tau : \mathcal{V} \to \Lambda : (u, v) \underset{\tau(u)}{\overset{\tau}{\Longrightarrow}} (\epsilon, \epsilon)\ )$$

Im Induktionsanfang ist $(m, n) = (0, 0)$. Nach Definition von n und m ist $\sigma(u) \equiv \sigma(v) \equiv u \equiv v \equiv \epsilon$ und mit $\tau := \mathsf{id},\ \lambda := \sigma$ folgt die Gültigkeit von $\mathcal{A}(u, v, \sigma)$.

Im Induktionsschritt gilt $\mathcal{A}(u', v', \sigma')$ für alle $(u', v', \sigma') \in U$ mit $|u'| < n$ oder $|u'| = n$ und $|u'| + |v'| < m$. Und es ist zu zeigen, daß $\mathcal{A}(\overline{u}, \overline{v}, \overline{\sigma})$ für alle $(\overline{u}, \overline{v}, \overline{\sigma}) \in U$ mit $|\overline{u}| \leq n,\ |\overline{v}| \leq m$ gilt..

Sei also o.B.d.A. $\sigma(u) \equiv \sigma(v)$ und $|\sigma(u)| = n,\ |u| + |v| = m$. Aus Symmetriegründen und wegen $(n, m) \neq (0, 0)$ kann man $|u| > 0$ wählen. Es folgt eine Fallunterscheidung nach den Anfangsbuchstaben von u und v.

(A) $u \equiv cr,\ v \equiv \epsilon,\ c \in \mathcal{V},\ r \in \Lambda$ und es muß gelten $\sigma(c) \equiv \sigma(u) \equiv \epsilon$. Dann ist

$$\sigma(\epsilon) \equiv \sigma(u) \equiv \sigma(c)\sigma(r) \equiv \sigma(\{c/\epsilon\}(r))$$

und für die Beträge gilt[3]:

$$\begin{array}{lllllllll} |\sigma(\{c/\epsilon\}(r))| & = & |\sigma(r)| & & & = & |\sigma(u)| & = & n \\ |\{c/\epsilon\}(r)| + |\{c/\epsilon\}(\epsilon)| & \leq & |r| + 0 & < & |cr| & = & |u| & = & |u| + |v| \quad = \quad m \end{array}$$

Somit ist die Induktionsvoraussetzung anwendbar und es gilt:

$$(u, v) \equiv (xr, \epsilon) \underset{\epsilon}{\overset{\{c/\epsilon\}}{\longrightarrow}} (\{c/\epsilon\}(r), \epsilon) \underset{\tau}{\overset{\tau(\{c/\epsilon\}(r))}{\Longrightarrow}} (\epsilon, \epsilon)$$

wobei $\tau, \lambda : \mathcal{V} \to \Lambda$ und $\sigma \equiv \lambda \circ \tau$. Mit $\sigma \circ \{c/\epsilon\} \equiv \sigma$ und Lemma 5.6 folgt schließlich

$$(u, v) \underset{(\tau \circ \{c/\epsilon\})(cr)}{\overset{\tau \circ \{c/\epsilon\}}{\Longrightarrow}} (\epsilon, \epsilon)$$

und es gilt zusätzlich $\sigma \equiv \lambda \circ \tau \circ \{c/\epsilon\}$.

(B) Jetzt sei $u \equiv cr,\ v \equiv cs$ mit $c \in \mathcal{S} \cup \mathcal{V}$ und $r, s \in \Lambda$. Es gilt $|r| < |u| = n$ und $\sigma(r) \equiv \sigma(s)$. Deshalb gibt es nach der Induktionsvoraussetzung $\sigma, \tau : \mathcal{V} \to \Lambda$, so daß $\sigma \equiv \lambda \circ \tau$ und

$$(u, v) \equiv (cr, cs) \underset{c}{\overset{\{\}}{\longrightarrow}} (r, s) \underset{\tau(r)}{\overset{\tau}{\Longrightarrow}} (\epsilon, \epsilon)$$

Lemma 5.6 ergibt dann

$$(u, v) \underset{\tau(cr)}{\overset{\tau}{\Longrightarrow}} (\epsilon, \epsilon)$$

(C) $u \equiv cr,\ v \equiv ds,\ c,d \in \mathcal{S},\ r,s \in \Lambda,\ c \not\equiv d$ führt sofort zu einem Widerspruch mit $\sigma(u) \equiv \sigma(v)$:

$$\sigma(u) \equiv \sigma(cr) \equiv c\sigma(r) \not\equiv d\sigma(s) \equiv \sigma(ds)\sigma(v)$$

(D) $u \equiv xr,\ v \equiv cs,\ c \in \mathcal{S} \cup \mathcal{V},\ x \in \mathcal{V},\ r,s \in \Lambda$ und $c \not\equiv x$ (sonst siehe (B)). Diese Variable x kann nun einmal nach ϵ abgebildet werden, oder sie wird durch ein Wort der Länge größer gleich eins ersetzt:

(D.1) Es sei $\sigma(x) \equiv \epsilon$ und wegen $\sigma \equiv \sigma \circ \{x/\epsilon\}$ ist $\sigma(\{x/\epsilon\}(r)) \equiv \sigma(r) \equiv \sigma(\{x/\epsilon\}(cs))$ und für die Beträge gilt:

$$\begin{array}{lllllllll} |\sigma(\{x/\epsilon\}(r))| & = & |\sigma(r)| & = & |\sigma(xr)| & = & |\sigma(u)| & = & n \\ |\{x/\epsilon\}(r)| + |\{x/\epsilon\}(cs)| & \leq & |r| + |cs| & < & |xr| + |cs| & = & |u| + |v| & = & m \end{array}$$

Die Induktionsvoraussetzung liefert dann $\tau, \lambda : \mathcal{V} \to \Lambda$, so daß $\sigma \equiv \lambda \circ \tau$ und

$$(u,v) \equiv (xr,cs) \xrightarrow[\epsilon]{\{x/\epsilon\}} (\{x/\epsilon\}(r), \{x/\epsilon\}(cs)) \xRightarrow[\tau(\{x/\epsilon\}(r))]{\tau} (\epsilon,\epsilon)$$

und mit Lemma 5.6 erhält man dann

$$(u,v) \xRightarrow[(\tau \circ \{x/\epsilon\})(xr)]{\tau \circ \{x/\epsilon\}} (\epsilon.\epsilon)$$

und zusätzlich gilt $\sigma \circ \{x/\epsilon\} \equiv \lambda \circ (\tau \circ \{x/\epsilon\})$.

(D.2) Jetzt sei $\sigma(x) \equiv \sigma(c)t$ mit $t \in \Lambda$. Darüber hinaus gelte $\sigma(c) \not\equiv \epsilon$ (Ansonsten betrachte man Fall (D.1) und einen Symmetrieübergang). Zuerst definiere man $\sigma' : \mathcal{V} \to \Lambda$ durch

$$\sigma'(y) :\equiv \begin{cases} \sigma(y) & \text{falls } y \not\equiv x \\ t & \text{falls } y \equiv x \end{cases}, \text{ für alle } y \in \mathcal{V}$$

Die Voraussetzung $c \not\equiv x$ ergibt $\sigma' \equiv \sigma \circ \{x/cx\}$. Weiter gilt dann

$$\sigma'(x\{x/cx\}(r)) \equiv \sigma'(x)\sigma'(\{x/cx\}(r)) \equiv t\sigma(r)$$

Damit rechnet man für den Induktionsparameter n nach

$$|\sigma'(x\{x/cx\}(r))| = |t\sigma(r)| \overset{\sigma(c) \not\equiv \epsilon}{<} |\sigma(c)t\sigma(r)| = |\sigma(x)\sigma(r)| = |u| = n$$

und es gelten die beiden Gleichungen

$$\begin{array}{lllllllll} \sigma(u) & \equiv & \sigma(xr) & \equiv & \sigma(c)t\sigma(r) & \equiv & \sigma(c)t\sigma'(\{x/cx\}(r)) \\ \sigma(v) & \equiv & \sigma(cs) & \equiv & \sigma(c)\sigma(s) & \equiv & \sigma(c)\sigma'(\{x/cx\}(s)) \end{array}$$

Mit der Voraussetzung $\sigma(u) \equiv \sigma(v)$ ergibt dies

$$\sigma'(x\{x/cx\}(r)) \equiv t\sigma'(\{x/cx\}(r)) \equiv \sigma'(\{x/cx\}(s))$$

Damit sind alle Bedingungen für die Anwendung der Induktionsvoraussetzung auf $(x\{x/cx\}(r), \{x/cx\}(s), \sigma') \in U$ erfüllt und man erhält $\sigma, \tau : \mathcal{V} \to \Lambda$ mit $\sigma' \equiv \lambda \circ \tau$ und

$$(u,v) \equiv (xr,cs) \xrightarrow[c]{\{x/cx\}} (x\{x/cx\}(r), \{x/cx\}(s)) \xRightarrow[\tau(x\{x/cx\}(r))]{\tau} (\epsilon,\epsilon)$$

[3] Dies ist zusammen mit Fall (D.1) die einzige Stelle, an der das erweiterte Induktionsschema benutzt werden muß. In allen anderen Fällen würde eine einfache Induktion über den Parameter $n := |\sigma(u)|$ genügen.

Wiederum bekommt man dann aus Lemma 5.6

$$(u,v) \xRightarrow[\tau(c)\tau(x\{x/cx\}(r))]{\tau\circ\{x/cx\}} (\epsilon,\epsilon)$$

Einmal gilt für die Substitutionen

$$\sigma \equiv \sigma' \circ \{x/cx\} \equiv \lambda \circ (\tau \circ \{x/cx\})$$

und desweiteren für das akzeptierte Wort

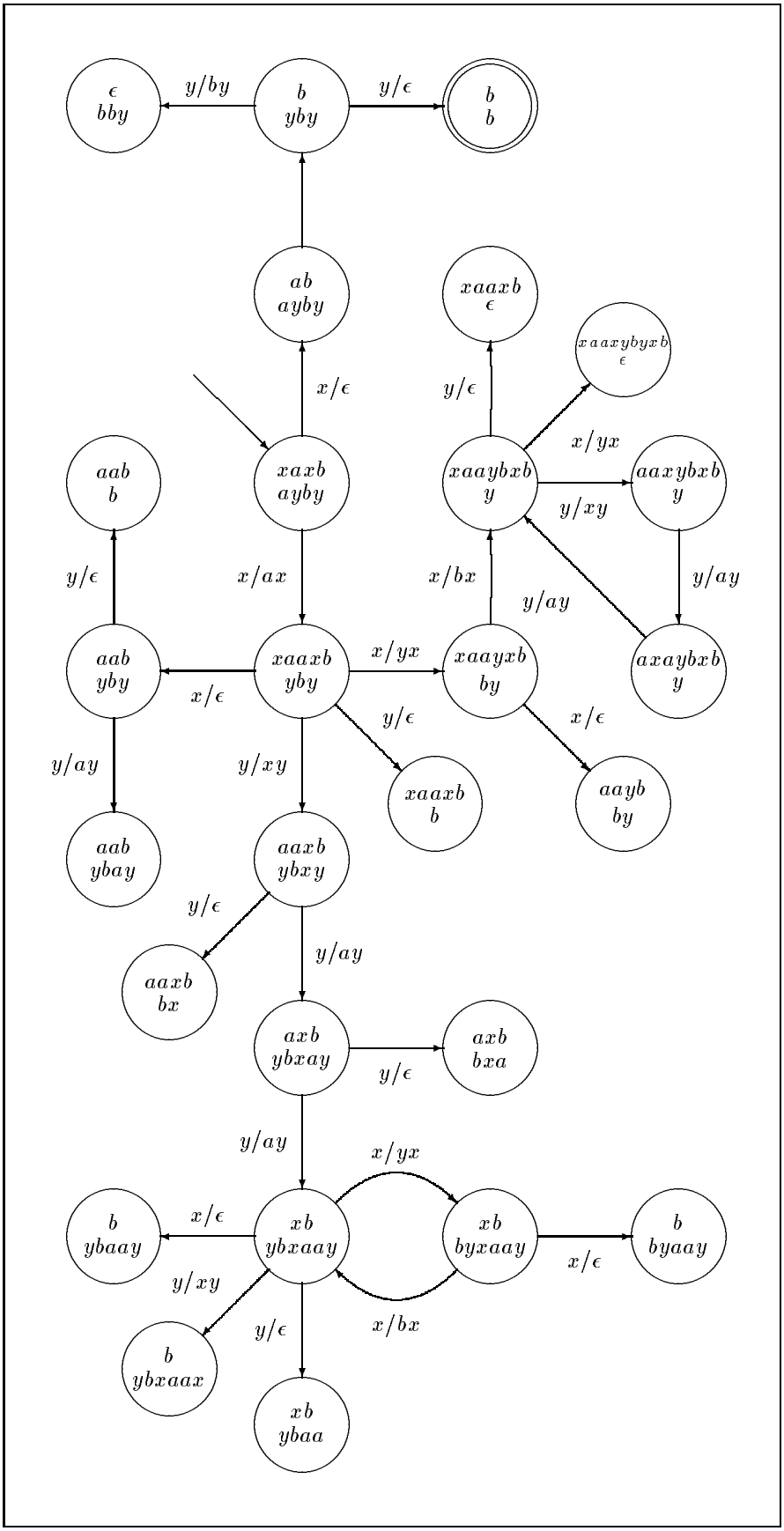
$$\begin{array}{lcll} \tau(c)\tau(x\{x/cx\}(r)) & \equiv & \tau(cx\{x/cx\}(r)) & \equiv \quad \tau(\{x/cx\}(x)\{x/cx\}(r) \\ & \equiv & (\tau\circ\{x/cx\})(xr) & \equiv \quad (\tau\circ\{x/cx\})(u) \end{array}$$

Gilt $\sigma(c) \equiv \sigma(x)t$ mit einem $t \in \Lambda\backslash\{\epsilon\}$, so ist entweder $\sigma(x) \equiv \epsilon$ und man landet bei Fall (D.1) oder es ist $c \in \mathcal{V}$ und aus Symmetriegründen reicht Fall (D.2).

