4. GI Theorietag

„Automaten und Formale Sprachen“

WSI–95–10

Herrsching bei München
29. und 30. September 1994

Markus Holzer (Herausgeber)
Theoretische Informatik/Formale Sprachen

Wilhelm-Schickard-Institut
Universität Tübingen
Sand 13
72076 Tübingen

E-Mail: holzer@informatik.uni-tuebingen.de
Telefon: (07071) 29-7568
Telefax: (07071) 68142

ISSN 0946-3852

Vorwort

Die bisherigen Theorietage „Automaten und Formale Sprachen“ fanden in Magdeburg (September 1991), Kiel (Oktober 1992), Schloß Dagstuhl (Oktober 1993) statt. Im September 1994 wurde die Tradition in der Bildungsstätte des Bayrischen Bauernverbandes in Herrsching bei München fortgesetzt. Es nahmen 32 Teilnehmer aus Deutschland und Österreich teil. Das wissenschaftliche Programm bestand aus angemeldeten Beiträgen der Teilnehmer. Die Kurzfassungen der Beiträge sind in diesem Bericht abgedruckt.

Ich danke allen Teilnehmern für Ihre interessanten Beiträge und die Bereitschaft zur wissenschafltichen Diskussion. Ich danke auch der Technischen Universität München für ihre Unterstützung und allen MitarbeiterInnen der Bildungsstätte des Bayrischen Bauernverbandes Herrsching für ihren Einsatz vor und während der Tagung. Ohne ihre organisatorische Hilfe wäre das Treffen nicht möglich gewesen. Dem nächsten Theorietag in Gießen wünsche ich viel Erfolg.

Ein besonderer Dank gilt Frau Muriel Quenzer und Frau Barbara Thums für die mühevolle Arbeit, diesen Bericht zu erstellen.

Tübingen, im Januar 1995

Markus Holzer
Klaus-Jörn Lange

Inhaltsverzeichnis

Vortragsprogamm

Donnerstag, den 29. September 1994

9:30–9:40 Begrüßung

9:40– Dietmar Wätjen, *Regulär kontrollierte k-limitierte T0L-Systeme*

Henning Fernau, *Programmierte Grammatiken und limitierte L Systeme als Sprachakzeptoren*

Hans Leiß, *Definierbarkeit formaler Sprachen durch nicht-positive Induktion*

Martin Kutrib, *Eingabeverfahren für parallele Automaten*

11:00–11:20 Kaffeepause

11:20– Walter Vogler, *Fairneß und Halbordnungssemantik*

Friedrich Otto, *Solvability of word equations modulo finite special confluent string-rewriting systems*

Klaus Reinhardt, *Vollständige Sprachen für Zählerautomaten*

12:20–14:00 Mittagspause

14:00– Rudolf Freund, *Some new results on array grammars*

Annegret Habel, *Collagen-Grammatiken*

Ralf Stiebe, *Neue Entscheidbarkeitsresultate für Kantengrammatiken*

15:00–15:30 Kaffeepause

15:30– Carsten Damm, *Nichtuniforme reguläre Sprachen*

Christian Herzog, *Pushdown automata with bounded nondeterminism and bounded ambiguity*

Gundula Niemann, *Highlights of Investigations on Weak Growing Context-Sensitive Grammars*

16:30–17:30 Vollversammlung der GI Fachgruppe 0.1.5

Freitag, den 30. September 1994

9:30– Franz Josef Brandenburg, *Kontrollsprachen mit freien und links Variablen*

Klaus P. Jantke, *Graphgrammatik-Konzepte in der Therapieplanung für komplexe dynamische Systeme*

Markus Holzer, *Funktionale Probleme für reguläre Sprachen*

Michael Bertol, *Algorithmen zur Normalformberechnung bzgl. Ersetzungssystemen auf Co-Graph-Monoiden*

11:00–11:20 Kaffeepause

11:20– Dietrich Kuske, *Minimale nichtdeterministische Automaten mit Nebenläufigkeitsrelationen*

Manfred Droste, *Erkennbare und aperiodische Sprachen in Nebenläufigkeitsmonoiden*

Holger Petersen, *Die Mächtigkeit von Zwei-Weg-Zählerautomaten auf beschränkten Sprachen*

12:20– Mittagspause und Abreise

Über Algorithmen zur Normalformberechnung für Ersetzungssysteme auf Co-Graph-Monoiden

Michael Bertol*

FB IV-Informatik
Institut für Informatik
Universität Stuttgart
Breitwiesenstr. 20-22
70565 Stuttgart

email: bertol@informatik.uni-stuttgart.de

Zusammenfassung

Wir betrachten das Normalformenproblem über frei partiell kommutativen Monoiden bezüglich eines endlichen und verkürzenden Ersetzungssystems. Bis jetzt war uns kein Algorithmus bekannt, der eine irreduzible Normalform für ein gegebenes Element in linearer Zeit berechnet. Es gelang uns eine Klasse von Abhängigkeitsalphabeten zu identifizieren, die den Entwurf linearer Algorithmen ermöglichen.

1 Einleitung

Die Theorie der Ersetzungssysteme über frei partiell kommutativen Monoiden (Spurmonoiden) vereinigt kombinatorische Aspekte der Wortersetzungen und Graphersetzungen. Diese Betrachtung führte zu effizienten Algorithmen zur Spurmanipulation. Wir untersuchen hier, ob das nicht-uniforme Normalformproblem (für verkürzende Ersetzungssysteme in linearer Zeit lösbar ist. Bekannt ist für den freien Fall, daß der Algorithmus von R. Book lineare Komplexität besitzt. Ebenso kann man irreduzible Formen für kommutative Monoide (Vektorersetzungssysteme) in Linearzeit bestimmen. Man könnte also erwarten, daß eine größere Klasse

*Partially supported by the ESPRIT Basic Research Actions No. 6317 ASMICS II.

von Monoiden existiert, für die das betrachtete Problem in linearer Zeit lösbar ist.

Wir stellen hier einige Ideen und Konstruktionen bereit, die es erlauben, einen Linearzeitalgorithmus für Cograph-Monoide zu entwerfen.

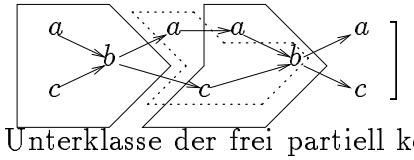
2 Notationen

Mit Σ bezeichen wir ein endliches *Alphabet*; ein *Abhängigkeitsalphabet* ist ein Paar (Σ, D), wobei $D \subseteq \Sigma \times \Sigma$ eine reflexive und symmetrische *Abhängigkeitsrelation* ist. Das Komplement $I = \Sigma \times \Sigma \setminus D$ bezeichen wir als *Unabhängigkeitsrelation*, und der Quotient $\mathbb{M} = \mathbb{M}(\Sigma, D) = \Sigma^*/\{ab = ba \mid (a,b) \in I\}$ bezeichnet das *frei partiell kommutative Monoid* oder *Spurmonoid*. Ein Element $t \in \mathbb{M}$ nennen wir eine Spur. $|t|$ ist die *Länge* der Spur t, $|t|_a$ die *a-Länge*, für $a \in \Sigma$. Mit $\mathrm{alph}(t) = \{a \in \Sigma \mid |t|_a \geq 1\}$ bezeichnen wir das *Alphabet einer Spur*. Wir notieren die Menge $\{1, \ldots, k\}$ mit $[k]$.

Eine Spur kann als eine partiell geordnete Multimenge betrachtet werden. Sei $w = a_1 \ldots a_n \in \Sigma^*$ und $a_i \in \Sigma$, für alle $i \in [n]$. Dann entspricht die geordnete Multimenge $(S_t, \sqsubseteq_t)$ der Spur $t = [w] \in \mathbb{M}$, mit $S_t = \{a_1, \ldots, a_n\}$ und Ordnung $\sqsubseteq_t$, die wie folgt definiert ist: $a_i \sqsubseteq_t a_j$, wenn sowohl $i \leq j$ als auch $(a_i, a_j) \in D$. Der Abhängigkeitsgraph (einer Spur) ist das Hassediagramm dieser Multimenge.

Eine *Subspur* s einer Spur t ist ein Untergraph des Abhängigkeitsgraphen von t, in dem alle Pfade, die in s beginnen und enden, vollständig in s enthalten sind. Jede Subspur s von t ist ein Faktor, d.h. $t = u \cdot s \cdot v$. Eine Faktorisierung ist aber nicht notwendigerweise eineindeutig. Beispielsweise gilt $u \cdot s \cdot v'v'' = uv' \cdot s \cdot v''$ mit $\mathrm{alph}(s) \times \mathrm{alph}(v') \subseteq I$. Aber für eine Faktorisierung $t = u \cdot s \cdot v$ sind die Spuren u, s und v eindeutig als Subspuren identifizierbar. Natürlich können unterschiedliche Subspuren als Elemente von $\mathbb{M}$ identisch sein.

Beispiel: Im Beispiel finden wir zwei Untergraphen, die Subspuren von t sind, und die die Spur $l = [acb]$ repräsentieren. Die Vorkommen sind wieder durch Polygone gekennzeichnet. Das gestrichelte Polygon umrandet einen Untergraphen, der kein Faktor ist.

$t = [acb][a][acb][ca] \cong$ 

Wir betrachten hier eine Unterklasse der frei partiell kommutativen Monoide, die Cograph-Monoide. Für zwei disjunkte Alphabete Σ, Δ definieren wir die *direkte Summe* durch

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

und das *Komplexprodukt* mit

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

Definition 2.1 (Cograph Monoid) *Sei $\mathcal{D}$ die kleinste Klasse von Abhängigkeitsalphabeten $\{(\Sigma, D) \mid D \subseteq \Sigma \times \Sigma, \mathrm{id}_\Sigma \subseteq D, D \subseteq D^{-1}\}$, die alle einelementigen Alphabete $(\{a\}, \{(a,a)\})$ enthält und die unter direkter Summe und unter Komplexprodukt abgeschlossen ist. Dann ist $\mathcal{M} = \{M(X) \mid X \in \mathcal{D}\}$ die Klasse der Cograph-Monoide.*

Wir nennen die Zusammenhangskomponenten von $(\Sigma, D) \in \mathcal{D}$ die *Dekomposition*, wenn der Graph (Σ, D) mehrere Zusammenhangskomponenten besitzt. Wenn (Σ, D) zusammenhängend ist, nennen wir die Menge der Komplemente der Zusammenhangskomponenten des Komplementgraphs $(\Sigma, (\Sigma \times \Sigma) - D)$ die *Dekomposition*. Damit gilt für $\mathbb{M}_\Sigma = \mathbb{M}(\Sigma, D_\Sigma)$ und $\mathbb{M}_\Delta = \mathbb{M}(\Delta, D_\Delta)$

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

Wobei $\mathbb{M}_\Sigma * \mathbb{M}_\Delta$ das übliche freie Produkt und $\mathbb{M}_\Sigma \times \mathbb{M}_\Delta$ das direkte Produkt ist.

Sei $\mathcal{T}$ die Menge der wohlgeformten Terme der Signatur, die alle Buchstaben $a \in \Sigma^*$, sowie die Funktionssymbole $*$ und $\times$ enthält ($*$ und $\times$ sind assoziativ und $\times$ zusätzlich kommutativ). Sei $\mathbb{M} = \mathbb{M}(\Sigma, D)$, wir definieren induktiv

$$h_{(\Sigma,D)} : \mathbb{M}(\Sigma, D) = \mathbb{M} \longrightarrow \mathcal{T}$$

Ist (Σ, D) ein einelementiges Alphabet, dann ist $h_{(\Sigma,D)}$ die Identität $x \longmapsto x$. Andernfalls sei $\mathcal{X} = \{(\Sigma_i, D_i) \mid i \in [k]\}$ die Dekomposition von (Σ, D).

(i) Ist (Σ, D) die direkte Summe $\oplus_{i\in[k]}(\Sigma_i, D_i)$, so definieren wir

$$\begin{aligned} h_{(\Sigma,D)} : \mathbb{M} \cong \prod_{i\in[k]} \mathbb{M}(\Sigma_i, D_i) &\longrightarrow \mathcal{T} \\ (t_1, \ldots, t_k) &\longmapsto \times(h_{(\Sigma_1,D_1)}(t_1), \ldots, h_{(\Sigma_k,D_k)}(t_k)) \end{aligned}$$

(ii) Ist (Σ, D) das Komplexprodukt $*_{i\in[k]}(\Sigma_i, D_i)$, so definieren wir

$$\begin{aligned} h_{(\Sigma,D)} : \mathbb{M} \cong \underset{i\in[k]}{*} \mathbb{M}(\Sigma_i, D_i) &\longrightarrow \mathcal{T} \\ t_1 \cdot \ldots \cdot t_n &\longmapsto *(h_{(\Sigma_{i_1},D_{i_1})}(t_1), \ldots, h_{(\Sigma_{i_n},D_{i_n})}(t_n)), \end{aligned}$$

wobei $t_1 \cdot \ldots \cdot t_n$ die gewöhnliche (alternierende) Faktorisierung ($1 \neq t_j \in \mathbb{M}(\Sigma_{i_j}, D_{i_j})$, für alle $j \in [n]$ und $i_{j'} \neq i_{j'+1}$, für alle $j' \in [n-1]$) ist.

Wir schreiben kurz h für $h_{(\Sigma,D)}$. Für $t \in \mathbb{M}(\Sigma, D)$ nennen wir $h(t)$ den *Spurterm*. Ist $T(\Sigma, D)$ das h-Bild $h_{(\Sigma,D)}(\mathbb{M}(\Sigma, D)) \subset \mathcal{T}$, dann bildet $(T(\Sigma, D), \circ, 1)$ einen Monoid, worin die 1 der leere Term ist und Konkatenation $\circ$ durch

$$l \circ t = \begin{cases} \times(l_1 \circ t_1, \ldots, l_n \circ t_n) & f = g = \times,\ m = n \\ (l_1, \ldots l_{m-1}, l_m \circ t_1, t_2, \ldots, t_n) & f = g = *,\ l_m, t_1 \in T(\Sigma_i, D_i) \\ (l_1, \ldots, l_m, t_1, \ldots, t_n) & f = g = *, \\ & l_m \in T(\Sigma_i, D_i) \neq T(\Sigma_j, D_j) \ni t_1 \end{cases}$$

definiert ist.

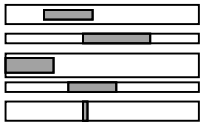
Das bemerkenswerte von Cograph-Monoid-Spuren ist, daß diese Darstellung sich rekursiv fortpflanzt und daß sich durch die Struktur dieser Objekte die Positionen von Faktoren effizient bestimmen lassen.

Lemma 2.2 *Sei* $\{(\Sigma_i, D_i) \mid i \in [k]\}$ *die Dekomposition des Abhängigkeitsalphabets* (Σ, D). *Wir bezeichen* $\mathbb{M}_i = \mathbb{M}(\Sigma_i, D_i)$.

(i) Im Falle eines direkten Produktes $t, l \in \prod_{i\in[k]} \mathbb{M}_i$ *gilt*

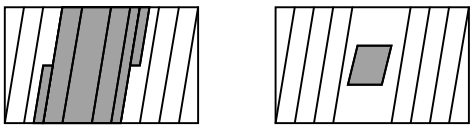
$$t = u \cdot l \cdot v \iff \forall i \in [k] \ : \ \pi_i(t) = \pi_i(u)\pi_i(l)\pi_i(v)$$

wobei π_i *die Projektion* $\mathbb{M}(\Sigma, D) \longrightarrow \mathbb{M}_i$ *ist.*



(ii) Im freien Produkt Fall $t, l \in *_{i\in[k]}\mathbb{M}_i$ *mit* $t = t_1 \cdot \ldots \cdot t_n$ *und* $l = l_1 \cdot \ldots \cdot l_m$ *gilt*

$$t = u \cdot l \cdot v \iff \begin{array}{l} \exists\, j \in [n] \ : \ t_{j+1} = l_1 \ \wedge \ldots \wedge \ t_{j+m-1} = l_{m-1} \ \wedge \\ \exists\, \tilde{u} \ : \ \tilde{u}l_1 = t_{j+1} \ \wedge \ \exists\, \tilde{v} \ : \ l_m\tilde{v} = t_{j+m} \\ \quad \textit{oder} \\ \exists i \in [k] \ : \ l \in \mathbb{M}_i \ \wedge \ \exists\, j \in [n]\ \exists\, \tilde{u}\ \exists\, \tilde{v} \ : \ t_j = \tilde{u}l\tilde{v} \end{array}$$



Die grauen Polygone zeigen einen Faktor, die diagonalen Linien stellen die alternierende Sequenz des freien Produktes dar, und die horizontalen Linien trennen die einzelen Projektionen im direkten Produkt.

Ein *Spurersetzungssystem* ist eine endliche Menge von Ersetzungsregeln $\mathcal{R} \subseteq \mathbb{M}(\Sigma, D) \times \mathbb{M}(\Sigma, D)$. Die *Derivationsrelation* $\Longrightarrow_{\mathcal{R}}$ ist durch $x \Longrightarrow_{\mathcal{R}} y$, mit $x = ulv, y = urv$ für ein $u, v \in \mathbb{M}$ und $l \to r \in \mathcal{R}$ definiert. Mit $\Longrightarrow^*_{\mathcal{R}}$ bezeichnen wir den reflexiven und transitiven Abschluss von $\Longrightarrow_{\mathcal{R}}$. Wir nennen $\mathcal{R}$ *verkürzend*, falls $|l| > |r|$ für alle $l \to r \in \mathcal{R}$ gilt. Eine Spur t ist *irreduzibel*, wenn $t \in \mathrm{Irr}(\mathcal{R}) = \{t \in \mathbb{M} \mid \neg\exists s : t \Longrightarrow_{\mathcal{R}} s\}$.

Die Darstellung als Term ermöglicht die Formulierung eines Algorithmus, der einen Spurterm bottom-up reduziert. Das führt uns zum Hauptresultat.

Theorem 2.3 *Sei* $\mathcal{R} \in \mathbb{M}^2$ *ein verkürzendes Spurersetzungssystem über einem Cograph-Monoiden* $\mathbb{M} \in \mathcal{M}$. *Weiter sei* $t \in \mathbb{M}$ *eine (beliebige) Spur. Dann ist es in linearer Zeit möglich einen irreduziblen Nachfolger* $t \Longrightarrow^*_{\mathcal{R}} \hat{t}$ *zu finden.*

Literatur

[1] I.J. Aalbersberg and G. Rozenberg. Theory of traces. *Theoretical Computer Science*, 60:1–82, 1988.

[2] A.V. Aho *Algorithms for finding patterns in strings* Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, Elsevier 1990

[3] P. Cartier and D. Foata. *Problèmes combinatoires de commutation et réarrangements.* Number 85 in Lecture Notes in Mathematics. Springer, Berlin-Heidelberg-New York, 1969.

[4] V. Diekert. *Combinatorics on Traces.* Number 454 in Lecture Notes in Computer Science. Springer, Berlin-Heidelberg-New York, 1990.

[5] A. 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.

[6] Vaughan Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Processing*, 15(1):33–71, 1986.

[7] J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series-parallel digraphs. *SIAM Journal of Computing*, 11(2):298–313, 1981.

Control Languages under the fl-mode of Rewriting

Franz J. Brandenburg

Universität Passau
Lehrstuhl für Informatik
Innstraße 33
94030 Passau

email: brandenb@informatik.uni-passau.de

We consider context-free grammars with regular control sets and matrix grammars and introduce a new mode of left derivations, called fl-mode for "first and leftmost" or "free and left variables". The fl-mode combines the leftmost and the free mode of rewriting. It is motivated by the equivalence of such grammars with multitape machines.

All definitions are based on context-free grammars and context-free productions. For background see the textbooks of Salomaa [Sa] and Dassow and Paun [DP]. A *matrix grammar* G = (N, T, M, S) combines its productions into matrices m = ($A_1 \to \alpha_1$,..., $A_r \to \alpha_r$) such that the productions are applied in that order. A control word is the string of labels of productions in a derivation. For a context-free grammar G and a regular set C define the *control language* generated by G under C is $L(G,C) = \{w \in T^* | S \Rightarrow^\pi w$ and $\pi \in C\}$. If only leftmost derivations are allowed then let $L_{left}(G,C) = \{w \in T* | S \Rightarrow^\pi w$ is a leftmost derivation and $\pi \in C$ $\}$ be the *left control language* of G and C. The classes of control and left control languages are denoted by L(X, Y) and L_{left}(X, Y), where X ranges over all context-free grammars and Y ranges over all regular sets.

Left control languages have been introduced by Ginsburg and Spanier [GS] and have been studied at depth by Greibach [Gr1]. For many classes of grammars X and languages Y, $_{left}$(X, Y) = {h(L∩L') | L = L(G) for G ∈X, L' ∈Y and h is an erasing homomorphism}. Hence, L_{left}(CFG, REG) = CFL, since the context-free languages are closed under homomorphism and intersection with regular sets. Moreover, the class of recursively enumerable sets RE can be characterized as RE = L_{left}(CFG, LINCFL) and RE = L_{left}(REG, DSPACE(log n)). The further can

be sharpened to L_{left}(X-LINCFG, LINCFL) = RE, where X-LINCFG is the class of extended linear context-free grammars. An extended linear context-free grammar distinguishes ordinary and erasing nonterminals. Accordingly, it has productions of the form A $\rightarrow$ uBv, A $\rightarrow$ x and E $\rightarrow \lambda$, where A and B are ordinary nonterminals, x is a string of terminals and u and v consist of terminals or of erasing nonterminals E which can only be rewritten to the empty word by E $\rightarrow \lambda$.
Such grammars generate (only) linear context-free languages, because the erasing nonterminals can be deleted from G. However, under control sets, the two types of nonterminals are important and the erasing nonterminals become effective for the control and selection of proper derivations. An extended linear context-free grammar is right-nonterminal bounded of degree two, see [Gr1].

On the other hand, for arbitrary derivations, the class of control languages L(CFG, REG) is recursive. This is due to the decidability of the reachability problem for Petri nets, see [Br, GW].

Definition
A matrix grammar G is in *fl-mode*, if for every matrix m = ($A_1 \rightarrow \alpha_1$,..., $A_r \rightarrow \alpha_r$) the first production $A_1 \rightarrow \alpha_1$,,must be applied leftmost and the other productions can be applied freely. "fl" stands for first leftmost. The "fl"-mode can be transferred to grammars with control sets by distinguishing some productions, which must be applied leftmost.

However, this concept is too powerful, since it can be used to simulate two counter machines.

THEOREM 1
For every recursively enumerable set L there is a matrix grammar (or a context-free grammar and a regular control set) in fl-mode G such that L = L(G).

The fl-mode becomes effective, if we strictly distinguish free and left nonterminals. Moreover, the free nonterminals and their productions generate only the empty word and thus operate as regulated rewriting.

Definition
A grammar G = (N, T, P, S) is an *sfl-grammar*, if N = J $\cup$ F with J $\cap$ F = $\emptyset$, and S = $S_l S_f$ where $S_l \in$ J and $S_f \in$ F. J contains the left nonterminals and F the free nonterminals. Accordingly, there are left productions of the form A $\rightarrow$ u with A $\in$ J and u $\in$ (J $\cup$T)* and free productions of the form B $\rightarrow$ v with B $\in$ F and v $\in$ F*.
Hence, for a free variable B, B $\Rightarrow$* w and w $\in$ T* implies w = λ.

G is an *sfl-matrix grammar*, if the first production of each matrix is a left production, and the other productions are free productions of the form given above.

An *sfl-control language* L is the language of an sfl-grammar and a regular control set C.

Standard arguments show that matrix grammars and grammars with regular control sets are equivalent under the sfl-mode, too.

LEMMA

A language L is generated by a context-free sfl-grammar and a regular control set C iff L is generated by a sfl matrix grammar.

The Petri net languages PNL play an important role for control languages. Recall that PNL is the smallest intersection closed full AFL containing the semi-Dyck set D_1'* and is the class of languages accepted by partially blind multicounter machines, a class investigated by Greibach [Gr2]. If these machines are extended by an auxiliary pushdown stack, we obtain the class PDPBC. If the auxiliary pushdown stack is one-reversal bounded these machines define the class 1-PDPBC. It has been proved at several places that PNL does not contain the set of palindromes PAL = {w w^R | w ∈ {a, b}*}, which is a linear context-free language. Hence, PDPBC and 1-PDPBC are proper extensions of PNL.

From AFL theory we obtain PDPBC = {h(L ∩ L') | L ∈ CFL, L' ∈ PNL, and h is an erasing homomorphism} and 1-PDPBC = {h(L ∩ L') | L ∈ LIN-CFL, L' ∈ PNL, and h is an erasing homomorphism}.

Our main results characterizes these classes in a new way.

THEOREM 2

PDPBC = {L_{left}(G, C) | G is a CFG, C ∈ PNL}
= {L_{sfl}(G, C) | G is a CFG, C is a regular set}
= {L_{sfl}(G, C) | G is a CFG, C ∈ PNL}

THEOREM 3

1-PDPBC = {L_{left}(G, C) | G is an extended linear CFG, C ∈ PNL}
= {L_{sfl}(G, C) | G is an extended linear CFG, C is a regular set}
= {L_{sfl}(G, C) | G is an extended linear CFG, C }∈ PNL}

and this class is recursive.

To the contrary, observe that ordinary linear grammars yield only a homomorphic replication, i.e.,{L(G, C) | G is a linear CFG, C ∈ PNL} = {L_{sfl}(G, C) | G is a linear CFG, C ∈ PNL} = {L | L = $h_1(w)h_2(w^R)$ | w ∈ C, C ∈ PNL, h_1 and h_2 are homomorphisms}, and this class is recursive.

