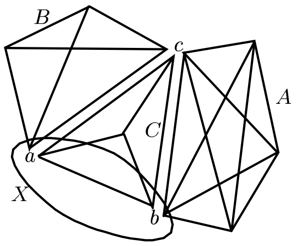
Effiziente Normalform-Algorithmen für Ersetzungssysteme über frei partiell kommutativen Monoiden



Von der Fakultät für Informatik
der Universität Stuttgart zur Erlangung
der Würde eines Doktors der Naturwissenschaften
genehmigte Abhandlung

von
Michael W. Bertol
aus München

Hauptberichter: Prof. Dr. V. Diekert
Mitberichter: Prof. Dr. M. Jantzen

Institut für Informatik der Universität Stuttgart, 1996

Danksagung

Ich danke meinen Kollegen der beiden Theorie-Abteilungen für ihr Interesse an meinen Themen, insbesondere danke ich Anca Muscholl und Holger Petersen für eine sehr gute Zusammenarbeit. Für interessante Anregungen und Gespräche bedanke ich mich bei Dan Teodosiu und Zoltán Ésik.

Dank der intensiven Unterstützung von Prof. Dr. Volker Diekert konnte die vorliegende Arbeit in so kurzer Zeit fertiggestellt werden. Seinem regen Interesse und seiner optimistschen Grundhaltung verdanke ich sehr viel.

Für die Übernahme des Berichtes möchte ich mich bei Prof. Dr. Matthias Jantzen bedanken.

Mein größter Dank gilt meiner Familie für ihre riesige Unterstützung.

Zusammenfassung

In dieser Arbeit werden frei partiell kommutative Monoide (Spurmonoide) und Ersetzungssysteme über ihnen betrachtet. Diese algebraischen Modelle zählen zu den temporalen Strukturen, mit denen sich insbesondere Nebenläufigkeit in verteilten oder parallelen Systemen mathematisch exakt modellieren läßt.
Ersetzungssysteme über diesen Strukturen erlauben es, Prozeßtransformationen in nebenläufigen Kontexten zu beschreiben. Es werden zwei Problemstellungen untersucht, das Konfluenzproblem und das Normalformenproblem.
Wir geben einen alternativen Beweis für die Unentscheidbarkeit des Konfluenzproblems und zeigen, daß sogar schon bei vier Buchstaben die Konfluenz eines verkürzenden Ersetzungssystems nicht mehr entscheidbar ist (bei einer Kommutation). Es werden verschiedene konkrete Darstellungen von frei partiell kommutativen Monoiden gegeben und Faktorisierungsdarstellungen der Elemente eingeführt, die die Konstruktion von Datenstrukturen für effiziente Algorithmen ermöglichen. Dabei führen besondere Eigenschaften der linken Seiten eines Ersetzungssystems wie der „Zusammenhang" aller linken Seiten oder eine „Sichtbarkeitseigenschaft" zu effizienten Algorithmen. Es wird eine Dekompositionsmethode entwickelt, die auf „Cographenmonoiden" führt, und mit der der Algorithmus von Book so verallgemeinert werden kann, daß die nicht-uniforme Zeitkomplexität linear bleibt.

Inhaltsverzeichnis

1 Einleitung

Der sequentiellen Bearbeitung rechenintensiver Probleme sind technologische Grenzen gesetzt. Ein Weg zur Überwindung dieser Grenzen ist Parallelisierung. Nebenläufigkeit, Parallelität und Verteiltheit haben sich innerhalb der letzten Dekade als sehr wichtige Konzepte in der Informatik erwiesen. Deshalb war es notwendig, eine mathematisch logisch fundierte Formalisierung solcher Systeme und insbesondere ihrer temporalen Struktur zu entwickeln. Diese Formalisierungen können zur Beschreibung und Validierung von Eigenschaften oder als Kalkül verwendet werden.

Einer der ersten dieser Ansätze waren die Petrinetzsemantiken, die von Hoare [Hoa78] definierten kommunizierenden sequentiellen Prozesse CSP und der von Milner und Hennessy [Mil80] eingeführte Kalkül CCS. Pratt schlug eine auf Halbordnungen basierende Semantik vor [Pra86]. Es folgte eine Vielzahl von sehr allgemeinen algebraischen und logischen Modellen wie Ereignisräume, monoidale Kategorien, Chu-Räume und viele mehr.

Ursprünglich wurden die frei partiell kommutativen Monoide oder Spurmonoide von Cartier und Foata [CF69] verwendet, um kombinatorische Probleme zu behandeln. Aufgrund ihrer algebraischen Eigenschaften und ihrer Verwandtschaft zur Theorie der formalen Sprachen erscheinen sie für weitere Untersuchungen attraktiv.

Gleichzeitig hat sich die Theorie der Spuren als ein adäquates Modell zur Analyse einer großen Klasse von nebenläufigen Vorgängen erwiesen. Der erste, der diese Verbindung entdeckt hat, war Mazurkiewicz. In [Maz77] wurden die frei partiell kommutativen Monoide als eine Semantik für nebenläufige Prozesse erkannt.

Die Verwandtschaft zu freien Monoiden motivierte Diekert in [Die87], einen Teil der Theorie der Semi-Thue-Systeme oder Wortersetzungssysteme für frei partiell kommutative Monoide zu verallgemeinern. Der Hintergrund war, Formalismen zur Beschreibung von statischen Ersetzungen in nebenläufigen Kontexten zu finden. Mit Spurersetzungssystemen können Programm- bzw. Prozeßtransformationen in der Spursemantik beschrieben werden. Die korrekte und effiziente Transformation von nebenläufigen und konkurrierenden Programmschemata ist eine Voraussetzung für den Umgang mit parallelen und verteilten Rechnermodellen bzw. Algorithmen.

1.1 Semantik nebenläufiger Vorgänge

Betrachten wir ein konkretes einführendes Beispiel. Sei ein nebenläufiges Programmsystem (beispielsweise ein nebenläufiges Datenbank-Transaktionssystem) durch die folgenden Transaktionen

$$\begin{array}{llll} T_a & x & := & f(x,z) \\ T_b & y & := & g(y,z) \\ T_c & z & := & h(x,y) \\ T_d & x & := & i(z) \end{array}$$

beschrieben. Dann, vorausgesetzt die Funktionen f, g, h, i besitzen keine Seiteneffekte, existiert die folgende die (Daten-)Abhängigkeiten beschreibende Funktion mit

$$\begin{array}{rcl} \alpha : \{a,b,c,d\} & \longrightarrow & \mathcal{P}\{x,y,z\} \\ a & \longmapsto & \{x,z\} \\ b & \longmapsto & \{y,z\} \\ c & \longmapsto & \{z,x,y\} \\ d & \longmapsto & \{x,z\} \end{array}$$

Wir identifizieren T_x mit x für $x \in \{a,b,c,d\}$. Das Bild einer Transaktion ist ihr Datenraum, hier also die Variablen. Durch diese Funktion wird ein Abhängigkeitsgraph (V,E) via $V = \{a,b,c,d\}$ und

$$(u,v) \in E \iff \alpha(u) \cap \alpha(v) \neq \emptyset \text{ für alle } u,v \in V$$

definiert. Sei nun $w \in \{a,b,c,d\}^*$ eine Ausführungssequenz, so ist das Resultat nicht von der Reihenfolge innerhalb Teilsequenzen, die nur die Transaktionen d und b enthalten, abhängig, denn es gilt $(b,d) \notin D$ oder äquivalent dazu $\alpha(b) \cap \alpha(d) \neq \emptyset$ — diese Aktionen können nebenläufig abgearbeitet werden. D.h. Sequenzen modulo der Kommutation $db = bd$ besitzen dieselbe Semantik — warum sollten sie unterschieden werden? Den diese Semantik spezifizierenden Graphen $(V,E) = (\Sigma, D)$ werden wir als Abhängigkeitsalphabet einführen.

Abstrahiert man von den Variablen im Beispiel, so erkennt man, daß jede exklusive Ressource (in einem System) diesen Typ von Abhängigkeiten induziert. Damit verfügen wir über eine mathematische Beschreibung nebenläufiger und konkurrierender Vorgänge bzw. äquivalenter serieller Sequenzen. Die durch D definierte Kommutationsrelation $ab = ba$ für alle $(a,b) \notin D$ induziert eine Kongruenz via $uabv = ubav$, $u,v \in \Sigma^*$, auf dem freien Monoid $\Sigma^* \cong \bigcup_{i \in \mathbb{N}} \Sigma^i$, und die Katenation, d.h. die sequentielle Komposition, wird als Operation ererbt.

Die Verwandtschaft dieser Struktur zu freien Monoiden motivierte eine Reihe von Mathematikern und Informatikern, andere bekannte Konzepte auf diese algebraischen Strukturen zu übertragen. Eines dieser Konzepte sind die hier betrachteten Ersetzungssysteme.

Das Einführen eines Ersetzungssystems führt zu einer operationellen Semantik für nebenläufige Vorgänge. Jede Transaktion läßt sich als ein Modifikator, eine Ersetzung, auffassen.

Betrachten wir als zweites Beispiel Petrinetze, das vermutlich populärste Modellierungswerkzeug in diesem Kontext. Sei $N = (S,T,F,M)$ ein Petrinetz mit den Stellen S, den Transitionen T, den Kanten $F \subseteq (T \times S) \cup (S \times T)$ und einer Markierung $M : S \to \mathbb{N}$.

Dann kann eine Berechnung dieses Petrinetzes[1] mit dem Ersetzungssystem

$$\mathcal{R} \quad = \quad \left\{ \left((t_v(s))_{s \in S} \rightarrow (t_n(s))_{s \in S} \right) \mid t \in T \right\}$$

über dem Monoid $\mathbb{M} = \mathbb{N}^M \cong \prod_{s \in M} \{s\}^*$ mit

$$t_v(s) = \begin{cases} 1 & sFt \\ 0 & \neg sFt \end{cases} \quad \text{und} \quad t_n = \begin{cases} 1 & tFs \\ 0 & \neg tFs \end{cases}$$

simuliert werden.

Ein anderes Spezifizierungswerkzeug sind die erkennbaren Mengen (über einem hier partiell kommutativen Monoid $\mathbb{M}(\Sigma, D)$). Sei $h : \mathbb{M}(\Sigma, D) \longrightarrow N$ ein Homomorphismus in ein endliches Monoid $N = (N, \cdot, 1)$ und $X \subseteq N$ eine Teilmenge so, daß die Sprache $L = h^{-1}(X)$ erkannt wird. Dann kann mit dem Ersetzungssystem

$$\{xa \rightarrow x \cdot h(a) \mid x \in N,\ a \in \Sigma\}$$

kontrolliert werden, ob ein Element $t \in \mathbb{M}(\Sigma, D)$ die Spezifikation $t \in L$ erfüllt, denn $1_N t \overset{*}{\Longrightarrow}_{\mathcal{R}} x \in X$ gilt genau dann, wenn $t \in L$.

Sei $t = [t_1 \dots t_i \dots t_n]$ die Spur, die eine Prozeßsequenz $t_1 \dots t_n$ modelliert. Dabei können einzelne Prozesse durchaus nebenläufig sein. Das bedeutet, t ist die Menge aller durch die Abhängigkeitsrelation $D \subseteq \Sigma \times \Sigma$ beschriebenen äquivalenten Sequentialisierungen der Folge $t_1 \dots t_n$. Eine Verfeinerung von Prozeß t_i durch Prozeß x sei durch eine Regel $t_i \rightarrow x$ beschrieben. x kann wiederum aus Prozessen $[x_1 \dots x_m]$ bestehen, die wieder nebenläufig seien können. Diese Prozesse können auch mit den Prozessen $t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_n$ kommutieren. Die Anwendung der Regel transformiert die Prozeßsequenz entsprechend.

$$[t_1 \dots t_{i-1}][t_i][t_{i+1} \dots t_n] \implies [t_1 \dots t_{i-1}][x_1 \dots x_m][t_{i+1} \dots t_n]$$

Mit der gleichen Methode lassen sich auch Teilprozesse abstrahieren, was sich durch Umkehren der Regel zu $x \rightarrow t_i$ erreichen läßt.

$$[t_1 \dots t_{i-1} x_1 \dots x_m t_{i+1} \dots t_n] \implies [t_1 \dots t_{i-1} t_i t_{i+1} \dots t_n]$$

Mit dem Formalismus „Ersetzungssystem modulo Kommutationen“ lassen sich Programmtransformationen, d.h. Compiler und Optimierer beschreiben.

Betrachten wir eine einfache prozedurale Programmiersprache mit den Variablen a, b, c, d, e und den Operationen $+, -$, sowie den Anweisungen **read** und **write** und dem Zuweisungsoperator :=. Das Rechnermodell verfüge über die Prozessoren P_A, $A \subseteq \{a, b, c, d, e\}$, einen Eingabekanal, einen Ausgabekanal und den entsprechenden Speicher für die Variablen. Jeder Prozessor P_A hat einen Akkumulator und verfügt über den Instruktionssatz

$$\text{LOAD}_A\ v,\ \text{STORE}_A\ v,\ \text{ADD}_A\ v,\ \text{SUB}_A\ v,\ \text{READ}_A \text{ und } \text{WRITE}_A.$$

[1] Feuersequenzen $f \in T^*$ (bei 1-sicheren Netzen) können als eine Spur interpretiert werden.

Ein Compiler könnte die Anweisungen wie folgt transformieren:

Programmkonstrukt S	Maschinenkonstrukt $M(S)$
$v_1 := v_2 + v_3$	$\mathrm{LOAD}_A\ v_2$; $\mathrm{ADD}_A\ v_3$; $\mathrm{STORE}_A\ v_1$
$v_1 := v_2 - v_3$	$\mathrm{LOAD}_A\ v_2$; $\mathrm{SUB}_A\ v_3$; $\mathrm{STORE}_A\ v_1$
read v	$\mathrm{READ}_{\{v\}}$; $\mathrm{STORE}_{\{v\}}\ v$
write v	$\mathrm{LOAD}_{\{v\}}\ v$; $\mathrm{WRITE}_{\{v\}}$

($A = \{v_1, v_2, v_3\}$ in der ersten und zweiten Zeile.)

Verwenden wir die im ersten Beispiel vorgestellte Methode um die einzelnen Aufgaben auf die Prozessoren zu verteilen, indem wir die Zuweisungen $v_1 := v_2 \circ v_3$, $\circ \in \{+, -\}$ mit Buchstaben $a_{v_1:=v_2\circ v_3}$ und die Anweisung **read** v mit b_v und **write** v mit c_v assoziieren, so erhalten wir die Abhängigkeiten

$$\begin{aligned}
(a_{v_1:=v_2\circ v_3}, a_{v_1':=v_2'\circ' v_3'}) \in D &\iff \{v_1, v_2, v_3\} \cap \{v_1', v_2', v_3'\} \neq \emptyset \\
(a_{v_1:=v_2\circ v_3}, b_v), (b_v, a_{v_1:=v_2\circ v_3}) \in D &\iff v \in \{v_1, v_2, v_3\} \\
(a_{v_1:=v_2\circ v_3}, c_v), (c_v, a_{v_1:=v_2\circ v_3}) \in D &\iff v \in \{v_1, v_2, v_3\} \\
(b_v, b_v), (c_v, c_v), (b_v, c_v), (c_v, b_v) \in D &
\end{aligned}$$

Die „Buchstaben" $\mathrm{LOAD}_A\ v$, $\mathrm{ADD}_B\ v$, $\mathrm{STORE}_C\ v$ kommutieren (nach Spezifikation des Maschinenmodells), wenn die Indizes nichtleeren Schnitt besitzen. Die „Buchstaben" $\mathrm{READ}_A\ v$ und $\mathrm{WRITE}_B\ v'$ sind genau dann abhängig, wenn $A \cap B \neq \emptyset$. Ansonsten ist, da es nur einen Eingabe- und einen Ausgabekanal gibt, $\mathrm{READ}_A\ v$ von $\mathrm{READ}_B\ v'$ bzw. $\mathrm{WRITE}_A\ v$ von $\mathrm{WRITE}_B\ v'$ für alle $A, B \subseteq \{a, b, c, d, e\}$ abhängig.

Das Ersetzungssystem $\mathcal{R}$ mit den Regeln $a_S \to M(S)$ (siehe Tabelle) simuliert einen automatisch (kanonisch) parallelisierenden Compiler. Dieser könnte noch um eine Optimierung erweitert werden, indem entsprechende Regeln eingeführt werden, die beispielsweise die kombinierten READ-STORE-Anweisungen durch eine einzige READ-Anweisung auf dem die gelesene Variable bearbeitenden Prozessor unter bestimmten Bedingungen (Kontexten) ersetzen.

Die Existenz von sehr effizienten Ersetzungsalgorithmen ermöglicht sogar ein Austauschen von Teilabläufen durch einen Dispatcher während des Abarbeitens der Prozesse. D.h. eine weitere denkbare Applikation ist die dynamische Optimierung von Prozeßsequenzen. Sei $t = t_1 \ldots t_n$ beispielsweise eine Sequenz von noch nicht abgearbeiteten Transaktionen in einem Datenbanksystem, die von links fortlaufend abgearbeitet und von rechts erweitert wird. Weiter seien in gewissen Kontexten effizientere Transaktionen möglich, die durch ein System von Regeln beschrieben sind. Im laufenden Betrieb eines solchen Systems treten Sequenzen von noch nicht bearbeiteten Transaktionen auf. Diese Sequenzen könnten dann mit Optimierungsregeln in effizientere Sequenzen transformiert werden, so daß die ursprüngliche Semantik erhalten bleibt. Diese Anwendung motiviert die Entwicklung von Algorithmen, die effizient Links-Reduktionen durchführen können.

Auch vom mathematischen Standpunkt ist das Studium von Ersetzungssystemen modulo Kommutation interessant. Die Theorie der Spurersetzungssysteme ist eine natürliche

Verallgemeinerung der Ersetzungssysteme über freien Monoiden, den sogenannten Semi-Thue-Systemen oder Wortersetzungssystemen. So konnten viele Resultate, die bereits für freie Monoide bekannt waren, auf die partiell kommutative Theorie verallgemeinert werden. Eines der interessantesten Resultate ist eine Version des Knuth-Bendix-Reduktionsalgorithmus von Diekert [Die87].

Eine der Herausforderungen ist der Entwurf effizienter Algorithmen für Ersetzungssysteme über frei partiell kommutativen Monoiden. In dieser Arbeit werden einige Methoden vorgestellt, wie man effizient algorithmisch Normalformen berechnet und warum es kein dem Konfluenzentscheidungsverfahren im Falle freier Monoide entsprechendes effektives Pendant im partiell kommutativen Fall geben kann.

1.2 Überblick

Die Arbeit wird von einer Reihe von Definitionen eingeleitet. Anschließend werden einige Repräsentationen diskutiert. Hiernach werden Ersetzungssysteme definiert und ein Unentscheidbarkeitsresultat von Narendran und Otto [NO88] verfeinert.

In den folgenden Abschnitten beschäftigen wir uns mit dem „Normalformenproblem“. Im Kapitel 5 wird für Ersetzungssysteme mit zusammenhängenden linken Seiten eine Datenstruktur vorgestellt, die effiziente Manipulationen ermöglicht und zu einem „$n \log n$“-Algorithmus führt. In dem darauffolgenden Kapitel wird eine Einschränkung, die „Sichtbarkeit“ untersucht. Sie führt zu einem „optimalen“ Reduktionsalgorithmus, der den Algorithmus von Book verallgemeinert.

Eine interessante Unterklasse der Abhängigkeitsalphabete entspricht einer schon 1970 von Dijkstra vorgeschlagenen Spezifizierungsrestriktion. Die von ihm eingeführten parallelen Programmkonstrukte

parbegin S1 || S2 || ... || Sn **parend** und ;

führen zu Abhängigkeitsalphabeten mit Cographstruktur. Sie finden sich im Cographenansatz in Kapitel 8 als Operatoren für das freie und das direkte Produkt wieder. Betrachtet man Prozessoren als Ressource R, besitzt die Ressourcenabbildung $\alpha : \Sigma \longrightarrow \mathcal{P}(R)$ (siehe oben) dann eine der folgenden Strukturen:

(i) Entweder existiert eine Zerlegung (der Anweisungen) $\Sigma = \Sigma_1 \dot\cup \Sigma_2$ so, daß die Menge der Ressourcen $\alpha(\Sigma) = \{\alpha(a) \subseteq R \mid a \in \Sigma\}$ in zwei Teile $T_1 \dot\cup T_2$ mit

$$\forall X \in T_1 \forall Y \in T_2 : X \cap Y = \emptyset$$

zerfällt (d.h. absolute Parallelität), und die restringierten Abbildungen $\alpha_{|\Sigma_1}$ und $\alpha_{|\Sigma_2}$ besitzen jeweils eine dieser strukturellen Eigenschaften rekursiv,

(ii) oder, wenn keine solche Zerlegung existiert, dann existiert eine andere Zerlegung $\Sigma = \Sigma_1 \dot\cup \Sigma_2$ so, daß die Menge der Ressourcen $\alpha(\Sigma) = \{\alpha(a) \subseteq R \mid a \in \Sigma\}$ in zwei Teile $T_1 \dot\cup T_2$ mit

$$\forall X \in T_1 \forall Y \in T_2 : X \cap Y \neq \emptyset$$

zerfällt, und die restringierten Abbildungen $\alpha_{|\Sigma_1}$ und $\alpha_{|\Sigma_2}$ besitzen jeweils eine der drei strukturellen Eigenschaften rekursiv (Damit ist das Mengensystem $T_1 \cup T_2$ in einem gewissen Sinn isomorph zum kartesischen Produkt $\Sigma_1 \times \Sigma_2$.),

(iii) oder die Anweisung ist atomar, d.h. unteilbar.

Diese rekursive Struktur von Abhängigkeitsalphabeten induziert ihrerseits Zerlegungen von Spuren, die zu einer einfacheren, modularen Charakterisierung der Faktoreigenschaft und einer interessanten Algorithmenstruktur führen.

Für die von Cographen-Abhängigkeitsalphabete erzeugte partiell kommutativer Monoide konnten Dekompositionstechniken entwickelt werden, die den Entwurf eines linearen Normalformalgorithmus ermöglichen.

Wir schließen mit einem Ausblick und einer Zusammenfassung der Resultate über Faktorisierungen. In einem Anhang sind einige Ideen zu Verallgemeinerungen der vorgestellten Verfahren und technische Beweise zusammengefaßt.

2 Spuren und ihre Darstellungen

Wir leiten diesen Abschnitt mit einer kurzen Einführung der grundlegenden Begriffe ein, zusammen mit einigen Notationen.

Ein *Abhängigkeitsalphabet* ist ein geordnetes Paar (Σ, D), wobei Σ ein endliches Alphabet und $D \subseteq \Sigma \times \Sigma$ eine reflexive und symmetrische Relation bezeichnet, die *Abhängigkeitsrelation*. Die komplementäre Relation $I = (\Sigma \times \Sigma) \setminus D$ wird als *Unabhängigkeitsrelation* oder als *Kommutationsrelation* bezeichnet. Wir bezeichnen das von Σ erzeugte freie Monoid mit Σ^*. Sei $\equiv_I \subseteq \Sigma^* \times \Sigma^*$ die Äquivalenzrelation, die durch die Menge geordneter Paare $\{(uabv, ubav) \mid u, v \in \Sigma^*,\ (a,b) \in I\}$ induziert wird. Die Relation $\equiv_I$ ist eine Kongruenz über Σ^* und das zugehörige Quotientenmonoid $\mathbb{M}(\Sigma, D) = \Sigma^*/{\equiv_I}$ ist das von (Σ, D) erzeugte *frei partiell kommutative Monoid* oder einfach *Spurmonoid*. Die Elemente dieses Monoids nennen wir *Spuren*.

Eine Spur $t \in \mathbb{M}(\Sigma, D)$ besitzt das aus den Buchstaben eines Repräsentanten bestehende *induzierte Alphabet* $\Sigma(t) \subseteq \Sigma$ und $(\Sigma, D)(t)$ bezeichnet das *induzierte Abhängigkeitsalphabet* $(\Sigma(t), D \cap (\Sigma(t) \times \Sigma(t))$. Mit $|t|$ bezeichnen wir die Länge eines Repräsentanten und mit $|t|_a$ die Anzahl der Vorkommen des Buchstaben a, seine a-Länge. $\mathbb{N}$ ist die Menge der natürlichen Zahlen, und mit $[n]$ bezeichnen wir die Menge $\{1, \ldots, n\} \subseteq \mathbb{N}$ für $n \in \mathbb{N}$.

2.1 Konkrete Beschreibungen

Wir haben Spuren als Äquivalenzklassen über Worten definiert. Die Wahl eines Repräsentanten ist nicht eineindeutig. Deshalb betrachten wir Normalformen. Die historisch erste ist die von Foata [CF69]. Wir bezeichnen mit $\mathcal{F}$ die Menge

$$\{F \subseteq \Sigma \mid (F \times F) \setminus id_\Sigma \subseteq I\}$$

der *elementaren Schritte*. Zu jedem elementaren Schritt $F \in \mathcal{F}$ existiert eine Spur $[F] = \prod_{a \in F} a \in \mathbb{M}(\Sigma, D)$. Das folgende Theorem etabliert die Foata-Normalform.

Theorem 1 *Sei $t \in \mathbb{M}(\Sigma, D)$. Dann existiert genau eine Sequenz elementarer Schritte*

$$(F_1, \ldots, F_r) \quad \in \quad \mathcal{F}^*$$

mit $\prod_{i \in [r]} F_i = t$ so, daß für alle $2 \leq i \leq r$ zwei abhängige Buchstaben $(a,b) \in D$ mit $a \in F_{i-1}$ und $b \in F_i$ existieren. $\prod_{i \in [r]} t_i$ notiert hier das Produkt $t_1 \cdot \ldots \cdot t_r$. ∎

Eine natürliche Darstellung von Spuren sind *Abhängigkeitsgraphen* oder *partiell geordnete Multimengen*. Diese Repräsentationen besitzen auch eine einfache geometrische Interpretation.

Ein *Abhängigkeitsgraph* ist ein knotenbeschrifteter azyklischer Graph $G = [V, E, \lambda]$ mit Knotenmenge V, Kantenmenge $E \subseteq V \times V$ und $\lambda : V \longrightarrow \Sigma$ als Beschriftung. Für alle Knoten $x, y \in V$ soll entweder eine Kante von x nach y oder von y nach x führen, falls $(\lambda(x), \lambda(y)) \in D$. Gemeinsam mit der Verknüpfung

$$\begin{aligned}[V_1, E_1, \lambda_1] \cdot [V_2, E_2, \lambda_2] \quad &= \\ &[V_1 \dot\cup V_2, E_1 \dot\cup E_2 \dot\cup \{(x, y) \in V_1 \times V_2 \mid (\lambda_1(x), \lambda_2(y)) \in D\}, \lambda_1 \dot\cup \lambda_2]\end{aligned}$$

und dem neutralen Element $[\emptyset, \emptyset, \emptyset]$ bilden die Abhängigkeitsgraphen ein Monoid. Das Symbol $\dot\cup$ bezeichnet die disjunkte Vereinigung. Wir bezeichnen das durch die Einpunkt-Abhängigkeitsgraphen $[\{1\}, \emptyset, \lambda : 1 \mapsto a]$, $a \in \Sigma$, erzeugte Monoid vorläufig mit $\mathbb{G}(\Sigma, D)$.

Entsprechend läßt sich der transitive Abschluß eines Abhängigkeitsgraphen als *partiell geordnete Multimenge* über den Buchstaben auffassen. Eine Multimenge M über Σ ist eine Abbildung $M : \Sigma \longrightarrow \mathbb{N}$ in die natürlichen Zahlen. Wir notieren M durch eine Aufzählung der Elemente mit Vielfachheiten und identifizieren stellenweise ein Element einer Multimenge mit einem einer gewöhnlichen Menge. Ist die Multimenge M mit einer partiellen Ordnung $\leq\ \subseteq M \times M$ versehen, so sprechen wir von einer *partiell geordneten Multimenge* $\langle M, \leq \rangle$. Die mengentheoretischen Operationen $\cup$, $\cap$, $\setminus$ etc. und die Relationen wie $\subseteq$, $\in$ etc. verwenden wir für Multimengen und geordnete Multimengen wie bei gewöhnlichen Mengen.

Die partiell geordneten Multimengen bilden gemeinsam mit der leeren Menge $\langle \emptyset, \emptyset \rangle$ und der Verknüpfung

$$\begin{aligned}\langle M_1, \leq_1 \rangle \cdot \langle M_2, \leq_2 \rangle \quad &= \\ &\langle M_1 \dot\cup M_2, \leq_1 \dot\cup \leq_2 \dot\cup \{(a, b) \in M_1 \times M_2 \mid (a, b) \in D\} \rangle\end{aligned}$$

ein Monoid. Das von den einelementigen partiell geordneten Mengen erzeugte Monoid bezeichnen wir vorläufig mit $\mathbb{O}(\Sigma, D)$.

Die eineindeutige Fortsetzung der Abbildungen $h_{\mathbb{G}} : [a] \longmapsto [\{1\}, \emptyset, 1 \mapsto a]$ bzw. $h_{\mathbb{O}} : [a] \longmapsto \langle \{a\}, \{(a, a)\} \rangle$ zu Homomorphismen $h_{\mathbb{G}} : \mathbb{M}(\Sigma, D) \longrightarrow \mathbb{G}(\Sigma, D)$ und $h_{\mathbb{O}} : \mathbb{M}(\Sigma, D) \longrightarrow \mathbb{O}(\Sigma, D)$ liefert zwei Isomorphismen. Damit erhalten wir zwei weitere Darstellungen für endlich erzeugte frei partiell kommutative Monoide.

Lemma 1 *Das durch das Abhängigkeitsalphabet erzeugte frei partiell kommutative Monoid* $\mathbb{M}(\Sigma, D)$ *ist isomorph zu dem Monoid der Abhängigkeitsgraphen* $\mathbb{G}(\Sigma, D)$ *und zu dem Monoid der partiell geordneten Multimengen* $\mathbb{O}(\Sigma, D)$*.* ■

Zu einer Spur $t = [a_1 \dots a_n] \in \mathbb{M}(\Sigma, D)$ mit $a_i \in \Sigma$ für alle $i \in [n]$ wird der dazugehörige Abhängigkeitsgraph $H_{\mathbb{G}}(t)$ gebildet, indem zunächst eine n-elementige Knotenmenge $V = \{v_1, \dots, v_n\}$ (wir betrachten Isomorphieklassen von Graphen) so beschriftet wird, daß $\lambda(v_i) = a_i$ gilt. Die Kantenmenge wird durch $E = \{(v_i, v_j) \mid i < j,\ (\lambda(v_i), \lambda(v_j)) \in D\}$ festgelegt. Interpretiert man den reflexiven und transitiven Abschluß dieses Graphen als partiell geordnete Multimenge, so erhält man $h_{\mathbb{O}}(t) = \langle M, \leq \rangle$.

Lemma 2 *Die partielle Ordnung* $\leq$ *einer partiell geordneten Multimenge* $h_{\mathbb{O}}(t) = \langle M, \leq \rangle \in \mathbb{O}(\Sigma, D)$ *ist der Schnitt über die Ordnung der Wortrepräsentanten von* $t \in \mathbb{M}(\Sigma, D)$,

$\leq = \bigcap_{w \in t} O(w)$. *Dabei ist $O(w)$ die durch die Reihenfolge der Buchstaben in w induzierte lineare Ordnung. Das heißt $O(1) = \emptyset$, $O(a) = \{(a,a)\}$ und $O(uv) = O(u) \cup O(v) \cup (u \times v)$ mit $a_1 \dots a_n \times b_1 \dots b_m = \{a_1, \dots, a_n\} \times \{b_1, \dots, b_m\}$, $a, a_1, \dots, a_n, b_1, \dots, b_m \in \Sigma$.*

Für jede topologische Permutation π der Spur $\langle M, \leq \rangle$ ist $a_{\pi(1)} \dots a_{\pi(n)}$, $a_1, \dots, a_n \in \Sigma$ ein Wortrepräsentant von $t = [a_1 \dots a_n]$. (Eine topologische Permutation ist eine bijektive Abbildung $\pi : [n] \longrightarrow [n]$ mit $a_{\pi(i)} < a_{\pi(i+1)}$, für alle $i \in [n-1]$.) ■

Im weiteren werden wir die Monoide $\mathbb{M}(\Sigma, D)$, $\mathbb{G}(\Sigma, D)$ und $\mathbb{O}(\Sigma, D)$ identifizieren. Das heißt, wir werden Elemente eines frei partiell kommutativen Monoids bis auf Isomorphie identifizieren.

2.2 Das Levi-Lemma

Das Lemma von Levi für Wörter aus freien Monoiden Σ^* besitzt die folgende Verallgemeinerung.

Lemma 3 (Levi-Lemma) *Seien $x, y, z, t \in \mathbb{M}(\Sigma, D)$ Spuren. Dann sind die folgenden Behauptungen zwei äquivalent.*

(i) $xy = zt$ *(ii)* $\exists r, u, v, s \in \mathbb{M}(\Sigma, D)$ *mit*
$x = ru$, $y = vs$, $z = rv$, $t = us$ *und* uIv. ■

Interessant ist das mit durch Äquivalenz $(i) \Leftrightarrow (ii)$, zusammen mit der Existenz eines positiven Gewichtes, einen Homomorphismus $w : \mathbb{M} \rightarrow \mathbb{N}$ mit $w^{-1}(0) = 1$, ein Monoid $\mathbb{M}$ als frei partiell kommutativ charakterisiert wird.

2.3 Das Einbettungstheorem

Eine konkrete Darstellung von $\mathbb{M}(\Sigma, D)$, die wir zur Beschreibung von Algorithmen verwenden werden, ist eine Einbettung in ein direktes Produkt freier Monoide. Bei der Konstruktion folgen wir dem kategoriellen Ansatz von Diekert [Die90b] und Fischer [Fis86].

Jedem Abhängigkeitsalphabet (Σ, D) ordnen wir (bijektiv) den ungerichteten, schlingenfreien Graphen $G(\Sigma, D) = (\Sigma, D \setminus id_\Sigma)$ zu, $D \subseteq \binom{\Sigma}{2}$. Betrachten wir die Kategorie, in der die Objekte die endlichen ungerichteten schlingenfreien Graphen und die Morphismen $h : (V, E) \longrightarrow (V', E')$ Abbildungen $V \longrightarrow V'$ sind, die der Bedingung $xy \in E \Longrightarrow h(x)h(y) \in E'$ für alle $x, y \in V$ genügen. Dann erhält man einen kontravarianten Funktor $\mathbb{M}$ in die Kategorie der Monoide

$$\begin{array}{rcl} (\Sigma, D \setminus id_\Sigma) & \longmapsto & \mathbb{M}(\Sigma, D) \\ h : (\Sigma, D_\Sigma \setminus id_\Sigma) \rightarrow (\Delta, D_\Delta \setminus id_\Delta) & \longmapsto & \mathbb{M}(h) : \mathbb{M}(\Delta, D_\Delta) \rightarrow \mathbb{M}(\Sigma, D_\Sigma), \end{array}$$

worin $\mathbb{M}(h)$ die eineindeutige Fortsetzung der Abbildung $y \longmapsto \prod_{x \in h^{-1}(y)} x$ ist, $y \in \Delta$. (Hier bezeichnet $\prod$ die Verknüpfung in $\mathbb{M}(\Sigma, D)$.)

Wir bezeichnen die kanonischen *Projektionen*

$$\mathbb{M}(\Sigma, D) \longrightarrow \mathbb{M}(\Delta, D \cap (\Delta \times \Delta))$$

auf ein von $\Delta \subseteq \Sigma$ induziertes Teilalphabet mit π_Δ. Das ist der durch die Abbildung $a \mapsto a$ für $a \in \Delta$ und $a \mapsto 1$ anderenfalls induzierte Homomorphismus. Für einelementige Alphabete $\Delta = \{a\}$ schreiben wir auch π_a für $\pi_{\{a\}}$.

Eine *Überdeckung* eines Abhängigkeitsalphabets (Σ, D) ist eine Familie $\mathcal{A}$ von Abhängigkeitsalphabeten, mit

$$\Sigma = \bigcup_{(\Delta, D_\Delta) \in \mathcal{A}} \Delta \quad \text{und} \quad D = \bigcup_{(\Delta, D_\Delta) \in \mathcal{A}} D_\Delta.$$

Theorem 2 *Sei (Σ, D) ein Abhängigkeitsalphabet und $\mathcal{A}$ eine Überdeckung von (Σ, D). Dann ist der Homomorphismus*

$$\pi : \mathbb{M}(\Sigma, D) \longrightarrow \prod_{(\Delta, D_\Delta) \in \mathcal{A}} \mathbb{M}(\Delta, D_\Delta); \quad t \longmapsto \Big(\pi_\Delta(t)\Big)_{(\Delta, D_\Delta) \in \mathcal{A}}$$

injektiv. ■

Unter einer *Clique* C von einem Abhängigkeitsalphabet (Σ, D) verstehen wir in diesem Kontext ein Teilalphabet $C \subseteq \Sigma$ mit der Eigenschaft $C \times C \subseteq D$. Eine *Cliquenüberdeckung* von (Σ, D) ist eine Familie $\mathcal{C}$ von Cliquen so, daß $\Sigma = \bigcup_{C \in \mathcal{C}} C$ und $D = \bigcup_{C \in \mathcal{C}} (C \times C)$. Da $\mathbb{M}(C, C \times C) \cong C^*$ erhalten wir eine zur Formulierung von effizienten Algorithmen geeignete Darstellung für Spuren.

Korollar 1 *Sei (Σ, D) ein Abhängigkeitsalphabet und $\mathcal{C}$ eine Cliquenüberdeckung. Dann ist der Homomorphismus*

$$\pi : \mathbb{M}(\Sigma, D) \longrightarrow \prod_{C \in \mathcal{C}} C^*; \quad t \longmapsto \Big(\pi_C(t)\Big)_{C \in \mathcal{C}}$$

injektiv. ■

Obgleich wir die verschiedenen Repräsentationen identifizieren, werden wir bei der Darstellung von Algorithmen die letztere bevorzugen, da zur Repräsentation bereits wohlbekannte Datenstrukturen, die Strings, existieren. Für Korrektheitsbeweise sind jedoch meistens Abhängigkeitgraphen bzw. partielle Ordnungen geeigneter, da sie ein intuitiveres Verständnis einer Beweissituation durch ihre geometrische Interpretation vermitteln.

3 Ersetzungssysteme

In diesem Abschnitt führen wir Spurersetzungssysteme ein.

Definition 1 (Spurersetzungssystem) *Ein* Spurersetzungssystem *ist eine endliche binäre Relation* $\mathcal{R}$ *über einem frei partiell kommutativen Monoid* $\mathbb{M} = \mathbb{M}(\Sigma, D)$.

Ein Spurersetzungssystem $\mathcal{R} \subseteq \mathbb{M}\times\mathbb{M}$ definiert eine *Derivations-* oder *Ableitungsrelation* $\Longrightarrow_{\mathcal{R}} \subseteq \mathbb{M} \times \mathbb{M}$ via $x \Longrightarrow_{\mathcal{R}} y$, falls

$$x = ulv,\ y = urv, \text{ für } u, v \in \mathbb{M} \text{ und } l \to r \in \mathcal{R}.$$

Für $(r, l) \in \mathcal{R}$ notieren wir kurz $r \to l$. Die iterierte Anwendung ist durch

$$\begin{array}{ll} x \overset{0}{\Longrightarrow}_{\mathcal{R}} x & \text{für alle } x \in \mathbb{M} \\ x \overset{n+1}{\Longrightarrow}_{\mathcal{R}} y & \text{falls ein } z \in \mathbb{M} \text{ existiert, mit } x \overset{n}{\Longrightarrow}_{\mathcal{R}} z \Longrightarrow_{\mathcal{R}} y \end{array}$$

für alle $n \in \mathbb{N}$ definiert. Mit $\overset{*}{\Longrightarrow}_{\mathcal{R}}$ notieren wir die reflexive und transitive Hülle $\bigcup_{n\in\mathbb{N}} \overset{n}{\Longrightarrow}_{\mathcal{R}}$ von $\Longrightarrow_{\mathcal{R}}$, und mit $\overset{*}{\Longleftrightarrow}_{\mathcal{R}}$ die symmetrische, reflexive und transitive Hülle ${}_{\mathcal{R}}\overset{*}{\Longleftarrow} \cup \overset{*}{\Longrightarrow}_{\mathcal{R}}$. Die Relation $\overset{*}{\Longleftrightarrow}_{\mathcal{R}}$ ist eine Kongruenz über dem frei partiell kommutativem Monoid $\mathbb{M}$. Wir notieren $\mathbb{M}/\mathcal{R}$ für den Quotienten $\{[t]_{\overset{*}{\Longleftrightarrow}_{\mathcal{R}}} \mid t \in \mathbb{M}\}$, mit $[t]_{\overset{*}{\Longleftrightarrow}_{\mathcal{R}}} = \{t' \in \mathbb{M} \mid t \overset{*}{\Longleftrightarrow}_{\mathcal{R}} t'\}$ und nennen zwei Ersetzungssysteme $\mathcal{R}$ und $\mathcal{S}$ *äquivalent*, wenn $\mathbb{M}/\mathcal{R} \cong \mathbb{M}/\mathcal{S}$.

3.1 Eigenschaften von Ersetzungssystemen

Ein Spurersetzungssystem $\mathcal{R} \subseteq \mathbb{M} \times \mathbb{M}$ ist

(i) *längenreduzierend*, wenn für alle Regeln $l \to r \in \mathcal{R}$ $|l| > |r|$ gilt,

(ii) *gewichtsreduzierend*, wenn ein Homomorphismus $w : \mathbb{M} \to \mathbb{N}$ existiert, mit $w^{-1}(0) = \{1\}$ und $w(l) > w(r)$ für alle Regeln $l \to r \in \mathcal{R}$,

(iii) *noethersch*, wenn es keine unendlichen Derivationen $t_0 \Longrightarrow_{\mathcal{R}} t_1 \Longrightarrow_{\mathcal{R}} t_2 \Longrightarrow_{\mathcal{R}} \dots$ gibt und

(iv) *konfluent*, wenn für alle difluenten Situationen $t' \ {}_{\mathcal{R}}\overset{*}{\Longleftarrow} t \overset{*}{\Longrightarrow}_{\mathcal{R}} t''$ sich auf ein Element $\hat{t}$ reduzieren lassen, $t' \overset{*}{\Longrightarrow}_{\mathcal{R}} \hat{t} \ {}_{\mathcal{R}}\overset{*}{\Longleftarrow} t''$.

(v) Ein Spurersetzungssystem besitzt die *Church-Rosser-Eigenschaft*, wenn $\overset{*}{\Longleftrightarrow}_{\mathcal{R}} \subseteq \overset{*}{\Longrightarrow}_{\mathcal{R}} \circ {}_{\mathcal{R}}\overset{*}{\Longleftarrow}$.

(vi) Es ist *lokal konfluent*, wenn ${}_{\mathcal{R}}\Longleftarrow \circ \Longrightarrow_{\mathcal{R}} \subseteq \overset{*}{\Longrightarrow}_{\mathcal{R}} \circ {}_{\mathcal{R}}\overset{*}{\Longleftarrow}$.

Die Menge der *reduziblen* Spuren $\bigcup_{(l,r)\in\mathcal{R}} \mathbb{M}\cdot l\cdot\mathbb{M}$ bezeichnen wir mit $\mathrm{Red}(\mathcal{R})$, und die Menge $\{t \in \mathbb{M} \mid \neg\exists s : t \Longrightarrow_{\mathcal{R}} s\} = \mathbb{M} \setminus \mathrm{Red}(\mathcal{R})$ der *irreduziblen* Spuren bezeichnen wir mit $\mathrm{Irr}(\mathcal{R})$. Wir nennen das System *normalisiert*, wenn jede rechte Seite irreduzibel ist und jede linke Seite bzgl. aller anderen linken Seiten irreduzibel ist. Wir nennen das System *vollständig*, falls es konfluent und noethersch ist. Eine *Links-Derivation* ist eine Sequenz

$$\begin{aligned} t = u_1 l_1 v_1 \Longrightarrow_{\mathcal{R}} u_1 r_1 v_1 = u_2 l_2 v_2 \Longrightarrow_{\mathcal{R}} \ldots \\ \ldots \overset{u}{\Longrightarrow}_{\mathcal{R}\ n-1} r_{n-1} v_{n-1} = u_n l_n v_n \Longrightarrow_{\mathcal{R}} t' \end{aligned}$$

mit $u_i \in \mathrm{Irr}(\mathcal{R})$, $(l_i, r_i) \in \mathcal{R}$ für alle $i \in [n]$. Es ist bekannt, daß Konfluenz lokale Konfluenz impliziert. Es ist unentscheidbar, ob ein Spurersetzungssystem noethersch ist. Ist ein (abstraktes) Ersetzungssystem noethersch, dann ist Konfluenz sogar mit lokaler Konfluenz äquivalent. Zu jedem vollständigen Ersetzungssystem existiert ein äquivalentes normalisiertes vollständiges Ersetzungssystem.

Wir untersuchen die folgenden zentralen Fragestellungen, das Konfluenzproblem und das Normalformproblem.

Problem 1 (Konfluenzproblem) *Gegeben sei ein festes Spurersetzungssystem $\mathcal{R}$ über einem Spurmonoid $\mathbb{M}$. Ist $\mathcal{R}$ konfluent?*

Das konstruktive Problem: „Gegeben sei ein Spurersetzungssystem $\mathcal{R}$ über einem Spurmonoid $\mathbb{M}$. Konstruiere ein äquivalentes vollständiges Ersetzungssystem.“ wurde bereits von Diekert durch Verallgemeinerung des Algorithmus von Knuth und Bendix auf den partiell kommutativen Fall bearbeitet [Die87]. Dieser Algorithmus benutzt intensiv Normalformberechnungen, was uns zu der folgenden Fragestellung führt.

Problem 2 (Normalformproblem) *Welche Komplexität hat das berechnen irreduzibler Elemente? Sei $\mathcal{R}$ ein noethersches Spurersetzungssystem über einem Spurmonoid $\mathbb{M}$ und eine Spur $t \in \mathbb{M}$ gegeben. In welcher (nicht-uniformen) Zeitkomplexität ist es möglich eine irreduzible Spur $\hat{t} \in \mathrm{Irr}(\mathcal{R})$ mit $t \overset{*}{\Longrightarrow}_{\mathcal{R}} \hat{t}$ zu ermitteln.*

Um später Komplexitätsaussagen treffen zu können definieren wir $\ell_{\mathcal{R}}(t) = \max\{n \in \mathbb{N} \mid t \overset{n}{\Longrightarrow}_{\mathcal{R}} t'\}$ als die Anzahl der maximal möglichen Ableitungsschritte (von t aus). Mit $d_{\mathcal{R}}(n)$ bezeichnen wir den Wert $\max\{\ell_{\mathcal{R}}(t) \in \mathbb{N} \mid |t| = n, t \in \mathbb{M}\}$. Dann ist $d_{\mathcal{R}}$ total, falls $\mathcal{R}$ noethersch ist und $d_{\mathcal{R}} \in \Omega(n)$, falls $\mathcal{R} \neq \emptyset$. Für nicht-leere gewichtsreduzierende Systeme ist $d_{\mathcal{R}} \in \Theta(n)$.

Fortan sei jedes Spurersetzungssystem noethersch, da sonst Derivationen existieren, in denen kein irreduzibles Element in endlich vielen Schritten erreicht wird, und wir effektive Verfahren betrachten.

4 Konfluenz von Spurersetzungssystemen

Die Konfluenz von längenreduzierenden Wortersetzungssystemen modulo Kommutationsregeln ist im allgemeinen nicht entscheidbar, wie Narendran und Otto [NO88] feststellten.

Wir geben hier einen neuen einfachen und direkten Beweis für die Unentscheidbarkeit, der nicht unmittelbar auf der Unentscheidbarkeit der Präperfektheit basiert und illustriert, wie eine Berechnung einer Turingmaschine durch einen Konfluenztest simuliert werden kann. Genauer konstruieren wir zu einer Turingmaschine M und einem Wort w ein endliches längenverkürzendes Ersetzungssystem $\mathcal{R}$ über $\{a, b, \$, \beta\}^*/\$\beta = \beta\$$ mit der Eigenschaft, daß $w \notin L(M)$ genau dann gilt, wenn $\mathcal{R}$ konfluent ist. Insbesondere gelang es zu zeigen, daß das betrachtete Problem bereits für ein Alphabet mit vier Buchstaben und einer Kommutationsregel unentscheidbar ist.