Damit sind alle Fälle abgedeckt und der Beweis ist beendet. ■

Man beachte, daß die Induktionsparameter symmetrisch bezüglich u und v sind, da sowohl $u{+}v{=}v{+}u$ als auch $|\sigma(u)| = |\sigma(v)|$ gilt.

Es bleiben noch ein paar Fragen offen. Wenn der erreichbare Teil dieser Automaten endlich ist, ist dann die akzeptierte Sprache, also die Menge der Unifkate, regulär, und wenn ja, ist es vielleicht sogar so, daß die Lösung dieses Unifikationsproblems immer regulär ist ?


ϵ
bby
y/by
b
yby
y/ϵ
b
b
ab
ayby
x/ϵ
xaaxb
ϵ
xaaxybyxb
ϵ
y/ϵ
x/yx
aab
b
xaxb
ayby
xaaybxb
y
aaxybxb
y
y/xy
y/ϵ
x/ax
x/bx
y/ay
y/ay
aab
yby
xaaxb
yby
x/yx
xaayxb
by
axaybxb
y
x/ϵ
y/ϵ
x/ϵ
y/ay
y/xy
xaaxb
b
aayb
by
aab
ybay
aaxb
ybxy
y/ϵ
aaxb
bx
y/ay
axb
ybxay
axb
bxa
y/ϵ
y/ay
x/yx
b
ybaay
x/ϵ
xb
ybxaay
xb
byxaay
b
byaay
x/ϵ
y/xy
x/bx
y/ϵ
b
ybxaax
xb
ybaa

Kapitel 6

Matching

Das Thema dieses Kapitels ist das *Matching* bzw. die *Filterung* wie es in [Pott89] genannt wird. Dies ist die Frage danach, ob ein Wort Instanz eines anderen Wortes ist, also kleiner ist bezüglich der Subsumptionsordnung.

6.1 Definition

Definition 6.1 (Matching)

Zwei Wörter mit Variablen $u, v \in \Lambda$ heißen *anpaßbar*, v *matcht* u , v ist eine *Instanz* von u oder u ist ein *Filter* für v genau dann, wenn es ein $\sigma: \mathcal{V} \to \Lambda$ gibt, so daß $\sigma(u) \equiv v$. Ein solches σ, falls es existiert, heißt *Matching* von v mit u oder auch *angleichende Substitution* .

Ein Vergleich mit 2.6.2, ergibt daß v matcht[1] u genau dann, wenn $u \leq v$.

6.2 Algorithmus

Das nun vorgestellte Verfahren zum Matchen zweier Wörter basiert auf dem im letzten Kapitel dargestellten Verfahren zur Unifikation zweier Wörter. Es stellt eine Spezialisierung im folgenden Sinne dar. Matching kann immer aufgefaßt werden als eine einseitige Unifikation, bei der die Variablen des einen beteiligten Wortes nicht eingeschränkt werden dürfen, gewissermaßen wie Konstanten behandelt werden. Um dies technisch in den Griff zu bekommen, wird das Alphabet erweitert und eine Variablenmenge $\mathcal{V}'$ eingeführt, aus der o.B.d.A. die Variablen der zu testenden Instanz stammen. Der zu testende Filter bleibt aber ein Wort über dem alten Alphabet.

[1] Das englische Verb „*match*“ läßt sich wohl am besten mit „*passen zu*“ oder „*zusammenpassen*“ übersetzen. Dabei scheinen im Deutschen wie auch im Englischen Objekt und Subjekt vertauschbar, obwohl in unserem Sinne die Beziehung auf keinen Fall symmetrisch sein sollte. Darüber hinaus läßt sich ein englisches Verb in deutschen Texten schlecht konjugieren und so wird im Folgenden von der Sprechweise „u matcht v“ Abstand genommen. Stattdessen wird die sprachliche Konstruktion mit einem der Substantive „Instanz“ oder „Filter“ benützt, wobei aber ausdrücklich drauf hingewiesen werden soll, daß deren Verwendung unsymmetrisch ist. Es gilt nämlich, u ist ein Filter von v genau dann, wenn v eine Instanz von u ist.

Definition 6.2 (Fixierung von Variablen)

Zu $\mathcal{V}$ wähle eine beliebige aber fixe Menge $\mathcal{V}'$, mit $\mathcal{V}' \cap (\mathcal{S} \cup \mathcal{V}) = \emptyset$, so daß es eine *Bijektion* $\iota : \mathcal{V} \to \mathcal{V}'$ gibt. Für alle $x \in \mathcal{V}$ definiere $x' := \iota(x)$.

Statt nun zu testen, ob $u \in (\mathcal{S} \cup \mathcal{V})^*$ ein Filter von $v \in (\mathcal{S} \cup \mathcal{V})^*$ ist, betrachtet man das Problem, ob ein $u \in (\mathcal{S} \cup \mathcal{V})^*$ ein Filter für $v \in (\mathcal{S} \cup \mathcal{V}')^*$ ist. Für eine angleichende Substitution σ erreicht man so, daß für jede Variable $x \in \mathcal{V}_u$ gilt $\sigma(x) \in (\mathcal{S} \cup \mathcal{V}')^*$.

Damit läßt sich wie im vorigen Kapitel ein Regelsystem oder, wenn man so will, ein Automat für Wortpaare aus $(\mathcal{S} \cup \mathcal{V} \cup \mathcal{V}')^2$ definieren, der sich von dem dort definierten insofern unterscheidet, daß die Zeichen aus $\mathcal{V}'$ wie Konstanten behandelt werden. Die durch die Regeln definierte Relation „$\Rightarrow$“ wird dabei mit demselben Symbol wie im letzten Kapitel bezeichnet, das aber nur in diesem Kapitel noch auftritt und somit eigentlich keine Verwirrung stiften sollte.

Definition 6.3 (Zustandübergangsfunktion $\to$)

Zu $r \in (\mathcal{S} \cup \mathcal{V}\mathcal{V}')^*,\ s \in (\mathcal{S} \cup \mathcal{V}')*,\ x \in \mathcal{V},\ c \in \mathcal{S} \cup \mathcal{V} \cup \mathcal{V}' \cup \{\epsilon\}$ definiere

$$\to \subseteq\ \Lambda^2 \times (\mathcal{V} \cup \mathcal{S} \cup \{\epsilon\}) \times \Lambda^{\mathcal{V}} \times \Lambda^2$$

als die kleinste Relation, die folgenden Bedingungen genügt:

Vergleich:	(cr, cs)	$\xrightarrow[c]{\sigma}$	(r, s),	falls $c \neq \epsilon,\ \sigma \equiv \{\}$.
ϵ-Substitution:	(xr, s)	$\xrightarrow[\epsilon]{\sigma}$	$(\sigma(r), \sigma(s))$,	falls $\sigma \equiv \{x/\epsilon\}$.
Aufspaltung:	(xr, s)	$\xrightarrow[c]{\sigma}$	$(x\sigma(r), \sigma(s))$,	falls $c \notin \{x, \epsilon\},\ \sigma \equiv \{x/cx\}$.

Ein kleiner Unterschied zur entsprechenden Definition bei der Unifikation besteht darin, daß man keine Symmetrie braucht, da Variablen, die substituiert werden können, nur in der ersten Komponente der Wortpaare auftreten. Die Entsprechende Definition von „$\Rightarrow$“ und der Sequenz akkumulierter Wörter wird dann wie in 5.5 definiert und Lemma 5.6 gilt hier entsprechend.

Satz 6.4 (Korrektheit und Vollständigkeit)

Zu $u, v \in \Lambda,\ \sigma : \mathcal{V} \to \Lambda$ mit $\sigma\,|_{\mathcal{V} \setminus \mathcal{V}_u} \equiv \mathsf{id}$ gilt: $\sigma(u) \equiv v$ **gdw.** $(u, \iota(v)) \xrightarrow[\iota \circ \sigma]{\iota(v)} (\epsilon, \epsilon)$

Beweis: Mit der Äquivalenz $\sigma(u) \equiv v$ **gdw.** $(\iota \circ \sigma)(u) \equiv \iota(v)$ reicht es aus für $u \in \Lambda,\ v \in (\mathcal{S} \cup \mathcal{V}')^*,\ \sigma : \mathcal{V} \to (\mathcal{S} \cup \mathcal{V} \cup \mathcal{V}')^*$ mit $\sigma\,|_{\mathcal{V} \setminus \mathcal{V}_u} \equiv \mathsf{id}$ folgendes zu zeigen:

$$\sigma(u) \equiv v \textbf{ gdw. } (u, v) \overset{v}{\underset{\sigma}{\Longrightarrow}} (\epsilon, \epsilon) \tag{6.1}$$

Für die Korrektheit, das heißt die Richtung von rechts nach links, setzt man im Lemma 6.5 $(r, s) :\equiv (\epsilon, \epsilon)$ und die gewünschte Aussage steht da. Die Rückrichtung wird dann von Lemma 6.6 gezeigt. ■

Lemma 6.5

Zu $u \in (\mathcal{S} \cup \mathcal{V} \cup \mathcal{V}')^*,\ v \in (\mathcal{S} \cup \mathcal{V}')^*,\ \sigma : \mathcal{V} \to \Lambda$ gilt:

$$(u, v) \underset{w}{\overset{\sigma}{\Longrightarrow}} (r, s) \textbf{ impliziert } \sigma(u)r \equiv vs$$

Beweis: Der Beweis wird durch Induktion über die Länge von „$\Rightarrow$“ geführt. Der Induktionsanfang ist trivial. Im Induktionsschritt unterscheide nun nach den drei Fällen aus der Definition von „$\to$“:

(A) Es gelte

$$(u, v) \underset{w}{\overset{\sigma}{\Longrightarrow}} (cr, cs) \underset{c}{\overset{\{\}}{\longrightarrow}} (r, s),$$

dann folgt nach Induktionsvoraussetzung $\sigma(u) \equiv wcr$ und $\sigma(v) \equiv wcs$, womit man nachrechnet:

$$\begin{array}{lcll} (\{\} \circ \sigma)(u) & \equiv & \sigma(u) & \equiv & wcr \\ (\{\} \circ \sigma)(v) & \equiv & \sigma(v) & \equiv & wcs \end{array}$$

(B) Es gelte

$$(u, v) \underset{w}{\overset{\sigma}{\Longrightarrow}} (xr, s) \underset{\epsilon}{\overset{\{x/\epsilon\}}{\longrightarrow}} (\{x/\epsilon\}(r), s),$$

dann liefert die Induktionvoraussetzung $\sigma(u) \equiv wxr$ und $\sigma(v) \equiv ws$, womit man nachrechnet:

$$\begin{array}{lcll} (\{x/\epsilon\} \circ \sigma)(u) & \equiv & \{x/\epsilon\}(wxr) & \underset{x \notin \mathcal{V}_s \subseteq \mathcal{V}_v}{\equiv} & \{x/\epsilon\}(wr) \\ (\{x/\epsilon\} \circ \sigma)(v) & \equiv & \{x/\epsilon\}(ws) & & \{x/\epsilon\}(w)s \end{array}$$

(C) Es gelte

$$(u, v) \underset{w}{\overset{\sigma}{\Longrightarrow}} (xr, cs) \underset{c}{\overset{\{x/cx\}}{\longrightarrow}} (x\{x/cx\}(r), s),$$

dann liefert die Induktionvoraussetzung $\sigma(u) \equiv wxr$ und $\sigma(v) \equiv wcs$, womit man nachrechnet:

$$\begin{array}{lcll} (\{x/cx\} \circ \sigma)(u) & \equiv & \{x/cx\}(wxr) & \underset{x \notin \mathcal{V}_s \subseteq \mathcal{V}_v}{\equiv} & \{x/cx\}(w)cx\{x/cx\}(r) \\ (\{x/cx\} \circ \sigma)(v) & \equiv & \{x/cx\}(wcs) & & \{x/cx\}(w)cs \end{array}$$

In allen Fällen liefert dann die Definition von „$\Rightarrow$“ das gewünschte Resultat. ■

Für die Vollständigkeit betrachte man am besten nur Pfade durch den Zustandübergansgraphen, die bei (ϵ, ϵ) enden.

Lemma 6.6 (Vollständigkeit)

Zu $u \in \Lambda,\ v \in (\mathcal{S} \cup \mathcal{V}'),\ \sigma : \mathcal{V}_u \to (\mathcal{S} \cup \mathcal{V}')$ mit $\sigma(u) \equiv v$ und $\sigma \mid_{\mathcal{V} \setminus \mathcal{V}_u} \equiv \mathsf{id}$ gilt:

$$(u, v) \underset{v}{\overset{\sigma}{\Longrightarrow}} (\epsilon, \epsilon)$$

Beweis: Der Beweis wird geführt durch Induktion über die Parameter $(m,n) := (|u|,|v|)$ geordnet durch umgekehrte lexikographische Ordnung.[2] Die Induktionsbehauptung[3] lautet dann:

$$\left.\begin{array}{c} \forall u \in (\mathcal{S} \cup \mathcal{V} \cup \mathcal{V}'),\ v \in (\mathcal{S} \cup \mathcal{V}'),\ \text{mit } |v| < n \text{ oder } |v| = n \text{ und } |u| < m, \\ \text{und } \forall \sigma : \mathcal{V}_u \to (\mathcal{S} \cup \mathcal{V}'),\ \text{mit } \sigma\,|_{\mathcal{V}\setminus\mathcal{V}_u} \equiv \mathsf{id} \text{ gilt}: \\ \sigma(u) \equiv v \textbf{ impliziert } (u,v) \underset{v}{\overset{\sigma}{\Longrightarrow}} (\epsilon,\epsilon) \end{array}\right\} \tag{6.2}$$

Im Induktionsanfang ist $u \equiv v \equiv \epsilon$ und es gilt $(u,v) \underset{\epsilon}{\overset{\mathsf{id}}{\Longrightarrow}} (\epsilon,\epsilon)$. Der Fall $m = 0$ und $n > 0$ kann nicht auftreten, und es sei also zunächst $n, m > 0$. Dann gibt es $d \in \mathcal{S} \cup \mathcal{V}'$ und $s \in (\mathcal{S} \cup \mathcal{V}')^*$, so daß $v \equiv ds$.

(A) $u \equiv cr,\ c \in \mathcal{S} \cup \mathcal{V}',\ c \not\equiv d$ führt zu einem Widerspruch, wegen $\sigma(u) \equiv c\sigma(r) \not\equiv ds \equiv v$.