References

[Br] F.J. Brandenburg
"Erasing in Context-free Control Languages"
unpublished manuscript (1995)

[DP] J. Dassow, G. Paun
"Regulated rewriting in formal language theory"
Springer-Verlag (1989)

[Gi] S. Ginsburg
"Algebraic and Automata-Theoretic Properties of Formal Languages"
North Holland Publ. Co., Amsterdam (1975)

[GS] S. Ginsburg and E.H. Spanier
"Control sets on grammars"
Math. Systems Theory 2 (1968), 159-177

[Gr1] S.A. Greibach
"Control sets of context-free grammar forms"
Comput. Syst. Sci. 15 (1977), 35-98

[Gr2] S.A. Greibach
"Remarks on blind and partially blind one-way multicounter machines"
Theor. Comput. Sci. 7 (1978), 311-324

[GW] J. Gonczarowski, M.K. Warmuth
"Scattered versus context-sensitive rewriting"
Acta Inform. 27 (1989), 81-95

[Sa] A. Salomaa
"Formal Languages"
Academic Press, New York (1973)

Nichtuniforme reguläre Sprachen

Carsten Damm
FB IV-Informatik
Universität Trier
54286 Trier
email: damm@uni-trier.de

Markus Holzer*
Institut für Informatik
Technische Universität München
Arcisstr. 21, 80290 München
email: holzer@informatik.uni-tuebingen.de

Endliche Automaten sind ebenso wie Turing-Maschinen ein typisch *uniformes* Berechnungsmodell: für jede Eingabe wird der gleiche Algorithmus verwendet. Demgegenüber stehen nichtuniforme Berechnungsmodelle, die für verschiedene Eingabelängen verschiedene Algorithmen verwenden. Typische Vertreter sind Folgen von Schaltkreisen. Dieses Herangehen ist nicht unrealistisch: ein Rechner mit interner 16 Bit-Arithmetik verwendet intern andere Algorithmen als ein Rechner mit 32 Bit Arithmetik.

Es ist nicht ohne weiteres möglich, Komplexitätsklassen zu vergleichen, die durch nichtuniforme und uniforme Berechnungsmodelle definiert sind, denn nichtuniforme Berechnungsmodelle können auch nichtrekursive Sprachen berechnen. Stattet man aber die uniformen Modelle mit Zusatzmechanismen aus, die ihnen erlauben, in Abhängigkeit von der Eingabelänge Algorithmen auszuwählen, so wird ein Vergleich möglich.

Es gibt mehrere Möglichkeiten, diese Zusatzmechanismen zu realisieren. Die folgende Variante stammt von Karp und Lipton [4] und hat den Vorteil, in ihrer Formulierung modellunabhängig zu sein.

Definition 1 *Sei S eine Teilmenge von $\{0,1\}^*$ und sei $h : \mathbf{N} \to \{0,1\}^*$, wobei $\mathbf{N}$ die Menge der natürlichen Zahlen ist. Mit $S : h$ wird die Menge $\{x | x\#h(|x|) \in S\}$ bezeichnet.*

Sei V eine Klasse von Sprachen über $\{0,1\}$ und F eine Klasse von Funktionen von $\mathbf{N}$ in $\{0,1\}^*$, dann wird definiert

$$V/F = \{S : h | S \in V, h \in F\}.$$

Als Funktionenklassen kann man beispielsweise *poly* oder *const* betrachten, die aus Funktionen von N in $\{0,1\}^*$ bestehen, für die $|f(n)|$ polynomial bzw. konstant

*Neue Adresse: Wilhelm-Schickhard-Institut für Informatik, Eberhard-Karls-Universität Tübingen, Sand 13, 72076 Tübingen.

beschränkt ist. Mit dieser Konstruktion kann man zu jeder uniformen Sprachklasse ein nichtuniformes Pendant betrachten. Übertragen auf ein ressourcenbeschränktes Berechnungsmodell A bedeutet dies, A erhält zusammen mit der Eingabe x der Länge n ein Wort $\alpha_n = h(n)$ und Zugehörigkeit zur Sprache hängt davon ab ob A das gesamte Wort $x\#\alpha_n$ akzeptiert. Karp und Lipton nannten dieses Modell „Turing machines that take advice — Turing Maschinen, die auf einen Rat hören“.

Wir untersuchen endliche Automaten mit polynomialen Advices. Die von ihnen akzeptierten Sprachen nennen wir nichtuniforme reguläre Sprachen, die entsprechende Sprachklasse wird mit mit $REG/poly$ bezeichnet.

Motiviert wurde diese Untersuchung dadurch, daß sich verschiedene Nichtuniformitätsmechanismen unterschiedlich auswirken auf die Berechnungskraft von Modellen mit geringen Ressourcen. Wir werden feststellen, daß nichtuniforme reguläre Sprachen „fast reguläre“ Sprachen sind, die Berechnungskraft von endlichen Automaten durch Advices also nicht sehr stark erhöht wird. Verwendet man allerdings einen anderen Nichtuniformitätsbegriff, so gewinnt man eine überraschende Charakterisierung einer wichtigen Schaltkreiskomplexitätsklasse ([1]). Wenn zusätzlich zu der endlichen Kontrolle noch $s(n)$ Speicherplatz benutzt werden darf, so stellt man fest, daß die entsprechenden Komplexitätsklassen erst zusammenfallen, sobald $s(n) = \Omega(\log n)$ gilt. Wir sind auf diesen Unterschied gestoßen bei der Untersuchung des Komplementabschlußes derartiger Sprachklassen [2].

Die folgenden Resultate wurden erzielt.

Satz 1 $REG/poly = REG/const$.

Viel Zusatzinformation hilft erst dann, wenn sie auch genutzt werden kann. Endliche Automaten können nur sehr wenig Nichtuniformität nutzen. [1] Aus diesem Ergebnis erhält man die folgende strukturelle Charakterisierung nichtuniformer regulärer Sprachen:

Satz 2 *Eine Sprache $L \subseteq \{0,1\}^*$ ist nichtuniform regulär genau dann, wenn sie „stückweise regulär“ ist, dh. wenn es ein $k \in \mathbf{N}$, reguläre Sprachen $L_1, \ldots, L_{2^k}$ sowie eine Abbildung $c : \mathbf{N} \to \{1, \ldots, 2^k\}$ gibt, so daß für alle $n \in \mathbf{N}$ gilt:*

$$L \cap \{0,1\}^n = L_{c(n)} \cap \{0,1\}^n.$$

In diesem Sinne sind nichtuniforme reguläre Sprachen „fast regulär“, Pumping-Argumente sind aber nicht unmittelbar anwendbar. Zum Beweis von Separierungen sind stärkere Hilfsmittel notwendig. Es stellt sich beispielsweise sofort die Frage, ob ein endlicher Automat dessen Advices höchstens die Länge k haben, mehr kann, als ein Automat, der $k+1$-Bit-Advices nutzen kann. Es gilt:

[1] Im Vergleich dazu kann ein Pushdown-Automat polynomiale Advices nutzen: Man kann zeigen $CFL/const \subset CFL/poly$.

Satz 3 $REG/0 \subset REG/1 \subset REG/2 \subset \ldots \subset REG/k \subset REG/(k+1) \subset \ldots$.

Die erste dieser Inklusionen folgt daraus, daß $REG/0 = REG$ gilt, während in $REG/1$ bereits nichtrekursive Sprachen liegen, beispielsweise alle unären Sprachen. Die anderen Separierungen können mit Hilfsmitteln der Kolmogorov-Komplexität bewiesen werden.

Satz 4 $\{0^n 1^n | n \in \mathbf{N}\} \notin REG/const.$

Auch dieser Satz kann mit einem Kolmogorov-Argument bewiesen werden. Ein anderer Beweis basiert auf van der Waerden's Theorem und dem Pumping Lemma. Van der Waerdens's Theorem ist auch der Schlüssel zur Separierung der gesamten Chomsky-Hierarchie mit konstanter Nichtuniformität.

Wir haben außerdem gezeigt, daß ein weiteres Nichtuniformitätsmodell, das von Ibarra und Ravikumar [3] eingeführt wurde, zu der gleichen Klasse nichtuniformer regulärer Sprachen führt, wie der Karp/Lipton-Ansatz.

Literatur

[1] David A. Mix Barrington. Bounded width polynomial size branching programs can recognize exactly those languages in NC^1. *Journal of Computer and Systems Sciences*, 38:150–164, 1989.

[2] Carsten Damm and Markus Holzer. Inductive counting below LOGSPACE. In *Mathematical Foundations of Computer Science*, volume 841 of *Lecture Notes in Computer Science*, pages 276–285, Košice, 1994. Springer-Verlag.

[3] Oscar Ibarra and B. Ravikumar. Sublogarithmic-space turing-machines, nonuniform space complexity, and closure properties. *Mathematical Systems Theory*, 21:1–17, 1988.

[4] Richard M. Karp and Richard J. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28:191–209, 1982.

Erkennbare und aperiodische Sprachen in Nebenläufigkeitsmonoiden

Manfred Droste

Institut für Algebra
Technische Universität Dresden
Mommsenstr. 13
01062 Dresden

email: droste@math.tu-dresden.de

Erkennbare und aperiodische Sprachen von Wörtern über einem endlichen Alphabet wurden intensiv untersucht. In der Theorie der Traces (Spuren) gab Ochmanski eine Verallgemeinerung von Kleenes klassischem Satz über reguläre Wortmengen, und Guaiana, Restivo und Salemi bewiesen Schützenbergers Koinzidenzresultat der aperiodischen und sternfreien Sprachen in freien Monoiden auch für Trace-Monoide. Wir geben eine weitere Verallgemeinerung dieser Resultate auf Sprachen in sog. Nebenläufigkeitsmonoiden an.

Spuren bilden ein mathematisches Modell für ein paralleles System, in dem von der Reihenfolge zweier unabhängiger Aktionen abgesehen wird; die Unabhängigkeit wird hierbei durch eine binäre Relation auf dem zugrunde liegenden Alphabet der Aktionen beschrieben. Wir betrachten hier ein allgemeineres Modell, in dem die Unabhängigkeit zweier Aktionen auch von dem jeweiligen Zustand des zugrunde liegenden Systems abhängt. Ein *Automat mit Nebenläufigkeitsrelationen* ist ein Quadrupel $\mathcal{A} = (S, E, T, \|)$, in dem S die Menge der Zustände, E die Menge der Aktionen (Ereignisse), $T \subseteq S \times E \times S$ die deterministische Transitionsrelation und $\| = (\|_s)_{s \in S}$ ein System von binären Unabhängigkeitsrelationen $\|_s$ $(s \in S)$ auf E ist. Zwei Berechnungsfolgen $(s, a, p)(p, b, r)$ und $(s, b, q)(q, a, r)$ werden für äquivalent erklärt, falls $a \|_s b$. Dies induziert auf der Menge $\mathrm{BF}(\mathcal{A})$ aller endlichen Berechnungsfolgen von $\mathcal{A}$ eine Kongruenz $\sim$, und der Quotient $\mathrm{M}(\mathcal{A}) = \mathrm{BF}(\mathcal{A})/\sim \cup \{0\}$, mit Konkatenation als Operation (und 0 als „Fehlerelement“, um eine überall definierte Monoidoperation zu erhalten) heißt *Nebenläufigkeitsmonoid* über $\mathcal{A}$. Falls $\mathcal{A}$ nur einen Zustand hat, erhalten wir so gerade die Trace–Monoide.

In vorhergehenden Arbeiten haben wir Zusammenhänge dieser Automaten mit der Theorie der Bereiche und mit Petri-Netzen mit Kapazitäten (place/

transition-nets) untersucht. Hier erhalten wir zunächst eine Charakterisierung der erkennbaren Sprachen in den Monoiden $\mathrm{M}(\mathcal{A})$, falls $\mathcal{A}$ ein endlicher Automat mit Nebenläufigkeitsrelationen ist: Unter einer geeigneten nötigen Voraussetzung, daß die Nebenläufigkeitsrelationen von $\mathcal{A}$ lokal voneinander abhängen, zeigen wir, daß eine Sprache in $\mathrm{M}(\mathcal{A})$ genau dann erkennbar ist, wenn sie sich aus den endlichen Teilsprachen von $\mathrm{M}(\mathcal{A})$ unter Verwendung der Operationen Vereinigung, Produkt und nebenläufige Iteration konstruieren läßt. Dies verallgemeinert den Satz von Ochmanski; ist hierbei die Unabhängigkeitsrelation leer, so reduziert sich die nebenläufige Iteration genau zu der klassischen Iteration und wir erhalten den Satz von Kleene.

In Nebenläufigkeitsmonoiden sind aperiodische Sprachen sternfrei, jedoch im allgemeinen nicht umgekehrt; Produkte aperiodischer Sprachen sind i.a. nicht mehr aperiodisch, sondern nur noch m–periodisch, wobei $m = m_{\mathcal{A}}$ von $\mathcal{A}$ abhängt. Unter einer geeigneten Voraussetzung an $\mathcal{A}$ ist jedoch $m_{\mathcal{A}} = 1$, so daß in diesem Fall die aperiodischen und die sternfreien Sprachen von $\mathrm{M}(\mathcal{A})$ wieder zusammenfallen. Dies verallgemeinert den Satz von Guaiana, Restivo und Salemi, und ist hier wieder die Unabhängigkeitsrelation trivial, so folgt der „klassische“ Satz von Schützenberger als Spezialfall.

Programmierte Grammatiken und limitierte L Systeme als Sprachakzeptoren

Henning Fernau*

Lehrstuhl Informatik
für Ingenieure und Naturwissenschaftler
Universität Karlsruhe
Am Fasangarten 5
76128 Karlsruhe

email: fernau@ira.uka.de

Henning Bordihn

Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg
Postfach 4120
39016 Magdeburg

email: bordihn@cs.uni-magdeburg.de

Bekanntermaßen gilt für die Chomsky-Hierarchie, daß generierende Typ-i-Grammatiken genauso beschreibungsmächtig sind wie akzeptierende Typ-i-Grammatiken, s. z.B. [10]. Interessanterweise gilt diese triviale Äquivalenz für regulierte bzw. eingeschränkt parallele Ersetzungsverfahren i. a. nicht mehr. Dieses Phänomen haben wir für zahlreiche Grammatiktypen festgestellt, wie unserem Bericht [1] sowie einigen Zeitschriftenveröffentlichungen [2, 7, 6] zu entnehmen ist. In diesem Kurzvortrag konzentrieren wir uns auf programmierte Grammatiken und limitierte L Systeme, jeweils mit kontextfreien Kernregeln.

Entscheidend ist, wie wir die für generierende Grammatiken gegebene Definition eines Ableitungsschrittes auf den akzeptierenden Fall übertragen. Wir wählten den folgenden **Ansatz:** Nimm Definition für den generierenden Fall und ändere

*Neue Adresse: Wilhelm-Schickhard-Institut für Informatik, Eberhard-Karls-Universität Tübingen, Sand 13, 72076 Tübingen.

sie möglichst wenig, so daß sie für den generierenden **und** den akzeptierenden Fall paßt.

Damit erhalten wir triviale Äquivalenzen zwischen akzeptierendem und generierendem Modus,

- falls bei gesteuerter Ersetzung nur Bedingungen an den Kontext 'um die Ersetzungsstelle herum' gestellt werden (z.B. Random Context) und
- falls reine, vollständige parallele Ersetzung (L Systeme) betrachtet wird.

Anders verhält es sich bei den folgenden Beispielen:

Beispiel 1. Eine programmierte Grammatik [8, 11, 10] G ist bestimmt durch ein Nichtterminalalphabet V_N, ein Terminalalphabet V_T, ($V_G = V_N \cup V_T$), eine endliche Menge P von Produktionen sowie das Startsymbol $S \in V_N$. Produktionen haben die Form $(r : \alpha \to \beta, \sigma(r), \phi(r))$, wobei $r : \alpha \to \beta$ eine mit r markierte Kernregel $\alpha \to \beta$ ($\alpha, \beta \in V_G^*$) ist und $\sigma(r), \phi(r) \subseteq \mathrm{Lab}(P)$ Markenmengen sind (Erfolgs- und Mißerfolgsfelder). Für (x, r_1) und (y, r_2) aus $V_G^* \times \mathrm{Lab}(P)$ definieren wir $(x, r_1) \Rightarrow (y, r_2)$ gdw. entweder $x = z_1 \alpha z_2$, $y = z_1 \beta z_2$, $(r_1 : \alpha \to \beta, \sigma(r_1), \phi(r_1)) \in P$ und $r_2 \in \sigma(r_1)$ oder $x = y$, falls die Regel $r_1 : \alpha \to \beta$ einer Produktion $(r_1 : \alpha \to \beta, \sigma(r_1), \phi(r_1)) \in P$ nicht anwendbar auf x ist und r_2 in $\phi(r_1)$ liegt. Wie üblich bezeichne $\overset{*}{\Rightarrow}$ den reflexiven transitiven Abschluß von $\Rightarrow$. Wir definieren

$$L_{\mathrm{gen}}(G) = \{w \in V_T^* \mid (S, r_1) \overset{*}{\Rightarrow} (w, r_2) \text{ für } r_1, r_2 \in \mathrm{Lab}(P)\},$$

$$L_{\mathrm{acc}}(G) = \{w \in V_T^* \mid (w, r_1) \overset{*}{\Rightarrow} (S, r_2) \text{ für } r_1, r_2 \in \mathrm{Lab}(P)\}.$$

$L_{\mathrm{gen}}(G) \in \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF,ac})$, gdw. für jede Kernregel $\alpha \to \beta$ aus G $\alpha \in V_N$ (Kontextfreiheit) gilt. $L_{\mathrm{acc}}(G) \in \mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF,ac})$, gdw. für jede Kernregel $\alpha \to \beta$ aus G $\beta \in V_N$ gilt. Gibt es keinen Vorkommenstest in G, d. h. gilt $\phi(r) = \emptyset$ für jede Regel $r \in \mathrm{Lab}(P)$, so schreiben wir genauer $L_{\mathrm{gen}}(G) \in \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF})$ bzw. $L_{\mathrm{acc}}(G) \in \mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF})$. Fallen Erfolgs- und Mißerfolgsfelder stets zusammen, d. h. $\phi(r) = \sigma(r)$ für jede Regel $r \in \mathrm{Lab}(P)$, so schreiben wir statt dessen $L_{\mathrm{gen}}(G) \in \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF,ut})$ bzw. $L_{\mathrm{acc}}(G) \in \mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF,ut})$. Das Fehlen von λ-Produktionen kennzeichnen wir durch CF-λ anstelle von CF.

Beispiel 2. Ein 1-limitiertes ET0L-System [12, 3] G ist bestimmt durch ein Nichtterminalalphabet V_N, ein Terminalalphabet V_T, ($V_G = V_N \cup V_T$), eine endliche Menge P von Tafeln sowie das Startsymbol $S \in V_N$. Eine Tafel ist eine endlich Menge P_i von kontextfreien Produktionen. $x \Rightarrow y$ (für $x, y \in V_G^*$) gdw. es eine Tafel P_i und Partitionen $x = x_0 \alpha_1 x_1 \cdots \alpha_n x_n$, $y = x_0 \beta_1 x_1 \cdots \beta_n x_n$ derart gibt, daß $\alpha_\nu \to \beta_\nu \in P_i$ für jedes $1 \leq \nu \leq n$ und $\alpha_\nu \neq \alpha_\mu$ für $\nu \neq \mu$ gilt, und jede linke Seite z einer Produktion aus P_i entweder gleich einem α_ν oder nicht enthalten in

$\mathrm{Sub}(x_0) \cup \mathrm{Sub}(x_1) \cup \cdots \cup \mathrm{Sub}(x_n)$ ist. Die zugehörigen Sprachfamilien schreiben wir $\mathcal{L}_{\mathrm{gen}}(\mathrm{1lET0L})$ bzw. $\mathcal{L}_{\mathrm{acc}}(\mathrm{1lET0L})$. Wie bei L Systemen kennzeichnen wir das Fehlen von λ-Produktionen durch ein P für 'propagierend'.

Ergebnisse. Die auf den ersten Blick durchaus erstaunlichen Zusammenhänge zwischen programmierter und limitiert-paralleler Ersetzung finden sich auch im akzeptierenden Fall. In den Fällen, in denen akzeptierender und generierender Modus gleichmächtig sind, ist der Nachweis nicht mehr trivial. Wir wollen hier aber anmerken, daß beispielsweise für programmierte Grammatiken mit leeren Mißerfolgsfeldern eine (beinahe) triviale Äquivalenz von akzeptierendem und generierenden Modus festgestellt werden kann. Wir listen im folgenden einige Ergebnisse auf.

- $\mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF}-\lambda) = \mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF}-\lambda) \subset \mathcal{L}(\mathrm{CS})$
- $\mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF}) \subset \mathcal{L}(\mathrm{REC}) \subset$
  $\subset \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF,ac}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF,ac}) = \mathcal{L}(\mathrm{RE})$
- $\mathcal{L}_{\mathrm{gen}}(\mathrm{1lEPT0L}) \subseteq \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF}-\lambda,\mathrm{ut}) \subset \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF}-\lambda,\mathrm{ac}) \subset \mathcal{L}(\mathrm{CS}) =$
  $\mathcal{L}_{\mathrm{acc}}(\mathrm{1lEPT0L}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF}-\lambda,\mathrm{ut}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF}-\lambda,\mathrm{ac})$
- $\mathcal{L}_{\mathrm{gen}}(\mathrm{1lET0L}) = \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF,ut}) \subseteq$
  $\subseteq \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF,ac}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{1lET0L}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF,ut}) = \mathcal{L}_{\mathrm{acc}}(\mathrm{P,CF,ac})$

Offene Frage. Ist die letzte Inklusion echt oder nicht? Wir konnten nur zeigen, daß es im Falle $\mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF,ut}) = \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF,ac})$ keinen Algorithmus gibt, der eine beliebige (P,CF,ac)-Grammatik in eine äquivalente (P,CF,ut)-Grammatik überführt (sonst wäre das Leerheitsproblem für (P,CF,ac)-Grammatiken entscheidbar).

Hinweis. Wir konnten jüngst zeigen [4], daß — entgegen einer ursprünglichen Vermutung — $\mathcal{L}_{\mathrm{gen}}(\mathrm{1lET0L})$ unentscheidbare Sprachen enthält. Es gilt sogar der folgenden Zusammenhang: $\mathcal{L}_{\mathrm{gen}}(\mathrm{1lET0L})$ umfaßt genau dann alle aufzählbaren Sprachen, wenn $\mathcal{L}_{\mathrm{gen}}(\mathrm{1lET0L})$ gegen Durchschnitt mit regulären Sprachen abgeschlossen ist.

Außerdem konnten wir noch ergänzend zeigen, daß

$$\mathcal{L}_{\mathrm{gen}}(\mathrm{1lEPT0L}) = \mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF}-\lambda,\mathrm{ut})$$

gilt [5]. Es gibt zudem weitere Kennzeichnungen dieser Sprachklassen mit Hilfe anderer Regulationsmechanismen, so wie dies auch schon von Rozenberg und Salomaa im Bericht [9] angedeutet wird.

Literatur

[1] H. Bordihn and H. Fernau. Accepting grammars and systems. Technical Report 9/94, Universität Karlsruhe, Fakultät für Informatik, 1994.

[2] H. Bordihn and H. Fernau. Accepting grammars with regulation. *International Journal of Computer Mathematics*, 53:1–18, 1994.

[3] J. Dassow. A remark on limited 0L systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 24(6):287–291, 1988.

[4] H. Fernau. Membership for 1-limited ET0L languages is not decidable. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 30(4):191–211, 1994.

[5] H. Fernau. Characterizations of unconditional transfer. (in preparation), 1995.

[6] H. Fernau. A note on uniformly limited ET0L systems with unique interpretation. To appear in Information Processing Letters, 1995.

[7] H. Fernau and H. Bordihn. Remarks on accepting parallel systems. *International Journal of Computer Mathematics*, 57(3+4), 1995. (To appear).

[8] D. J. Rosenkrantz. Programmed grammars and classes of formal languages. *Journal of the Association for Computing Machinery*, 16(1):107–131, 1969.

[9] G. Rozenberg and A. K. Salomaa. Context-free grammars with graph-controlled tables. Technical Report DAIMI PB-43, Institute of Mathematics at the University of Aarhus, Ny Munkegade, DAN–8000 Aarhus C, January 1975.

[10] A. K. Salomaa. *Formale Sprachen.* Berlin: Springer, 1975.

[11] E.D. Stotskii. Control of the conclusion in formal grammars. *Problemy peredachi informacii; translated: Problems of information transmission*, 7(3):257–270, 1971 (Translation 1973).

[12] D. Wätjen. k-limited 0L systems and languages. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 24(6):267–285, 1988.

Some new results on array grammars

Rudolf Freund

Technische Universität Wien
Institut für Computersprachen
Resselgasse 3, A-1040 Wien
Österreich

email: freund@csdec1.tuwien.ac.at

Abstract. We report on some new results obtained in investigating the generative power of #-context-free two-dimensional array productions (i.e. of two-dimensional array productions with the left hand side consisting of one non-terminal symbol surrounded by some blank symbols #, but with no restrictions imposed on the right hand side) in combination with well-known control mechanisms for regulated rewriting ([2]), i.e. in two-dimensional programmed array grammars and matrix array grammars as well as ordered array grammars.

1 Introduction

By imposing specific restrictions on two-dimensional array productions regular, context-free, and monotonic two-dimensional array grammars are obtained. The corresponding families of two-dimensional array languages form a Chomsky-hierarchy like in the string case ([1], [7]). Yet these results are obtained with a very restricted form of context-free two-dimensional array productions, i.e. with the right hand side being a finite connected pattern of non-blank symbols only. In the one-dimensional case, these restricted context-free array productions allow the generation of regular one-dimensional array languages only ([4]).

In this paper we report on some new results obtained for array grammars in combination with well-known control mechanisms for regulated rewriting ([2]), i.e. for two-dimensional programmed array grammars and matrix array grammars as well as ordered array grammars. Especially results on the generative power of #-context-free two-dimensional array productions in combination with these control mechanisms are stated; more detailed informations and full proofs can be found in [5].