4.1 Unentscheidbarkeit

Unter einer deterministischen Turingmaschine mit einem einseitig unendlichem Band verstehen wir ein Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, F, \not b)$, wobei $\Sigma \subseteq \Gamma$ das Eingabealphabet, Γ das Bandalphabet und Q die Zustandsmenge ist. δ ist eine partiell definierte Transitionsfunktion $Q \times \Gamma \longrightarrow Q \times (\Gamma \setminus \{\not b\}) \times \{\lhd, \rhd\}$, $q_0 \in Q$ ist der Startzustand und $F \subseteq Q$ sind die Finalzustände. $\not b \in \Gamma \setminus \Sigma$ bezeichnet das Blanksymbol. Dabei beschreibt $\delta(q, a) = (q', a', \rhd)$, daß die Maschine bei Lesen des Zeichen a in Zustand q das Zeichen a' schreibt, in Zustand q' wechselt und den Kopf ein Zeichen nach rechts bewegt. $\delta(q, a) = (q', a', \lhd)$ beschreibt eine analoge Aktion mit einer Linksbewegung des Kopfes.

Eine *Konfiguration* ist ein Tupel (u, q, v), wobei die Maschine in Zustand q ist, sich der Kopf auf dem ersten Symbol von v befindet und die Bandinschrift $uv \in (\Gamma \setminus \{\not b\})^*$ gefolgt von einer Blanksequenz ist. Mit $\longmapsto_M$ bezeichnen wir die *Einschritttransitionsrelation*

$$\begin{array}{ll} (u, q, av) \longmapsto_M (ua', q', v) & \text{falls } \delta(q, a) = (q', a', \rhd), \\ (ub, q, av) \longmapsto_M (u, q', ba'v) & \text{falls } \delta(q, a) = (q', a', \lhd), b \in \Gamma \end{array}$$

und mit $\overset{*}{\longmapsto}_M$ den reflexiven transitiven Abschluß. Mit $L(M)$ bezeichnen wir die akzeptierte Sprache $\{w \in \Sigma^* \mid (1, q_0, w) \overset{*}{\longmapsto}_M (u, q_f, v), q_f \in F\}$.

Ohne Einschränkung haben genau die akzeptierenden Konfigurationen (u, q_f, v), $q_f \in F$ keine Folgekonfiguration, d.h. wir normieren die Maschine so, daß sie auf nichtakzeptierten Eingaben nicht terminiert, indem wir einen neuen Zustand $f \notin Q$ einführen und die Transitionsfunktion für Zustände $q \notin F$ wie folgt erweitern:

$$\delta(q, a) \quad = \quad \begin{cases} \delta(q, a) & \text{falls schon definiert} \\ (f, a', \rhd) & \text{anderenfalls, } a' \neq \not b,\ a' \text{ beliebig} \end{cases}$$

Wir erhalten eine Maschine, die ein Wort $w \in L(M)$ durch Halten in endlich vielen Schritten akzeptiert und für $w \notin L(M)$ nicht terminiert. Ferner nehmen wir an, daß keine Blanksymbole geschrieben werden.

Zu einer solchen Turingmaschine M konstruieren wir ein simulierendes längenreduzierendes Ersetzungssystem $\mathcal{R}$ über dem Alphabet $Q \dot\cup \Gamma \dot\cup \{\$\}$. Für Rechtsbewegungen führen wir die Regeln

$$\begin{array}{rcll} bqa\$\$ & \to_{\mathcal{R}} & ba'q', & \delta(q,a) = (q', a', \rhd), \\ bq\not{b}\$\$ & \to_{\mathcal{R}} & ba'q'\not{b}, & \delta(q,\not{b}) = (q', a', \rhd) \end{array}$$

ein, und für Linksbewegungen fügen wir die Regeln

$$\begin{array}{rcll} bqa\$\$ & \to_{\mathcal{R}} & qba', & \delta(q,a) = (q', a', \lhd) \\ bq\not{b}\$\$ & \to_{\mathcal{R}} & qba'\not{b}, & \delta(q,\not{b}) = (q', a', \lhd) \end{array}$$

für alle $a, b \in \Gamma \setminus \{\not{b}\}$ hinzu.

Weiter enthält das Ersetzungssystem die Regeln

$$ab\$\$ \to_{\mathcal{R}} a\$b \quad \text{und} \quad a\$b\$\$ \to_{\mathcal{R}} a\$\$b$$

für $a, b \in \Gamma$. Sie erlauben einen Transport der \$-Symbole in einem Wort nach links. Bei der Simulation eines Berechnungsschrittes werden immer zwei \$-Symbole konsumiert und ein Transport eines \$-Symbols um ein Zeichen kostet ein \$.

Lemma 4 *(i) $\mathcal{R}$ ist konfluent über $(Q \cup \Gamma \cup \{\$\})^*$.*

(ii) Es gilt $(u,q,v) \overset{*}{\vdash}_M (u',q',v')$ genau dann, wenn eine Zahl $n \in \mathbb{N}$ existiert, mit $uqv\not{b}\$^n \overset{}{\Longrightarrow}_{\mathcal{R}} u'q'v'_1\$v'_2\$ \dots v'_k\not{b}\$$ und $v' = v'_1v'_2 \dots v'_k$, $v'_1, \dots, v'_k \in \Gamma \setminus \{\not{b}\}$.*

(iii) Es ist $w \in L(M)$ genau dann, wenn eine natürliche Zahl $n \in \mathbb{N}$ existiert mit

$$q_0w\not{b}\$^n \overset{*}{\Longrightarrow}_{\mathcal{R}} uq_fv\not{b}\$\$ \in \mathrm{Irr}(\mathcal{R}),$$

$q_f \in F$ und $v\not{b}\$\$ \in \mathrm{Irr}(\mathcal{R})$.

Beweis: Es ist wohlbekannt, wie man für noethersche Wortersetzungssysteme die Konfluenz mit Hilfe kritischer Paare testen kann, siehe [Jan88]. Im vorliegenden Fall ist die Menge der kritischen Paare offensichtlich leer. (Beachte: Die Maschine ist deterministisch.) Die zweite Aussage folgt unmittelbar aus der Konstruktion mit Induktion. Die dritte Behauptung ist evident, da in $uq_fv\not{b}\$\$$ wegen der Normierung der Turingmaschine kein \$ konsumiert werden kann. ∎

Seien $\mathbf{0}$, α, β und γ vier neue Symbole. Das neue Alphabet sei also $\Delta = \{\alpha, \beta, \gamma, \mathbf{0}, \$\} \cup \Gamma \cup Q$. Erweitern wir jetzt $\mathcal{R}$ noch um die Regeln $x\mathbf{0} \to_{\mathcal{R}} \mathbf{0}$ und $\mathbf{0}x \to_{\mathcal{R}} \mathbf{0}$ für alle Symbole $x \in \Delta$ so bleibt das System konfluent, da $w \overset{*}{\Longrightarrow}_{\mathcal{R}} \mathbf{0}$ für alle $w \in \Delta^*\mathbf{0}\Delta^*$. $\mathbf{0}$ ist ein absorbierendes Symbol.

Zu einem Wort $w \in \Sigma^*$ definieren wir das Regelsystem $\mathcal{R}(w)$ durch Erweiterung von $\mathcal{R}$ um die drei Regeln

$$\begin{array}{rcl} \alpha^{|w|+2}\beta & \to_{\mathcal{R}} & q_0w\not{b} \\ \beta\gamma & \to_{\mathcal{R}} & \mathbf{0} \\ \not{b}\gamma & \to_{\mathcal{R}} & \mathbf{0}. \end{array}$$

Bemerkung 1 *(i) $\mathcal{R}(w)$ ist konfluent, da sich alle kritischen Situationen auflösen lassen:*

$$\mathbf{0} \ {}_{\mathcal{R}}\!\overset{*}{\Longleftarrow} q_0 w \mathbf{0} \ {}_{\mathcal{R}}\!\Longleftarrow q_0 w \not{b} \gamma \ {}_{\mathcal{R}}\!\Longleftarrow \alpha^{|w|+2} \beta \gamma \Longrightarrow_{\mathcal{R}} \alpha^{|w|+2} \mathbf{0} \overset{*}{\Longrightarrow}_{\mathcal{R}} \mathbf{0}$$

(ii) $w \notin L(M)$ genau dann, wenn für alle $n \in \mathbb{N}$

$$q_0 w \not{b} \$^n \gamma \overset{*}{\Longrightarrow}_{\mathcal{R}} uqv \not{b} \gamma \Longrightarrow_{\mathcal{R}} uqv\mathbf{0} \overset{*}{\Longrightarrow}_{\mathcal{R}} \mathbf{0}$$

gilt.

Sei jetzt β und \$ unabhängig, d.h. $\beta\$ = \$\beta$. Betrachten wir die kritischen Situationen über dem erzeugten Spurmonoid $\mathbb{M}(\Delta, D_\Delta)$ mit $\Delta = Q \cup \Gamma \cup \{\alpha, \beta, \gamma, \$, \mathbf{0}\}$ und $D_\Delta = \Delta^2 \setminus \{(\$, \beta), (\beta, \$)\}$

$$\alpha^{|w|+2} \beta \$^n \gamma \quad = \quad \alpha^{|w|+2} \$^n \beta \gamma,$$

so läßt sich die rechte Seite zu $\mathbf{0}$ via

$$\alpha^{|w|+2} \$^n \beta \gamma \Longrightarrow_{\mathcal{R}} \alpha^{|w|+2} \$^n \mathbf{0} \overset{*}{\Longrightarrow}_{\mathcal{R}} \mathbf{0}$$

ableiten. Die linke Seite läßt sich via $\alpha^{|w|+2} \beta \$^n \gamma \Longrightarrow_{\mathcal{R}} q_0 w \not{b} \$^n \gamma$ ableiten. Das entstandene Element läßt sich nur dann zu $\mathbf{0}$ für alle $n \in \mathbb{N}$ ableiten, falls $w \notin L(M)$:

$$q_0 w \not{b} \$^n \gamma \overset{*}{\Longrightarrow}_{\mathcal{R}} uqv \not{b} \gamma \Longrightarrow_{\mathcal{R}} uqv\mathbf{0} \overset{*}{\Longrightarrow}_{\mathcal{R}} \mathbf{0}.$$

Andernfalls existiert eine natürliche Zahl $n \in \mathbb{N}$ und eine Derivation

$$\alpha^{|w|+2} \beta \$^n \gamma \Longrightarrow_{\mathcal{R}} q_0 w \not{b} \$^n \gamma \overset{*}{\Longrightarrow}_{\mathcal{R}} uq_f v \not{b} \$^2 \gamma \in \mathrm{Irr}(\mathcal{R})$$

zu einem irreduziblen Wort verschieden von $\mathbf{0}$. Wir erhalten:

Lemma 5 *Wenn $\mathcal{R} \subseteq \mathbb{M} \times \mathbb{M}$ konfluent ist, dann gilt $w \notin L(M)$.* ∎

Für die Umkehrung benötigen wir eine konkrete Darstellung von Faktorisierungen, um weiter Aussagen machen zu können. Diese entnehmen wir aus dem Anhang.

Ein *Faktorvorkommen*

$$i : l = \langle M_l, \leq_l \rangle \longrightarrow t = \langle M_t, \leq_t \rangle$$

von l in der Spur t ist eine ordnungserhaltende Inklusion $i : M_l \hookrightarrow M_t$ so, daß sich t durch $u \cdot i(l) \cdot v$ faktorisieren läßt[2]. Das folgende im Anhang bewiesene Lemma stellt eine Charakterisierung für Faktorisierungen bereit.

Lemma 6 *Seien $t, l \in \mathbb{O}(\Sigma, D)$ Spuren mit $t = \langle M_t, \leq_t \rangle$ und $l = \langle M_l, \leq_l \rangle$. Weiter sei $M_l \subseteq M_t$ eine Teilmultimenge und $\leq_l$ die auf M_l induzierte partielle Ordnung $\leq_t \cap (M_l \times M_l)$. Dann ist l ein Faktor in t ($t = ulv$ für $u, v \in \mathbb{M}(\Sigma, D)$) genau dann, wenn für alle Elemente $a, b \in M_l$ und $c \in M_t$ gilt $a \leq_t c \leq_t b \Longrightarrow c \in M_l$.*

[2] Viele dieser Begriffe werden im Anhang näher erläutert. Dort befinden sich auch visuelle Veranschaulichungen und technische Beweise.

Beweis: Sei l ein Faktor in $t = ulv \in \mathbb{O}(\Sigma, D)$ mit Spuren $u = \langle M_u, \leq_u \rangle$, $v = \langle M_v, \leq_v \rangle$ und $t = \langle M_t, \leq_t \rangle$. Daraus folgt $M_t = M_u \dot\cup M_v \dot\cup M_l$. Seien $a, b \in M_l$ und $c \in M_t$ mit $a \leq_t c \leq_t b$. Dann gilt $a \leq_t c \implies c \notin M_u$ und $c \leq_t b \implies c \notin M_v$. Also liegt c in M_l.

Sei nun $l = \langle M_l, \leq_l \rangle$ eine durch $\leq_t$ partiell geordnete Teilmenge von t so, daß für alle $a, b \in M_l$ $a \leq_l c \leq_l b \implies c \in M_l$ gilt. Liegen alle minimalen und maximalen Elemente von t in M_l, so ist $t = l$ ein Faktor ($u = v = 1$). Anderenfalls existiert ein minimaler (oder ein maximaler) Buchstabe a in t, der nicht in l vorkommt. Damit faktorisiert sich $t = at'$ (oder $t = t'a$) und t' enthält (mit Induktion über die Kardinalität von M_t) den Faktor l. Das heißt $t = au'lv'$ (oder $t = u'lv'a$). Man beachte, die Divisionsrelation ist noethersch. ∎

Seien l und t wie im Lemma und sei i wie oben beschrieben, dann führt die Charakterisierung zu einer Aufteilung der Spuren in

(i) den Teil $\langle i(l), \leq'_t \rangle$, der das Vorkommen enthält

(ii) den Teil, der vor dem Vorkommen liegt

$$\mathrm{pre}_t(i) = \langle \{a \in M_t \mid a <_t i(b), b \in M_l\}, \leq'_t \rangle,$$

(iii) den Teil, der nach dem Vorkommen liegt,

$$\mathrm{post}_t(i) = \langle \{a \in M_t \mid i(b) <_t a, b \in M_l\}, \leq'_t \rangle,$$

(iv) und den von dem Vorkommen unabhängigen verbleibenden Teil

$$\mathrm{ind}_t(i) = t \setminus (\mathrm{pre}_t(l) \cup \mathrm{post}_t(l) \cup l).$$

$\leq'_t$ notiert die von $\leq_t$ auf jeweiligen Teilmenge induzierte Ordnung.

Wir sagen zwei Vorkommen $i_j : l_j \longrightarrow t$, $j = 1, 2$, sind *separiert*, wenn i_j bereits in $\mathrm{pre}_t(i_{j'}) \cdot \mathrm{ind}(i_{j'})$ vorkommt, $\{j, j'\} = \{1, 2\}$. Die Spur t wird von zwei Vorkommen $i_j : l_j \longrightarrow t$, $j = 1, 2$, *induziert*, wenn alle Buchstaben zwischen Buchstaben aus $i_1(l_1)$ und $i_2(l_2)$ liegen:

$$t = \langle \{a \in M_t \mid i_j(b) \leq_t a \leq i_{j'}(c), b \in l_j, c \in l_{j'}, \{j, j'\} = \{1, 2\}\}, \leq'_t \rangle.$$

Um den Beweis des Lemmas 8 zu verkürzen verwenden wir einige bekannte Resultate aus [Die90b], die wir hier zitieren und an unsere Notation anpassen. Sei $\mathcal{R}$ ein noethersches Ersetzungssystem. Wir definieren die Reduktionsordnung $\preceq$ via $x \preceq y \Leftrightarrow y \overset{*}{\Longrightarrow}_{\mathcal{R}} uxv$ für $u, v \in \mathbb{M}$.

Lemma 7 *Sei $\mathcal{R} \subseteq \mathbb{M}^2$ ein noethersches Spurersetzungssystem, $t \in \mathbb{M}$ und $i : l \longrightarrow t$ ein Faktorvorkommen mit $l \to r \in \mathcal{R}$. Für alle $a \in \{b \in \Sigma \mid \{b\} \times \Sigma(l) \subseteq I\}$ gilt $ra \overset{*}{\Longrightarrow}_{\mathcal{R}} x \;{}_{\mathcal{R}}\overset{*}{\Longleftarrow} ar$. Dann lassen sich die Derivationen $u_1 r v_1 \;{}_{\mathcal{R}}\Longleftarrow t \Longrightarrow_{\mathcal{R}} u_2 r v_2$ aus den Faktorisierungen $u_1 \cdot i(l) \cdot v_1 = u_2 \cdot i(l) \cdot v_2 = t$ zusammenführen (d.h. es existiert $\hat{t}$ mit $u_1 r v_1 \overset{*}{\Longrightarrow}_{\mathcal{R}} \hat{t} \;{}_{\mathcal{R}}\overset{*}{\Longleftarrow} u_2 r v_2$), wenn $\mathcal{R}$ auf allen Spuren $t' \prec t$ konfluent ist.* ∎

Betrachten wir die in [Die90b] definierte Menge der kritischen difluenten Situationen $CT(l_1, l_2, \mathcal{R})$. Das sind Spuren $t \in \mathbb{M}$, bei denen zwei Faktorvorkommen von linken Seiten existieren, bei denen sich die Reduktionen gegenseitig auswirken. Formal enthält $CT(l_1, l_2, \mathcal{R})$ Spuren $t \in \mathbb{M}$,

(i) die von zwei Vorkommen $i_j : l_j \longrightarrow t$, $j = 1, 2$, induziert werden so, daß die Vorkommen nicht separiert sind, und

(ii) für alle linken Seiten $l \in L_{\mathcal{R}} = \{l \mid (l \to r \in \mathcal{R}\}$, die in t vorkommen $k : l \longrightarrow t$, gilt für $j = 1$ oder $j = 2$, daß t von den Vorkommen k und i_j induziert wird und diese Vorkommen nicht separiert sind.

Mit dieser Definition konnte in [Die90b] folgendes Theorem bewiesen werden.

Theorem 3 *Sei $\mathcal{R}$ ein noethersches Ersetzungssystem, $ra \overset{*}{\Longrightarrow}_{\mathcal{R}} x \; {}_{\mathcal{R}}\overset{*}{\Longleftarrow} ar$ für alle $a \in \{b \in \Sigma \mid \{b\} \times \Sigma(l) \subseteq I\}$ und $l \to r \in \mathcal{R}$. Dann ist $\mathcal{R}$ konfluent, wenn alle Paare aus*

$$\begin{aligned} \mathit{CritPair}(\mathcal{R}) \;=\; \{(t_1, t_2) \mid \;& t_1 \Longleftarrow_{l_1 \to r_1} t \Longrightarrow_{l_2 \to r_2} t_2,\ t \in CT(l_1, l_2, \mathcal{R}) \\ & l_j \to r_j,\ t_j = u_j r_j v_j,\ t = u_j l_j v_j;\ (j \in \{1, 2\})\} \end{aligned}$$

zusammenführbar sind. ∎

Damit können wir die fehlende Implikation beweisen.

Lemma 8 *Wenn $w \notin L(M)$, dann ist $\mathcal{R}$ konfluent.*

Beweis: Zu keiner Regel $l \to r \in \mathcal{R}$ existiert ein von l unabhängiger Buchstabe $a \in \Sigma$. Es reicht also zu zeigen, daß alle Paare $(t_1, t_2) \in \mathrm{CritPair}(\mathcal{R})$ zusammenführbar sind.

Keine linke Seite enthält eine andere als Faktor. Linke Seiten der Form $aqb\$\$$, $ab\$\$$, $a\$b\$\$$, $x\mathbf{0}$, $\mathbf{0}x$, $\bar{b}\gamma$ sind immer separiert. Sie führen also nicht zu kritischen Situationen.

Es verbleibt die Konfluenz auf Spuren $t \in CT(\alpha^{|w|+2}\beta, \beta\gamma, \mathcal{R})$ zu zeigen. Spuren aus $CT(l_1, l_2, \mathcal{R})$ lassen sich in neun Teile zerlegen:

$$\begin{tikzcd} p_1 \arrow[r] \arrow[d] \arrow[dr] & y_1 \arrow[r] & q_2 \\ s_1 \arrow[d] & s \arrow[ur] \arrow[dl] & s_2 \arrow[u] \\ q_1 & y_2 \arrow[l] & p_2 \arrow[l] \arrow[u] \arrow[ul] \end{tikzcd}$$

durch

$$\begin{aligned} p_i &= l_i \cap \mathrm{pre}(l_j) \\ s_i &= l_i \cap \mathrm{ind}(l_j) \quad = \quad \mathrm{ind}(l_j) \\ q_i &= l_i \cap \mathrm{post}(l_j) \qquad\qquad \{i, j\} = \{1, 2\} \\ s &= l_1 \cap l_2 \\ y_1 &= \mathrm{post}(l_1) \cap \mathrm{pre}(l_2) \\ y_2 &= \mathrm{post}(l_2) \cap \mathrm{pre}(l_1) \end{aligned}$$

Das Diagramm für $l_i = \alpha^{|w|+2}\beta$ und $l_j = \beta\gamma$ entartet zu

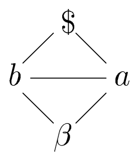
$$\begin{tikzcd}
 & \$^n \arrow[dr] & \\
\alpha^{|w|+2} \arrow[ur] \arrow[dr] & & \gamma \\
 & \beta \arrow[ur] &
\end{tikzcd}$$

da notwendigerweise $s = \beta$, $p_i = \alpha^{|w|+2}$, $p_j = 1$, $q_i = 1$, $q_j = \gamma$ und $y_i \in \$^*$, $y_j = 1$ gilt, $\{i, j\} = \{1, 2\}$. Doch

$$\mathbf{0} \;{}_{\mathcal{R}}\!\overset{*}{\Longleftarrow}\; q_0 w \bar{b}\$^n\gamma = t_1 \;{}_{\mathcal{R}}\!\Longleftarrow t \Longrightarrow_{\mathcal{R}} t_2 = \alpha^{|w|+2}\$^n\mathbf{0} \overset{*}{\Longrightarrow}_{\mathcal{R}} \mathbf{0}$$

ist eine konfluente Situation, wenn $w \notin L(M)$. Damit enthält $\mathrm{CritPair}(\mathcal{R})$ nur konfluente Paare. Also ist $\mathcal{R}$ konfluent. ∎

Theorem 4 *Es ist nicht entscheidbar, ob ein längenreduzierendes Spurersetzungssystem über einem Monoid $\mathbb{M}(\Sigma, D)$ konfluent ist, wenn das Abhängigkeitsalphabet (Σ, D) mindestens vier Buchstaben enthält und genau ein Paar dieser vier Buchstaben unabhängig ist, d.h. (Σ, D) enthält den induzierten Untergraphen*



Beweis: Da die universelle Sprache $\{\langle M, w\rangle \mid M \text{ akzeptiert } w\}$ nicht entscheidbar ist, ist es unentscheidbar, ob ein längenreduzierendes Ersetzungssystem über einem Monoid mit nur einer Kommutation konfluent ist. Mit der Codierung $c : \mathbb{N} \longrightarrow \{a, b\}^*$ mit $c(i) = aba^{i+2}b^{i+2}$ läßt sich das Alphabet (Δ, D_Δ) (s.o.) wie folgt darstellen:

$$\begin{array}{lcll}
a_i & \longmapsto & c(i) & a_i \in \Gamma = \{a_1, \ldots, a_{|\Gamma|}\} \\
q_i & \longmapsto & c(|\Gamma| + i) & q_i \in Q = \{q_1, \ldots, q_{|Q|}\} \\
\alpha & \longmapsto & c(|\Gamma| + |Q| + 1) & \\
\bar{b} & \longmapsto & c(|\Gamma| + |Q| + 2) & \\
\gamma & \longmapsto & c(|\Gamma| + |Q| + 3) & \\
\mathbf{0} & \longmapsto & c(|\Gamma| + |Q| + 4) & \\
\beta & \longmapsto & \beta & \\
\$ & \longmapsto & \$ &
\end{array}$$

Der durch diese Abbildung induzierte Homomorphismus $h : \mathbb{M}(\Delta, D_\Delta) \longrightarrow \mathbb{M}(\Sigma, D)$ hat die Eigenschaft:

> $h(z)$ besitzt genau dann ein Bild einer linken Seite $h(l)$ als Faktor, wenn z die linke Seite l als Faktor besitzt.

Formal sei

$$h(\mathcal{R}) \quad = \quad \{h(l) \rightarrow_{h(\mathcal{R})} h(r) \mid l \rightarrow_{\mathcal{R}} r \in \mathcal{R}\}.$$

Es gilt $h(t) \overset{*}{\Longrightarrow}_{h(\mathcal{R})} h(t')$ genau dann, wenn $t \overset{*}{\Longrightarrow}_{\mathcal{R}} t'$. ∎

Im Kapitel 7 werden wir Dekompositionen von Monoiden näher untersuchen. Zur Formulierung des nächsten Korollars greifen wir einige Definitionen vorweg.

Seien (Σ, D_Σ) und (Δ, D_Δ) zwei disjunkte Abhängigkeitsalphabete ($\Sigma \cap \Delta = \emptyset$). Dann definieren wir die *direkte Summe* via

$$(\Sigma, D_\Sigma) \oplus (\Delta, D_\Delta) \;:=\; (\Sigma \cup \Delta, D_\Sigma \cup D_\Delta),$$

und das *Komplexprodukt* mit

$$(\Sigma, D_\Sigma) * (\Delta, D_\Delta) \;:=\; \big(\Sigma \cup \Delta, D_\Sigma \cup D_\Delta \cup (\Sigma \times \Delta) \cup (\Delta \times \Sigma)\big).$$

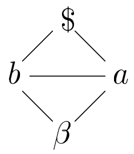
Die kleinste Subklasse der Abhängigkeitsalphabete, die unter direkter Summe und Komplexprodukt abgeschlossen ist, und die einelementigen Abhängigkeitsalphabete enthält, bezeichnet man als *Cographen-Abhängigkeitsalphabete*.

Die Spurmonoide, die von diesen Abhängigkeitsalphabeten erzeugt werden, nennen wir *Cographen-Monoide*.

Korollar 2 *Es ist nicht entscheidbar, ob ein längenreduzierendes Spurersetzungssystem über einem Cographen-Monoid konfluent ist.*

Beweis: Das Monoid $\mathbb{M}(\Sigma, D) \cong \{a, b\}^* * \mathbb{N}^2$ ist ein Cographen-Monoid.

$$(\Sigma, D) \quad =$$



$$= \quad (a * b) * (\$ \oplus \beta)$$

∎

Als offene Frage verbleiben noch die Fälle, bei denen die Abhängigkeitsalphabete diesen induzierten Untergraph nicht besitzen. Beispielsweise ist noch offen, ob die Konfluenz auf dem einfachen Abhängigkeitsalphabet $(\Sigma, D) = a - b - c$ entscheidbar ist.

5 Algorithmen für Ersetzungssysteme mit zusammenhängenden linken Seiten

Im folgenden sei $\mathbb{M} = \mathbb{M}(\Sigma, D)$ das frei partiell kommutative von (Σ, D) erzeugte Monoid. Weiter bezeichne $\mathcal{C}$ eine Cliquenüberdeckung des erzeugenden Alphabets. Das Ersetzungssystem bezeichnen wir mit $\mathcal{R}$ und die Menge der linken Seiten $\{l \in \mathbb{M}(\Sigma, D) \mid (l, r) \in \mathcal{R}\}$ mit $L_{\mathcal{R}}$. Für eine Clique $C \in \mathcal{C}$ und eine Spur $t \in \mathbb{M}$ verwenden wir für die Projektion $\pi_C(t) \in C^*$ auf das Cliquenalphabet $C \in \mathcal{C}$ die Bezeichnung t_C.

Ein zentrales Problem bei der Faktorersetzung bzw. bei der Faktorerkennung ist die nicht lokale Charakterisierung eines Faktors wie in Lemma 6. Da wir i.allg. nicht den vollständigen Abhängigkeitsgraphen oder gar seinen transitiven Abschluß in weniger als quadratischer Zeit abspeichern können, ist eine Idee, auf Strukturen mit geringerer Größe ohne Informationsdefekt auszuweichen. Eine dieser Strukturen ist das Hasse-Diagramm des Abhängigkeitsgraphen, eine andere, die wir hier verwenden, ist die Worttupeldarstellung aus Korollar 1. Diese Darstellungen erlauben leider nicht „globale" Bedingungen, wie beispielsweise die Faktorbedingung in Lemma 6.

$$\forall a, b \in M_l \, \forall c \in M_t : a \leq_t c \leq_t b \implies c \in M_l$$

in konstanter Zeit zu entscheiden.

5.1 Das Berechnungsmodell

Bevor wir Algorithmen und Komplexitätsresultate formulieren, ein Wort über das Maschinenmodell. Wir gehen bei den folgenden Komplexitätsaussagen von einem Rechnermodell mit indiziertem Speicherzugriff aus (RAM) und abstrahieren bei der Darstellung der Algorithmen auf eine prozedurale imperative Programmiersprache. Sie stellt Selektionen (if-Statements), Iterationen (while- und for-Schleifen) und Rekursion als Kontrollstrukturen und Zuweisungen als Modifikatoren bereit. Wir nehmen an, daß jede Abfrage und jede primitive Operation (wie Addition, Subtraktion, Zeigeroperationen, numerische Vergleiche etc.) auf Speicherzellen bzw. Variablen $\mathcal{O}(1)$ Zeiteinheiten benötigt.

5.2 Die C^*-Bäume

Wir stellen im folgenden eine Spur $t \in \mathbb{M}$ durch ihr Projektionstupel $(t_C)_{C \in \mathcal{C}}$ dar. Zur effizienten Manipulation führen wir weitere Zugriffsstrukturen ein. Jedes Wort t_C, $C \in \mathcal{C}$ dieses Tupels wird mit einer baumartigen Verweisstruktur versehen, die mit weiteren Informationen beschriftet ist. Buchstaben, die das gleiche Urbild bezüglich des Projektionshomomorphismus $\pi : \mathbb{M} \longrightarrow \prod_{C \in \mathcal{C}} C^*$ haben, sollen sich (gegenseitig) referenzieren. D.h. in $t = (t_C)_{C \in \mathcal{C}}$ besitzen das i-te a Symbol in $t_C = u_C a v_C$, $a \in C \in \mathcal{C}$, $|u_C|_a = i - 1$, Referenzen aufeinander.

In diesem Abschnitt beschreiben wir die baumartigen Zugriffsstrukturen. Ein *C^*-Baum* ist ein über einem Wort $w \in C^*$ errichteter Baum. Sei $m \geq 3$ eine natürliche Zahl. Dann *repräsentiert* ein C^*-Baum T das Wort $w = w_1 \dots w_n$ mit $w_i \in C$, $i \in [n]$, wenn die Blätter die Buchstaben $w_i \in C$ sind und der geordnete Baum den folgenden Struktureigenschaften genügt.

(i) Jeder innere Knoten besitzt mindestens 2 und höchstens m Kinder.

(ii) Jedes Blatt besitzt dieselbe Distanz zum Wurzelknoten.

(iii) Die Ordnung der Blätter entspricht der Ordnung der Buchstaben in w.

Die lineare Ordnung $O(w)$ $(w_1 < w_2 < \ldots < w_n)$ der Buchstaben $w_i \in \Sigma$, $i \in [n]$, in $w = w_1 \dots w_n$ legt in natürlicher Weise die Ordnung auf den Blättern fest und läßt sich auf zu einer partiellen Ordnung auf den inneren Knoten wie folgt fortsetzen. Ein innerer Knoten ist kleiner als ein anderer, wenn alle nachfolgenden Knoten des ersten Knoten kleiner als die des zweiten sind.

Weiter besitzt ein C^*-Baum mit Knotenmenge V zwei Beschriftungen

$$\lambda_1 : V \longrightarrow \mathcal{P}(\mathcal{C}) \quad \text{und} \quad \lambda_2 : V \longrightarrow \mathcal{P}(L_{\mathcal{R}}).$$

Sie sollen eine effiziente Berechnung von Faktorisierungen, Faktorvorkommen und Konkatenationen ermöglichen. Dazu genügen die Beschriftungen den folgenden Bedingungen:

(i) Jeder Knoten v in T ist via λ_1 mit dem Alphabet des durch den Unterbaum (mit Wurzel v und Blattsequenz w) induzierten Alphabets $\lambda_1(v) = \Sigma(w) \subseteq C$ beschriftet.

(ii) Jeder Knoten v in T ist via λ_2 mit einer Teilmenge der linken Seiten $l \in L_{\mathcal{R}}$ beschriftet, deren Projektion $\pi_C(l) \in C^* \setminus \{1\}$ in dem durch v induzierten Teilwort beginnt. Genauer besitzt der durch v induzierte Teilbaum ein zum Buchstaben a korrespondierendes Blatt, so daß das durch den gesamten Baum repräsentierte Wort w sich wie folgt faktorisieren läßt $w = u \cdot a l'_C \cdot v$ mit $\pi_C(l) = l_C = a l'_C$.

Eine Spur $t \in \mathbb{M}$ wird durch eine Baumfamilie $(T_C)_{C \in \mathcal{C}}$ repräsentiert, wenn für alle $C \in \mathcal{C}$ der C^*-Baum T_C in der Familie das Wort t_C repräsentiert. Die λ_2-Beschriftungen sollen dann genau die linken Seiten $l \in L_{\mathcal{R}}$ enthalten, bei denen sich die Faktorisierungen auf der repräsentierten Projektion zu einer Faktorisierung der repräsentierten Spur fortsetzen läßt. D.h. für die repräsentierte Projektion $w = t_C \in C^*$ existiert ein Faktorvorkommen $i : l \longrightarrow t$ mit $w = w' i(a) l'_C w''$ genau dann, wenn jeder Vorfahre des Blattes, das zum Buchstaben $i(a)$ korrespondiert, mit Mengen, die l enthalten, beschriftet ist.

Die Idee ist, daß die Baumstruktur mit der λ_1-Beschriftung eine effiziente Rekonstruktion der transitiven Abhängigkeiten erlaubt und, daß die Baumstruktur mit der λ_2-Beschriftung eine effiziente Berechnung von Faktorisierungen ulv mit $l \in L_{\mathcal{R}}$ ermöglicht.

Formal sei T_C ein Baum aus der Familie. Sei w' das von dem Unterbaum mit Wurzel v induzierte Teilwort von w, dann definieren wir

$$\begin{aligned}
\lambda_1(v) &= \Sigma(w') \\
\lambda_2(v) &= \{l \in L_{\mathcal{R}} \mid \exists i : l \longrightarrow t \text{ Faktorvorkommen mit } i(\pi_C(l)) = al'_C \\
&\qquad [(l'_C = l''_C l'''_C \wedge w' = xal''_C \wedge w = x'w'l'''_C y') \vee w' = xal'_C y]\}
\end{aligned}$$

Wir zitieren an dieser Stelle ein Korollar, in dem Faktorisierungen charakterisiert werden (siehe Anhang A).

Korollar 3 *Seien $t, l \in \mathbb{M}(\Sigma, D)$. Dann ist $l = \langle M_l, \leq_l \rangle$ ein Faktor in $t = \langle M_t, \leq_t \rangle$ genau dann, wenn*

(i) l ein Vorkommen in t ist, d.h. für alle Cliquen $C, C' \in \mathcal{C}$ mit $C \cap \Sigma(l) \neq \emptyset$ und $C' \cap \Sigma(l) \neq \emptyset$ existieren Wörter $u_C, v_C \in C^*$ mit $t_C = u_C l_C v_C$ und $\pi'_C(u_C) = \pi_C(u_{C'})$, sowie

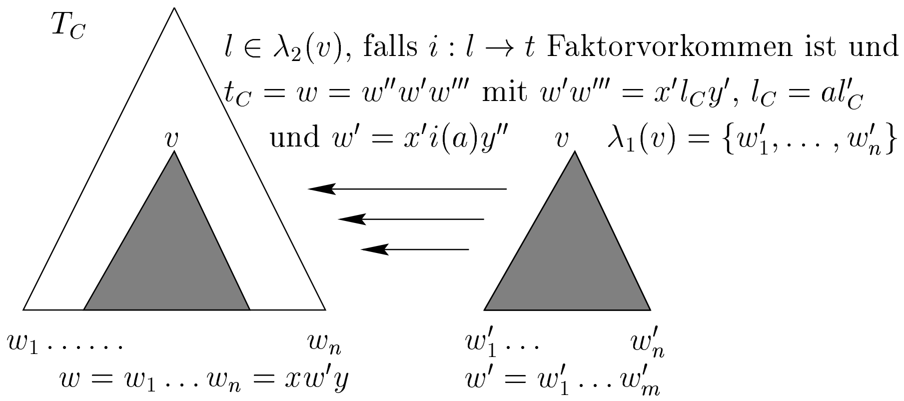
(ii) für alle verbleibenden Cliquen $C \in \mathcal{C}$, $\Sigma(l) \cap C = \emptyset$ existieren Faktorisierungen $u_C v_C = t_C$ so, daß die Beziehung $\pi_C(u_{C'}) = \pi_{C'}(u_C)$ für alle $C, C' \in \mathcal{C}$ erfüllt ist. ■

Sei $t \in \mathbb{M}$, $l \in L_{\mathcal{R}}$ und $\mathcal{C} = \{C_1, \ldots, C_k\}$ mit $\Sigma(l) \cap C_j \neq \emptyset$ für $1 \leq j \leq i$ und $\Sigma(l) \cap C_j = \emptyset$ für $i+1 \leq j \leq k$. Dann gilt $t = x \cdot l \cdot y$ genau dann, wenn

$$\begin{pmatrix} t_{C_1} \\ \vdots \\ t_{C_i} \\ t_{C_{i+1}} \\ \vdots \\ t_{C_k} \end{pmatrix} = \begin{pmatrix} x_{C_1} \\ \vdots \\ x_{C_i} \\ x_{C_{i+1}} \\ \vdots \\ x_{C_k} \end{pmatrix} \cdot \begin{pmatrix} l_{C_1} \\ \vdots \\ l_{C_i} \\ 1 \\ \vdots \\ 1 \end{pmatrix} \cdot \begin{pmatrix} y_{C_1} \\ \vdots \\ y_{C_i} \\ y_{C_{i+1}} \\ \vdots \\ y_{C_k} \end{pmatrix} = \begin{pmatrix} x_{C_1} \cdot a_{C_1} l'_{C_1} \cdot y_{C_1} \\ \vdots \\ x_{C_i} \cdot a_{C_i} l'_{C_i} \cdot y_{C_i} \\ x_{C_{i+1}} \cdot y_{C_{i+1}} \\ \vdots \\ x_{C_k} \cdot y_{C_k} \end{pmatrix}$$

und jeder Vorfahre v in der repräsentierenden Datenstruktur eines zu einem der Buchstaben $a_{C_1}, \ldots, a_{C_i}$ korrespondierenden Blattes mit einer Menge $\lambda_2(v)$, die l enthält, beschriftet ist, und für $1 \leq j, j' \leq i$ die Bedingung $|x_{C_j}|_a = |x_{C_{j'}}|_a$ für alle $a \in C \cap C' \cap \Sigma(l)$ erfüllt ist.

Die folgende Abbildung illustriert die Baumstruktur.



Die *Höhe* HEIGHT(v) = HEIGHT(T) eines Baumes T mit Wurzel v ist rekursiv definiert. Jedes Blatt v besitzt die Höhe HEIGHT$(v) = 1$ und die Höhe eines inneren Knoten v berechnet sich rekursiv HEIGHT$(v) = 1 + \max_{c \in \text{children}(v)}$ HEIGHT(c). (Entsprechend sind Tiefe und Distanz definiert.)

Aus den strukturellen Eigenschaften des Baumes folgt eine Schranke für die Pfadlängen zur Wurzel.

Lemma 9 *Sei T ein C^*-Baum der Höhe h, dann liegt die Anzahl der Knoten von T zwischen $2^{h+1} - 1$ und $\frac{m^{h+1}-1}{m-1}$. Die Länge des repräsentierten Wortes liegt zwischen 2^h und m^h.*

Damit ist die Distanz von einem Buchstaben zur Wurzel proportional zu $\log n$.

Beweis: Sei $n = n(h)$ die Anzahl der Knoten. Die Anzahl der Knoten mit Wurzeldistanz i liegt zwischen 2^i und m^i. Es gilt

$$2^{h+1} - 1 = \sum_{i=0}^{h} 2^i \leq n(h) \leq \sum_{i=0}^{h} m^i = \frac{m^{h+1} - 1}{m - 1}$$

■

Sei $\mathcal{C}$, wie oben beschrieben, eine Cliquenüberdeckung. Dann *repräsentiert* das C^*-Baumtupel $(T_C)_{C \in \mathcal{C}}$ die Spur t dann, wenn für alle $C \in \mathcal{C}$ der Baum T_C das Wort $\pi_C(t)$ repräsentiert (mit korrekter Beschriftung). Blätter, die zu Buchstaben korrespondieren, die das gleiche Urbild bezüglich des Projektionshomomorphismus $\pi : \mathbb{M} \longrightarrow \prod_{C \in \mathcal{C}} C^*$ haben, sollen sich referenzieren.

Diese strukturellen Eigenschaften ermöglichen es beispielsweise in logarithmischer Zeit zu entscheiden, ob ein durch Positionen in den Wörtern t_C, $C \in \mathcal{C}$ bzw. via Blättern in T_C beschriebenes Vorkommen ein Faktor ist. Für den Beweis beschreiben wir die Methoden, um einige (primitive) Prädikate entscheiden zu können.

Der erste Algorithmus berechnet zu einem gegebenen mit dem Buchstaben a beschrifteten Blatt das, falls existent, größte sich links von v befindliche mit b beschriftete Blatt. D.h. für einen das Wort $w = w'aw''$ repräsentierenden C^*-Baum T und einen den Buchstaben a repräsentierenden Knoten v soll der Knoten, der das letzte b in w' repräsentiert, berechnet werden. (Man beachte, daß ein lineares sondieren auch lineare Zeit- und nicht logarithmische Zeitkomplexität besitzt.) Ein geeignetes Auf- und wieder Absteigen in der Baumstruktur ist jedoch in der angestrebten Zeitschranke möglich.

Algorithm pred(T, v, b)
  Comment Input: a nonempty C^*-tree T and a leaf v
      representing a letter a.
    Output: the right-most b labeled leaf to the left of v.
  Method
    if v is the root of T **then return** no predecessor
      else $v' \leftarrow v$
        $v \leftarrow$ the father of v'
        while $b \notin \lambda_1(v'')$ for a brother to the left of v
          and v is not the root **do**
          $v' \leftarrow v$
          $v \leftarrow$ father of v'
        end_while
        if v' has no b-labeled brother to the left
          then return no b-labeled predecessor
          else while v is no leaf **do**
              let v be the right-most b-labeled brother to the left
            return (v)
        end_if
    end_if
  end_Method

Lemma 10 *Sei T ein C^*-Baum, v ein mit a beschriftetes Blatt und $b \in C$ ein Buchstabe. Dann ist pred(T, v, b) das mit b beschriftete Blatt (falls existent), das sich links von v befindet, so daß kein weiteres mit b beschriftetes Blatt sich zwischen v und pred(T, v, b) befindet.*

Beweis: Repräsentiere der C^*-Baum T das Wort $w \in C^*$ und sei v der Blattknoten, der den Buchstaben a in $w = w'aw''$ repräsentiert. Ist v die Wurzel, so gilt $w = a$ und das Resultat existiert nicht.

In der ersten Schleife wird der Pfad $(v, v_1, v_2, \ldots, v_k)$ von v in Richtung Wurzel so lange erforscht, bis entweder die Wurzel erreicht ist oder die Beschriftung des letzten Knoten $\lambda_1(v_k)$ den Buchstaben b enthält. Es gilt, daß die Variable v den Wert v_i beim i-ten Durchlauf hat, wenn v_i der i-te Vorfahr von dem Eingabeknoten v ist. Weiter ist v' der $(i-1)$-te Vorfahr.

Existiert kein Vorfahr v_k so, daß einer der linken Brüder von v_{k-1} mit einer Menge, die b enthält, beschriftet ist, dann gilt $b \notin \Sigma(w')$ und der Algorithmus berechnet das richtige Resultat. Existiert ein solcher Bruder, so gelten die folgenden Invarianten für die zweite Schleife: $b \in \lambda_1(v)$ und, wenn v'_{k-i} der Wert von v in der i-ten Iteration ist, dann gilt $\mathrm{HEIGHT}(v_i) = \mathrm{HEIGHT}(v'_i)$, v'_i ist Vater von v'_{i-1} und es existiert kein Knoten x mit der gleichen Höhe, der sich links von v_i' und rechts von v_i befindet und mit einer Menge die b enthält beschriftet ist. Ansonsten wäre x ein Nachkomme eines Bruders von v_{k-1}, der sich rechts von v_{k-1}' befindet, was im Widerspruch zur Wahl von $v_{k-1}', \ldots, v_{k-i+1}'$ steht. Man beachte $b \notin \lambda_1(v_i)$ für alle $i < k$. Damit ist nach k Iterationen das Resultat der der Semantik entsprechende b repräsentierende Knoten mit $w' = u'bu''$ und $b \notin \Sigma(u'')$. ∎

Mit dieser Technik sind wir in der Lage, in einer Spur $t = ubvaw \in \mathbb{M}$ das b-Vorkommen $(a, b) \in D$ zu finden mit $b = \max\{b \in M_t \mid b <_t a\}$, $t = \langle M_t, \leq_t \rangle$. Induktiv über die Anzahl der Buchstaben in einer Zusammenhangskomponente von a in (Σ, D) können wir auf diese Weise die (vollständige) transitive Relation $\leq_t$ entscheiden.

Wir bezeichnen im folgenden den reflexiven und transitiven Abschluß der Relation D mit D^* und bezeichnen die Menge $\{b \in \Sigma \mid (a, b) \in X\}$ mit $X(a)$, bzw. $X(Y) = \bigcup_{y \in Y} X(y)$, $X \in \{D, D^*\}$.

Algorithm PRED$((T_C)_{C \in \mathcal{C}}, (v_C)_{a \in C \in \mathcal{C}}, b)$
  Comment Input: a representation $(T_C)_{C \in \mathcal{C}}$ of $t \in \mathbb{M}$, a representation $(v_C)_{a \in C \in \mathcal{C}}$ of a, and a letter $b \in \Sigma$.
  Output: the representation of $b = \max\{b \mid b <_t a\}$.
  Method
    if $b \notin D^*(a)$ **then return** no such predecessor possible
      else for all $c \in D^*(a) \setminus \{a\}$ and $C \in \mathcal{C}$ with $c \in C$ **do**
          initialize $x_{c,C}$ as undefined.
        end_for
        $X \leftarrow \{a\}$
        for all $C \in \mathcal{C}$ with $a \in C$ **do**
          initialize $x_{a,C}$ by v_C.
        end_for
        while $X \neq D^*(a)$ **do**
          $Y \leftarrow D(X) \setminus X$
          for all $c \in Y$ and $d \in X$ **do**
            for all $C \in \mathcal{C}$ with $c \in C$ **do**
              $x_{c,C} \leftarrow \max(\text{pred}(T_C, x_{C,d}, c), x_{C,c})$
            end_for
          end_for
          $X \leftarrow X \cup Y$
        end_while
        return $((x_{C,b})_{b \in C \in \mathcal{C}})$
    end_if
  end_Method

Lemma 11 *Sei $(T_C)_{C \in \mathcal{C}}$ eine die Spur $t = \langle M_t, \leq_t \rangle$ repräsentierende Baumfamilie, $(v_C)_{a \in C \in \mathcal{C}}$ eine den Buchstaben a in t repräsentierende Blattfamilie und $b \in \Sigma$. Dann ist das Resultat von*

$$PRED((T_C)_{C \in \mathcal{C}}, (v_C)_{a \in C \in \mathcal{C}}, b)$$

eine den Buchstaben $b = \max\{b \mid b <_t a\}$ *repräsentierende Blattfamilie* $(v'_C)_{b\in C\in\mathcal{C}}$.