(B) $u \equiv dr$ ergibt mit $|r| < n$ und der Induktionsbehauptung (6.2):

$$(u,v) \underset{d}{\overset{\{\}}{\longrightarrow}} (s,r) \underset{r}{\overset{\sigma}{\Longrightarrow}} (\epsilon,\epsilon)$$

Bei der Anwendung der Induktionsbehauptung beachte man, daß $\mathcal{V}\setminus\mathcal{V}_u = \mathcal{V}\setminus\mathcal{V}_s$ und $\sigma(s) \equiv r$.

(C) $u \equiv xr,\ x \in \mathcal{V}$ führt zur Fallunterscheidung nach $\sigma(x)$:

$\underline{\sigma(x) \equiv \epsilon}$**:** Es ist $(\sigma \circ \{x/\epsilon\})(r) \equiv \sigma(xr) \equiv \sigma(u) \equiv ds$ und $|\{x/\epsilon\}(r)| \leq |r| < |u| = n$. Definiert man nun $\sigma' : \mathcal{V} \to (\mathcal{S} \cup \mathcal{V}')^*$ durch

$$\sigma'(y) :\equiv \begin{cases} \sigma(y) & \text{falls } y \not\equiv x \\ y & \text{falls } y \equiv x \end{cases} \quad \text{für alle } y \in \mathcal{V}$$

so gilt $\sigma'\,|_{\mathcal{V}\setminus\mathcal{V}_{\{x/\epsilon\}(r)}} \equiv \mathsf{id}$, $\sigma'(\{x/\epsilon\}(r)) \equiv v \equiv ds$, so daß mit der Induktionsbehauptung (6.2) folgt:

$$(u,v) \underset{\epsilon}{\overset{\{x/\epsilon\}}{\longrightarrow}} (\ \{x/\epsilon\}(r),\ ds) \underset{ds}{\overset{\sigma'}{\Longrightarrow}} (\epsilon,\epsilon)$$

Mit $\sigma' \circ \{x/\epsilon\} \equiv \sigma$ und der Definition von „$\Rightarrow$“ folgt der Rest.

$\underline{\sigma(x) \not\equiv \epsilon}$**:** Wegen $\sigma(x) \equiv ds$ gibt es ein $t \in (\mathcal{S} \cup \mathcal{V} \cup \mathcal{V}')^*$ mit $\sigma(x) \equiv dt$ und $t\sigma(r) \equiv \sigma(s)$. Es sei $\sigma' : \mathcal{V}_u \to \mathcal{V}'$ definiert durch:

$$\sigma'(y) :\equiv \begin{cases} \sigma(y) & \text{falls } y \not\equiv x \\ t & \text{falls } y \equiv x \end{cases} \quad \text{für alle } y \in \mathcal{V}$$

Damit rechnet man $(\sigma' \circ \{x/dx\})(x) \equiv \sigma'(dx) \equiv d\sigma(x) \equiv dt \equiv \sigma(x)$ nach, womit die Gleichung $\sigma \equiv \sigma' \circ \{x/dx\}$ sofort folgt. Die Variablenbedingung für σ' ist dann genau wie im letzten Fall erfüllt. Es gilt noch

$$\sigma'(x\{x/dx\}(r)) \equiv t(\sigma' \circ \{x/dx\})(r) \equiv t\sigma(r) \equiv \sigma(s)$$

und so ist mit $|s| < |ds|$ die Induktionsbehauptung anwendbar und ergibt:

$$(xr, ds) \underset{d}{\overset{\{x/dx\}}{\longrightarrow}} (\ x\{x/dx\}(r),\ s) \underset{s}{\overset{\sigma'}{\Longrightarrow}} (\epsilon,\epsilon)$$

Die Definition von „$\Rightarrow$“ ergibt den Rest.

[2] $(m_1,n_1) < (m_2,n_2)$ gelte genau genau, wenn $n_1 < n_2$ oder $n_1 = n_2$ und $m_1 < m_2$. Dies ist wieder dasselbe Induktionsschema, wie beim Beweis der Vollständigkeit bei der Unifikation.

[3] Man beachte die Verschärfung der Behauptung durch $u \in (\mathcal{S} \cup \mathcal{V} \cup \mathcal{V}')^*$.

Im Falle $n = 0$ und $m > 0$ reichen die Argumente aus **(A)** und $\sigma(x) \equiv \epsilon$, indem man dort $ds :\equiv \epsilon$ setzt, und beachtet, daß der vorderste Buchstabe von u keine Konstante sein kann. ■

Die Formulierung $\sigma : \mathcal{V}_u \to (\mathcal{S} \cup \mathcal{V}')^*$ mit $\sigma \mid_{\mathcal{V} \setminus \mathcal{V}_u} \equiv \mathsf{id}$ soll dabei folgendes bedeuten:

$$\sigma : \mathcal{V} \to \Lambda, \quad \sigma(x) \equiv \begin{cases} x & \text{falls } x \in \mathcal{V} \setminus \mathcal{V}_u \\ w & \text{falls } x \in \mathcal{V}_u, \text{ mit einem } w \in (\mathcal{S} \cup \mathcal{V}')^* \end{cases}$$

oder anders ausgedrückt σ verändert keine Variablen, die nicht in u vorkommen, und substituiert Variablen aus u immer durch Wörter über dem Alphabet mit den fixierten Variablen $\mathcal{V}'$.

Man beachte den feinen Unterschied zwischen diesem Vollständigkeitsresultat und Satz 5.9. Bei der Unifikation gibt es, wenn überhaupt eine Lösung existiert, immer unendlich viele, und es ist im allgemeinen nicht möglich alle Unifikate im Endlichen aufzuzählen. Aus diesem Grund wurde der Vollständigkeitssatz bei der Unifikation im Prinzip nur auf allgemeinste Unifikationen ausgelegt. Damit war es möglich, daß bei manchen Unifikationsproblemen der Algorithmus eine endliche Repräsentation der Lösungsmenge liefert. Hier hat man nun die Situation, daß tatsächlich alle relevanten[4] Lösungen erzeugt werden können, da dies nur endlich viele sind. Somit erhält man eine etwas stärkere Aussage, was den Aufwand des erneuten Beweises rechtfertigt.

Um aus diesen Ergebnissen einen brauchbaren Algorithmus ableiten zu können fehlt noch die Terminierung. Es ist also noch die Frage zu beantworten, ob der Zustandsübergangsgraph endlich ist, und wenn ja, wie groß ist seine Komplexität in Relation zur Länge der Eingabeworte.

Satz 6.7

Zu $u \in \Lambda$ und $v \in (\mathcal{S} \cup \mathcal{V}')^*$ gibt es maximal $2^{|u|+2\cdot|v|}$ Pfade durch den Zustandsübergangsgraphen vom Startzustand (u, v) zum Finalzustand (ϵ, ϵ).

Beweis: Die zweite Komponente der Zustände auf einem Pfad P besteht nur aus Suffixen von v. Bei jedem Zustandsübergang wird dann entweder der vorderste Buchstabe des Wortes in der zweiten Komponente weggestrichen oder eine Variable in u durch ϵ ersetzt. Da es höchstens $|u|$ Variablen in u gibt, gilt $j \leq |u|+|v|$ für die Länge j von P. Auf einem Pfad der Länge j gibt es nun maximal $\binom{j}{|v|}$ Möglichkeiten diejenigen Zustandsübergänge auszuwählen, die keine ϵ-Substitution darstellen. Es gibt dann $2^{|v|}$ Partitionen des Wortes v, wie die Buchstaben von v entweder von einer Aufspaltung oder einem Vergleich akzeptiert werden. Summiert man über alle mögliche Längen von P auf, so erhält man als obere Schranke für die Anzahl von Pfaden:

$$\sum_{j=|v|}^{|u|+|v|} \binom{j}{|v|} \cdot 2^{|v|} = 2^{|v|} \cdot \sum_{j=0}^{|u|} \underbrace{\binom{j+|v|}{|v|}}_{=\binom{j+|v|}{j+|v|-|v|}} = 2^{|v|} \cdot \sum_{j=0}^{|u|} \binom{j+|v|}{j}$$

$$\leq 2^{|v|} \cdot \sum_{j=0}^{|u|+|v|} \binom{|u|+|v|}{j} = 2^{|v|} \cdot 2^{|u|+|v|} = 2^{|u|+2\cdot|v|}$$

■

Die Länge eines Pfades ist beschränkt durch $|uv|$. Auf jedem Pfad liegen darüberhinaus also höchstens $|uv|$ Zustände, was eine obere Schranke für die vom Startzustand erreichbaren Zustände von $|uv| \cdot 2^{|u|+2\cdot|v|}$ ergibt.

[4] Alle bis auf Variablenumbenennung, die nur Variablen aus dem Filter verändern.

An einem Beispiel soll dargestellt werden, daß die Abschätzung für die Anzahl der Pfade schon recht gut ist[5], aber die obere Schranke für die Anzahl Zustände wohl noch viel zu hoch ausfällt.

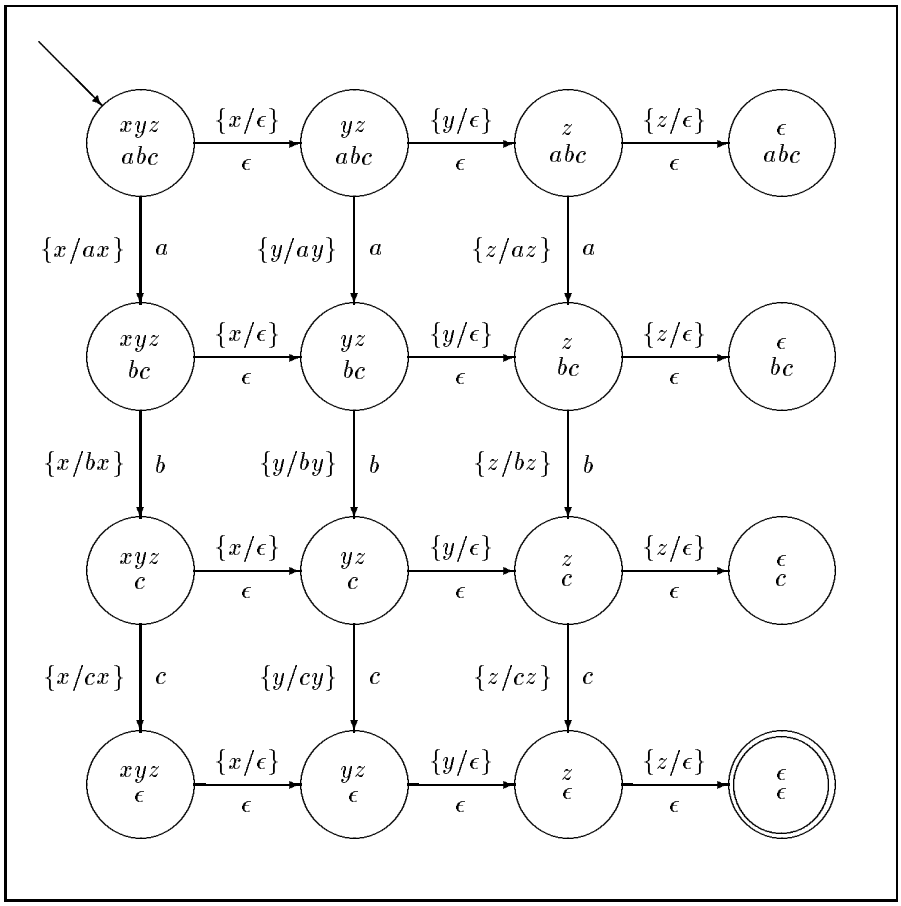


Abbildung 6.1: Dies ist ein Automat für den Test, ob xyz ein Filter von abc ist. Die Anzahl Pfade vom Startzustand links oben zum Finalzustand rechts unten ist $\binom{5}{2} = 10$ während es $4 \cdot 4 = 16$ Zustände gibt. Setzt man dieses Beispiel fort mit den Worten $x_1 x_2 \cdots x_n$ und $aa \cdots a$ (n-mal), so wächst die Anzahl Pfade auf $\binom{2n-1}{n}$ und die Anzahl Zustände auf $(n+1)^2$.

Dennoch zeigt dieses Beispiel, daß die Komplexität des Algorithmus im „*worst-case*“ mindestens in $\mathbf{O}(|u|^2)$ liegt, da ja alle Zustände des Automaten generiert werden müssen und deren Anzahl mindestens quadratisch mit der Länge der Eingabe wächst.

[5]Das Beispiel aus Abbildung 6.1 liefert ja einen Automaten mit $\binom{2n-1}{n}$ Pfaden vom Startzustand zum Finalzustand, wenn die Länge der Eingabeworte jeweils n ist. Diese Funktion wächst aber schneller als jedes Polynom.

Kapitel 7

Generalisierung

7.1 Definition

Als letztes Konzept soll die Generalisierung zweier Wörter mit Variablen betrachtet werden. Eine Definition und ein Algorithmus für die leere Theorie findet man zum Beispiel in [Plot70].

Definition 7.1 (Generalisierung)

Zu zwei Wörtern mit Variablen $u, v \in \Lambda$ heiße $g \in \Lambda$ eine *Generalisierung* von u und v genau dann, wenn es zwei Substitutionen $\sigma, \tau : \mathcal{V} \to \Lambda$ gibt, so daß $\sigma(g) \equiv u$ und $\tau(g) \equiv v$. Dabei heißen σ und τ *angleichende* Substitutionen.

Mit anderen Worten ist g eine Generalisierung von u und v genau dann, wenn g ein Filter von v und u ist. Anders als bei der Unifikation oder dem Matching existiert eine Generalisierung von Wörtern aber immer[1], und deshalb hat ein entsprechender Algorithmus nur die Aufgabe, alle relevanten Lösungen aufzuzählen.

Mit relevant ist natürlich einmal modulo der in Kapitel 3 definierten Normalisierung gemeint. So suchen wir also nur solche g, für die es keine kleinere Generalisierung g' gibt, mit $g = g'$. Ein weiterer Gesichtspunkt wird sein, daß nur die spezifischsten[2] Generalisierungen interessieren:

Definition 7.2 (Spezifischste Generalisierungen)

Sei $g \in \Lambda$ eine Generalisierung von $u, v \in \Lambda$, so heißt g eine *spezifischste Generalisierung* von u und v genau dann, wenn für alle weiteren Generalisierungen h von u und v, die eine Instanz von g sind, schon $g = h$ folgt.