In the string case, the additional use of λ-rules of the form $A \to \lambda$ increases the generative power of monotonic context-free productions in such a way that programmed grammars with appearance checking and matrix grammars with appearance checking using arbitrary context-free productions allow the generation of any recursively enumerable string language, which is not possible with programmed grammars without appearance checking respectively matrix grammars without appearance checking ([2]). In contrast to the string case, the additional use of blank rules of the form $A \to \#$ in an even more astonishing way increases the generative power of context-free two-dimensional array productions: From the results shown in [4] we know that the generative power of matrix array grammars without and even with appearance checking using only monotonic context-free array productions in some sense is rather small, e. g. there are monotonic array languages that cannot be generated by matrix array grammars with appearance checking using monotonic context-free array productions; yet as it is proved in [5], even matrix array grammars *without* appearance checking and programmed array grammars *without* appearance checking using #-context-free two-dimensional array productions can already generate any recursively enumerable two-dimensional array language (using other proof techniques, in [3] it could only be shown that matrix array grammars *with* appearance checking and programmed array grammars *with* appearance checking can generate any recursively enumerable array language).

2 Definitions

For an alphabet V, by V^{2+} we denote the set of two-dimensional non-empty finite arrays of symbols in V (patterns obtained by marking a finite number of unit squares of the plane with symbols in V; as neither the origin nor the axes of the plane are fixed, each pattern is identified by its marked squares, without reference to its "position" in the plane). The elements of V^{2+} are called *arrays* over V and sets of arrays are called *array languages*.

Given an array $x \in V^{2+}$ (for some alphabet V) and a finite pattern α of symbols in $V \cup \{\#\}$, we say that α is a sub-array of x, if we can place α on x such that all squares of α marked by symbols in V coincide with the corresponding symbols in x and each blank symbol # in α corresponds to a blank symbol # in x.

An *array grammar* is a construct $G = (N, T, P, S, \#)$, where N, T are disjoint alphabets, $S \in N$, # is a special (blank) symbol, and P is a finite set of array productions of the form $\alpha \to \beta$, where α, β are finite patterns over $N \cup T$ of the same shape. The derivation relation $x \Longrightarrow y$ is defined, for $x, y \in (N \cup T)^{2+}$, if there is a rule $\alpha \to \beta \in P$ such that α is a sub-array of x and y is obtained by replacing α in x by β. The reflexive and transitive closure of $\Longrightarrow$ is denoted by $\Longrightarrow^*$, and the array language generated by G is defined by

$$L(G) = \{x \in T^{2+} \mid S \Longrightarrow^* x\}.$$

An array production $\alpha \to \beta$ in P is said to be *monotonic* if the non-# symbols in α are not replaced by # in β; $\alpha \to \beta$ in P is called *#-context-free* if the left-hand side α consists of one non-terminal symbol surrounded by some blank symbols #; $\alpha \to \beta$ in P is said to be *context-free* if, moreover, the right-hand side β consists of connected non-# symbols only; if $\alpha \to \beta$ in P is of one of the following forms, then it is called *regular:*

$$\# \; A \to B \; a \;, \quad A \; \# \to a \; B \;, \quad \begin{matrix} \# \\ A \end{matrix} \to \begin{matrix} B \\ a \end{matrix} \;, \quad \begin{matrix} A \\ \# \end{matrix} \to \begin{matrix} a \\ B \end{matrix} \;, \quad A \to a \;,$$

$A, B \in N$, $a \in T$.

G is called an *arbitrary, #-context-free, monotonic, context-free,* respectively *regular* array grammar, if every production in P is an arbitrary, #-context-free, monotonic, context-free, respectively regular array production (we also say that G respectively the productions in P are of type $ENUMA$, $\#-CFA$, $MONA$, CFA, and $REGA$). The corresponding families of array languages shall be denoted by $L(ENUMA)$, $L(\#-CFA)$, $L(MONA)$, $L(CFA)$, and $L(REGA)$.

We now shortly introduce the notions for the specific control mechanisms we deal with in the sequel, i. e. *ordered grammars, programmed grammars,* and *matrix grammars* ([2]).

An *ordered array grammar* is a construct $G_O = (N, T, (P, <), S, \#)$,where V_N, V_T, P, and S are defined as for an array grammar and $<$ is a partial order relation on the array productions in P. The derivation relation for G_O is defined in that way that an array production p from P is only applicable to an underlying array v, if no other array production q from P with $p < q$ is applicable to v.

A *programmed array grammar (or graph controlled array grammar) with appearance checking* is a construct $G_P = (N, T, (R, L_i, L_f), S, \#)$, where N, T, S are as usual and R is a finite set of rules r of the form $(l(r) : p(l(r)), \sigma(l(r)), \varphi(l(r)))$, where $l(r) \in Lab(G_P)$, $Lab(G_P)$ being a set of labels associated (in a one-to-one manner) to the rules r in R, $p(l(r))$ is an array production over $N \cup T$, $\sigma(l(r)) \subseteq Lab(G_P)$ is the *success field* of the rule r, and $\varphi(l(r))$ is the *failure field* of the rule r; $L_i \subseteq Lab(G_P)$ is the set of initial labels, and $L_f \subseteq Lab(G_P)$ is the set of final labels. For $r = (l(r) : p(l(r)), \sigma(l(r)), \varphi(l(r)))$ and $v, w \in (N \cup T)^{2+}$ we define $(v, l(r)) \vdash_{G_P} (w, k)$ if and only if

- **either** $p(l(r))$ is applicable to v, the result of the application of the array production $p(l(r))$ to v is w, and $k \in \sigma(l(r))$,
- **or** $p(l(r))$ is not applicable to v, $w = v$, and $k \in \varphi(l(r))$.

The array language generated by G_P is

$$L(G_P) = \{w \in T^{2+} \mid (S, i) \vdash_{G_P} (w_1, l_1) \vdash_{G_P} \ldots (w_k, l_k),\ k \geq 1,\ w_j \in (N \cup T)^{2+} \text{ and } l_j \in Lab(G_P) \text{ for } 1 \leq j \leq k,\ w_k = w,\ i \in L_i,\ l_k \in L_f\}\,.$$

If the failure fields $\varphi(l(r))$ are empty for all $r \in R$, then G_P is called a *programmed grammar without appearance checking.*

A *matrix array grammar* is a construct $G_M = (N, T, (M, F), S, \#)$, where N, T, S are as usual and M is a finite set of matrices, $M = \{m_i \mid 1 \leq i \leq n\}$, where the matrices m_i are sequences of the form $m_i = (m_{i,1}, ..., m_{i,n_i})$, $n_i \geq 1$, $1 \leq i \leq n$, and the $m_{i,j}$, $1 \leq j \leq n_i$, $1 \leq i \leq n$, are array productions over $N \cup T$, and F is a subset of $\bigcup_{1 \leq i \leq n,\, 1 \leq j \leq n_i} \{m_{i,j}\}$. For $m_i = (m_{i,1}, ..., m_{i,n_i})$ and $v, w \in (N \cup T)^{2+}$ we define $v \vdash_{m_i} w$ if and only if there are $w_0, w_1, ..., w_{n_i} \in (N \cup T)^{2+}$ such that $w_0 = v$, $w_{n_i} = w$, and for each $j, 1 \leq j \leq n_i$,

- **either** w_j is the result of the application of $m_{i,j}$ to w_{j-1},
- **or** $m_{i,j}$ is not applicable to w_{j-1}, $w_j = w_{j-1}$, and $m_{i,j} \in F$.

The language generated by G_M is

$$L(G_M) = \{w \in T^{2+} \mid \; S \vdash_{m_{i_1}} w_1 ... \vdash_{m_{i_k}} w_k = w,\; w_j \in (N \cup T)^{2+} \text{ and } m_{i_j} \in M \text{ for } 1 \leq j \leq k\}.$$

If $F = \emptyset$ then G_M is called a *matrix array grammar without appearance checking.*

An ordered array grammar, a programmed array grammar, respectively a matrix array grammar is said to be of type $ENUMA$, $\# - CFA$, $MONA$, CFA, respectively $REGA$, if every array production appearing in this grammar is of the corresponding type. For

$$X \in \{ENUMA, \# - CFA, MONA, CFA, REGA\}$$

by

$$O(X),\; P_{ac}(X),\; M_{ac}(X),\; P(X),\; M(X)$$

we denote the array languages generated by ordered array grammars, programmed array grammars with appearance checking, matrix array grammars with appearance checking, programmed array grammars without appearance checking, respectively matrix array grammars without appearance checking of type X.

3 Results

Detailed proofs of most of the results stated in this section can be found in [5].

Theorem 1. *(Chomsky-hierarchy of two-dimensional array languages)*

$$L(REGA) \subsetneq L(CFA) \subsetneq L(MONA) \subsetneq L(ENUMA) \text{ and}$$
$$L(REGA) \subsetneq L(CFA) \subsetneq L(\# - CFA) \subsetneq L(ENUMA).$$

Example 1. The set of solid squares over the one-letter alphabet $\{a\}$ is a regular array language ([8]).

Whereas in the string case the control mechanism of an ordering on the productions does not increase the generative power of regular grammars, there are regular ordered array grammars generating array languages that cannot even be generated by (#-)context-free array grammars.

Example 2. According to [8], the set R_H of hollow rectangles of thickness one over the one-letter alphabet $\{a\}$ cannot be generated by a context-free array grammar, but $R_H \in O(REGA)$. For the generation of the set S_H of hollow squares (compare with lemma 1 in [6]) regular array productions are not sufficient, but $S_H \in M_{ac}(CFA)$.

Lemma 1. For all $X, Y \in \{REGA, CFA, \# - CFA, MONA, ENUMA\}$ with $X \subseteq Y$ as well as for all $U \in \{P, P_{ac}, M, M_{ac}, O\}$, $\quad U(X) \subseteq U(Y)$.

Lemma 2. For every $X \in \{REGA, CFA, \# - CFA, MONA, ENUMA\}$,

$$L(X) \subseteq P(X) \subseteq P_{ac}(X),\ L(X) \subseteq M(X) \subseteq M_{ac}(X),\ L(X) \subseteq O(X),$$

$$M(X) \subseteq P(X),\ M_{ac}(X) \subseteq P_{ac}(X),\ O(X) \subseteq M_{ac}(X).$$

Whereas matrix array grammars without and even with appearance checking using context-free array productions only cannot even generate all monotonic array languages (which follows from the results proved in [4]), the results proved in [5] show that matrix array grammars without appearance checking, programmed array grammars without appearance checking, and ordered array grammars using #-context-free array productions can already generate any recursively enumerable array language (which contradicts the results obtained in the string case):

Theorem 2. $X(\#-CFA) = Y(ENUMA)$ for every $X \in \{P, P_{ac}, M, M_{ac}, O\}$ and every $Y \in \{L, P, P_{ac}, M, M_{ac}, O\}$.

The main idea used in the simulations of the proof of theorem 2 is the following: By using only context-free array productions suitable *working areas*, i. e. specific squares like that depicted in Figure 1 (also compare with Example 1) can be generated (at the beginning of a simulation, the start symbol S is surrounded by symbols B, the symbols E seperate the working area against the blank symbols around the square, and the symbol H stands for the symbols controlling the simulation). In such a working area the blank symbol # is represented by the symbol B; hence a non-context-free array production like $AD \to XY$, with A, D, X, Y being non-terminal symbols, can be simulated by the sequence of #-context-free array productions $D \to \#$ and $A\# \to XY$.

H	E	E	E	E	E	E
E	B	B	B	B	B	E
E	B	B	B	B	B	E
E	B	B	S	B	B	E
E	B	B	B	B	B	E
E	B	B	B	B	B	E
E	E	E	E	E	E	E

Figure 1. Example for an initial working area.

4 Conclusion and open problems

The main results reported in this paper (see [5] for the detailed proofs) are based on the construction of suitable working areas, i. e. specific squares (see Figure 1) that can be generated by using context-free two-dimensional array productions only. Constructing suitable working areas for the one-dimensional case is a straight-forward exercise, so the proofs of these results can immediately be adapted for the one-dimensional case. Yet if these results should also be proved for higher dimensions (i. e. for $n \geq 3$) by using the same proof techniques, then the construction of suitable working areas (i. e. specific n-dimensional hyper-cubes) by using n-dimensional context-free array productions only or at least by using n-dimensional (#-)context-free matrix array grammars without appearance checking respectively n-dimensional (#-)context-free programmed array grammars without appearance checking remains as a non-trivial task for future research. Yet another proof technique allows the generation of the desired working areas by ordered array grammars for arbitrary n.

References

[1] C. R. Cook and P. S.-P. Wang, A Chomsky hierarchy of isotonic array grammars and languages, *Computer Graphics and Image Processing* **8** (1978) pp. 144-152.

[2] J. Dassow and Gh. Păun, *Regulated Rewriting in Formal Language Theory* (Springer, Berlin, 1989).

[3] R. Freund, Regulated rewriting of arrays, *"1. Automatentag"*, Magdeburg, 1991.

[4] R. Freund and Gh. Păun, One-dimensional matrix array grammars, *J. Inform. Process. Cybernet.* **EIK 29 (6)** (1993) pp. 1-18.

[5] R. Freund, Control mechanisms on #-context-free array grammars. In: Gh. Păun (ed.), Mathematical Aspects of Formal Languages, Series in Computer Science, Vol. 43, pp. 97–136 (WSP, Singapore, 1994).

[6] J. Dassow, R. Freund, and Gh. Păun, Cooperating array grammar systems, *accepted for International Journal of Pattern Recognition and Artificial Intelligence.*

[7] P. S.-P. Wang, Some new results on isotonic array grammars, *Information Processing Letters* **10** (1980) pp. 129-131.

[8] Y. Yamamoto, K. Morita, and K. Sugata, Context- sensitivity of two-dimensional regular array grammars. In: P. S.-P. Wang (ed.), *Array Grammars, Patterns and Recognizers*, WSP Series in Computer Science, Vol. 18, 17–42 (WSP, Singapore, 1989).

Chain-Code Pictures and Collages Generated by Hyperedge Replacement

Annegret Habel*

Universität Bremen
Fachbereich Mathematik und Informatik
Postfach 330 440
28334 Bremen

e-mail: habel@informatik.uni-hildesheim.de

1 Introduction

The generation and recognition of artificial pictures and patterns are challenging tasks in computer science and other applied areas. In the literature, one encounters quite a variety of syntactic approaches where classes of patterns are described by grammars. Among these approaches are chain-code grammars and collage grammars. Chain-code grammars, as introduced in [MRW82], are based on string grammars over the alphabet $\{u, d, l, r, \uparrow, \downarrow\}$ and provide devices for the generation of picture languages by interpreting the symbols u, d, l, r of a string as commands for drawing a unit line segment in the direction up, down, left, and right, respectively, and the symbols $\uparrow, \downarrow$ for lifting the pen up and down. Collage grammars, as introduced in [HK91, HKT93], are based on hyperedge replacement in a geometric environment and provide context-free syntactic devices for the generation of picture languages by joining the parts (sets of points) of a collage.

In this paper, we relate context-free chain-code grammars and collage grammars and show that the two syntactic devices for the generation of picture languages are incomparable. Restricting to regular chain-code grammars and a suitable type of regular collage grammars, the approaches become comparable. As a consequnce, all undecidability results known for regular chain-code picture languages can be carried over to regular collage languages. In particular, it can be shown that it is undecidable whether or not the language of a regular collage

*Neu Adresse: Universität Hildesheim, Institut für Informatik, Marienburger Platz 22, 31141 Hildesheim.

grammar contains only (a) pictures with holes, (b) disconnected pictures, or (c) connected patterns.

A long version of the paper is given in Dassow, Habel, and Taubenberger [DHT95].

2 Chain-Code Picture Languages

A picture in the sense of [MRW82] consists of a finite set of unit lines in the Euclidean space considered as a square grid. A word over the alphabet $\{u, d, l, r, \uparrow, \downarrow\}$ is a picture description in the sense that it represents a traversal of a picture where the interpretation of the symbols $u, d, l, r, \uparrow, \downarrow$. A set of picture descriptions forms a picture description language. The interpretation of the words in the picture description language yields a set of pictures, a so-called chain-code picture language.

Assumption 2.1 Let $\mathbb{Z}^2$ denote the two-dimensional Euclidean space with integer coordinates. For $z = (x, y) \in \mathbb{Z}^2$, $u(z) = (x, y+1)$, $d(z) = (x, y-1)$, $l(z) = (x-1, y)$, and $r(z) = (x+1, y)$. The neighbourhood of z is defined as $N(z) = \{u(z), d(z), l(z), r(z)\}$. For $z \in \mathbb{Z}^2$ and $z' \in N(z)$, $\langle z, z' \rangle$ denotes the (undirected) unit line connecting z and z'.

Definition 2.2 (picture description words, grammars, and languages)
A *picture* is a finite set of unit lines in the grid $\mathbb{Z}^2$. A *drawn picture* is a triple $Q = (P, z, s)$ where P is a picture, $z \in \mathbb{Z}^2$ is a point, and $s \in \{\uparrow, \downarrow\}$ gives the state pen-up or pen-down.

Let $\pi = \{u, d, l, r, \uparrow, \downarrow\}$. Every word in π^* is called a *picture description* or a *π-word*, every language over π is called a *picture description language* or a *π-language*, and every Chomsky-grammar generating a π-language is called a *picture description grammar* or a *π-grammar*.

The *drawn picture* described by a π-word w, denoted by $dpic(w)$, is defined inductively as follows: If $w = \lambda$, then $dpic(w) = (\emptyset, (0,0), \downarrow)$. If $w = vb$ for some $v \in \pi^*$, $b \in \pi$ and $dpic(v) = (P, z, s)$, then

$$dpic(w) = \begin{cases} (P \cup \{\langle z, b(z)\rangle\}, b(z), \downarrow) & \text{if } s = \downarrow \text{ and } b \in \{u, d, l, r\} \\ (P, b(z), \uparrow) & \text{if } s = \uparrow \text{ and } b \in \{u, d, l, r\} \\ (P, z, \downarrow) & \text{if } b = \downarrow \\ (P, z, \uparrow) & \text{if } b = \uparrow \end{cases}$$

The *chain-code picture* described by a π-word w, denoted by $pic(w)$, is the picture P underlying the drawn picture $dpic(w) = (P, z, s)$. The *chain-code picture language* described by a π-language L, denoted by $pic(L)$, is defined by $pic(L) = \{pic(w) | w \in L\}$.

A picture language B is called *regular* (*context-free*, etc.) if $B = pic(L(G))$ for some regular (context-free, etc.) π-grammar G. The class of all regular and context-free picture languages is denoted by $\mathcal{B}(REG)$ and $\mathcal{B}(CF)$, respectively.

3 Collage Grammars

A collage consists of a set of parts being geometric objects, and a sequence of so-called pin-points. To allow the generation of sets of collages, a collage can be decorated by hyperedges. A hyperedge has a label and an ordered finite set of tentacles, each of which is attached to a point. The hyperedges in decorated collages serve as place holders for (decorated) collages. A hyperedge may be replaced by a decorated collage, if there exists a transformation t from a given set of admissible transformations which maps each pin-point to the corresponding attachment point of the hyperedge. In this case, the result of the replacement is the decorated collage obtained from the original collage by removal of the hyperedge and addition of the transformed image of the replacing collage.

Assumption 3.1 Let $\mathbb{R}^2$ denote the Euclidean space of dimension 2. Moreover, let $TRANS$ be an arbitrary, but fixed set of affine transformations $t{:}\mathbb{R}^2 \to \mathbb{R}^2$.

Definition 3.2 (decorated collages, collage grammars and languages)
A *collage* in $\mathbb{R}^2$ is a pair $(PART, pin)$ where $PART$ is a finite set of parts, each part being a set of points in $\mathbb{R}^2$, and $pin \in (\mathbb{R}^2)^*$ is a sequence of pairwise distinct *pin-points*. Given a set LAB of *labels*, a *(hyperedge-) decorated collage* in $\mathbb{R}^2$ over LAB is a system $C = (PART, EDGE, att, lab, pin)$, where $(PART, pin)$ is a collage in $\mathbb{R}^2$, $EDGE$ is a finite set of *hyperedges*, and $att{:}EDGE \to (\mathbb{R}^2)^*$ and $lab{:}EDGE \to LAB$ are mappings, assigning a sequence of pairwise distinct *attachment-points* as well as a *labeling* to each hyperedge. A decorated collage with empty set of hyperedges and empty mappings att and lab may be seen as a collage. The class of all collages is denoted by $\mathcal{C}$. The class of all decorated collages over LAB is denoted by $\mathcal{C}(LAB)$.

A *(context-free) production* over LAB is of the form $A \to R$ where A is a symbol in LAB and R is a decorated collage over LAB. The application of a production $A \to R$ to a decorated collage C proceeds in three steps: (1) Choose a hyperedge e with label A and a transformation t from $TRANS$ which maps the pin-points of R to the attachment points of e, (2) remove e from C, yielding $C - \{e\}$, and (3) add the transformed image $t(R)$ to the remainder, yielding the decorated collage $C' = C - \{e\} + t(R)$.

We write $C \Longrightarrow C'$ and say C' is *directly derived from* C, if C' can be obtained from C by the application of a production. We write $C \Longrightarrow^* C'$, and say C' is *derivable from* C, if $C = C'$ or there is some finite sequence of decorated collages $C_0, C_1, \ldots, C_k$ with $C_0 = C$ and $C_k = C'$ such that for all i, C_{i+1} is directly derived from C_i. (Thus $\Longrightarrow^*$ is the reflexive and transitive closure of $\Longrightarrow$.)

A *(context-free) collage grammar* is a system $CG = (LAB, PROD, START)$ where LAB is a finite set of *labels*, $PROD$ is a finite set of *productions* over LAB, and $START$ is a decorated collage over LAB, called the *axiom*. The *collage language* generated by CG, $L(CG)$ for short, consists of all collages which can be derived from $START$ by applying productions of $PROD$.

A collage grammar $CG = (LAB, PROD, START)$ is *regular* if $START$ is a decorated collage with one hyperedge and empty pin-point sequence, and the right-hand side of each production contains at most one hyperedge.

A collage language L is called *regular (context-free)* if there is a regular (context-free) collage grammar CG such that $L = L(CG)$. The class of all regular and context-free collage languages is denoted by $\mathcal{L}(REG)$ and $\mathcal{L}(CF)$, respectively.

4 Comparison

A picture in the sense of [MRW82] may be seen as a collage in $\mathbb{R}^2$ where each part is a unit line and the sequence of pin-points is empty. Vice versa, there are collages in $\mathbb{R}^2$ consisting of unit lines as well as other types of parts, such as triangles, circuits, etc. Therefore the following lemma is obvious.

Lemma 4.1 *There is a context-free collage language L such that $L \notin \mathcal{B}(CF)$.*

Lemma 4.2 *There is a context-free π-language L such that $pic(L) \notin \mathcal{L}(CF)$.*

Assumption 4.3 In the following, we restrict the set of admissible transformations to the group of all translations generated by the basis translations t_u, t_d, t_l, t_r: $\mathbb{R}^2 \to \mathbb{R}^2$ given by $t_u(x,y) = (x, y+1)$, $t_d(x,y) = (x, y-1)$, $t_l(x,y) = (x-1, y)$, and $t_r(x,y) = (x+1, y)$ (for $(x,y) \in \mathbb{R}^2$).

Definition 4.4 (decorated pictures, picture grammars and languages)
A *decorated picture* is a decorated collage $C = (PART, EDGE, att, lab, pin)$ in which all parts in $PART$ are unit lines in the grid, all hyperedges in $EDGE$ are attached to points in $\mathbb{Z}^2$, i.e. $att(e) \in (\mathbb{Z}^2)^*$ for $e \in EDGE$, and $pin \in (\mathbb{Z}^2)^*$.

A *picture grammar* is a collage grammar $CG = (LAB, PROD, START)$ in which $START$ and the right-hand sides of the productions in $PROD$ are decorated pictures. A picture language L is called *regular* (*context-free*) if there is a regular (context-free) picture grammar CG such that $L = L(CG)$.

Theorem 4.5 *For every regular π-grammar G, there is a regular picture grammar CG such that $pic(L(G)) = L(CG)$.*

Theorem 4.6 *For every regular picture grammar CG, there is a regular π-grammar G such that $pic(L(G)) = L(CG)$.*

5 Undecidability

Pictures in the sense of definition 2.2 may be considered as undirected graphs, where the the set of nodes is given by the set of points of $\mathbb{Z}^2$ belonging to the picture, and the set of edges is given by the set of unit lines of the picture. Therefore, we may ask whether or not the graph $graph(P)$ of a picture P satisfies a suitable graph property $PROP$ or not.

Theorem 5.1 *It is undecidable whether or not a regular picture language contains (a) a simple curve, (b) a closed simple curve, (c) an Eulerian picture, (d) an Eulerian cycle, (e) a tree, (f) a regular picture, (g) a Hamiltonian picture, (h) a Hamiltonian cycle, (i) a picture unit-line-colourable by three colours, (j) a 2-connected picture.*

A picture P is *connected* if the set $pattern(P)$ of all points of unit lines is connected, i.e., if each two points $x, y \in pattern(P)$ are connected by a continous line $L \subseteq pattern(P)$. If P is not connected, it is said to be *disconnected*. Moreover, a picture P is said to possess a *hole*, if there exist paths in $pattern(P)$ that are not continously deformable into each other.

Theorem 5.2 *It is undecidable whether or not a regular picture language contains only (a) disconnected pictures or (b) connected pictures.*

Theorem 5.3 *It is undecidable whether or not a regular picture language contains a picture without holes.*

Remark In this section, it is shown that it is undecidable whether or not the language of a regular picture grammar contains only (a) disconnected pictures or (b) connected pictures. In Drewes, Kreowski [DK95], for fairly different classes of collage grammars (linear, pin-bounded, non-overlapping collage grammars and compact, pin-bounded, non-overlapping collage grammars, respectively) corresponding undecidability results are proved. Due to the non-comparability of the selected grammar classes, those results are not comparable to the results given in this paper.