Beweis: Wenn $(a,b) \notin D^*$, existiert keine Ordnungsrelation $a \leq_t b$ in t, da für eine Partition $\Sigma = \Sigma_a \dot\cup \Sigma_b$ die Isomorphie $\mathbb{M}(\Sigma_a, D\cap\Sigma_a^2) \times \mathbb{M}(\Sigma_b, D\cap\Sigma_b^2) \cong \mathbb{M}(\Sigma, D)$ gilt, wenn $D = (D\cap\Sigma_a^2)\dot\cup(D\cap\Sigma_b^2)$. Anderenfalls hat die while-Schleife die Invariante: Für alle $d \in X$ repräsentiert $x_d = (x_{C,d})_{d\in C\in\mathcal{C}}$ das Vorkommen $d = \max\{d \mid d \leq_t a\}$ oder, falls dieses Maximum nicht existiert, ist $(x_{C,d})_{d\in C\in\mathcal{C}}$ undefiniert. Diese Aussage bleibt invariant, da pred die oben definierte Semantik besitzt. Durch die Initialisierung ist die Invariante beim Betreten der while-Schleife erfüllt. Die Maximalität von x_d wird durch die Nachbedingung von pred bzw. den Maximumoperator zugesichert, der mit der folgenden Methode berechnet werden kann.

Algorithm $\max(u,v)$
  Comment Input: two leaves in a C^* tree T
            Output: the right-most leaf of $\{u,v\}$
  Method
    if $u = v$ **then return** (u)
      else $\langle u', v'\rangle \leftarrow \langle u, v\rangle$
          let u'' be the ancestor of u' and v'' be the ancestor of v'
          while $u'' \neq v''$ **do**
              $\langle u', v'\rangle \leftarrow \langle u'', v''\rangle$
              let u'' be the ancestor of u' and v'' be the ancestor of v'
          end_while
          if u' to the right of v' **then return** (u)
            else return (v)
          end_if
    end_if
  end_Method

■

Die Zeitkomplexität dieses Verfahrens ist logarithmisch.

Lemma 12 *Sei* T *ein das Wort* $w \in C^*$ *repräsentierender* C^**-Baum. Dann terminiert die Prozedur pred in* $\mathcal{O}(\log|w|)$ *Zeiteinheiten.*

Repräsentiert $(T_C)_{C\in\mathcal{C}}$ *die Spur* $t \in \mathbb{M}$, *so terminiert die Prozedur PRED in* $\mathcal{O}(|\Sigma|\cdot|D|\cdot\log|w|) = \mathcal{O}(\log|w|)$ *Zeiteinheiten.*

Beweis: Die Komplexität ist in beiden Fällen proportional zur Länge der traversierten Pfade, die maximal mit logarithmischer Ordnung wächst. ■

Analog läßt sich durch iteriertes Berechnen das Maximum von jeder Buchstabenmenge berechnen. Dual können auf diese Weise auch minimale Buchstabenvorkommen, die größer sind, berechnet werden.

Damit läßt sich in logarithmischer Zeit entscheiden, ob ein Vorkommen i von $l = (l_C)_{C\in\mathcal{C}}$ in t eine Faktorisierung $t = (t_C)_{C\in\mathcal{C}}$ induziert. Erinnern wir uns: Wir repräsentieren ein Vorkommen durch Teilwörter in den Projektionen wie in Korollar 3 Teil (i). Also entsprechen Vorkommen Teilmengen von Blättern, die sich gegenseitig referenzieren. Damit können wir Vorkommen mit zu den minimalen Buchstaben in $i(l)$ korrespondierenden Blattfamilien identifizieren. Man beachte, daß nicht alle diese Familien Vorkommen repräsentieren, aber jedes Vorkommen sich durch eine Familie repräsentieren läßt.

Lemma 13 *Sei $(T_C)_{C\in\mathcal{C}}$ eine Repräsentation der Spur $t \in \mathbb{M}$ und $(v_C)_{C\in\mathcal{C}}$ eine Blattfamilie, die ein Vorkommen i von $l = (l_C)_{C\in\mathcal{C}}$ in t so repräsentiert, daß die Blätter, die zu den minimalen Buchstaben in $i(l_C)$ korrespondieren, bekannt sind. D.h. für alle $a \in \Sigma$ mit $l = al'$ repräsentieren $(v'_{a,C})_{a\in C\in\mathcal{C}}$ die Buchstaben $\pi(i(a))$. (Damit referenzieren die Knoten, die den gleichen Buchstaben repräsentieren, sich gegenseitig.) Dann kann man in $\mathcal{O}(\log|t|)$ Zeiteinheiten entscheiden, ob das Vorkommen ein Faktor ist.*

Beweis: Das Prädikat $\mathrm{pre}_t(l) \cap \mathrm{post}_t(l) = \emptyset$ in Korollar 14, eine unmittelbare Folgerung aus der oben angeführten Charakterisierung, läßt sich mit logarithmischem Aufwand entscheiden, siehe Lemma 12.

Effizienter (d.h. mit kleineren Konstanten) läßt sich die Faktoreigenschaft überprüfen, wenn man explizit nach beeinträchtigenden[3] (zerstörenden) Buchstaben sucht, siehe Abschnitt A.3 oder Lemma 40 und doppelte Berechnungen durch PRED vermeidet. Ohne Einschränkung reicht es aus, nach beeinträchtigenden Buchstaben c zwischen minimalen und maximalen Buchstaben aus $i(l)$ zu suchen bzw. deren Nichtexistenz zu testen. ■

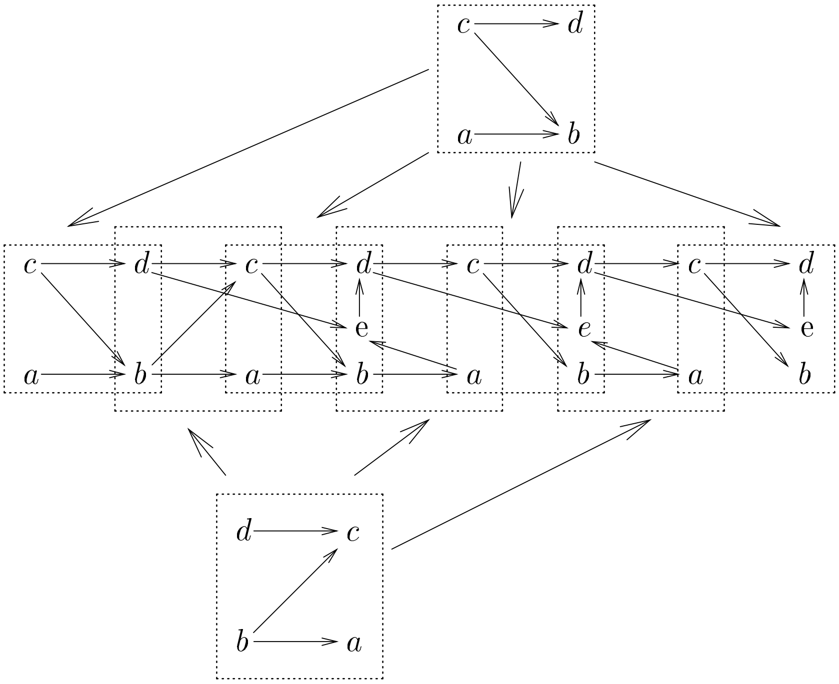
Um die bis jetzt eingeführten Begriffe und Strukturen zu verdeutlichen, betrachten wir das folgende Beispiel.

Beispiel 1 *Sei (Σ, D) das Abhängigkeitsalphabet*

$$\left(\{a,b,c,d,e\}, \left[\begin{tikzcd}[row sep=small] & & e \arrow[dll, dash] \arrow[dr, dash] & \\ a \arrow[r, dash] & b \arrow[r, dash] & c \arrow[r, dash] & d \end{tikzcd}\right]\right)$$

und die Spur $t = [cabdacbaedcbaedcbed]$. Sei $L_{\mathcal{R}} = \{l_1, l_2, l_3\}$ mit $l_1 = [acbd]$, $l_2 = [bdca]$ und $l_3 = [aed]$. Wir erhalten das Redukt des Abhängigkeitsgraphen von t mit den Vorkommen $i_1, i_2, i_3, i_4 : l_1 \longrightarrow t$ und $j_1, j_2, j_3 : l_2 \longrightarrow t$ (durch Polygonzüge in der Zeichnung hervorgehoben).

[3] Dieser Begriff wird im Anhang präzisiert.



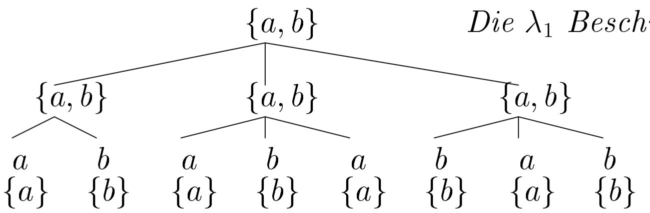
Unterscheiden wir für den Moment verschiedene Vorkommen des gleichen Buchstaben durch Indizes. Beispielsweise bezeichnet a_3 den dritten Buchstaben a in t, also das a in $i_2(l_1)$. Die Vorkommen von l_3 bezeichnen wir mit k_1 und k_2. Dann beeinträchtigen die Buchstaben e folgende Vorkommen:

$$\begin{array}{lcl} e_1 & \rhd & i_2, i_3 \\ e_2 & \rhd & i_3, i_4 \\ e_3 & \rhd & i_4 \end{array} \qquad \begin{array}{lcl} e_1 & \rhd & j_2 \\ e_2 & \rhd & j_3 \end{array} ,$$

Kein Vorkommen wird durch ein a, b, c oder d beeinträchtigt, und die Vorkommen i_1, j_1, k_1 und k_2 sind Faktorvorkommen. Die Spur wird durch das Tupel

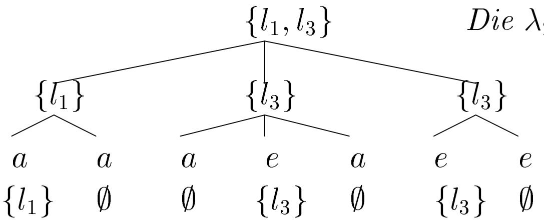
$$\pi(t) \quad = \quad \begin{pmatrix} abababab \\ cbcbcbcb \\ cdcdcdcd \\ dededed \\ aaaeaee \end{pmatrix}$$

dargestellt. Hier beispielsweise $T_{\{a,b\}}$ (nur mit λ_1 Beschriftungen)



Die λ_1 Beschriftungen von $T_{\{a,b\}}$

und $T_{\{a,e\}}$ (nur mit λ_2 Beschriftungen)



Die λ_2 Beschriftungen von $T_{\{a,b\}}$

5.3 Der Datentyp

Wir beweisen konstruktiv durch Angabe von Algorithmen, daß es effektiv und effizient möglich ist, diese Bäume aufzubauen, Teile zu löschen, sie zu zerlegen und wieder zusammenzufügen. Dazu führen wir die Operationen CONCATENATE, SPLIT, DELETE und INSERT ein, die vorerst nur die Beschriftung λ_1 und die strukturellen Eigenschaften erhalten. Hiernach stellen wir eine Methode bereit, um die verbleibende Beschriftung zu korrigieren. Dieses Vorgehen ermöglicht den Entwurf des Reduktionsalgorithmus.

INSERT

Um einen Buchstaben $a \in C$ in einen das Wort w repräsentierenden C^*-Baum an der Stelle, die durch das Präfix w' von w beschrieben sei, einzufügen, verwenden wir die folgende Methode. Sei b das zum letzten Buchstaben in w' korrespondierende Blatt. Falls kein solches Blatt existiert ($w' = \lambda$), so ist das neue Blatt der erste (kleinste) Buchstabe. Dann fügen wir neben b das neue Blatt an und restaurieren die strukturellen Eigenschaften entlang des Pfades zur Wurzel. Besitzt ein Knoten zu viele Kinder, so wird er in zwei Knoten geteilt und rekursiv weiter verfahren.

Das Einfügen des neuen Blattes geschieht damit bottom-up. Das ermöglicht die effiziente Wiederherstellung der strukturellen Eigenschaften.

Algorithm INSERT (T, a, b)
  Comment Input: a nonempty C^*-tree T and a new letter a that
          should be inserted at the right of b.
        Output: a revisted C^*-tree with the new leaf a.
  Method
    if T consists of a single leaf c **then**
      create a new root r
      if there exists b with $b \leq a$, i.e., a will not be minimal
        then make a the left child of r and c the right child
        else make c the left child of r and a the right child
      end_if
      label r by $\lambda_1(r) = \bigcup_{c \in \text{children}(r)} \lambda_1(c)$
    else if there exists b with $b \leq a$, i.e., a will not be minimal
        then let f be the father of the leaf b
            make a be a child of f immedeately to the right of b
        else let c be the left-most leaf in T

let f be the father of the leaf c
make a be a child of f immedeately to the left of c
end_if
relabel f by $\lambda_1(f) = \bigcup_{c \in \text{children}(f)} \lambda_1(c)$
if f has less than m children
then propagate the labeling by PROPAGATE1(f, T)
else f has m children
incorporate the vertex f with its $m+1$ children with
INCORPORATE(f, T)
end_if
end_if
end_Method

Für ein a repräsentierendes Blatt v gilt (ohne das es in den Beschreibungen ausgeführt ist) $\lambda_1(v) = a$. Die Prozedur INCORPORATE stellt die strukturellen Eigenschaften rekursiv wie folgt her.

Procedure INCORPORATE (v, T)
Comment Input: a nonempty ordered and labeled tree T containing
one vertex v with $m+1$ children.
Output: a corresponding C^*-tree.
Method
create a new vertex v'
make the two right-most children of v the left and the right child of v'
relabel v' by $\lambda_1(v') \leftarrow \bigcup_{c \in \text{children}(v')} \lambda_1(c)$
relabel v by $\lambda_1(v) \leftarrow \bigcup_{c \in \text{children}(v)} \lambda_1(c)$
if v has no father in T **then**
create a new root r
make v' the left child and v the right child of r
label r by $\lambda_1(r) \leftarrow \bigcup_{c \in \text{children}(r)} \lambda_1(c)$
else let f be the father of v
make v' a child of f immedeately to the right of v
relabel f by $\lambda_1(f) \leftarrow \bigcup_{c \in \text{children}(f)} \lambda_1(c)$
if f now has m children **then** INCORPORATE(T, f)
else propagate the labeling by PROPAGATE1(T, f)
end_if
end_if
end_Method

Lemma 14 *Sei T ein geordneter Baum und v ein innerer Knoten in T der Tiefe t. Weiter sollen alle Knoten, die sich nicht auf dem Pfad von v zur Wurzel von T befinden, C^*-Bäume induzieren und alle inneren Knoten außer v zwischen 2 und m Kinder besitzen. Der Knoten v besitzt genau $m+1$ Kinder. Dann liefert die Prozedur INCORPORATE(v, T) einen C^*-Baum, der das selbe Wort wie T repräsentiert, mit korrekter λ_1-Beschriftung in $\mathcal{O}(t)$ Zeiteinheiten.*

Beweis: Die Prozedur spaltet den Knoten v mit $m+1$ Nachfolgern in zwei Knoten mit mindestens 2 und höchstens $m-1$ Knoten auf. Dabei wird ein neues Kind des Vaterknoten f von v generiert. Hatte dieser Knoten bereits m Kinder, so restauriert diese Prozedur rekursiv die Struktureigenschaft und die Beschriftung. Hat dieser Knoten weniger als m Nachfolger, so besitzt der Baum bereits die Struktureigenschaft und die Prozedur PROPAGATE1 aktualisiert die λ_1-Beschriftung des Baumes. Die Ordnung auf den Blättern bleibt erhalten. ∎

Die Prozedur PROPAGATE1 vererbt die korrekte Beschriftungsinformation auf dem Pfad zur Wurzel.

Procedure PROPAGATE1 (T, v)
  Comment Input: a wrong labeled C^*-tree T with a vertex v.
              Output: a revisted C^*-tree.
  Method
    if v is the root **then** terminate
      else let f be the father of v
          relabel f by $\lambda_1(f) = \bigcup_{c \in \text{children}(f)} \lambda_1(c)$
          PROPAGATE1(T, f)
    end_if
  end_Method

Lemma 15 *Sei T ein geordneter Baum und v ein innerer Knoten in T mit Tiefe t. Weiter sollen alle Knoten v', die sich nicht auf dem Pfad von v zur Wurzel von T befinden, mit dem Blattalphabet $\lambda_1(v')$ des durch v' induzierten Baumes beschriftet sein. Dann liefert die Prozedur PROPAGATE1(T, v) einen C^*-Baum mit gleicher Ordnung und korrekter λ_1-Beschriftung in $\mathcal{O}(t)$ Zeiteinheiten.*

Beweis: Wir zeigen die folgende Invariante für die Datenstruktur: Für alle inneren Knoten v gilt

$$\lambda_1(v) \quad = \bigcup_{c \in \text{children}(v)} \lambda_1(c).$$

Sei children$(v) = \{v_1, \ldots, v_k\}$ und sei $w_i \in C^*$ das von Knoten v_i induzierte Teilwort von $w = x_i w_i y_i$, $i \in [k]$, und w' das von v induzierte Teilwort $w_1 \ldots w_k$. Dann gilt $\Sigma(w_i) = \lambda_1(v_i)$, $w' = w_1 \ldots w_k$ und damit $\Sigma(w') = \bigcup_{i \in [k]} \Sigma(w_i) = \bigcup_{i \in [k]} \lambda_1(v_i)$. ∎

Korollar 4 *Sei T ein das Wort w repräsentierender C^*-Baum, w' ein Präfix von $w = w'w''$ und b der letzte Buchstabe von w' oder $w' = 1$. Dann berechnet der Algorithmus INSERT(T, a, b) einen C^*-Baum, der das Wort $w'aw''$ repräsentiert in $\mathcal{O}(\log |w|)$ Zeiteinheiten.* ∎

Man beachte, daß die λ_2-Beschriftungen bis jetzt noch nicht betrachtet wurden.

DELETE

Analog ist der Algorithmus DELETE zum Entfernen des Blattes a formuliert.

Algorithm DELETE (T, a)
  Comment Input: a nonempty C^*-tree T with a leaf letter a that
  should be removed.
  Output: a revisted C^*-tree without the leaf a.
  Method
    if T consists of a single leaf c **then**
      delete this leaf and return the empty tree
      else let f be the father of a
        delete the leaf a
        relabel f by $\lambda_1(f) = \bigcup_{c \in \text{children}(f)} \lambda_1(c)$
        if f has less than 2 children **then** REMOVE(f, T)
          else propagate the labeling by PROPAGATE1(f, T)
        end_if
    end_if
  end_Method

Die Prozedur REMOVE stellt die strukturellen Eigenschaften invers zu INCORPORATE wieder wie folgt her.

Procedure REMOVE (v, T)
  Comment Input: a nonempty ordered and labeled tree T containing
  one vertex v with only 1 child.
  Output: a corresponding C^*-tree.
  Method
    if v is the root of T **then**
      delete v and return the empty tree
      else let f be the father of v
        if v has a immediate brother v' with more than 2 children **then**
          if v' is the left brother of v **then**
            let the right-most child of v' be now the left-most child of v
            else let the left-most child of v' be the right-most child of v
          end_if
          relabel v by $\lambda_1(v) \leftarrow \bigcup_{c \in \text{children}(v)} \lambda_1(c)$
          relabel f by $\lambda_1(f) \leftarrow \bigcup_{c \in \text{children}(f)} \lambda_1(c)$
          relabel v' by $\lambda_1(v') \leftarrow \bigcup_{c \in \text{children}(v')} \lambda_1(c)$
          propagate the labeling by PROPAGATE1(v, T) and
          PROPAGATE1(v', T)
          else let v' be a brother of v with 2 children
            if v' is the left brother of v
              then let the child of v be now the left-most child of v'
              else let the child of v be the right-most child of v'

end_if
delete the vertex v
relabel v' by $\lambda_1(v') = \bigcup_{c \in \text{children}(v')} \lambda_1(c)$
relabel f by $\lambda_1(f) = \bigcup_{c \in \text{children}(f)} \lambda_1(c)$
if f now has only one child **then** REMOVE(T, f)
else propagate the labeling by PROPAGATE1(T, f)
end_if
end_if
end_if
end_Method

Lemma 16 *Sei T ein geordneter Baum und v ein innerer Knoten in T der Tiefe t. Weiter sollen alle Knoten, die sich nicht auf dem Pfad von v zur Wurzel von T befinden, C^*-Bäume induzieren und alle inneren Knoten außer v zwischen 2 und m Kinder besitzen. Der Knoten v besitzt genau einen Kind. Dann liefert die Prozedur REMOVE(v, T) einen C^*-Baum, der dasselbe Wort repräsentiert, mit korrekter λ_1-Beschriftung in $\mathcal{O}(t)$ Zeiteinheiten.*

Beweis: Analog dem Beweis zu Lemma 14. ■

Korollar 5 *Sei T ein das Wort w repräsentierender C^*-Baum und $w'a$ ein Präfix von $w = w'aw''$. Dann berechnet der Algorithmus DELETE(T, a) einen C^*-Baum, der das Wort $w'w''$ repräsentiert in $\mathcal{O}(\log|w|)$ Zeiteinheiten.* ■

CONCAT

Betrachten wir als nächste Operation die Primitive CONCAT. Seien T_1 und T_2 zwei C^*-Bäume, die die Wörter w_1 und w_2 repräsentieren. Die Primitive CONCAT soll einen C^*-Baum T berechnen, der das Wort $w = w_1 w_2$ repräsentiert.

Sei h_1, die Höhe von T_1 und h_2 die von T_2. Die Prozedur IMPLANT baut T_1 in T_2 oder T_2 in T_1 ein, abhängig von der Höhendifferenz der Bäume. Sei $h = \min\{h_1, h_2\}$ die Höhe von dem Knoten v im Randpfad des höheren Baumes. Ein Randpfad ist der Pfad beginnend beim minimalen Blatt (bezüglich der Blattordnung) von T_2 zur Wurzel von T_2 beziehungsweise der Pfad, der beim maximalen Blatt von T_1 beginnt und an der Wurzel von T_1 endet. Ist der Knoten der Höhe h auf dem (längeren) Pfad gefunden, wird der Wurzelknoten des kleineren Baums als neues Kind eingefügt. Die Alphabetbeschriftung und die Struktureigenschaften werden wie beim Einfügen eines Knotens restauriert. Sind beide Bäume gleich hoch, so wird einfach eine neue Wurzel mit richtiger Beschriftung und den alten Wurzeln als Kindern kreiert. Den trivialen Fall, wenn einer der beiden Bäume leer ist, betrachten wir nicht.

Algorithm IMPLANT(T_1, T_2)
Comment Input: two nonempty C^*-trees T_1 and T_2.

Output: a revisted C^*-tree representing the concatenation
of the yields of T_1 and T_2.

Method

if HEIGHT(T_1) = HEIGHT(T_2) **then**
create a new root r
make the root of T_1 the right children of r and
make the root of T_2 the left children of r
relabel r by $\lambda_1(r) \leftarrow \bigcup_{c\in\text{children}(r)} \lambda_1(c)$
else assume HEIGHT(T_1) > HEIGHT(T_2)
otherwise interchange T_1 and T_2 and
interchange „left“ with „right“ **in**
let v be the right-most path of T_1 such that
DEPTH(v) = HEIGHT(T_1) − HEIGHT(T_2)
let f be the father of v
make the root of T_2 to the right-most child of f
relabel f by $\lambda_1(f) = \bigcup_{c\in\text{children}(f)} \lambda_1(c)$
if f has more than m children **then** INCORPORATE(f, T)
else propagate the labeling by PROPAGATE1(f, T)
end_if
end_if
end_Method

Lemma 17 *Sei T_1 ein das Wort w_1 repräsentierender C^*-Baum der Höhe h_1 und T_2 ein das Wort w_2 repräsentierender C^*-Baum der Höhe h_2. Dann kombiniert diese Prozedur die Bäume zu einem C^*-Baum, der das Wort w_1w_2 repräsentiert, in $\mathcal{O}(|h_1 - h_2|)$ Zeiteinheiten unter der Annahme, daß v nicht bestimmt werden muß.*

Beweis: Die Korrektheit ergibt sich aus der Semantik der Prozeduren INCORPORATE und PROPAGATE1. Die Zeitschranke ist offensichtlich, da je Knoten auf dem Pfad, der bei v beginnt und an der Wurzel des höheren Baumes endet, nur $\mathcal{O}(1)$ Elementaroperationen getätigt werden. Die Anzahl dieser Knoten ist $|h_1 - h_2|$. ∎

Damit läßt sich die Primitive CONCAT, die zwei C^*-Bäume konkatenieren soll, realisieren.

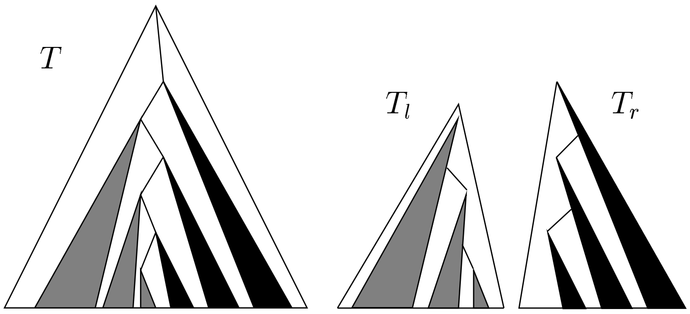
Korollar 6 *Sei T_1 ein das Wort w_1 repräsentierender C^*-Baum und T_2 ein das Wort w_2 repräsentierender C^*-Baum. Dann existiert ein Algorithmus IMPLANT(T_1, T_2), der die Bäume zu einem C^*-Baum T, der das Wort w_1w_2 repräsentiert, in der Zeit $\mathcal{O}(\log|w_1| + \log|w_2|)$ kombiniert.*

Beweis: Die Suche auf den Randpfaden nach dem Knoten v (wie in Lemma 17 beschrieben) ist in $\mathcal{O}(\log|w_1| + \log|w_2|)$ Zeitschritten möglich. Damit ergibt sich die Behauptung mit Lemma 17. ∎

SPLIT

Betrachten wir als letzte Primitive die Operation SPLIT. Für einen C^*-Baum T, der das Wort $w = w'aw''$ repräsentiert, und dem zum Buchstaben a korrespondierenden Blatt liefert SPLIT eine Partition von T in zwei C^*-Bäume T_1 und T_2 so, daß T_1 das Wort w' und T_2 das Wort aw'' repräsentiert.

Um einen C^*-Baum T effizient zu spalten, betrachtet man den bei a beginnenden und bei der Wurzel endenden Pfad in T. Ein Schnitt entlang diesem Pfad zerlegt den Baum T in eine Menge von Teilbäumen, deren Wurzelknoten Kinder von Knoten sind, die sich auf dem Pfad befinden. Die Bäume, die sich links vom Schnitt befinden, werden mit dem vorher vorgestellten Algorithmus konkateniert. Analog werden der Knoten a und die Bäume, die sich rechts befinden ebenfalls konkateniert.



Der folgende Algorithmus enthält die Details.

Algorithm DIVIDE (T, a)
  Comment Input: a nonempty C^*-tree T and leaf letter a.
        Output: two C^*-trees devided by the leaf a.
  Method
    let v be the a-labeled leaf.
    if v is the root of T **return**$(\langle 1, T\rangle)$
      else $L \leftarrow []$ $R \leftarrow [v]$
        while v is not the root **do**
          let f be the father of v and
          let $v_1, \ldots, v_{i-1}, v, v_{i+1}, \ldots, v_k$ be the children of f
            in order from left to right
          $L \leftarrow L[v_1, \ldots, v_{i-1}]$ $R \leftarrow [v_{i+1}, \ldots, v_k]R$
          purge v if v is not the leaf
          $v \leftarrow f$
        end_while
    Comment At this point T has been devided into two forrests –
        the left forrest, which consists of all trees with leaves
        to the left of the leaf a, (in list L) and the right
        forest (in list R), which consists of all trees with leaves
        to the right of a including leaf a.

Nodes on the path from a to the root of T except
leaf a are removed.
while there is more than one tree in the left forrest **do**
let T' and T'' be the two right-most trees in the left forest
insert the tree IMPLANT(T', T'') at the right end
end_while
while there is more than one tree in the right forrest **do**
let T' and T'' be the two left-most trees in the right forest
insert the tree IMPLANT(T', T'') at the left end
end_while
return the two trees
end_if
end_Method

Lemma 18 *Sei T ein das Wort $w = w'aw''$ repräsentierender C^*-Baum. Dann zerteilt der Algorithmus SPLIT(T, a) = DIVIDE(T, a) den C^*-Baum T, in zwei C^*-Bäume T_1 und T_2, von denen der erste das Wort w' und der zweite das Wort aw'' repräsentiert. Der Algorithmus benötigt $\mathcal{O}(\log |w|)$ Zeiteinheiten.*

Beweis: In der Aussage ist das den Buchstaben a repräsentierende Blatt mit dem Buchstaben selbst identifiziert. Die Listen L und R sind in der Länge durch $\mathcal{O}(\log |w|)$ beschränkt, da je Knoten auf dem Pfad von v zur Wurzel höchstens m neue Elemente angefügt werden. (Man beachte, daß in einer Liste (einem Pfad) aufeinanderfolgende Knoten mit konstantem Abstand in $\mathcal{O}(|h_1 - h_2|) = \mathcal{O}(1)$ Zeiteinheiten gefunden werden kann, insbesondere wenn man noch zusätzlich Höheninformation mitführt.) Dann folgt die Aussage durch Aufaddieren der Zeiten mit Lemma 17. ∎

5.4 Operationen auf der Darstellung

Ziel dieses Abschnittes ist aus den vielen Primitiven die Operationen SPLITTRACE und CONCATTRACE für die Spurdarstellung bereitzustellen. Zur Erinnerung: Wir schränken uns auf Spurersetzungssysteme mit zusammenhängenden linken Seiten ein. Die Intention besteht darin, Spurrepräsentationen effizient faktorisieren bzw. konkatenieren zu können. Dazu benötigen wir einen neuen Begriff.

Ein *Schnitt* in einer Spur $t \in \mathbb{M}$ ist ein Tupel $s = (w_C)_{C \in \mathcal{C}}$ mit der Eigenschaft, daß für alle $C \in \mathcal{C}$ für die Projektionen $t_C = w_C w'_C$ gilt und $\pi^{-1}(s) \in \mathbb{M}$. D.h. für alle $C, C' \in \mathcal{C}$ mit $a \in C \cap C'$ gilt $|w_C|_a = |w_{C'}|_a$. Wir stellen einen Schnitt durch ein Blättertupel $(v_C)_{C \in \mathcal{C}}$, wobei v_C ein Blatt in T_C ist, dar. Dieses Tupel repräsentiert die minimalen Buchstaben aus $\pi^{-1}((w'_C)_{C \in \mathcal{C}})$. Damit entspricht ein Schnitt einer Faktorisierung $t = \pi^{-1}((w_C)_{C \in \mathcal{C}}) \cdot \pi^{-1}((w'_C)_{C \in \mathcal{C}})$. π ist der Homomorphismus im Projektionslemma.

5.4.1 CONCATTRACE

Sei $T^i = (T^i_C)_{C \in \mathcal{C}}$ eine Darstellung der Spur t^i, $i \in \{1, 2\}$. Dann soll die Prozedur CONCATTRACE auf den Argumenten T^1 und T^2 eine Darstellung $T = (T_C)_{C \in \mathcal{C}}$ der Spur $t = t^1 \cdot t^2$ berechnen.

Diese Berechnung erfolgt vorerst Komponentenweise auf den Projektionen. Danach werden die λ_2-Beschriftungen rekonstruiert.

Algorithm CONCATTRACE (T^1, T^2)
  Comment Input: two trace representations.
    Output: the representation of the catenation.
  Method
    $\Lambda \leftarrow \emptyset$
    for each $C \in \mathcal{C}$ **do**
      $T_C \leftarrow$ CONCAT(T^1_C, T^2_C)
    end_for
    Comment At this point the λ_2-labeling is violated.
      All other properties are OK.
    for each $C \in \mathcal{C}$ **do**
      for all $l \in L_{\mathcal{R}}$ with $\Sigma(l) \cap C \neq \emptyset$ **do**
        let r be the maximal scann range $|\pi_C(l)| - 1$
        let $v_1, \ldots, v_r$ be the right-most leaves in T^1_C
        $\Lambda \leftarrow \Lambda \cup \{v_1, \ldots, v_r\}$
        for $i := 1$ **to** r **do**
          while there is a factor l with minimal letter v_i
              but $l \notin \lambda_2(v_i)$ **do**
            $\lambda_2(v_i) \leftarrow \lambda_2(v_i) \cup \{l\}$
            let $P = \{p_C \mid$ factor l has minimal letter p_C
                (leaf) at C-projection $\}$
            for all $p \in P$ **do** $\lambda_2(p) \leftarrow \lambda_2(p) \cup \{l\}$ **end_for**
            $\Lambda \leftarrow \Lambda \cup P$
          end_while
        end_for
      end_for
    end_for
    while $\Lambda \neq \emptyset$ **do**
      let $v \in \Lambda$ $\Lambda \leftarrow \Lambda \setminus \{v\}$ PROPAGATE2(v)
    end_while
    return(T)
  end_Method

Die Prozedur PROPAGATE2 vererbt analog zu PROPAGATE1 die Beschriftung Λ_2 eines Knoten v an seine Vorfahren.

Procedure PROPAGATE2 (v)

Comment Input: a wrong labeled C^*-tree T with a vertex v.
Output: a revisted C^*-tree.

Method
  if v is the root **then** terminate
    else let f be the father of v
      relabel f by $\lambda_2(f) = \bigcup_{c \in \text{children}(f)} \lambda_2(c)$
      PROPAGATE2(f)
  end_if
end_Method

Um die Korrektheit dieses Verfahrens zu zeigen, benötigen wir eine strukturelle Aussage über Faktorisierungen, bei denen zusammenhängende Spuren involviert sind, die Schnittlemmata. Sie sichern die Korrektheit der λ_2-Beschriftungen bis auf einen Teil konstanter Größe nach dem Konkatenieren bzw. Faktorisieren zu.

Lemma 19 (Schnittlemma A) *Sei $t = t_1 \cdot t_2 \in \mathbb{M}(\Sigma, D)$, $l \in \mathbb{M}(\Sigma, D)$ eine zusammenhängende Spur und $i : l \longrightarrow t$ ein Faktorvorkommen. Dann folgt aus $i(l) \cap (\max(t_1) \cup \min(t_2)) = \emptyset$ entweder $i(l) \cap t_1 = \emptyset$ oder $i(l) \cap t_2 = \emptyset$.*

Beweis: Sei $t = \langle M_t, \leq_t \rangle$. Angenommen $i(l) \cap t_1 \neq \emptyset$ und $i(l) \cap t_2 \neq \emptyset$. Da $t = t_1 \cdot t_2$ gilt für alle Buchstaben a_1 in t_1 und b_2 in t_2 mit $(a_1, b_2) \in D$ die Relation $a_1 \leq_t b_2$, und da l zusammenhängend ist, existiert ein maximaler Buchstabe a in $\max(t_1)$ und ein minimaler Buchstabe b in $\min(t_2)$ mit $(a, b) \in D$ und zwei Buchstaben c in $i(l) \cap t_1$ und d in $i(l) \cap t_2$ mit $c \leq_t a \leq_t b \leq_t d$. Damit ist $i : l \longrightarrow t$ aber kein Faktorvorkommen, ein Widerspruch (a oder b beeinträchtigen i). ∎

Analog läßt sich folgende Aussage beweisen.

Lemma 20 (Schnittlemma B) *Sei $t = t_1 \cdot t_2 \in \mathbb{M}(\Sigma, D)$, $l \in \mathbb{M}(\Sigma, D)$ eine zusammenhängende Spur und sei $i : l \longrightarrow t$ ein Faktorvorkommen mit $i(l) \cap t_1 \neq \emptyset$ und $i(l) \cap t_2 \neq \emptyset$. Dann gilt $i(l) \cap \max(t_1) \neq \emptyset$ und $i(l) \cap \min(t_2) \neq \emptyset$.*

Beweis: Angenommen $i(l) \cap \max(t_1) = \emptyset$ und $i(l) \cap \min(t_2) = \emptyset$. Da $i(l) \cap t_1 \neq \emptyset$, existiert ein Buchstabe a in l mit $i(a)$ in t_1 und da $i(l) \cap t_2 \neq \emptyset$, existiert ein Buchstabe b in l mit $i(b)$ in t_2. Da l zusammenhängend ist, existiert ein ungerichteter Pfad von a nach b in l, also auch von $i(a)$ nach $i(b)$ in t. Aber dieser Pfad enthält (mindestens) einen Buchstaben aus $\max(t_1)$ und einen aus $\min(t_2)$ im Widerspruch zu $i(l) \cap \max(t_1) = \emptyset$ und $i(l) \cap \min(t_2) = \emptyset$. ∎

Damit können wir das folgende Resultat zeigen.

Lemma 21 *Sei $\mathcal{R}$ ein endliches Spurersetzungssystem mit zusammenhängenden linken Seiten. Dann ist das Resultat*

$$CONCATTRACE(T_1, T_2)$$

korrekt beschriftet, wenn die Eingaben T_1, T_2 korrekt beschriftet waren.

Beweis: Die Beschriftung λ_1 wird bereits in CONCAT korrekt durchgeführt. Die Beschriftung λ_2 ist nach der CONCAT-Operation evtl. fehlerhaft. Sei T^i eine t^i repräsentierende Struktur, $i \in \{1, 2\}$. Dann wird durch die unbeschriftete Struktur die Spur $t^1 \cdot t^2$ repräsentiert (komponentenweise Konkatenation). Durch diese Konkatenation bleiben alle Faktorisierungen von t^1 und t^2 erhalten, denn aus $t^1 = u^1lv^1$ bzw. $t^2 = u^3lv^2$ folgt $t = t^1t^2 = u^1lv^1t^2$ bzw. $t = t^1u^2lv^2$. Es können aber durch die Konkatenation neue Faktorvorkommen $l \longrightarrow t$, $l \in L_{\mathcal{R}}$ entstanden sein. Jedes dieser neuen Faktorvorkommen enthält mindestens einen maximalen Buchstaben aus t^1 (siehe Lemma 20). Für alle linken Seiten $l \in L_{\mathcal{R}}$ werden diese Vorkommen untersucht. Der betrachtete Bereich ist ausreichend, da nur Vorkommen, die an diesen Positionen beginnen, einen Buchstaben aus $\max(t^1)$ besitzen können. Da diese Analyse für jede Projektion durchgeführt wird, sind danach alle Faktorvorkommen entdeckt, die einen nichtleeren Schnitt mit $\max(t^1)$ besitzen, und da bei jedem neu entdeckten Faktor l in $t = ulv$ alle Minimalpositionen p_C, d.h. Verweise auf a_C mit $\pi_C(l) = a_C l'_C$, $\pi_C(t) = t_C = \pi_C(u) a_C l'_C \pi_C(v)$, für alle $C \in \mathcal{C}$ mit $\lambda_2(v) \cup \{l\}$, markiert werden (diese Positionen werden alle in Λ gespeichert), sind nach der Ausführung der for-Schleifen alle Blätter korrekt λ_2-beschriftet.

Analog wie für die Beschriftung λ_1 gilt die Invariante für die Datenstuktur: Für alle inneren Knoten v gilt

$$\lambda_2(v) \quad = \bigcup_{c \in \mathrm{children}(v)} \lambda_2(c).$$

Die Prozedur PROPAGATE2 beschriftet alle neu beschrifteten Nachfolger (diese sind alle in Λ abgelegt) und mit Induktion folgt, daß alle λ_2-Beschriftungen hiernach korrekt sind. ■

Theorem 5 *Sei T^1 eine die Spur t^1 repräsentierende Struktur und T^2 eine die Spur t^2 repräsentierende Struktur. Dann kombiniert*

$$\mathit{CONCATTRACE}(T^1, T^2)$$

die Strukturen zu einer $t^1 \cdot t^2$ repräsentierenden Struktur T in $\mathcal{O}(\log|t^1| + \log|t^2|)$ Zeiteinheiten.

Beweis: Mit dem Resultat über die C^*-Bäume ist ein Konkatenieren der einzelnen Komponenten T^1_C mit T^2_C in $\mathcal{O}(\log|\pi_C(t^1)| + \log|\pi_C(t^2)|) \leq \mathcal{O}(\log|t^1| + \log|t^2|)$ möglich. Die Bedingung der innersten Iteration läßt sich in

$$\mathcal{O}(\log|t^1| + \log|t^2|) = \mathcal{O}(\max(\log|t^1|, \log|t^2|) + 1)$$

testen, und dieser Test wird höchstens $c = \sum_{l \in L_{\mathcal{R}}, C \in \mathcal{C}} |\pi_C(l)| = \mathcal{O}(1)$ oft durchgeführt. Damit ist die Kardinalität von Λ durch $|\Lambda| = \mathcal{O}(1)$ beschränkt. Jeder PROPAGATE2 Aufruf benötigt $\mathcal{O}(\max(\log|t^1|, \log|t^2|) + 1)$ Zeiteinheiten und es gibt insgesamt $|\Lambda| = \mathcal{O}(1)$ viele Aufrufe. Zusammengefaßt ergibt sich die Zeitkomplexität $\mathcal{O}(\log|t^1| + \log|t^2|)$.

Das vorhergehende Lemma sichert mit dem Korrektheitsresultat für die Methode CONCAT die strukturellen Eigenschaften von T zu. ■

5.4.2 SPLITTRACE

Invers zur Operation CONCATTRACE zerteilt die Operation SPLITTRACE eine Darstellung $T = (T_C)_{C \in \mathcal{C}}$ der Spur t entlang eines Schnittes $a = (a_C)_{C \in \mathcal{C}}$ in T. Als Resultat erhalten wir die Darstellungen $T^i = (T^i_C)_{C \in \mathcal{C}}$ der Spuren t^i, $i \in \{1, 2\}$ mit der Eigenschaft $t^1 \cdot t^2 = t$. Weiter korrespondiert das Tupel der minimalen Elemente von t^2, $\pi(\min(t^2))$ zum Schnitt $(a_C)_{C \in \mathcal{C}}$.

Auch hier wird mit einer Basisoperation die Darstellung erst komponentenweise faktorisiert. Danach werden die fehlerhaften λ_2-Beschriftungen korrigiert.

```
Algorithm SPLITTRACE (T, a)
  Comment Input: a trace representation with cut a.
          Output: two representations devided by leaves in a.
  Method
    Λ ← ∅
    for each C ∈ 𝒞 do
      ⟨T¹_C, T²_C⟩ ← SPLIT(T_C, a_C)
    end_for
    Comment At this point the λ₂-labeling is violated.
            All other properties are OK.
    for each C ∈ 𝒞 do
      for all l ∈ L_ℛ with Σ(l) ∩ C ≠ ∅ do
        let r be the maximal scann range |π_C(l)| − 1
        let v₁, . . . , v_r be the right-most leaves in T¹_C
        Λ ← Λ ∪ {v₁, . . . , v_r}
        for i := 1 to r do
          while there is no factor l with minimal letter v_i
                and l ∈ λ₂(v_i) do
            λ₂(v_i) ← λ₂(v_i) \ {l}
            let P = {p_C | the linked references of v_i to the
                           (leaves) at the other C-projections }
            for all p ∈ P do λ₂(p) ← λ₂(p) \ {l}
            Λ ← Λ ∪ P
          end_while
        end_for
      end_for
    end_for
    while Λ ≠ ∅ do
      let v ∈ Λ    Λ ← Λ \ {v}    PROPAGATE2(v)
    end_while
    return(T)
  end_Method
```

Analog zu Theorem 5 erhalten wir aus Lemma 19 das

Theorem 6 *Sei $\mathcal{R}$ ein endliches Spurersetzungssystem mit zusammenhängenden linken Seiten. Weiter sei T eine die Spur t repräsentierende Struktur und a ein die Faktorisierung $t = t_1 \cdot t_2$ repräsentierender Schnitt. Dann partitioniert*

$$\mathit{SPLITTRACE}(T, a)$$

die Struktur in eine Spurrepräsentation T_1 von t_1 und T_2 von t_2 in $\mathcal{O}(\log |t|)$ Zeiteinheiten. ∎

5.5 Der Reduktionsalgorithmus

Mit den bis jetzt formulierten Primitiven können wir eine Prozedur FACTORIZE formulieren, mit der ein Faktorvorkommen via die zweite Beschriftung gefunden wird. Die Faktorisierung kann mit der oben vorgestellten Faktorisierungsmethode durchgeführt werden, nachdem ein Schnitt mit den primitiven Operationen berechnet wurde. Diese Datenstuktur enthält Zugriffspfade auf alle Faktorisierungen $u \cdot l \cdot v$. D.h. ersetzt man die Beschriftungen durch entsprechende Parikh-Bilder

$$\begin{aligned} \lambda_1' : t_C &\longrightarrow \mathbb{N}^C \\ \lambda_2' : t_C &\longrightarrow \mathbb{N}^{L_\mathcal{R}}, \end{aligned}$$

so läßt sich die Wahl eines Faktorvorkommens i und der Faktorisierung von $\mathrm{ind}(i)$ vollständig festlegen. Unter anderem ist auch die Wahl einer kanonischen Links-Derivation möglich. Die Berechnung der Normalform erfolgt mit der Methode REDUCE.

PROCEDURE REDUCE(t)
  Comment Input: C^* tree trace representation.
      Output: the representation of the normal form.
  Method
  while $\exists l \in \lambda_2(t)$ **do**
    calculate two cuts c_1 c_2, factorizing t into u, l, and v
    $\langle u, v' \rangle \leftarrow$ SPLITTRACE(t, c_1)
    $\langle x, v \rangle \leftarrow$ SPLITTRACE(v', c_2)
    $t \leftarrow$CONCATTRACE(u, r)
    $t \leftarrow$CONCATTRACE(t, v)
  end_ while
  return(t)
end_Method

Aus der Vollständigkeit und Korrektheit der Beschriftungsinformation ergibt sich

Theorem 7 *Sei $\mathbb{M} = \mathbb{M}(\Sigma, D)$ ein frei partiell kommutatives Monoid und $\mathcal{R}$ ein endliches noethersches Spurersetzungssystem mit nur zusammenhängenden linken Seiten, dann existiert ein Algorithmus, der eine beliebige, durch eine Wahl spezifizierbare, irreduzible Form einer Spur $t \in \mathbb{M}$ in der Zeit $\mathcal{O}(|t| + d_\mathcal{R} \log d_\mathcal{R})$ berechnet.*

Beweis: Mit den Beschriftungen und den oben vorgestellten Primitiven kann ein Faktor $l \in L_{\mathcal{R}}$ in einer Repräsentation einer Spur t in $\mathcal{O}(\log|t|)$ identifiziert werden. Beispielsweise findet man einen Faktor, indem man von einer mit $l \in X$ beschriebenen Wurzel aus zu einem mit $l \in \lambda_2(v)$ beschrifteten Blatt v absteigt. Dabei wird man durch die λ_2-Beschriftung geführt. Hat man dieses Blatt gefunden, so kann das Vorkommen über den Zusammenhang von l vervollständigt werden. Die Schnitte lassen sich unter Verwendung der Methode PRED vervollständigen. Ist die Spur irreduzibel, so sind alle λ_2-Beschriftungen der Wurzeln leer. Rechts- und Linksquotienten bzw. die Schnitte c_1 und c_2 lassen sich in $\mathcal{O}(\log|t|)$ Zeiteinheiten konstruieren, und die anschließende Konkatenation ist auch in $\mathcal{O}(\log|t|)$ Zeit möglich. ∎

Theorem 8 *Sei $\mathbb{M} = \mathbb{M}(\Sigma, D)$ ein frei partiell kommutatives Monoid und $\mathcal{R}$ ein noethersches Spurersetzungssystem mit nur zusammenhängenden linken Seiten, dann existiert ein Algorithmus, der eine Links-Derivation einer Spur $t \in \mathbb{M}$ in der Zeit $\mathcal{O}(|t| + d_{\mathcal{R}} \log d_{\mathcal{R}})$ berechnet.* ∎

Die Links-Derivation läßt sich auch bzgl. einer lexikographischen Ordnung definieren. Da die oben definierte Links-Derivation berechnet werden kann und nur konstannt viele unvergleichbare Reduktionsschritte existieren können Links-Derivation bzgl. aller Verfeinerungen dieser Ordnung ($\ll$, siehe Anhang A.2) auch in dieser Zeit berechnet werden.