[1] Jede Variable ist ein Filter für alle Wörter.
[2] oder vielleicht speziellsten?

7.2 Existenz und Uneindeutigkeit

Satz 7.3 (Uneindeutigkeit der Generalisierung)

Falls $|\mathcal{S}| > 1$ ist, dann gibt es im allgemeinen nicht nur eine spezifischste Generalisierung.

Beweis:(Gegenbeispiel) O.B.d.A. sei $\mathcal{S} = \{a, b\}$. Dann definiere $u :\equiv ab$, $v :\equiv ba$, $g_1 :\equiv xay$ und $g_2 :\equiv xby$, wobei augenscheinlich g_1 wie auch g_2 Generalisierungen von u und v sind. Angenommen es gäbe jetzt bis auf „=“ nur eine spezifischste Generalisierung von u und v, dann gibt es $\sigma, \tau, \delta_1, \delta_2 : \mathcal{V} \to \Lambda$, mit $\sigma(g_1) \equiv g$, $\tau(g_2) \equiv g$, $\delta_1(g) \equiv u$ und $\delta_2(g) \equiv v$. Mit 2.12.3 folgt $\#(a, g) \geq 1$ und $\#(b, g) \geq 1$. Somit gibt es $l, m, r \in \Lambda$, so daß entweder $g \equiv lambr$, was zu einem Widerspruch zu $\delta_2(g) \equiv ba$ führt, oder $g \equiv lbmar$, was nicht mit $\delta_1(g) \equiv ab$ zusammenpaßt. ■

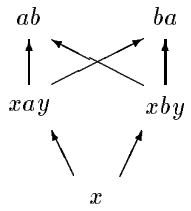


Abbildung 7.1: $(\Lambda, \leq)$ ist kein Durchschnittshalbverband.

Dieser Satz ist äquivalent dazu, daß die partielle Ordnung $(\Lambda/=, \leq)$ kein Durchschnittshalbverband ist, also das Infimum zweier Wörter mit Variablen bezüglich $\leq$ nicht immer existiert (siehe Abbildung 7.1, ein Pfeil steht für die Relation $\leq$). Wie das Gegenbeispiel zeigt, existiert zwar immer eine untere Schranke, diese muß aber nicht immer eindeutig sein.

Über die Anzahl der spezifischsten Generalisierungen zweier Wörter ist aber noch nichts gesagt. Sie könnte auch, wie im Falle der Unifikation, unendlich sein. Eine positive Antwort liefert folgender Satz:

Satz 7.4 (Anzahl spezifischster Generalisierungen ist endlich)

Für $u, v \in \Lambda$ gibt es ein $k \in \mathbb{N}$, $M := \{g_1, \ldots, g_k\} \subseteq \Lambda$, wobei alle g_i spezifischste Generalisierungen von u und v sind, $|g_i| \leq |uv|$ gilt und für jede spezifischste Generalisierung h von u und v es ein $g \in M$ gibt, mit $g = h$.

Beweis: Sei $G := \{g \in \Lambda \mid g \text{ ist spezifischste Generalisierung von } u \text{ und } v\}$, $\mathcal{W} \subseteq \mathcal{V}$ eine beliebige $|uv|$-elementige Teilmenge der Variablen und $\mathcal{S}_{uv} \subseteq \mathcal{S}$ die Menge der in uv vorkommenden Konstantenzeichen. Das nachfolgende Lemma besagt nun, daß aus jeder Klasse $[g]_= \in G/=$ es einen Vertreter $h \in [g]_=$ gibt, mit $|h| \leq |uv|$. Überdies kann h wegen 2.12.3 aus $(\mathcal{S}_{uv} \cup \mathcal{W})^*$ gewählt werden. Zusammengenommen ergibt das

$$h \in \bigcup_{i=0}^{|uv|} (\mathcal{S}_{uv} \cup \mathcal{W})^i =: H$$

In H liegt also jeweils ein Vertreter der Klasse einer spezifischsten Generalisierung von u und v ($G/_= \subseteq H/_= := \{[h]_= \mid h \in H\}$).

Die Abschätzung $|\mathcal{S}_{uv}| \leq |uv|$ ergibt $|\mathcal{S}_{uv} \cup \mathcal{W}| \leq 2 \cdot |uv|$ und damit

$$|H| = \sum_{i=0}^{|uv|} \left|(\mathcal{S}_{uv} \cup \mathcal{W})^i)\right| = \sum_{i=0}^{|uv|} |\mathcal{S}_{uv} \cup \mathcal{W}|^i \leq \sum_{i=0}^{|uv|} (2 \cdot |uv|)^i = \frac{(2 \cdot |uv|)^{|uv|+1} - 1}{2 \cdot |uv| - 1},$$

womit sich H als endlich erweist. Wähle nun $M := H \cap G$, so ist auch M endlich und es gilt

$$(M/_=) \;=\; ((H \cap G)/_=) \;=\; (H/_=) \cap (G/_=) \;=\; (G/_=)$$

■

In diesem Satz kann auch $k = 0$ gelten, also könnte es noch den Fall geben, daß es zwar untere Schranken (Generalisierungen) zweier Wörter gibt, aber die Menge der größten (spezifischsten) unteren Schranken (Generalisierungen) leer ist.

Satz 7.5 (Existenz einer spezifischsten Generalisierung)

Für alle $u, v \in \Lambda$ gibt es eine spezifischste Generalisierung g.

Beweis: Sei $G := \{g \in \Lambda \mid g \text{ ist Generalisierung von } u \text{ und } v\}$ und $\mathcal{W} \subseteq \mathcal{V}$ eine beliebige $|uv|$-elementige Teilmenge der Variablen. Damit definiere nun $H := \bigcup_{i=0}^{|uv|} (\mathcal{S}_{uv} \cup \mathcal{W})^i$ wie im letzten Beweis. Angenommen G besäße kein Maximum bezüglich „$\leq$“, so wäre dies gleichbedeutend damit, daß für alle $g \in G$ es ein $g' \in G$ gibt, so daß $g < g'$.[3] Da H endlich ist (siehe letzten Satz), ist H^{max} nicht leer. Für $u, v \not\equiv \epsilon$ liegt o.B.d.A $x \in \mathcal{V}$ in H. Somit ist auch $H^{max} \cap G$ nicht leer.

Sei nun $h \in H^{max} \cap G$. Nach der Annahme gibt es dann ein $g \in G$ mit $h < g$. Lemma 7.7 liefert wiederum ein g' mit $g' \in G$, $g \leq g'$ und $|g'| \leq |uv|$. Darüber hinaus kann nach einer Variablenumbenennung g' aus H gewählt werden. Zusammengenommen ergibt das $h < g \leq g'$, was einen Widerspruch zur Maximalität von h darstellt. ■

Nimmt man die beiden letzten Sätze zusammen, so ergibt dies die Hauptaussage dieses Kapitels:

Satz 7.6

Für alle $u, v \in \Lambda$ gibt es mindestens eine und höchstens endlich viele **spezifischste** Generalisierungen modulo $=$.

Das nachfolgende Lemma ist dazu äquivalent, daß es zu jeder Generalisierung g zweier Wörter u und v, die länger ist als beide Wörter zusammen, eine Generalisierung g' von u und v gibt, die eine Instanz von g ist (oder anders ausgedrückt, die spezifischer ist als g) und darüber hinaus höchstens gleich lang ist wie u und v zusammen.

[3] „$<$“ soll dabei der nicht reflexive Teil von „$\leq$“ sein. Es gilt also für zwei Wörter $u, v \in \Lambda$: $u < v \;\Leftrightarrow\; u \leq v$ und $u \neq v$.

Lemma 7.7

Zu allen $u, v, g \in \Lambda,\ \sigma, \tau : \mathcal{V} \to \Lambda$ mit

$$\sigma(g) \equiv u,\ \tau(g) \equiv v \text{ und } |u| + |v| < |g|$$

gibt es ein $g' \in \Lambda$ und eine Substitution $\delta : \mathcal{V} \to \Lambda$, so daß:

$$\delta(g) \equiv g' \qquad \begin{array}{lcl} \sigma(g') & \equiv & u \\ \tau(g') & \equiv & v \end{array} \qquad |g'| < |g|$$

Beweis: Sei $g \equiv g_1 \ldots g_{|g|}$ mit $g_i \in \mathcal{V} \cup \mathcal{S}$ für $1 \leq i \leq |g|$. Definiere $G := \{1, \ldots, |g|\},\ I := \{i \in G \mid \sigma(g_i) \equiv \epsilon\}$ und $J := \{j \in G \mid \tau(g_j) \equiv \epsilon\}$. Das relative Komplement $\overline{M}$ einer Teilmenge M von G sei definiert als $\overline{M} := \{k \in G \mid k \notin M\}$. Damit gilt $\left|\overline{I}\right| = |u|$ und $\left|\overline{J}\right| = |v|$, womit man nachrechnet:

$$\begin{array}{rcccl} |I \cap J| & = & |G| - \left|\overline{I \cap J}\right| & = & |G| - \left|\overline{I} \cup \overline{J}\right| \\ & \geq & |G| - (\left|\overline{I}\right| + \left|\overline{J}\right|) & = & |g| - (|u| + |v|) \\ & > & 0 & & \end{array}$$

Also ist $I \cap J \neq \emptyset$, und es gibt ein $i \in G$, mit $\sigma(g_i) \equiv \tau(g_i) \equiv \epsilon$. Weiter definiere:

$$\begin{array}{rcl} \delta(x) & :\equiv & \left\{ \begin{array}{lcl} x & , & x \not\equiv g_i \\ \epsilon & , & x \equiv g_i \end{array} \right. ,\ x \in \mathcal{V} \\ g' & :\equiv & \delta(g), \end{array}$$

und damit sind die Behauptungen des Lemmas erfüllt. ■

Man wundert sich vielleicht, daß σ und τ auch die angleichenden Substitutionen für g' sind. Dies liegt daran, daß sich g und g' genau darin unterscheiden, daß in g' alle Vorkommen von g_i gestrichen worden sind. Andererseits bilden τ und σ gerade g_i auf ϵ ab, und so verhalten sich die beiden bei Anwendung auf g genau gleich wie bei Anwendung auf g'.

7.3 Ein „*brute-force*“-Algorithmus

Es folgt die Spezifikation eines Generalisierungsalgorithmus:

Eingabe: $u, v \in \Lambda$.

Ausgabe: $M := \{g_1, \ldots, g_k\} \subseteq \Lambda$, eine Menge von spezifischsten Generalisierungen von u und v, mit $g_i \neq g_j$ für $i \neq j$. Dabei soll es für alle weiteren spezifischsten Generalisierungen h von u und v ein $g \in M$ geben, mit $g = h$ und $|g| \leq |h|$.

Im Beweis von Satz 7.4 in Verbindung mit Satz 7.6 konnte man folgende Beobachtung machen: Als mögliche Kandidaten, die eine Generalisierung zweier gegebener Wörter u und v sein könnten, reicht es aus, Repräsentanten der „=“-Äquivalenzklassen kleinergleich als $|uv|$ zu betrachten. Darüber hinaus genügt es, diese aus einem endlichen Alphabet zu wählen. Mit den Bezeichnungen aus dem Beweis von Satz 7.4 beschränkt sich also die Suche auf die Wörter aus $\mathcal{S}_{uv} \cup \mathcal{W}$.

Damit läßt sich nun ein erster Algorithmus zur Bestimmung der spezifischsten Generalisierung beschreiben: In einem „*generate and test*"-Verfahren durchlaufe man alle Wörter w aus $\mathcal{S}_{uv} \cup \mathcal{W}$ und teste, ob w eine Generalisierung von u und v ist. Dieser Test läßt sich an Hand des Matchingalgorithmus aus Kapitel 6 durchführen. Es gilt nämlich, daß w eine Generalisierung von u und v ist genau dann, wenn w sowohl ein Filter für u als auch für v ist. Unter diesen Generalisierungen muß man jetzt nur noch die Maxima bezüglich „$\leq$" herausfiltern, die man wieder mit dem Matchingalgorithmus durch paarweisen Vergleich findet. Dabei muß man darauf achten, daß „$\leq$" nicht antisymmetrisch über Λ modulo Variablenumbenennung ist. [4] Deshalb sollte unter den semantisch gleichen Wörtern, die ja immer noch unter den Maxima vorhanden sein könnten, eines bevorzugt werden, wobei man am besten immer das kleinste auswählt.

```
( Set(Λ) ) Generalisierungen(u, v ∈ Λ)
{
    ( Set(V) ) W with |W| == |uv|;
    ( Set(Λ) ) M := ∅, H := ⋃_{i=0}^{|uv|} (S_uv ∪ W)^i;

    forall w in H do {
        if ( match(u, w) && match(v, w) )
            M = M ∪ {w};
    return M;
} :
```

Abbildung 7.2: Auswahl aller relevanten Generalisierungen.

Eine Typdeklaration von *Objekt* zum Typ *Typ* wird als `( Typ ) Objekt` geschrieben und mit `( Set(Λ) ) M;` wird die Variable M als Teilmenge von Λ deklariert.

Die äußere Schleife im letzten Algorithmus von Abbildung 7.3 durchläuft bei einem Aufruf alle aus der Prozedur „Generalisierung" in Abbildung 7.2 übernommenen potentiellen spezifischsten Generalisierungen w aus H^{spez}. Jetzt versucht man paarweise alle anderen w' aus H^{spez} mit w zu matchen und umgekehrt.

Gelingt der Match in beide Richtungen, d.h. sind beide Wörter semantisch gleich, wird das längere oder ein beliebiges der beiden, wenn sie gleich lang sind, aus M, der Liste der Kandidaten für spezifischste Generalisierungen, gestrichen. Gelingt der Match für ein bestimmtes w nicht, so bleibt w erstmal als Kandidat erhalten. Hat sich bei diesem Test w' als echter Filter für w herausgestellt, so kann auch w' keine spezifischste Generalisierung mehr sein. Diese so gefundenen w' werden in der Variablen N aufgesammelt, um sie dann nach Durchlaufen der inneren Schleife aus der Menge der möglichen Kandidaten zu entfernen.