References

[DHT95] Jürgen Dassow, Annegret Habel, Stefan Taubenberger. Chain-code pictures and collages generated by hyperedge replacement, 1995. Submitted for publication.

[DH93] Jürgen Dassow, Friedhelm Hinz. Decision problems and regular chain code picture languages. Discrete Applied Mathematics 45, 29–49, 1993.

[DK95] Frank Drewes, Hans-Jörg Kreowski. (Un-)decidability of geometric properties of pictures generated by collage grammars. Fundamenta Informaticae, 1995. To appear.

[HK91] Annegret Habel, Hans-Jörg Kreowski. Collage grammars. Lecture Notes in Computer Science 532, 411–429, 1991.

[HKT93] Annegret Habel, Hans-Jörg Kreowski, Stefan Taubenberger. Collages and patterns generated by hyperedge replacement. Languages of Design 1, 125–145, 1993.

[MRW82] Hermann A. Maurer, Grzegorz Rozenberg, Emo Welzl. Using string languages to describe picture languages. Information and Control 54, 155–185, 1982.

Pushdown Automata with Bounded Nondeterminism and Bounded Ambiguity

Christian Herzog

J. W. Goethe-Universität
Fachbereich Informatik,
Postfach 11 19 32
60054 Frankfurt

email: herzog@psc.informatik.uni-frankfurt.de

The concepts of ambiguity and nondeterminism play a fundamental role in automata theory, and there are some famous open problems regarding nondeterminism, for example the $\mathcal{P}$ versus $\mathcal{NP}$ or the DLBA versus NLBA problem. For the pushdown automata (PDA) model, many questions concerning ambiguity and nondeterminism already have been answered, but some problems worth addressing still remain. Forbidding ambiguity or nondeterminism in PDA's has advantages for some applications of PDA's, for example for the syntax checking of programming languages. On the other hand, this restriction has the disadvantage that not every context-free language can be described by an unambiguous or deterministic PDA. Second, this restriction may change complexity, that is, for some languages the minimal ambiguous or nondeterministic automaton is much smaller than the minimal unambiguous or deterministic one.

We want to examine the situation between completely forbidding ambiguity or nondeterminism and allowing an arbitrary amount of ambiguity or of nondeterminism. Therefore, we introduce measures for the amount of ambiguity and of nondeterminism in PDA's. In particular, we are interested in PDA's where this amount is finite. We then examine the corresponding classes of languages and the savings in complexity obtained.

Measuring the amount of ambiguity in context-free grammars (CFG) by the supremum of the number of leftmost derivations of words is well known. By counting accepting computations instead of leftmost derivations, this measure is easily transferred to PDA's in such a way that CFG's and PDA's, both with an amount of ambiguity bounded by the same finite constant, describe the same class of languages.

There also are several measures for the amount of nondeterminism in a PDA, for example the so called “minmax”-measure by Salomaa and Yu from [SY93]. The minmax-measure of an input word is the minimal number of nondeterministic steps necessary to accept this word. The minmax-measure of a PDA is the supremum of the minmax-measures of all words accepted by this PDA. Salomaa and Yu show that PDA's with minmax-measure zero describe exactly the class of DCFL's, PDA's with minmax-measure one or equivalently with finite measure describe finite unions of DCFL's, and PDA's with infinite measure describe general CFL's.

We introduce a new measure called the branching of a PDA, which is similar to the minmax-measure, but also considers the multiplicities of the nondeterministic steps counted in the minmax-measure. Contrary to the two levels in the hierarchy of classes of languages accepted by PDA's with finite minmax-measure, our measure will yield an infinite hierarchy. The branching of a PDA is the analogon of a corresponding measure for finite automata by Goldstine, Kintala and Wotschke from [GKW90]. The branching of a computation is the product of the multiplicities of all the branchings of single moves in this computation. The branching of a word is the minimum of branchings of accepting computations on this word. The branching of a PDA is the supremum of the branchings of all words accepted by this PDA.

We then prove that PDA's with branching k describe exactly the union of k DCFL's and that PDA's with infinite branching describe general CFL's. This means that, regarding the class of describable languages, it is equivalent whether the nondeterminism in a PDA with branching k is distributed over its computations in any way or the nondeterminism consists of simply choosing one of k DPDA's. This is one reason why we consider the branching a more natural measure than the minmax-measure, if the amount of nondeterminism is finite. Another reason why we chose this measure is that the branching of a PDA reflects the amount of parallelism needed for a real-time simulation of this PDA, that is, the number of copies that have to be created during a simulation.

Moreover, we show that the classes of languages accepted by PDA's with ambiguity or branching bounded by finite constants form an infinite hierarchy. This is done by exhibiting languages for all k, k' with $k \leq k'$ that can be accepted by a PDA with ambiguity k and branching k' but by no PDA with ambiguity less than k or branching less than k'. We call these languages inherently ambiguous of degree k and inherently nondeterministic of degree k'.

Finally, we study the savings in complexity that can be achieved by describing the same language by PDA's with different amounts of ambiguity and nondeterminism. We will show that it sometimes might be preferable to use more ambiguity or more nondeterminism in the description of a given language than necessary because then the description can be much smaller.

Schmidt and Szymanski showed in [SS77] that (ambiguous) PDA's can achieve arbitrarily large savings over unambiguous PDA's, that is, for every recursive

function there is language such that the difference between the size of the smallest ambiguous PDA for this language and the size of the smallest unambiguous PDA is not bounded by this function. We then say that the tradeoff from ambiguous PDA's to unambiguous PDA's is nonrecursive. In addition to this result, it was shown by Borchhardt in [Bo92] that for all finite k, the tradeoff from CFG's with ambiguity k to CFG's with ambiguity less than k is nonrecursive. We show that this tradeoff still is nonrecursive, if CFG is replaced by PDA. We also know from Valiant, [Va76] that the tradeoff from unambiguous (nondeterministic) PDA's to deterministic PDA's is nonrecursive. We prove for all finite k that the tradeoff from unambiguous PDA's with branching k to PDA's with branching less than k is nonrecursive, too. On the other hand, we prove that the the tradeoff from PDA's with branching k to those with branching k and ambiguity k is at most exponential, that is, the size of the minimal PDA with branching k and ambiguity k is at most exponential in the size of the minimal PDA with branching k.

References

[Bo92] I. Borchhardt: *Nonrecursive Tradeoffs between Context-free Grammars with Different Constant Ambiguity*, Master Thesis, J.W. Goethe-Universität Frankfurt/Main, 1992 (in German)

[GKW90] J. Goldstine, C. Kintala, D. Wotschke: *On Measuring Nondeterminism in Regular Languages*, Information and Computation 86 (1990), p.179–194

[He94] C. Herzog: *Pushdown Automata with Bounded Nondeterminism or Bounded Ambiguity*, Master Thesis, J.W. Goethe-Universität Frankfurt/Main, 1994 (in German)

[SY93] K. Salomaa, S. Yu: *Limited Nondeterminism for Pushdown Automata*, EATCS Bulletin 50 (1993), p.186–193

[SS77] E.M. Schmidt, T. Szymanski: *Succinctness of Descriptions of Unambiguous Context-free Languages*, SIAM Journal on Computing 6 (1977), p.547–553

[Va76] L. Valiant: *A Note on the Succinctness of Descriptions of Deterministic Languages*, Information and Control 32 (1976), p.139–145

Counting Problems for Alternating Finite Automata

Markus Holzer*

Institut für Informatik
Technische Universität München
Arcisstr. 21, 80290 München

email: holzer@informatik.uni-tuebingen.de

During the past several years, the topic of "'counting"' has appeared in many different settings in complexity and formal language theory. For example, Álvarez and Jenner [1] studied the following counting problems:

1. The census function:
   - Given a finite automaton A and 1^n, how many words w up to length n are accepted by A?
2. The ranking function:
   - Given a finite automaton A and a word w, how many words lexicographically smaller than or equal to w are accepted by A?
3. The maximal word function:
   - Given a finite automaton A and a word w, what is the lexicographically greatest word smaller or equal to w accepted by A?

If the regular languages are coded by deterministic and nondeterministic finite automata the above problems are complete for logarithmic space-bounded counting classes, like $\#\mathcal{L}$, *span*-$\mathcal{L}$, and *opt*-$\mathcal{L}$ [1].

It is well-known, that the complexity of a problem heavily relies on the coding. We show that the above problems become complete for polynomially time-bounded counting classes like $\#\mathcal{P}$ and *opt*-$\mathcal{P}$, if the regular language is coded by an alternating finite automaton [2]. In addition we study two other counting problems, which were in investigated by Álvarez and Jenner [1], and show $\#\mathcal{P}$- and *opt*-$\mathcal{P}$-completeness.

*Neue Adresse: Wilhelm-Schickhard-Institut für Informatik, Eberhard-Karls-Universität Tübingen, Sand 13, 72076 Tübingen.

References

[1] C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107:3–30, 1993.

[2] M. Holzer. On the space and time complexity of some problems for alternating finite automata. Unpublished manuscript, Dezember 1993.

Graphgrammatik-Konzepte in der Therapieplanung für komplexe dynamische Prozesse

Klaus P. Jantke

HTWK Leipzig
FB Informatik
Mathematik & Naturwiss.
Postfach 66
04251 Leipzig

email: jantke@informatik.th-leipzig.de

Oksana Arnold

Universität Leipzig
Institut für Wirtschaftsinformatik
Marschnerstr. 31
04109 Leipzig

Mit dem wachsenden Niveau der Produktivkräfte steigt auch der Komplexitätsgrad der technischen Prozesse, die vom Menschen beherrscht werden müssen. Die wettbewerblichen Bedingungen führen darüber hinaus dazu, technische Anlagen noch stärker als bisher an ihre Leistungsgrenzen heranzufahren. Im Falle von Störungen wird der Druck größer, mit minimalen Verlusten in kürzester Zeit wieder ein Normalregime zu erreichen. Dadurch werden Menschen als Bediener komplexer dynamischer Systeme immer öfter in Entscheidungssituationen gebracht, die sie überfordern.

Die Notwendigkeit einer zunehmenden Computerunterstützung für die Überwachung und Steuerung komplexer dynamischer Prozesse ist offensichtlich.

Während computergestützte Diagnose von Störungen ein schon intensiv bearbeitetes Forschungs- und Anwendungsgebiet ist, steht die automatische Generierung von Plänen zur Behebung von Störungen noch weitestgehend am Anfang. Sie ist untrennbar mit anderen Problemen der Wissensverarbeitung wie etwa der Diagnose oder der Simulation verbunden.

Planung ist ein zentrales Gebiet der Künstlichen Intelligenz, aber Therapieplanung für komplexe dynamische Prozesse mit harten Echtzeitanforderungen und der Notwendigkeit simultaner Aktionen wird kaum beherrscht. In einer bemerkenswert großen Zahl von Arbeiten zur Planung ist nicht einmal ein formalisiertes Plankonzept vorhanden. In solchen Arbeiten ist es dann natürlich auch nicht möglich, genauer darauf einzugehen, ob etwa nicht nur die Nebenläufigkeit, sondern sogar die Gleichzeitigkeit bestimmter Aktionen gefordert ist und wie dies realisiert wird.

Im vorliegenden Ansatz wurde ein auf graphentheoretischen Begriffen aufbauender Planbegriff entwickelt, einem Planer zugrunde gelegt, implementiert und erfolgreich getestet. Die Implementierungen liegen sowohl in Lisp (Allegro Common Lisp) als auch in Prolog (Quintus Prolog) unter Unix auf Sun Workstations vor.

Es liegen bereits umfangreiche Beschreibungen des Ansatzes und der Ergebnisse vor. Eine besonders detaillierte Darstellung der Besonderheiten der Domäne, die die Spezifik des gewählten Ansatzes begründen, findet man in [AJ94c]. Es wird dort insbesondere herausgearbeitet, warum STRIPS-artige Planungsansätze (vgl. [FN71]) nicht in Betracht kommen. Einige vergleichende Fragestellungen zu Grundbegriffen der Planung findet man in [AJ94a]. Als ein Überblick über Planungsansätze, der eine Einordnung der vorliegenden Herangehensweise zumindest teilweise unterstützt, ist [Her93] zu empfehlen. Die logischen Grundlagen sind ausführlich in [Arn94] dargelegt worden, insbes. die temporallogischen Aspekte. Zur Einordnung der Arbeiten ist [SM88] zu empfehlen. Unter dem speziellen Gesichtspunkt der Programmsynthese und des induktiven Lernens ist [AJ94b] die geeignetste Darstellung. Zur Einordnung in das induktive Lernen ist [AS83] nach wie vor eine herausragende Quelle. Schließlich sei darauf verwiesen, daß eine erste Erörterung der graphentheoretischen Sicht des vorliegenden Planungsansatzes in [AJ94d] vorgenommen wurde. Wegen der im vorliegenden Beitrag aus Platzgründen fehlenden syntaktischen Details muß auf [AJ94c] bzw. [AJ94d] verwiesen werden.

Im Zentrum der Darstellung liegen die entwickelten graphentheoretischen Plankonzepte und ihre Anwendung für die Generierung von Therapieplänen. Für praktische Belange sind die theoretisch begründeten Eigenschaften der Sprache aller erzeugbaren Pläne interessant. Auf der Basis gerichteter, zyklenfreier Graphen werden hierarchisch strukturierte Graphen konstruiert. Im Unterschied zu bisherigen Konzepten wie Pin-Graphen ist unser Substitutionskonzept einfacher, dafür gibt es aber i.allg. alternative Ersetzungen, die nach Präferenzen geordnet sind. Es werden die folgenden Begriffe (vgl. [AJ94c], [AJ94d]) zugrundegelegt:

Definition 1

Eine *hierarchisch strukturierte Familie von Plänen* $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$ ist eine

endliche Menge von Pin–Graphen $\mathcal{G}_i = [V_i, E_i, P_i^{in}, P_i^{out}, C_i, sub_i]$ $(i = 1 \ldots k)$ so daß für alle $i \in \{1, \ldots, k\}$ gilt:

1. $\mathcal{G}_i' = [V_i, E_i]$ ist ein endlicher, gerichteter, zyklenfreier Graph mit den Knoten- und Kantenmengen V_i bzw. E_i.
2. $P_i^{in} \cup P_i^{out}$ sind definiert durch
   (a) $P_i^{in} = \{\, v \mid v \in V_i \wedge \neg\exists u \in V_i\, (\,(u,v) \in E_i\,)\,\}$
   (b) $P_i^{out} = \{\, v \mid v \in V_i \wedge \neg\exists u \in V_i\, (\,(v,u) \in E_i\,)\,\}$
3. Die Knoten in $C_i \subseteq V_i$ heißen Komplexknoten.
4. $sub_i : C_i \to 2^{\{1,\ldots,k\}} \setminus \emptyset$ ist eine Substitutionsabbildung, die angibt, welche Graphen $\mathcal{G}_j$ für welche Komplexknoten in C_i eingesetzt werden dürfen. □

Über den Substitutionsbegriff zur Expansion von sogen. Komplexknoten entsteht ein Ableitungs- bzw. Grammatik-Konzept. Pläne $\mathcal{G}$ sind Normalformen bzgl. dieses Ableitungskonzepts bzw. Elemente der jeweiligen formalen Sprache $\mathcal{L}(\mathcal{F})$. Eine hierarchisch strukturierte Familie von Graphen $\mathcal{F}$ definiert eine formale Sprache $\mathcal{L}(\mathcal{F})$, die über dem zugrundeliegenden Therapiewissen alle potentiell möglichen Pläne enthält.

Definition 2

Für jede hierarchisch strukturierte Familie von Plänen $\mathcal{F}$ existiert eine Halbordnung $\preceq_{\mathcal{F}}$ auf $\{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$, die angibt, welche Graphen entsprechend $\{sub_i\}_{i=1\ldots k}$ ineinander eingesetzt werden dürfen.

$$\mathcal{G}_i \preceq_{\mathcal{F}} \mathcal{G}_j \qquad \underline{\text{gdw.}} \qquad \exists c \in C_i\, (\, j \in sub_i(c)\,).$$

Die transitive Hülle von $\preceq_{\mathcal{F}}$ heißt $\preceq_{\mathcal{F}}^{+}$. □

Um disjunkte Vereinigungen bilden zu können, werden Knoten d eines Graphen $\mathcal{G}_j$ bei Substitution für einen Knoten c in $c.d$ umbenannt. Diese Notation wird auf Mengen elementweise fortgesetzt, d.h. $x.Y = \{x.y \mid y \in Y\}$.

Definition 3

Für jede hierarchisch strukturierte Familie von Plänen $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$, jeden Graphen $\mathcal{G}_i \in \mathcal{F}$, jeden Komplexknoten $c \in C_i$, und jeden Index eines einsetzbaren Graphen $j \in sub_i(c)$ wird definiert:
Die *Substitution von $\mathcal{G}_j$ in $\mathcal{G}_i$* für $c \in C_i$ liefert einen Pin–Graphen $\mathcal{G}_i[c \hookleftarrow \mathcal{G}_j]$ der Struktur $[V, E, P^{in}, P^{out}, C, sub]$, der wie folgt definiert wird:

1. $V = (\, V_i \setminus \{c\}\,) \;\cup\; c.V_j$
2. $c.E \;=\; \{\, c.d \mid d \in E\,\}$
3. $E = (\,(E_i \cup c.E_j) \setminus (V_i \times \{c\} \cup \{c\} \times V_i)\,) \cup$
   $((\{v \mid (v,c) \in E_i\} \times c.P_j^{in}) \cup (c.P_j^{out} \times \{v \mid (c,v) \in E_i\}))$

4. (a) $P^{in} = P_i^{in} \setminus \{c\} \cup c.P_j^{in}$ falls $c \in P_i^{in}$
   (b) $P^{in} = P_i^{in}$ falls $c \notin P_i^{in}$
5. (a) $P^{out} = P_i^{out} \setminus \{c\} \cup c.P_j^{out}$ falls $c \in P_i^{out}$
   (b) $P^{out} = P_i^{out}$ falls $c \notin P_i^{out}$
6. $C = (\, C_i \setminus \{c\}\,) \cup c.C_j$
7. $sub = sub_{i/C_i \setminus \{c\}} \cup c.sub_j$ □

Diese Begriffsbildung wird auf beliebige Graphen $\mathcal{G}$ der entsprechenden Struktur fortgesetzt, so daß im allgemeinen $\mathcal{G}[c \hookleftarrow \mathcal{G}_j]$ erklärt ist.

Definition 4

Substitutionen definieren eine *Ableitungsrelation* $\Rightarrow_{\mathcal{F}}$ zwischen Pin–Graphen.

Für $\mathcal{G}$ und $\mathcal{G}'$ schreibt man $\mathcal{G} \Rightarrow_{\mathcal{F}} \mathcal{G}'$ gdw. $\exists\, \mathcal{G}_j \in \mathcal{F} : \mathcal{G}' = \mathcal{G}[c \hookleftarrow \mathcal{G}_j]$.

Mit $\Rightarrow_{\mathcal{F}}^{+}$ wird die transitive Hülle von $\Rightarrow_{\mathcal{F}}$ bezeichnet.

Falls $\mathcal{G}$ irreduzibel ist (in Normalform) bzgl. $\Rightarrow_{\mathcal{F}}^{+}$, schreibt man $\mathcal{G}\!\downarrow_{\mathcal{F}}$. □

Eine *ausgezeichnete Familie* („rooted family“) definiert die Teilsprache aller derjenigen Pläne, die zu einer festen Zielspezifikation erzeugbar sind. Ein *hierarchisch strukturierter Plan* $\mathcal{P}$ ist eine ausgezeichnete Familie, die nur noch eine einzige Normalform besitzt $\mathcal{G}$, also eine Sprache der Kardinalität 1 definiert. Das ist das Ergebnis der Plangenerierung. Der eigentliche Plan ist die zugehörige Normalform. Wegen der Szenarien der verschiedenartigen Benutzung der beiden Ergebnisformen der Planung muß auf [AJ94b] (evtl. auch [AJ94c] oder [AJ94d]) verwiesen werden.

Definition 5

Für jede hierarchisch strukturierte Familie von Plänen $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$ und jeden Graphen $\mathcal{G}_i \in \mathcal{F}$ heißt $\mathcal{R} = [\mathcal{F}, \mathcal{G}_i]$ eine *ausgezeichnete Familie*,

gdw. $\forall \mathcal{G}_j \in \mathcal{F}\,(\, \mathcal{G}_i \neq \mathcal{G}_j \;\Rightarrow\; \mathcal{G}_i \preceq_{\mathcal{F}}^{+} \mathcal{G}_j\,)$.

Eine ausgezeichnete Familie $\mathcal{P} = [\mathcal{F}, \mathcal{G}_i]$ heißt *hierarchisch strukturierter Plan*,

gdw. $\mathcal{G}_i$ hat eine eindeutige Normalform $\mathcal{G}^*$ bzgl. $\Rightarrow_{\mathcal{F}}^{+}$. □

Damit sind die folgenden grundlegenden Sprachkonzepte nahegelegt.

Definition 6

Eine hierarchisch strukturierte Familie von Plänen $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$ und eine ausgezeichnete Familie $\mathcal{R} = [\mathcal{F}, \mathcal{G}_i]$ definiert jeweils eine *formale Graph-Sprache* wie folgt:

$$\mathcal{L}(\mathcal{F}) \;=\; \{\, \mathcal{G} \mid \exists \mathcal{G}_i \in \mathcal{F}\,(\, \mathcal{G}_i \Rightarrow_{\mathcal{F}}^{+} \mathcal{G}\,)\,\}$$

$\mathcal{L}(\mathcal{R}) \;=\; \{\, \mathcal{G} \mid (\mathcal{G}_i \Rightarrow_{\mathcal{F}}^{+} \mathcal{G} \,\}$ □

Die der Therapieplangenerierung zugrundeliegenden formalen Sprachen über einer gegebenen statischen Wissensrepräsentation sind kontext-frei. Diese Erkenntnis erlaubt eine entsprechende Behandlung in den unterschiedlichen Phasen der Wissensverarbeitung im Rahmen des Konzepts der *wissensbasierten Prozeßführung*. So existieren z.B. für die Phase des Erwerbs von Therapiewissen (d.h. beim Aufbau der Grammatik durch Bestimmung der Graphen in $\mathcal{F}$) Standardverfahren für Korrektheitstest, etwa für den Test, ob $\mathcal{L}(\mathcal{F})$ nicht leer ist.

Die bisher eingeführten Konzepte sind eigentlich nur die grundlegendsten. Auf ihnen baut eine nächste Ebene von Konzepten auf, die dazu dienen, nur solche Therapiepläne zu generieren, die mit dem verfügbaren Prozeßwissen *konsistent* sind. Eine kurze Motivation muß genügen:

Man strebt in der Planung stets solche Pläne an, die ausführbar sind, und das Ziel, zu dessen Erreichung sie generiert wurden, auch erreichen. In einfacheren Bereichen und in Spielwelten ist die Ausführbarkeit von Plänen beweisbar, so daß das Problem der Plankorrektheit deduktiv behandelt werden kann. Aufgrund der Spezifika der Domäne verfahrenstechnischer Prozesse (vgl. insbes. [AJ94b]) kann man in solchen Anwendungen jedoch die Ausführbarkeit von Plänen i.allg. nicht beweisen. Hier sei aus der Vielzahl der Gründe nur einer zur Illustration genannt: Zielgrößen verfahrenstechnischer Prozesse können meist nicht direkt eingestellt werden. Statt dessen muß man sie über Stellgrößen indirekt beeinflussen. Wenn aber der zu steuernde Prozeß gestört ist (und gerade dann soll ja die Therapieplanung eingesetzt werden), beherrscht man die funktionalen Zusammenhänge zwischen Stell- und Zielgrößen typischerweise nicht vollständig. Demzufolge haben generierte Therapiepläne immer einen hypothetischen Charakter. Therapieplangenerierung für komplexe dynamische Prozesse ist inhärent *induktiv* (vgl. [AJ94b]).

Therapieplanung braucht aber sinnvollerweise ein Qualitätskriterium für Pläne. Dieses Kriterium ist die *Konsistenz* in Bezug auf eine gegebene Wissensbasis. Auf der Basis einer entsprechenden Logik, die ausführlich in [Arn94] beschrieben ist, wird aus der Sprache aller erzeugbaren Pläne die Teilsprache $\mathcal{L}_{\mathcal{TR}}^{cons}(\mathcal{F})$ der konsistenten Pläne (vgl. [AJ94b]) ausgezeichnet. $\mathcal{TR}$ referenziert dabei die als Wissensbasis vorausgesetzte sogen. Technologierepräsentation.

These: Bei einer Spezifikation des Konsistenzbegriffs, der komplexen dynamischen Prozessen gerecht wird, muß sich die Sprache der konsistenten Therapiepläne $\mathcal{L}_{\mathcal{TR}}^{cons}(\mathcal{F})$ i.allg. als nicht mehr kontext-frei erweisen.

Die Autoren sind gegenwärtig u.a. mit dem Studium derartiger Konsistenzbegriffe und der zugehörigen Graph-Sprachen befaßt.

Literatur

[AJ94a] Oksana Arnold and Klaus P. Jantke. Grundbegriffe der Planung I. Studie der Forschungsgruppe Algorithmisches Lernen, HTWK Leipzig (FH), FB Informatik, Mathematik & Naturwissenschaften, August 1994. Studie #01/94, Version 1.0.

[AJ94b] Oksana Arnold and Klaus P. Jantke. Therapy plan generation as program synthesis. In Setsuo Arikawa and K.P. Jantke, editors, *Algorithmic Learning Theory including Analogical and Inductive Inference, AII'94 & ALT'94*, volume 872 of *LNAI*, pages 40–55. Springer–Verlag, 1994.