Korollar 7 *Sei $\mathbb{M} = \mathbb{M}(\Sigma, D)$ ein frei partiell kommutatives Monoid und $\mathcal{R}$ ein noethersches Spurersetzungssystem mit nur zusammenhängenden linken Seiten, dann existiert ein Algorithmus, der eine Links-Derivation bzgl. einer Verfeinerung der Ordnung $\ll$ einer Spur $t \in \mathbb{M}$ in der Zeit $\mathcal{O}(|t| + d_{\mathcal{R}} \log d_{\mathcal{R}})$ berechnet.* ∎

Theorem 9 *Sei $\mathbb{M} = \mathbb{M}(\Sigma, D)$ ein frei partiell kommutatives Monoid und $\mathcal{R}$ ein gewichtsreduzierendes Spurersetzungssystem mit nur zusammenhängenden linken Seiten, dann existiert ein Algorithmus, der eine beliebige, durch eine Wahl spezifizierbare, irreduzible Form einer Spur $t \in \mathbb{M}$ in der Zeit $\mathcal{O}(|t| \log |t|)$ berechnet.* ∎

Bemerkung 2 *Man beachte, daß alle Resultate nur nicht-uniforme Komplexitätsaussagen sind.*

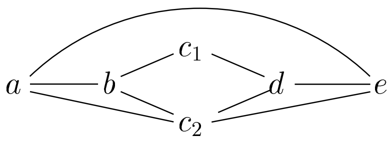
Betrachtet man beispielsweise das Wortproblem für kommutative Semigruppen. Im uniformen Fall besteht eine Instanz aus einem Tupel (Σ, S, u, v), wobei S ein die Gruppe beschreibendes Gleichungssystem ist (modulo Kommutativität). u und v sind Vektoren aus $\mathbb{N}^{\Sigma}$ und die Frage lautet: „Ist die Gleichung $u \equiv v \pmod S$ wahr?“. Dieses Problem ist sogar exponentiell-platzhart [MM82]. Im nicht-uniformen Fall transformiert man das Glei-
chungssystem lediglich in ein äquivalentes konfluentes und gewichtsreduzierendes Ersetzungssystem $S' \subseteq \mathbb{N}^{\Sigma} \times \mathbb{N}^{\Sigma}$, von dem wir in linearer Zeit mit einer der oben beschriebenen Algorithmen entscheiden können, ob $u \overset{*}{\Longleftrightarrow}_{\mathcal{R}} v$ erfüllt ist.

Die Restriktion auf Spurersetzungssysteme mit zusammenhängenden linken Seiten ist in gewisser Hinsicht natürlich. Interpretiert man das Ersetzungssystem als operationelle Semantik eines nebenläufigen Systems, so erscheint es unnatürlich, wenn Komponenten, die in jedem Kontext vollständig parallel ablaufen können, durch eine einzige Interaktion ersetzt werden sollen.

Eine Konsequenz aus der Bedingung der zusammenhängenden linken Seiten ist, daß sich dann alle interagierenden Faktoren in einer konstanten Umgebung befinden. Betrachtet man die durch die Kanten des Redukts von dem Abhängigkeitsgraphen induzierte Metrik, so befinden sich alle Faktoren l', die von einer Ersetzung $u'l'v' = ulr \Longrightarrow_{\mathcal{R}} urv$ betroffen sind, in der Kugel $D_R(r)$ in urv mit dem Radius $R = \max\{|l| \mid l \in L_{\mathcal{R}}\}$. Man beachte, daß sich die Abstände bzgl. dieser Metrik aber im Lauf einer Berechnung stark variieren können.

Beispiel 2 *Sei*

$(\Sigma, D) \quad =$



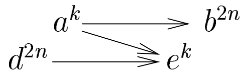
und $\mathcal{R}$ beschrieben durch folgende Ersetzungsregeln.

$$b^2c_1d^2 \Longrightarrow_{\mathcal{R}} b^2d^2, \quad b^2c_2 \Longrightarrow_{\mathcal{R}} b^2c_1, \quad d^2c_2b^2 \Longrightarrow_{\mathcal{R}} bc_2d, \quad ae \Longrightarrow_{\mathcal{R}} c_2$$

Alle linken Seiten sind zusammenhängend.

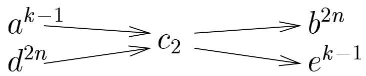
Beginnen wir mit der Spur $t = a^kb^{2n}c_1d^{2n}e^k$, $k \geq 1$ und $n \geq 2$. Man beachte, daß ae kein Faktor ist (Sprungsequenz[4] abc_1de), und daß die Buchstaben a, e in t die Distanz $d \geq 4n+2$ besitzen. Der erste Ersetzungsschritt führt zu

$t_1 \quad =$



und der nächste zu

$t_2 \quad =$



Nach n weiteren Ersetzungen, erhalten wir $t_3 = a^{k-1}b^nc_2d^ne^{k-1}$ und, da $n \geq 2$, ersetzen wir das durch $t_4 = a^{k-1}b^nc_1d^ne^{k-1}$.

Dieses Beispiel illustriert wie sich die Abstände verändern. Dabei treten insbesondere sehr große und globale Abstandsänderungen auf, wenn Artikulationsknoten eliminiert werden und keine verbindenene Knoten eingefügt werden. Dann ändern sich Abstände von endlichen Werten auf unendliche.

[4]Sprungsequenzen sind im Anhang definiert.

6 Effizienzsteigerung durch Sichtbarkeit bei Ersetzungssystemen mit zusammenhängenden linken Seiten

6.1 Sichtbarkeit

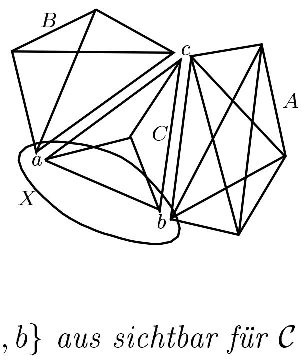
Die Baumstrukturen wurden wegen der fehlenden transitiven Relationen im Abhängigkeitsgraphen eingeführt. Für Abhängigkeitsalphabete, die nur „sichtbare" Buchstaben enthalten, kann auf diese Zugriffsstrukturen verzichtet werden.

Sei (Σ, D) wieder ein Abhängigkeitsalphabet und $\mathbb{M} = \mathbb{M}(\Sigma, D)$ das erzeugte Monoid. $\mathcal{C}$ sei eine Cliquenüberdeckung von (Σ, D) und die Spuren $t \in \mathbb{M}$ seien in der Projektionsdarstellung $\pi(t) \in \prod_{C \in \mathcal{C}} C^*$. $\mathcal{R}$ sei, wie gehabt, das Ersetzungssystem und $L_{\mathcal{R}}$ die Menge der linken Seiten, die alle zusammenhängend sein sollen.

Wir nennen einen Buchstaben $c \in \Sigma$ von $X \subseteq \Sigma$ aus bezüglich der Cliquenüberdeckung $\mathcal{C}$ *sichtbar*[5], wenn für alle Cliquen $A, B \in \mathcal{C}$ mit $c \in A \cap B$ eine Clique $C \in \mathcal{C}$, ein Buchstabe $a \in A \cap X$ und ein Buchstabe $b \in B \cap X$ mit $\{a, b, c\} \subseteq C$ existiert.

Beispiel 3 *Sei $X \subseteq \Sigma$. Dann ist jeder Buchstabe $c \in X$ von X aus sichtbar. Wir setzen $a = b = c \in X$ und wählen eine Clique $C \in \mathcal{C}$, die c enthält.*

In dem Abhängigkeitsalphabet



ist der Buchstabe c von $X = \{a, b\}$ aus sichtbar für $\mathcal{C} = \{A, B, C\}$

[5]Man beachte, daß die Sichtbarkeit von der zugrundeliegenden Cliquenüberdeckung abhängig ist und daß damit jede Clique einen nichtleeren Schnitt mit X besitzt.

Seien $\mathcal{C} = \{C_1, \dots, C_k\}$ und $t, l \in \mathbb{M}$ mit $\pi(t) = (t_C)_{C\in\mathcal{C}} \in \prod_{C\in\mathcal{C}} C^*$ und $\pi(l) = (l_C)_{C\in\mathcal{C}} \in \prod_{C\in\mathcal{C}} C^*$. Sei weiter

$$\begin{pmatrix} t_{C_1} \\ \vdots \\ t_{C_k} \end{pmatrix} = \begin{pmatrix} u_{C_1} \cdot l_{C_1} \cdot v_{C_1} \\ \vdots \\ u_{C_k} \cdot l_{C_k} \cdot v_{C_k} \end{pmatrix} = \begin{pmatrix} u_{C_1} \\ \vdots \\ u_{C_k} \end{pmatrix} \cdot \begin{pmatrix} l_{C_1} \\ \vdots \\ l_{C_k} \end{pmatrix} \cdot \begin{pmatrix} v_{C_1} \\ \vdots \\ v_{C_k} \end{pmatrix},$$

$u_{C_i}, v_{C_i}, l_{C_i}, t_{C_i} \in C_i^*$ für alle $i \in [k]$. Dann entspricht nicht jede dieser Faktorisierungen (in $\prod_{C\in\mathcal{C}} C^*$) einer Faktorisierung $\pi(u) \cdot \pi(l) \cdot \pi(v)$ mit $\pi(u) = (u_C)_{C\in\mathcal{C}}$, $\pi(l) = (l_C)_{C\in\mathcal{C}}$ und $\pi(v) = (v_C)_{C\in\mathcal{C}}$. Die notwendige und hinreichende Bedingung ist ein Korollar aus Lemma 42 bzw. folgt aus der Charakterisierung in Lemma 6 im vorigen Kapitel.

Korollar 8 *Sei $\mathcal{C}$ eine Cliquenüberdeckung von (Σ, D) und $t, l \in \mathbb{M}(\Sigma, D)$. Dann gilt $t = u \cdot l \cdot v$ genau dann, wenn es Wörter $u_C, v_C \in C^*$ für alle $C \in \mathcal{C}$ gibt mit*

(i) $\pi_C(t) = u_C \pi_C(l) v_C$ für alle $C \in \mathcal{C}$ und

(ii) $|u_A|_a = |u_B|_a$ für alle $A, B \in \mathcal{C}$ und für alle $a \in A \cap B$.

Es gilt dann $\pi(u) = (u_C)_{C\in\mathcal{C}}$ und $\pi(v) = (v_C)_{C\in\mathcal{C}}$ für eine Faktorisierung $t = u \cdot l \cdot v$ in $\mathbb{M}(\Sigma, D)$. ∎

Das folgende Beispiel illustriert ein Problem, das durch unsichtbare Buchstaben auftreten an einer Derivation.

Beispiel 4 *Sei* $(\Sigma, D) = \left[\begin{tikzcd}[row sep=small] & & & c \arrow[dlll, dash] \arrow[drrr, dash] & & & \\ a \arrow[rr, dash] & & d \arrow[rr, dash] & & e \arrow[rr, dash] & & b \end{tikzcd}\right]$ *das Abhängigkeitsalphabet und $l = [adeb]$. Es gibt genau eine Cliquenüberdeckung $\mathcal{C} = \{A, B, C, D, E\}$ mit $A = \{a, c\}$, $B = \{b, e\}$, $C = \{b, c\}$, $D = \{a, d\}$ und $E = \{d, e\}$. Der Buchstabe c ist unsichtbar von $\Sigma(l) = \{a, d, e, b\}$, da keine Clique existiert, die $\{a, b, c\}$ enthält. Betrachten wir die beiden einelementigen Spurersetzungssysteme*

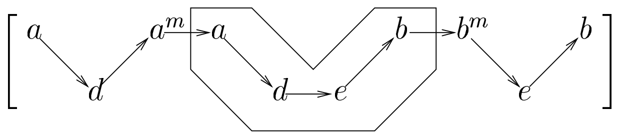
$$\mathcal{R}_1 = \{l \to 1\} \quad \textit{und} \quad \mathcal{R}_2 = \{l \to c\}$$

und untersuchen eine Reduktion der Spur $t = [ada^m adebb^m eb]$. Wir erhalten für t die Darstellung

$$t = \begin{pmatrix} aa^m a \\ ebb^m eb \\ bb^m b \\ ada^m ad \\ ddee \end{pmatrix} \cong \left[\begin{tikzcd}[column sep=small] a \arrow[dr] & & a^m \arrow[r] & a \arrow[dr] & & & b \arrow[r] & b^m \arrow[dr] & & b \\ & d \arrow[ur] & & & d \arrow[r] & e \arrow[ur] & & & e \arrow[ur] & \end{tikzcd}\right]$$

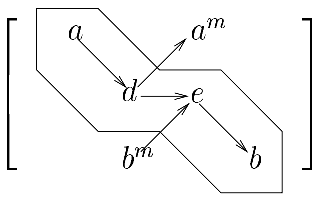
Es existiert genau eine Faktorisierung $t = ulv$, die in der Darstellung durch Rechtecke in jeder Komponente hervorgehoben ist.

$$t = \begin{pmatrix} aa^m\boxed{a} \\ \boxed{eb}\,b^m eb \\ \boxed{b}\,b^m b \\ ada^m\boxed{ad} \\ d\boxed{de}\,e \end{pmatrix} \cong$$



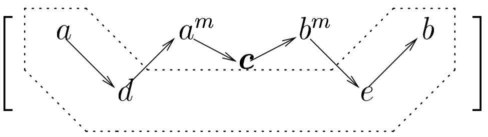
Sei $t \Longrightarrow_{\mathcal{R}_1} t_1$ und $t \Longrightarrow_{\mathcal{R}_2} t_2$. Nach dem Reduktionsschritt erhalten wir ein neues Vorkommen sowohl in t_1 als auch in t_2, das ein Kandidat für eine Faktorisierung ist.

$$t_1 = \begin{pmatrix} \boxed{a}\,a^m \\ b^m\boxed{eb} \\ b^m\boxed{b} \\ \boxed{ad}\,a^m \\ \boxed{de} \end{pmatrix} \cong$$



In t_1 ist der hervorgehobene Teil ein Faktor, was hier in konstanter Zeit überprüft werden kann. (Eine Konsequenz aus Theorem 12.)

$$t_2 = \begin{pmatrix} \boxed{a}\,a^m\boldsymbol{c} \\ b^m\boxed{eb}\,b^m \\ \boldsymbol{c}b^m\boxed{b} \\ \boxed{ad}\,a^m \\ \boxed{de} \end{pmatrix} \cong$$



In t_2 sind die markierten Teile jedoch kein Faktor. Die Anzahl der cs vor dem a ist 0, hingegen ist die Anzahl der cs, die vor einem b liegen, 1.

Das Problem besteht also darin, eine dynamische Struktur zu finden, mit der die Parikh-Bilder $|u_C|_a$ und $|u_{C'}|_a$ für $C, C' \in \mathcal{C}$ und $a \in C \cap C'$ einer Faktorisierung $(u_C l_C v_C)_{C\in\mathcal{C}}$ schnell verglichen werden können. Leider sind diese Werte nicht durch Konstanten wie bei freien Monoiden beschränkt. Gleichzeitig sollen auch Ersetzungen effizient durchführbar sein. Beispielsweise könnte man zu jedem Buchstaben das Parikh-Bild des von ihm induzierten Präfix assoziieren. Das hätte aber zur Folge, daß ein Ersetzungsschritt globale Aktualisierungen nach sich ziehen würde.

Für sichtbare Buchstaben gilt das folgende (für die Korrektheit des Algorithmus zentrale) Lemma

Lemma 22 *Sei $l \in \mathbb{M}(\Sigma, D)$ und sei $\mathcal{C}$ eine Cliquenüberdeckung von (Σ, D). Weiter sei $\pi(t) = (t_C)_{C\in\mathcal{C}} = (u_C)_{C\in\mathcal{C}} \cdot (l_C)_{C\in\mathcal{C}} \cdot (v_C)_{C\in\mathcal{C}}$ in $\prod_{C\in\mathcal{C}} C^*$ das π-Bild einer Spur $t \in \mathbb{M}(\Sigma, D)$ so, daß $|u_A|_a = |u_B|_a$ für alle $a \in \Sigma(l)$ und für alle $A, B \in \mathcal{C}$ mit $a \in A \cap B$. Dann gilt für zwei Cliquen $A, B \in \mathcal{C}$ und einen von $\Sigma(l)$ aus sichtbaren Buchstaben $c \in A \cap B$ die Beziehung $|u_A|_c = |u_B|_c$.*

Beweis: Seien $A, B \in \mathcal{C}$ beliebige Cliquen und c von $\Sigma(l)$ aus sichtbar. Dann ist für $c \in \Sigma(l)$ die Behauptung trivial. Sei $c \notin \Sigma(l)$, $a \in \Sigma(l) \cap A$ und $b \in \Sigma(l) \cap B$, so daß eine Clique $C \in \mathcal{C}$ mit $\{a, b, c\} \subseteq C$ existiert (a, b, C existieren, da c sichtbar ist). Wir zeigen: $|u_A|_c = |u_B|_c$. Sei $|u_A|_c < |u_B|_c$. Aus der Voraussetzung folgt $|u_C|_a = |u_A|_a$ und $|u_C|_b = |u_B|_b$ sowie $a, b \in \Sigma(l)$. Betrachten wir das erste a in l, d.h. das i-te a in t mit $i = |u_A|_a + 1 = |u_C|_a + 1$. Weiter betrachten wir das erste b in l, also das j-te b in t mit $j = |u_B|_b + 1 = |u_C|_b + 1$. Dann stehen vor dem j-ten b mehr cs als vor dem i-ten a in $\pi_{\{a,b,c\}}(t_C)$. Also steht ein c zwischen diesem a und diesem b in $\pi_{\{a,b,c\}}(t) = \pi_{\{a,b,c\}}(t_C)$ und damit steht auch ein c zwischen dem i-ten a und dem j-ten b in t_C. Formal sei $x_a \in C^*$ das Präfix von t_C, das mit dem i-ten a endet und $x_b \in C^*$ das Präfix von t_C, das mit dem j-ten b endet. Dann gilt

$$\begin{array}{rclcrcl} t_C & = & u_C & l_C & v_C & = & x_a \quad y_a \\ & = & u_C & t'_C a l''_C b l'''_C & v_C & = & x_b \quad y_b \end{array}$$

für ergänzende $y_a, y_b \in C^*$. Da aber dann $|u_A|_C + 1 = |x_a|_c < |x_b|_c$, existiert $z \in C^*$ mit $x_a z = x_b$ und $|z|_c > 1$. Also enthält $l''_C b = z$ mindestens ein c, im Widerspruch zu $c \notin \Sigma(l)$. Entsprechendes gilt für $|u_A|_c > |u_B|_c$ ∎

6.2 Ein Linearzeit-Reduktionsalgorithmus

Schränken wir uns auf sichtbare Buchstaben ein, so können wir auf die C^*-Baumstruktur verzichten.

Theorem 10 *Sei $\mathcal{C}$ eine Cliquenüberdeckung von einem Abhängigkeitsalphabet (Σ, D), $\mathcal{R} \subseteq \mathbb{M}(\Sigma, D) \times \mathbb{M}(\Sigma, D)$ ein Ersetzungssystem mit den linken Seiten $L_\mathcal{R}$. Jede linke Seite sei zusammenhängend und alle Buchstaben $a \in \Sigma$ seien von $\Sigma(l)$ aus sichtbar für jede linke Seite $l \in L_\mathcal{R}$. Dann kann eine Derivation $t \overset{n}{\Longrightarrow}_\mathcal{R} t'$ in $\mathcal{O}(n + |t|)$ Zeiteinheiten berechnet werden.*

Beweis: Wir beweisen das Theorem durch Angabe eines Algorithmus, der eine Derivation $ulv \overset{1}{\Longrightarrow}_\mathcal{R} urv$, $l \to r \in \mathcal{R}$, in konstanter Zeit nach $\mathcal{O}(|t|)$ Initialisierungsschritten berechnen kann.

Wir repräsentieren in diesem Algorithmus eine Spur $t \in \mathbb{M}(\Sigma, D)$ als Worttupel $\pi(t) = (t_C)_{C \in \mathcal{C}}$. Weiter sollen wieder Referenzen zwischen den Buchstaben $a \in \Sigma$ in den Komponenten von $\pi(a)$ in $\pi(t')\pi(a)\pi(t'')$ existieren, die beispielsweise durch Zeigerstrukturen realisiert werden können, und es soll eine Liste L_l der Faktorvorkommen $l \longrightarrow t$ für alle $l \in L_\mathcal{R}$ berechnet sein. Alle diese Strukturen lassen sich durch eine Vorwegberechnung in $\mathcal{O}(|t|)$ mit dem folgenden Algorithmus konstruieren.

Die Idee des Algorithmus ist die Verwaltung von Längeninformation, d.h. für jede Clique und jedes Präfix u_C von t_C sollen das Parikh-Bild $(|u_C|_a)_{C \in \mathcal{C}, a \in C} = (n(C, a, |u_C|))_{C \in \mathcal{C}, a \in C}$ berechnet werden. Dabei Identifizieren wir das Präfix mit seiner Länge. Alle „synchronen" Muster, d.h. Faktorisierungen $t_C = u_C l_C v_C$ aller Komponenten mit der Eigenschaft $|u_C|_a = |u_{C'}|_a$ für alle $A, B \in \mathcal{C}$ mit $a \in A \cap B$ sollen durch entsprechende Verweise als

Faktorvorkommen in eine Liste L_l aufgenommen werden. Diese können beispielsweise durch Tupel $(|u_C|)_{C\in\mathcal{C}, C\cap\Sigma(l)\neq\emptyset}$ repräsentiert werden.

PROCEDURE PREPROCESSING(t)

- **Comment** Input: a trace in word representation.
  Output: the internal representation of the trace.
- **Method**
  - **for all** $(C,a)\in\mathcal{C}\times\Sigma$ **do** $\eta_{C,a}\leftarrow 0$ **end_for**
  - **for all** $l\in L_{\mathcal{R}}$ **do** $L_l\leftarrow []$ **end_for**
  - **for all** $C\in\mathcal{C}$ **do** $t_C\leftarrow 1$ **end_for**
  - **while** $t=at'$ **do**
    - $t\leftarrow t'$
    - **for all** $(b,C)\in\Sigma\times\mathcal{C}$ **do**
      - **if** $a\in C$ and $b=a$
        - **then** $t_C\leftarrow t_Ca$
          - $\eta_{C,a}\leftarrow\eta_{C,a}+1$
          - $n(C,a,|t_C|)\leftarrow\eta_{C,a}$
        - **else** $n(C,b,|t_C|)\leftarrow n(C,b,|t_C|-1)$
      - **end_if**
    - **end_for**
  - **end_while**
  - **for all** $l\in L_{\mathcal{R}}$ **do**
    - **let** $a\in\Sigma(l)$ and $C\in\mathcal{C}$ with $a\in C$
    - $w\leftarrow t_C$
    - **while** $w=bw'$ **do**
      - **if** $w=\pi_C(l)w''$ **then**
        - scan along an undirected path through l the factorizations $t_C=u_Cl_Cv_C$ via cross references
        - **if** there exists a factorization $(t_C)_{C\in\mathcal{C}}=(u_Cl_Cv_C)_{C\in\mathcal{C}}$ with $\eta(a,A,|u_A|)=\eta(a,B,|u_B|)$ for all $A,B\in\mathcal{C}$ and all $a\in A\cap B$
          - **then** append the positions $(|u_C|)_{C\in\mathcal{C},C\cap\Sigma(l)\neq\emptyset}$ to L_l
        - **end_if**
      - **end_if**
    - **end_while**
  - **end_for**
  - **return** $((t_C)_{C\in\mathcal{C}},(L_l)_{l\in L_{\mathcal{R}}})$
- **end_Method**

Innerhalb der ersten drei Schleifen werden die Datenstrukturen initialisiert. Die darauf folgende while-Schleife ermittelt die einzelnen Projektionen t_C und die Längeninformation $|u_C|_a$ für die Präfixe u_C von t_C für alle $a\in C$. In der zweiten while-Schleife werden „synchrone“ Faktorisierungen als Vorkommen entdeckt und in den Listen L_l gespeichert.

Die Korrektheit folgt aus der Invariante $|u_C|_a=n(a,C,|u_C|)$ für alle $A\in\mathcal{C}$ und $a\in C$. Da die Spuren $l\in L_{\mathcal{R}}$ zusammenhängend sind, läßt sich der Scan vor der letzten Selektion in konstanter Zeit auf der Verweisstruktur durchführen. Die Referenzen ermöglichen die eineindeutige Zuordnung der Faktoren in anderen Komponenten (l ist zusammenhängend, alle Buchstaben sind von $\Sigma(l)$ aus sichtbar). Die Zeitkomplexität summiert sich zu $\mathcal{O}(|\mathcal{C}| \cdot |L_{\mathcal{R}}| \cdot |t| \cdot \sum_{l \in L_{\mathcal{R}}} |l|) = \mathcal{O}(|t|)$. Eine effizientere Methode liefert ein Fortschreiten mit Fehlerfunktionen wie im Algorithmus von Knuth, Morris und Pratt. Das Beispiel illustriert die Vorgehensweise für einen Scan.

Beispiel 5 *Sei* $(\Sigma, D) = a - b - c$, $t = \begin{pmatrix} ababa \\ bcb \end{pmatrix}$ *und* $l = \begin{pmatrix} aba \\ cb \end{pmatrix}$. *Das erste Vorkommen von aba in der ersten Komponente von* $\begin{pmatrix} \boxed{a\boldsymbol{b}a}ba \\ \boldsymbol{b}cb \end{pmatrix}$, *ist nicht die Projektion eines Faktors l, denn das b in in diesem Vorkommen korrespondiert zum ersten b in t, also auch zum ersten in der zweiten Komponente. Vor diesem b steht aber kein c. Das zweite Vorkommen von aba,* $\begin{pmatrix} ab\boxed{a\boldsymbol{b}a} \\ b\boxed{c\boldsymbol{b}} \end{pmatrix}$, *in der ersten Komponente ababa induziert ein Faktorvorkommen von l, denn das b in diesem Vorkommen korrespondiert zu dem b in der zweiten Komponente, vor dem ein c steht.*

Sind die Listen der Faktorvorkommen initialisiert, können wir einen Ersetzungsschritt in konstanter Zeit so durchführen, daß die Vollständigkeit dieser Listen eine Invariante ist. D.h., nach dem Ersetzungsschritt enthält die Liste L_l wieder genau alle Faktorvorkommen von l in der modifizierten Spur t für alle $l \in L_{\mathcal{R}}$. Wie im vorigem Kapitel bezeichnen wir mit $\max(t)$ die maximalen Buchstaben in t, d.h. die Buchstaben $a \in C$ für die die Projektion $t_C = t'_C a$ auf eine Clique $C \in \mathcal{C}$ mit a endet. Dual dazu bezeichnet $\min(t)$ die minimalen Buchstaben.

PROCEDURE REDUCEnSTEPS(t, n)
  Comment Input: trace in representations.
    Output: the decendant in representation.
  Method
    while $n > 0$ **and** $\exists i \in L_l$ for some $l \in L_{\mathcal{R}}$ with $l \to r \in \mathcal{R}$ **do**
      $n \leftarrow n - 1$
      factorize t by the tuple of positions i into $u = (u_C)_{C \in \mathcal{C}}$,
        $l = (l_C)_{C \in \mathcal{C}}$ and $v = (v_C)_{C \in \mathcal{C}}$ purge l and
        purge all occurrences $j : l' \longrightarrow t$ in $L_{l'}$ having
        an overlap with $i(l) \cup \max(u) \cup \min(v)$.
      $t \leftarrow u \cdot r \cdot v$
      insert all new occurrences $j : l' \longrightarrow t$, $l \in L_{\mathcal{R}}$ having
        some nonempty overlap with $r \cup \max(u) \cup \min(v)$
    end_ while
  return$(\langle t, L' \rangle)$
end_Method

Die erste Komplexanweisung

factorize t by i into $u = (u_C)_{C \in \mathcal{C}}$, $l = (l_C)_{C \in \mathcal{C}}$ and $v = (v_C)_{C \in \mathcal{C}}$
purge l and all occurrences $j : l' \longrightarrow t$ in $L_{l'}$ having
an overlap with $i(l) \cup \max(u) \cup \min(v)$.

läßt sich in konstanter Zeit abarbeiten. Eine Faktorisierung $t_C = u_C \cdot l_C \cdot v_C$ kann für alle $C \in \mathcal{C}$ in konstanter Zeit über die maximalen bzw. minimalen Buchstaben des Vorkommens von l berechnet werden. Jedes der Faktorvorkommen, die entfernt werden sollen, korrespondiert zu einer Faktorisierung $t_C = u'_C \cdot l'_C \cdot v'_C$ für alle $C \in \mathcal{C}$ so, daß ein $A \in \mathcal{C}$ existiert mit $||u_A| - |u'_A|| < |l'_C|$ (vgl. Lemma 19 und 20). Diese können in $\mathcal{O}(\sum_{l \in L_\mathcal{R}, C \in \mathcal{C}} |l_C|^2) = \mathcal{O}(1)$ Zeit aus den Listen $L_{l'}$, $l' \in L_\mathcal{R}$ über entsprechende Referenzen entfernt werden. Das Eliminieren von l und das Einsetzen von r geht offensichtlich in konstanter Zeit.

Analog zum Entfernen der überlappenden Faktoren werden von der Komplexanweisung

insert all new occurrences $j : l' \longrightarrow t$, $l \in L_\mathcal{R}$ having
some nonempty overlap with $r \cup \max(u) \cup \min(v)$

alle noch in den Listen fehlenden Faktorvorkommen in konstanter Zeit eingetragen. Die Konkatenation geschieht komponentenweise durch Verkettung der einzelnen Strings. Für alle in $L_{l'}$ fehlenden Faktorvorkommen, $l \in L_\mathcal{R}$, existiert eine Clique so, daß sich $t = urv = u'l'v'$ faktorisieren läßt mit $||u_C| - |u'_C|| < |l'_C|$ (vgl. Lemma 19). Diese Faktorisierungen lassen sich über die Zusammenhangsbedingung in $\mathcal{O}(\sum_{l \in L_\mathcal{R}, C \in \mathcal{C}} |l_C|^2) = \mathcal{O}(1)$ Zeiteinheiten auffinden bzw. die Liste $L_{l'}$ läßt sich in dieser Zeit vervollständigen.

Über die gegenseitigen Referenzierungen der Buchstaben a_C in $t = t'at''$ mit $\pi(t) = \pi(t')(a_C)_{C \in \mathcal{C}} \pi(t'')$ werden nur Faktorisierungen

$$\begin{pmatrix} t_{C_1} \\ \vdots \\ t_{C_k} \end{pmatrix} = \begin{pmatrix} u'_{C_1} \\ \vdots \\ u'_{C_k} \end{pmatrix} \cdot \begin{pmatrix} l'_{C_1} \\ \vdots \\ l'_{C_k} \end{pmatrix} \cdot \begin{pmatrix} v'_{C_1} \\ \vdots \\ v'_{C_k} \end{pmatrix}$$

mit $\mathcal{C} = \{C_1, \ldots, C_k\}$ und $|u'_A|_a = |u'_B|_a$ für alle $A, B \in \mathcal{C}$, $a \in A \cap B \cap \Sigma(l)$ betrachtet. Da alle Buchstaben von allen Mengen $\Sigma(l) \subseteq \Sigma$ aus sichtbar (bzgl. $\mathcal{C}$) sind, folgt mit Lemma 22, daß jede so berechnete Faktorisierung die Bedingung $|u'_A|_c = |u'_B|_c$ für alle $A, B \in \mathcal{C}$ und $c \in A \cap B$ erfüllt. Nach Lemma 8 induziert dann dieses Vorkommen die Faktorisierung $t = \pi^{-1}((u'_C)_{C \in \mathcal{C}}) \cdot l' \cdot \pi^{-1}((v'_C)_{C \in \mathcal{C}})$.

Wir erhalten als Invariante die Semantik der Datenstruktur. Damit ist das Resultat des Algorithmus ein Derivat der Eingabe. Die Zeitkomplexität für die gesammte Berechnung ist $\mathcal{O}(|t| + n)$, da der Rumpf der Iteration konstante Zeit benötigt. ∎

Ein iteriertes Anwenden der oben beschriebenen Methode führt zu dem folgenden Resultat.

Theorem 11 *Sei $\mathcal{C}$ eine Cliquenüberdeckung von einem Abhängigkeitsalphabet (Σ, D), $\mathcal{R} \subseteq \mathbb{M}(\Sigma, D) \times \mathbb{M}(\Sigma, D)$ ein Ersetzungssystem mit den linken Seiten $L_\mathcal{R}$. Jede linke Seite sei zusammenhängend und alle Buchstaben $a \in \Sigma$ seien von $\Sigma(l)$ aus sichtbar für* *jede linke Seite $l \in L_{\mathcal{R}}$. Dann kann eine irreduzible Form $\hat{t} \in \mathrm{Irr}(\mathcal{R})$ von $t \in \mathbb{M}(\Sigma, D)$ in $\mathcal{O}(d_{\mathcal{R}})$ Zeit berechnet werden.*

Beweis: Die Spur ist irreduzibel, wenn alle Listen L_l für $l \in L_{\mathcal{R}}$ leer sind. Ein Reduktionsschritt läßt sich in $\mathcal{O}(1)$ Zeiteinheiten durchführen. ∎

Korollar 9 *Sei $\mathcal{C}$ eine Cliquenüberdeckung von einem Abhängigkeitsalphabet (Σ, D), $\mathcal{R} \subseteq \mathbb{M}(\Sigma, D) \times \mathbb{M}(\Sigma, D)$ ein Ersetzungssystem mit den linken Seiten $L_{\mathcal{R}}$. Jede linke Seite sei zusammenhängend, es existiert eine Clique $C \in \mathcal{C}$ mit $\pi_C(l) \neq 1$ für alle $l \in L_{\mathcal{R}}$ und alle Buchstaben $a \in \Sigma$ seien von $\Sigma(l)$ aus sichtbar für jede linke Seite $l \in L_{\mathcal{R}}$. Dann kann eine Links-Derivation $t \overset{*}{\Longrightarrow}_{\mathcal{R}} \hat{t} \in \mathrm{Irr}(\mathcal{R})$ in $\mathcal{O}(d_{\mathcal{R}})$ Zeit berechnet werden.*

Beweis: Seien i und j zwei Vorkommen von zusammenhängenden Spuren l. Wir definieren die *Vorkommenordnung* $\ll$:

$$i \ll j \quad :\Longleftrightarrow \quad \mathrm{Post}(j) \subseteq \mathrm{Post}(i).$$

Die Listen L_l, $l \in \mathcal{R}$, sind kanonisch durch $\ll$ linear geordnet (siehe Lemma 44) und diese Ordnung kann lokal durch konstante Operationen aufrecht erhalten werden. Die relative Position eines Vorkommen $i : l \longrightarrow t$ zu einem Vorkommen $j : l' \longrightarrow t$ ist auf der Projektion t_C überprüfbar. Seien i und j durch die Faktorisierungen $u_C l_C v_C = u'_C l'_C v'_C$ induziert. Dann impliziert $|u_C| < |u'_C|$ die Relation $i \ll j$. Bei Gleichheit kann eine entsprechende Analyse auf einer anderen Komponente erfolgen. Damit ist $i \ll j$ innerhalb einer durch die Konstante $\max\{|l| \mid l \in L_{\mathcal{R}}\}$ beschränkten Umgebung in konstanter Zeit entscheidbar. Stellt man alle Vorkommen in einer Liste L durch Verweise auf Positionen in t_C dar, dann läßt sich eine Links-Derivation durch iteriertes reduzieren eines minimalen Faktors berechnen. Ein minimaler Faktor kann in dieser Datenstruktur in konstanter Zeit bestimmt werden. ∎

Korollar 10 *Sei $\mathcal{C}$ eine Cliquenüberdeckung von einem Abhängigkeitsalphabet (Σ, D), $\mathcal{R} \subseteq \mathbb{M}(\Sigma, D) \times \mathbb{M}(\Sigma, D)$ ein gewichtsreduzierendes Ersetzungssystem mit den linken Seiten $L_{\mathcal{R}}$. Jede linke Seite sei zusammenhängend und alle Buchstaben $a \in \Sigma$ seien von $\Sigma(l)$ aus sichtbar für jede linke Seite $l \in L_{\mathcal{R}}$. Dann kann eine irreduzible Form $\hat{t} \in \mathrm{Irr}(\mathcal{R})$ von $t \in \mathbb{M}(\Sigma, D)$ in $\mathcal{O}(|t|)$ Zeit berechnet werden.*

Bemerkung 3 *Die Voraussetzungen sind insbesondere für längenverkürzende Semi-Thue Systeme erfüllt. Damit verallgemeinert dieses Resultat die Aussage von Book [Boo82]. Wie im vohergehenden Kapitel in Korollar 7 lassen sich auch Links-Derivation bzgl. lexikographischen Ordnung in linearer Zeit berechnen.*

Andere Anwendungen des Sichtbarkeitskonzeptes führen zu ähnlichen Komplexitätsresultaten. So wurde beispielsweise in [BD93] gezeigt:

Theorem 12 *Für eine Cliquenüberdeckung $\mathcal{C}$, ein längenverkürzendes Einregelsystem $\{l \to r\} = \mathcal{R}$ mit l zusammenhängend, $\pi_C(l) \neq 1$ für alle $C \in \mathcal{C}$, so daß (nur) alle Buchstaben $a \in \Sigma(r)$ von $\Sigma(l)$ aus sichtbar sind, existiert ein Linearzeitalgorithmus, der eine Normalform berechnet.*

7 Dekomposition von Spurmonoiden

Die Darstellung von Spuren durch die Cliquenprojektionen bzw. die Sichtbarkeitsbedingung führt zu effizienten Algorithmen. Der Ansatz zur Darstellung von einer Subklasse von Spuren, der in diesem Kapitel entwickelt wird, liefert weitere effiziente Algorithmen.

Wir entwickeln eine Methode, die es erlaubt, für eine große Klasse frei partiell kommutativer Monoide eine fixe „Zerfaserung" der Monoidstruktur anzugeben, die es ermöglicht, modular „zerfaserte" Faktoren zu rekombinieren.

Seien (Σ, D_Σ) und (Δ, D_Δ) zwei disjunkte Abhängigkeitsalphabete ($\Sigma \cap \Delta = \emptyset$). Dann definieren wir die *direkte Summe* via

$$(\Sigma, D_\Sigma) \oplus (\Delta, D_\Delta) \;:=\; (\Sigma \cup \Delta, D_\Sigma \cup D_\Delta),$$

und das *Komplexprodukt* mit

$$(\Sigma, D_\Sigma) * (\Delta, D_\Delta) \;:=\; \big(\Sigma \cup \Delta, D_\Sigma \cup D_\Delta \cup (\Sigma \times \Delta) \cup (\Delta \times \Sigma)\big).$$

Definition 2 *Wir bezeichnen die kleinste Subklasse der Abhängigkeitsalphabete, die unter direkter Summe und Komplexprodukt abgeschlossen ist, und die einelementigen Abhängigkeitsalphabete enthält, als* Cographen-Abhängigkeitsalphabete.

Die Spurmonoide, die von diesen Abhängigkeitsalphabeten erzeugt werden, nennen wir Cographen-Monoide.

Es gibt eine graphentheoretische Charakterisierung für Cographen-Abhängigkeitsalphabete. Interpretiert man das Abhängigkeitsalphabet (Σ, D) als ungerichteten Graph und entfernt alle Schlingen, so ist das Abhängigkeitsalphabet (Σ, D) genau dann ein Cographen-Abhängigkeitsalphabet, wenn der Graph $(\Sigma, D \setminus \{(a,a) \mid a \in \Sigma\})$ keinen induzierten Untergraph der Form

$$\big(\{a,b,c,d\}, \{(a,b),(b,c),(c,d)\}\big) \quad = \quad [a - b - c - d]$$

enthält, $\{a,b,c,d\} \subseteq \Sigma$, d.h. falls (Σ, D) nicht P_4 als induzierten Untergraph enthält.

Eine *Deskription* eines Abhängigkeitsalphabetes ist ein linearer Term, der aus disjunkten Abhängigkeitsalphabeten und den (Infix-)operatoren $+$ und $*$ besteht. Die *Dekomposition* eines Abhängigkeitsalphabets (Σ, D) berechnet sich wie folgt. Ist der Graph (Σ, D) nicht zusammenhängend, so ist die Dekomposition $\mathrm{Dec}(\Sigma, D)$ die Menge der Zusammenhangskomponenten von (Σ, D). Ist (Σ, D) zusammenhängend, dann besteht die Menge $\mathrm{Dec}(\Sigma, D)$ aus den Komplementen der Zusammenhangskomponenten des Komplements von (Σ, D). Dabei verstehen wir unter dem Komplement von (Σ, D) den Graphen $(\Sigma, (\Sigma \times \Sigma) \setminus D)$.

Proposition 1 *Eine Dekomposition* $\{(\Sigma_i, D_i) \mid i \in [n]\}$ *von* (Σ, D) *entspricht der kanonischen Deskription. Ist* (Σ, D) *nicht zusammenhängend, so gilt* $(\Sigma, D) = \oplus_{i\in[n]}(\Sigma_i, D_i)$. *Anderenfalls gilt* $(\Sigma, D) = *_{i\in[n]}(\Sigma_i, D_i)$. ∎

Weiter bezeichnen wir mit $\mathrm{Dec}^*(\Sigma, D)$ die Menge aller Subalphabete $(\Delta, D_\Delta) \subseteq (\Sigma, D)$, die durch Deskriptionen über Elementen aus $Z(\Sigma, D)$ darstellbar sind. $Z(\Sigma, D)$ ist dabei die rekursive Zerlegung in nicht-dekompositionierbare Alphabete, d.h.

- $Z(\Sigma, D) = \{(\Sigma, D)\}$, wenn $\mathrm{Dec}(\Sigma, D) = \{(\Sigma, D)\}$.
- Anderenfalls gilt $Z(\Sigma, D) = \bigcup\{Z(\Delta, D_\Delta) \mid (\Delta, D_\Delta) \in \mathrm{Dec}(\Sigma, D)\}$.

Proposition 2 *Für ein Abhängigkeitsalphabet ist die Menge* $\mathrm{Dec}^*(\Sigma, D)$ *mit den Operationen*

$$\begin{aligned}(\Delta_1, D_1) \vee (\Delta_2, D_2) &= \bigcap\{(\Delta, D) \in \mathrm{Dec}^*(\Sigma, D) \mid \Delta_1 \subseteq \Delta, \Delta_2 \subseteq \Delta\}\\ (\Delta_1, D_1) \wedge (\Delta_2, D_2) &= \bigcup\{(\Delta, D) \in \mathrm{Dec}^*(\Sigma, D) \mid \Delta \subseteq \Delta_1, \Delta \subseteq \Delta_2\}\end{aligned}$$

ein Verband. ∎

Die graphentheoretischen Mengenoperationen sind komponentenweise definiert.

Für ein Abhängigkeitsalphabet (Σ, D) und eine Teilmenge $\Delta \subseteq \Sigma$ definieren wir das *von* Δ *induzierte Abhängigkeitsalphabet* $\langle\Sigma, D\rangle(\Delta)$ durch

$$\langle\Sigma, D\rangle(\Delta) = \bigvee\{(\Delta', D') \in \mathrm{Dec}(\Sigma, D) \mid \Delta \subseteq \Delta'\}.$$

Das ist das kleinste mit der Cographen-Dekomposition verträgliche Abhängigkeitsalphabet, daß Δ umfaßt.

Lemma 23 *Seien* $\mathbb{M}_\Sigma = \mathbb{M}(\Sigma, D_\Sigma)$ *und* $\mathbb{M}_\Delta = \mathbb{M}(\Delta, D_\Delta)$ *zwei frei partiell kommutative Monoide, erzeugt von den disjunkten Abhängigkeitsalphabeten* (Σ, D_Σ) *und* (Δ, D_Δ). *Dann gilt*

$$\begin{aligned}\mathbb{M}((\Sigma, D_\Sigma) \oplus (\Delta, D_\Delta)) &\cong \mathbb{M}_\Sigma \times \mathbb{M}_\Delta\\ \mathbb{M}((\Sigma, D_\Sigma) * (\Delta, D_\Delta)) &\cong \mathbb{M}_\Sigma * \mathbb{M}_\Delta,\end{aligned}$$

wobei $\mathbb{M}_\Sigma * \mathbb{M}_\Delta$ *das freie Produkt und* $\mathbb{M}_\Sigma \times \mathbb{M}_\Delta$ *das direkte Produkt von* $\mathbb{M}_\Sigma$ *und* $\mathbb{M}_\Delta$ *notiert.* ∎

7.1 Der Term-Morphismus

Sei $\mathrm{Term}(\Sigma, D)$ die Menge der *wohlgeformten Terme* über der Signatur

$$\Big\langle \times_i \mid i \in \mathbb{N}, *_j \mid j \in \mathbb{N}, t \mid t \in \mathbb{M}(\Sigma, D)\Big\rangle,$$

d.h. die aus den Funktionssymbolen $\times$ und $*$ beliebiger Arität und den Atomen (nullstellige Terme) $t \in \mathbb{M}(\Sigma, D)$ bestehenden Terme. Formal benötigen wir eigentlich eine mit Deskriptionen und Aritäten indizierte Signatur, aber der Lesbarkeit wegen verzichten wir auf diese syntaktisch schwerfällige Notation[6].

Wir definieren eine Abbildung

$$h_{(\Sigma,D)} : \mathbb{M}(\Sigma, D) \longrightarrow \mathrm{Term}(\Sigma, D),$$

die jeder Spur einen solchen Term zuordnet.

- Ist (Σ, D) nuklear, d.h. $\mathrm{Dec}(\Sigma, D) = \{(\Sigma, D)\}$, dann ist die Abbildung die Einbettung $h_{(\Sigma,D)} : x \longmapsto x$.
- Anderenfalls sei (Σ, D) zerlegbar mit der Dekomposition $\mathrm{Dec}(\Sigma, D) = \{(\Sigma_i, D_i) \mid i \in [k]\}$ und $k > 1$.
  (i) Ist das Abhängigkeitsalphabet $(\Sigma, D) = \oplus_{i\in[k]}(\Sigma_i, D_i)$ eine direkte Summe, so definieren wir für $\mathbb{M}(\Sigma, D) \cong \prod_{i\in[k]} \mathbb{M}(\Sigma_i, D_i)$

$$\begin{array}{rcl} h_{(\Sigma,D)} : \mathbb{M}(\Sigma, D) & \longrightarrow & \mathcal{T} \\ (t_1, \ldots, t_k) & \longmapsto & \times(h_{(\Sigma_1,D_1)}(t_1), \ldots, h_{(\Sigma_k,D_k)}(t_k)) \end{array}$$

  (ii) Ist (Σ, D) das Komplexprodukt $*_{i\in[k]}(\Sigma_i, D_i)$, d.h. $\mathbb{M}(\Sigma, D) \cong \mathop{*}\limits_{i\in[k]} \mathbb{M}(\Sigma_i, D_i)$, dann definieren wir

$$\begin{array}{rcl} h_{(\Sigma,D)} : \mathbb{M}(\Sigma, D) & \longrightarrow & \mathcal{T} \\ t_1 \cdot \ldots \cdot t_n & \longmapsto & *(h_{(\Sigma_{i_1},D_{i_1})}(t_1), \ldots, h_{(\Sigma_{i_n},D_{i_n})}(t_n)), \end{array}$$

  wobei $t_1 \cdot \ldots \cdot t_n$ die gewöhnliche (alternierende) Faktorsequenz notiert mit $\{i_1, \ldots, i_n\} \subseteq [k]$ und $i_j \neq i_{j+1}$ für $j \in [n-1]$ und $t_j \in \mathbb{M}(\Sigma_{i_j}, D_{i_j})$ für $j \in [n]$.

Ist (Σ, D) ein Cographen-Abhängigkeitsalphabet, so sind die Bilder Terme über der oben beschriebenen Signatur, die nur Wörter $a^n \in a^* \cong \mathbb{N}^{\{a\}} \cong \mathbb{N}$, $n \in \mathbb{N}$, als Atome enthalten. Abhängigkeitsalphabete (Σ, D), die sich nicht auf diese Weise zerlegen lassen, d.h. $\mathrm{Dec}(\Sigma, D) = \{(\Sigma, D)\}$, nennen wir *nuklear*. Wir nennen ein Abhängigkeitsalphabet *atomar*, wenn es nur einen Buchstaben besitzt.