Es bleibt noch der Fall, daß die beiden Wörter unvergleichbar bezüglich „$\leq$" sind. Dann kann man keinen Rückschluß ziehen, und das nächste Paar wird betrachtet.

Die innere Schleife wird quadratisch oft in der Anzahl der aus der Prozedur „Generalisierung" übernommenen Wörter durchlaufen. Da aber schon in dieser Prozedur die Schleife $|H|$-mal ausgeführt wird und $|H|$ exponentionell mit $|uv|$ wächst (vergleiche mit 7.4) ist bei diesem Algorithmus nur mit einer sehr schlechten Komplexität zu rechnen.[5]

[4] Es gilt zum Beispiel $x \leq xy$ und $xy \leq x$, aber x geht nicht durch Variablenumbenennung aus xy hervor.

[5] Das „worst-case"-Beispiel liefern zwei Wörter u und v, die nur aus verschiedenen Variablen bestehen. Alle zu u und v semantisch gleichen Wörter sind diejenigen, die nur aus Variablen bestehen und in denen eine Variable nur

```
( Set(Λ) ) spezifischste_Generalisierungen(u, v ∈ Λ)
{
    ( Set(Λ) ) M = H^spez = Generalisierung(u, v);

    while H^spez ≠ ∅ do {
        choose w ∈ H;
        H^spez = H^spez\{w};

        ( Set(Λ) ) N := ∅;
        forall w' in H^spez do {
            if ( match(w', w) && not ( match(w, w') && |w| < |w'| ) )
                M := M\{w};
            elseif ( match(w, w') ) {
                N := N ∪ {w'};
                M := M\{w'};
            };
        };
        H^spez := H^spez\N;
    };
    return M;
}
```

Abbildung 7.3: Auswahl aller relevanten **spezifischsten** Generalisierungen.

7.4 Anzahl epsilonfreier Generalisierungen ist endlich

Betrachtet man den Beweis für die endliche Anzahl spezifischster Generalisierungen von Satz 7.4 genauer, so stellt man fest, daß er nicht auf allgemeine Generalisierungen übertragbar ist. Der Grund dafür liegt in dem kombinatorischen Lemma, das nur die Länge von spezifischsten Generalisierungen einschränkt, aber nichts über die Länge von anderen Generalisierungen aussagt. Dies ist auch nicht möglich, wie folgendes Beispiel zeigt: Sei $\mathcal{V} = \{x_1, x_2, x_3, \dots\}$ mit $x_i \not\equiv x_j$ für $i \neq j$, dann konstruiere die Folge $(w_i) \in \Lambda^{\mathbb{N}}$ von Wörtern induktiv durch die Gleichung:

$$w_n \quad :\equiv x_n \quad \begin{array}{l} x_1 \\ x_1 x_2 \\ x_1 x_2 x_3 \\ \vdots \\ x_1 x_2 \cdots x_n \end{array}$$

Durch die spezielle Konstruktion sind alle w_i schon in Normalform, es läßt sich nämlich keine Zerlegung der Variablen $\{x_1, \dots, x_n\}$ angeben, die den Bedingungen aus Definition 3.12 genügen würde. Da die w_i alle unterschiedlich lang sind, können sie auch nicht durch Variablenumbenennung auseinander hervorgehen und sind deshalb alle semantisch verschieden. Auf der anderen Seite gilt $\{x_1/\epsilon, \dots, x_n/\epsilon, x_{n+1}/a\}w_{n+1} \equiv aa$ für eine feste Konstante $a \in \mathcal{S}$. Damit ist gezeigt, daß es im

einmal vorkommt. Diese sind auch zugleich alle Generalisierungen der beiden Wörter. Mit der Einschränkung auf H gibt es für die Auswahl der Variablen, die nur einmal vorkommt, genau $n := |uv|$ Möglichkeiten. Zur Länge i können nun auf $(n-1)^{i-1}$ Arten die restlichen $i-1$ Variablen gewählt werden. Eine weiterer Faktor i kommt für die Wahl der Position der einzelnen Variable hinzu. Insgesamt liefert die Prozedur „Generalisierung“ also $n \cdot \sum_{i=1}^{n-1} i \cdot (n-1)^{i-1} = \mathbf{\Omega}\left((n-1)^n\right)$ Generalisierungen von u und v.

allgemeinen modulo „=“ unendlich viele Filter eines Wortes gibt und somit auch unendlich viele Generalisierungen[6] zweier Wörter:

Lemma 7.8

Zu einem Wort mit Variablen gibt es modulo „=“ im allgemeinen eine unendliche Menge paarweise nicht vergleichbarer Filter, die beliebig lang sind.

Was bei diesem Beispiel auffällt ist, daß bei den Substitutionen beidesmal die gleichen Variablen durch ϵ ersetzt werden, um *aa* bzw. *bb* zu erhalten. Betrachtet man eine Generalisierung als Verschmelzung der Information, die in beiden Wörtern steckt, tragen solche Variablen nichts zum Informationsgewinn durch Generalisieren bei. Beschränkt man sich auf Generalisierungen, zu denen es Substitutionen gibt, bei denen das nicht auftritt, so kann man die Endlichkeit wieder retten.

Definition 7.9 (Epsilonfreie Generalisierung)

Zu zwei Wörtern $u, v \in \Lambda$ heiße g eine *epsilonfreie Generalisierung* von u und v genau dann, wenn es Subsstitutionen $\sigma, \tau : \mathcal{V} \to \Lambda$ gibt, so daß $\sigma(g) \equiv u$ und $\tau(g) \equiv v$ und für alle $x \in \mathcal{V}_g$ entweder $\sigma(x) \not\equiv \epsilon$ oder $\tau(x) \not\equiv \epsilon$ ($\{\sigma(x), \tau(x)\} \neq \{\epsilon\}$).

Das im folgenden beschriebene Verfahren zur Aufzählung aller epsilonfreien Generalisierungen besteht aus zwei Phasen. In der ersten Phase werden zunächst alle linearen epsilonfreien Generalisierungen erzeugt, in denen jede Variable nur einmal vorkommt. In der zweiten Phase werden zusätzlich zu den bisher erzeugten Generalisierungen g alle Generalisierungen hinzugenommen, die aus Unifikation zweier Variablen aus $\mathcal{V}_g$ entstehen.

Definition 7.10 (Lineare Wörter)

Ein Wort $w \in \Lambda$ heißt *linear* genau dann, wenn für alle $x \in \mathcal{V}_w$ gilt: $\#(x, w) = 1$.

Der Terminus *linear* spielt zum Beispiel in [StEk93] eine wichtige Rolle. Dort heißt ein Term t *linear* genau dann, wenn jede Variable in t nur einmal vorkommt.

Lemma 7.11

Zu jedem $w \in \Lambda$ gibt es ein lineares Wort $w' \in \Lambda$ und eine Substitution $\sigma : \mathcal{V} \to \Lambda$, so daß $\sigma(w') \equiv w$, wobei $\sigma : \mathcal{V}_{w'} \to \mathcal{V}_w$ gilt.

Beweis: Sei $w \equiv w_1 \cdots w_n$. Falls $w_i \in \mathcal{V}$, dann setze $w_i' :\equiv x_i$, und $\sigma(x_i) :\equiv w_i$. Im anderen Fall setze $w_i' :\equiv w_i$ und $\sigma(w_i) :\equiv w_i$. Dabei sei $\{x_1, x_2, \dots\} \subseteq \mathcal{V}$, wobei $x_i \not\equiv x_j$ fü $i \neq j$. ■

Dieses Lemma liefert das für den folgenden Algorithmus wichtige Argument, daß es ausreicht, alle linearen epsilonfreien Generalisierungen zu finden. Mögliche weitere nichtlineare epsilonfreie Generalisierungen lassen sich dann durch Unifikation von Variablen der linearen Generalisierungen

[6] Die w_i sind Generalisierungen der Wörter *aa* und *bb*.

finden, wobei nach der Unifikation noch getestet werden muß, ob das so erhaltene Wort tatsächlich eine Generalisierung ist[7]. Dieser Test läßt sich wie beim Generalisierungsalgorithmus des vorhergehenden Kapitels mittels des Filter-Algorithmus durchführen.

Um den Notationsaufwand zu reduzieren, gelte o.B.d.A. $\{x_1, x_2, x_3, \dots\} \subseteq \mathcal{V}$, wobei die x_i paarweise verschieden sind, und für Wörter u und v, für die eine Generalisierung g gefunden werden soll, gelte $(\mathcal{V}_u \cup \mathcal{V}_v) \cap \{x_1, x_2, \dots\} = \emptyset$ und $\mathcal{V}_g \subseteq \{x_1, x_2, \dots\}$. Mit anderen Worten, es werden neue Variablen $\{x_1, x_2, \dots\}$ eingeführt, die nicht in den Wörtern vorkommen, für die eine Generalisierung gefunden werden soll, und zusätzlich verlangt man, daß die Generalisierungen nur neue Variablen enthalten.

Das oben erwähnte Verfahren, das nun vorgestellt werden soll, basiert wie die Algorithmen für die Unifikation und das Matching auf dem Automatenprinzip. Wie bei diesen Automaten bestehen die Zustände auch hier wieder aus (diesmal geordneten) Paaren von Wörtern mit Variablen. Während der Akzeption eines Wortes muß man hier zwei Substitutionen generieren, was zusammen mit einem technischen Kniff zu einer Beschriftung der Zustandsübergänge mit drei Labels führt. Der technische Kniff besteht darin, den gesamten Indeterminismus in den Graphen des Automaten zu verlegen und aus der Definition der akzeptierten Sprache herauszunehmen. So erreicht man, daß ein Pfad vom Startknoten zum Finalzustand genau eine Generalisierung mit den zwei dazugehörenden Substitutionen beschreibt. Dieses Problem des Indeterminismus tritt dann auf, wenn die beiden Wörter mit demselben Präfix beginnen. Nun kann eine Generalisierung einmal genau dasselbe Präfix besitzen, oder an dieser Stelle steht in der Generalisierung eine Variable[8]. Wollte man die neue Variable an diesem Zustandsübergang definieren, so müßte man sich merken, wieviele Variablen schon erzeugt wurden. Diese globale Information kann einmal nicht eindeutig definiert sein und wenn doch, dann läßt sie sich nur aus dem gesamten Zustandsübergangsgraphen ablesen. Für einen festen Pfad durch den Graphen, kann man diese Anzahl eindeutig bestimmen, und so sollte die Namensgebung der neuen Variablen in der Definition der akzeptierten Sprache vorgenommen werden.

Hier besteht nur die Möglichkeit die Zustandsübergänge danach zu unterscheiden, ob eine neue Variable generiert werden soll, oder nicht.

Definition 7.12 (Zustandsübergangsfunktion)

Zu $c, d, r, s \in \Lambda$, $\{c, d\} \neq \{\epsilon\}$ definiere die Relation $\rightarrow \subseteq \Lambda^2 \times \Lambda \times \Lambda^2$ durch:

$$(cr, ds) \xrightarrow[c,d]{\epsilon} (r, s) \qquad \text{(kürzer auch } (cr, ds) \longrightarrow (r, s)\text{)}$$

$$(cr, cs) \xrightarrow[c,c]{c} (r, s) \qquad \text{(kürzer auch } (cr, cs) \xrightarrow{c} (r, s)\text{),}$$

wobei im zweiten Fall $c \not\equiv \epsilon$ vorausgesetzt wird.

Steht also über dem Pfeil ein ϵ so wird eine neue Variable erzeugt. Im anderen Fall steht über dem Pfeil das gemeinsame Präfix der beiden Wörter, das in die Generalisierung übernommen werden soll.

Die Voraussetzung $c \not\equiv \epsilon$ könnte man auch fallen lassen, würde sich damit aber beliebig lange Pfade im Automaten einhandeln, da solche Übergänge mit $c \equiv \epsilon$ einen Zustandsübergang von einem Zustand zu sich selbst erlauben würden, und so insbesondere der Graph nicht mehr zyklenfrei wäre.

[7] Ist man nur an epsilonfreien Generalisierungen interessiert, so muß man sogar auch noch die Epsilonfreiheit testen, wie folgendes Beispiel zeigt. Mit den Substitutionen $\{x_1/a, x_2/\epsilon, x_3/\epsilon, x_4/a\}$ und $\{x_1/\epsilon, x_2/a, x_3/a, x_4/\epsilon\}$ erweist sich das Wort $x_1x_2x_3x_4$ als eine epsilonfreie lineare Generalisierung von aa und wenn man in den Substitutionen a durch b ersetzt genau so von bb. Eine Instanz von $x_1x_2x_3x_4$, die durch Unifikation von x_3 und x_4 mit x_1 entsteht, ist $x_1x_2x_1x_1$. Dies ist zwar immer noch eine Generalisierung von aa und bb, aber keine epsilonfreie mehr, da x_1 in beiden Fällen durch ϵ ersetzt werden muß.

[8] Hier werden nicht nur spezifischste Generalisierungen gesucht.

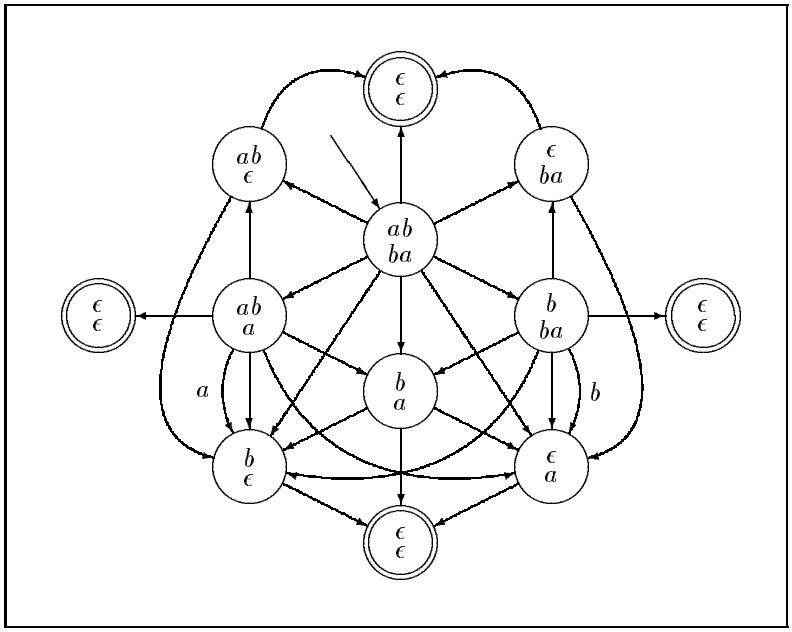


Abbildung 7.4: Dieser Automat akzeptiert alle linearen epsilonfreien Generalisierungen der Wörter ab und ba. Dabei werden die Abkürzungen aus der Definition von „$\rightarrow$“ benützt. D.h. ist ein Pfeil mit keinem Wort beschriftet, so wird eine neue Variable erzeugt, und implizit in den angleichenden Substitutionen diese Variable durch die weggestrichenen Präfixe ersetzt. Ansonsten wird der Pfeil mit einem gemeinsamen Präfix der beiden Wörter beschriftet und dieses wird in die Generalisierung übernommen. Bei diesem Automaten ist letzteres nur an zwei Stellen der Fall, nämlich dort wo die gemeinsamen Buchstaben a und b in die Generalisierung übernommen werden. Man beachte noch, daß die Zustandübergangsfunktion transitiv abgeschlossen ist.