[AJ94c] Oksana Arnold and Klaus P. Jantke. Therapy plan generation in complex dynamic environments. ICSI Report TR-94-054, International Computer Science Institute, Berkeley, California, October 1994.

[AJ94d] Oksana Arnold and Klaus P. Jantke. Therapy plans as hierarchically structured graphs. In *Fifth International Workshop on Graph Grammars and their Application to Computer Science, Williamsburg, Virginia, USA*, November 1994.

[Arn94] Oksana Arnold. A logic of constraints for dynamic process control. WISCON Report 09/94, HTWK Leipzig (FH), Fachbereich IMN, December 1994.

[AS83] Dana Angluin and Carl H. Smith. A survey of inductive inference: Theory and methods. *Computing Surveys*, 15:237–269, 1983.

[FN71] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to theorem proving in problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Her93] Joachim Hertzberg. KI-Handlungsplanung – Woran wir arbeiten, und woran wir arbeiten sollten. In Otthein Herzog, Thomas Christaller, and Dieter Schütt, editors, *Grundlagen und Anwendungen der Künstlichen Intelligenz. 17. Fachtagung für Künstliche Intelligenz (KI'93)*, pages 3–27. Springer-Verlag, 1993.

[SM88] Yoav Shoham and D. McDermott. Problems in formal temporal reasoning. *Artificial Intelligence*, 36:49–61, 1988.

Dependence Orders for Computations of Concurrent Automata*

Felipe Bracho
IIMAS
Universidad Nacional
Autónoma de México
01000 México D.F.

Manfred Droste and Dietrich Kuske
Institut für Algebra
TU Dresden
Mommsenstr. 13
D-01062 Dresden

email: {droste,kuske}@math.tu-dresden.de

In the theory of concurrency, many different models have been investigated intensively including, *e.g.*, Petri nets, CCS and CSP. The behaviour of Petri nets led Mazurkiewicz ([Ma77, Ma85]) to the investigation of trace alphabets and their associated trace monoids, a mathematical model for the sequential behaviour of a parallel system in which the order of two independent actions is regarded as irrelevant. *Trace theory* has now a well–developed mathematical theory, see [AR88, Di90] for surveys. One of its foundational results ([Ma85]), used in many difficult applications (*cf.* [Oc85, Di90]) is that each element of the (algebraically defined) trace monoid has a graph–theoretical representation. Here, we generalized this result and related versions of it to the context of automata with concurrency relations.

Let us recall basic notions of trace theory and of automata with concurrency relations. As introduced by Mazurkiewicz [Ma85], trace alphabets are pairs (E, D) consisting of a set E of unlabeled actions or events and a reflexive symmetric binary relation D in E indicating when two actions are dependent. Two sequences ab and ba are declared equivalent if $(a, b) \notin D$. This generates a congruence $\sim$ on the free monoid $E^\star$ of all words over E, and the quotient $M(E, D) := E^\star/\sim$ is called the *trace monoid* (or free partially commutative monoid) over (E, D).

In trace alphabets, a single binary relation on E is used to represent the concurrency relation for all pairs of actions. Here, we will consider a more general model of labeled transition systems in which the concurrency relation depends not only on the two arriving actions, but also on the present state of the system. An *automaton with concurrency relations* is a tuple $\mathcal{A} = (Q, E, T, \|)$ where Q

*An extended abstract of these results appeared as [BDK95]

is the set of states, E as before the set of events or actions, $T \subseteq Q \times E \times Q$ the transition relation (assumed deterministic), and $\|= (\|_q)_{q \in Q}$ is a collection of concurrency relations $\|_q$ for E, indexed by the possible states $q \in Q$. Let $\mathrm{CS}(\mathcal{A})$ comprise all finite computation sequences of $\mathcal{A}$, with concatenation as (partially defined) monoid operation. We declare two sequences $(p,a,q)(q,b,r)$ and $(p,b,q')(q',a,r)$ equivalent, if $a \|_p b$. As before, this induces a congruence $\sim$ on $\mathrm{CS}(\mathcal{A})$; thus intuitively, two computation sequences are equivalent, if they represent "interleaved views" of a single computation. The quotient $\mathrm{M}(\mathcal{A}) = \mathrm{CS}(\mathcal{A})/\sim \cup \{0\}$ (formally supplemented with 0 to obtain an everywhere defined monoid operation) is called the *concurrency monoid associated with* $\mathcal{A}$.

Automata with concurrency relations were introduced and studied in [Dr90, Dr92, BD93, BD94]. Their domains of computation sequences are closely related with event domains and dI–domains arising in denotational semantics of programming languages. These automata also generalize asynchronous transition systems ([Be87, WN94]). For applications of related structures we refer to [Le78, BC88, PSS90, KP90]. Very recently, a formalization using several independence relations of the operational semantics of Occam was given in [BR94].

It seems that many results of the literature for trace monoids $\mathrm{M}(E,D)$ can be generalized, under suitable assumptions, to concurrency monoids $\mathrm{M}(\mathcal{A})$. An adjunction between automata with concurrency relations and place/transition–systems generalizing the one between asynchronous transition systems and Petri–nets (*cf.* [NRT92, WN94]) was given in [DS93]. The results of [Oc85, GRS92] on recognizable and on aperiodic languages in trace theory could be generalized to concurrency monoids in [Dr94a, Dr94b]. Now by a fundamental result in trace theory, mentioned before, the elements of a trace monoid can be represented pictorially nicely by certain labeled graphs, or even labeled partially ordered sets. There is even a multiplication of (isomorphism classes of) such graphs, yielding a monoid which turns out to be isomorphic to the trace monoid $\mathrm{M}(E,D)$.

These results we wish to generalize here to concurrency monoids $\mathrm{M}(\mathcal{A})$. For this, a useful (and almost necessary) assumption is that $\mathcal{A}$ is stably concurrent. This means that the concurrency relations of $\mathcal{A}$ depend locally (but not globally) on each other. As shown in [Ku94b], stably concurrent automata generate precisely the class of dI–domains, and these distributivity properties will be crucial here.

We now give a summary of our results. Let $\mathcal{A}$ be a fixed stably concurrent automaton and γ a computation sequence of $\mathcal{A}$. First we define two labeled partial orders $\mathrm{DO}(\gamma)$ and $\mathrm{PR}(\gamma)$ associated with γ as follows. We let $\mathrm{DO}(\gamma)$ comprise all enumerated occurrences (a,i) of actions $a \in E$ within the computation sequence γ. We put $(a,i) \sqsubseteq (b,j)$, if in each computation sequence δ equivalent to γ the i–th occurrence of a precedes the j–th occurrence of b. Finally, we label the element (a,i) with a. This provides a straightforward generalization of the dependence or occurrence orders (see [Ma85]) known in trace theory.

Then we define a labeled partial order $\mathrm{PR}(\gamma)$ as follows. For any $x, y \in \mathrm{M}(\mathcal{A})$

put $x \leq y$ if x divides y, i.e. there is $z \in \mathrm{M}(\mathcal{A})$ with $x \cdot z = y$. This partial order is known as the prefix ordering in trace theory (see [Ma85]). Given a computation sequence γ with equivalence class $[\gamma]$ in $\mathrm{M}(\mathcal{A})$, we may associate with it the partially ordered set $(\mathrm{Pr}(\gamma), \leq)$ of all prime elements of the distributive lattice $([\gamma]\downarrow, \leq)$ (see [Ku94b]). This procedure is standard in lattice theory ([Bi73]) and also known from the theory of stable event structures ([Wi87]). This poset $(\mathrm{Pr}(\gamma), \leq)$ carries a natural labeling function taking values in E, which completes the definition of $\mathrm{PR}(\gamma)$. One of our main results is that, although defined quite differently, the labeled partial orders $\mathrm{DO}(\gamma)$ and $\mathrm{PR}(\gamma)$ are isomorphic. This result provides the basis for our further investigation of the dependence order $\mathrm{DO}(\gamma)$. We show that the linear extensions of the partial order $\mathrm{DO}(\gamma)$ precisely give rise to the computation sequences δ equivalent to γ. In particular, two computation sequences γ and δ are equivalent if (and, clearly, only if) their dependence orders $\mathrm{DO}(\gamma)$ and $\mathrm{DO}(\delta)$ coincide. In [BDK94] we characterize for an arbitrary labeled poset when it is isomorphic to a dependence order $\mathrm{DO}(\gamma)$, for some computation sequence γ. Finally, we define a multiplication on the set of (isomorphism classes of) dependence orders $\mathrm{DO}(\gamma)$ and show that we obtain a monoid isomorphic to the concurrency monoid $\mathrm{M}(\mathcal{A})$. This generalizes classical results of Mazurkiewicz ([Ma85, Di90]).

From the present work the question arises which further results of trace theory resting on the representation of traces by graphs can now be transferred into the more general context of concurrent automata. This has recently been achieved for logical definability of recognizable and aperiodic languages. Also, it is possible to represent the infinite computations of a stably concurrent automaton by dependence orders. This result is the basis for the investigation of recognizable languages of infinite computations.

References

[AR88] Aalbersberg, I.J. and G. Rozenberg: *Theory of traces.* Theor. Comp. Science 60 (1988), 1-82.

[Be87] Bednarczyk, M.: *Categories of asynchronous systems.* Ph.D. thesis, University of Sussex, 1987.

[BC88] Boudol, G. and I. Castellani: *A non–interleaving semantics for CCS based on proved transitions.* Fundam. Inform. 11 (1988), 433–452.

[BR94] Börger, E. and D. Rosenzweig: *Occam: Specification and compiler correctness. Part I: Simple mathematical interpreters.* To appear.

[BD93] Bracho, F. and M. Droste: *From domains to automata with concurrency.* In: 20th ICALP, LNCS 700, Springer, 1993, pp. 669–681.

[BD94] Bracho, F. and M. Droste: *Labelled domains and automata with concurrency.* Theor. Comp. Science, in press.

[BDK94] Bracho, F., M. Droste and D. Kuske: *Representation of computations in concurrent automata by dependence orders.* Internal report of TU Dresden,

MATH-AL-3-1994.

[BDK95] Bracho, F., M. Droste and D. Kuske: *Dependence orders for computations of concurrent automata.* In: STACS'95, Lecture Notes in Computer Science vol. 900, Springer 1995, 467-478.

[Bi73] Birkhoff, G.: *Lattice Theory.* AMS, Providence, 1973.

[Di90] Diekert, V.: *Combinatorics on Traces.* LNCS 454, Springer, 1990.

[Dr90] Droste, M.: *Concurrency, automata and domains.* In: *17th ICALP*, LNCS 443, Springer, 1990, pp. 195–208.

[Dr92] Droste, M.: *Concurrent automata and domains.* Intern. J. of Found. of Comp. Science 3 (1992), 389–418.

[Dr94a] Droste, M.: *A Kleene theorem for recognizable languages over concurrency monoids.* In: 21st ICALP, LNCS 820, Springer 1994, pp. 388–399. (Full version to appear in Theor. Comp. Science)

[Dr94b] Droste, M.: *Aperiodic languages over concurrency monoids.* To appear.

[DS93] Droste, M. and R.M. Shortt: *Petri nets and automata with concurrency relations – an adjunction.* In: *Semantics of Programming Languages and Model Theory* (M. Droste, Y. Gurevich, eds.), Gordon and Breach Science Publ., OPA (Amsterdam), 1993, 69–87.

[GRS92] Guaiana, G., A. Restivo and S. Salemi: *Star–free trace languages.* Theor. Comp. Science 97 (1992), 301–311.

[KP90] Katz, S. and D. Peled: *Defining conditional independence using collapses.* In: *Semantics for Concurrency* (M.Z. Kwiatowska, M.W. Shields, R.M. Thomas, eds.), Proc. of the Int. BCS–FACS Workshop at Leicester, 1990, Springer, 262–280.

[Ku94a] Kuske, D.: *Modelle nebenläufiger Prozesse – Monoide, Residuensysteme und Automaten.* Dissertation, Universität GHS Essen, 1994.

[Ku94b] Kuske, D.: *Nondeterministic automata with concurrency relations.* Colloquium on Trees in Algebra and Programming, LNCS 787, Springer, 1994, 202–217.

[Le78] Levy, J.J.: *Réductions correctes et optimales dans le λ–calcul.* Ph.D. Thesis, Université Paris VII, 1978.

[Ma77] Mazurkiewicz, A.: *Concurrent program schemes and their interpretation.* DAIMI Report PB–78, Aarhus University, Aarhus, 1977.

[Ma85] Mazurkiewicz, A.: *Trace theory.* In: *Advanced Course on Petri Nets*, LNCS 188, Springer, 1985, pp. 279–324.

[NRT92] Nielsen, M., G. Rosenberg and P.S. Thiagarajan: *Elementary transition systems.* Theor. Comp. Science 96 (1992), 3–33.

[Oc85] Ochmanski, E.: *Regular behaviour of concurrent systems*, Bull. Europ. Assoc. for Theoretical Computer Science 27 (1985), 56–67.

[PSS90] Panangaden, P., V. Shanbhogue and E.W. Stark: *Stability and sequentiability in dataflow networks.* In: *17th ICALP*, LNCS 443, Springer, 1990, pp. 308–321.

[S89] Stark, E.W.: *Connections between a concrete and an abstract model of concurrent systems.* In: *Proceedings of the 5th Conf. on the Mathematical Foundations of Programming Semantics*, LNCS 389, Springer, 1989, pp. 52–

74.

[Wi87] WINSKEL, G.: *Event structures.* In: *Petri Nets: Applications and Relationships to Other Models of Concurrency* (W. Brauer, W. Reisig, G. Rozenberg, eds.), LNCS 255, Springer, 1987, pp. 325–392.

[WN94] WINSKEL, G. and M. NIELSEN: *Models for concurrency.* In: *Handbook of Logic in Computer Science* (S. Abramsky, D.M. Gabbay, T.S.E. Maibaum, eds.), to appear.

Eingabeverfahren für parallele Automaten

Martin Kutrib
AG Informatik
Universität Gießen
Arndtstr. 2
35392 Gießen

Thomas Worsch
Fakultät für Informatik
Universität Karlsruhe
Am Fasangarten 5
76128 Karlsruhe

email: kutrib@informatik.uni-giessen.de

Aufgrund des heutigen technischen Entwicklungsstandes ist das Interesse an Modellen für massiv parallele Rechnersysteme gewachsen. Unter dem Oberbegriff *Polyautomaten* sind verschiedene modifizierte und verallgemeinerte Varianten der von J. von Neumann eingeführten Zellularräume zusammengefaßt [5, 3]. Polyautomaten können als Modelle für parallele Rechensysteme aufgefaßt werden, die über sehr viele einfache Verarbeitungselemente ohne gemeinsamen Speicher verfügen. Die Kommunikationsmöglichkeiten sind dabei derart eingeschränkt, daß globale Ergebnisse nur aus parallelen lokalen Entscheidungen gewonnen werden können.

Werden Polyautomaten hinsichtlich ihrer Fähigkeiten, formale Sprachen zu erkennen, betrachtet, so lassen sich bisher im wesentlichen zwei verschiedene Eingabeverfahren unterscheiden: Die einzelnen Zellen der Zellularautomaten lesen in einem initialen Zeitschritt jeweils ein Zeichen des Eingabewortes (*paralleler* Eingabemodus). In iterativen Arrays [1] ist eine Zelle ausgezeichnet, die die Zeichen der Eingabe nacheinander verarbeitet (*sequentieller* Eingabemodus).
Im Zusammenhang mit Pipeline-Verarbeitung schlägt R. Vollmar [4] zur Erhöhung des Durchsatzes einen *schrägen* Eingabemodus vor.

Wir betrachten ein zellulares System, bei dem das Eingabeverfahren selbst ein Parameter der Definition ist und konzeptionell dem Bereich zwischen parallelem und sequentiellem Eingabemodus zuzuordnen ist [2]. Für einige naheliegende Verfahren (formalisiert durch sogenannte Verteiler) werden erste Ergebnisse über den Einfluß auf die zur Berechnung notwendige Zeit und Hardware vorgestellt. Anhand der Sprache $L_a = \{\mathtt{a}^{2^{2^n}} \mid n \in \mathbf{N}\}$ kann gezeigt werden, daß im Vergleich mit Zellularautomaten gleichzeitig die benötigte Zeit und Hardware reduzieren läßt. Andererseits ist es für die Sprache der Palindrome nicht möglich, Hardware

(und somit Zeit) einzusparen.

Beschränkt man die Zahl der Zellen auf eine Konstante, dann werden mit einem *Modulo*-Verteiler genau die regulären Sprachen charakterisiert, während in die entsprechende Komplexitätsklasse des ähnlichen *Substring*-Verteilers auch echt kontextsensitive Sprachen fallen.

Weiterhin kann nachgewiesen werden, daß für obige Verteiler $\sqrt{n}$-Zeit nicht ausreicht, um alle regulären Sprachen zu erkennen. Dies ist aber mit $\sqrt{n}$-Zellen in $2\sqrt{n}$-Zeit möglich, wobei die Komplexitätsklasse wiederum echt kontextsensitive Sprachen enthält.

Literatur

[1] Cole, S. N. *Real-time computation by n-dimensional iterative arrays of finite-state machines*. IEEE Conference Record of 7^{th} Annual Symposium on Switching and Automata Theory, 1966, pp. 53–77.

[2] Kutrib, M. und Worsch, Th. *Investigation of different input modes for cellular automata*. Parcella '94, Akademie Verlag, Berlin, 1994.

[3] Vollmar, R. *Algorithmen in Zellularautomaten*. Teubner, Stuttgart, 1979.

[4] Vollmar, R. *Some remarks on pipeline processing by cellular automata*. Computers and Artificial Intelligence 6 (1987), 263–278.

[5] von Neumann, J. *Theory of Self-Reproducing Automata*. Herausgegeben und vervollständigt von A. W. Burks. University of Illinois Press, Urbana, 1966.

Non-Monotone Fixed-Point Definability and Tabular Recognition of Languages

Hans Leiß

Centrum für Informations- und Sprachverarbeitung
Universität München
Wagmüllerstr. 23
D-80538 München

email: leiss@cis.uni-muenchen.de

W.Rounds[2] has given uniform grammatical characterizations of the *EXPTIME* and *PTIME* languages in terms of positive inductive (least-fixed-point) definitions. From such a definition, he can read off time complexity bounds for language recognition by a deterministic Turing machine. Applied to context-free or head grammars, the method gives fairly poor bounds of $O(n^{12})$ resp. $O(n^{18})$.

We improve Rounds' analysis in two respects: first, we introduce a modified class of language definitions that allow restricted forms of negative elementary inductions, and second, we show how to build table-driven recognizers from such definitions. We thereby can, for example, recognize the boolean closure of context-free resp. head languages in $O(n^3)$ resp. $O(n^6)$ *RAM*-steps.

Our method is based on the existence of fixed-points for a class of 'bounded' operators that need not be monotone, but we assume a well-founded quasi-ordering $\leq$ on the underlying set, such as the subword-relation. Bounded fixed-point definability may also be useful in data base queries, since the stages of the induction only depend on the relation $\leq$ rather than the size of the structure.

1 Non-Monotone Operators with Fixed Points

By Tarski and Knaster's theorem, a monotone operator $\Gamma : 2^A \to 2^A$ has a least fixed point $\Gamma^\infty \subseteq A$, defined in stages $\Gamma^{<\alpha}$ by transfinite induction along the ordinals α. The fixed point can be equipped with a quasi-ordering by comparing elements $a \in \Gamma^\infty$ according to $|a|_\Gamma$, the least ordinal α such that $a \in \Gamma(\Gamma^{<\alpha})$. Our first observation is that on every set with a suitable quasi-ordering there is a natural class of operators with fixed points:

Definition 1.1 *Let $(A, \leq)$ be a quasi-ordering, i.e. a set A with a reflexive and transitive relation $\leq$. For $a, b \in A$, define $a < b$ iff $a \leq b \wedge b \not\leq a$ and $a \sim b$ iff $a \leq b \wedge b \leq a$, and for $S \subseteq A$ use*

$$S^{\leq a} := \{\, b \in S \mid b \leq a \,\}, \quad S^{<a} := \{\, b \in S \mid b < a \,\}, \quad S^{a} := \{\, b \in S \mid b \sim a \,\}.$$

We call $\leq$ a well-quasi-ordering *on A, if $\leq$ is a quasi-ordering and $<$ is well-founded. An operator $\Gamma := (\Gamma_1, \ldots, \Gamma_n)$ with components $\Gamma_i : 2^A \times \cdots \times 2^A \to 2^A$ is called* $<$-bounded, *if for each Γ_i, each $a \in A$ and all sets $S_1, \ldots, S_n \subseteq A$,*

$$a \in \Gamma_i(S_1, \ldots, S_n) \iff a \in \Gamma_i(S_1^{<a}, \ldots, S_n^{<a}).$$

We call Γ norm-bounded, *if $(A, \leq)$ is given by a norm $|\cdot| : A \to \eta$, with $a \leq b$ iff $|a| \leq_{On} |b|$. We write $S^{<|a|}$ when $\leq$ on A comes from a norm.*

Theorem 1.2 *Each $<$-bounded operator $\Gamma := (\Gamma_1, \ldots, \Gamma_n) : (2^A)^n \to (2^A)^n$ on a well-quasi-ordering $(A, \leq)$ has a unique fixed-point $\Gamma^\infty = (\Gamma_1^\infty, \ldots, \Gamma_n^\infty)$, where the $\Gamma_i^{<a}$ and Γ_i^∞ are simultaneously defined by*

$$\begin{aligned} \Gamma_i^{<a} &:= \bigcup \{\, \Gamma_i(\Gamma_1^{<b}, \ldots, \Gamma_n^{<b})^b \mid b < a \,\} \\ \Gamma_i^{\infty} &:= \bigcup \{\, \Gamma_i(\Gamma_1^{<a}, \ldots, \Gamma_n^{<a})^a \mid a \in A \,\} \end{aligned}$$

In fact, $\Gamma_i^\infty = \{\, b \in A \mid b \in \Gamma_i(\Gamma_1^{<b}, \ldots, \Gamma_n^{<b}) \,\}$.

2 Definable Bounded Fixed-Point Operators

For every relational first-order language, we introduce a class of *bounded* fixed-point formulas to define bounded operators. For the case of 'concatenation' formulas (viewing a and $\cdot$ as syntactic sugar for relation symbols):

Definition 2.1 *The* concatenation bounded fixed-point formulas *over Σ are*

$$\begin{aligned} \varphi \;:\equiv\; & x = a \mid x_1 = x_2 \mid x_1 = x_2 \cdot x_3 \mid S(x) && (a \in \Sigma) \\ & \mid \neg\varphi \mid (\varphi_1 \vee \varphi_2) \mid \exists x_1 < x_2 \; \varphi && (x_1 \not\equiv x_2) \\ & \mid \mu(S_1, \ldots, S_n)(\lambda x_1 \, \varphi_1, \ldots, \lambda x_n \, \varphi_n)(x), \end{aligned}$$

where in the last clause, the set variables S_i are pairwise distinct, $Indfree(\varphi_i) \subseteq \{\, x_i \,\}$, and no φ_i contains an atomic subformula $S(x_i)$ with S free in φ_i.

These formulas are interpreted in $\mathcal{L} = (\Sigma^*, \cdot, a)_{a \in \Sigma}$, the set of all finite strings over the alphabet Σ. Depending on which relation $<$ on Σ^* we use to interprete $<$ on $\mathcal{L}$, we talk of *length-bounded* and *subword-bounded* formulas.

Theorem 2.2 *On each structure $\mathcal{A}$ where $\leq$ is a well-quasi-ordering on A, every bounded fixed-point-formula $\varphi(x) := \mu(S_1, \ldots, S_n)(\lambda x_1\, \varphi_1, \ldots, \lambda x_n\, \varphi_n)(x)$ defines a simultaneous $<$-bounded operator $\Gamma_\varphi := (\Gamma_{\varphi_1}, \ldots, \Gamma_{\varphi_n})$ by*

$$\Gamma_{\varphi_i}(S_1, \ldots, S_n) := \{\, a \in A \mid \mathcal{A} \models \varphi_i(a, S_1, \ldots, S_n) \,\}.$$

In particular, the meaning of φ on $\mathcal{A}$ is well defined:

$$\mathcal{A} \models \mu(S_1, \ldots, S_n)(\lambda x_1\, \varphi_1, \ldots, \lambda x_n\, \varphi_n)(a) \iff a \in \Gamma^\infty_{\varphi_1}.$$

Fixed points Γ^∞_φ need not exist if we allow a subformula $S_j(x_i)$ in φ_i; we have to exclude $\mu(S)(\lambda x.\neg S(x))(x)$, but we can use formulas like $\mu(S)(\lambda x.\forall y < x.\neg S(y))(x)$. In contrast to least fixed point formulas, bounded formulas satisfy the following 'local substructure invariance' property:

Theorem 2.3 *If $\varphi(x) = \mu(S_1, \ldots, S_n)(\lambda x_1\, \varphi_1, \ldots, \lambda x_n\, \varphi_n)(x)$ is a bounded formula, then for any structure $\mathcal{A}$ with a well-quasi-ordering $\leq$ and any $a \in A$,*

$$\mathcal{A} \models \varphi(a) \iff \mathcal{A}^{\leq a} \models \varphi(a),$$

where $\mathcal{A}^{\leq a}$ is the substructure of $\mathcal{A}$ with universe $A^{\leq a}$

For $\mathcal{L}^{\leq w}$, with $\leq$ as the subword relation, this invariance captures the *intuitive* meaning of 'context-free', but goes beyond the formal notion. Our choice of bounded formulas can be justified by a syntactic characterization of those first-order formulas $\varphi(x, S)$ that are both $\leq$-local and define $<$-bounded operators.