Sei $\mathbb{T}(\Sigma, D)$ das $h_{(\Sigma,D)}$-Bild von $\mathbb{M}(\Sigma, D)$, also

$$\mathbb{T}(\Sigma, D) \quad = \quad h_{(\Sigma,D)}(\mathbb{M}(\Sigma, D)).$$

Dann ist die Menge $\mathbb{T}(\Sigma, D) \subseteq \mathrm{Term}(\Sigma, D)$ mit dem neutralen Element $h_{(\Sigma,D)}(1)$ und der wie folgt induktiv definierten Verknüpfung $\circ = \circ_{(\Sigma,D)}$ ein Monoid. Auch hier verzichten wir auf die Indizierung.

[6]Bei der Implementierung der Algorithmen hat sie sich jedoch als hilfreich erwiesen, da mit dieser Indizierung festgestellt werden kann, welche Rekursionsinvokation abgearbeitet werden muß.

(i) Sind $s = \times(s_1, \ldots, s_k)$ und $t = \times(t_1, \ldots, t_k)$, so gilt

$$s \circ t \quad = \quad \times(s_1 \circ t_1, \ldots, s_k \circ t_k).$$

(ii) Sind $s = *(s_1, \ldots, s_m)$ und $t = *(t_1, \ldots, t_n)$, dann gilt

$$s \circ t \quad = \quad *(s_1, \ldots s_{m-1}, s_m \circ t_1, t_2, \ldots, t_n),$$

falls $s_m, t_1 \in \mathbb{T}(\Sigma_i, D_i)$ für ein $i \in [k]$ und die Verknüpfung setzt sich rekursiv fort. Falls $s_m \in \mathbb{T}(\Sigma_i, D_i)$ und $t_1 \in \mathbb{T}(\Sigma_j, D_j)$ aus verschiedenen Monoiden $\mathbb{T}(\Sigma_i, D_i) \neq \mathbb{T}(\Sigma_j, D_j)$ stammen, d.h. $i \neq j$, so ist das Ergebnis der Konkatenation

$$s \circ t \quad = \quad *(s_1, \ldots, s_m, t_1, \ldots, t_n)$$

die einfache Verkettung der Argumente.

(iii) Anderenfalls ist (Σ, D) nuklear und wir definieren die Verknüpfung via die Monoidkonkatenation $s \circ t = st$.

Wir erhalten eine weitere Repräsentation für Spuren.

Theorem 13 *Die Abbildung $h : \mathbb{M}(\Sigma, D) \longrightarrow \mathbb{T}(\Sigma, D)$ ist ein injektiver Homomorphismus.*

Beweis: Sei $s, t \in \mathbb{M}(\Sigma, D)$ und $h = h_{(\Sigma,D)}$. Wir zeigen die Beziehung

$$h(s \cdot t) \quad = \quad h(s) \circ h(t)$$

mit Induktion. Ist (Σ, D) nuklear, so ist die Aussage trivial. Anderenfalls zerfällt (Σ, D) in eine Summe oder ein Komplexprodukt. Sei $\mathrm{Dec}(\Sigma, D) = \{(\Sigma_i, D_i) \mid i \in [k]\}$. Wir unterscheiden die Fälle $(\Sigma, D) = \oplus_{i\in[k]}(\Sigma_i, D_i)$ und $(\Sigma, D) = *_{i\in[k]}(\Sigma_i, D_i)$.

Ist $(\Sigma, D) = \oplus_{i\in[k]}(\Sigma_i, D_i)$, so gilt nach Lemma 23 die Isomorphie

$$\mathbb{M}(\Sigma, D) \quad \cong \quad \prod_{i\in[k]} \mathbb{M}(\Sigma_i, D_i).$$

Wir stellen ein Element $x \in \prod_{i\in[k]} \mathbb{M}(\Sigma_i, D_i)$ als Vektor $x = \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix}$ dar. Die Konkatenation und das h-Bild berechnen sich komponentenweise. Wir schreiben für $h_{(\Sigma_i,D_i)}$ kurz h_i. Damit ergibt sich mit Induktion

$$h(s \cdot t) \;=\; \begin{pmatrix} h_1(s_1 \cdot t_1) \\ \vdots \\ h_k(s_k \cdot t_k) \end{pmatrix} \;=\; \begin{pmatrix} h_1(s_1) \circ h_1(t_1) \\ \vdots \\ h_k(s_k) \circ h_k(t_k) \end{pmatrix} \;=\; h(s) \circ h(t).$$

Ist $(\Sigma, D) = *_{i\in[k]}(\Sigma_i, D_i)$ und $s = s_1 \cdot \ldots \cdot s_n$ und $t = t_1 \cdot \ldots \cdot t_m$ mit $s_i \in \mathbb{M}(\Sigma_{n_i}, D_{n_i})$ und $t_j \in \mathbb{M}(\Sigma_{m_j}, D_{m_j}) \setminus \{1\}$, $i \in [n]$, $j \in [m]$ und $n_i, m_j \in [k]$. Dann gilt $s_n \in \mathbb{M}(\Sigma_{n_n}, D_{n_n})$ und $t_1 \in \mathbb{M}(\Sigma_{m_1}, D_{m_1})$. Wir unterscheiden die zwei Fälle $n_n \neq m_1$ und $n_n = m_1$.

Ist $n_n \neq m_1$, so erhalten wir

$$\begin{aligned} h(s \cdot t) &= h(s_1 \cdot \ldots \cdot s_n \cdot t_1 \cdot \ldots \cdot t_m) \\ &= *(h_{n_1}(s_1), \ldots, h_{n_n}(s_n), h_{m_1}(t_1), \ldots, h_{m_m}(t_m)) \\ &= *(h_{n_1}(s_1), \ldots, h_{n_n}(s_n)) \circ *(h_{m_1}(t_1), \ldots, h_{m_m}(t_m)) \\ &= h(s) \circ h(t) \end{aligned}$$

Im Fall $n_n = m_1$ erhalten wir eine alternierende Sequenz der Länge $n + m - 1$

$$s \cdot t \quad = \quad s_1 \cdot \ldots \cdot s_{n-1} \cdot (\mathbf{s_n} \cdot_{\mathbb{M}(\Sigma_{\mathbf{m_1}}, \mathbf{D_{m_1}})} \mathbf{t_1}) \cdot t_2 \cdot \ldots \cdot t_m.$$

Die Multiplikation $s_n \cdot_{\mathbb{M}(\Sigma_{m_1}, D_{m_1})} t_1$ in dem Monoid $\mathbb{M}(\Sigma_{n_n}, D_{n_n}) = \mathbb{M}(\Sigma_{m_1}, D_{m_1})$ ist hervorgehoben, und das Resultat ist von 1 verschieden. Wir erhalten

$$\begin{aligned} h(s \cdot t) &= h(s_1 \cdot \ldots \cdot s_{n-1} \cdot (s_n \cdot t_1) \cdot t_2 \cdot \ldots \cdot t_m) \\ &= *(h_{n_1}(s_1), \ldots, h_{m_1}(s_n \cdot t_1), \ldots, h_{m_m}(t_m)) \\ &= *(h_{n_1}(s_1), \ldots, h_{n_n}(s_n) \circ h_{m_1}(t_1), h_{m_2}, \ldots, h_{m_m}(t_m)) \\ &= *(h_{n_1}(s_1), \ldots, h_{n_n}(s_n)) \circ *(h_{m_1}(t_1), \ldots, h_{m_m}(t_m)) \\ &= h(s) \circ h(t) \end{aligned}$$

mit Induktion über die Verbandsstruktur von $\mathrm{Dec}^*(\Sigma, D)$.

Die Injektivität beweist man analog. Sei $s \neq t$ und $s, t \in \mathbb{M}(\Sigma, D)$. Ist (Σ, D) die Summe $\oplus_{i \in [k]}(\Sigma_i, D_i)$, so existiert eine Komponente $i \in [k]$ mit $s_i \neq t_i$ und $h(s) \neq h(t)$ ergibt sich induktiv. Für $(\Sigma, D) = *_{i \in [k]}(\Sigma_i, D_i)$ sind entweder die alternierenden Sequenzen $s_1 \cdot \ldots \cdot s_n$ und $t_1 \cdot \ldots \cdot t_m$ verschieden lang, d.h. $n \neq m$, oder es existiert ein Index $i \in [n]$ mit $s_i \neq t_i$. In beiden Fällen ergibt sich die Behauptung $h(s) \neq h(t)$ induktiv. Die Induktionsverankerung ist trivial. ∎

7.2 Charakterisierung von Faktoren

Im nächsten Lemma charakterisieren wir Faktoren in Spuren aus zerfaserten Monoiden.

Lemma 24 *Sei $\{(\Sigma_i, D_i) \mid i \in [k]\}$ die Dekomposition des Abhängigkeitsalphabetes (Σ, D). Wir notieren $\mathbb{M}_i = \mathbb{M}(\Sigma_i, D_i)$, für $i \in [k]$. Weiter sei $t, l \in \mathbb{M}(\Sigma, D)$.*

(i) Im Fall $(\Sigma, D) = \oplus_{i \in [k]}(\Sigma_i, D_i)$ liegt t, l in $\prod_{i \in [k]} \mathbb{M}_i \cong \mathbb{M}(\Sigma, D)$ (direktes Produkt) und es gilt $t = ulv$ für $u, v \in \mathbb{M}(\Sigma, D)$ genau dann, wenn

- $\forall i \in [k] \ : \ \pi_i(t) = \pi_i(u)\pi_i(l)\pi_i(v),$

worin π_i den Projektionshomomorphismus

$$\mathbb{M}(\Sigma, D) \cong \prod_{i \in [k]} \mathbb{M}(\Sigma_i, D_i) \longrightarrow \mathbb{M}_i$$

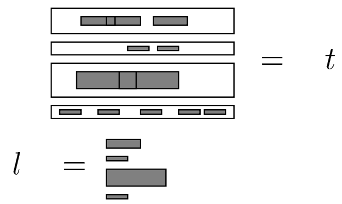
notiert.

(ii) Falls $(\Sigma, D) = *_{i\in[k]}(\Sigma_i, D_i)$ *liegt* t, l *in* $*_{i\in[k]}\mathbb{M}_i \cong \mathbb{M}(\Sigma, D)$ *(freies Produkt) mit den alternierenden Sequenzen* $t = t_1 \cdot \ldots \cdot t_n$ *und* $l = l_1 \cdot \ldots \cdot l_m$*. Dann gilt* $t = ulv$ *für* $u, v \in \mathbb{M}(\Sigma, D)$ *genau dann, wenn entweder*

- $\exists\, j \in \{0\} \cup [n-2] \;:\; t_{j+2} = l_2 \;\wedge \ldots \wedge\; t_{j+m-1} = l_{m-1}$*, und* $\exists\, x, y \;:\; xl_1 = t_{j+1} \;\wedge\; l_m y = t_{j+m}$*, oder*
- $\exists i \in [k] \;:\; l \in \mathbb{M}_i$ *mit* $t_j = xly$ *für* $x, y \in \mathbb{M}_i$ *und ein* $j \in [n]$*.*

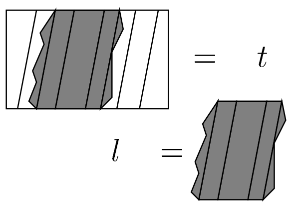
Beweis: Die Aussage ist eine unmittelbare Konsequenz aus der Termdarstellung, siehe Theorem 13 bzw. Lemma 23. Kombinatorisch kann sie direkt aus dem Lemma von Levi hergeleitet werden. ∎

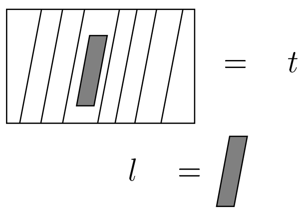
Die folgenden Abbildungen illustrieren die Situation am Abhängigkeitsgraphen. Im Falle eines direkten Produktes zerfällt der Abhängigkeitsgraph in Zusammenhangskomponenten. Tritt in den Komponenten jeweils eine Projektion von l als ein Faktor auf, so lassen sich die einzelnen Faktorprojektionen zu einem Faktor l rekombinieren.



Die Faktoren $l = (l_1, \ldots, l_k)$ sind in der Illustration grau hervorgehoben. Jedes Vorkommen $i : l_m \longrightarrow t$ der Projektionen l_m, $m \in [k]$ läßt sich mit jedem anderen Vorkommen $j : l_{m'} \longrightarrow t$, $m' \in [k]$ kombinieren[7].

Im Falle eines freien Produktes unterscheiden wir die beiden Möglichkeiten $n = 1$ und $n > 1$ mit $l = l_1 \cdot \ldots \cdot l_n$. Ist $n > 1$, so existiert in der Sequenz $t = t_1 \cdot \ldots \cdot t_m$ eine Position, ab der das Vorkommen beginnt und mit einem Teil der Sequenz wie in der Zeichnung angedeutet übereinstimmt. Ist $n = 1$, so existiert eine Position i, und t_i hat den Faktor l.





[7] In Notation $i_m \nabla j_{m'}$, d.h. die Relation ∇ ist ein Quader $I_1 \times \ldots \times I_k$, siehe Anhang A.

8 Auf Dekompositionen basierende Algorithmen

8.1 Dekomposition von Ersetzungssystemen

In diesem Abschnitt betrachten wir wie *Dekompositionen* von Ersetzungssystemen ererbt werden und wie sie bei dem Entwurf von Reduktionsalgorithmen genutzt werden können. Faktoren, die isomorph zu Elementen aus dieser Dekompositionsmenge sind, werden eine entscheidende Rolle einnehmen. Sei $\mathcal{R}$ ein endliches noethersches Spurersetzungssystem über einem Cographen-Monoid $\mathbb{M} = \mathbb{M}(\Sigma, D)$. Wir schränken uns hier auf Cographen-Monoide ein, um die komplizierteren Betrachtungen bei Rekursionsabbrüchen zu vermeiden.

Wieder bezeichnen wir mit $L_{\mathcal{R}}$ die Menge der linken Seiten $\{l \in \mathbb{M} \mid l \to r \in \mathcal{R}\}$. Für eine Spur $t \in \mathbb{M}$ bezeichnet

$$\begin{array}{rcll} \mathrm{Pref}(t) & = & \{\alpha \in \mathbb{M} \mid \alpha\beta = t\} & \textit{die Präfixe und} \\ \mathrm{Suff}(t) & = & \{\beta \in \mathbb{M} \mid \alpha\beta = t\} & \textit{die Suffixe.} \end{array}$$

Für Mengen $S \subseteq \mathbb{M}$ definieren wir

$$\mathrm{Pref}(S) = \bigcup_{s \in S} \mathrm{Pref}(s) \quad \text{und} \quad \mathrm{Suff}(S) = \bigcup_{s \in S} \mathrm{Suff}(s).$$

Die *Dekomposition* einer linken Seite $l \in L_{\mathcal{R}}$ ist durch $\mathrm{Dec}(l) = \mathrm{Dec}(h(l))$ via

$$\begin{array}{rcl} \mathrm{Dec}(\times(l_1, \ldots, l_k)) & = & \displaystyle\bigcup_{i \in [k]} \mathrm{Dec}(l_i) \cup \{(l_1, \ldots, l_k)\} \\ \mathrm{Dec}(*(l_1, \ldots, l_n)) & = & \begin{cases} \{l_1 \cdot \ldots \cdot l_n\} & \text{falls } n > 1 \\ \mathrm{Dec}(l_1) \cup \{l_1\} & \text{falls } n = 1 \end{cases} \end{array}$$

definiert. In der obigen Definition fassen wir l_1 in der letzten Zeile als Subterm auf und berechnen die Dekomposition rekursiv. Läßt sich keine der obigen Gleichungen anwenden, so ist l ein Element eines nuklearen Monoids. In diesem Fall setzen wir $\mathrm{Dec}(l) = \{l\}$.

Für ein Ersetzungssystem $\mathcal{R}$ legen wir die folgenden Mengen fest:

$$\begin{array}{rcll} \mathrm{Dec}(\mathcal{R}) & = & \bigcup\{\mathrm{Dec}(l) \mid l \in L_{\mathcal{R}}\} & \textit{die Dekomposition,} \\ \mathrm{Pref}(\mathcal{R}) & = & \mathrm{Pref}(L_{\mathcal{R}}) & \textit{die Präfixe und} \\ \mathrm{Suff}(\mathcal{R}) & = & \mathrm{Suff}(L_{\mathcal{R}}) & \textit{die Suffixe.} \end{array}$$

8.2 Term-Beschriftung

Sei $t \in \mathbb{M}$ eine Spur. Dann beschriften wir den Term $h(t)$ mit drei Mengen. Die ersten beiden Beschriftungen $P(h(t))$ und $S(h(t))$ sind die Mengen

$$P(h(t)) = \mathrm{Suff}(t) \cap \mathrm{Pref}(\mathcal{R}) \text{ und } S(h(t)) = \mathrm{Pref}(t) \cap \mathrm{Suff}(\mathcal{R}).$$

Die dritte Beschriftung $L(h(t))$ beschreibt die Positionen, an denen potentielle Faktorvorkommen von $d \in \mathrm{Dec}(\mathcal{R})$ gefunden werden können so, daß dort eine Ersetzung möglich wird. Für freie Produkte soll diese Beschriftung auf die Stellen in der alternierenden Sequenz hinweisen, an denen eine Spur aus $\mathrm{Dec}(\mathcal{R})$ beginnt. Bei direkten Produkten soll lediglich bekannt sein, ob sich Subfaktoren von einer Spur aus $\mathrm{Dec}(\mathcal{R})$ zu einem Faktor kombinieren lassen. Formal definieren wir

$$\begin{aligned} L(\times(t_1, \ldots, t_k)) &= \bigcup_{i \in [k]} \{d \in \mathbb{M} \mid \forall i \in [k] : \pi_i(d) \neq 1 \ \Rightarrow\ L_{d_i}(t_i) \neq \emptyset\} \\ L(*(t_1, \ldots, t_n)) &= \Big(L_d(h(t))\Big)_{d \in \mathrm{Dec}(\mathcal{R})}, \end{aligned}$$

wobei $L_d(t) \subseteq [n]$ die (geordnete) Menge natürlicher Zahlen $\{i_1, \ldots, i_{n_d}\}$ mit der Eigenschaft

$$i_j \in L_d \begin{cases} \bullet \text{ falls } h(d) = *(d_1, \ldots, d_m),\ m > 1 \text{ und} \\ h(t) = *(t_1, \ldots, t_{i_j - 1}, u \circ d_1, d_2, \ldots, d_{m-1}, d_m \circ v, t_{i_j + m} \ldots, t_n) \\ \bullet \text{ oder falls } h(d) = *(d_1) \text{ und} \\ h(t) = *(t_1, \ldots, t_{i_j - 1}, u \circ d_1 \circ v, t_{i_j + 1}, \ldots, t_n) \end{cases}$$

für $u, v \in h(\mathbb{M})$. Im ersten Fall beschreibt die Position i_j ein Faktorvorkommen $d_1 \cdot \ldots \cdot d_m \longrightarrow t$, so daß ein echtes Präfix d_1 existiert, das gleichzeitig ein Suffix des Subterms t_{i_j} ist. Im zweiten Fall wird eine nicht-leere Menge von Faktorvorkommen $d_1 \longrightarrow t$ durch i_j beschrieben, die in dem Subterm t_{i_j} vorkommen.

Ist das Monoid nuklear, d.h. $t = h(t)$, so beschriften wir den Term mit der Menge

$$L(t) \quad = \quad \{i : d \longrightarrow t \mid d \in \mathrm{Dec}(\mathcal{R})\}$$

aller Faktorvorkommen. Da wir nur Cographen-Monoide betrachten, erhalten wir als nukleare Terme nur atomare. D.h. wir erhalten den Spezialfall $(\Sigma, D) = (\{a\}, \{(a, a)\})$, also $\mathbb{M}(\Sigma, D) = \mathbb{M} = \{a\}^* \cong \mathbb{N}^{\{a\}}$. Das bedeutet die Menge der Faktorvorkommen $a^m \longrightarrow a^n$ läßt sich durch

$$L(a^n) \quad = \quad \{i \in \mathbb{N} \mid 1 \leq i \leq n - m + 1\} \quad = \quad [n - m + 1]$$

repräsentieren.

Mit Lemma 24 existiert für $h(t) = *(t_1, \ldots, t_n)$ eine Faktorisierung $t = x \cdot d \cdot y$ mit $d \in \mathrm{Dec}(\mathcal{R})$ genau dann, wenn ein Index i in der Beschriftung $L_d(h(t))$ vorhanden ist. Für $h(t) = \times(t_1, \ldots, t_k)$ gilt $d \in L(h(t))$ genau dann, wenn $x, y \in \mathbb{M}$ existieren mit $t = x \cdot d \cdot y$.

Wir verwenden nicht die Menge aller Kombinationen

$$\left\{(i_1, \ldots, i_k) \in [n_1] \times \ldots \times [n_k] \;\middle|\; \forall j \in [k] : \pi_j(d) \neq 1 \implies i_j \in L_{t_j}\right\},$$

da diese im allgemeinen von quadratischer Ordnung ist. Beispielsweise besitzt für $\mathbb{M} = \{a\}^* \times \{b\}^*$, $t = \begin{pmatrix} a^n \\ b^n \end{pmatrix}$ und $d = \begin{pmatrix} a \\ b \end{pmatrix}$ die Menge aller möglichen Kombinationen von a- und b-Faktoren, $[n] \times [n]$, die Kardinalität n^2, wohingegen die Anzahl der Elemente in der Beschriftung $|L(\times(t_1, \ldots, t_k))| \leq |\{d_2 \mid d_1 d_2 d_3 \in \mathrm{Dec}(\mathcal{R})\}| = \mathcal{O}(1)$ von kleiner Ordnung ist. Für $*$-Terme gilt $|L_d(*(t_1, \ldots, t_n))| \leq n$ für alle $d \in \mathrm{Dec}(\mathcal{R})$.

Wir nennen einen Term $h(t) = f(s_1, \ldots, s_n)$, $f \in \{\times, *\}$, *korrekt beschriftet*, wenn alle Subterme $s_1, \ldots, s_n$ korrekt beschriftet sind und der Term selbst mit $P(t)$, $S(t)$ und $L(t)$ beschriftet ist.

8.3 Der abstrakte Datentyp

Wir repräsentieren eine Spur $t \in \mathbb{M}$ als einen knotenbeschrifteten, dem Term $h(t)$ entsprechenden Baum[8]. Die Knoten dieses Baums sind mit den Beschriftungen $P(s)$, $S(s)$ und $L(s)$ für die repräsentierten Subterme s von $h(t)$ versehen.

Für die Datenstruktur definieren wir die Operationen SPLIT, PROJECT, COMPOSE und CONCAT so, daß die Konsistenz der Beschriftung invariant unter Anwendung dieser Operationen ist, und daß die Resultate wieder Bäume mit derselben Struktur sind. Diese Operationen ermöglichen ein Zerlegen und Zusammenfügen von Spuren in dieser Repräsentation.

Diese Vorgehensweise ermöglicht es schließlich einen rekursiven Algorithmus zur Faktorisierung bzw. Normalformenberechnung zu entwerfen. Die Primitiven besitzen die folgende Semantik.

(i) SPLIT$(i, *(t_1, \ldots, t_n))$ — faktorisiert den $*$-Term $*(t_1, \ldots, t_n)$ an der Stelle $i \in [n]$ in zwei $*$-Terme $*(t_1, \ldots, t_{i-1})$ und $*(t_i, \ldots, t_n)$. Die Katenation der Resultatterme ist wieder der ursprüngliche Term.

(ii) PROJECT$(i, \times(t_1, \ldots, t_k))$ — extrahiert den Term t_i, $i \in [k]$.

(iii) COMPOSE$(t_1, \ldots, t_k)$ — rekombiniert die Terme t_i der Spuren $t'_i = h^{-1}(t_i) \in \mathbb{M}_i$, $i \in [k]$ so, daß der Resultatterm $\times(t_1, \ldots, t_k)$ die Spur $(t'_1, \ldots, t'_k) \in \prod_{i \in [k]} \mathbb{M}_i$ darstellt. Die Anzahl der Argumente hängt nur vom zugrundeliegenden induzierten Abhängigkeitsalphabet ab. Da wir nicht-uniforme Algorithmen untersuchen ist diese Anzahl durch eine konstante beschränkt. Es gilt $k \in \mathcal{O}(|(\Sigma|) = \mathcal{O}(1)$. Die Terme t_i sind $*$-Terme oder Atome, und es sollen nur solche Invokationen auftreten, so daß der berechnete Term wieder ein h-Bild ist.

[8]Ein geordneter Baum stellt einen Term dar, wenn die Blätter die nullstelligen Symbole sind und die inneren Knoten (mit den entsprechenden Funktionssymbolen markiert) den Funktionssymbolen von nicht-atomaren Subtermen entsprechen.

(iv) CONCAT(t_1, t_2) — berechnet die Katenation $t_1 \circ t_2$ der Argumentterme.

Zur Komplexitätsanalyse der unten vorgestellten Algorithmen benötigen wir noch das folgende Lemma.

Lemma 25 *Seien $t, l \in \mathbb{M}(\Sigma, D)$ Spuren, die durch den Baum $h(t)$, bzw. $h(l)$ repräsentiert sind. Atomare Terme a^n seien durch Strings repräsentiert. Dann können die Prädikate*

$$\textit{(i)}\ l \in \mathrm{Pref}(t),\ \textit{(ii)}\ l \in \mathrm{Suff}(t)\ \textit{und}\ \textit{(iii)}\ l = t$$

in $\mathcal{O}(\min(|t|, |l|))$ Zeiteinheiten entschieden werden.

Beweis: *(iii)* folgt aus *(i)* und *(ii)* mit $t = l \iff (l \in \mathrm{Pref}(t)) \wedge (t \in \mathrm{Pref}(l))$.

```
PREDICATE PREF(l, t)
  Comment Input: two traces in representation.
          Output: boolean (l ∈ Pref(t)).
  Method
     if h(t) = *(t_1, ..., t_m) and h(l) = *(l_1, ..., l_n) then
       if m < n then return (false)
         else for all 1 ≤ i ≤ n − 1 do
                 if t_i ≠ l_i then return (false) end_if
              end_for
              return(PREF(l_n, t_n))
       end_if
     else if h(t) = ×(t_1, ..., t_m) and h(l) = ×(l_1, ..., l_n) then
       if m ≠ n then error
         else for all (1 ≤ i ≤ m) do
                 if not PREF(l_i, t_i) then return (false) end_if
              end_for
              return(true)
       end_if
     else if t, l ∈ a* are basic representations then
       if |t| < |l| then return (false)
                else return (true)
       end_if
     otherwise error
  end_Method
```

Mit der Charakterisierung in Lemma 24 folgt, daß PREF$(t, l) = \mathit{true}$ genau dann, wenn $l \in \mathrm{Pref}(t)$. Genauer gilt für $t = *(t_1, \ldots, t_m)$ und $l = *(l_1, \ldots, l_n)$

$$t = l \circ u \quad\iff\quad \forall i \in [n-1] : t_i = l_i \ \wedge\ d_n \in \mathrm{Pref}(t_n),$$

und für $t = \times(t_1, \ldots, t_m)$ und $l = \times(l_1, \ldots, l_n)$

$$t = l \circ u \quad \Longleftrightarrow \quad m = n \ \wedge \ \forall i \in [m] : l_i \in \mathrm{Pref}(t_i).$$

Der Fall $l, t \in a^*$ ist offensichtlich korrekt.

Die Komplexität ist $\mathcal{O}(\min(|l|, |t|))$, da bei jeder Invokation beide Argumentstrukturen — aufsummiert über alle rekursiven Aufrufe — entweder sich um mindestens ein Funktionssymbol verkleinern oder das Resultat nach dem Lesen der kleineren der beiden Strukturen feststeht.

Sei $T(l,t)$ der Zeitaufwand zum Entscheiden der drei Prädikate. Dann gilt für die Präfixberechnung

$$\begin{array}{rcl} \left.\begin{array}{r} T(*(t_1, \ldots, t_m), *(l_1, \ldots, l_n)) \\ T(\times(t_1, \ldots, t_m), \times(l_1, \ldots, l_n)) \end{array}\right\} & = & \mathcal{O}(1) + \displaystyle\sum_{i \le \min(m,n)} T(t_i, l_i) \\ T(at, al) & = & \mathcal{O}(1) + T(l, t), \text{ wenn } l, t \in a^*. \end{array}$$

Da $\sum_{i \in [m]} |h^{-1}(t_i)| = |t|$ für alle t mit $h(t) = *(t_1, \ldots, t_m)$ oder $h(t) = \times(t_1, \ldots, t_m)$ gilt, folgt *(i)* und dual *(ii)* mit Induktion. ∎

Bemerkung 4 *Dieses Resultat gilt nicht nur für Spuren aus Cographen-Monoiden. Für alle Darstellungen gilt die rekursive Beziehung*

$$al \in \mathrm{Pref}(at) \quad \Longleftrightarrow \quad l \in \mathrm{Pref}(t),$$

die für nukleare Terme verwendet werden kann. Minimale Buchstaben a in beispielsweise einem Hasse-Diagramm oder die diese Buchstaben repräsentierenden Buchstaben in den Wörtern $t_C = at'_C$, $a \in C \in \mathcal{C}$, eines Worttupels $(t_C)_{C \in \mathcal{C}}$ lassen sich in $\mathcal{O}(|\Sigma|) = \mathcal{O}(1)$ finden. Diese $t \in \mathbb{M}$ repräsentierenden Strukturen besitzen maximal $\mathcal{O}(|\Sigma| \cdot |t|) = \mathcal{O}(|t|)$ Knoten bzw. Kanten bzw. Funktionssymbole.

8.3.1 Algorithmische Primitive

Es folgen die Beschreibungen der Methoden zu den oben aufgeführten Operationen. Die Algorithmen werden im Laufe einer Berechnung nur strukturverträgliche Terme, d.h. Terme $h_{(\Delta, D_\Delta)}(t)$ und $h_{(\Sigma, D_\Sigma)}(t)$ mit $(\Sigma, D_\Sigma) = (\Delta, D_\Delta)$, so verknüpfen, daß die resultierende Terme wieder wohlstrukturiert sind.

SPLIT

Die zulässigen Eingaben von SPLIT sind korrekt beschriftete Terme $*(t_1, \ldots, t_n)$ und Zahlen $i \in [n+1]$. Das Resultat ist das Termpaar $\langle *(t_1, \ldots, t_{i-1}), *(t_i, \ldots, t_n) \rangle$ mit korrekter Beschriftung.

FUNCTION SPLIT$(i, *(t_1, \ldots, t_i, \ldots, t_n))$

Comment Input: a natural number and a trace representation.
Output: the representation of two traces
according to the semantic.

Method

if $(1 < i \leq n)$ **then** $\langle t'_1, t'_2 \rangle \leftarrow \langle *(t_1, \ldots, t_{i-1}), *(t_i, \ldots, t_n) \rangle$

$P(t'_1) \leftarrow P(t_{i-1}) \cup \{l_1 \cdot \ldots \cdot l_j \in \mathrm{Pref}(\mathcal{R}) \mid t_1 \cdot \ldots \cdot t_{i-1} = \alpha l_1 \cdot \ldots \cdot l_j, j > 1\}$

$P(t'_2) \leftarrow P(t_n) \cup \{l_1 \cdot \ldots \cdot l_j \in \mathrm{Pref}(\mathcal{R}) \mid t_i \cdot \ldots \cdot t_n = \alpha l_1 \cdot \ldots \cdot l_j, j > 1\}$

$S(t'_1) \leftarrow S(t_1) \cup \{l_1 \cdot \ldots \cdot l_j \in \mathrm{Suff}(\mathcal{R}) \mid t_1 \cdot \ldots \cdot t_{i-1} = l_1 \cdot \ldots \cdot l_j \beta, j > 1\}$

$S(t'_2) \leftarrow S(t_i) \cup \{l_1 \cdot \ldots \cdot l_j \in \mathrm{Suff}(\mathcal{R}) \mid t_i \cdot \ldots \cdot t_n = l_1 \cdot \ldots \cdot l_j \beta, j > 1\}$

for all $d \in \mathrm{Dec}(\mathcal{R})$ **do**

$$\begin{aligned} L_d(t'_1) \leftarrow (L_d(t) \cap [i-1]) - \{i-j-1 \mid\ & d_1 \in P(t'_1),\ d_2 \in S(t'_2), \\ & d_1 \neq 1 \neq d_2 \\ & h(d_1) = *(d'_1, \ldots, d'_j), \\ & \text{with } h = h_{\langle \Sigma, D \rangle (\Sigma(t))} \\ & d_1 d_2 = d\} \end{aligned}$$

$L_d(t'_2) \leftarrow L_d(t) \cap \{i, \ldots, n\}$

end_for

return ($\langle t'_1, t'_2 \rangle$ with labeling S, P, L, resp.)

else if $i = 1$ **then return**$(\langle 1, *(t_1, \ldots, t_n) \rangle)$ **end_if**

else if $i = n + 1$ **then return**$(\langle *(t_1, \ldots, t_n), 1 \rangle)$ **end_if**
with empty labeling ($S = P = \emptyset$) for 1 and
the old labeling for $*(t_1, \ldots, t_n)$

otherwise error

end_Method

Durch die Zuweisungen

$$\begin{aligned} P(t'_1) &\leftarrow P(t_{i-1}) \cup \{l_1 \cdot \ldots \cdot l_j \in \mathrm{Pref}(\mathcal{R}) \mid t_1 \cdot \ldots \cdot t_{i-1} = \alpha l_1 \cdot \ldots \cdot l_j, j > 1\} \\ P(t'_2) &\leftarrow P(t_n) \cup \{l_1 \cdot \ldots \cdot l_j \in \mathrm{Pref}(\mathcal{R}) \mid t_i \cdot \ldots \cdot t_n = \alpha l_1 \cdot \ldots \cdot l_j, j > 1\} \\ S(t'_1) &\leftarrow S(t_1) \cup \{l_1 \cdot \ldots \cdot l_j \in \mathrm{Suff}(\mathcal{R}) \mid t_1 \cdot \ldots \cdot t_{i-1} = l_1 \cdot \ldots \cdot l_j \beta, j > 1\} \\ S(t'_2) &\leftarrow S(t_i) \cup \{l_1 \cdot \ldots \cdot l_j \in \mathrm{Suff}(\mathcal{R}) \mid t_i \cdot \ldots \cdot t_n = l_1 \cdot \ldots \cdot l_j \beta, j > 1\} \end{aligned}$$

werden die Präfix- und Suffixbeschriftungen für beide Resultatterme berechnet. Dabei wird die rekursive Struktur, die auch im Entscheidungsalgorithmus PREF verwendet wird, genutzt. In der for-Schleife werden anschließend die L-Beschriftungsfamilien ermittelt, indem die Listen L_l in Teile gespalten werden, die zu den resultierenden Termen korrespondieren, und die Markierungen, die in dem ersten Term beginnen und im zweiten enden, entfernt werden. Offensichtlich gilt folgendes Lemma.

Lemma 26 *Sei $t = *(t_1, \ldots, t_n)$ ein korrekt beschrifteter Term und $i \in [n+1]$. Dann ist das Resultat SPLIT(i, t) ein Paar (t', t'') von korrekt beschrifteten Termen mit $t' = *(t_1, \ldots, t_{i-1})$ und $t'' = *(t_i, \ldots, t_n)$.* ■

Die Beschriftungen P und S können in $\mathcal{O}(\sum_{l \in \mathrm{Dec}(\mathcal{R})} |l|) = \mathcal{O}(1)$ mit Lemma 25 bruteforce berechnet werden. Der Algorithmus nützt jedoch die hierarchische Struktur der Dekompositionen, um Mehrfachberechnungen zu vermeiden.

Der Rumpf in der for-Schleife zur Berechnung der L-Beschriftungen kann ohne weitere Datenstrukturen nicht in konstanter Zeit bearbeitet werden. Das Aufspalten der (geordneten) Mengen erfordert bei einer Implementierung durch beispielsweise eine Familie von Listen im worst-case $\mathcal{O}(n)$ Zeiteinheiten. Eine Effizienzsteigerung erreicht man durch Verwenden der in Kapitel 5.2 beschriebenen C^*-Baume mit $C = \mathbb{N}$. D.h. die Mengen L_l werden durch Bäume dargestellt, deren Blätter zu Positionen $i \in \mathbb{N}$ korrespondieren. Jede der Mengen L_d läßt sich dann in $\mathcal{O}(\log |L_d|) = \mathcal{O}(\log n)$ Zeit mit der Splitoperation für C^*-Bäume separieren, woraus die (nicht-konstante) Laufzeit $\mathcal{O}(\log n)$ resultiert. Wir erhalten

Lemma 27 *Sei $t = *(t_1, \ldots, t_n)$ ein korrekt beschrifteter Term und $i \in [n+1]$. Dann terminiert die effiziente Implementierung nach Aufruf mit SPLIT(i, t) in $\mathcal{O}(\log n)$ Zeiteinheiten.* ∎

Man beachte, daß das Argument i als Zeiger interpretiert werden kann. Dann entfällt die logarithmische Suche in der komplexen Zuweisung

$$\langle t_1', t_2' \rangle \leftarrow \langle *(t_1, \ldots, t_{i-1}), *(t_i, \ldots, t_n) \rangle,$$

aber die Partitionierungen der Mengen (Listen)

$$\begin{aligned} L_d(t_1') &\leftarrow (L_d(t) \cap [i-1]) - \{i - j \mid \; && d_1 \in P(t_1'), d_2 \in S(t_2'), \\ & && d_1 \neq 1 \neq d_2, \\ & && h_{\langle \Sigma, D \rangle (\Sigma(t))}(d_1) = *(d_1', \ldots, d_j'), \\ & && d_1 d_2 = d\} \\ L_d(t_2') &\leftarrow L_d(t) \cap \{i, \ldots, n\} \end{aligned}$$

benötigen ohne zusätzliche Information logarithmische Zeit mit $\mathbb{N}$-Bäumen. Man beachte, daß der Homomorphismus $h = h_{\langle \Sigma, D \rangle (\Sigma(t))}$ derselbe wie bei $h(t) = *(t_1, \ldots, t_n)$ ist. Eine Verbesserung der Konstanten erhält man, indem nur verträgliche Faktorisierungen von $d = *(d_1', \ldots, d_n')$ betrachtet werden. Das sind die $n-1$ Faktorisierungen in

$$\Big\{ \big(*(d_1'), *(d_2', \ldots, d_n') \big), \; \big(*(d_1', d_2'), *(d_3', \ldots, d_n') \big), \ldots, \big(*(d_1', \ldots, d_{n-1}'), *(d_n') \big) \Big\}.$$

Durch Verzichten auf die Operation SPLIT werden wir später einen Linearzeitalgorithmus angeben können.

PROJECT

Die Methode PROJECT ergibt sich aus der rekursiven Definition der Beschriftung.

FUNCTION PROJECT$(i, \times(t_1, \ldots, t_i, \ldots, t_k))$
  Comment Input: a natural number and a trace representation.
    Output: the representation of a trace
      according to the semantic.

Method
  if $(1 \le i \le k)$ **then return**(t_i)
    otherwise error
end_Method

Offensichtlich benötigt die Methode $\mathcal{O}(k) = \mathcal{O}(1)$ Zeiteinheiten, da wir das Abhängigkeitsalphabet (Σ, D) als fest annehmen. Wir erhalten

Lemma 28 *Sei $t = \times(t_1, \ldots, t_k)$ ein korrekt beschrifteter Term und $i \in [k]$. Dann ist das Resultat von PROJECT(i,t) der korrekt beschriftete Term $h(\pi_i(h^{-1}(t)))$ wobei π_i die i-te Projektion $\mathbb{M} \cong \prod_{j\in[k]} \mathbb{M}_j \longrightarrow M_i$ ist.* ■

COMPOSE

Invers zu PROJECT fügt COMPOSE Terme zu einem Produktterm zusammen.

FUNCTION COMPOSE$(t_1, \ldots, t_k)$
  Comment Input: a tupel of trace representations.
    Output: the representation of a trace according
      to the semantic.
  Method
    if $\times(t_1, \ldots, t_k)$ is a well formed term
      w.r.t. the corresponding submonoid
      then $t \leftarrow \times(t_1, \ldots, t_k)$
        $S(t) \leftarrow \prod_{1\le i\le k} S(t_i) \cap \mathrm{Suff}(\mathcal{R})$
        $P(t) \leftarrow \prod_{1\le i\le k} P(t_i) \cap \mathrm{Pref}(\mathcal{R})$
        $L(t) \leftarrow \emptyset$
        for all $d \in \mathrm{Dec}(\mathcal{R})$ **do**
          let $h(d) = \times(d_1, \ldots, d_k)$ a term corresponding to the
          structure of t
          $B \leftarrow$true
          for all $1 \le i \le k$ **do** $B \leftarrow B \wedge (L_{d_i} \neq \emptyset)$ **end_for**
          if B **then** $L(t) \leftarrow L(t) \cup \{d\}$ **end_if**
        end_for
        return(t with labeling S, P, L)
      otherwise error
  end_Method

Die Semantik der zwei komplexen Zuweisungen

$$\begin{aligned} S(t) &\leftarrow \prod_{1\le i\le k} S(t_i) \cap \mathrm{Suff}(\mathcal{R}) \\ P(t) &\leftarrow \prod_{1\le i\le k} P(t_i) \cap \mathrm{Suff}(\mathcal{R}) \end{aligned}$$

ist eine komponentenweise Analyse der Kombination der Prä- bzw. Suffixvorkommen. Die Korrektheit folgt wieder aus Lemma 24. Für Präfixe gilt bei strukturgleichen Termen

$$\times(d_1, \ldots, d_k) \in \mathrm{Pref}(\times(t_1, \ldots, t_k)) \iff \forall i \in [k] : d_i \in \mathrm{Pref}(t_i).$$

Auch hier ist eine brute-force Berechnung möglich, verschlechtert aber die Konstanten. Für Suffixe gilt Duales. Die Korrektheit der L-Beschriftung folgt aus Lemma 24

$$\begin{aligned} \times(t_1, \ldots, t_k) &= u \circ \times(d_1, \ldots, d_k) \circ v \\ \iff \quad & \forall i \in [k] : h(\pi_i(h^{-1}(u))) \circ d_i \circ h(\pi_i(h^{-1}(v))) \end{aligned}$$

mit Induktion. Man beachte, die induzierten Abhängigkeitsalphabete sind paarweise disjunkt. Wir erhalten

Lemma 29 *Seien t_i korrekt beschriftete Spurterme mit den induzierten paarweise disjunkten Abhängigkeitsalphabeten $(\Sigma_i, D_i) = \langle \Sigma, D\rangle(\Sigma(t_i))$, $i \in [k]$. Ist $\oplus_{i\in[k]}(\Sigma_i, D_i) \in$ $\mathrm{Dec}^*(\Sigma, D)$, dann berechnet COMPOSE$(t_1, \ldots, t_k)$ den Term $\times(t_1, \ldots, t_k)$ mit korrekter Beschriftung.* ■

Auch diese Methode benötigt

$$\mathcal{O}(|\Sigma| \cdot (|\mathrm{Dec}(\mathcal{R})| + |\mathrm{Suff}(\mathcal{R})| + |\mathrm{Pref}(\mathcal{R})|)) = \mathcal{O}(1)$$

Zeit.

CONCAT

Unter Verwendung der bereits vorgestellten Methoden können wir einen Algorithmus für die Katenation ableiten. Wir verfahren dabei wie in der Definition von $\circ$ im vorigem Kapitel.

FUNCTION CONCAT(t_1, t_2)
  Comment Input: two trace representations.
            Output: the representation of the catenation.
  Method
    if $t_1 = \times(t'_1, \ldots, t'_k)$ **and** $t_2 = \times(t''_1, \ldots, t''_k)$
      then for all $(1 \leq i \leq k)$ **do**
            $\langle t_i^1, t_i^2\rangle \leftarrow \langle$PROJECT$(i, t_1)$, PROJECT$(i, t_2)\rangle$
            $t_i''' \leftarrow$CONCAT(t_i^1, t_i^2)
          end_for
          return (COMPOSE$(t_1''', \ldots, t_k''')$)
    else if $t_1 = *(t'_1, \ldots, t'_{n_1})$ **and** $t_2 = *(t''_1, \ldots, t''_{n_2})$ **then**
        if $t'_{n_1} \in \mathbb{M}_i \neq \mathbb{M}_j \ni t''_1$, where $\mathbb{M} = *_{i\in[k]}\mathbb{M}_i$
          then $s \leftarrow *(t'_1, \ldots, t'_{n_1}, t''_1, \ldots, t''_{n_2})$
              let $s = *(s_1, \ldots, s_n)$
              $P(s) \leftarrow P(s_n) \cup \{l_1 \cdot \ldots \cdot l_j \in \mathrm{Pref}(\mathcal{R}) \mid$

$s_1 \cdot \ldots \cdot s_n = \alpha l_1 \cdot \ldots \cdot l_j, j > 1\}$
$S(s) \leftarrow S(s_1) \cup \{l_1 \cdot \ldots \cdot l_j \in \mathrm{Suff}(\mathcal{R}) \mid$
$s_1 \cdot \ldots \cdot s_n = l_1 \cdot \ldots \cdot l_j \beta, j > 1\}$
for all $d \in \mathrm{Dec}(\mathcal{R})$ **do**
$L_d(s) \leftarrow L_d(t_1) \cup L_d(t_2) \cup$
$\{n_1 - i \mid h_{\langle\Sigma,D\rangle(\Sigma(s))}(d) = *(d_1, \ldots, d_m), m > 1$
$*(d_1, \ldots, d_i) \in P(t_1),$
$*(d_{i+1}, \ldots, d_m) \in S(t_2)\}$
end_for
else $\langle s_1', s_2' \rangle \leftarrow$SPLIT$(n_1, t_1)$
$\langle s_1'', s_2'' \rangle \leftarrow$SPLIT$(2, t_2)$
$s \leftarrow$CONCAT$(s_1'$,CONCAT(CONCAT$(s_2', s_1''), s_2''))$
end_if
return s with labeling P, S, L
else if $t_1 \in a^*$ and $t_2 \in a^*$
then let $t_1 = a^n$ and $t_2 = a^m$
$t \leftarrow a^{n+m}$
for all $d \in \mathrm{Dec}(\mathcal{R}) \cap a^*$ **do**
$L_d \leftarrow [n + m + 1 - |d|]$
end_for
end_if
return s with labeling P, S, L
otherwise error
end_Method

Sind die zwei Eingabeparameter korrekt beschriftete Spurterme der Form $\times(x_1, \ldots, x_k)$, so führen wir die Konkatenation auf k Konkatenationen der durch PROJECT berechneten Subterme zurück. (Diese sind natürlich nur definiert, falls $t_1 = h_{(\Sigma,D)}(\tilde{t}_1)$ und $t_2 = h_{(\Sigma,D)}(\tilde{t}_2)$ dieselbe Struktur besitzen. Die Resultate werden anschließend mit COMPOSE zu einem korrekt beschrifteten Spurterm zusammengefügt. Besitzen die Terme die Form $*(x_1, \ldots)$, so wird entsprechend der Definition der Konkatenation $\circ$ verfahren. Der rekursive Fall wird entsprechend der Definition unter Verwendung der Operationen SPLIT und CONCAT (für ein Monoid, das von einem Abhängigkeitsalphabet, das sich unter (Σ, D) bzgl. der Verbandsordnung in $\mathrm{Dec}^*(\Sigma, D)$ befindet) bearbeitet. Der verbleibende Fall wird durch Fusionieren der beiden Bäume an der Wurzel bearbeitet. Ist das Abhängigkeitsalphabet (Σ, D) atomar, so verwendet man die Konkatenationsoperation auf der Wortrepräsentation. D.h. die beiden Strings a^n und a^m müssen konkateniert werden. Existiert ein $d \in \mathrm{Dec}(\mathcal{R})$, das vollständig in einem atomaren Submonoid a^* liegt, so muß die L-Beschriftung durch die Menge $[n+m+1-|d|]$ bzw. die diese Menge repräsentierende Zahl $n + m + 1 - |d|$ aktualisiert werden.

Lemma 30 *Seien t_1 und t_2 zwei korrekt beschriftete Spurterme aus $h(\mathbb{M})$. Dann berechnet die Funktion CONCAT den korrekt beschrifteten Spurterm $t_1 \circ t_2$.* ■

Man beachte COMPOSE, PROJECT, SPLIT und auch mit Induktion CONCAT berechnen korrekt beschriftete Terme.