Definition 7.13 (Akzeptierte Sprache)

Sei nun $(u_0, v_0) \xrightarrow[c_1,d_1]{l_1} (u_1, v_1) \xrightarrow[c_2,d_2]{l_2} \cdots \xrightarrow[c_n,d_n]{l_n} (u_n, v_n)$, so sei für $0 < i \leq n$:

$$\sigma_0 :\equiv \tau_0 :\equiv \mathsf{id} \qquad w_0 :\equiv \epsilon$$

$$b_i :\equiv \begin{cases} x_i \text{ falls } c_i \equiv \epsilon \\ c_i \text{ sonst} \end{cases} \qquad w_i :\equiv w_{i-1} b_i$$

$$\sigma_i :\equiv \{x_i/c_i\} \circ \sigma_{i-1} \qquad \tau_i :\equiv \{x_i/d_i\} \circ \tau_{i-1}$$

Damit schreibe nun $(u_0, v_0) \underset{\sigma_n,\tau_n}{\overset{w_n}{\Longrightarrow}} (u_n, v_n)$ und $|\Rightarrow| :\equiv n$.

Die Definition der σ_n, τ_n und von w_n erfolgt strikt von links nach rechts und so folgt unmittelbar folgendes Lemma:

Lemma 7.14 $(u, v) \underset{\sigma,\tau}{\overset{w}{\Longrightarrow}} (r, s)$ **impliziert** $(uu', vv') \underset{\sigma,\tau}{\overset{w}{\Longrightarrow}} (ru', sv')$

Satz 7.15 $(u,v) \overset{w}{\underset{\sigma,\tau}{\Longrightarrow}} (r,s)$ **impliziert** $\sigma(w)r \equiv u,\ \tau(w)s \equiv v$

Beweis: Der Beweis wird durch Induktion über das n aus Definition 7.13 geführt. Im Induktionsanfang bei $n = 0$ ist nichts zu zeigen und es gelte die Behauptung nun für $n-1 \geq 0$. Als erstes hat man:

$$(u_0,v_0) \underset{\sigma_{n-1},\tau_{n-1}}{\overset{w_{n-1}}{\Longrightarrow}} (u_{n-1},v_{n-1}) \underset{c_n,d_n}{\overset{l_n}{\longrightarrow}} (u_n,v_n)$$

Die Definition von σ_n, τ_n und b_n ergibt immer $\sigma_n(b_n) \equiv c_n$ und $\tau_n(b_n) \equiv d_n$, womit man für σ_n nachrechnet:

$$\begin{array}{lllll} \sigma_n(w_n)u_n & \equiv & \sigma_n(w_{n-1}b_n)u_n & \equiv & \sigma_n(w_{n-1})\sigma_n(b_n)u_n \\ & \equiv & \sigma_{n-1}(w_{n-1})c_n u_n & \equiv & \sigma_{n-1}(w_{n-1})u_{n-1} \\ & \overset{\text{Ind.-Beh.}}{\equiv} & u_0 & & \end{array}$$

Analog geht der Beweis für τ_n. ■

Durch Einsetzen von (ϵ,ϵ) für (r,s) erhält man dann:

Korollar 7.16 (Korrektheit)

Für alle $u, v \in \Lambda$ folgt aus $(u,v) \overset{w}{\underset{\sigma,\tau}{\Longrightarrow}} (\epsilon,\epsilon)$, daß w eine epsilonfreie Generalisierung von u und v ist, mit angleichenden Substitutionen σ und τ.

Beweis: Bei der Definition der σ_n und τ_n wurde immer vorausgesetzt, daß entweder $c_n \not\equiv \epsilon$ oder $d_n \not\equiv \epsilon$. Somit kann keine Variable gleichzeitig von σ_n und τ_n auf ϵ abgebildet werden. ■

Satz 7.17 (Vollständigkeit)

Sei g eine lineare, epsilonfreie Generalisierung von u und v, wobei $u,v,g \in (\mathcal{V}\backslash\{x_1,x_2,x_3,\dots\} \cup \mathcal{S})^*$. Die angleichenden Substitutionen seien σ bzw. τ, so daß für alle $x \in \mathcal{V}_g$ gilt: $\{\sigma(x),\tau(x)\} \neq \{\epsilon\}$. Dann gibt es eine Variablenumbenennung $\pi : \mathcal{V} \to \mathcal{V}$ auf $\mathcal{V}_g$, Substitutionen $\sigma', \tau' : \mathcal{V} \to \Lambda$, so daß gilt:

$$(u,v) \underset{\sigma',\tau'}{\overset{\pi(g)}{\Longrightarrow}} (\epsilon,\epsilon), \quad \sigma' \circ \pi\,|_{\mathcal{V}_g} \equiv \sigma, \quad \tau' \circ \pi\,|_{\mathcal{V}_g} \equiv \tau. \tag{7.1}$$

Beweis: Der Beweis wird geführt durch Induktion über die Länge von g. Dabei wird zusätzlich gezeigt, daß

$$\pi\,|_{\mathcal{V}\backslash\mathcal{V}_g} \equiv \mathsf{id} \tag{7.2}$$

Im Induktionsanfang ist $g \equiv u \equiv v \equiv \epsilon$ und nach Definition 7.13 gilt $(\epsilon,\epsilon) \overset{\mathsf{id}(\epsilon)}{\underset{\mathsf{id},\mathsf{id}}{\Longrightarrow}} (\epsilon,\epsilon)$.

Sei also nun $|g| > 0$. Die Voraussetzung $\{\sigma(x),\tau(x)\} \neq \{\epsilon\}$ für alle $x \in \mathcal{V}_g$ ergibt $g \equiv hb$, mit $b \in \mathcal{S} \cup \mathcal{V}$, $h \in \Lambda$ und $\sigma(b)$ oder $\tau(b)$ ungleich ϵ. Dann gibt es auch $r,s,c,d \in \Lambda$ mit

$$rc \equiv u,\ sd \equiv v,\ \sigma(h) \equiv r,\ \tau(h) \equiv s,\ \sigma(b) \equiv c,\ \tau(b) \equiv d.$$

Damit erweist sich h als lineare epsilonfreie Generalisierung von r und s und die Induktionsbehauptung läßt sich anwenden. Dies ergibt die Existenz von Substitutionen $\mu, \nu : \mathcal{V} \to \Lambda$ und einer Variablenumbenennung $\rho : \mathcal{V} \to \mathcal{V}$ auf $\mathcal{V}_h$, so daß

$$(r, s) \xRightarrow[\mu,\nu]{\rho(h)} (\epsilon, \epsilon), \quad n :\equiv |\Rightarrow| + 1 \quad \text{und} \quad \rho\,|_{\mathcal{V}\setminus\mathcal{V}_h} \equiv \mathsf{id} \tag{7.3}$$

Vor allem sind aber $\mu \circ \rho$ und $\nu \circ \rho$ auf $\mathcal{V}_h$ identisch mit σ und τ:

$$\mu \circ \rho\,|_{\mathcal{V}_h} \equiv \sigma, \quad \nu \circ \tau\,|_{\mathcal{V}_h} \equiv \tau \tag{7.4}$$

Das Lemma 7.14 ergibt dann

$$(u, v) \equiv (rc, sd) \xRightarrow[\mu,\nu]{\rho(h)} (c, d).$$

Mit $\sigma' :\equiv \{x_n/c\} \circ \mu$, $\tau' :\equiv \{x_n/d\} \circ \nu$ und b_n aus Definition 7.13 erhält man:

$$(u, v) \xRightarrow[\sigma',\tau']{\rho(h)b_n} (\epsilon, \epsilon) \tag{7.5}$$

Für den Rest werden zwei Fälle unterschieden:

Im ersten Fall übernimmt die Generalisierung den letzten Buchstaben von u und v, es gilt also: $\sigma(b) \equiv \tau(b) \equiv b \equiv c \equiv d$. Setze nun $\pi :\equiv \rho$ ($\pi(b) \equiv \rho(b) \equiv b$). Da g linear ist, folgt $b \notin \mathcal{V}_h$, und somit auch $b \notin \mathcal{V}_{\pi(h)} \supseteq \mathcal{V}_{\rho(h)}$. Auf $\mathcal{V}_h$ erbt π die Injektivität von ρ, und mit dem letzten Argument erweist π als injektiv auf ganz $\mathcal{V}_g$. Nun ist $\pi(g) \equiv \rho(g)$, und x_n tritt in den betrachteten Wörtern nirgends auf, woraus sich die Gültigkeit der beiden rechten Gleichungen in (7.1) ergibt.

Im zweiten Fall ist der letzte Buchstabe von u und v verschieden und es gilt $\sigma(b) \equiv c \not\equiv d \equiv \tau(b)$. Offensichtlich muß dann $b \in \mathcal{V}$ liegen und es gilt $b_n \equiv x_n$. Definiere nun

$$\pi\,|_{\mathcal{V}\setminus\{b\}} :\equiv \rho, \quad \pi(b) :\equiv x_n \tag{7.6}$$

Aus der Definition 7.13 folgt $x_n \notin \mathcal{V}_{\pi(h)}$. Damit gilt für alle $y \in \mathcal{V}_h$: $\pi(y) \not\equiv x_n \equiv \pi(b)$. Nach Definition ist π schon eine Variablenumbenennung auf $\mathcal{V}_h$ und mit dem letzten Resultat dann sogar auf $\mathcal{V}_g$.

Jetzt fehlt noch die Gültigkeit der rechten beiden Gleichungen aus (7.1), wobei aus Symmetriegründen nur die erste nachgerechnet wird:

$$\sigma' \circ \pi\,|_{\mathcal{V}_h} \equiv \{x_n/c\} \circ \mu \circ \pi\,|_{\mathcal{V}_h} \overset{(7.6)}{\equiv} \{x_n/c\} \circ \mu \circ \rho\,|_{\mathcal{V}_h} \overset{(7.4)}{\equiv} \{x_n/c\} \circ \sigma\,|_{\mathcal{V}_h} \equiv \sigma_{\mathcal{V}_h} \tag{7.7}$$

Die letzte Gleichheit in (7.7) gilt, da $x_n \notin \mathcal{V}_{\sigma(h)} \subseteq \mathcal{V}_{\sigma(g)} = \mathcal{V}_u$. Wegen $\mathcal{V}_g = \mathcal{V}_h \cup \{b\}$ muß nur noch die Gleichheit auf b gezeigt werden:

$$(\sigma' \circ \pi)(b) \equiv (\{x_n/c\} \circ \mu \circ \pi)(b) \equiv \{x_n/c\}(\mu(x_n)) \equiv \{x_n/c\}(x_n) \equiv c \equiv \sigma(b)$$

In Beiden Fällen gilt $\pi\,|_{\mathcal{V}\setminus\mathcal{V}_g} \equiv \rho\,|_{\mathcal{V}\setminus\mathcal{V}_g} \equiv \mathsf{id}$, womit auch noch der Zusatz (7.2) erfüllt ist. Weiter ist beidesmal $\pi(g) \equiv \rho(h)\pi(b) \equiv \rho(h)b_n$ womit sich (7.5) in den linken Teil von (7.1) verwandelt. ■

Die Menge der Zustände eines solchen Automaten ist endlich. Auch gibt es keine Zyklen, da ein Wort des Paares eines Zustandes bei einem Zustandübergang um ein Präfix gekürzt wird. Damit ist auch die Menge der Pfade durch den Zustandsübergangsgraphen des Automaten endlich und es gibt eine Surjektion von ihr auf die Menge aller epsilonfreien, linearen Generalisierungen modulo „=“, wie der letzte Satz zeigt. In Kombination mit Lemma 7.11 und der Beobachtung, daß es nur endlich viele Variablenunifikationen[9] über einer endlichen Variablenmenge gibt, beweist dies den Kernsatz dieses Abschnittes:

[9] Eine Substitution $\sigma : \mathcal{V} \to \Lambda$ heiße eine Variablensubstitution genau dann, wenn $\sigma : \mathcal{V} \to \mathcal{V}$.

Satz 7.18

Die Menge der epsilonfreien Generalisieungen modulo „=“ von zwei Wörtern ist endlich.

Das in diesem Abschnitt beschriebene Verfahren zur Aufzählung aller epsilonfreien Generalisierung sollte im Wesentlichen nur die Endlichkeit der Lösungsmenge zeigen. Darüberhinaus kann es aber die Prozedur „Generalisierungen“ im Algorithmus zur Bestimmung der spezifischsten Generalisierungen ersetzen, denn:

Lemma 7.19

Zu jeder spezifischsten Generalisierung $g \in \Lambda$ zweier Wörter $u, v \in \Lambda$ gibt es eine epsilonfreie spezifischste Generalisierung $h \in \Lambda$ von u und v mit $g = h$.