3 Tabular Recognition of Bounded Fixed-Point Languages

Let $\varphi(x) := \mu(S_1, \ldots, S_n)(\lambda x_1\, \varphi_1, \ldots, \lambda x_n\, \varphi_n)(x)$ be a bounded fixed-point formula. To evaluate $\mathcal{L} \models \varphi(w)$, we can use $\mathcal{L}^{\leq w}$ and proceed as follows:

First, we build a 'table' $\Gamma^{<w}_\varphi$ of all $\Gamma^\infty_\varphi(v)$ for $v < w$, where $\Gamma^\infty_\varphi(v)$ is the bit-vector of the boolean values of $v \in \Gamma^\infty_{\varphi_1}, \ldots, v \in \Gamma^\infty_{\varphi_n}$: in a loop through all $v < w$, respecting $<$, compute the (missing) bits $\Gamma^\infty_{\varphi_i}(v) = \Gamma_{\varphi_i}(\Gamma^{<v}_\varphi)^v$ using the stored subtable $\Gamma^{<v}_\varphi$ as $\vec{S}$, and store the results. Second, evaluate φ_1 on input w, using the table $\Gamma^{<w}_\varphi$ as $\vec{S}$.

Clearly, $\{\, w \in \Sigma^* \mid \mathcal{L}^{\leq w} \models \varphi(w) \,\} \in \mathit{PTIME}$, and if $|w|$ is the length of $w \in \Sigma^*$, $\{\, w \in \Sigma^* \mid \mathcal{L}^{\leq |w|} \models \varphi(w) \,\} \in \mathit{EXPTIME}$.

On $\mathcal{L}^{\leq w}$, a bounded induction needs a stage $\Gamma^{<v}_\varphi$ for each $v < w$, which gives a small table of $O(|w|^2)$ many fields – the recogniton table for w in the algorithm of Cocke-Younger-Kasami, if $\varphi(x)$ is a context-free grammar in Chomsky normal form. To ensure that stage $\Gamma^{<v}_\varphi$ can be computed efficiently from stages $\Gamma^{<u}_\varphi$, $u < v$, individual quantification in bounded formulas has to be restricted further:

Definition 3.1 *A bounded formula* $\mu(S_1,\ldots,S_n)(\lambda x\varphi_1,\ldots,\lambda x\varphi_n)(x)$ *is* a decomposition grammar (in binary normal form) *if each* $\varphi_i(x,\vec{S})$ *is a boolean combination of formulas*

$$\exists x_1 < x \ldots \exists x_k < x\,(x = t(x_1,\ldots,x_k) \wedge \psi(x_1,\ldots,x_k,S_1,\ldots,S_n)), \qquad (1)$$

where $x,x_1,\ldots,x_k$ *are pairwise distinct individual variables, either* $k=0$ *and* $t \equiv a$ *for some* $a \in \Sigma$, *or* $k=2$ *and* $t \equiv x_1 \cdot x_2$, *and* $\psi(\vec{x},\vec{S})$ *is a conjunction of formulas* $S_i(x_j)$ *and their negations.*

Theorem 3.2 *The class of languages definable by decomposition grammars contains the context-free languages, is closed under the boolean operations* $\cup,\cap,\neg$, *the regular operations of* $\cdot,^*$, *and under substitution.*

In the description of natural languages, one sometimes uses *segmented strings* $(w_1,\ldots,w_m)$, i.e. strings $w_1\cdots w_m$ segmented into several consecutive substrings $w_1,\ldots,w_m$, and a relational interpretation of the syntactic categories. Let Op be a set of operations $\circ : (\Sigma^*)^m \times (\Sigma^*)^m \to (\Sigma^*)^m$ that are definable by

$$(x_1,\ldots,x_m) \circ (y_1,\ldots,y_m) := (v_1,\ldots,v_m), \qquad (2)$$

where $v_1,\ldots,v_m$ are nonempty words over $\{x_1,\ldots,x_m,y_1,\ldots,y_m\}$ and each x_i and y_j occurs exactly once in $v_1\cdots v_m$. Bounded formulas are interpreted in

$$\mathcal{L}_m := ((\Sigma^*)^m,\ \circ,\ (a,\epsilon,\ldots,\epsilon),\ldots,(\epsilon,\ldots,\epsilon,a))_{a\in\Sigma,\circ\in Op}.$$

Now let t in (1) be $(\epsilon,\ldots,a,\ldots,\epsilon)$ for some $a \in \Sigma$ or $x_1 \circ x_2$ for some $\circ \in Op$. To interprete $<$, use the natural notion of subword as factor.

On a *RAM*, the complexity of computing the table M for an input $w \in \mathcal{L}_m$ is proportional to (a) the number of $v \leq w$ times (b) the relative cost of computing a field $M(v)$, for which one has to determine all splittings of v into subwords $\vec{v}$ such that $v = t(\vec{v})$. ¿From the linearity condition in (2), one obtains:

Theorem 3.3 *For any normal form decomposition grammar* $\varphi(x)$ *for* m*-segmented strings, a recognition table* M *for input* $(w_1,\ldots,w_m)$ *can be constructed in* $O(\max_i |w_i|^{3m})$ *many RAM-steps.*

For $m=1$, this gives the familiar $O(|w|^3)$ bound for language recognition with respect to context-free grammars in Chomsky normal form. For $m=2$, we get an $O(|(w_1,w_2)|^6)$ bound for head grammars. The bound for CF-recognition holds also for a DTM, but we have not checked whether the other cases also do.

References

[1] K. Roach. Formal properties of head grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*, pages 293–348. John Benjamins, Amsterdam 1987.

[2] W. Rounds. A logic for linguistic descriptions and an analysis of its complexity. *Computational Linguistics*, 14(4):1–9, 1988.

Strategien für unendliche Spiele auf endlichen Graphen

Helmut Lescow

Institut für Informatik und
Praktische Mathematik
Universität Kiel
24098 Kiel

email: hel@informatik.uni-kiel.d400.de

Unendliche Spiele eignen sich als Modell für viele Anwendungen in der Informatik, in denen nicht-terminierende interaktiv aufgebaute Berechnungen auftreten. Z.B. lassen sich Betriebssysteme in dieser Weise modellieren. Wir beschränken uns auf den Fall zweier "Parteien", etwa eines Benutzers und des Systems, die abwechselnd Züge, nämlich Eingaben bzw. Berechnungen und Ausgaben, machen. Wir erhalten so eine unendliche Folge von Aktionen. Der Gewinn der Partie hängt nun davon ab, ob die gesamte Folge eine vorgegebene Bedingung erfüllt.

Auf abstrakter Ebene betrachten wir unendliche 2-Personen-Spiele mit vollständiger Information nach Gale und Stewart [GS53]. Die Spieler 0 und 1 wählen abwechseld Aktionen aus einem Alphabet, so daß die geamte Partie durch ein ω-Wort charakterisiert ist. Die Gewinnbedingung ist durch eine ω-Sprache gegeben und besagt, daß Spieler 1 gewinnt, falls das sich durch die Partie ergebende Wort zur Sprache gehört. Ein klassisches Beispiel für eine Gewinnbedingung ist die Angabe einer ω-Sprache durch einen deterministischen Mullerautomaten, dessen graphische Repräsentation man sich als das "Spielbrett" vorstellen kann, auf dem die Spieler eine Figur entlang der Kanten abwechselnd bewegen. Eine Strategie ist nun eine Funktion, die in Abhängigkeit vom endlichen bisherigen Spielverlauf die nächste Aktion eines Spielers bestimmt. Eine Gewinnstrategie ist eine Strategie, die einem Spieler für beliebige Züge des Mitspielers den Gewinn der Partie garantiert. Für ein gegebenes Spiel sind die folgenden Fragen zu klären:

1. Gibt es für einen der Spieler eine Gewinnstrategie?

2. Wenn es eine Gewinnstrategie gibt, für welchen der Spieler stellt sie eine Gewinnstrategie dar?

3. Wie sieht eine (effiziente) Gewinnstrategie aus?

Büchi und Landweber konnten diese Fragen für Spiele mit einer durch Mullerautomaten gegebenen Gewinnbedingung in [BL69] lösen. Sie konnten zeigen, daß solche Spiel determiniert sind, d.h. es gibt immer eine Gewinnstrategie für einen der Spieler, die durch einen endlichen I/O-Automaten realisierbar ist. Außerdem geben sie ein Verfahren an, das entscheidet, welcher Spieler eine solche Strategie besitzt und das einen Strategie-Automaten berechnet.

Die Spieldarstellung, die hier als Variante zum Mullerautomaten benutzt werden soll, geht auf McNaughton [McN93] zurück. Statt eines deterministischen Automaten dient ein bipartiter gerichteter endlicher Graph mit Knotenmenge $Q = Q_0 \dot{\cup} Q_1$ und unbeschrifteten Kanten als “Spielbrett”. Die Aktionen bestehen dann aus der Wahl eines adjazenten Knotens; in $q \in Q_0$ wählt Spieler 0 einen Nachfolgerknoten aus Q_1 und umgekehrt. Mit diesem Graphen sind verschiedene Gewinnbedingungen für Spieler 1 realisierbar, z.B.: Spieler 1 gewinnt, falls

1. ein Knoten aus einer gegebenen Zustandsmenge $F \subseteq Q$ erreicht wird (offene Spiele);

2. eine Knoten aus einer gegebenen Zustandsmenge $F \subseteq Q$ unendlich oft erreicht wird (Büchi-Spiele);

3. die Menge der unendlich oft besuchten Knoten in einer gegebenen Menge $\mathcal{F} \subseteq 2^Q$ liegt (Muller-Spiele).

Für die offenen sowie die Büchi-Spiele lassen sich “no-memory-Strategien” finden, also Strategien, die für die Berechnung des Folgezustands nur den aktuellen Zustand benötigen. Geht man zur Darstellung 3. über, so existiert eine solche Strategie nicht. Für eine wichtige Klasse innerhalb der Mullerspiele geben wir aber ein Verfahren an, das einen Strategieautomaten mit polynomial großem Speicher in der Zustandszahl des Spielgraphen liefert. Standardkonstruktionen liefern Strategieautomanten mit exponentieller Zustandszahl. Anschließend zeigen wir, daß diese obere Schranke für die Speichergröße auch eine untere Schranke darstellt.

Es handelt sich dabei um Muller-Spiele, deren Schleifenmenge $\mathcal{F}$ gegen Obermengenbildung abgeschlossen ist. Wir nehmen dabei an, wir haben $\mathcal{F}$ gegeben mit k minimalen Mengen $F_1, \ldots, F_k$ und spielen auf einem Spielgraphen mit n Knoten. Wir zeigen, daß ein Speicher der Größe n^k genügt, um das Spiel zu gewinnen. Die Idee besteht darin, einen Speicher zu benutzen, der in jeder minimalen Menge ein momentanes Ziel angibt, also einen Speicher $M \subseteq F_1 \times \cdots \times F_k$.

Zur Berechnung der Strategie nutzen wir aus, daß wir wissen, daß wir für offene Spiele Strategien ohne zusätzlichen Speicher erhalten. Zu $T \subseteq Q$ sei $Force(T)$ die Menge der Zustände im Spielgraphen, für die es no-memory-Strategien gibt, die das Erreichen eines Knotens aus T erzwingen. Es ist außerdem bekannt, daß

Spieler 1 eine no-memory-Strategie besitzt, die von $Q \setminus Force(T)$ das Erreichen jedes Knotens aus T verhindert.

Es sei $I = \{1, \ldots, k\}$ eine Indexmenge. Wir berechnen für alle Tupel $m \in \mathsf{X}_{i \in I} F_i$ die Mengen $Force(T_m)$, wobei T_m die Menge der Knoten ist, die im Tupel m vorkommen. Falls $\bigcup_{i \in I} F_i \not\subseteq Force(T_m)$ ist, dann muß es einen Knoten p in einer Schleife F_j geben, von dem aus Spieler 1 das Erreichen jedes Knotens aus T_m verhindern kann. Wenn es in einer Schleife F_j einen solchen Knoten gibt, entfernen wir j aus der Indexmenge und wiederholen die Berechnung. Das Verfahren führen wir solange durch, bis I stabil bleibt. Wir nehmen o.B.d.A. an, daß es dies bereits bei $I = \{1, \ldots, k\}$ der Fall ist. (Anderfalls erhalten wir eine einfachere Strategie, die die Strategiegröße von der Größe der Indexmenge abhägt, wir wir noch zeigen werden.)

Auf diese Weise erhalten wir für jedes m eine no-memory-Strategie, die es Spieler 0 ermöglicht, von jedem Knoten aus $Force(T_m) \supseteq \bigcup_{i \in I} F_i$ einen Knoten aus T_m zu erreichen. Diese Strategie sei σ_m.

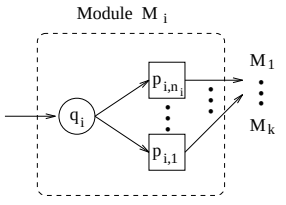
Sei $P_i = (p_{i,1}, \ldots, p_{i,k_i})$ eine beliebige Permutation der Knoten in F_i und $m = (q_1, \ldots, q_k)$ ein Speicherinhalt. Dabei sei $p_{i,j+1}$ der P_i-Nachfolger von $p_{i,j}$ und $p_{i,1}$ der P_i-Nachfolger von p_{i,k_i}. Die Strategie für Spieler 0 sieht dann wie folgt aus: Er benutzt die no-memory-Strategie σ_m solange bis ein Knoten aus $q_i \in T_m$ erreicht wird. Der Speicherinhalt wird aktualisiert zu $m' = (q_1, \ldots, q_{i-1}, q_i', q_{i+1}, \ldots, q_k)$ falls q_i' P_i-Nachfolger von q_i ist. D.h., das momentane Ziel q_i wurde erreicht und dann gegen das neue Ziel q_i' ausgetauscht. Spieler 0 nutzt dann die Strategie $\sigma_{m'}$ bis ein anderes momentanes Ziel erreicht ist.

Offensichtlich handelt es sich hierbei um eine Gewinnstrategie für Spieler 0, da unendlich oft ein Fortschritt in einer der Mengen erzielt wird. Das reicht aus, da $\mathcal{F}$ gegen Obermengebildung abgeschlossen ist und somit egal ist welche Knoten außerdem noch unendlich oft besucht werden.

Die Anzahl der möglichen Speicherinhalte ist dabei $\prod_{i=1}^{k} |F_i| \leq n^k$. Damit ist auch klar, daß eine kleinere Indexmenge zu einem geringeren Speicheraufwand führen würde.

Um zu seigen, daß n^k auch eine unters Schranke darstellt, geben wir eine Familie von Spielen an, deren Schleifenmengen gegen Obermengenbildung abgeschlossen sind.

Wir setzen dazu einen Spielgraphen G_k aus k Modulen zusammen. Das Modul M_i ist dabei definiert durch:
M_i besitzt einen Knoten q_i in Q_0 und mehrere Knoten $p_{i,1}, \ldots, p_{i,n_i}$ in Q_1. Von jedem Knoten $p_{i,j}$ geht eine Kante zu jedem Knoten q_l in anderen Modulen, und die einzige Kante zu $p_{i,j}$ kommt von q_i. Dabei seien P_j die Knoten im Modul M_j. Damit sehen die Module wie in der figur rechts aus.

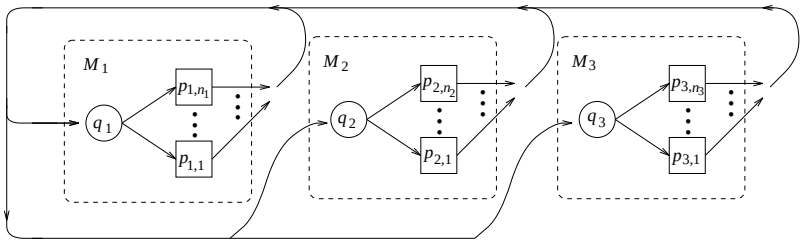


Die Muller-Bedingung ist durch

$$\mathcal{F} = \{F | F = \bigcup_{i \in I} P_i, \quad I \subseteq \{1, \ldots, k\}\}.$$

gegeben.

Beispiel: Für $k = 3$ sieht der Graph dann folgendermaßen aus:



Durch Induktion über die Anzahl der benutzten Module läßt sich zeigen, daß die minimale Speicher, der für eine Gewinnstrategie im Spiel auf G_k benötigt wird eine Größe von $\prod_{i=1}^{k} n_i$ hat, wobei $n_i = |P_i \cap Q_1| = |P_i| - 1$ ist. Dies benutzen wir als Hilfsbehauptung.

Damit können wir zeigen, daß eine untere Schranke der Speichergröße von n^k erreicht werden kann, indem wir die Modulgrößen geschickt wählen: Wir nehmen M_1 mit $n_1 = 2^k$ Knoten in Q_1 und für die anderen Module M_i mit 2^{k-1-i} Knoten in Q_1. Die Spielgraph hat damit $n = 2n_1 + k < 3n_1$ Knoten. Mit der Hilfsbehauptung erhalten wir dann

$$|M| = \prod_{i=1}^{k} n_i = \prod_{i=1}^{k} \frac{n_1}{2^i} = \frac{{n_1}^k}{2^{k+1}} = \frac{1}{2} \cdot {n_1}^{k-1} \in n^{\Omega(k)} \qquad \square$$

Literatur

[BL69] J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.* 138 (1969), 295 – 311.

[GS53] D. Gale and F.M. Stewart. Infinite games with perfect information. In H.W. Kuhn and A.W. Tucker, editors, *Contributions to the Theory of Games*, in Annals of Mathematics Studies 28 Princeton Univ. Press, Princeton, N.J. (1953), 245 – 266.

[Le95] H. Lescow. On polynomial-size programs winning finite state games. In CAV'95. (erscheint demnächst).

[McN93] R. McNaughton. Infinite games played on finite graphs. *Ann. Pure Appl. Logic* 65 (1993), 149 – 184.

Über schwach wachsend kontextsensitive Grammatiken*

Gerhard Buntrock und Gundula Niemann

Theoretische Informatik
Universität Würzburg,
Am Exerzierplatz 3
97072 Würzburg

email: {bunt, niemann}@informatik.uni-wuerzburg.de

Es ist bekannt, daß die Klasse der kontextsensitiven Sprachen (*CSL*) durch Grammatiken charakterisiert wird, die nur nichtverkürzende Regeln enthalten. Wenn wir uns auf Regeln beschränken, die verlängernd sind, erhalten wir die Klasse der wachsend kontextsensitiven Grammatiken (*GCSG*), die die Klasse der wachsend kontextsensitiven Sprachen (*GCSL*) generieren. Es ist ebenfalls bekannt, daß *GCSL* durch quasi wachsend kontextsensitive Grammatiken (*QG-CSG*) charakterisiert werden kann, das sind Grammatiken, die bezüglich einer (Zeichen-)Bewertung wachsen. Solch eine Bewertung ordnet jedem Zeichen einen Wert zu und jedem Wort die Summe der Werte seiner Zeichen.

Hier bewerten wir sowohl die Zeichen als auch die Positionen in einer Regel: Jeder Stelle in einer Regel wird ein bestimmter natürlicher Wert zugeordnet. Die Bewertung einer Regelseite ist die Summe der Produkte der Zeichenbewertung und der Stellenbewertung von jedem Zeichen. Eine kontextsensitive Grammatik heißt schwach wachsend kontextsensitive Grammatik (*WGCSG*) bezüglich einer Stellenbewertung s, falls es eine Zeichenbewertung gibt, so daß für jede Regel die Bewertung der linken Seite niedriger ist als die der rechten. Wir bezeichnen die zugehörige Sprachklasse mit $WGCSG_s$. Die *WGCSG* bezüglich konstanter Stellenbewertungen sind *QGCSG*, charakterisieren also auch *GCSL*.

Die Bewertung von Stellen gibt uns die Möglichkeit, zwei Zeichen zu vertauschen, in der Weise, daß die höher bewerteten Zeichen in die eine Richtung wandern können und die niedriger bewerteten in die entgegengesetzte. Das heißt, für jede nicht konstante Stellenbewertung ist die zugehörige Sprachklasse

*Als Extended Abstract dieser Arbeit ist *On Weak Growing Context-Sensitive Grammars* erschienen in dem Tagungsband des zweiten Symposiums *Latin American Theoretical Informatics (LATIN)*, LNCS 911, pp. 180–194, Springer, April 1995.

nicht in *GCSL* enthalten. Außerdem gibt es eine kontextsensitive Grammatik, die bezüglich keiner Stellenbewertung eine *WGCSG* ist.

Es ist klar, daß der Abschluß von $WGCSG_s$ unter allgemeinen Homomorphismen die Klasse aller rekursiv aufzählbaren Sprachen liefert. Mit Standardargumenten kann das folgende gezeigt werden: Für jede Stellenbewertung s ist die Klasse $WGCSG_s$ abgeschlossen unter ε-freien Homomorphismen, Vereinigung, ε-freier regulärer Substitution, Konkatenation und Schnitt mit regulären Mengen. Was fehlt, zum Beispiel um eine AFL zu erhalten, ist der Abschluß unter inversen Homomorphismen. Diese Eigenschaft kann hier weder gezeigt noch widerlegt werden; ein Ergebnis ist die Äquivalenz dieser Frage zu der, ob alle linear platzbeschränkten Automaten (LBA) durch solche LBA simuliert werden können, die eine zusätzliche exponentielle Zeitschranke besitzen. Der Grund für diese Äquivalenz liegt darin, daß der Abschluß unter inversen Homomorphismen von $WGCSG_s$ für jede Stellenbewertung *CSL* ist, wie wir später sehen werden.

Wir unterscheiden gleichmäßige und ungleichmäßige Stellenbewertungen: eine Stellenbewertung ist gleichmäßig, wenn sie als Anfangsstück einer Exponentialfunktion geschrieben werden kann. Die Basis der zugehörigen Exponentialfunktion nennen wir den Wachstumsfaktor der Stellenbewertung.

Es stellt sich heraus, daß die *WGCSG* bezüglich jeder ungleichmäßigen Stellenbewertung die Klasse *CSL* charakterisieren. Das hängt damit zusammen, daß es keine Fortsetzungen solcher Stellenbewertungen auf Satzformen gibt, die die Werte von Satzformen immer genau dann anwachsen lassen, wenn wir eine schwach wachsende Regel anwenden. Das liegt daran, daß das Verhältnis zwischen den Werten der Positionen unterschiedlich sein kann, je nachdem, ob man eine Regelseite oder ein Teilwort einer Satzform betrachtet. Dieser Effekt kann ausgenutzt werden, um beliebig lange Ableitungen zu erhalten und der Arbeit eines Automaten (d. h. hier wie im folgenden einen LBA) mit einer simulierenden Grammatik zu folgen. Tatsächlich können wir die Möglichkeit, zwei Symbole zu vertauschen, zur Simulation der Kopfbewegungen eines Automaten benutzen, wobei wir in geeigneter Weise ein Zusatzgewicht anhängen oder freigeben.

Durch die Anwendung einiger Regeln, die gerade den oben beschriebenen Effekt ausnutzen, kann dieses Zusatzgewicht verschluckt werden. Derselbe Effekt ist auch dafür verantwortlich, daß noch nicht bekannt ist, ob es für jede *WGCSG* bezüglich einer ungleichmäßigen Stellenbewertung eine *WGCSG* in einer Normalform der Ordnung 2 gibt.

Auf der anderen Seite läßt sich eine gleichmäßige Stellenbewertung unendlich fortsetzen, um beliebig lange Satzformen in einer Ableitung zu bewerten; dabei erhöht sich die Bewertung der Satzform mit jeder Regelanwendung. Das führt zu einer exponentiellen Schranke für die Ableitungslänge, wobei die Basis der beschränkenden Exponentialfunktion der Wachstumsfaktor der Stellenbewertung ist. Es stellt sich heraus, daß eine schwach wachsend kontextsensitive Sprachklasse bezüglich einer gleichmäßigen Stellenbewertung durch deren Wachstumsfaktor charakterisiert wird. Das sieht man leicht ein, wenn man beachtet, daß für *WG-*

CSG bezüglich einer gleichmäßigen Stellenbewertung die klassische Methode zur Überführung in eine Normalform der Ordnung 2 leicht angepaßt werden kann. Diese Resultate lassen sich direkt auch auf monoton fallende Stellenbewertungen übertragen, und die Sprachklassen, die zu einem Wachstumsfaktor und seinem reziproken Wert gehören, fallen zusammen. Wir schreiben für $WGCSL_s$ mit $s = \mathrm{w}^n$ auch $WGCSL_{\mathrm{w}}$.

Andererseits kann ein Automat mit zusätzlicher exponentieller Zeitschranke durch eine *WGCSG* bezüglich einer gleichmäßigen Stellenbewertung simuliert werden, indem man ein Instrument zum Ablegen von Gewichten benutzt (wie oben bereits gesagt wurde, können sie hier nicht verschluckt werden). Dieses Instrument ist eine Zeichenkette, die die Zahl Null zu einer gewissen Basis repräsentiert, und eine Sammlung von längenerhaltenden Regeln, die ein Hochzählen in der gewählten Zahlendarstellung realisieren. Daher nennen wir dieses Instrument einen Zähler. Jedes abzulagernde Gewicht wird als eine Einheit zum Hochzählen behandelt. Für jede Länge der Darstellung der Null erhalten wir eine maximale Anzahl, die auf diese Weise gezählt werden kann. Die Funktion, die jeder Anfangslänge diese Maximalzahl zuordnet, ist die Kapazität des Zählers. Da die größte Basis der Zahlendarstellung, die bei einer Stellenbewertung s verwendet werden kann, vom Wachstumsfaktor $\mathrm{w}(s)$ abhängt, ist die maximale Kapazität eine Exponentialfunktion, deren Basis von $\mathrm{w}(s)$ abhängt. Auf diese Weise erhalten wir einen Zusammenhang zwischen den verschiedenen Wachstumsfaktoren und den verschiedenen exponentiellen Zeitschranken der Automaten. Genauer gesagt bilden die schwach wachsend kontextsensitiven Sprachklassen, die zu verschieden Wachstumsfaktoren gehören, eine lineare inklusionsgeordnete Hierarchie, die die exponentielle Zeithierarchie für *CSL* charakterisiert, das heißt die Klassen der Form T–*NSPACE-TIME*(n, c^n) (die Sprachen, die von linear platzbeschränkten Automaten mit einem Band und einer zusätzlichen Zeitschranke erkannt werden). Desweiteren gilt: Wenn wir *WGCSL* bezüglich einer beliebigen gleichmäßigen Stellenbewertung unter inversen Homomorphismen abschließen, erhalten wir *CSL*. Das kann gezeigt werden, indem man wieder die oben vorgestellten Zähler benutzt.