Lemma 31 *Seien t_1 und t_2 korrekt beschriftete Terme. Dann terminiert der Algorithmus CONCAT in $\mathcal{O}(\log|t_1| + \log|t_2| + 1)$ Zeiteinheiten.*

Beweis: Sei $\mathrm{Dec}(\Sigma, D) = \{(\Sigma_i, D_i) \mid i \in [k]\}$. Im Fall $t_1 = \times(t'_1, \ldots, t'_k)$ und $t_2 = \times(t''_1, \ldots, t''_k)$ erhalten wir für die Komplexität $T_{\mathrm{CONCAT}}(t_1, t_2)$

$$\begin{aligned}
&\mathcal{O}\Bigg(\sum_{i\in[k]} \Big(T_{\mathrm{CONCAT}}(t'_i, t''_i) + T_{\mathrm{PROJECT}}(t_1) + T_{\mathrm{PROJECT}}(t_2)\Big) + \\
&\qquad T_{\mathrm{COMPOSE}}(t'''_1, \ldots, t'''_k) + 1\Bigg) \\
= \quad & \mathcal{O}(\log|t_1| + \log|t_2| + 1)
\end{aligned}$$

Für $t_1 = *(t'_1, \ldots, t'_{n_1})$ und $t_2 = *(t''_1, \ldots, t''_{n_2})$ erhalten wir

$$\begin{aligned}
T_{\mathrm{CONCAT}}(t_1, t_2) &= \mathcal{O}(1 + T_{\mathrm{SPLIT}}(t_1) + T_{\mathrm{SPLIT}}(t_2) + T_{\mathrm{CONCAT}}(t'_{n_1}, t''_1)) \\
&= \mathcal{O}(\log|t_1| + \log|t_2| + 1)
\end{aligned}$$

mit der effizienteren Implementierung von SPLIT. Für die Basisfälle gilt in dieser Repräsentation die Beziehung

$$T_{\mathrm{CONCAT}}(t_1, t_2) = \mathcal{O}(|\Sigma| + |\mathrm{Dec}(\mathcal{R})|) = \mathcal{O}(1).$$

■

8.3.2 Quotientenberechnung

Sei $t = uv \in \mathbb{M}$. Dann bezeichnen wir $u^{-1}t = v$ als *Linksquotienten* und $tv^{-1} = u$ als *Rechtsquotienten*. Ersetzungen werden wir durch Berechnen der Quotienten $d_2^{-1}t_2$ und $t_1 d_1^{-1}$ für Faktorisierungen $d_1 d_2 \in L_{\mathcal{R}}$ und $t_1 t_2 = t$ durchführen.

Analog wie bei CONCAT verwenden wir auch zur Quotientenberechnung die primitiven Operationen zur Formulierung eines rekursiven Berechnungsverfahrens.

FUNCTION rightQUOTIENT(t, l)
  Comment Input: two trace representations.
        Output: the representation of the quotient tl^{-1}.
  Method
    if $l \notin P(t)$ **then error**
      else if $t = \times(t_1, \ldots, t_k)$ and $l = \times(l_1, \ldots, l_k)$
        then for all $(1 \leq i \leq k)$ **do**
            $s_i \leftarrow$rightQUOTIENT(PROJECT(i, t),
                    PROJECT(i, l))
          end_for
          return (COMPOSE$(s_1, \ldots, s_k)$)
      else if $t = *(t_1, \ldots, t_n)$ and $l = *(l_1, \ldots, l_m)$

then if $m = 1$ **then**
$\langle s_1, s_2 \rangle \leftarrow$SPLIT$(n, t)$
interprete s_2 as $\times$-term
return (CONCAT(s_1,rightQUOTIENT(s_2, l_1)))
else $\langle s, t' \rangle \leftarrow$ SPLIT$(n - m + 1, t)$
$\langle l', l'' \rangle \leftarrow$ SPLIT$(2, l)$
return (rightQUOTIENT(s, l'))
end_if
else if $t \in a^*$ and $l \in a^*$ and $|l| \leq |t|$
then let $t = a^n$ and $l = a^m$
$t \leftarrow a^{n-m}$
for all $d \in \mathrm{Dec}(\mathcal{R}) \cap a^*$ **do**
$L_d \leftarrow [n - m + 1 - |d|]$
end_for
return t with labeling P, S, L
otherwise error
end_Method

Wir erhalten das folgende Lemma.

Lemma 32 *Seien* $t, l \in h(\mathbb{M})$, $l \in \mathrm{Dec}(\mathcal{R}$ *zwei korrekt beschriftete Spurterme und* $h^{-1}(l)$ *sei ein Suffix von* $h^{-1}(t)$, *d.h.* $h^{-1}(l) \in P(t)$. *Dann berechnet die Funktion rightQUOTIENT einen korrekt beschrifteten Spurterm* tl^{-1}. ∎

Die Komplexität wird von der Zeit für ein SPLIT zu $\mathcal{O}(\log|t| + \log|l|)$ (mit der effizienteren Implementierung von SPLIT) bestimmt. Da wir uns auf Quotientenberechnungen tu^{-1}, $u \in \mathrm{Pref}(L_{\mathcal{R}})$ beschränken, würde konstante Zeit ausreichen, denn für alle modifizierenden Operationen existieren Zugriffspfade auf die entsprechenden Teile der Datenstruktur. So ließen sich die Beschriftungslisten durch lineares Sondieren von maximalen Elementen aus in konstanter Zeit aktualisieren. Aber, da wir die Zugriffsstruktur für SPLIT noch verwalten, besitzt diese Realisierung die Komplexität $\mathcal{O}(\log|t|)$.

8.4 Dekomponierbarkeit

Um die komplizierten Basisfälle zu umgehen haben wir nur Cographen-Monoide betrachtet. Mit den gleichen Methoden lassen sich Algorithmen für eine größere Klasse von Spurmonoiden angeben. Die Basisfälle, bei denen die Rekursionen terminieren, lassen sich entsprechend verallgemeinern, so daß die Algorithmen die selbe Zeitkomplexität besitzen.

Ein Abhängigkeitsalphabet ist bzgl. des Ersetzungssystems $\mathcal{R}$ *dekomponierbar* oder *zerlegbar*, wenn die Menge $Z(\mathcal{R})$ nur atomare Spuren $d \in a^*$, $a \in \Sigma$, oder Spuren, die sich über ein freies Produkt faktorisieren, enthält. D.h. für alle $d \in Z(\mathcal{R})$ gilt $h(d) = *(d_1, \ldots, d_n)$ mit $n > 1$ oder $d \in a^*$. Dabei ist $Z(\mathcal{R}) = \bigcup_{l \in L_{\mathcal{R}}} Z(h(l))$ und

$Z(h(l))$ ist rekursiv wie folgt definiert:

$$
\begin{aligned}
Z(\times(l_1,\ldots,l_k)) &= \textstyle\bigcup_{i\in[k]} Z(l_i) \qquad\qquad & Z(a^n) &= \{a^n\} \quad a\in\Sigma,\, n\in\mathbb{N}\\
Z(*(l_1,\ldots,l_k)) &= \begin{cases} \{*(l_1,\ldots,l_k)\} & n>1\\ Z(l_1) & n=1\end{cases}
\end{aligned}
$$

Damit gilt für $d \in \mathrm{Dec}(\mathcal{R})$ entweder, daß die entsprechende Beschriftung von Subtermen ererbt werden kann, oder $d \in Z(\mathcal{R})$. Die Laufzeit der Algorithmenteile für die Basisfälle aus $Z(\mathcal{R})$ bei zerlegbaren Abhängigkeitsalphabeten liegt in $\mathcal{O}(1)$.

Bemerkung 5 *Cographen-Abhängigkeitsalphabete sind bzgl. jedem (noetherschen) Ersetzungssystem dekomponierbar.*

Wir betrachten fortan trotzdem nur Cographen-Monoide, da die Formulierung der Basisfälle komplizierter ist.

8.5 Ein $\mathcal{O}(n\log n)$-Reduktionsalgorithmus

Die dem folgenden Algorithmus zugrundeliegende Idee ist ein iteriertes Durchführen von Reduktionen „von links nach rechts“, wie in dem folgenden Algorithmenskelett dargestellt.

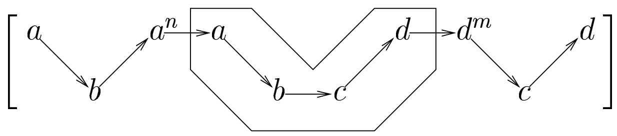
$v \leftarrow 1$;
while $t \neq 1$ **do**
    let $t = at'$ for some $a \in \Sigma, t' \in \mathbb{M}$;
    $v := va; t := t'$;
    if $v = v'l$ for some $l \to r \in \mathcal{R}$ **then** $v \leftarrow v'; t \leftarrow rt$ **end_if**
end_while
return v

Die Algorithmenskelett berechnet ein Derivat v der Spur t, $t \overset{*}{\Longrightarrow}_{\mathcal{R}} v$. Die Spur v ist aber nach Verlassen der Schleife nicht notwendigerweise irreduzibel, wie das Beispiel zeigt.

Beispiel 6 *Sei* $(\Sigma, D) = [a - b - c - d]$, $t = aba^n abcdd^m cd$ *und* $\mathcal{R} = \{abcd \to 1\}$. *Dann berechnet der Algorithmus die Derivation*

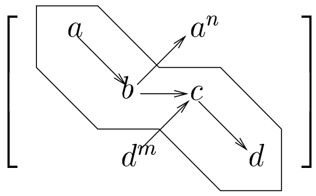
$$t = aba^n \cdot abcd \cdot d^m cd \Longrightarrow_{\mathcal{R}} aba^n d^m cd = t' \notin \mathrm{Irr}(\mathcal{R}),$$

denn die Spur

$t \quad =$ 

wird auf

$t' \quad =$



reduziert. Die Faktoren aus $L_{\mathcal{R}}$ sind mit Polygonzügen markiert. Dieses Beispiel illustriert auch, daß durch Zurücksetzen über einen gelesenen Teil konstanter Größe die Irreduzibilität nicht zugesichert werden kann.

Die folgende Methode CONCATandREDUCE vermeidet diesen Effekt durch Nutzen der Termstruktur.

Die Auswahl einer Faktorisierung

Die Methode FACTORIZE berechnet eine Faktorisierung $t_1 \cdot t_2$ von t so, daß t einen Faktor $l \in \mathrm{Dec}(\mathcal{R})$ an der Schnittstelle enthält, d.h. $t = t_1' l t_2'$, wobei ein Präfix von l ein Suffix von t_1 ist, und das ergänzende Suffix von l ein Präfix von t_2 ist. D.h. $l = l_1 l_2$, $t_1 = t_1' l_1$ und $t_2 = l_2 t_2'$. Dabei werden Faktorisierungen betrachtet, die sich an der Termstruktur orientieren. Die L-Beschriftungen ermöglichen ein zielgerichtetes navigieren in dem Term darstellenden Baum.

PROCEDURE FACTORIZE(l, t)
  Comment Input: a trace representation.
            Output: the factorization according to the semantic.
  Method
    if $t = *(t_1, \ldots, t_n)$ and $L_l(t) \neq \emptyset$
      then choose $j \in L_l(t)$ e.g. $j \leftarrow \min L_l(t)$
          if $l = *(l_1, \ldots l_m)$ with $m > 1$
            then return (SPLIT($t, j+1$))
            else $l = *(l_1)$
                $\langle t', t'' \rangle \leftarrow$SPLIT($t, j$)
                $\langle u, v \rangle \leftarrow$SPLIT($t'', 2$)
                interprete u as $\times$-term
                $\langle x, y \rangle \leftarrow$FACTORIZE($l, u$)
                return($\langle$CONCAT(t', x),CONCAT(y, v)$\rangle$)
          end_if
      else if $t = \times(t_1, \ldots, t_k)$ **and** $l = (l_1, \ldots, l_k) \in L(t)$
          for all $(1 \leq j \leq k)$ **do**
              $t_j \leftarrow$PROJECT(t, j)
              $\langle u_j, v_j \rangle \leftarrow$FACTORIZE($l_j, t_j$)
          end_for
          return($\langle$COMPOSE($u_1, \ldots, u_k$),COMPOSE($v_1, \ldots, v_k$)$\rangle$
      else if $t \in a^*$ and $L(t) \neq \emptyset$ **then return**($\langle a, a^{|t|-1} \rangle$)

(with maintained labeling)
else if $l = 1$ **then return** $(\langle 1, t\rangle)$
otherwise there is no factorization possible
end_Method

Wieder führt eine zu oben analoge Fallstudie zu dem folgenden Lemma.

Lemma 33 *Sei $t \in h(\mathbb{M})$ ein korrekt beschrifteter Spurterm mit $h^{-1}(t) = u \cdot l \cdot v$ für ein $l \in \mathrm{Dec}(\mathcal{R})$. Dann berechnet FACTORIZE(l, t) ein Paar von korrekt beschrifteten Spurtermen $\langle t_1, t_2\rangle$ mit $t = t_1 \circ t_2$, $l = l_1 l_2$, sowie $l_1 \in P(t_1)$ und $l_2 \in S(t_2)$.* ∎

Insbesondere berechnet FACTORIZE für jede linke Seite $l \in L_{\mathcal{R}}$, die als Faktor in einer Spur $t = ulv$ vorkommt, eine Faktorisierung $t = t_1 t_2$ mit $t_1 = ul_1$ und $t_2 = l_2 v$ für zwei Spuren l_1, l_2 mit $l_1 l_2 = l$.

Die Komplexität beträgt $\mathcal{O}(\log |t|)$ (mit der effizienteren Implementierung von SPLIT). Die Datenstruktur C^*-Baum ermöglicht den Zugriff auf das minimale Element $\min L_l(t)$ in logarithmischer Zeit. Da die Positionen über die C^*-Bäume vergleichbar sind, können so Faktorisierungen $t = t_1 \cdot t_2$ mit $t_1 \in \mathbb{M} \setminus \mathbb{M}l\mathbb{M}$ für Links-Derivationen berechnet werden.

Eine Reduktionsmethode für Links-Derivationen

Die Intention dieses Entwurfs ist es, daß die folgenden Zusicherungen invariant gegenüber Reduktionsschritten sind.

(i) Der Resultatterm ist ein korrekt beschrifteter Spurterm.

(ii) Der Resultatterm ist ein Links-Derivat von der Konkatenation der Eingabe.

(iii) Ist die erste Komponente der Eingabe irreduzibel, so ist auch das Resultat irreduzibel.

In natürlicher Weise erhält man einen Reduktionsalgorithmus vom Book-Typ.

PROCEDURE CONCATandREDUCE(s, t)
Comment Input: two trace representations.
Output: a descendant of st.
Method
if exists $l = l_1 l_2 \to r \in \mathcal{R}$ with $l_1 \in S(s)$ and $l_2 \in P(t)$ **then**
then let l be minimal choosen
$s' \leftarrow$rightQUOTIENT(s, l_1)
$t' \leftarrow$leftQUOTIENT(t, l_2)
return(CONCATandREDUCE(
CONCATandREDUCE$(s', r), t'$))

```
else if s = ×(s_1, ..., s_k) and t = ×(t_1, ..., t_k) and
        there exists l → r with l = (l_1, ..., l_k) ∈ L(s ∘ t)
        let l be minimal choosen
        then for all i ∈ [k] do
                s_i ← PROJECT(s, i)
                t_i ← PROJECT(t, i)
                if L_{l_i}(s_i) ≠ ∅
                   then ⟨s'_i, s''_i⟩ ← FACTORIZE(l_i, s_i)
                        s_i ← s'_i
                        t_i ← CONCAT(s''_i, t_i)
                   else if l_i = l'_i l''_i with l'_i ∈ P(s_i), l''_i ∈ S(t_i)
                        then do nothing
                        else ⟨t'_i, t''_i⟩ ← FACTORIZE(l_i, t_i)
                             s_i ← CONCAT(s_i, t'_i)
                             t_i ← t''_i
                        end_if
                   end_if
             end_for
             s ←COMPOSE(s_1, ..., s_k)
             t ←COMPOSE(t_1, ..., t_k)
             let l = l_1 · l_2 with l_1 ∈ P(s) and l_2 ∈ S(t)
             s' ←rightQUOTIENT(s, l_1)
             t' ←leftQUOTIENT(t, l_2)
             return(CONCATandREDUCE(
                          CONCATandREDUCE(s', r), t'))
else if s = ∗(s_1, ..., s_n) and t = ∗(t_1, ..., t_m) ....
```

Comment analogous as in the direct product case — we skip this part

```
    otherwise st ∈ Irr(R)
            return(CONCAT(s, t))
end_Method
```

Seien s und t die Eingaben. Existiert eine Faktorisierung, so daß ein Präfix einer linken Seite ein Suffix von s und das zugehörige Suffix dieser linken Seite ein Präfix von t ist, dann werden die entsprechenden Quotienten und Konkatenationen durchgeführt, um dieses Vorkommen zu ersetzten. Existiert keine solche Faktorisierung, so wird eine entsprechende berechnet, indem Teile von t an s bzw. Teile von s an t konkateniert werden. Im Falle eines direkte Produktes ist die Beschreibung dieser Berechnung ausgeführt. Der atomare Fall ist trivial, und bei freien Produkten verfährt man entsprechend. Existiert keine Faktorisierung dieser Art, so ist der Spurterm $s \circ t$ irreduzibel und wir können die Konkatenation als Resultat ausgeben.

Lemma 34 *Seien $s, t \in h(\mathbb{M})$ und $\mathcal{R}$ ein noethersches Ersetzungssystem.* $\mathbb{M}$ *ist ein Cographen-Monoid.*

(i) Der Resultatterm $h(t')$ von CONCATandREDUCE(s,t) ist ein korrekt beschrifteter Spurterm.

(ii) Es gilt $h^{-1}(s) \cdot h^{-1}(t) \overset{*}{\Longrightarrow}_{\mathcal{R}} t'$.

(iii) Aus $h^{-1}(s) \in \mathrm{Irr}(\mathcal{R})$ folgt $t' \in \mathrm{Irr}(\mathcal{R})$.

Beweis: Seien s und t wie oben Spurterme gleicher Struktur. Da nur die oben eingeführten Operationen und CONCATandREDUCE auf s und t angewendet werden ist das Resultat mit Induktion über die Anzahl der Transformationen CONCAT, SPLIT, PROJECT, COMPOSE, FACTORIZE, rightQUOTIENT und leftQUOTIENT mit den vorhergehenden Lemmata wieder ein korrekt beschrifteter Spurterm.

Die verbleibenden Behauptungen zeigen wir simultan. Ist die Bedingung der ersten Verzweigung erfüllt, so ist das Resultat ein Derivat von $s \circ t$:

$$st = s'l_1l_2t' = s'lt' \Longrightarrow_{\mathcal{R}} s'rt' \overset{*}{\Longrightarrow}_{\mathcal{R}} \hat{t}$$

Der Übersichtlichkeit wegen identifizieren wir t mit $h(t)$. Da jeder Rechtsquotient einer irreduziblen Spur wieder irreduzibel ist, bleibt das erste Argument irreduzibel, und wir erhalten die dritte Aussage.

Besteht die Eingabe aus zwei $\times$-Termen $s = \times(s_1, \ldots, s_k)$ und $t = \times(t_1, \ldots, t_k)$. Dann finden wir nach Lemma 24 einen Faktor $(l_1, \ldots, l_k)$ genau dann, wenn für alle $i \in [k]$ sich die Komponenten $h^{-1}(s_i \circ t_i) = u_il_iv_i$ faktorisieren lassen. Nach Wahl der linken Seite wird eine Faktorisierung gefunden so, daß das erste Argument irreduzibel bleibt. Das kann beispielsweise via FACTORIZE geschehen. Durch eine erschöpfende Suche, in der über alle linken Seiten, die als Faktoren vorkommen, variiert wird, läßt sich in logarithmischer Zeit eine Faktorisierung $u \cdot l \cdot v$ mit $u \in \mathrm{Irr}(\mathcal{R})$ finden. Man Beachte, daß in konstanter Zeit entschieden werden kann, ob ein beschrifteter Term irreduzibel ist. Damit bleibt das erste Argument irreduzibel, wenn s und t $\times$-Terme sind. Das Resultat ist ebenfalls ein Derivat:

$$\begin{pmatrix} s_1 \\ \vdots \\ s_k \end{pmatrix}\begin{pmatrix} t_1 \\ \vdots \\ t_k \end{pmatrix} = \begin{pmatrix} v_1 \\ \vdots \\ v_k \end{pmatrix} = \begin{pmatrix} v_1' \\ \vdots \\ v_k' \end{pmatrix}\begin{pmatrix} v_1'' \\ \vdots \\ v_k'' \end{pmatrix} =$$
$$\begin{pmatrix} v_1'(l_1^1)^{-1} \\ \vdots \\ v_k'(l_k^1)^{-1} \end{pmatrix} l \begin{pmatrix} (l_1^2)^{-1}v_1'' \\ \vdots \\ (l_k^2)^{-1}v_k'' \end{pmatrix} \Longrightarrow_{\mathcal{R}} \begin{pmatrix} v_1'(l_1^1)^{-1} \\ \vdots \\ v_k'(l_k^1)^{-1} \end{pmatrix} r \begin{pmatrix} (l_1^2)^{-1}v_1'' \\ \vdots \\ (l_k^2)^{-1}v_k'' \end{pmatrix}$$

Für $*$-Terme und atomare Terme führen wir den Beweis nicht aus. ∎

Korollar 11 *Sei $t \in \mathbb{M}$ eine Spur, dann gilt für den resultierenden Spurterm*

$$h(\hat{t}) \quad = \quad \mathit{CONCATandREDUCE}(1, h(t)),$$

$t \overset{*}{\Longrightarrow}_{\mathcal{R}} \hat{t} \in \mathrm{Irr}(\mathcal{R})$. ∎

Bemerkung 6 *Wir hätten die Spuren zuerst konkatenieren können, und auf der so entstandenen Spur wäre ein sukzessives Reduzieren möglich gewesen. Diese Darstellung verdeutlicht jedoch die Problematik des Auffindens eines einer Faktorisierung* $u \cdot l \cdot v$ *im Falle eines direkten Produkts. Man beachte die rekursiven Verflechtungen bei der Berechnung einer Faktorisierung . Die „gierige“ Vorgehensweise bei diesem Entwurf ähnelt der von Books Algorithmus. Außerdem kann die selbe Methode auch zur Konstruktion der Datenstruktur genutzt werden.*

Verwendet man die Methode FACTORIZE, so wird der Algorithmus sehr viel übersichtlicher und der Korrektheitsbeweis nahezu trivial.

```
PROCEDURE REDUCE⁺(t)
  Comment Input: trace representation.
          Output: the representation of the normal form.
  Method
    while ∃l ∈ L_R with (l = l_1 · ... · l_m and L_l(t) ≠ ∅ or
                         l = (l_1, ..., l_k) and l ∈ L(t) ≠ ∅)
                         and l → r ∈ R do
      ⟨u, v⟩ ←FACTORIZE(l, t)
      let  l = l_1 l_2 with l_1 ∈ P(u) and l_2 ∈ S(v)
      s ←CONCAT(leftQUOTIENT(u, l_1), r)
      t ←CONCAT(rightQUOTIENT(s, l_2), v)
    end_ while
    return(t)
  end_Method
```

Lemma 35 *Die Methode CONCATandREDUCE*$(h(s), h(t)) = h(t')$ *benötigt maximal* $\mathcal{O}(n)$ *primitive Operationen, wobei* n *die maximale Anzahl der Ersetzungen* $st \overset{n}{\Longrightarrow}_{\mathcal{R}} t'$ *ist.*

Beweis: Bei jedem Aufruf von CONCATandREDUCE wird, falls existent, l in st durch r für eine Regel $l \to r \in \mathcal{R}$ ersetzt. Dazu sind nur konstant viele komplexe Elementaroperationen notwendig. ∎

Wir erhalten damit das folgende Komplexitätsresultat.

Theorem 14 *Sei* (Σ, D) *ein Cographen-Abhängigkeitsalphabet und* $\mathcal{R}$ *ein noethersches Ersetzungssystem über* $\mathbb{M} = \mathbb{M}(\Sigma, D)$. *Dann kann für eine Spur* $t \in \mathbb{M}$ *eine Links-Derivation* $t \overset{*}{\Longrightarrow}_{\mathcal{R}} \hat{t}$ *in* $\mathcal{O}(d_{\mathcal{R}} \log d_{\mathcal{R}})$ *berechnet werden.* ∎

Korollar 12 *Sei* (Σ, D) *ein Cographen-Abhängigkeitsalphabet und* $\mathcal{R}$ *ein gewichtsreduzierendes Ersetzungssystem über* $\mathbb{M} = \mathbb{M}(\Sigma, D)$. *Dann kann für eine Spur* $t \in \mathbb{M}$ *eine Links-Derivation* $t \overset{*}{\Longrightarrow}_{\mathcal{R}} \hat{t}$ *in* $\mathcal{O}(|t| \log |t|)$ *berechnet werden.* ∎

Man beachte, daß mit diesem Algorithmus auch Ersetzungssystem mit nicht-zusammenhängenden linken Seiten bearbeitet werden können.

Bemerkung 7 *Dieses Resultat läßt sich auf dekomponierbare Abhängigkeitsalphabete verallgemeinern. Die Behandlung der Spuren über nuklearen Alphabeten ist bei allen Algorithmen in der Worttupel-Repräsentation dann innerhalb der angegebenen Zeitgrenzen möglich.*

Theorem 15 *Sei (Σ, D) ein Abhängigkeitsalphabet und $\mathcal{R}$ ein noethersches Ersetzungssystem, so daß $\mathbb{M} = \mathbb{M}(\Sigma, D)$ dekomponierbar ist. Dann kann für eine Spur $t \in \mathbb{M}$ jede Links-Derivation $t \overset{*}{\Longrightarrow}_{\mathcal{R}} \hat{t}$ in $\mathcal{O}(d_{\mathcal{R}} \log d_{\mathcal{R}})$ berechnet werden.* ∎

Wie schon oben bemerkt läßt sich diese Version des Reduktionsalgorithmus so modifizieren, daß jede beliebige Faktorisierungssequenz frei gewählt werden kann. D.h. spezifiziert durch Zugriffspfade ist die freie Auswahl der Vorkommen und die Auswahlfreiheit bei den durch die Vorkommen induzierten Faktorisierungen in einer Derivation

$$\begin{aligned} t = u_1 l_1 v_1 \Longrightarrow_{\mathcal{R}} u_1 r_1 v_1 = u_2 l_2 r_2 \Longrightarrow_{\mathcal{R}} \cdots \\ \cdots \Longrightarrow_{\mathcal{R}} u_{n-1} r_{n-1} v_{n-1} = u_n l_n v_n \Longrightarrow_{\mathcal{R}} \hat{t}, \end{aligned}$$

möglich.

Theorem 16 *Sei (Σ, D) ein Abhängigkeitsalphabet und $\mathcal{R}$ ein noethersches Ersetzungssystem so, daß $\mathbb{M} = \mathbb{M}(\Sigma, D)$ dekomponierbar ist. Dann kann für eine Spur $t \in \mathbb{M}$ jede irreduzible Form $t \overset{*}{\Longrightarrow}_{\mathcal{R}} \hat{t}$ in $\mathcal{O}(d_{\mathcal{R}} \log d_{\mathcal{R}})$ berechnet werden.* ∎

8.6 Ein Linearzeit-Reduktionsalgorithmus

Nutzen wir die spezielle Struktur der Abhängigkeitsalphabete bzw. die der Spurdarstellungen, so kann eine irreduzible Form in linearer Zeit berechnet werden kann. Genauer kann eine Derivation $t \overset{n}{\Longrightarrow}_{\mathcal{R}} \hat{t} \in \mathrm{Irr}(\mathcal{R})$ in $\mathcal{O}(n)$ Zeiteinheiten berechnet werden kann.

Bei dem vorhergehenden Ansatz besaßen die Operationen SPLIT sowie CONCAT, rightQUOTIENT, leftQUOTIENT und FACTORIZE logarithmische Zeitkomplexität. Diese resultierte aus der Laufzeit von SPLIT, wegen der Verwaltung der Beschriftungen L_d in $\mathbb{N}^*$-Bäumen. Das war notwendig, da die Listen geordnet aufgeteilt werden mußten, um die Invariante „Resultate sind korrekt beschriftet" aufrecht erhalten zu können. Die zeitintensive Operation ist also das Zerteilen der Spur bzw. das Aufrechterhalten der Beschriftungen. Dieses Problem umgehen wir, indem die Beschriftungen noch enger an die Term- bzw. Baumstruktur gekoppelt werden.

Der modifizierte abstrakte Datentyp

Wir entwickeln in diesem Abschnitt Datenstrukturen und Primitiven, um einen linearen Normalform-Algorithmus zu formulieren. Dazu modifizieren wir die Beschriftung leicht.

Wir nennen einen Spurterm t *schwach beschriftet*, wenn alle Subterme schwach beschriftet sind und, falls t selber ein $*$-Term ist, t die Beschriftung $L(t)$ trägt (siehe Abschnitt 8.2). Die fehlenden Beschriftungen können in konstanter Zeit dynamisch berechnet werden, wenn Prädikate der Form $L_d(*(t_1, \ldots, t_n)) \neq \emptyset$ in konstanter Zeit entschieden werden können.

Lemma 36 *Sei $t = \times(t_1, \ldots, t_k)$ ein schwach beschrifteter $\times$-Term so, daß die Prädikate $L_d(t_i) \neq \emptyset$ für alle $i \in [k]$ und alle $d \in \mathrm{Dec}(\mathcal{R})$ in konstanter Zeit entschieden werden können. Dann kann die Beschriftung $L(t)$ in konstanter Zeit berechnet werden.*

Beweis: Die Menge

$$L(t) \;=\; \{d \in \mathrm{Dec}(\mathcal{R}) \mid h(d) = \times(d_1, \ldots, d_k), \forall i \in [k] : d_i \neq 1 \Rightarrow L_{d_i} \neq \emptyset\}$$

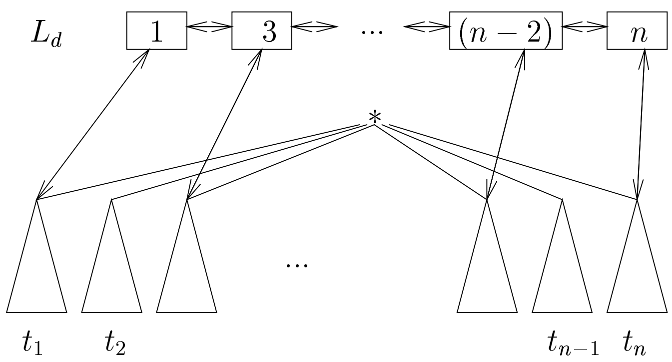
enthält höchstens $\mathcal{O}(|\mathrm{Dec}(\mathcal{R})|) = \mathcal{O}(1)$ viele Elemente, deren Darstellung die konstante Größe $\mathcal{O}(\max\{|l| \mid l \in L_{\mathcal{R}}\}) = \mathcal{O}(1)$ besitzt. Der Test $L_{d_i} \neq \emptyset$ kann iteriert für alle $i \in [k]$ in konstanter Zeit durchgeführt werden. ■

Als Datenstruktur verwenden wir für Terme wieder Bäume. Präziser verwenden wir Bäume mit drei Knotensorten: Mit $*$ oder $\times$ assoziierte innere Knoten und mit Wörtern aus a^*, $a \in \Sigma$ assoziierte Blätter. Die $\times$-Knoten haben eine durch das entsprechende direkte Produkt bestimmte konstante Anzahl von Kindern, denn die bei dem Produkt involvierten Alphabete sind disjunkt. Damit ist die maximale Anzahl der Kinder (in diesem Fall $|\Sigma|$) unabhängig von der zu reduzierenden Spur. $*$-Knoten können mehr als $\mathcal{O}(1)$ Kinder besitzen. Die Anzahl dieser Kinder ist nur linear in der Länge der zu reduzierenden Spur beschränkt. Die Bäume besitzen wie die Terme konstante Tiefe und die Deskriptionen haben konstante Größe.

Die Elemente $i \in L_d(t)$ einer L-Beschriftung eines Spurterms

$$t \quad = \quad *(t_1, \ldots, t_n)$$

stellen wir als Verweise auf den i-ten Subterm t_i dar. In der folgenden Abbildung ist die Zeigerstruktur für einen $*$-Knoten illustriert. Man beachte, daß die Ordnung in der Liste L_d, $d \in \mathrm{Dec}(\mathcal{R})$ verloren gehen kann, und daß ein Subterm, der den Faktor d enthält, in $\mathcal{O}(1)$ Zeiteinheiten gefunden werden kann.



Um Faktoren in der Spur darzustellen, haben wir schon in früheren Kapiteln Vorkommen-Morphismen und Positionen in Worttupeln kennengelernt. Der Positionsbegriff für diese Repräsentation orientiert sich (wie auch die Darstellung durch Terme) an der Dekomposition des Abhängigkeitsalphabetes.

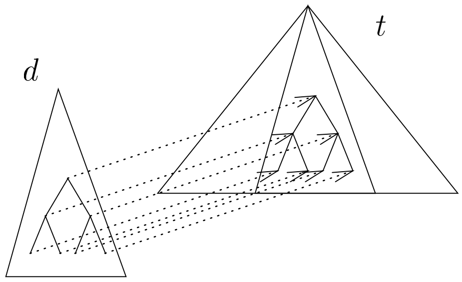
Sei (Σ, D) das Abhängigkeitsalphabet und $\mathrm{Dec}(\Sigma, D) = \{(\Sigma_i, D_i) \mid i \in [k]\}$ die Dekomposition. Unter einem *Vorkommenbaum* für $d \in \mathrm{Dec}(\mathcal{R})$ (bzgl. (Σ, D)) verstehen wir den den Term $h(d)$ repräsentierenden Baum p. D.h. der den folgenden Bedingungen genügende Baum.

(i) Gilt $(\Sigma, D) = \oplus_{i\in[k]}(\Sigma_i, D_i)$ und besitzt d die Form $\times(d_1, \ldots, d_k)$, so besitzt p eine Wurzel mit k Söhnen, und der i-te Sohn ist die Wurzel eines Vorkommenbaums von d_i bzgl. (Σ_i, D_i) für alle $i \in [k]$.

(ii) Gilt $(\Sigma, D) = *_{i\in[k]}(\Sigma_i, D_i)$ und besitzt d die Form $*(d_1, \ldots, d_m)$, so besitzt p eine Wurzel mit m konsekutiven Söhnen (in t), und der j-te Sohn ist die Wurzel eines Vorkommenbaums von d_j bzgl. des entsprechenden Abhängigkeitsalphabetes für alle $j \in [m]$.

(iii) Ist (Σ, D) nuklear (atomar), so besteht p aus einem Knoten.

Ein *(Baum-)Vorkommen* $i : l \longrightarrow t$ von d in einem Spurterm repräsentierenden Baum t ist ein Vorkommenbaum d, der via Referenzen zu einem zu ihm isomorphen Unterbaum von t gebunden ist, so daß bei $l = *(l_1, \ldots, l_m)$ l_1 linkspassend und l_m rechtspassend ist, falls $m > 1$. Dieser Teilbaum repräsentiert den Faktor d.



Ein *rechtspassendes* Vorkommen ist ein Suffixvorkommen. Entsprechend ist ein *linkspassends* Vorkommen ein Präfixvorkommen. Damit besitzt ein rechtspassendes Baumvorkommen $j : d \longrightarrow t$ keinen unreferenzierten $*$-Knoten als Bruder, der sich rechts von einem referenzierten Knoten befindet. Alle Teilbäume, die sich rechts von einem referenzierten $*$-Knoten befinden, sind dann vollständig referenziert. Genauer ist das Vorkommen $*(d_1, \ldots, d_m) \longrightarrow *(t_1, \ldots, t_n)$ rechtspassend, wenn $d_1 = t_1, \ldots, t_{m-1} = d_{m-1}$ und $d_m \longrightarrow t_m$ rekursiv rechtspassend ist. Das Vorkommen $\times(d_1, \ldots, d_k) \longrightarrow \times(t_1, \ldots, t_k)$ ist rechtspassend, wenn alle Vorkommen $d_i \longrightarrow t_i$, $i \in [k]$ rechtspassend sind. Analog dazu behandelt man linkspassende Baumvorkommen $j : d \longrightarrow t$.

Anders formuliert sei $i : l \longrightarrow t$ ein Faktorvorkommen, dann korrespondieren die Blättern in diesem Teilbaum mit den Elementen aus $i(l)$. Die Knoten von einem Vorkommen $i : d \longrightarrow t$ sind mit entsprechenden L-Beschriftungen assoziiert. Genauer gilt für $d = \times(d_1, \ldots, d_k)$, daß ein Verweis auf $i(d_j)$ in $L_{d_j}(t_j)$ mit $t = \times(t_1, \ldots, t_k)$ existiert. Für einen $*$-Term $d = *(d_1, \ldots, d_m)$ findet sich der Verweis auf die Wurzel von $i(d)$ in $L_d(t)$ wieder. Ist also ein Spurterm schwach beschriftet, dann läßt sich in konstanter Zeit ein Vorkommen von $d \in \mathrm{Dec}(\mathcal{R})$ identifizieren.

Lemma 37 *Sei t ein schwach beschrifteter Spurterm in der oben beschriebenen Repräsentation. Dann kann ein Baumvorkommen $i : d \longrightarrow t$ in konstanter Zeit ermittelt werden.*

Beweis: Gilt $t \in a^*$ (der atomare Fall), so ist die Aussage trivial.

Falls $t = \times(t_1, \ldots, t_k)$ ein $\times$-Term ist mit $d = \times(d_1, \ldots, d_k) \in L(t)$, so besitzt t den Faktor d. Man beachte, daß unter Verwendung der Aussage aus dem vorhergehenden Lemma dieses Prädikat in konstanter Zeit entschieden werden kann. Dann besitzen die Terme t_i die Faktoren d_i und es können in konstanter Zeit die Vorkommenbäume für $d_i \longrightarrow t_i$ berechnet werden. Diese lassen sich zu einem Vorkommenbaum in konstanter Zeit kombinieren, siehe Lemma 24.

Falls $t = *(t_1, \ldots, t_n)$ ein $*$-Term mit $L_d(t) \neq \emptyset$ ist, so enthält t den Faktor d. Wir unterscheiden die zwei Fälle $d = *(d_1)$ und $*(d_1, \ldots, d_m)$ mit $m > 1$. Sei $i \in L_d(t)$ ein Verweis auf den Term t_i. Im ersten Fall erzeugen wir eine Wurzel mit der entsprechenden Referenzierung, an der wir ein Vorkommenbaum von $d \longrightarrow t_i$ anhängen. Dieser läßt sich rekursiv mit konstantem Aufwand berechnen. Im zweiten Fall berechnen wir den Teilbaum von t_i, der das Suffix d_1 repräsentiert. Dieses muß existieren, da $i \in L_d(t)$. Das ist mit der Methode, die im Beweis von Lemma 25 beschrieben wird, in konstanter Zeit möglich. Analog identifizieren wir den Teilbaum, der zu dem Präfix d_m in t_{i+m-1} korrespondiert. Damit erhalten wir den rechtspassenden und den linkspassenden Vorkommenteilbaum. Der Vorkommenteilbaum setzt sich dann aus einer Wurzel und den Bäumen, die den Termen t_j für $i < j < i+m-1$ entsprechen, sowie dem rechts- und dem linkspassenden Vorkommenteilbaum, zusammen, siehe Lemma 24. Von den so ermittelten m konsekutiven Bäume werden Kopien erzeugt, die ihre Originale (knotenweise) referenzieren. Die Kopien werden ihrem Index nach geordnet unter die auf den Vater von t_j verweisende Wurzel gehängt. Alle diese Operationen erfordern konstante Zeit, da Zeigerstrukturen zwischen Beschriftungen und Baumteilen die entsprechenden Zugriffe ermöglichen. Die berechnete Struktur besitzt ebenfalls konstante Größe. Die Anzahl der Invokationen ist durch $|\Sigma|$ beschränkt. ■

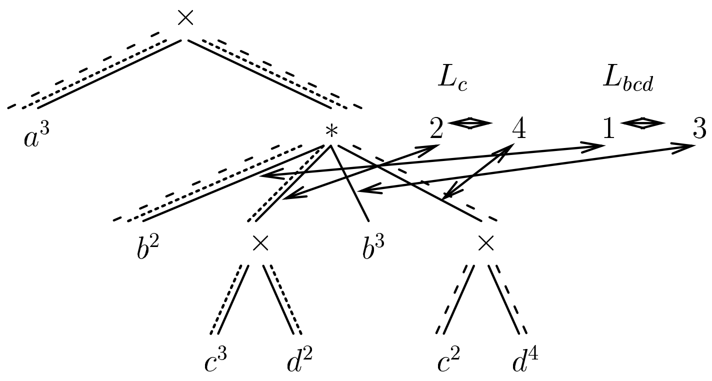
Beispiel 7 *Sei* $(\Sigma, D) = a \oplus (b * (c \oplus d)) = [a \quad c - b - d]$. *und*

$$t \quad = \times(a^3, *(b^2, \times(c^3, d^2), b^3, \times(c^2, d^4))).$$

Weiter sei $l = [abcd]$ eine linke Seite. Damit ist

$$\mathrm{Dec}(l) = \{a, b, c, d, cd, bcd, abcd\}.$$

Wir erhalten den repräsentierenden Baum



Es sind nicht alle Beschriftungen dargestellt. Der mit punktierten Linien markierte Teilbaum ist ein nicht linkspassendes Vorkommen von l. Man beachte das zweite b. Der mit gestrichelten Linien markierte Baum ist kein Vorkommen.

Da wir nicht mit Konkatenations- bzw. Faktorisierungsoperationen arbeiten, benötigen wir einen Mechanismus, um die Datenstruktur (lokal) zu modifizieren. Hierzu führen wir ein Positionskonzept ein.

Eine *Position* p in einer Baumrepräsentation t ist eine Baumstruktur, die eine Faktorisierung $u \circ v$ beschreibt. Genauer ist eine Position ein Unterbaum in t mit folgenden Eigenschaften:

(i) Gilt $t = \times(t_1, \ldots, t_k)$, so ist die Wurzel von t die Wurzel von p, und p besitzt k Kinder, wobei für alle $i \in [k]$ das i-te Kind die Wurzel einer Position in t_i ist.

(ii) Gilt $t = *(t_1, \ldots, t_n)$, so besteht p aus einem Knoten, der die Wurzel eines Teilbaums t_i, $i \in [n]$ referenziert, oder p ist eine Position in t_i.

(iii) Ist t nuklear (atomar), so ist p ein Blatt, das mit einer Faktorisierung $u \cdot v$ von t beschriftet ist.

Jede Position beschreibt eine Faktorisierung eindeutig. Präziser, jedes einen $*$-Term $*(t_1, \ldots, t_n)$ referenzierende Blatt, das den Subterm t_i, $i \in [n]$ referenziert, beschreibt die Faktorisierung $*(t_1, \ldots, t_i) \circ *(t_{i+1}, \ldots, t_n)$. Jeder $\times$-Knoten beschreibt die Faktorisierung durch die Faktorisierungen der einzelnen Komponenten. Bei Cographen-Monoiden sind die Positionen in Atomen a^n Faktorisierungen $a^i \cdot b^j$ mit $i + j = n$. Diese lassen sich einfach durch Stringpositionen repräsentieren.

Die primitiven Operationen

Bis jetzt haben wir die statische Termstruktur mit der Beschriftung betrachtet. In diesem Abschnitt beschreiben wir, wie diese Strukturen erzeugt und modifiziert werden können. Dazu führen wir die zwei Basisoperationen INSERT und DELETE mit der folgenden Semantik ein. Sei p eine Position in t, $a \in \Sigma$ ein Buchstabe und t die zu modifizierende Datenstruktur.

(i) Dann berechnet INSERT(a,t,p) eine schwach beschriftete Struktur t', in der nach der durch p beschriebenen Position ein a eingefügt wird. Genauer, wenn in der Position p ein Pfad zu einem Blatt a^n in t existiert, so wird dieses Blatt zu a^{n+1} durch Anfügen eines Symbols a geändert, und die Beschriftungen sowie die Position werden entsprechend aktualisiert. Ist kein Blatt mit dieser Beschriftung vorhanden, so beschreibt die Position eineindeutig, an welche Stelle das a eingefügt werden muß. Ist ein neues Blatt generiert worden, so wird die Position entsprechend angepaßt, daß sie dieses neue Blatt auch referenziert.

(ii) Dual berechnet DELETE(a,t,p) eine schwach beschriftete Struktur t', in der an der durch p beschriebenen Position das Blatt a^n durch a^{n-1} ersetzt wurde. Genauer existiert in p ein Pfad zu einem Blatt a^n in t. Dieses Blatt wird auf a^{n-1} durch Entfernen eines as geändert, und die Beschriftungen werden entsprechend der Definition aktualisiert. Wir fordern, daß p ein Blatt a^n mit $n > 0$ referenziert. Muß ein Blatt entfernt werden, d.h. $n = 1$, so wird die Position entsprechend angepaßt.

INSERT

Wir beschreiben die Operation INSERT in der gewohnten Notation. Der Algorithmus geht wie folgt vor: Zuerst wird die Stelle über die Position identifiziert, an der das a eingefügt werden soll. Falls die Position nicht direkt auf ein Blatt verweist, so wird ein entsprechender Teilbaum erzeugt. Leere Teilbäume repräsentieren das neutrale Element im entsprechenden Submonoid. Ist das a eingebaut, so werden die Beschriftungen bottom-up korrigiert.

PROCEDURE FIND(a,t,p)
  Comment Input: weakly labeled trace representation,
            a letter a and a position p.
        Output: an a^*-labeled leaf in t and a position.
  Method
    let v be the root of p.
    if v has a child and the alphabet corresponding
      to this child contains the letter a **then**
      let v' be this child
      let t' the v'-referenced subtree of t
      $\langle p', l\rangle \leftarrow$ FIND(a,t',p')
      replace the v'-rooted subtree of p by p'
      return(p,l)
    else if $t = *(t_1,\ldots,t_n)$ **then**
          let t_i be the p-referenced subterm
          if the alphabet corresponding to t_{i+1}, $i < n$ contains a
            then create position v' referencing on t_{i+1}
                $\langle p', l\rangle \leftarrow$ FIND(a,t_{i+1},v')
                replace child of v by the tree p'
            else create a new tree t'_i between t_i and t_{i+1}

```
                create position $v'$ referencing on $t'_i$
                $\langle p'', l\rangle \leftarrow$ FIND$(a, t', v')$
                let $p$ be the position with root $v'$
                    where the root of $p''$ is the child of $v'$
        end_if
        return $(\langle p, l\rangle)$
    else if $t = \times(t_1, \ldots, t_k)$ then
        let $v'$ be the child of $v$ where the alphabet contains $a$
        let $t_i$ be the $v'$ referenced subtree of $t$
        let $p''$ be $v'$-rooted subtree of $p$
        $\langle p', l\rangle \leftarrow$ FIND$(a, t_i, p'')$
        replace the subtree $p''$ by $p'$ in $p$
        return $(\langle p, l\rangle)$
    otherwise if there is no $a^*$ labeled leaf
                then create the leaf $l$ labeled by $a^0$ and
                      let $p$ be the position of this leaf
                else let $l$ be this leaf
            end_if
            return $(\langle p, l\rangle)$
    end_if
end_Method
```

Beim Abstieg in dem Positionsbaum wird man durch das vom Subterm induzierte Abhängigkeitsalphabet geführt. Dabei verstehen wir hier unter dem von einem Term $h(t)$ induzierten Alphabet das Indexalphabet (Σ, D) des Homomorphismus $h = h_{(\Sigma,D)}$.

Wenn eine Position einen Subterm t_i in einem $*$-Term $t = *(t_1, \ldots, t_n)$ referenziert und das von t_i induzierte Alphabet den Buchstaben a enthält, so findet man das gesuchte Blatt in t_i. Enthält das induzierte Alphabet den Buchstaben nicht, aber das von t_{i+1} induzierte Alphabet, so findet man das Blatt in t_{i+1}. Ansonsten muß ein neuer Subterm bzw. Unterbaum t'_i nach t_i und, falls t_{i+1} existiert, vor t_{i+1} eingebaut werden. Das Resultat ist dann ein $*$-Term $*(t_1, \ldots, t_i, t'_i, t_{i+1}, \ldots, t_n)$ und die Position muß so verändert werden, daß sie t'_i referenziert. Die Erweiterung der Position um die Referenzen auf t'_i können rekursiv erfolgen.