Beweis: Wähle h als eine beliebige Normalform von g. Dann gilt $h = g$ und h ist auch eine spezifischste Generalisierung von u und v. Wäre nun h nicht epsilonfrei, so gäbe es $\sigma, \tau : \mathcal{V} \to \Lambda$ und eine Variable $x \in \mathcal{V}_h$, so daß gilt:

$$\sigma(h) \equiv u, \ \tau(h) \equiv v \text{ und } \sigma(x) \equiv \tau(x) \equiv \epsilon$$

Definiere nun $h' :\equiv \{x/\epsilon\}(h)$, dann gilt

$$\begin{array}{lllll} \sigma(h') & \equiv & \sigma(\{x/\epsilon\}(h)) & \equiv & \sigma(h) \equiv u \\ \tau(h') & \equiv & \tau(\{x/\epsilon\}(h)) & \equiv & \tau(h) \equiv v \end{array}$$

Damit ist auch h' eine Generalisierung von u und v, und es gilt $h \neq h'$, da h eine Normalform ist. Dies ergibt eine Widerspruch dazu, daß h eine spezifischste Generalisierung ist. ■

Ausblick

Die Beispiele in der Arbeit haben gezeigt, daß die Struktur von Wörtern mit Variablen, wenn man sie unter dem Gesichtspunkt der semantischen Gleicheit betrachtet, unerwartet komplex ist. Daraufhin wurden zwei Normalisierungen eingeführt, mit denen sich die Gleichheit entscheiden ließ. Bei der zweiten Normalisierung „$\triangleright$“ fehlt einmal ein effizienter Algorithmus, der eine beste Zerlegung bestimmt, und zweitens ist noch nicht klar, ob eine Reduktion mit „$\triangleright_0$“ immer sofort zu einer Normalform führt.

Im Kapitel 4, über die Größte Gemeinsame Teilfolge zweier Wörter ohne Variablen, wurde ein neuer Zugang zu diesem Thema vorgestellt. Die Komplexität der dort behandelten Algorithmen muß auch noch näher bestimmt werden. Darüberhinaus ergeben sich vielleicht auch noch praktische Anwendungen für die kleinste gemeinsame Oberfolge zweier Wörter.

Was die Unifikation von Wörten mit Variablen betrifft, sind noch am meisten Fragen offen. Die brennendste Frage ist wohl, ob die Lösungsmenge eines Unifikationsproblem immer regulär ist, und, wenn ja, wie sich der entsprechende endliche Automat erzeugen läßt. Im anderen Fall, wenn es auch Unifikationsprobleme mit nichtregulärer Lösungsmenge gibt, steht noch die Klassifizierung von Unifikationsproblemen nach der Regularität ihrer Lösungsmenge aus.

Die Komplexitätsanalyse des Matching-Algorithmus brachte auch nur eine schlechte obere Schranke. Dies gilt ebenfalls für die beiden vorgestellten Verfahren zur Erzeugung von spezifischsten Generalisierungen.

Anhang A

Verschiedene Hilfssätze

A.1 Reguläre Ausdrücke

Grundsätzlich sei hier verwiesen auf [ASU88] oder [Perr90] Hier werden reguläre Ausdrücke mit ihrer Expansion, das heißt mit der Menge der Wörter, die sie beschreiben, gleichgesetzt. Auf diesen Mengen von Wörtern werden folgende Operatoren definiert:

Definition A.1 (Syntax für Reguläre Operatoren)

Seien A, B Reguläre Ausdrücke, so sind es auch:

1. $A \cdot B := AB := \{ab \mid a \in A, b \in B\}$
2. $A \mid B := A \cup B$
3. $A^* := \{a_1 \cdots a_k \mid k \geq 0, a_i \in A\}$
4. $A^+ := A(A^*)$

A.2 Transitiver Abschluß

Definition A.2

Auf der Menge der binären Relationen $A \times A$ über einer festen Menge A, definiere den transitiven Abschlußoperator Tr auf einem $R \subseteq A \times A$ wie folgt:

$$\mathsf{Tr} : (A \times A) \to (A \times A), \ mit$$
$$(a, b) \in \mathsf{Tr}(R)$$
$$\textbf{gdw.}$$
$$\exists k > 1, \ a_1, \ldots, a_k \in A, \ mit \ (a_i, a_{i+1}) \in A \times A,$$
$$für \ i = 1, \ldots, k-1, \ und \ a_1 = a, \ a_k = b.$$

Diesen Operator Abschlußoperator zu nennen, wird durch folgendes Lemma gerechtfertigt.

Lemma A.3 Tr ist monoton und idempotent.

Beweis:

(A) Tr monoton: Seien $R, S \subseteq A \times A$, und zusätzlich gelte $R \subseteq S$. Weiter seien $r_0, \ldots, r_k \in A$ mit $(r_i, r_{i+1}) \in R$ für $i = 1, \ldots, k-1$, also $(r_0, r_k) \in R$. Die Voraussetzung liefert auch $(r_i, r_{i+1}) \in S$ und somit auch $(r_0, r_k) \in S$, womit $\mathsf{Tr}(R) \subseteq \mathsf{Tr}(S)$ folgt.

(B) Tr idempotent: Für $R \subseteq A \times A$ folgt nach Definition A.2 $R \subseteq \mathsf{Tr}(R)$ und damit natürlich erst recht $\mathsf{Tr}(R) \subseteq \mathsf{Tr}(\mathsf{Tr}(R))$. Für die umgekehrte Inklusion sei $(a, b) \in \mathsf{Tr}(\mathsf{Tr}(R))$ beliebig. Damit gibt es ein $l > 1$, $a = a_1, a_2, \ldots, a_k = b \in A$, mit $(a_i, a_{i+1}) \in \mathsf{Tr}(R)$, für $1 \leq i < l$. Eine weitere Anwendung von A.2 ergibt die Existenz von $k_1, \ldots, k_{l-1} \in \mathbb{N}$, alle > 1,

$$\left.\begin{array}{llll} a_{1,1}, & a_{1,2}, & \ldots, & a_{1,k_1} \\ a_{2,1}, & a_{2,2}, & \ldots, & a_{2,k_2} \\ \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \\ a_{l-1,1}, & a_{l-1,2}, & \ldots, & a_{l-1,k_l} \end{array}\right\} \in A \quad , \qquad \begin{array}{llll} a_{1,1} = a_1 \, , & & a_{1,k_1} & = a_2 \\ a_{2,1} = a_2 \, , & & a_{1,k_2} & = a_3 \\ \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \\ a_{l-1,1} = a_{l-1}, & & a_{l-1,k_{l-1}} & = a_l \end{array} \quad ,$$

so daß für alle $1 \leq i < l$, $1 \leq j < k_i$: $(a_{ij}, a_{ij+1}) \in R$. Woraus sich insgesamt $(a, b) = (a_1, a_l) \in \mathsf{Tr}(R)$ erschließen läßt.

■

Lemma A.4

Zu $R, S \subseteq A \times A$ gilt:

$$R \subseteq \mathsf{Tr}(S) \Rightarrow \mathsf{Tr}(R) \subseteq \mathsf{Tr}(S)$$

Beweis: Mit A.3 erhält man:

$$R \subseteq \mathsf{Tr}(S) \overset{\text{Monotonie}}{\Longrightarrow} \mathsf{Tr}(R) \subseteq \mathsf{Tr}(\mathsf{Tr}(S)) \overset{\text{Idempotenz}}{\Longrightarrow} \mathsf{Tr}(R) \subseteq \mathsf{Tr}(S)$$

■

A.3 Injektive Erweiterung einer Abbildung

Satz A.5

Sei $\mathsf{f}: M' \to M$ eine injektive Abbildung von $M' \subseteq M$ nach M und M' sei endlich. Dann gibt es eine injektive Erweiterung von $\mathsf{g}: M \to M$ von M nach M, also $\mathsf{g}(m') = \mathsf{f}(m')$ für alle $m' \in M'$ und g ist injektiv.

Beweis: O.b.d.A. gelte $\mathsf{f}(m) = m$ für $m \notin M'$. Für $m \in \mathsf{f}(M')$ definiere $\mathsf{i}: \mathsf{f}(M') \cup M' \to M'$ durch

$$\mathsf{i}(m) := \begin{cases} \mathsf{f}^{-1}(m) & \text{falls } m \in \mathsf{f}(M') \\ m & \text{sonst} \end{cases} \quad , \text{ für alle } m \in \mathsf{f}(M') \cup M'. \tag{A.1}$$

Dies ist wohldefiniert, da f injektiv auf M' ist und somit $\mathsf{f}^{-1}(m)$ einelementig für alle $m \in \mathsf{f}(M')$. Wegen der Endlichkeit von M' gibt es zu jedem $m \in \mathsf{f}(M')$ ein minimales $p \in \mathbb{N}$ und dazu in

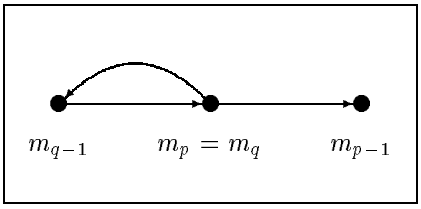


Abbildung A.1: Beispiel zum Fall $p \neq q$ im Beweis zur Erweiterung einer injektiven Abbildung. Die Pfeile entsprechen der Beziehung $m \mapsto \mathsf{f}(m)$. Dabei wird angenommen, daß $p > 0$ gilt, und zu einem Widerspruch geführt.

Abhängigkeit von p ein minimales $q \in \mathbb{N}$, so daß $\mathsf{i}^p(m) = \mathsf{i}^q(m)$. Falls $p \neq q$, so ist $p = 0$ und deshalb $m \in M'$. Denn sonst ist mit $m_n := \mathsf{i}^n(m)$ einmal $m_p \in M'$, $\mathsf{i}(m_{q-1}) = m_p$ als auch $\mathsf{i}(m_{p-1}) = m_p$ $(p > 0)$. Da q minimal gewählt wurde, ist $m_{p-1} \neq m_{q-1}$, was im Widerspruch zur Wohldefiniertheit von f steht. Dies bedeutet andererseits für alle $m \in \mathsf{f}(M')\backslash M'$, daß es ein $p \in \mathbb{N}$ gibt mit $\mathsf{i}^p(m) = \mathsf{i}^{p+n}(m)$ für alle $n \in \mathbb{N}$. Im Sinne obiger Schreibweise definiere in diesem Fall $m_\infty := \mathsf{i}^p(m)$.[1] Damit sei nun

$$\mathsf{g}(m) := \begin{cases} \mathsf{f}(m) & \text{falls } m \in M' \\ m_\infty & \text{falls } m \in \mathsf{f}(M')\backslash M' \\ m & \text{falls } m \in M \backslash (\mathsf{f}(M') \cup M') \end{cases}, \text{ für alle } m \in M.$$

g ist offensichtlich eine Erweiterung von f. Es bleibt noch zu zeigen, daß g auch injektiv auf ganz M ist. Dazu seien $x, y \in M$ beliebig gewählt. Für $x, y \in M'$ ist nichts zu zeigen. Wenn nun nur $x \in M'$, so bleiben zwei Fälle. Einmal sei $y \in M \backslash (\mathsf{f}(M') \cup M')$, woraus sofort $\mathsf{g}(y) = y \neq \mathsf{f}(x) = \mathsf{g}(x)$ folgt. Im anderen Fall ist $y \in \mathsf{f}(M')\backslash M'$ und $\mathsf{g}(y) = y_\infty$. Aus der Definition von i und y_∞ folgt, daß es kein $m \in M'$ gibt, mit $\mathsf{f}(m) \equiv y_\infty$. Somit gilt $\mathsf{g}(x) = \mathsf{f}(x) \neq y_\infty = \mathsf{g}(y)$. Da auch i injektiv ist, gilt

$$\mathsf{g}(x) = x_\infty = y_\infty = \mathsf{g}(y) \;:\Leftrightarrow\; x = y, \text{ für alle } x, y \in \mathsf{f}(M')\backslash M'$$

Für $x, y \in M \backslash (\mathsf{f}(M') \cup M')$ folgt trivialerweise aus $\mathsf{g}(x) = \mathsf{g}(y)$ nach Definition $x = y$. Es bleibt also noch der Fall $x \in M \backslash (\mathsf{f}(M') \cup M')$ und $y \in \mathsf{f}(M')\backslash M'$. Hier gilt aber $\mathsf{g}(x) = x \notin M'$ und $\mathsf{g}(y) = y_\infty \in M'$. Also ist auch g injektiv. ■

Die Existenzaussage dieses Beweises läßt sich auch einfacher zeigen, indem man die Mächtigkeit der Mengen $M\backslash M'$ und $M\backslash \mathsf{f}(M')$ betrachtet. Da M' endlich ist und f injektiv, besitzen diese die gleiche Mächtigkeit, und es gibt eine injektive Abbildung von der einen Menge in die andere. g läßt sich nun einfach aus diesen kombinieren, da deren Domain und Co-Domain jeweils disjunkt liegen.[2] Der Vorteil des ersten Beweises liegt aber darin, daß sich aus ihm ein konstruktives Verfahren ableiten läßt, um die injektive Erweiterung zu berechnen.

A.4 Ein Induktionsschema über zwei Parameter

Im Kapitel 5 über Unifikation zweier Wörter wurde im Beweis zu Satz 5.9 ein komplizierteres Induktionsschema verwendet, das hier etwas genauer betrachtet werden soll.

[1] Hier wurde bewußt auf die Minimalität von p keinen Wert gelegt. Obwohl nun p auf mehrere Arten gewählt werden kann, bleibt die Definition von m_∞ dennoch eindeutig.

[2] Als Beispiel für die Notwendigkeit der Endlichkeitsvoraussetzung betrachte zum Beispiel die Funktion $\mathsf{h} : \mathbb{N} \to \mathbb{N}$, mit $\mathsf{h}(2n) := n$ für alle $n \in \mathbb{N}$. Sie ist zwar injektiv, kann aber nicht injektiv auf ganz $\mathbb{N}$ fortgesetzt werden, da der Co-Domain von h schon ganz $\mathbb{N}$ umfaßt.