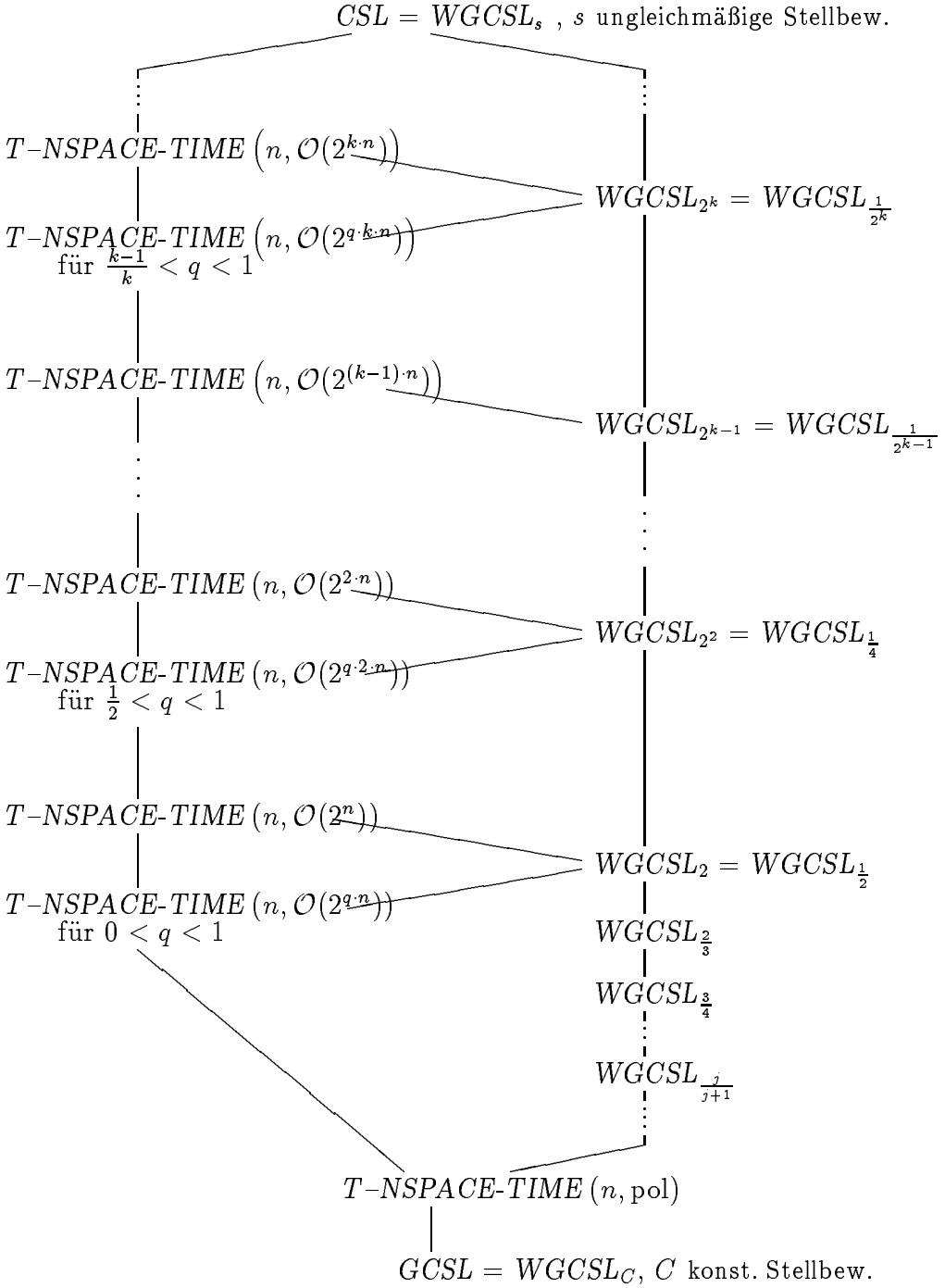
Also erhalten wir die Äquivalenz der folgenden Fragestellungen:

- Kollabiert die exponentielle Zeithierarchie für *CSL*? Das heißt: Können alle linear platzbeschränkten Automaten durch solche mit einer zusätzlichen universellen exponentiellen Zeitschranke simuliert werden?

- Gibt es eine gleichmäßige Stellenbewertung, so daß die entsprechende Klasse schwach wachsend kontextsensitiver Sprachen ganz *CSL* enthält?

- Gibt es eine gleichmäßige Stellenbewertung, so daß die entsprechende Klasse schwach wachsend kontextsensitiver Sprachen abgeschlossen ist unter inversen Homomorphismen?

- Gibt es eine ungleichmäßige Stellenbewertung, für die folgendes gilt: jede schwach wachsend kontextsensitive Grammatik bezüglich dieser Stellenbewertung läßt sich in eine äquivalente in einer Normalform der Ordnung 2 umformen, die bezüglich irgendeiner Stellenbewertung schwach wachsend kontextsensitiv ist?



Solvability of Word Equations Modulo Finite Special Confluent String-Rewriting Systems

Friedrich Otto

Fachbereich Mathematik/Informatik
Universität GH Kassel
Postfach 101380
34109 Kassel

email: otto@theory.informatik.uni-kassel.de

The problem of deciding whether an equation has a solution is one of the most fundamental problems in mathematics. Under the name of "unification" this problem has received much attention in the computer science literature. For various theories algorithms have been developed that not only allow to decide whether a given equation has a solution modulo the theory considered, but that in the affirmative also compute a basis for the set of all solutions, that is, a "complete set of most general unifiers" (See [2] for an overview).

One of the most important results in this area is Makanin's proof that it is decidable whether a word equation $u \equiv v$ has a solution in a free semigroup [6]. Here u and v are strings that contain letters from a given alphabet Σ as well as variables from a set of variables V, and the equation $u \equiv v$ is said to be solvable if there exists a morphism (a "solution") $\phi : V \to \Sigma^*$ such that $\phi(u)$ and $\phi(v)$ are identical strings. This means that "unifiability modulo associativity" is decidable.

Makanin also proved that the solvability of word equations in free groups is decidable [7]. Since the free group in n generators can be seen as a factor monoid of the free monoid in $2n$ generators modulo the congruence generated by a finite special and confluent string-rewriting system, it is only natural to ask whether the solvability of word equations can be generalized to a still larger class of non-free monoids. Here a string-rewriting system R is called *special* if the left-hand side of each rule of R is a non-empty string, while the right-hand side is the empty string λ, and it is called *confluent* if the reduction relation induced by R is confluent.

Book [3] introduces a restricted class of logical formulae that he calls “linear sentences.” The purely existential linear sentences are just disjunctions and/or conjunctions of word equations such that no variable has more than one occurrence in the whole formula. Book proves that it is decidable in polynomial time whether an existential linear sentence is valid with respect to an interpretation that is induced by a finite, monadic, and confluent string-rewriting system. Here a string-rewriting system is called *monadic* if it is length-reducing, and the right-hand side of each rule is of length at most one, that is, it is the empty string or a single letter. In fact, Book considers constrained solutions in that he allows that, for each variable occuring in the sentence under consideration, a regular set may be specified as the domain for that variable.

Recently Oleshchuk has applied the technique of narrowing to word equations [8], thus showing that it is decidable whether a word equation of a certain restricted form has a solution modulo a finite string-rewriting system that is *homogeneous* of degree 2, that is, each rule of the system has a left-hand side of length 2 and the empty string λ as right-hand side. Actually, Oleshchuk works with finite homogeneous systems of degree 2 that are in addition confluent, and then he uses a result of Book [4] which states that, for each finite and homogeneous string-rewriting system of degree 2, there exists a confluent system of the same type that is isomorphic to the original system under a morphism that identifies some of the letters.

To which classes of finite string-rewriting systems can these decidability results be generalized? On the one hand, it is not hard to see that the Post Correspondence Problem can be reduced to the problem of deciding whether a word equation is solvable modulo a finite monadic and confluent string-rewriting system (see, e.g., [5] Section 4.5). On the other hand, Adian has shown that there exists a finite homogeneous string-rewriting system S_3 of degree 3 such that the word problem for S_3 is undecidable [1], and hence, since the word problem of S_3 is obviously a special case of the problem of deciding whether a word equation has a solution modulo S_3, the latter problem is also undecidable.

Here we show that the decidability result does not carry over to the class of all finite and special string-rewriting systems that are confluent. We construct a finite string-rewriting system S that is special and confluent such that it is undecidable in general whether a given word equation has a solution mod S. In addition, this shows that for finite special and confluent string-rewriting systems the validity of non-linear sentences in the sense of Book is undecidable.

References

[1] S.I. Adian (1966), *Defining Relations and Algorithmic Problems for Groups and Semigroups*, Proceedings Steklov Inst. of Math. 85 (Amer. Math. Soc., Providence, RI, 1967).

[2] F. Baader and J. Siekmann (1993), “Unification theory,” in: D.M. Gabbay, C.J. Hogger, and J.A. Robinson (eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming* (Oxford University Press, Oxford, UK).

[3] R.V. Book (1983), “ Decidable sentences of Church-Rosser congruences,” *Theoretical Computer Science* 24, 301–312.

[4] R.V. Book (1984), “ Homogeneous Thue systems and the Church-Rosser property,” *Discrete Mathematics* 48, 137–145.

[5] R.V. Book and F. Otto (1993), *String-Rewriting Systems*, Springer : New York.

[6] G.S. Makanin (1977), “ The problem of solvability of equations in a free semigroup,” *Math. USSR Sbornik* 32, 129–198.

[7] G.S. Makanin (1983), “ Equations in a free group,” *Math. USSR Izvestija* 21, 483–546.

[8] V. Oleshchuk (1994), “ Word equations over Thue systems,” Talk presented at the *Conference on Semigroups, Automata and Languages*, Porto, Portugal, June 20-25,1994.

Die Mächtigkeit von Zwei-Weg-Zählerautomaten auf beschränkten Sprachen

Holger Petersen

Universität Hamburg
Fachbereich Informatik
Vogt-Kölln-Str. 30
22527 Hamburg

email: petersen@informatik.uni-hamburg.de

In diesem Vortrag werden Zwei-Weg-Zählerautomaten als Sprachakzeptoren untersucht. Dabei stammt die Motivation aus dem Interesse für Berechenbarkeitsmodelle unterhalb der Mächtigkeit von Turingmaschinen und dem Vergleich verschiedener Speicherstrukturen, hier Zähler und Keller. Wir erinnern daran, daß ein Zwei-Weg-Zählerautomat (2dc) mit einer (deterministischen) endlichen Kotrolle ausgestattet ist und seine beidseitig mit einer Endmarkierung versehene Eingabe mit Hilfe eines Kopfes liest, der sich in beide Richtungen bewegen kann. Als Speicher besitzt er einen Zähler, der auf Null getestet werden kann. In analoger Weise ist der Zwei-Weg-Kellerautomat (2dpda) definiert.

Bei der Chomsky-Hierarchie kennen wir intuitiv einfache trennende Beispielsprachen. Durch besondere Einfachheit zeichnen sich *beschränkte* Sprachen (engl. bounded languages) aus. Eine Sprache L soll beschränkt heißen, wenn es eine endliche Menge von Wörtern $w_1, w_2, \ldots w_k$ gibt, so daß

$$L \subseteq w_1^* w_2^* \cdots w_k^*$$

Für Zwei-Weg-Automaten scheint die Konstruktion solcher Zeugensprachen schwierig zu sein, eventuell auch unmöglich (s.u.). Ďuriš und Galil zeigen in ihrem Artikel [2], daß 2dpda eine (allerdings nicht beschränkte) Sprache erkennen, die von keinem 2dc akzeptiert wird. In diesem Zusammenhang erwähnen sie auch einige einfache Sprachen, die zur Trennung anderer Automatentypen dienen.

Die Sprache $\{x\$x^R \mid x \in \{0,1\}^*\}$ aus [3] kann auf naheliegende Weise von einem 2dc erkannt werden (x^R bezeichnet das Spiegelwort zu x). Gleiches gilt für die endliche markierte Konkatenation oder den markierten Sternabschluß dieser

Sprache. In [5] wird gezeigt, daß Dyck-Sprachen mit mehreren Sorten Klammern von 2dc erkannt werden. Schließlich läßt sich auch COPY $= \{xx \mid x \in \{0,1\}^*\}$ von einem 2dc akzeptieren. Hierbei wird der Index des zur Zeit zu vergleichenden Zeichens auf dem Zähler gehalten und mit Hilfe des Kopfes die halbe Eingabelänge abwechselnd addiert und subtrahiert. Die Laufzeit dieser Lösung liegt in $O(|w|^2)$, wobei w die Eingabe ist.

Es ist bemerkenswert, daß sich bei *Mehrzählerautomaten* mit der Technik aus [3] eine Verbesserung auf $O\left(\frac{|w|^2}{\log|w|}\right)$ erzielen läßt. Eine genauere Analyse zeigt, daß drei Zähler hierfür ausreichen, und sogar zwei, falls ein Zähler mit der Eingabelänge verglichen werden kann. Andererseits kann mit Hilfe von Kreuzungsfolgen und Abzählargumenten [1] oder Kolmogorov-Komplexität bewiesen werden, daß eine Laufzeit dieser Größenordnung optimal ist. Offen bleibt die Frage, ob für Ein- oder Zweizählerautomaten (ohne zusätzliche Operationen) die zuerst angegebene Lösung optimal ist.

In [2] wird gezeigt, daß die Sprache

$$L = \{x_0\#x_1\#\ldots\#x_k \mid x_j \in \{0,1\}^*, 0 \leq j \leq k, x_i = x_0 \text{ für ein } 1 \leq i \leq k\}$$

von keinem 2dc erkannt wird. Das gleiche gilt für die Sprache, bei der x_0 gespiegelt auftritt. Diese Resultate zeigen, daß für Zwei-Weg-Automaten das Speichermedium Keller mächtiger als ein Zähler ist und daß die Klasse der von 2dc erkannten Sprachen unvergleichbar mit der Klasse der kontextfreien Sprachen ist (betrachte z.B. COPY).

Es bleibt in [2] offen, ob

$$\tilde{L} = \{0^n 1^{n^2} \mid n \geq 1\}$$

von einem 2dc erkannt wird. Von dieser Sprache wird in [4] gezeigt, daß sie kein Mehrzählerautomat mit fester Umkehrschranke auf den Zählern akzeptieren kann.

Wir beschreiben im Vortrag die Arbeitsweise eines 2dc, der $\tilde{L}$ erkennt. Es ist leicht, zu prüfen, ob die Eingabe die Form $0^n 1^{k \cdot n}$ hat. Ist k nicht zu groß, dann kann eine Division durch wiederholtes Abziehen und Zählen in der niederwertigen n-ären Stelle erfolgen. Die Hauptschwierigkeit ist daher, den Wert von k zu begrenzen. Dazu wird die größte Zweierpotenz kleiner oder gleich $k \cdot n$ berechnet, dazu n addiert, und durch wiederholte Division durch 2 eine Abschätzung der Quadratwurzel aus $k \cdot n$ mit n verglichen. Eine weitere beschränkte Sprache, die 2dc erkennen können, ist

$$\hat{L} = \{0^n 1^{2^n} \mid n \geq 1\}$$

Hier wird zunächst getestet, ob die Eingabe die Form $0^n 1^{2^k}$ hat. Dann wird 2^k wiederholt durch 2 dividiert bis sich eine ungerade Zahl ergibt. Die Anzahl der Wiederholungen kann in den höherwertigen Binärstellen gespeichert werden. Eine detailierte Darstellung der Algorithmen (in etwas verallgemeinerter Form) findet sich in [9].

Als Folgerungen erhalten wir die bekannten Resultate, daß 2dc mit fester Umkehrschranke auch auf beschränkten Sprachen schwächer als allgemeine 2dc sind und daß das Leerheitsproblem für 2dc auf beschränkten Sprachen unentscheidbar ist (durch Reduktion des 10ten Hilbert-Problems, ein direkterer Beweis benutzt Zwei-Zähler-Automaten [7]).

Abschließend erwähnen wir, daß 2dc mit fester Umkehrschranke auf beschränkten Sprachen äquivalent zu umkehrbeschränkten 2dpda sind [6], während die Beziehung von allgemeinen 2dc und 2dpda auf beschränkten Sprachen weiterhin offen bleibt. Eine Sprache, deren Untersuchung sich hier anbietet, ist $\{0^{n^2} \mid n \geq 1\}$, die gemäß [8] von einem 2dpda erkannt wird.

Literatur

[1] A.Cobham: *The recognition problem for the set of perfect squares,* Conf. Record 7th IEEE annual Symp. on Switching and Automata Theory (1966) 78–87.

[2] P.Ďuriš, Z.Galil: *Fooling a two way automaton or one pushdown store is better than one counter for two way machines,* TCS 21 (1982) 39–53.

[3] P.C.Fischer, A.R.Meyer, A.L.Rosenberg: *Counter machines and counter languages,* Math. Systems Theory 2 (1968) 265–283.

[4] E.M.Gurari, O.Ibarra: *Two-way counter machines and diophantine equations,* JACM 29 (1982) 863–873.

[5] G.Hotz, J.Messerschmidt: *Dyck-Sprachen sind in Bandkomplexität log n analysierbar,* Techn. Bericht, Fachbereich Angewandte Mathematik und Informatik, Universität des Saarlandes Saarbrücken (1974).

[6] O.H.Ibarra, T.Jiang, N.Tran, H.Wang: *On the equivalence of two-way pushdown automata and counter machines over bounded languages,* Proc. STACS 93, LNCS 665, 354–364.

[7] M.L.Minsky: *Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines,* Annals of Mathematics, 74, 3 (1961), 437–455.

[8] B.Monien: *Deterministic two-way one-head pushdown automata are very powerful,* IPL 18 (1984) 239–242.

[9] H.Petersen: *Two-way one-counter automata accepting bounded languages,* SIGACT News 25(3), Sept. 1994, 102–105.

Vollständige Sprachen für Zählerautomaten

Klaus Reinhardt*

Institut für Informatik der Universität Stuttgart
Breitwiesenstraße 20–22
70565 Stuttgart

email: reinhardt@informatik.uni-tuebingen.de

Sprachklassen, die von nichtdeterministischen Automaten einer bestimmten Art, welche λ-übergänge haben dürfen und keiner Platz- oder Zeitbeschränkung unterliegen, erkannt werden, sind abgeschlossen unter rationaler Transduktion. Dies ist gleichbedeutend mit dem Abschluß unter Homomorphismen, inversen Homomorphismen und dem Schnitt mit regulären Sprachen [Ber79]. Der Speicher des Automaten kann durch eine Sprache beschrieben werden, welche für die Klasse vollständig bezüglich rationaler Transduktion ist.

Bekannte Beispiele hierfür sind die kontextfreien Sprachen, für welche die Dyck-Sprache vollständig ist, wobei öffnende bzw. schiesende Klammern das Ein- bzw. Auskellern eines entsprechenden Sybols beschreiben oder die restringierten Einzählersprachen (ROCL) für welche die Semi-Dyck-Sprache $D_1' = \{w \in \{a,b\}^* \mid |w|_a = |w|_b \wedge \forall uv = w\ |u|_a \geq |u|_b\}$ vollständig ist, wobei a dem Inkrement und b dem Dekrement des Zählers entspricht.

Besitzt ein Automat mehrere Zähler, so erhält man eine vollständige Sprache durch das entsprechende disjunkte Shuffle von vollständigen Sprachen für einzelne Zähler [Gre78]. Bekannt ist, daß ein Multicounterautomat (aus vielen schwachen Zählern) nur entscheidbare Sprachen erkennen kann (dies entspricht dem Erreichbarkeitsproblem in Petrinetzen bzw. in Vektoradditionssystemen); mit zwei Zählern mit Nulltest werden jedoch schon die rekursiv aufzählbaren Sprachen charakterisiert.

Entscheidbar wiederum sind die Sprachen, die ein *Prioritätsmulticounterautomat* erkennen kann, welcher nach folgendem Prizip arbeitet [Rei94b]: Außer dem Nulltest von Zähler 1 kann auch, falls Zähler 1 Null ist, Zähler 2 auf Leerheit getestet werden oder allgemein, falls die Zähler 1 bis $k-1$ den Wert Null haben,

*Neue Adresse: Wilhelm-Schickhard-Institut für Informatik, Eberhard-Karls-Universität Tübingen, Sand 13, 72076 Tübingen.

kann Zähler k auf Leerheit getestet werden. Dieser wird beschrieben durch ein 6-Tupel

$$A = (k, Z, \Sigma, \delta, z_0, E)$$

mit Zustandsmenge Z, Eingabealphabet Σ, Übergangsrelation

$$\delta \subseteq (Z \times (\Sigma \cup \{\lambda\}) \times \{0 \dots k\}) \times (Z \times \{-1, 0, 1\}^k),$$

Anfangszustand z_0, akzeptierende Zustände $E \subseteq Z$, Konfigurationenmenge $C_A = Z \times \Sigma^* \times I\!N^k$, Anfangskonfiguration $\sigma_A(x) = \langle z_0, x, 0^k \rangle$ und Konfigurationsübergangsrelation

$$\langle z, ax, n_1, ..., n_k \rangle \underset{A}{\vdash} \langle z', x, n_1 + i_1, ..., n_k + i_k \rangle$$

dismath gdw. $z, z' \in Z, a \in \Sigma \cup \{\lambda\}, \langle (z, a, j), (z', i_1, ...i_k) \rangle \in \delta, \ \forall i \leq j \ n_i = 0$ (und $n_{j+1} > 0$ oder $j = k$)[1].

Sei $PD_0 := \{\lambda\}$, $\Sigma_k := \{a_1, b_1, c_1, ..., a_k, b_k, c_k\}$ und $PD_k :=$

$$(\{w \in (\Sigma_k \setminus \{c_k\})^* \mid \Pi_{\Sigma_{k-1}}(w) \in PD_{k-1} \ \wedge \\ |w|_{a_k} = |w|_{b_k} \ \wedge \ \forall uv = w \ |u|_{a_k} \geq |u|_{b_k} \} c_k)^*.$$

Die Sprache PD_k ist vollständig bezüglich rationaler Transduktion in der Klasse k-PMC der von einem Prioritätsmulticounterautomaten mit k Zählern erkannten Sprachen. Hierbei entspricht a_i dem Inkrementieren des Zählers i, b_i dem Dekrementieren des Zählers i und c_i dem Nulltest des Zählers i und allen Zählern $l \leq i$, da aus $wc_iv \in PD_k$ folgt daß $\forall l \leq i \ |w|_{a_l} = |w|_{b_l}$ gilt.

Ein Überblick über entscheidbare Sprachklassen, deren Automaten und die dazugehörigen, unter rationaler Transduktion vollständigen Sprachen gibt Inklusionsdiagramm 1.

Sowohl die Hierarchie k-PMC als auch die Hierarchien mit blinden und schwachen Zählern sind echt.

Ebenfalls von Interesse ist ein Zähler (auf den ganzen Zahlen vergleiche BLIND in [Gre78]), bei dem außer Inkrement und Dekrement anstelle eines Nulltest einen Natürlichkeitstest (d.h. Zahlerinhalt ≥ 0) als Instruktion möglich ist. Mit 4 solchen Zählern werden wieder die rekursiv aufzählbaren Sprachen charakterisiert.

Einen *Prioritätsnatürlichkeitstestautomaten* kann man analog zum Prioritätsmulticounterautomaten definieren; die einzigen Unterschiede sind dabei die neue Konfigurationenmenge $C_A = Z \times \Sigma^* \times \mathbb{Z}^k$ und die Konfigurationsübergangsrelation $\langle z, ax, n_1, ..., n_k \rangle \underset{A}{\vdash} \langle z', x, n_1 + i_1, ..., n_k + i_k \rangle$, welche jetzt die Bedinung $z, z' \in Z, a \in \Sigma, \langle (z, a, j), (z', i_1, ...i_k) \rangle \in \delta, \ \forall i \leq j \ n_i \geq 0$ hat.