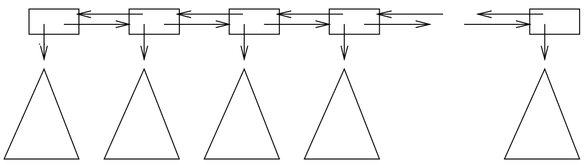
Referenziert die Position einen $\times$-Term $\times(t_1, \ldots, t_k)$, so existiert ein Subterm t_i, dessen induziertes Alphabet den Buchstaben a enthält. Dieser kann rekursiv nach dem Blatt durchsucht werden.

Damit sind wir in der Lage, die Primitive INSERT zu formulieren.

PROCEDURE INSERT(a, t, p)
Comment Input: weakly labeled trace representation t,
a letter a, and a position p.
Output: trace weakly labeled representation where the a
is inserted at position p and a modified position.

```
Method
  ⟨p, v⟩ ← FIND(a, t, p)
  replace the label a^n of leaf v by a^(n+1)
  while v is not the root of t do
    v' ← v
    v ← father of v'
    if v is corresponding to a *-term t' = *(t_1, ..., t_i, ..., t_n) then
      let t_i corresponding to v'
      for all d ∈ Dec(R) where h(d) = *(d_1, ..., d_m)
             for the recursion of h corresponding to t' do
        if m = 1
          then reconstruct L(t_i) with Lemma 36
               if d ∈ L(t_i)
                 then create a reference to t_i in L_d(t')
                        if there is none
                 else delete the reference to t_i in L_d(t')
                        if there is one
               end_if
          else (when m > 1)
               for all 1 ≤ j ≤ m do
                 if d_1 ∈ Suff(t_1 · ... · t_{i-j}) and
                   d_2 · ... · d_m ∈ Pref(t_{i-j+1} · ... · t_n)
                   then create a reference to t_{i-j} in L_d(t')
                          if there is none
                   else delete the reference to t_{i-j} in L_d(t')
                          if there is one
    end_if end_for end_if end_for end_if
  end_while
  return (⟨t, p⟩)
end_Method
```

Die Beschriftungsinformation wird hier für alle Baumknoten, die sich in der „Nähe" eines Pfadknoten befinden, aktualisiert. „Nähe" bedeutet hier in einer Umgebung der Distanz $m = \mathcal{O}(1)$, wobei die Distanz die Entfernung von zwei Geschwistern in der repräsentierenden Baumstruktur ist. Wir stellen einen inneren Knoten durch eine geordnete doppelt verkettete Liste wie in der Zeichnung dar. D.h. die Distanz zwischen zwei Geschwistern ist die Länge der sie verbindenden Liste.



Damit können in dieser Struktur quasi benachbarte Knoten im Redukt des Abhängigkeitsgraphen in der in Kapitel 5 erwähnten Metrik sehr weit voneinander entfernt sein, ohne daß die Struktur quadratischen Platz benötigt. Wir können das folgende Resultat zeigen.

Lemma 38 *Sei t eine Spurrepräsentation mit einer Position p, die die Faktorisierung uv repräsentiert. Dann berechnet INSERT(a, t, p) eine schwach beschriftete Spurrepräsentation von $uav = t'$. Die modifizierte Position repräsentiert danach die Faktorisierung $u'v$ mit $u' = ua$.*

Beweis: Die Stelle, an der das a eingefügt werden soll, wird top-down mit FIND berechnet. Verweist die Position auf ein mit a^n beschriftetes Blatt in t, so wird diese Blatt in FIND durch Absteigen gefunden. Jedem Term in $t \in h(\mathbb{M})$ in t kann eineindeutig das Abhängigkeitsalphabet (Σ, D) zugeordnet werden, das der Rekursionsinvokation des Homomorphismus $h = h_{(\Sigma,D)}$ entspricht. Damit ist auch mit jedem inneren Knoten v das vom Term, der durch den Teilbaum mit Wurzel v repräsentiert wird, induzierte Alphabet assoziiert.

Existiert kein Pfad in der Position, der an einem a^*-beschrifteten Blatt endet, so wird für die Kette von Abhängigkeitsalphabeten in $\mathrm{Dec}^*(\Sigma, D)$, die bei (Σ, D) beginnt und zu $(\{a\}, \{(a, a)\})$ absteigt, ein Pfad in t zu einem mit a^0 beschrifteten Blatt kreiert. Dabei wird gleichzeitig der Termbaum und die Position entsprechend erweitert.

Danach wird dem Blatt ein a angefügt. An dieser Stelle im Algorithmus repräsentiert der Baum $t = uv$ die Spur, an der durch die Position p beschriebene Stelle, ein Blatt eingefügt wurde. Damit repräsentiert der Resultatterm uav. Nur die Beschriftung ist jetzt noch inkonsistent.

In der while-Schleife in dem Algorithmus wird die L-Beschriftung wieder hergestellt. Wir zeigen dazu die folgende Invariante: Der dem Subterm t' entsprechende Teilbaum, also der Teilbaum mit der Wurzel v, ist schwach beschriftet.

Beim Betreten der Schleife ist die Invariante erfüllt, v ist dann ein Blatt, und Blätter sind nicht beschriftet. Ist t' ein $\times$-Term, so ist t' bereits schwach beschriftet. Wenn $t' = *(t_1, \ldots, t_n)$, also t' ein $*$-Term ist, dann bleiben alle Beschriftungen $L_d(t') \cap [i-m]$ und $L_d(t') \cap \{j \in [n] \mid j > i\}$ erhalten, wobei t_i der Subterm ist, in dem das a eingefügt wurde und m die Arität von $d = *(d_1, \ldots, d_m)$, dem betrachteten Element $d \in \mathrm{Dec}(\mathcal{R})$, ist. Diese Subterme bzw. die korrespondierenden Bäume t_k, $k \in [n]$ sind alle schwach beschriftet. Für die Stellen $(i-m), \ldots, i$ werden die L_d-Beschriftungsinformationen entsprechend der Definition von L_d für alle $d \in \mathrm{Dec}(\mathcal{R})$, die vorkommen können, neu berechnet. Damit ist der ganze Term t' schwach beschriftet.

Beim Verlassen der while-Schleife ist v der Wurzelknoten. Damit ist der gesamte Baum, d.h. das Resultat, schwach beschriftet. ∎

Die verwendete Datenstruktur ermöglicht dies sogar in $\mathcal{O}(1)$ Zeiteinheiten, denn alle involvierten Prädikate lassen sich in $\mathcal{O}(1)$ Zeiteinheiten entscheiden und die Anzahl der Iterationen bzw. Inkarnationen ist durch $\mathcal{O}(|\Sigma| \cdot |\mathrm{Dec}(\mathcal{R})| \cdot \max\{|l| \mid l \in L_{\mathcal{R}}\}) = \mathcal{O}(1)$ beschränkt.

DELETE

Analog formulieren wir die Operation DELETE. Hier verlangen wir jedoch, daß die Position einen Pfad besitzt, der in einem Blatt endet, das auf a^n in t verweist.

PROCEDURE DELETE(a, t, p)
 Comment Input: weakly labeled trace representation,
  a letter a, and a position p.
  Output: weakly labeled trace representation where the a
   is deleted at position p and a modified position.
 Method
  decent the path in t (given by p) ending at an a^*-labeled leaf
  replace the leaf a^{n+1} by a^n
  let v be this leaf
  if $n = 0$ **then**
   while v is not the root of t and v has no child **do**
    $v' \leftarrow$ father of v
    purge v in t and the corresponding references in i
    $v \leftarrow v'$
   end_while
   restore the reference in i
  end_if
  while v is not the root of t **do**
   $v' \leftarrow v$
   $v \leftarrow$ father of v'
   if v is corresponding to a $*$-term $t' = *(t_1, \ldots, t_i, \ldots, t_n)$ **then**
    let v' corresponding to t_i
    for all $d \in \mathrm{Dec}(\mathcal{R})$ where $h(d) = *(d_1, \ldots, d_m)$
      for the recursion of h corresponding to t' **do**
     if $m = 1$
      then reconstruct $L(t_i)$ with Lemma 36
        if $d \in L(t_i)$
         then create a reference to t_i in $L_d(t')$
           if there is none
         else delete the reference to t_i in $L_d(t')$
           if there is one
        end_if
      else (when $m > 1$)
       for all $1 \leq j \leq m$ **do**
        if $d_1 \in \mathrm{Suff}(t_1 \cdot \ldots \cdot t_{i-j})$ and
         $d_2 \cdot \ldots \cdot d_m \in \mathrm{Pref}(t_{i-j+1} \cdot \ldots \cdot t_n)$
         then create a reference to t_{i-j} in $L_d(t')$
           if there is none
         else delete the reference to t_{i-j} in $L_d(t')$
           if there is one

```
                    end_if
                end_for
            end_if
        end_for
    end_if
end_while
return (⟨t, p⟩)
end_Method
```

Wir entfernen ein a aus t an der Position analog wie wir ein Blatt einfügen. Lediglich die Fortsetzung des Zugriffspfades muß durch eine Verkürzung dieses Pfades ersetzt werden, wenn Knoten aus t entfernt werden müssen.

Lemma 39 *Sei t eine Spurrepräsentation mit einer Position p, die auf ein mit a^n beschriftetes Blatt verweist und damit eine Faktorisierung uav darstellt. Dann berechnet DELETE(a, t, p) eine Spurrepräsentation $t' = uv$, in der das a entfernt wurde. p wird entsprechend modifiziert, so daß die Position danach die Faktorisierung uv darstellt. Das Resultat t' ist schwach beschriftet.* ■

Auch diese Methode besitzt eine konstante Zeitkomplexität. Fassen wir diese Teilresultate zusammen, so erhalten wir das folgende Theorem.

Theorem 17 *Sei (Σ, D) ein Cographen-Abhängigkeitsalphabet und $\mathcal{R}$ ein noethersches Ersetzungssystem über $\mathbb{M} = \mathbb{M}(\Sigma, D)$. Dann existiert ein Algorithmus, der für eine Spur $t \in \mathbb{M}$ eine Normalform $\hat{t} \in \mathrm{Irr}(\mathcal{R})$, $t \stackrel{*}{\Longrightarrow}_{\mathcal{R}} \hat{t}$, in $\mathcal{O}(d_{\mathcal{R}})$ Zeiteinheiten berechnet.*

Beweis: Wir haben bereits gezeigt, wie ein Faktorvorkommen in konstanter Zeit berechnet werden kann und wie einzelne Buchstaben aus der darstellenden Struktur entfernt bzw. eingefügt werden können. Es reicht also aus einen Algorithmus zu beschreiben, der in konstanter Zeit eine bereits durch einen Vorkommenbaum identifizierte linke Seite einer Regel $l \to r \in \mathcal{R}$ durch die rechte Seite r ersetzt.

Dabei gehen wir wie folgt vor. Zuerst entfernen wir buchstabenweise alle Teile der linken Seite, auf die der Vorkommenbaum verweist. Die zuletzt berechnete Position nutzen wir, um an dieser Stelle die rechte Seite einzufügen. Dieses Vorgehen wird so lange iteriert bis der Resultatterm bzw. Resulatbaum irreduzibel ist.

Wir deklarieren die Operation REPLACE, die das Gewünschte leistet.

```
FUNCTION REPLACE(i, l, r, t)
    Comment Input: a trace representation t = ulv,
                   a tree occurrence i of l in t,
                   and a right-hand side with l → r ∈ R,
            Output: the representation of a trace urv
                    according to the semantic.
```

```
Method
    let p be the position of the minimal letters of the
        occurrence uniquely extended by minimal letters of t
    while l ≠ 1 do
        let  a be a minimal letter in l = al'
        if there is no path in p ending in an a*-labeled vertex
           then extend the position to one pointing on such a leaf
        end_if
        ⟨t, p⟩ ← DELETE(a, t, p)
        l ← l'
    end_while
    let r' be a copy of r
    while r' ≠ 1 do
        let  a be a minimal letter in r' = ar''
        ⟨t, p⟩ ← INSERT(a, t, p)
        r' ← r''
    end_while
    return (t)
end_Method
```

Die Invarianten der Schleifen sind offensichtlich. Bei der ersten while-Schleife kann die die Position p nur auf minimale Buchstaben von l oder minimale Buchstaben von v mit $t = ulv$ verweisen. Es werden nur Buchstaben entfernt, auf die i verweist. Es gilt damit $t = u \cdot l' \cdot v$ wenn $t = u \cdot l \cdot v$ die Eingabe war und i das Vorkommen ist, das diese Faktorisierung induziert. Nach Verlassen der while-Schleife ist die Repräsentation von uv berechnet, und die Position stellt diese Faktorisierung dar. Die zweite while-Schleife besitzt die Invariante $t = u \cdot r''' \cdot v$, wobei r''' der Quotient ${r'}^{-1}r$ ist.

Damit kann in konstanter Zeit ein Derivationsschritt nachvollzogen werden. Wir erhalten die Aussage. ■

Korollar 13 *Sei (Σ, D) ein Cographen-Abhängigkeitsalphabet und $\mathcal{R}$ ein gewichtsreduzierendes Ersetzungssystem über einem Cographen-Monoid $\mathbb{M} = \mathbb{M}(\Sigma, D)$. Dann kann für eine Spur $t \in \mathbb{M}$ eine Normalform $\hat{t} \in \mathrm{Irr}(\mathcal{R})$ in $\mathcal{O}(|t|)$ berechnet werden.* ■

Bemerkung 8 *Mit dieser Version des Reduktionsalgorithmus kann nicht mehr jede Faktorisierung gewählt werden. (Es ist kein Zugriff mehr auf ein beliebiges Listenelement vorhanden.) Auch läßt sich keine Information über die Position der ersetzten Faktorvorkommen relativ zueinander ohne linearen Suchaufwand ermitteln. So kann man beispielsweise nicht zusichern, eine Links-Derivation*

$$\begin{aligned} t = u_1 l_1 v_1 \Longrightarrow_{\mathcal{R}} u_1 r_1 v_1 = u_2 l_2 r_2 \Longrightarrow_{\mathcal{R}} \cdots \\ \cdots \Longrightarrow_{\mathcal{R}} u_{n-1} r_{n-1} v_{n-1} = u_n l_n v_n \Longrightarrow_{\mathcal{R}} \hat{t}, \end{aligned}$$

mit $\hat{t} \in Irr(\mathcal{R})$ und $u_i \in Irr(\mathcal{R})$ für alle $i \in [n]$ zu berechnen.

Wir haben uns in der Darstellung auf Cographen-Monoide eingeschränkt. Aber auch hier ist die Verallgemeinerung auf dekomponierbare Monoide möglich. Das führt zur folgenden Verallgemeinerung von Books Resultat.

Theorem 18 *Sei (Σ, D) ein Abhängigkeitsalphabet und $\mathcal{R}$ ein noethersches Ersetzungssystem über $\mathbb{M} = \mathbb{M}(\Sigma, D)$ so, daß $\mathbb{M}$ dekomponierbar ist. Dann existiert ein Algorithmus, der für eine Spur $t \in \mathbb{M}$ eine Normalform $\hat{t} \in \mathrm{Irr}(\mathcal{R})$, $t \overset{*}{\Longrightarrow}_{\mathcal{R}} \hat{t}$, in $\mathcal{O}(d_{\mathcal{R}})$ Zeiteinheiten berechnet.* ∎

9 Ausblick

Wir haben eine Vielzahl von Reduktionsalgorithmen für Spurersetzungssysteme vorgestellt. Darunter waren sogar einige linear — also optimal. Die auf den C^*-Bäumen basierenden Algorithmen besitzen die etwas schlechtere Zeitkomplexität „$\mathcal{O}(d_{\mathcal{R}} \log d_{\mathcal{R}})$“. Sie sind jedoch wesentlich effizienter als die auf endlichen Automaten basierenden quadratischen Methoden. Man kann sich diese Effizienzsteigerung am Beispiel „Sortieren durch Vergleichen“ veranschaulichen.

Die Verbesserung der Zeitschranke entspricht der Effizienzsteigerung beim Ersetzten von Insertionsort durch beispielsweise Heapsort. Die linearen Reduktionsalgorithmen können beide als eine Verallgemeinerung des Reduktionsalgorithmus von Book betrachtet werden.

Offen bleibt die Existenz von effizienteren bzw. allgemeineren Methoden. In den hier betrachteten Klassen basierte die Lösung auf einer adäquaten Repräsentation von Spuren. Eine Algorithmus, der den allgemeinen unzusammenhängenden Fall auch abdeckt könnte beispielsweise auf Darstellungen, wie sie im Anhang beschrieben sind, aufbauen. Die Kardinalitätsresultate und die Intervallbeschreibungen sind ein Indiz für subquadratische Algorithmen.

Eine Verallgemeinerung des Reduktionsalgorithmus aus Kapitel 5 könnte auf Bäumen mit verteilten Intervallbeschriftungen, die zur Intervallrelation $\mathbb{I}$ korrespondieren, basieren. Vermutlich lassen sich diese Bäume so verwalten, daß die Kombinationsrelation ∇ dynamisch rekonstruierbar bleibt (vermutliche Komplexität $\mathcal{O}(d_{\mathcal{R}} \cdot \log d_{\mathcal{R}} \cdot \log\log d_{\mathcal{R}})$).

Eine auch für die Applikationen nicht zu vernachlässigende Fragestellung ist: Welche konstanten Faktoren verbirgt die $\mathcal{O}$-Notation oder inwieweit sind die angegebenen Resultate für Applikationen, wie sie in der Einführung beschrieben sind, aussagekräftig? Inwieweit sich die vorgestellten Algorithmen in den angedeuteten Bereichen realisieren lassen wird sich zeigen.

Wir schließen mit dem letzten offenen Problem in diesem Kontext, der zentralen Frage nach der Komplexität:

> *Welche* **uniforme** *und* **nicht-uniforme** *Zeitkomplexität besitzt das Normalformenproblem für Spurmonoide?*

A Anhang: Faktoren und Faktorisierungen

In diesem Kapitel sind die Definitionen und Resultate über Faktorisierungen zusammengefaßt. Viele der Sätze wurden bei der Konstruktion der Algorithmen implizit verwendet. Im Abschnitt über allgemeine Faktoren werden Methoden bereitgestellt, die eventuell zu einem effizienten Algorithmus führen, der das allgemeine Normalformenproblem löst.

A.1 Darstellungen von Faktorisierungen

Ein *Faktor* einer Spur $t \in \mathbb{M}(\Sigma, D)$ ist eine Spur $l \in \mathbb{M}(\Sigma, D)$, für die zwei Spuren u und v mit $t = u \cdot l \cdot v$ existieren. Unter einer *Faktorisierung* von t verstehen wir das Tripel $(u, l, v) \in \mathbb{M}(\Sigma, D) \times \mathbb{M}(\Sigma, D) \times \mathbb{M}(\Sigma, D)$ mit $u \cdot l \cdot v = t$.

Wir bezeichnen den Faktor einer Spur $t \in \mathbb{M}(\Sigma, D)$, der aus allen minimalen Buchstaben besteht, mit $\min(t)$ und der, der aus allen maximalen besteht, mit $\max(t)$. Aus der Darstellung von Spuren als partielle Ordnungen erhalten wir eine Charakterisierung von Faktoren.

Lemma 40 *Seien $t, l \in \mathbb{O}(\Sigma, D)$ Spuren mit $t = \langle M_t, \leq_t \rangle$ und $l = \langle M_l, \leq_l \rangle$. Weiter sei $M_l \subseteq M_t$ eine Teilmultimenge und $\leq_l$ die auf M_l induzierte partielle Ordnung $\leq_t \cap (M_l \times M_l)$. Dann ist l ein Faktor in t ($t = ulv$ für $u, v \in \mathbb{M}(\Sigma, D)$) genau dann, wenn für alle Elemente $a, b \in M_l$ und $c \in M_t$ gilt $a \leq_t c \leq_t b \implies c \in M_l$.*

Beweis: Sei l ein Faktor in $t = ulv \in \mathbb{O}(\Sigma, D)$ mit Spuren $u = \langle M_u, \leq_u \rangle$, $v = \langle M_v, \leq_v \rangle$ und $t = \langle M_t, \leq_t \rangle$. Daraus folgt $M_t = M_u \dot\cup M_v \dot\cup M_l$. Seien $a, b \in M_l$ und $c \in M_t$ mit $a \leq_t c \leq_t b$. Dann gilt $a \leq_t c \implies c \notin M_u$ und $c \leq_t b \implies c \notin M_v$. Also liegt c in M_l.

Sei nun $l = \langle M_l, \leq_l \rangle$ eine durch $\leq_t$ partiell geordnete Teilmenge von t so, daß für alle $a, b \in M_l$ $a \leq_l c \leq_l b \implies c \in M_l$ gilt. Liegen alle minimalen und maximalen Elemente von t in M_l, so ist $t = l$ ein Faktor ($u = v = 1$). Anderenfalls existiert ein minimaler (oder ein maximaler) Buchstabe a in t, der nicht in l vorkommt. Damit faktorisiert sich $t = at'$ (oder $t = t'a$) und t' enthält (mit Induktion über die Kardinalität von M_t) den Faktor l. Das heißt $t = au'lv'$ (oder $t = u'lv'a$). Man beachte die Divisionsrelation ist noethersch. ∎

Sei $t = \langle M, \leq_t \rangle$ eine partiell geordnete Multimenge. Dann verstehen wir unter dem durch $N \subseteq M$ induzierten *Ideal* $I_N(t)$ die Spur $i = \langle M_i, \leq_i \rangle$ mit $M_i = \{a \in M \mid \exists b \in N : a \leq_t b\}$ und $\leq_i = \leq_t \cap (M_i \times M_i)$. Dual definieren wir den von N induzierten *Filter* $F_N(t)$ als die Spur $f = \langle M_f, \leq_f \rangle$ mit $M_f = \{a \in M \mid \exists b \in N : b \leq_t a\}$ und $\leq_f = \leq_t \cap (M_f \times M_f)$.

Die Menge der Ideale einer Spur t zusammen mit den Verknüpfungen $I_N(t) \sqcap I_{N'}(t) = I_{N \cap N'}(t)$ und $I_N(t) \sqcup I_{N'}(t) = I_N(t) \cup I_{N'}(t)$ bilden einen endlichen Verband. Dual bilden die Filter einer Spur t zusammen mit den Verknüpfungen $F_N(t) \sqcup F_{N'}(t) = I_{N\cup N'}(t)$ und $F_N(t) \sqcap F_{N'}(t) = F_N(t) \cap F_{N'}(t)$ einen Verband[9].

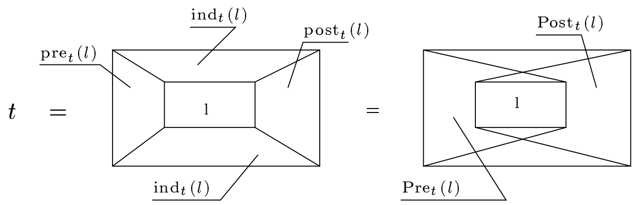
Die Charakterisierung in Lemma 40 führt zu einer durch einen Faktor l induzierten Aufteilung. Sei $l = \langle M_l, \leq_l\rangle$ eine *Subspur* von $t = \langle M_t, \leq_t\rangle \in \mathbb{O}(\Sigma, D)$, d.h. $M_l \subseteq M_t$ und $\leq_l = \leq_t \cap (M_l \times M_l)$. Dann definieren wir

$$\begin{array}{rcl} \mathrm{pre}_t(l) & = & \sqcup_{a\in M_l} I_a(t) \setminus l \\ \mathrm{post}_t(l) & = & \sqcup_{a\in M_l} F_a(t) \setminus l \\ \mathrm{ind}_t(l) & = & \langle M_i, \leq_i\rangle, \end{array} \qquad \begin{array}{rcl} \mathrm{Pre}_t(l) & = & \sqcup_{a\in M_l} I_a(t) \\ \mathrm{Post}_t(l) & = & \sqcup_{a\in M_l} F_a(t) \end{array}$$

wobei die letzte Spur der verbleibenden Teil $((\mathrm{pre}_t(l))^{-1}t(\mathrm{post}_t(l))^{-1})l^{-1}$ von t ist. In der Terminologie der Ideale und Filter läßt sich das vorhergehende Lemma wie folgt formulieren.

Korollar 14 *Seien $t, l \in \mathbb{O}(\Sigma, D)$ Spuren mit $t = \langle M_t, \leq_t\rangle$ und $\langle M_l, \leq_l\rangle$. Weiter sei $M_l \subseteq M_t$ eine Teilmultimenge und $\leq_l$ die auf M_l induzierte partielle Ordnung $\leq_t \cap (M_l \times M_l)$. Dann ist l ein Faktor in t ($t = ulv$ für $u, v \in \mathbb{M}(\Sigma, D)$) genau dann, wenn* $\mathrm{Pre}_t(l) \cap \mathrm{Post}_t(l) = l$ *bzw.* $\mathrm{pre}(l) \cap \mathrm{post}(l) = \emptyset$. ■

Die Definition von $\mathrm{pre}_t(l)$, $\mathrm{post}_t(l)$ und $\mathrm{ind}_t(l)$ ist in der folgenden Graphik illustriert.



Jede Faktorisierung (u, l, v) induziert eine Faktorisierung in $\mathbb{M}((\Sigma, D)(l))$ via Projektionen $(\pi_{\Sigma(l)}(u), l, \pi_{\Sigma(l)}(v))$. Diese Faktorisierungen von $\pi_{\Sigma(l)}(t)$ nennen wir *Vorkommen* von l in t. Jedes durch Faktorisierungen induzierte Vorkommen entspricht einer Vielfalt von Zerlegungen in die Faktoren $\mathrm{pre}_t(l)$, $\mathrm{post}_t(l)$, u', v' und l mit $u'v' = \mathrm{ind}_t(l)$.

Lemma 41 *Für jede Faktorisierung (u, l, v) von t existieren Spuren u' und v' mit* $\mathrm{ind}_t(l) = u'v'$, $u = \mathrm{pre}_t(l)u'$ *und* $v = v'\mathrm{post}_t(l)$.

Existiert umgekehrt eine Subspur $l = \langle M_l, \leq_t \cap (M_l \times M_l)\rangle$ mit $\mathrm{ind}_t(l) = u'v'$ *und* $u = \mathrm{pre}_t(l)u'$, $v = v'\mathrm{post}_t(l)$, *dann ist (u, l, v) eine Faktorisierung.*

Beweis: Aus $t = ulv$ folgt $\mathrm{pre}_t(l) \cap \mathrm{post}_t(l) = \emptyset$. Wir erhalten den Quotienten $w = (\mathrm{pre}_t(l))^{-1}t(\mathrm{post}_t(l))^{-1}$ durch induktives Kürzen. Da l ein Faktor ist, gilt $w = u'lv'$. Insbesondere folgt aus $(u' \cup v') \cap (\mathrm{pre}_t(l) \cup \mathrm{post}_t(l)) = \emptyset$ die Unabhängigkeit von $u'v'$ und l, d.h. $(\Sigma(u') \cup \Sigma(v')) \times \Sigma(l) \subseteq I$. Die andere Implikation ist evident. ■

[9]Diese Eigenschaft ist eine direkte Folgerung aus dem Lemma von Levi.

Im folgenden sei $\mathbb{M} = \mathbb{M}(\Sigma, D)$ das frei partiell kommutative von (Σ, D) erzeugte Monoid. Weiter bezeichne $\mathcal{C}$ eine Cliquenüberdeckung des erzeugenden Alphabets. Aus der Einbettung von $\mathbb{M}(\Sigma, D)$ in das direkte Produkt (freier Monoide) $\prod_{C\in\mathcal{C}} C^*$ erhalten wir

Lemma 42 *Sei $(t_C)_{C\in\mathcal{C}}$ ein Worttupel in $\prod_{C\in\mathcal{C}} C^*$. Dann existiert eine Spur t mit $(t_C)_{C\in\mathcal{C}} = \pi(t)$ genau dann, wenn für alle Cliquen $C, C' \in \mathcal{C}$ die Bedingung $\pi_C(t_{C'}) = \pi_{C'}(t_C)$ erfüllt ist.*

Für eine Clique $C \in \mathcal{C}$ und eine Spur $t \in \mathbb{M}$ verwenden wir für die Projektion $\pi_C(t) \in C^*$ auf das Cliquenalphabet $C \in \mathcal{C}$ die Bezeichnung t_C.

Beweis: Da π_Δ, $\Delta \subseteq \Sigma$, ein Homomorphismus ist und $\pi_\Delta \circ \pi_\Gamma = \pi_{\Delta\cap\Gamma}$ folgt aus $(\pi_C(t))_{C\in\mathcal{C}} = (t_C)_{C\in\mathcal{C}} = \pi(t)$ für alle $C, C' \in \mathcal{C}$ die Gleichheit $\pi_C(\pi_{C'}(t)) = \pi_C(t_{C'}) = \pi_{C'}(t_C) = \pi_{C'}(\pi_C(t))$.

Sei die Bedingung $\pi_C(t_{C'}) = \pi_{C'}(t_C)$ für alle Cliquen $C, C' \in \mathcal{C}$ erfüllt. Gilt $t_C = 1$ für alle $C \in \mathcal{C}$, so ist $t = 1$. Anderenfalls existiert eine Clique $C \in \mathcal{C}$ mit $t_C = at'_C$. Aus der Bedingung folgt, daß für alle Wörter $t_{C'}$ mit $a \in C' \in \mathcal{C}$ gilt $t_{C'} = at'_{C'}$. Damit ist $(t_C)_{C\in\mathcal{C}} = \pi(a) \cdot (t'_C)_{C\in\mathcal{C}}$ und a ein minimaler Buchstabe von t. Mit Induktion existiert eine Spur t' mit $\pi(t') = (t'_C)_{C\in\mathcal{C}}$ und $t = at'$. ■

Als Korollar erhalten wir eine weitere Charakterisierung von Faktoren.

Korollar 15 *Seien $t, l \in \mathbb{M}(\Sigma, D)$. Dann ist $l = \langle M_l, \leq_l\rangle$ ein Faktor in $t = \langle M_t, \leq_t\rangle$ genau dann, wenn*

(i) l ein Vorkommen in t ist, d.h. für alle Cliquen $C \in \mathcal{C}$ mit $C \cap \Sigma(l) \neq \emptyset$ Wörter $u_C, v_C \in C^*$ existieren, mit $t_C = u_C l_C v_C$ und für alle $C, C' \in \mathcal{C}$ mit $C' \cap \Sigma(l) \neq \emptyset$ und $C \cap \Sigma(l) \neq \emptyset$ $\pi_{C'}(u_C) = \pi_C(u_{C'})$ gilt, sowie

(ii) für alle verbleibenden Cliquen $C \in \mathcal{C}$, $C \cap \Sigma(l) = \emptyset$ Faktorisierungen $t_C = u_C v_C$ existieren, so daß für alle $C, C' \in \mathcal{C}$ die Beziehung $\pi_C(u_{C'}) = \pi_{C'}(u_C)$ erfüllt ist. ■

Als Konsequenz ergibt sich aus *(i)* und *(ii)* $\pi_C(v_{C'}) = \pi_{C'}(v_C)$ für alle $C, C' \in \mathcal{C}$. Wir erhalten eine Faktorisierung

$$\left(\pi^{-1}((u_C)_{C\in\mathcal{C}}),\, l,\, \pi^{-1}((v_C)_{C\in\mathcal{C}})\right).$$

Bemerkung 9 *Die Bedingung $\pi_C(t_{C'}) = \pi_{C'}(t_C)$ für alle $C \in \mathcal{C}$ ist äquivalent zu $|t_C|_a = |t_{C'}|_a$ für alle $a \in \mathcal{C} \cap C'$ und für alle $C, C' \in \mathcal{C}$ in diesem Kontext.*

A.2 Ordnungen für Vorkommen

Betrachten wir ein *Vorkommen* (u, l, v), $\pi_{\Sigma(l)}(t) = ulv$, (von l in t) nicht mehr als ein Tupel, sondern als einen *Vorkommen-Morphismus* $i : l \longrightarrow t$ zwischen zwei Spuren.

Dabei soll ein Vorkommen-Morphismus $i : l = (l_C)_{C \in \mathcal{C}} \longrightarrow t = (t_C)_{C \in \mathcal{C}}$ ein Tupel von Wort-Morphismen $(i_C : l_C \longrightarrow t_C)_{C \in \mathcal{C}}$ sein. Ein *Wort-Morphismus* $j : v \longrightarrow w = xvy$ ist eine die Ordnung von v und die Nachbarschaft[10] auf den Projektionen $\pi_{\Sigma(l)}(t_C)$ erhaltende Injektion, die jedem Buchstaben a in v einen Buchstaben a in w zuordnet. Sei $v = v_1 \ldots v_m$, $w = w_1 \ldots w_n$ mit $v_1, \ldots, v_m, w_1, \ldots, w_n \in \Sigma$ und $j(v_1) = w_{k+1}$, $\ldots$, $j(v_m) = w_{k+m}$, dann sind die Wörter $j(v) = j(v_1) \ldots j(v_m)$ und $v = v_1 \ldots v_m$ (als Elemente des freien Monoids Σ^*) identisch. Wir verwenden dieses Konzept, lediglich um verschiedene Vorkommen der gleichen Spur formalisieren zu können.

$$\begin{array}{ccccccc} & & v_1 & \cdots & v_m & & \\ & & \downarrow & \downarrow\downarrow\downarrow & \downarrow & & \\ & & i(v_1) & \cdots & i(v_m) & & \\ w_1 & \cdots & w_{k+1} & \cdots & w_{k+m} & \cdots & w_n \end{array}$$

Da diese Morphismen Teilwörter identifizieren und somit nichts anderes als Positions- beziehungsweise Längenangaben sind, verwenden wir als Namen Indizierungsbezeichner i und j. Ein Vorkommen-Morphismus ist also eine spezielle Inklusion einer Subspur.

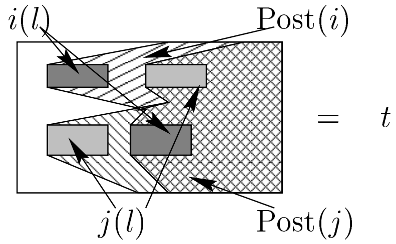
Ein Vorkommen $i : l \longrightarrow t$ ist ein *Faktorvorkommen*, falls $i = (i_C)_{C \in \mathcal{C}}$ ein Vorkommen ist, das eine Faktorisierung induziert. D.h. für alle $C \in \mathcal{C}$ hat t_C die Form $u_C\, i_C(l)\, v_C$ und es existieren Spuren u und v mit $\pi(u) = (u_C)_{C \in \mathcal{C}}$ und $\pi(v) = (v_C)_{C \in \mathcal{C}}$. Ein Faktorvorkommen $i : l \longrightarrow t$ legt somit die Ideale $\mathrm{pre}(i) = \mathrm{pre}_t(i(l))$ und $\mathrm{Pre}(i) = \mathrm{Pre}_t(i(l))$ und die Filter $\mathrm{post}(i) = \mathrm{post}_t(i(l))$ und $\mathrm{Post}(i) = \mathrm{Post}_t(i(l))$ eineindeutig fest.

Bemerkung 10 *In dieser Darstellung wird auch deutlich, daß Vorkommen (sowie Faktorvorkommen) zusammen mit den Spuren (als Objekte) eine Kategorie (Subkategorie) bilden. Dieser Fakt kann dazu verwendet werden, effizienten Faktorisierungsalgorithmen oder Algorithmen, die maximale Überlappungen bzw. kritische Paare berechnen, zu entwerfen.*

Seien $i : l_i \longrightarrow t$ und $j : l_j \longrightarrow t$ zwei Faktorvorkommen. Wir definieren die *Faktorordnung* $\ll$ via

$$i \ll j \quad :\Longleftrightarrow \quad \mathrm{Post}(j) \subseteq \mathrm{Post}(i).$$

Analog kann man eine Ordnung mit dem dualen Operator Pre definieren.



[10] Der Begriff Nachbarschaft setzt eine Topologie auf Worten voraus — die erzeugenden offenen Überdeckungen sind die Infixe. Wir verwenden diesen Begriff, um die kategorielle Struktur des Mustererkennungsproblems (Faktorisierungsproblems) zu unterstreichen. (Die Faktoren als Objekte mit der Faktorrelation als Morphismus ist wieder eine Kategorie.)

Offensichtlich ist $\ll$ nicht notwendigerweise eine lineare Ordnung.

Lemma 43 *Seien $l, t \in \mathbb{M}(\Sigma, D)$ zwei Spuren, dann ist die Relation $\ll$ eine Ordnung auf der Menge der Faktorvorkommen $i : l \longrightarrow t$.*

Beweis: Offensichtlich ist $\ll$ reflexiv. Die Relation $\ll$ ist transitiv, da aus $i \ll j$ und $j \ll k$ folgt $\mathrm{Post}(k) \subseteq \mathrm{Post}(j) \subseteq \mathrm{Post}(i)$ und somit $i \ll k$. Seien $i \ll j$ und $j \ll i$, dann gilt $u = \mathrm{Post}(i) = \mathrm{Post}(j)$. Da sowohl i als auch j Faktorvorkommen sind, gilt $u = lu'$ und kein Buchstabe a aus u' ist bezüglich der partiellen Ordnung $\leq_t$ von $t = \langle M_t, \leq_t \rangle$ kleiner als ein Buchstabe aus $l = \langle M_l, \leq_l \rangle$. Anderenfalls wäre $\mathrm{Pre}_t(l) \cap \mathrm{Post}_t(l) \neq l$ und damit i kein Faktorvorkommen. Alle minimalen Buchstaben $\min \mathrm{Post}_t(l)$ liegen in $i(l)$ bzw. $j(l)$. Mit Induktion über die Anzahl der Elementarfaktoren von l erhalten wir $i = j$ und damit die Antisymmetrie von $\ll$. ∎

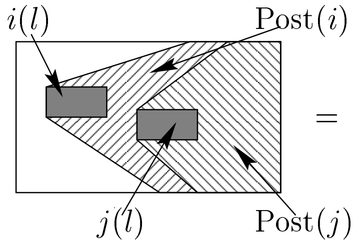
A.3 Zusammenhängende Faktoren

Wir bezeichnen eine Spur t als *zusammenhängend*, wenn der Abhängigkeitsgraph zusammenhängend ist oder äquivalent, wenn das induzierte Abhängigkeitsalphabet $(\Sigma, D)(l)$ ein zusammenhängender Graph ist.

Wir analysieren in diesem Abschnitt die relativen Beziehungen zwischen Vorkommen bzw. Faktorvorkommen.

Seien $i, j : l \longrightarrow t$ zwei Vorkommen von einer zusammenhängenden Spur l. Wir definieren die *Vorkommenordnung* $\lll = \lll_{(l,t)}$ analog via

$$i \lll j \quad :\Longleftrightarrow \quad \mathrm{Post}(j) \subseteq \mathrm{Post}(i).$$



Lemma 44 *Seien $l, t \in \mathbb{M}(\Sigma, D)$ zwei Spuren und l zusammenhängend. Dann ist die Relation $\lll = \lll_{(l,t)}$ eine lineare Ordnung.*

Beweis: Die Reflexivität und die Transitivität folgt aus den vorhergehenden Betrachtungen. Die Linearität zeigen wir induktiv. Seien $i, j : l \longrightarrow t$ zwei Faktorvorkommen (l zusammenhängend). Existieren zwei Buchstaben a und b in $l = \langle M_l, \leq_l \rangle$ mit $a \leq_l b$, so gilt auch $i(a) \leq_t i(b)$, da i ordnungserhaltend ist. Sind a, b zwei Buchstaben so, daß kein Buchstabe $c \in \Sigma(l)$ mit $a \leq_l c \leq_l b$ existiert, so existiert auch kein Buchstabe c in t

mit $i(a) \leq_t c \leq_t i(b)$. Würde ein Buchstabe c in t existieren mit $i(a) \leq_t c \leq_t i(b)$, dann existiert in der Projektion

$$\pi_{\{a,b,c\}}(u,l,v) \quad = \quad \Big(\pi_{\{a,b,c\}}(u), \pi_{\{a,b,c\}}(l), \pi_{\{a,b,c\}}(v)\Big)$$

ein c zwischen $i(a)$ und $i(b)$, $\pi_{\{a,b,c\}}(t) = w\, i(a)\, x\, c\, y\, i(b)\, z$, im Widerspruch zu $\pi_{\{a,b,c\}}(l) = \alpha\pi_{\{a,b,c\}}(l)\beta$ mit $\alpha, \beta \in \{a,b,c\}^*$ ($\mathrm{pre}_{\pi_{\{a,b,c\}}(i(l))}(i(a)) = \alpha$ und $\mathrm{post}_{\pi_{\{a,b,c\}}(i(l))}(i(b)) = \beta$).

Behauptung: Sei $a <_l b$, a, b in M_l, mit $(a,b) \in D$, so daß kein Buchstabe c mit $a <_l c <_l b$ existiert und $i \lll j$. Dann gilt

$$i(a) \leq_t j(a) \implies i(b) \leq_t j(b) \quad \text{und} \quad i(b) \leq_t j(b) \implies i(a) \leq_t j(a).$$

Aus der Voraussetzung folgt die Vergleichbarkeit von $i(a)$, $i(b)$, $j(a)$ und $j(b)$. Sei $i(a) \leq_t j(a)$ aber $j(b) \leq_t i(b)$. Damit gilt $i(a) \leq_t j(a) \leq_t j(b) \leq_t i(b)$. Das steht im Widerspruch zur Nichtexistenz eines Buchstaben c zwischen a und b, falls $i(a) \neq j(a)$ und $i(b) \neq j(b)$.

$$\begin{tikzcd}
i(a) \arrow[rr] \arrow[dr] \arrow[drrr] & & j(a) \arrow[dl] \arrow[dr] & \\
& i(b) & & j(b) \arrow[ll]
\end{tikzcd}$$

Für $i(a) = j(a)$ und $i(b) = j(b)$ ist die Implikation trivial. Analog folgert man aus $j(a) \leq_t i(a)$ und $i(b) \leq_t j(b)$ einen Widerspruch.

Ist $l = a$ ein einzelner Buchstabe, so ist $\lll$ eine lineare Ordnung, da die Abhängigkeitsrelation D reflexiv ist. Da l zusammenhängend ist, existiert zu jedem Buchstaben b in l ein ungerichteter Pfad

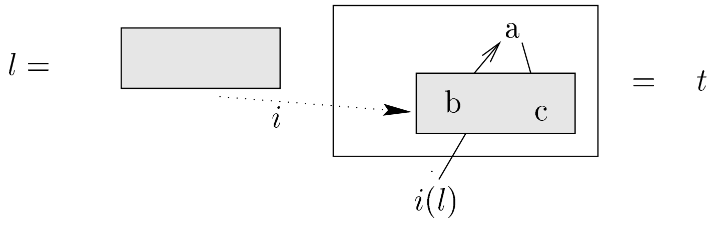
$$a = a_1 \gtrless_1 a_2 \gtrless_2 \dots \gtrless_{n-1} a_n = b,$$

mit $\gtrless_k \in \{<_l, >_l\}$ und $(a_k, a_{k+1}) \in D$, $k \in [n-1]$. Wir bezeichnen die duale Ordnung zu $\leq$ mit $\geq$. Mit dem vorhergehenden Resultat schließen wir mit Induktion über die Länge n des Pfades

$$\begin{array}{ccccccc}
i(a_1) & \gtrless'_1 & i(a_2) & \gtrless'_2 & \dots & \gtrless'_{n-1} & i(a_n) \\
\gtrless & & \gtrless & & & & \gtrless \\
j(a_1) & \gtrless'_1 & j(a_2) & \gtrless'_2 & \dots & \gtrless'_{n-1} & j(a_n),
\end{array}$$

mit $\gtrless'_k, \gtrless \in \{\leq_t, \geq_t\}$, $k \in [n-1]$. Damit gilt $\mathrm{Post}(i) \subseteq \mathrm{Post}(j)$ oder $\mathrm{Post}(j) \subseteq \mathrm{Post}(i)$ und $\lll$ ist eine lineare Ordnung. ■

Seien $t, l \in \mathbb{M}(\Sigma, D)$ zwei Spuren. Wir sagen, ein Buchstabe a in $t = \langle M_t, \leq_t \rangle$ *beeinträchtigt* ein Vorkommen $i : l \longrightarrow t$, falls a nicht zu $i(l)$ gehört und zwei Buchstaben b, c in l mit $i(b) \leq_t a \leq_t i(c)$ existieren. Wir notieren diesen Zusammenhang mit $a \rhd i$.



Sei $a \in \Sigma$ und sei $i : l \longrightarrow t$, $l, t \in \mathbb{M}(\Sigma, D)$, ein Vorkommen. Wir bezeichnen mit $(a) \rhd i$ die Menge der das Vorkommen i *beeinträchtigenden Buchstaben* a in t, $\{a \in M_t \mid \exists b, c \in M_l : i(b) \leq_t a \leq_t i(c), a \notin i(l)\}$. Die Teilmenge aller Buchstaben $a \in \Sigma$, die in $t = \langle M_t, \leq_t \rangle$ vorkommen, ist durch $\leq_t$ linear geordnet.

Lemma 45 *Seien $l, t \in \mathbb{M}(\Sigma, D)$ Spuren und i ein Vorkommen $i : l \longrightarrow t$. Dann ist die Menge $(a) \rhd i$ ein Intervall bezüglich $\leq_t$, $t = \langle M_t, \leq_t \rangle$.*

Beweis: Seien a_1, a_2 und a_3 drei (Vorkommen des) Buchstaben a in $t = \langle M_t, \leq_t \rangle$ mit $a_1 \leq_t a_2 \leq_t a_3$, und das Vorkommen $i : l \longrightarrow t$ werde von den Buchstaben a_1 und a_3 beeinträchtigt. (In Zeichen: $a_1 \rhd i$ und $a_3 \rhd i$.) Dann beeinträchtigt auch der Buchstabe a_2 das Vorkommen i, denn aus $a_1 \rhd i$ folgt die Existenz eines Buchstaben b in $l = \langle M_l, \leq_l \rangle$ mit $i(b) \leq_t a_1$ und aus $a_3 \rhd i$ folgt die Existenz eines Buchstaben c in l mit $a_3 \leq_t i(c)$. Zusammengefaßt beeinträchtigt a_2 das Vorkommen i via

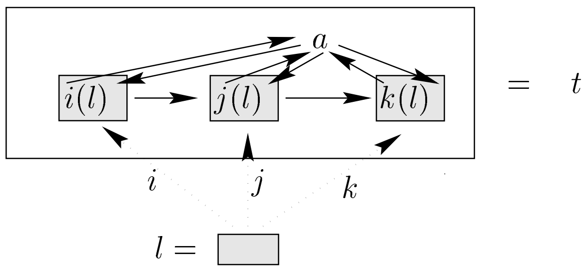
$$i(b) \leq_t a_1 \leq_t a_2 \leq_t a_3 \leq i(c).$$

∎

Analog zur Menge der beeinträchtigenden Buchstaben bezeichnen wir mit $a \rhd (l, t)$ die Menge der *beeinträchtigten Vorkommen* von l in t, $\{i : l \longrightarrow t \mid a \rhd i\}$.

Lemma 46 *Seien $l, t \in \mathbb{M}(\Sigma, D)$ Spuren und l zusammenhängend. Dann ist die Menge $a \rhd (t, l)$ ein Intervall bezüglich $\lll = \lll_{(l,t)}$.*

Beweis: Wir leiten aus der Existenz dreier Vorkommen $i, j, k : l \longrightarrow t$ und eines Buchstabens a in t mit $i \lll j \lll k$ und $a \rhd i$, $a \rhd k$ aber $a \not\rhd j$ einen Widerspruch her. Aus $a \rhd k$ folgt die Existenz eines minimalen Buchstaben b in l mit $k(b) \leq_t a$. Da $i \lll k$ gilt $j(b) \leq_t k(b) \leq_t a$. Aus $a \rhd i$ folgt die Existenz eines maximalen Buchstaben c in l mit $a \leq_t i(c)$, und aus $i \lll j$ folgt $a \leq_t i(c) \leq_t j(c)$. Da $a \notin \Sigma(l)$, beeinträchtigt der Buchstabe a das Vorkommen j, $j(b) \leq_t a \leq_t k(c)$, ein Widerspruch.

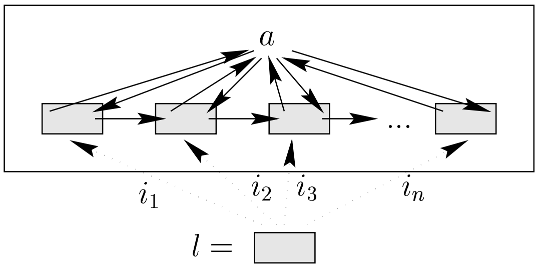


∎

Interessanterweise ist die Anzahl der von einem Buchstaben beeinträchtigten durch eine von $|t|$ unabhängige Konstante beschränkt.