Satz A.6

Sei U eine beliebige Menge (Universum von Objekten) und $\mathcal{A}$ ein Prädikat über U. Weiter seien zwei Funktionen $\mathsf{f}, \mathsf{g}: U \to \mathbb{N}$ gegeben und es gelte

$$\left.\begin{array}{lll} \forall x & (\quad \mathsf{f}(x) < n & \to \quad \mathcal{A}(x) \quad), \\ & (\quad \mathsf{f}(x) = n \wedge \mathsf{g}(x) < m & \to \quad \mathcal{A}(x) \quad) \\ & \Longrightarrow & \\ \forall x & (\quad \mathsf{f}(x) \leq n \wedge \mathsf{g}(x) \leq m & \to \quad \mathcal{A}(x) \quad) \end{array}\right\} \quad (m,n) \neq (0,0) \tag{A.2}$$

$$\forall x \quad \mathsf{f}(x) = 0 \wedge \mathsf{g}(x) = 0 \quad \to \quad \mathcal{A}(x) \tag{A.3}$$

dann gilt $\forall x\ \mathcal{A}(x)$.

Beweis: Prädikatenlogische Umformungen führen $(A.2)$ in folgende Form über:

$$\begin{array}{c} \exists x\ \mathsf{f}(x) \leq n \wedge \mathsf{g}(x) \leq m \wedge \neg\mathcal{A}(x) \\ \Longrightarrow \\ \exists x\ (\mathsf{f}(x) < n \vee (\mathsf{f}(x) = n \wedge \mathsf{g}(x) < m)) \wedge \neg\mathcal{A} \end{array} \tag{A.4}$$

Angenommen der Satz würde nicht gelten. Die Voraussetzungen seien also erfüllt, aber die Konklusio sei falsch, dann gibt es eine Teilmenge V von U, so daß $U \neq \emptyset$ und für alle $x \in V$ gilt $\neg\mathcal{A}(x)$. Das Induktionsaxiom für die natürlichen Zahlen besagt, daß jede nichtleere Teilmenge von $\mathbb{N}$ ein Minimum besitzt. Deshalb besitzt $\mathsf{f}(V)$ ein Minimum n, und es gibt eine nichtleere Teilmenge W von V mit $\mathsf{f}(W) = \{n\}$. Dasselbe Argument auf $\mathsf{g}(W)$ angewendet, liefert ein Minimum m von $\mathsf{g}(W)$. Damit hat man ein $y \in U$ gefunden mit $\mathsf{f}(y) = n$, $\mathsf{g}(y) = m$ und $\neg\mathcal{A}(y)$.

Falls $(m,n) = (0,0)$, so ergibt das einen Widerspruch zu $(A.3)$. Andernfalls kann man $(A.4)$ anwenden und bekommt ein $x \in U$ mit $\neg\mathcal{A}(x)$, d.h. $x \in V$. Das „Oder“ aus der Konklusio von $(A.4)$ aufgelöst ergibt zwei Fälle. Einmal ist $\mathsf{f}(x) < n$, im Widerspruch zur Minimalität von n. Oder es ist $\mathsf{f}(x) = n$, d.h. $x \in W$, und es gilt $\mathsf{g}(x) < m$. Das führt zu einem Widerspruch zur Minimalität von m. ■

A.5 Anzahl gemeinsamer Teilfolgen wächst exponentiell

Satz A.7

Falls $|\mathcal{S}| > 1$ ist, dann gibt es zu jedem Wort $u \in \mathcal{S}^*$ in Abhängigkeit von $|u|$ im allgemeinen exponentiell viele Teilfolgen g $(g \mid u)$.

Beweis: Für den Beweis, daß es eine exponentielle untere Schranke gibt, sei o.B.d.A. $S = \{0,1\}$. Zu zeigen ist dann, daß es für jedes hinreichend große $n \in \mathbb{N}$ ein $u \in \{0,1\}^n$ gibt, für das die Anzahl der Teilfolgen von u größer oder gleich q^n ist, mit einem $q > 1$.

Definiere u als $(01)^m$, mit $m := \frac{n}{2}$. Damit sind insbesondere alle $g \in \{0,1\}^*$, mit $|g| = m$ Teilfolgen von u. Von diesen gibt es genau 2^m Stück. Mit $q := \sqrt{2}$ und $2^m = 2^{\frac{n}{2}} = (\sqrt{2})^n = q^n$ folgt, daß es mindestens exponentiell viele Teilfolgen eines Wortes gibt.

Auf der anderen Seite stammt jeder Buchstabe einer Teilfolge g von u von genau einer Position im Wort u. Die Anzahl Möglichkeiten diese Positionen auszuwählen ist 2^n, und so gibt es höchstens 2^n Teilfolgen von einem Wort u der Länge n. ■

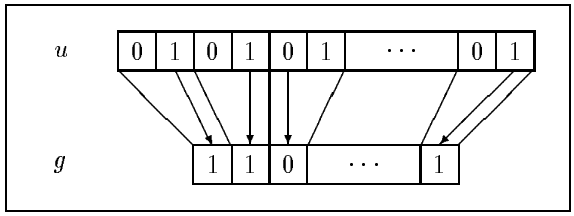


Abbildung A.2: Beispiel zur Konstruktion der 2^m Teilfolgen im Beweis zur exponentiellen Anzahl der Teilfolgen eines Wortes. Dabei sei $|g| = \frac{|u|}{2}$.

A.6 Prologimplementierung der Unifikation

Dieser Abschnitt beschreibt eine Prologimplementierung des in Kapitel 5 vorgestellten Algorithmus.

Prolog wurde deshalb als Implementierungssprache gewählt, weil die beschriebenen Verfahren regelbasiert sind. Die in Prolog eingebaute Tiefensuche läßt dann eine einfache Übersetzung der Regeln zu. Der verwendete Sprachdialekt ist KAP, ein am Institut für Logik, Komplexität und Deduktionssysteme entwickeltes kompilierendes Prologsystem ([KAP91]).

Wörter werden dargestellt als Listen über Konstanten und Variablen. Somit ist das leere Wort die leere Liste „`[ ]`“. Substitutionen bestehen aus Listen aus Paaren (Liste aus zwei Elementen) von Variablen und der Liste, durch die die entsprechende Variable ersetzt wird. Als Konsistenzbedingung wird zusätzlich gefordert, daß die einzelnen Variablen paarweise verschieden sind.

Der folgende Prologcode stellt nur einen Ausschnitt mit den wichtigsten Klauseln dar. Die Eingabe- und Ausgabemodule wurden ganz weggelassen. Ebenso fehlt die Definition der Variablen- und Konstantenrepräsentation.

Das Prädikat `delta/6` beschreibt genau einen Zustandsübergang. Also etwas unscharf:

$$\texttt{delta}(\texttt{U},\texttt{V},\texttt{C},\texttt{Sub},\texttt{R},\texttt{S})\ \textbf{gdw.}\ (\texttt{U},\texttt{V}) \xrightarrow[\texttt{C}]{\texttt{Sub}} (\texttt{R},\texttt{S})$$

Die **SaW** wird dann durch `delta_star_plus/6` modelliert:

$$\texttt{delta\_hull\_plus}(\texttt{U},\texttt{V},\texttt{W},\texttt{Subs},\texttt{R},\texttt{S})\ \textbf{gdw.}\ (\texttt{U},\texttt{V}) \xRightarrow[\texttt{W}]{\texttt{Subs}} (\texttt{R},\texttt{S})$$

```
module unification.

export delta_hull_plus/6.
export delta/6.

from var_cons import is_var/1.

private apply_sub/4.
private apply_sub/3.
private apply_subs/3.
private apply_subs_on_sub/3.
private apply_subs_on_singleton/3.

private delta_not_symmetric/6.

body.
```

```
% Anwendung von einer Substitution auf ein Wort
apply_sub([ ],X,X).
apply_sub([X,R],V,W) :-
   apply_sub(X,R,V,W).
apply_sub(_,_,[ ],[ ]) :- !.
apply_sub(X,R,[X|TOld],TNew) :-
   !,
   apply_sub(X,R,TOld,T),
   append(R,T,TNew).
apply_sub(X,R,[H|T],[H|TNew]) :-
   apply_sub(X,R,T,TNew).

% Anwendung von Substitutionen auf ein Wort
apply_subs([ ],W,W).
apply_subs(_,[ ],[ ]).
apply_subs(Subs,[HOld|TOld],TNew) :-
   apply_subs_on_singleton(Subs,HOld,H),
   apply_subs(Subs,TOld,T),
   append(H,T,TNew).

% Anwendung von Substitutionen auf eine Konstante
apply_subs_on_singleton(_,[ ],[ ]) :-  !.
apply_subs_on_singleton([ ],S,[S]).
apply_subs_on_singleton([[X,R]|_],X,R) :-  !.
apply_subs_on_singleton([_|T],S,R) :-
   apply_subs_on_singleton(T,S,R).

apply_subs_on_sub(Subs,[ ],Subs).
apply_subs_on_sub(Subs,[X,R],[[X,RNew]|SubsNew]) :-
   apply_subs(Subs,R,RNew),
   delete(Subs,[X,_],SubsNew).

% delta ist symmetrischer Abschluss von delta_not_symmetric
delta(U,V,C,Sub,R,S) :-
   delta_not_symmetric(U,V,C,Sub,R,S).
delta(U,V,C,Sub,R,S) :-
   delta_not_symmetric(V,U,C,Sub,S,R).

% (1) Vergleich:
% delta(cr,cs,c,{},r,s) if c<>[ ]

delta_not_symmetric([C|R],[C|S],C,[ ],S,R).

% (2) Epsilon-Substitution:
% delta(Xr,s,[ ],{X/[ ]},r{X/[ ]},s{X/[ ]} if is_var(X)

delta_not_symmetric([X|R],S,[ ],[X,[ ]],RNew,SNew) :-
   is_var(X),
   apply_sub(X,[ ],R,RNew),
   apply_sub(X,[ ],S,SNew).

% (3) Aufspaltung:
```

```
% delta(Xr,cs,c,{X/cX},[X|r{X/cX}], s{X/cX}) if is_var(X) & c<>X & C<>[ ]

delta_not_symmetric([X|R],[C|S],C,[X,[C,X]],[X|RNew],SNew) :-
   is_var(X),
   not(C=X),
   apply_sub(X,[C,X],R,RNew),
   apply_sub(X,[C,X],S,SNew).

delta_hull_plus([ ],[ ],[ ],[ ],[ ],[ ]) :-
   !.
delta_hull_plus(U,V,W,Subs,R,S) :-
   delta(U,V,C,Sub,UNew,VNew),
   delta_hull_plus(UNew,VNew,WOld,SubsOld,R,S),
   apply_subs_on_sub(SubsOld,Sub,Subs),
   apply_subs_on_singleton(SubsOld,C,CNew),
   append(CNew,WOld,W).

end_module.
```

Ein ähnliches Programm wurde für das Matching geschrieben, das aber auf den gleichen Ideen beruht und hier nicht mehr aufgeführt werden soll.

A.7 Auflistung der verwendeten binären Relationen

Es geht genauer um die binären Relationen über $\mathsf{Mon}(\mathcal{V}, \mathcal{S})$. Die in der Spalte mit der Beschriftung „Definition“ aufgeführte Zahl ist die Seitenzahl der Definition der entsprechenden Relation.

Syntax	Definition	Bedeutung
$\equiv$	8	Syntaktische Gleichheit
$\sqsubseteq$	9	Substring Beziehung
$\prec$	9	$u \prec v :\Leftrightarrow u$ ist kürzer als v
$\leq$	10	Filter, Matching
$=$	10	Semantische Gleichheit
$\subseteq$	17	reflexives „ist einfacher“
$\vdash$	19	einfache Simplifikation
$\rhd$	27	direkte Simplifikation

Anhang B

Verzeichnisse

Literaturverzeichnis

[ASU88] Aho A.V., Sethi R. und Ullman J.D., *Compilerbau, Teil1* (Addison-Wesley, San Juan, 1988) 113–118.

[Ba91] Baader F., Unification, Weak Unification, Upper Bound, Lower Bound, and Generalization Problems, in: *4th International Conference on Rewriting Techniques and Applications*, Como 1991, Lecture Notes in Computer Science **488**.

[KAP91] A. Bockmayr, C. Brzoska, E. Heinz, P. Lukowicz, H. H. Rath, C. Scharnhorst, B. Schiele, *KA-Prolog: Sprachdefinition*, Interner Bericht 19/91, Fakultät für Informatik, Universität Karlsruhe, 1991.

[BuLi82] Buchberger, B. und Loos, R., Algebraic Simplification, in: *Computer Algebra, Symbolic and Algebraic Computation* (Springer Verlag, Wien, 2nd ed., 1983) 11–16.

[DeJo90] Dershowitz, N. und Jouannaud, J.-P., Rewrite Systems, in: *Handbook of Theoretical Computer Science, Vol. B* (North-Holland, Amsterdam, 1990), 243–320.

[Schm92] Schmitt, P. H. Theorie der logischen Programmierung (Springer–Verlag, Berlin Heidelberg, 1992)

[StEk93] Eker S., Associative Matching for Linear Terms. *ERCIM Research Reports, ERCIM-93-R012*.

[Jac185] Jacobson, N., Basic Algebra, Algebra I (W. H. Freeman and Company, New York, 1985).

[Jaff90] Jaffar, J., Minimal and Complete Word Unification, in: *J. ACM 37*, **1** (Januar 1990), 47–85.

[Maka77] Makanin, G.S., The problem of solvability of equations in a free semigroup. *Math. USSR Sbornik 32*, **2** (1977), 129–198 (In AMS, (1979)).

[Meak88] Meakin, J., Automata and the word problem, in: *Proc. 16-th LITP Spring School on Theoretical Computer Science*, Ramatouelle, France, Lecture Notes in Computer Science **386** (Springer-Verlag, Berlin, 1988).

[Perr90] Perrin, D., Finite Automata, in: *Handbook of Theoretical Computer Science, Vol. B* (North-Holland, Amsterdam, 1990), 1–57.

[Plot70] PLotkin, G.D., A Note on Inductive Generalization, *Machine Intelligence*, **5**(1970), 153–163.

[Plot72] Plotkin, G.D., Building-in Equational Theories. *Machine Intelligence*, **7**(1972), 73–90.

[Pott89] Pottier, L., La généralisation de termes en théorie équationelle. Cas associatif-commutatif. *INRIA Sophia Antipolis*, Juin 1989.

[Siek89] SIEKMANN, J. H., Unification Theory: A Survey. *Special issue on unification, Journal of Symbolic Computation*, **7** (1989).

[Tichy84] TICHY, W. F., The String–to–String Correction Problem with Block Moves. *ACM Transactions on Computer Systems*, **2** (1984), 309—321.

opagebreak[4]

Index