Lemma 47 *Seien $l, t \in \mathbb{M}(\Sigma, D)$ zwei Spuren und a ein Buchstabe in t. Dann ist die Kardinalität der Menge $a \rhd (l, t)$ der beeinträchtigten Vorkommen durch die Konstante $|l| \cdot |\Sigma|$ beschränkt.*

Beweis: Wir führen die Annahme $|a \rhd (l,t)| > |l| \cdot |\Sigma|$ zu einem Widerspruch. Sei $|a \rhd (l,t)| > |l| \cdot |\Sigma|$, dann existieren mindestens $n \geq |\Sigma|$ paarweise disjunkte Vorkommen $i_1 \lll i_2 \lll \ldots \lll i_n$. Zwei Vorkommen $i, j : l \longrightarrow t$ sind disjunkt, falls $i(l) \cap j(l) = \emptyset$. Wäre $n < |\Sigma|$, so würde ein Buchstabe b in l und zwei Vorkommen $i, j \in \{i_1, \ldots, i_n\}$ existieren mit $i \neq j$ und $i(b) = j(b)$, ein Widerspruch (siehe Beweis zu Lemma 44). Sind $i_1, \ldots, i_n$ disjunkte Vorkommen, so folgt aus Lemma 44, für alle Buchstaben d in l und für alle $i, j \in \{i_1, \ldots i_n\}$, $i(d) <_t j(d)$, für $i \neq j$ und $i \lll j$.



$= \quad t$

Aus $a \rhd i_n$ folgt die Existenz eines Buchstaben b in l mit

$$i_1(b) <_t i_2(b) <_t \ldots <_t i_n(b) <_t a,$$

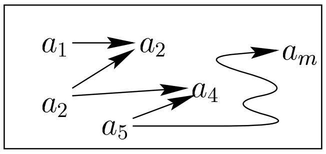
und aus $a \rhd i_1$ folgt die Existenz eines Buchstaben c in l mit

$$a <_t i_1(c) <_t i_2(c) <_t \ldots <_t i_n(c).$$

Da l zusammenhängend ist, existiert ein ungerichteter Pfad minimaler Länge

$$b = a_1 \gtrless_1 a_2 \gtrless_2 \ldots \gtrless_{m-1} a_m = c$$

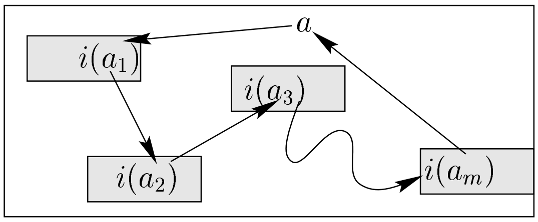
von b nach c, $\gtrless_k \in \{<_l, >_l\}$ für alle $k \in [m-1]$ im Abhängigkeitsgraphen von $l = \langle M_l, \leq_t \rangle$ mit $(a_k, a_{k+1}) \in D$ für alle $k \in [n-1]$, $n \leq |\Sigma|$.



$= \quad l$

Da m minimal ist, kann kein Buchstabe in $\{a_1, \ldots, a_{n-1}\} \subseteq M_l$ (Knoten im zugehörigen Abhängigkeitsgraphen) mehrmals vorkommen, d.h. $k < |\Sigma|$. Damit erhalten wir einen Widerspruch zur Antisymmetrie

$$a \leq_t i_1(a_m) <_t i_2(a_{m-1}) <_t \ldots <_t i_{n-1}(a_2) <_t i_n(a_1) \leq_t a.$$



$= \quad t$

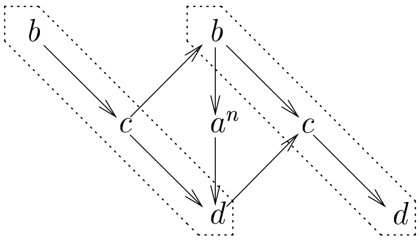
■

Die Kardinalitäten der Mengen $a \rhd i$ für einen Buchstaben $a \in \Sigma$ einer Spur $t \in \mathbb{M}(\Sigma, D)$ mit den Vorkommen $i, j : l \longrightarrow t$, $t \in \mathbb{M}(\Sigma, D)$, sowie die Kardinalitäten von Intervallschnitten $(a \rhd i) \cap (a \rhd j)$ sind nicht durch eine von t unabhängige Funktion beschränkt, wie man folgendem Beispiel entnehmen kann.

Beispiel 1 *Betrachten wir das Abhängigkeitsalphabet*

$$(\Sigma, D) \quad = \quad \left(\{a, b, c, d\}, \begin{bmatrix} a & — & d \\ | & & | \\ b & — & c \end{bmatrix} \right).$$

Sei $l, t \in \mathbb{M}(\Sigma, D)$ *mit* $l = [bcd]$ *und* $t = [bc][b][a^n][d][cd]$, $|t| = \mathcal{O}(n)$. *Es existieren zwei Vorkommen* $i, j : l \longrightarrow t$ *und es gilt* $|a \rhd i| = |a \rhd j| = |(a \rhd i) \cap (a \rhd j)| = n$. *Die durch die Buchstaben a beeinträchtigten Vorkommen sind in der folgenden Graphik durch die Polygonzüge gekennzeichnet.*



Die Schnittlemmata stellen eine Beziehung zwischen Faktorvorkommen zusammenhängender Spuren und Faktorisierungen bereit.

Lemma 48 (Schnittlemma A) *Sei* $t = t_1 \cdot t_2 \in \mathbb{M}(\Sigma, D)$, $l \in \mathbb{M}(\Sigma, D)$ *eine zusammenhängende Spur und* $i : l \longrightarrow t$ *ein Faktorvorkommen. Dann folgt aus* $i(l) \cap (\max(t_1) \cup \min(t_2)) = \emptyset$ *entweder* $i(l) \cap t_1 = \emptyset$ *oder* $i(l) \cap t_2 = \emptyset$.

Beweis: Sei $t = \langle M_t, \leq_t \rangle$. Angenommen $i(l) \cap t_1 \neq \emptyset$ und $i(l) \cap t_2 \neq \emptyset$. Da $t = t_1 \cdot t_2$ gilt für alle Buchstaben a_1 in t_1 und b_2 in t_2 mit $(a_1, b_2) \in D$ die Relation $a_1 \leq_t b_2$, und da l zusammenhängend ist, existiert ein maximaler Buchstabe a in $\max(t_1)$ und ein minimaler Buchstabe b in $\min(t_2)$ mit $(a, b) \in D$ und zwei Buchstaben c in $i(l) \cap t_1$ und d in $i(l) \cap t_2$ mit $c \leq_t a \leq_t b \leq_t d$. Damit ist $i : l \longrightarrow t$ aber kein Faktorvorkommen, ein Widerspruch (a oder b beeinträchtigen i). ∎

Umgekehrt kann man schließen:

Lemma 49 (Schnittlemma B) *Sei* $t = t_1 \cdot t_2 \in \mathbb{M}(\Sigma, D)$, $l \in \mathbb{M}(\Sigma, D)$ *eine zusammenhängende Spur und sei* $i : l \longrightarrow t$ *ein Faktorvorkommen mit* $i(l) \cap t_1 \neq \emptyset$ *und* $i(l) \cap t_2 \neq \emptyset$. *Dann gilt* $i(l) \cap \max(t_1) \neq \emptyset$ *und* $i(l) \cap \min(t_2) \neq \emptyset$.

Beweis: Angenommen $i(l) \cap \max(t_1) = \emptyset$ und $i(l) \cap \min(t_2) = \emptyset$. Da $i(l) \cap t_1 \neq \emptyset$, existiert ein Buchstabe a in l mit $i(a)$ in t_1 und da $i(l) \cap t_2 \neq \emptyset$, existiert ein Buchstabe b in l mit $i(b)$ in t_2. Da l zusammenhängend ist, existiert ein ungerichteter Pfad von a nach b in l, also auch von $i(a)$ nach $i(b)$ in t. Aber dieser Pfad enthält (mindestens) einen Buchstaben aus $\max(t_1)$ und einen aus $\min(t_2)$ im Widerspruch zu $i(l) \cap \max(t_1) = \emptyset$ und $i(l) \cap \min(t_2) = \emptyset$. ∎

A.4 Allgemeine Faktoren

In diesem Abschnitt werden einige Eigenschaften von Faktorisierungen und Techniken zur Faktorisierung und deren Darstellung entwickelt.

Wir sagen, daß die Vorkommen $i : u \longrightarrow t$ und $j : v \longrightarrow t$ *kombinierbar* sind, wenn kein Pfad im Abhängigkeitsgraph von t gleichzeitig in $i(u)$ beginnt und $j(v)$ endet oder in $j(v)$ beginnt und in $i(u)$ endet. D.h. es existiert kein Buchstabe a in u und b in v mit $i(a) \leq_t j(b)$ oder $j(b) \leq_t i(a)$. Wir notieren den reflexiven Abschluß der Kombinationsrelation mit $i \triangledown j$.

Eine *Zusammenhangskomponente* $u = \langle M_u, \leq_u \rangle$ von $v = \langle M_v, \leq_v \rangle$ ist eine Subspur deren Abhängigkeitsgraph ein maximaler zusammenhängender Untergraph des Abhängigkeitsgraphen von v ist. Jede Spur $l \in \mathbb{M}(\Sigma, D)$ zerfällt in ihre Zusammenhangskomponenten $\mathrm{Con}(l) = \{l_1, \ldots, l_n\}$. Es gilt für jede Permutation $\pi : [n] \longrightarrow [n]$ die Identität $l = l_{\pi(1)} \cdot \ldots \cdot l_{\pi(n)}$. Seien $i_k : l_k \longrightarrow t$ Faktorvorkommen. Wir sagen, daß das Tupel $(i_1, \ldots, i_n)$ *kombinierbar* ist, wenn alle Komponenten paarweise kombinierbar sind, d.h. wenn für alle $k, k' \in [n]$ $i_k \triangledown i_{k'}$ gilt und notieren

$$\nabla \quad = \quad \{(i_1, \ldots, i_n) \mid i_k : l_k \longrightarrow t, i_k \triangledown i_{k'} \, (k, k' \in [n])\}.$$

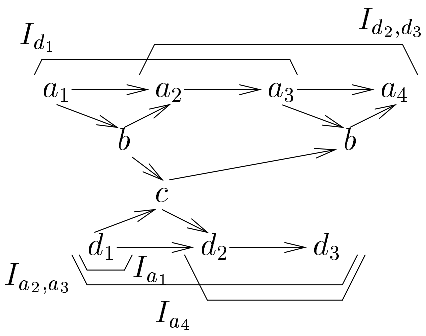
Mit der Charakterisierung in Lemma 40 ist $i : l \longrightarrow t$ mit $i(a) = i_k(a)$ für $a \in l_k$ genau dann ein Faktorvorkommen, wenn $(i_1, \ldots, i_n) \in \nabla$. Die Relation $\triangledown$ besitzt eine Intervalleigenschaft.

Lemma 50 *Sei $l \in \mathbb{M}(\Sigma, D)$ mit $u, v \in \mathrm{Con}(l)$ und $i : u \longrightarrow t$ ein Vorkommen von u. Dann ist die Menge $\triangledown(v, i) = \{j : v \longrightarrow t \mid i \triangledown j\}$ ein Intervall bezüglich $\lll$.*

Beweis: Seien $j_1, j_2, j_3 : v \longrightarrow t$ Vorkommen in t mit $j_1 \triangledown i$, $j_3 \triangledown i$ und $j_1 \lll j_2 \lll j_3$. Wir leiten einen Widerspruch aus $\neg j_2 \triangledown i$ her. Existiert ein Buchstabe a in v und ein Buchstabe b in $i(u)$ mit $j_2(a) \leq_t b$, so gilt $j_1(b) \leq_t j_2(b) \leq_t a$, im Widerspruch zu $j_1 \triangledown i$. Existiert ein Buchstabe c in v und ein Buchstabe b in $i(u)$ mit $b \leq_t j_2(c)$, so gilt $b \leq_t j_2(c) \leq_t i_3(c)$, im Widerspruch zu $j_3 \triangledown i$. Damit gilt $i \triangledown j_2$ ■

Das folgende Beispiel illustriert diese Intervalleigenschaft.

Beispiel 2 *Sei $(\Sigma, D) = [a-b-c-d]$ und $l = [ad]$, $\mathrm{Con}(l) = \{a, d\}$. Wir indizieren die Buchstaben und identifizieren ein (Faktor-)vorkommen $i : a \longrightarrow a_i$ mit dem indizierten Buchstaben a_i. In der Spur $t = [abadcdabad]$ sind die Intervalle $\{a_i : a_i \longrightarrow t \mid a_i \triangledown d_j\}$ mit I_{d_j} gekennzeichnet, entsprechend ist $I_{a_i} = \{d_j : d_j \longrightarrow t \mid d_j \triangledown a_i\}$.*



Die Menge der kombinierbaren Vorkommen berechnet sich zu

$$\nabla \quad = \quad \bigcup_{k\in[4]} (\{a_k\} \times I_{a_k}) \cup \bigcup_{k\in[3]} (\{d_k\} \times I_{d_k}).$$

Diese Menge kann polynomiell viele Elemente enthalten. Sei $k = |\mathrm{Con}(l)|$ die Anzahl der Zusammenhangskomponenten, so gilt $|\nabla| = \mathcal{O}(|t|^k)$. Betrachten wir beispielsweise $\Sigma = \{a_i \mid i \in [k]\}$, $\mathbb{M} = \mathbb{N}^\Sigma$, $t = (a_i^n)_{i\in[k]}$, $l = (a_i)_{i\in[k]}$, $|\nabla| = n^k$.

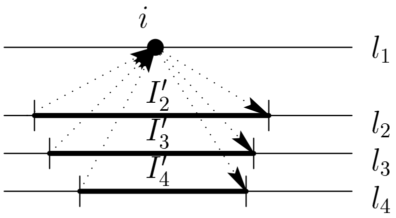
Diese Beobachtung motiviert die nächste Definition. Sei t, l und $\triangledown$ wie oben, dann definieren wir für ein Intervall I von Vorkommen $i : l_{k'} \longrightarrow t$, die Intervalle

$$\nabla(I, l_k) \quad = \quad \{i_k : l_k \longrightarrow t \mid \forall i \in I : i_k \nabla i\},$$

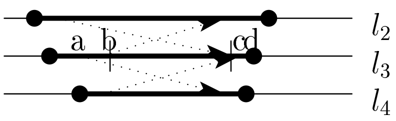
für alle $k \in [n]$ und das Intervalltupel $(I_1, \dots, I_n)(i) = (I_1(i), \dots, I_n(i))$ durch

$$\begin{aligned}
I'_k(i) &= \nabla(\{i\}, l_k), \quad k \in [n] \setminus \{k'\} \\
I_k(i) &= \bigcap_{k''\in[n]\setminus\{k,k'\}} \nabla(I'_{k''}(i), l_k) \cap I'_k(i), \quad k \in [n] \setminus \{k'\} \\
I_{k'}(i) &= \bigcap_{k''\in[n]\setminus\{k'\}} \nabla(I_{k''}(i), l_{k'}).
\end{aligned}$$

Die folgende Graphik illustriert das Vorgehen. Sei $\mathrm{Con}(l) = \{l_1, l_2, l_3, l_4\}$ und $i : l_1 \longrightarrow t$.



Im ersten Bild ist dargestellt, wie die Intervalle I'_x festgelegt sind. Sie werden von den letzten Vorkommen, die vor i, und den ersten Vorkommen, die nach i liegen, links bzw. rechts beschränkt.



Im zweiten Bild wird das Intervall I_{l_3} bestimmt. Dabei ist

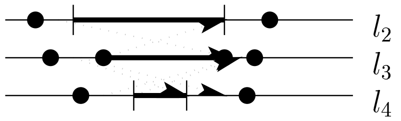
a das größte l_3-Vorkommen, das vor einem Vorkommen in I'_{l_4} liegt,

b das größte l_3-Vorkommen, das vor einem Vorkommen in I'_{l_2} liegt,

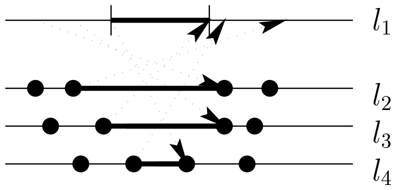
c das kleinste l_3-Vorkommen, das nach einem Vorkommen in I'_{l_4} liegt,

d das kleinste l_3-Vorkommen, das nach einem Vorkommen in I'_{l_2} liegt.

Das Intervall, das dadurch bestimmt wird, ist berandet durch das Maximum der größten Elemente und das Minimum der kleinsten. Im nächsten Bild ist die Berechnung der verbleibenden I'_x-Intervalle skizziert.



Der letzte Schritt ist die Berechnung des l_1-Intervalls $I_{l_1}(i)$, das durch die Intervalle I'_x erzeugt wird.



Sei $\mathrm{Con}(l) = \{l_1, \ldots, l_n\}$, dann definieren wir die *Intervallrelation*

$$\mathbb{I}(l,t) \quad = \quad \{(I_1, \ldots, I_n)(i) \mid i : l_k \longrightarrow t,\ \forall k' \in [n] : \emptyset \neq I_{k'} = I_{k'}(i)\}.$$

Lemma 51 $\mathbb{I}(l,t)$ *besitzt höchstens* $|t|$ *Elemente, und es gilt*

$$(I_1, \ldots, I_n) \in \mathbb{I} \quad \Longrightarrow \quad I_1 \times \ldots \times I_n \subseteq \nabla$$

und für alle kombinierbaren Tupel $(i_1, \ldots, i_n) \in \nabla$ *existiert ein Intervalltupel*

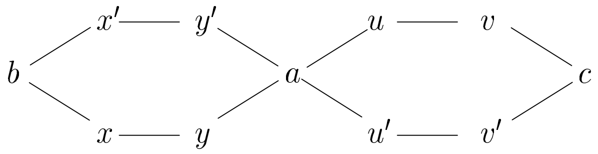
$$(I_1, \ldots, I_n) \in \mathbb{I} \text{ mit } (i_1, \ldots, i_n) \in I_1 \times \ldots \times I_n.$$

Beweis: Sei $(I_1, \ldots, I_n)(i) = (I_1, \ldots, I_n) \in \mathbb{I}$ und $(i_1, \ldots, i_n) \in I_1 \times \ldots \times I_n$ mit $(i_1, \ldots, i_n) \notin \nabla$. Ohne Einschränkung sei $i : l_1 \longrightarrow t$. Dann existieren $k, k' \in [n]$ und zwei Buchstaben, a in l_k und b in $l_{k'}$, mit $i_k(a) \leq_t i_{k'}(b)$ oder $i_{k'}(b) \leq_t i_k(a)$. Ohne Einschränkung sei $i_k(a) \leq_t i_{k'}(b)$. Dann ist $i_k \neq i$ und $i_{k'} \neq i$, da $I_k \subseteq I'_k = \nabla(\{i\}, l_k)$ bzw. $I_{k'} \subseteq I'_{k'} = \nabla(\{i\}, l'_k)$. Also gilt $k \neq 1 \neq k'$. Aber das steht im Widerspruch zu $I_k \subseteq \nabla(I'_{k'}, l_k)$ bzw. $I_{k'} \subseteq \nabla(I'_k, l_{k'})$.

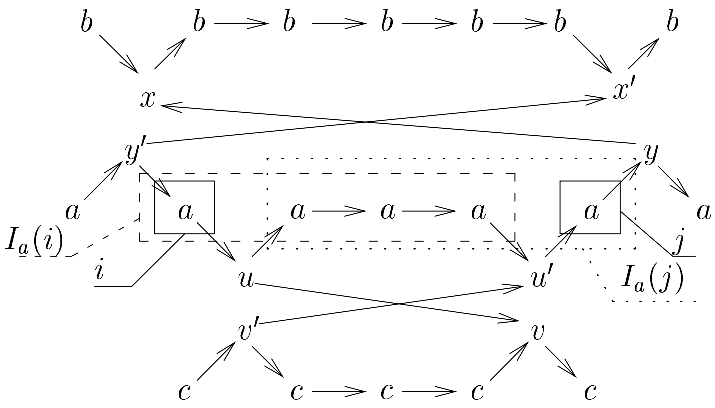
Sei $(i_1, \ldots, i_n) \in \nabla$. Dann ist auch $(I_1, \ldots, I_n)(i_1) \in \mathbb{I}$. Da für alle $k, k' \in [n]$ $i_k \nabla i_{k'}$ gilt $(i_1, \ldots, i_n) \in (I_1, \ldots, I_n)(i_1) \in \mathbb{I}$. ■

Beispiel 3 *Sei* $\mathbb{M} = \mathbb{M}(\Sigma, D)$ *das durch das Abhängigkeitsalphabet*

$(\Sigma, D) \quad =$



erzeugte Monoid. Sei $l = [abc]$ *und* t *die Spur*



Dann ist $\mathrm{Con}(l) = \{a, b, c\}$ *und das Vorkommen* i *(mit einem Quadrat in der Abbildung gekennzeichnet) induziert das durch das gestrichelte Polygon berandete Intervall* $I_a(i)$*. Das Vorkommen* j *erzeugt das durch das gepunktete Polygon berandete Intervall* $I_a(j)$*.*

Sei (Σ, D) ein Abhängigkeitsalphabet. Ein *Sprungwort* von $a \in \Sigma$ nach $b \in \Sigma$ ist ein Wort $w \in \Sigma^*$ mit $w = w_1 w_2 \ldots w_n$, $w_i \in \Sigma$, $w_i \neq w_j$ für $i \neq j$, $w_1 = a$, $w_n = b$ und $(w_i, w_{i+1}) \in D$ für alle $i \in [n-1]$. In einer Spur $t = \langle M_t, \leq_t \rangle$ ist $(s_1, s_2, \ldots, s_n)$ eine *Vorwärts-Sprungsequenz* von s_1 nach s_n zum Sprungwort w, falls $w_i = s_i \in M_t$ für alle $i \in [n]$ ist und s_{i+1} der bzgl. $\leq_t$ kleinste Buchstabe w_{i+1} mit $s_i <_t s_{i+1}$ für alle $i \in [n-1]$ ist. Analog ist $(s_1, s_2, \ldots, s_n)$ eine *Rückwärts-Sprungsequenz*, wenn s_{i+1} der bzgl. $\leq_t$ größte Buchstabe w_{i+1} mit $s_{i+1} <_t s_i$ ist, $i \in [n-1]$. Die Anzahl der Vorwärts-Sprungsequenzen bzw. Rückwärts-Sprungsequenzen von x nach y bzw. die der Sprungwörter ist beschränkt durch die Anzahl der doppelpunktfreien Pfade $(x, p_1, \ldots, y)$ in (Σ, D).

Beispiel 4 *Es gibt vier Sprungwörter von* b *nach* c *im obigem Beispiel:*

$$\begin{array}{ll} b\,xy\,a\,uv\,c & b\,xy\,a\,u'v'\,c \\ b\,x'y'\,a\,uv\,c & b\,x'y'\,a\,u'v'\,c \end{array}$$

Eine Vorwärts-Sprungsequenz führt beispielsweise von dem ersten c *in* t *über* v' *und* u'*, dem vorletzten* a*,* y *und* x *zum zweiten* b*.*

Ein einfaches Induktionsargument zeigt:

Lemma 52 *Sei $(M_t, \leq_t) = t \in \mathbb{M}(\Sigma, D)$. Dann existiert für $a, b \in \Sigma$, $a \neq b$ ein Sprungwort w und eine Vorwärts-Sprungsequenz $(s_1, \dots, s_n)$ mit $s_1 = a$, $s_n = b_1$ von a nach b_1 genau dann, wenn kein $b_2 \in M_t$ existiert mit $a \leq b_2 \leq_t b_1$.*

Dual existiert für $a, b \in \Sigma$, $a \neq b$ ein Sprungwort w und eine Rückwärts-Sprungsequenz $(s_1, \dots, s_n)$ mit $s_1 = a$, $s_n = b_1$ von a nach b_1 genau dann, wenn kein $b_2 \in M_t$ existiert mit $b_1 \leq_t b_2 \leq_t a$. (b_1 und b_2 unterscheiden Mehrfachvorkommen von b in $\mathbb{M}_t$.) ∎

Sprungwörter bzw. Sprungsequenzen ermöglichen es, die Faktoreigenschaft von Vorkommen bzw. die Kombinierbarkeit direkt zu entscheiden.

Korollar 16 *Seien $l, t \in \mathbb{M}(\Sigma, D)$ und sei $\mathrm{Con}(l) = \{l_1, \dots, l_n\}$. Weiter seien $i : l_k \longrightarrow t$ und $j : l_{k'} \longrightarrow t$ Faktorvorkommen. Dann existiert für $a \in \Sigma(l_k)$ und $b \in \Sigma(l_{k'})$ ein Sprungwort w und eine zugehörige*

- *Vorwärts-Sprungsequenz von $i(a)$ nach b' mit $b' \leq_t j(b)$ oder*
- *Vorwärts-Sprungsequenz von $j(b)$ nach a' mit $a' \leq_t i(a)$ oder*
- *Rückwärts-Sprungsequenz von $j(b)$ nach a' mit $i(a) \leq_t a'$ oder*
- *Rückwärts-Sprungsequenz von $i(a)$ nach b' mit $j(b) \leq_t b'$*

genau dann, wenn $l_k \not\triangledown l_{k'}$. ∎

Die Ordnungen $\lll_{l_k}$, $k \in [n]$, induzieren auf Intervallen $I, I' \subseteq \{i_k \mid i_k : l_k \longrightarrow t\}$ durch

$$I = [i, j] \lll_{l_k} [i', j'] = I' \quad \Longleftrightarrow \quad i \lll_{l_k} i' \ \wedge \ j \lll_{l_k} j'$$

eine Ordnung. Wir notieren $(I_1, \dots, I_n)(i) \lll (I_1, \dots, I_n)(j)$, wenn $I(l_k, i) = I_k(i) \lll_{l_k} I_k(j) = I(l_k, j)$ für alle $k \in [n]$ gilt.

Lemma 53 *Seien $i, j : l_k \longrightarrow t$ mit $i \lll_{l_k} j$, dann gilt*

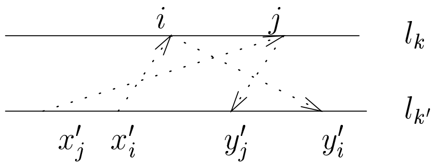
$$i \lll_{l_k} j \quad \Longrightarrow \quad \nabla(\{j\}, l_{k'}) \lll_{l_{k'}} \nabla(\{i\}, l_{k'}).$$

Sind I, J Intervalle über $\{i_k \mid i_k : l_k \longrightarrow t\}$, so gilt

$$I \lll_{l_k} J \quad \Longrightarrow \quad \nabla(I, l_{k'}) \lll_{l_{k'}} \nabla(J, l_{k'}).$$

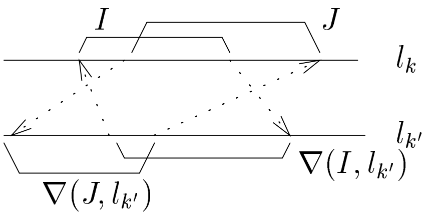
Mit anderen Worten: ∇ ist ein monotoner Operator.

Beweis: Angenommen sei $i \neq j$ und $i \lll_{l_k} j$, aber nicht $\nabla(\{i\}, l_{k'}) \lll_{l_{k'}} \nabla(\{i\}, l_{k'})$. Sei weiter $\nabla(\{i\}, l_{k'}) = [x_i, y_i]$ und $\nabla(\{j\}, l_{k'}) = [x_j, y_j]$. Dann gilt $\neg(x_i \lll_{l_{k'}} x_j)$ oder $\neg(y_i \lll_{l_{k'}} y_j)$.



Bezeichnen wir die äußeren begrenzenden Elemente eines Intervalls $[x, y]$ mit x' bzw. y'. D.h. x' ist das größte Vorkommen, das kleiner als x ist, und y' ist das kleinste Vorkommen, das größer als y ist. Wenn $\neg(x_i \lll_{l_{k'}} x_j)$, existieren Buchstaben a in $l_{k'}$ und b in l_k mit $x_j(a) \leq_t x'_i(a) \leq_t i(a) \leq_t j(b)$, ein Widerspruch zu $x_j \nabla j$. Analog folgt aus $\neg(y_i \lll_{l_{k'}} y_j)$ die Existenz von a und b mit $i(b) \leq_t j(b) \leq_t y'_j(a) \leq_t y_i(a)$.

Die zweite Aussage ergibt sich durch analoge Betrachtungen.



Seien $[x_I, y_I] = I$, $[x_J, y_J] = J$, $[x_{\nabla(I)}, y_{\nabla(I)}] = \nabla(I, l_{k'})$ und $[x_{\nabla(J)}, y_{\nabla(J)}] = \nabla(J, l_{k'})$. Sei wieder $I \neq J$ und $I \lll_{l_k} J$. Aus $\neg(x_{\nabla(I)} \lll x_{\nabla(J)})$ folgern wir die Existenz zweier Buchstaben a, b wie oben mit $x_{\nabla(J)}(b) \leq_t x'_{\nabla(I)}(b) \leq_t x_I(a) \leq_t x_J(a)$ und aus $\neg(y_{\nabla I} \lll y_{\nabla(J)})$ folgern wir a, b mit $y_I(a) \leq_t y_J(a) \leq_t y'_{\nabla(J)}(b) \leq_t y_{\nabla(I)}(b)$. In beiden Fällen ein Widerspruch zu $(I \times \nabla(I, l_{k'})) \cup (J \times \nabla(J, l_{k'})) \subseteq \nabla$. ∎

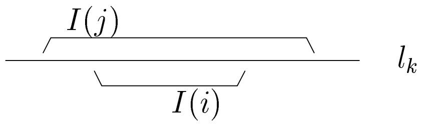
Lemma 54 *Seien $i, j : l_k \longrightarrow t$ mit $i \lll_{l_k} j$. Dann gilt $(I_1, \ldots, I_n)(i) \lll (I_1, \ldots, I_n)(j)$ mit $(I_1, \ldots, I_n)(i), (I_1, \ldots, I_n)(j) \in \mathbb{I}$.*

Beweis: Für $i = j$ ist nichts zu zeigen. Sei also $i \neq j$ und $i \lll_{l_k} j$. Für $k' \neq k$ gilt $I'(l_{k'}, i) = [x_i, y_i] \lll_{l_{k'}} [x_j, y_j] = I'(l_{k'}, j)$. Damit gilt für alle $k' \in [n] \setminus \{k\}$ $I(l_{k'}, i) = I_{k'}(i) \lll_{l_k} I_{k'}(j) = I(l_{k'}, j)$, da alle involvierten Intervalle $I'_{k'}(j)$ größer oder gleich $I'_{k'}(i)$ sind. Also gilt auch $I_k(i) = \bigcap_{k' \in [n] \setminus \{k\}} \nabla(l_k, I'_{k'}(i)) \lll_{l_k} \bigcap_{k' \in [n] \setminus \{k\}} \nabla(l_k, I'_{k'}(j)) = I'_k(j)$. ∎

Wir sagen: Das Intervall $[x_1, y_1]$ ist *strikt* in dem Intervall $[x_2, y_2]$ *enthalten*, falls $x_2 \lll x_1$, $x_2 \neq x_1$ und $y_1 \lll y_2$, $y_1 \neq y_2$. Wir haben in Beispiel 3 gesehen, daß sich die Intervalle $I(i, l_k)$, $i : l_k \longrightarrow t$ durchaus überlappen können, aber sie können sich nicht strikt enthalten.

Korollar 17 *Seien $i, j : l_k \longrightarrow t$. $I(l_k, i)$ kann nicht strikt in $I(l_k, j)$ enthalten sein.*

Beweis:



Ohne Einschränkung ist $i \neq j$. Angenommen $I(l_k, i)$ ist strikt in $I(l_k, j)$ enthalten und $i \lll_{l_k} j$. Dann gilt $[x_i, y_i] = I_k(i) \lll I_k(j) = [x_j, y_j]$ nach obigem Lemma, aber nicht $x_j \lll_{l_k} x_i$ und $x_j \neq x_i$, ein Widerspruch. Gilt $j \lll_{l_k} i$, so folgt $I_k(j) = [x_j, y_j] \lll [x_i, y_i] = I_k(i)$, im Widerspruch zu $y_i \lll_{l_k} y_j$ und $y_i \neq y_j$. ■

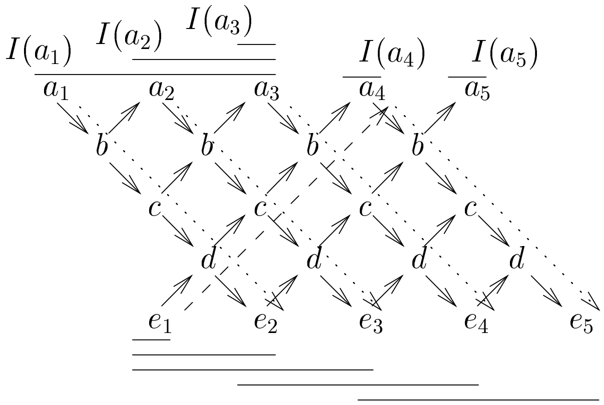
Korollar 18 *Die Ordnung* $\lll$ *auf* $\{I_k(i) \mid i : l_k \longrightarrow t\}$ *ist linear.* ■

Die maximale Anzahl $N_I(i)$ der Faktorvorkommen $i : l_k \longrightarrow t$ überdeckenden Intervalle $I_k(j)$, $N_I(i) = |\{I_k(j) \mid j : l_k \longrightarrow t, i \in I_k(j)\}|$, ist durch eine von $|t|$ unabhängige Konstante beschränkt.

Lemma 55 *Sei* $i : l_k \longrightarrow t$ *ein Faktorvorkommen. Dann existiert eine Konstante* $c = c(\Sigma, D, l)$ *mit* $N_I(i) \leq c$.

Vor dem Beweis betrachten wir das folgende illustrative Beispiel.

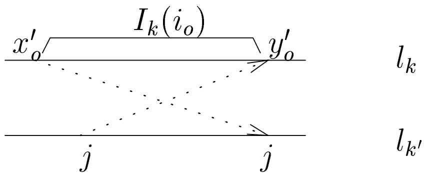
Beispiel 5 *Sei* $(\Sigma, D) = [a - b - c - d - e]$ *und* $l = [ae]$. *Betrachten wir die den dritten Buchstaben a überdeckenden, in der Spur erzeugten Intervalle.*



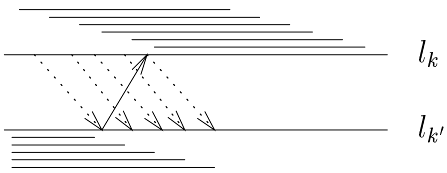
Wir identifizieren Vorkommen wieder mit einzelnen Buchstaben. Die drei a_3 *überdeckenden Intervalle* $\{I(a_1), I(a_2), I(a_3)\}$ *haben unterschiedliche linke Ränder. Deshalb muß für jedes Intervall eine Rückwärts-Sequenz von einem e zu dem nächst kleineren a vor dem linken Rand existieren. (Durch gepunktete Pfeile von links oben nach rechts unten gekennzeichnet.) Da es hier nur ein Sprungwort gibt, muß es für zwei Intervalle verschiedene Sprungsequenzen geben. Doch bereits bei drei Rückwärts-Sprungsequenzen existiert eine Vorwärts-Sprungsequenz (mit einem gestrichelten Pfeil gekennzeichnet), die weitere Überdeckungen verhindert.*

Beweisskizze: Sei $\mathrm{Con}(l) = \{l_1, \ldots, l_n\}$ die Menge der Zusammenhangskomponenten von l, $S(\Sigma, D) = \{w_1, \ldots, w_{n'}\}$ die Menge der Sprungwörter und $n'' = \max\{|w| \mid w \in S(\Sigma, D)\}$. Angenommen es existiert ein Faktorvorkommen $i : l_k \longrightarrow t$ mit $N_I(i) > c := 2nn'n''$. Seien $i_o : l_k \longrightarrow t$, $o \in [m]$, die Faktorvorkommen mit $i \in I_k(i_o)$ für alle $o \in [m]$ und $I_k(i_o) \neq I_k(i_{o'})$ für alle $o \neq o'$. Für alle Intervalle $I_k(i_o) = [x_o, y_o]$, existiert dann

- eine Rückwärts-Sequenz, die von einem Buchstaben aus $j(l_{k'})$ startet und in $x'_o(l_k)$ endet, oder
- eine Vorwärts-Sequenz, die von einem Buchstaben aus $j(l_{k'})$ startet und in $y'(l_k)$ endet.



Da $N_I(i) = m > c$, existiert ein Sprungwort $w \in S(\Sigma, D)$ so, daß $n'' \geq |w|$ Vorwärts-Sprungsequenzen, die in verschiedenen Faktorvorkommen $x'_o : l_k \longrightarrow t$ beginnen.



Sei $I_{k'}(i_o) = [\overline{x_o}, \overline{y_o}]$. Dann existieren Buchstaben b, c mit $\bar{x}_1(a) \leq_t y_1(b)$ (siehe Skizze), ein Widerspruch zu $I_k(i_1) \neq \emptyset$ bzw. $I_k(i_1) \cap I_k(i_m) \neq \emptyset$. ■

Literatur

[Aho90] Alfred V. Aho. Algorithms for finding patterns in strings. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 255–300. Elsevier, Amsterdam, 1990.

[AR88] IJsbrand Jan Aalbersberg and Grzegorz Rozenberg. Theory of traces. *Theoretical Computer Science*, 60:1–82, 1988.

[BD93] Michael Bertol and Volker Diekert. On efficient reduction-algorithms for some trace rewriting systems. In H. Common and J.-P. Jouannaud, editors, *Proceedings of the Term Rewriting.*, number 909 in Lecture Notes in Computer Science, pages 114–126, Berlin-Heidelberg-New York, 1993. Springer.

[BD96] Michael Bertol and Volker Diekert. Trace rewriting: Computing normal forms in time $\mathcal{O}(n \log n)$. In C. Puech and R. Reischuk, editors, *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science 1996*, number 1046 in Lecture Notes in Computer Science, pages 269–280, Berlin-Heidelberg-New York, 1996. Springer.

[Ber95] Michael Bertol. Efficient rewriting in cograph trace monoids. In H. Reichel, editor, *Proceedings of the 10th Fundamentals of Computation Theory (FCT '95), Dresden (Germany) 1995*, number 965 in Lecture Notes in Computer Science, pages 146–155, Berlin-Heidelberg-New York, 1995. Springer.

[BL87] Ronald Book and Hai-Ning Liu. Rewriting systems and word problems in a free partially commutative monoid. *Information Processing Letters*, 26:29–32, 1987.

[BMS83] Alberto Bertoni, Giancarlo Mauri, and Nicoletta Sabadini. Concurrency and commutativity. Technical report, Instituto di Cibernetica, Università di Milano, 1983. Presented at the Workshop on Petri nets, Varenna (Italy), 1982.

[BO93] Ronald Book and Friedrich Otto. *String Rewriting Systems.* Texts and Monographs in Computer Science. Springer Verlag, New York, 1993.

[Boo82] Ronald Book. Confluent and other types of Thue systems. *J. Assoc. Comput. Mach.*, 29:171–182, 1982.

[CF69] Pierre Cartier and Dominique Foata. *Problèmes combinatoires de commutation et réarrangements.* Number 85 in Lecture Notes in Mathematics. Springer, Berlin-Heidelberg-New York, 1969.

[Cho90] Christian Choffrut. A remark on the representation of trace monoids. *Semigroup Forum*, 40:143–152, 1990.

[CL84] Mireille Clerbout and Michel Latteux. Partial commutations and faithful rational transductions. *Theoretical Computer Science*, 34:241–254, 1984.

[Cle84] Mireille Clerbout. *Commutations Partielles et Familles de Langages.* Thèse, Université des Sciences et Technologies de Lille (France), 1984.

[CLRW95] Mireille Clerbout, Michel Latteux, Yves Roos, and Pierre-André Wacrenier. A result on confluence for one-rule trace-rewriting systems. Technical Report IT-95-275, Laboratoire d'Informatique Fondamentale de Lille, Universite des Sciences et Technologies de Lille, Apr. 1995.

[CM88] Robert Cori and Yves Métivier. Approximation of a trace, asynchronous automata and the ordering of events in a distributed system. In *Proceedings of the 15th International Colloquium on Automata, Languages and Programming (ICALP'88)*, number 317 in LNCS, pages 147–161. Springer, 1988.

[CMZ90] Robert Cori, Yves Métivier, and Wiesław Zielonka. Applications of trace approximation to asynchronous automata and message passing with bounded time-stamps in asynchronous networks. Technical report, Univ. Bordeaux, 1990.

[Cou90] Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 193–242. Elsevier, Amsterdam, 1990.

[CP85] Robert Cori and Dominique Perrin. Automates et commutations partielles. *R.A.I.R.O. — Informatique Théorique et Applications*, 19:21–32, 1985.

[Dau88] Max Dauchet. Termination of rewriting is undecidable in the one-rule case. In M. Chytil et al., editors, *Proceedings of the 13th Symposium on Mathematical Foundations of Computer Science (MFCS'88), Carlsbad (CSSR) 1988*, number 324 in Lecture Notes in Computer Science, pages 262–288, Berlin-Heidelberg-New York, 1988. Springer.

[Die87] Volker Diekert. On the Knuth-Bendix completion for concurrent processes. In Th. Ottmann, editor, *Proceedings of the 14th International Colloquium on Automata, Languages and Programming (ICALP'87), Karlsruhe (FRG) 1987*, number 267 in Lecture Notes in Computer Science, pages 42–53, Berlin-Heidelberg-New York, 1987. Springer. Appeared also in a revised version in Theoretical Computer Science 66:117-136, 1989.

[Die89] Volker Diekert. Word problems over traces which are solvable in linear time. In B. Monien et al., editors, *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science (STACS'89), Paderborn (FRG) 1989*, number 349 in Lecture Notes in Computer Science, pages 168–180, Berlin-Heidelberg-New York, 1989. Springer. Revised and extended version in Theoretical Computer Science 74 (1990) 3-18.

[Die90a] Volker Diekert. Combinatorial rewriting on traces. In C. Choffrut et al., editors, *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science (STACS'90), Rouen (France) 1990*, number 415 in Lecture Notes in Computer Science, pages 138–151, Berlin-Heidelberg-New York, 1990. Springer.

[Die90b] Volker Diekert. *Combinatorics on Traces.* Number 454 in Lecture Notes in Computer Science. Springer, Berlin-Heidelberg-New York, 1990.

[Die92] Volker Diekert. Mathematical aspects of trace theory. *Mitt. Math. Ges. Hamburg*, 12:1171–1181, 1992. Special issue of the tricentenary.

[Die93] Volker Diekert. Rewriting, semi-commutations, and Möbius functions. In Z. Ésik, editor, *Proceedings of the 9th Fundamentals of Computation Theory (FCT'93), Szeged (Hungary) 1993*, number 710 in Lecture Notes in Computer Science, pages 1–15, Berlin-Heidelberg-New York, 1993. Springer. Invited Lecture.

[Die94] Volker Diekert. A partial trace semantics for Petri nets. *Theoretical Computer Science*, 134:87–105, 1994. Special issue of ICWLC 92, Kyoto (Japan).

[DJ90] Noam Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier, Amsterdam, 1990.

[DMR95] Volker Diekert, Anca Muscholl, and Klaus Reinhardt. On codings of traces. In E.W. Mayr and C. Puech, editors, *Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95), 1995*, number 900 in Lecture Notes in Computer Science, pages 385–396, Berlin-Heidelberg-New York, 1995. Springer.

[DR95] Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces.* World Scientific, Singapore, 1995.

[Dub85] Christine Duboc. Some properties of commutation in free partially commutative monoids. *Information Processing Letters*, 20:1–4, 1985.

[Dub86] Christine Duboc. On some equations in free partially commutative monoids. *Theoretical Computer Science*, 46:159–174, 1986.

[Fis86] W. Fischer. Über erkennbare und rationale Mengen in freien partiell kommutativen Monoiden. Report FBI-HH-B-121/86, Fachbereich Informatik der Universität Hamburg, Hamburg, 1986. (Diplomarbeit 1985).

[Har78] Michael A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley Publishing Company, 1978.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the Association of Computing Machinery*, 21(8):666–677, 1978.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, London, 1985.

[HU69] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley publishing company, 1969.

[Hue80] Gérard Huet. Confluent reduction: Abstract properties and applications to term rewriting systems. *J. Assoc. Comput. Mach.*, 27:797–821, 1980.

[HY92] Kosaburo Hashiguchi and Kazuya Yamada. String matching problems over free partially commutative monoids. *Information and Computation*, 101:131–149, 1992.

[Jan84] R. Janicki. Mazurkiewicz traces semantics for communicating sequential processes. In *Proceedings of the 5th European Workshop on Appl. and Theory of Petri Nets*, pages 50 – 70, Aarhus, 1984.

[Jan88] Matthias Jantzen. *Confluent String Rewriting.* EATCS Monographs on Theoretical Computer Science 14. Springer, Berlin-Heidelberg-New York, 1988.

[Kel73] Robert M. Keller. Parallel program schemata and maximal parallelism I. Fundamental results. *Journal of the Association of Computing Machinery*, 20(3):514–537, 1973.

[KMP77] Donald E. Knuth, James H. Morris, Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6:323–350, 1977.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the Association of Computing Machinery*, 21:558–564, 1978.

[LL90] L. Lamport and L. Lynch. Distributed computing: Models and methods. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 1157–1200. Elsevier, Amsterdam, 1990.

[Lot83] M. Lothaire. *Combinatorics on Words.* Addison-Wesley, Reading, M.A., 1983.

[Maz77] Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.

[Maz84] Antoni Mazurkiewicz. Traces, histories, graphs: Instances of a process monoid. In M. P. Chytil et al., editors, *Proceeding of the 11th Symposium on Mathematical Foundations of Computer Science (MFCS'84), Prague, (Czechoslovakia) 1984*, number 176 in Lecture Notes in Computer Science, pages 115–133, Berlin-Heidelberg-New York, 1984. Springer.

[Maz87] Antoni Mazurkiewicz. Trace theory. In W. Brauer et al., editors, *Petri Nets, Applications and Relationship to other Models of Concurrency*, number 255 in Lecture Notes in Computer Science, pages 279–324, Berlin-Heidelberg-New York, 1987. Springer.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*. Number 92 in Lecture Notes in Computer Science. Springer, Berlin-Heidelberg-New York, 1980.

[Mil90] Robin Milner. Operational and algebraic semantics of concurrent processes. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 1201–1242. Elsevier, Amsterdam, 1990.

[MM82] Ernst W. Mayr and Albert R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Math.*, 46:305–329, 1982.

[MT90] K. Mehlhorn and A. Tsakalidis. Data structures. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 301–335. Elsevier, Amsterdam, 1990.

[NO88] Paliath Narendran and Friedrich Otto. Preperfectness is undecidable for Thue systems containing only length-reducing rules and a single commutation rule. *Information Processing Letters*, 29:125–130, 1988.

[Ott89] Friedrich Otto. On deciding confluence of finite string rewriting systems modulo partial commutativity. *Theoretical Computer Science*, 67:19–36, 1989.

[Ott95] Friedrich Otto. On confluence versus strong confluence for one-rule trace-rewriting systems. *Mathematical Systems Theory*, 28:363–384, 1995.

[Pra86] Vaughan R. Pratt. Modelling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986.

[Rei85] Wolfgang Reisig. *Petri Nets (an Introduction)*. Number 4 in EATCS Monographs on Theoretical Computer Science. Springer, Berlin-Heidelberg-New York, 1985.

[WD95] Celia Wrathall and Volker Diekert. On confluence of one-rule trace-rewriting systems. *Mathematical Systems Theory*, 28:341–361, 1995.

[WDO92] Celia Wrathall, Volker Diekert, and Friedrich Otto. One-rule trace-rewriting systems and confluence. In I. M. Havel and V. Koubek, editors, *Proceedings of the 17th Symposium on Mathematical Foundations of Computer Science (MFCS'92), Prague, (Czechoslovakia), 1992*, number 629 in Lecture Notes in Computer Science, pages 511–521, Berlin-Heidelberg-New York, 1992. Springer.

[Wei94] Mark Allan Weiss. *Data Structures and Algorithm Analysis*. Benjamin/Cummings Publishing Company, 1994.

Index

Symbole