Ein solcher Automat kann einen Prioritätsmulticounterautomaten mit der doppelten Anzahl von Zählern simulieren. Dabei wird ein Zähler durch zwei

[1] Automaten gemäß der Definition mit bzw. ohne diesen Zusatz lassen sich leicht ineinander überführen

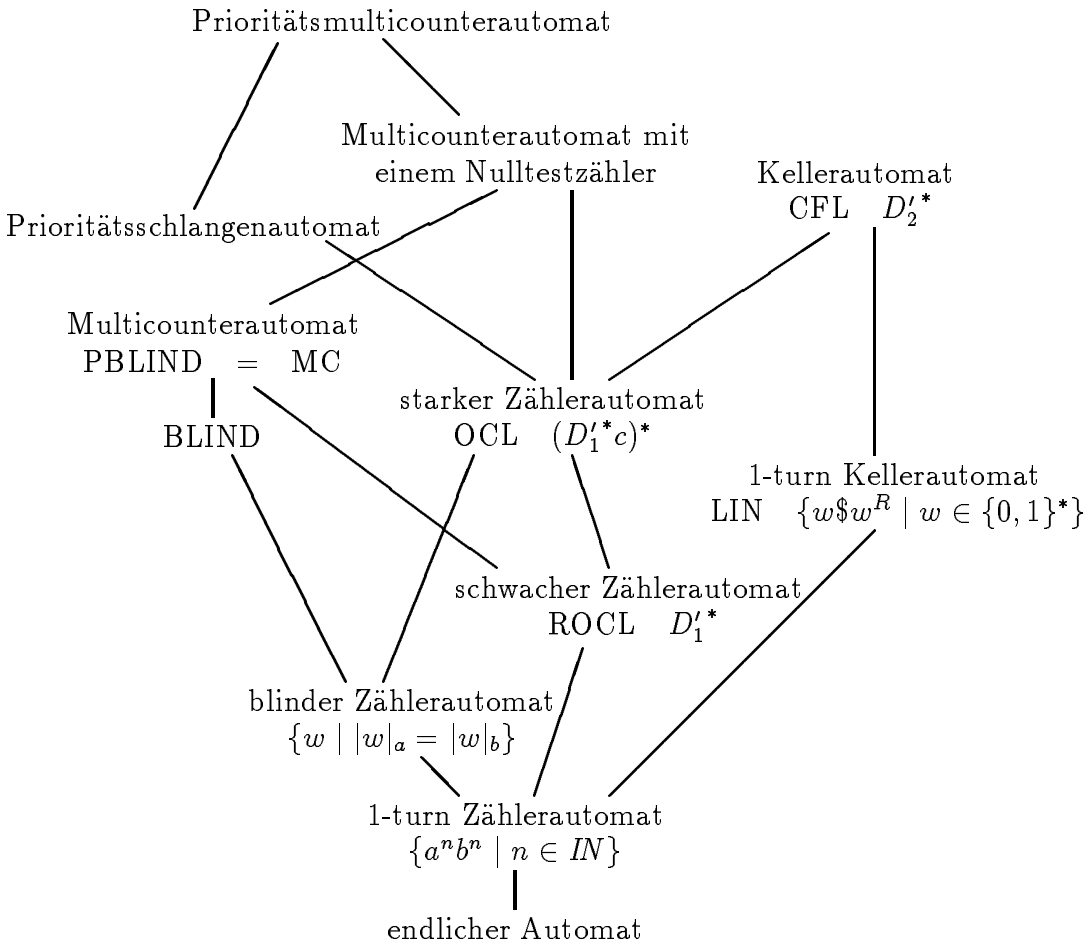


Abbildung 1: Inklusionsdiagramm.

Zähler mit umgekehrtem Vorzeichen (Inkrementieren und Dekrementieren vertauscht) simuliert. Einer der Zähler wird bei jedem Schritt auf ≥ 0 geprüft wird und der Nulltest wird durch einen Natürlichkeitstest des 'Gegenzählers' simuliert.

Umgekehrt kann ein solcher Automat auch von einem Prioritätsmulticounterautomaten mit der doppelten Anzahl von Zählern simuliert werden. Der Inhalt eines Zählers wird als Differenz zweier Zähler dargestellt und für jede Veränderung des simulierten Zählers wird nichtdeterministisch entweder der positive Teil oder der negative Teil in umgekehrter Weise verändert. Der Natürlichkeitstest wird durch einen Nulltest des negativen Teils simuliert.

Im Gegensatz zu Prioritätsmulticounterautomaten läßt sich vermuten, daß zugelassen werden kann, daß die ersten zwei oder vielleicht auch drei (aber keinesfalls vier) Zähler unabhängig von ihrer Priorität auf Natürlichkeit getestet werden können, ohne daß das Leerheitsproblem dadurch unentscheidbar wird.

Ein solcher Automat kann Sprachen erkennen, die nicht in PMC liegen.

Literatur

[Ber79] J. Berstel. *Transductions and context-free languages.* Teubner Studienbücher, Stuttgart, 1979.

[Gre78] S. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoret. Comput. Sci.*, 7:311–324, 1978.

[Rei94b] K. Reinhardt. *Prioritätszählerautomaten und die Synchronisation von Halbspursprachen.* PhD thesis, Institut für Informatik, Universität Stuttgart, 1994.

Some New Decision Results for Edge Grammars

Ralf Stiebe

Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Postfach 4210
39116 Magdeburg

email: stiebe@cs.uni-magdeburg.de

1 Introduction

Graph generating devices were intensively studied in recent years. We shall deal here with edge grammars as introduced by F. Berman [1]. These grammars generate pairs of words. All pairs whose components are words of the same length n are seen as edges of a graph G_n while the components define the nodes of this graph.
Formally, an *edge grammar* is a quintuple $\Gamma = (N, W, T, P, S)$ where N and W are alphabets, T is a subset of $(W \cup \{\lambda\})^2$, λ denoting the empty word, and $\Gamma' = (N, T, P, S)$ is a (string) grammar with the sets of nonterminals N, of terminals T, of rules P, and the start symbol $S \in N$.
Γ is called context-sensitive, context-free, linear, regular, respectively, iff Γ' is a context-sensitive, context-free, linear, regular, respectively, grammar. If moreover, T is a subset of W^2 then the edge grammar is said to be even.
A word over T can in a natural way be seen as a pair of words over W. The word $z \in T^*$ corresponds to the pair $(p_1(z), p_2(z))$ where p_1, p_2 are the homomorphisms (projections)

$$p_i : T^* \mapsto W^*,\ p_i(a_1, a_2) = a_i, a_i \in W \cup \{\lambda\}, i = 1, 2.$$

For a given integer n, we define

$$\begin{aligned} E_n(\Gamma) &= \{(v, w) : v, w \in W^+, |v| = |w| = n, v = p_1(z), w = p_2(z), z \in L(\Gamma')\} \\ V_n(\Gamma) &= \{v \in W^+ : |v| = n, (v, w) \in E_n(\Gamma) \text{ or } (w, v) \in E_n(\Gamma)\} \\ G_n(\Gamma) &= (V_n(\Gamma), E_n(\Gamma)) \end{aligned}$$

$G_n(\Gamma)$ is a directed graph with the set of nodes $V_n(\Gamma)$ and the set of edges $E_n(\Gamma)$. Since we can obtain from any directed graph a unique indirected one edge grammars can be seen as generators of indirected graphs as well.
The set $\mathbf{G}(\Gamma) = \{G_n(\Gamma) : G_n(\Gamma) \text{ non-empty }, n \geq 1\}$ is called the *graph language* generated by Γ.
Some important graph families, such as the set of all complete binary trees, or the set of all complete graphs, can be generated by even regular edge grammars, see [2]. Note that these graph families cannot be described by context-free hyperedge replacement or node label controled graph grammars.
On the other hand, it can be shown by consideration of the growth function that several families of graphs, e.g. the set of all trees, are not generatable by even regular edge grammars.

2 Results

The decision problems to be regarded are questions similar to problems known from classical formal language theory, such as membership, equivalence, or emptiness. Additionally, we consider the following questions for an edge grammar Γ and a graph-theoretical property P.

Q1: Does $\mathbf{G}(\Gamma)$ contain a graph with property P?

Q2: Do all graphs of $\mathbf{G}(\Gamma)$ have property P?

Some results concerning classical problems were given in [2]. Berman and Shannon showed, moreover, undecidability for the questions **Q1** and **Q2** and several properties in the context-sensitive case; for the question **Q2**, these results were improved by Dassow [3] who proved undecidability for linear edge grammars and special subgraph properties.

As regards classical problems we could obtain the following new results.

Theorem 1 *The emptiness problem is decidable for context-free edge grammars.*

Theorem 2 *For an even regular edge grammar Γ and a graph G, it is decidable whether $\mathbf{G}(\Gamma)$ contains a graph*

a) with the subgraph G,

b) with the induced subgraph G,

c) which is isomorphic to G.

Theorem 3 *The subgraph problem is in general undecidable for regular and even-linear edge grammars.*

Theorem 4 *There are even-regular edge grammars Γ_1, Γ_2 such that the following questions are undecidable for regular, even-linear edge grammars Γ.*

a) Is $\mathbf{G}(\Gamma)$ equal to $\mathbf{G}(\Gamma_1)$?

b) Is $\mathbf{G}(\Gamma)$ disjoint with $\mathbf{G}(\Gamma_2)$?

Now we shall note the new results concerning graph-theoretical properties.

Theorem 5 *The question whether some graph generated by an edge grammar contains a loop is*

a) decidable for even-context-free egde grammars,

b) undecidable for regular edge grammars.

Theorem 6 *The questions whether the maximal degrees of the graphs generated by an edge grammar are bounded, bounded by a given number k, respectively, are*

a) decidable for even-regular egde grammars,

b) undecidable for even-linear edge grammars.

Theorem 7 *The questions* **Q1** *and* **Q2** *are undecidable for even-regular edge grammars and the properties given below.*

a) connectedness

b) bipartiteness

c) acyclicity

d) planarity

e) edge/node k-colorability, $k \geq 2$

f) Eulerianity

g) Hamiltonicity

3 Conclusions

We could improve many of the decision results obtained earlier. The most interesting open classical problems are the intersection emptiness and equivalence problems for the even-regular case and the membership problem for the classes between even-regular and context-sensitive edge grammars.
For a number of graph theoretical properties we have shown undecidability of the questions **Q1** and **Q2** for the family of even-regular edge grammars. Decidability could be obtained for some “local” properties. An interesting question is whether these results can be generalized to a larger class of properties, e.g. properties that can be expressed by means of first order logic.

References

[1] F. Berman, Edge grammars and parallel computation, *Proc. Allerton Conf. on Communication, Control, and Computing,* 214-223, 1983.

[2] F. Berman and G. Shannon, Representing graph families with edge grammars, *Information Science* **70** (1993), 241-269.

[3] J. Dassow, Decision problems for edge grammars, *Proc. MFCS*, 1994.

Decomposing Large Petri Nets into Synchronized Blocks

Andreas Stübinger

Universität Passau
Lehrstuhl für Informatik
Innstraße 33
94030 Passau

email: stuebing@fmi.uni-passau.de

We introduce synchronized blocks $\mathcal{SB}$ as a new scheme to analyse large Petri nets. Our scheme consists of blocks and a method to synchronize them. Blocks are Petri nets of a simple structure defined by a graph grammar. The graph grammar incorporates important concepts of high level programming languages. The synchronization of blocks is done by merging equally labelled nodes. Such glueings of graphs are well known concepts in graph grammar theory.

Our approach is motivated by automatic graph drawing being easy for blocks. There is a need for tools which help drawing Petri nets in a nice way. For blocks the underlying graph grammar is a suited tool, since the graph grammar is chosen in a way to support simple drawings of blocks.

Our goal is a characterization of the class of Petri nets defined by synchronized blocks.

Definition 1
Let $\mathcal{G} = (N_\mathcal{G}, P_\mathcal{G}, S_\mathcal{G})$ be a graph grammar as follows:

- $N_\mathcal{G}$ = (P,T,S) is a set of nonterminal nodes. A label P represents a place, T a transition and S the marked place.
- $S_\mathcal{G} = \{S\}$ is the initial start symbol.
- $P_\mathcal{G}$ is the set of productions as shown in figure 1.

A *block* $\mathcal{B} = (P, T, F, M_0)$ is a directed bipartite graph $g \in S(\mathcal{G})$.

In order to model the behaviour of two or more concurrent working systems, we provide some sort of synchronization. We assume that all nodes (places and transitions) of one block have distinct labels. If there would be two or more

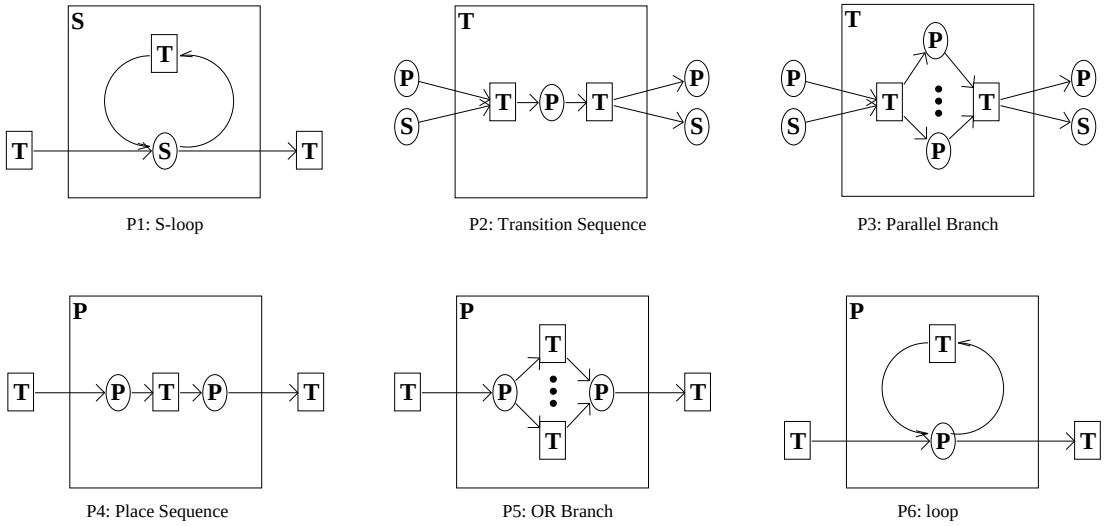


Figure 1: The set $P_{\mathcal{G}}$ of productions for $\mathcal{G}$. The dots in P3 and P5 have the meaning of "create as many branches as needed".

equally labelled nodes in one block this action (transition) or state (place) would be represented by one node only, as these equally labelled nodes represent the same action or state. Our synchronization operation uses the approach of merging transitions, i.e. if some actions of different blocks are equally labelled, they represent a *common action*, so these transitions will be merged in the synchronization operation. Places, representing *internal states* of a block, are kept disjoint.

Definition 2
Let $\mathcal{B}_i = (P_i, T_i, F_i, M_{0,i})$ be blocks from $S(\mathcal{G})$. Define the *Synchronized block net* $\mathcal{N} = \diamond\mathcal{B}_i$ with $\mathcal{N} = (P, T, F, M_0)$ as follows:

$$P = \dot{\cup} P_i \qquad T = \cup T_i$$
$$F = \dot{\cup} F_i \qquad M_0 = \dot{\cup} M_{0,i}$$

Places are unified disjointly; common labelled transitions representing common actions in different blocks are merged into one transition. The definition gets effective if the blocks $\mathcal{B}_i$ are not pairwise disjoint.

Complexity of Synchronized Blocks

We have to deal with the complexity of deciding whether a given input Petri net belongs to the class of synchronized blocks. For a Petri net $\mathcal{N} = (P, T, F, M_0)$

the problem is to find $k = |M_0|$ blocks, i.e. as many blocks as places are marked by the initial marking M_0.

Problem Decomposition of $\mathcal{SB}$

INSTANCE Petri net $\mathcal{N} = (P, T, F, M_0)$, the block defining graph grammar $\mathcal{G}$ of definition 1.

QUESTION Is there a decomposition of $\mathcal{N}$ into $k = |M_0|$ blocks $\mathcal{B}_i = (P_i, T_i, F_i, M_{0,i})$, such that $\mathcal{N} = \diamond\mathcal{B}_i$?

Lemma
The problem “Decomposition of $\mathcal{SB}$” belongs to NP.

Proof:
Choose $\mathcal{B}_i$, $i \in \{1, \ldots, |M_0|\}$
Verify whether $\mathcal{B}_i \in S(\mathcal{G})$, $i \in \{1, \ldots, k\}$ in $O(n)$.
Verify whether $\diamond\mathcal{B}_i = \mathcal{N}$, $i \in \{1, \ldots, k\}$ in $O(n)$. □

We are interested in an efficient layout algorithm based on the class of synchronized block nets. Therefore we are looking for a polynomial time algorithm to decompose a given Petri net into its constituting blocks.

In our talk we will show some problems of such an algorithm by using the example input net of figure 2.

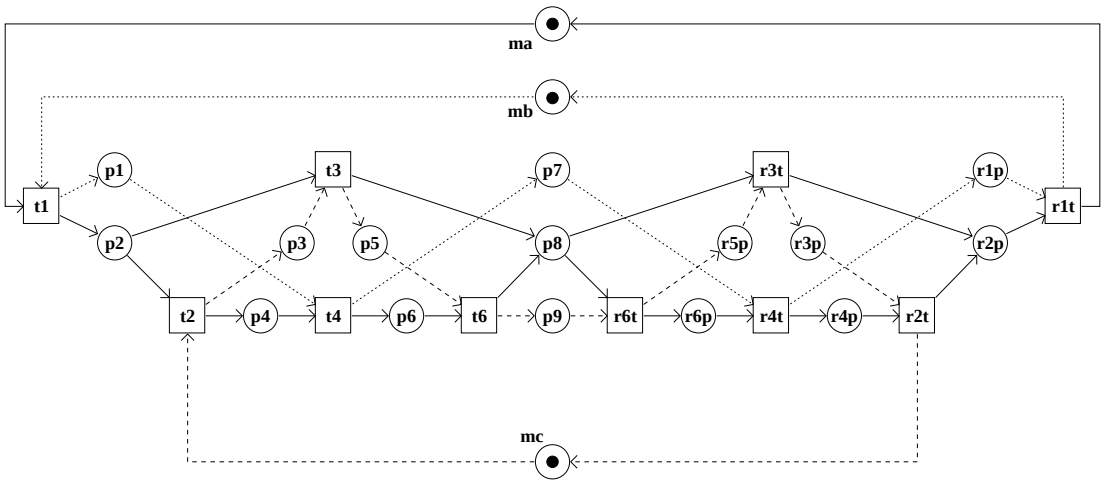


Figure 2: Example input Petri net. The different line styles mark the different constituing blocks.

Fairness and Partial Order Semantics

Walter Vogler*

Institut für Mathematik
Universität Augsburg
86135 Augsburg

email: vogler@informatik.uni-augsburg.de

Abstract

Causality-based partial order semantics allows an easy formulation of weak fairness. It is demonstrated that this is true also for another partial order semantics, namely for partial words.

1 Introduction

The interleaving approach describes a run of a concurrent system as a sequence of actions. In contrast to this, we can also model a run as a partial order of actions; this way we additionally give information on the independence of actions. Most common are partial orders that describe the causality in system runs, e.g processes in the model of safe Petri nets; but there are also others like partial words [Gra81]. Partial order semantics is intuitively appealing, but it has to be asked whether the effort to handle partial orders instead of sequences is really worthwhile; and when comparing the respective merits of partial order and interleaving semantics, one should also take into account partial orders that are not so much causality-based.

Often, system runs are only of interest if they satisfy (weak) fairness, which requires that no component of a concurrent system unnecessarily stops, i.e. that an action is eventually taken if all needed resources are continuously available. Processes allow a particularly simple characterization of fairness: let π be a process and w one of its linearizations; then π is maximal (w.r.t. the prefix-relation) if and only if w is fair [Rei84]. The purpose of this note is to show that this result depends on the prefix-relation for partial orders and not on the concept of causality; it holds just as well for partial words. The full version of this note also considers so-called interval semiwords.

*This work was partially supported by the DFG and the ESPRIT Basic Research Working Group 6067 CALIBAN (CAusal calcuLI BAsed on Nets).

2 Petri nets and partial words

A *(Petri) net* $N = (S, T, W, M_N)$ consists of finite disjoint sets S of *places* and T of *transitions*, the *weight function* $W : S \times T \cup T \times S \to \{0, 1\}$, and the *initial marking* $M_N : S \to I\!N_0$. We assume that the reader knows some Petri net notions like firing sequence, step, reachable marking, presets ${}^\bullet t$ etc. A net is *safe* if for all places s and reachable markings M we have $M(s) \leq 1$.

General assumption We assume that, for the rest of this abstract, some fixed net N is given which is safe and has no transitions with empty preset.

We are interested in modelling system runs by partial orders. Hence, for us a (finitely preceded, labelled) *partial order* $p = (E, <, l)$ consists of a set E of *events*, a labelling $l : E \to T$, and an irreflexive, transitive relation $<$ on E such that for each $e' \in E$ there are only finitely many e with $e < e'$. If $e < e'$, we call e a *predecessor* of e' and e' a *successor* of e. If $e < e'$ or $e = e'$, we write $e \leq e'$. Two events e and e' are *concurrent*, $e \; co \; e'$, if neither $e < e'$ nor $e' < e$.

$E' \subseteq E$ is *left-closed*, if for all $e' < e \in E'$ we have $e' \in E'$; in this case, $(E', <', l')$ is a *prefix* of $(E, <, l)$, where $<'$ and l' are the appropriate restrictions of $<$ and l. $(E, <, l)$ is an *augmentation* of $(E, <', l)$, if $<'$ is a subset of $<$. If $<$ is *total*, i.e. for all events e and e' we have $e < e'$, $e = e'$ or $e' < e$, then $(E, <, l)$ is a *linearization* of $(E, <', l)$. In this case, we can view $(E, <, l)$ as a sequence of transitions; this is also true for infinite E, since all partial orders are finitely preceded by definition.

Most often a partial order semantics is given by so-called processes; in this approach, the partial order models causality. Another view is that concurrency is more than arbitrary interleaving but includes it. This idea is formalized in the partial words of [Gra81]: in such a partial order, any set of pairwise concurrent elements represents a step that can be fired provided the precedences prescribed by the partial order are observed.

- We call a partial order $(E, <, l)$ a *partial word* of N if for all finite disjoint subsets B and C of E we have: if all elements of C are pairwise concurrent and B and $B \cup C$ are left-closed, then the transitions in $l(C)$ are *concurrently enabled* under $M_N + \sum_{e \in B} W(l(e), .) - W(., l(e))$, i.e.

$$\sum_{e \in C} W(., l(e)) \leq M_N + \sum_{e \in B} W(l(e), .) - W(., l(e)).$$

It is not hard to see (by induction on $|B|$) that the marking mentioned is a reachable marking, so by the general assumption for N equally labelled events cannot be concurrent. Thus, a set C as above and, hence, any set of pairwise concurrent events in a partial word has at most $|T|$ elements. Therefore, a partial word is at most countable and it has a linearization. (The latter is not true for uncountable partial orders, since linearizations have to be finitely preceded.)

It is not difficult to see that the set of partial words is closed under augmentation; i.e., if p is an augmentation of a partial word q, then p is a partial word, too; see [Vog92] for the finite case. This shows that concurrency in partial words is just seen as a possibility; possibly concurrent transitions can also be performed sequentially.

The above is an extension of the original definition to infinite partial orders; it is sensible as the following result shows (which also holds for unsafe or infinite N).

Proposition 2.1 *A partial order is a partial word of N if and only if all finite prefixes are partial words of N.*

3 Fairness and partial words

A system run is (weakly) fair, if each action occurs eventually provided the necessary resources are continuously available; weak fairness is also called justice or finite-delay property. The standard definition of a fair firing sequence is as follows:

Definition 3.1 A firing sequence $t_1 t_2 \ldots$ is *fair** if either it is finite and no transition is enabled under the marking reached or it is infinite and for $M_N[t_1\rangle M_1[t_2\rangle M_2 \ldots$ we have: if for some $t \in T$ and $i \in IN$ we have $M_j[t\rangle$ for all $j \geq i$, then $t = t_j$ for some $j > i$. □

This definition is not fully adequate for concurrent systems: a resource is continuously available, if it is available in all states between activities *and during* all activities. Hence, we use the following refined version of fairness.

Definition 3.2 A firing sequence $t_1 t_2 \ldots$ is *fair* if either it is finite and no transition is enabled under the marking reached or it is infinite and for $M_N[t_1\rangle M_1[t_2\rangle M_2 \ldots$ we have: if for some $t \in T$ and $i \in IN$ we have that t and t_{j+1} are concurrently enabled under M_j for all $j \geq i$, then some t_j with $j > i$ equals t. □

Definitions 3.1 and 3.2 really differ. Take a net with a single, marked place and two transitions t and t', where t is on a loop with the place, while t' is just in its postset. Intuitively, an infinite sequence of t's should be fair, since the only resource is repeatedly in use; it is in fact fair, but not fair*.

With Definition 3.2, maximality of a process corresponds to fairness of an arbitrary linearization [Rei84]. Thus, fairness for processes can be defined much more easily than for firing sequences. Our aim is to show analogous results for partial words and interval semiwords. We start with a result which demonstrates the close relation between fairness and the prefix-relation on partial orders.

Theorem 3.3 *A firing sequence w is fair if and only if – regarded as a partial word – it is maximal w.r.t. the prefix-relation.*

Proof: For finite w this is obvious. If an infinite w is not maximal, we can extend it with one event e. The predecessors of e form a prefix v of w, and by definition of a partial word $l(e)$ is concurrently enabled with each transition after v under the appropriate marking; hence, w is not fair.

If w is not fair due to some $t \in T$ and $i \in IN$, we add a t-labelled event e succeeding the first i events of w. We have to check the requirement for all appropriate B and C. If $e \in B$, it is sufficient to check $B - \{e\}$ and $C \cup \{e\}$ since e is a maximal event. Hence, in the only interesting case C consists of e and a concurrent event, and $l(C)$ is enabled by the assumption on t and i. □

The theorem below, our main result, is immediately implied by 3.3 and the following lemma, which we state without proof.

Lemma 3.4 *Let p be an augmentation of a partial word q. Then q is a maximal partial word if and only if p is.*

Theorem 3.5 *Let w be a linearization of a partial word p. Then p is maximal w.r.t. the prefix-relation if and only if w is fair.*

This result can also be used for an independent proof of the result in [Rei84], since processes can be identified with partial words which are not proper augmentations of others [Kie88].

References

[Gra81] J. Grabowski. On partial languages. *Fundamenta Informaticae*, IV.2:428–498, 1981.

[Kie88] A. Kiehn. On the interrelationship between synchronized and non-synchronized behaviour of Petri nets. *J. Inf. Process. Cybern. EIK*, 24:3–18, 1988.

[Rei84] W. Reisig. Partial order semantics versus interleaving semantics for CSP-like languages and its impact on fairness. In J. Paredaens, editor, *Automata, Languages and Programming*, Lect. Notes Comp. Sci. 172, 403–413. Springer, 1984.

[Vog92] W. Vogler. *Modular Construction and Partial Order Semantics of Petri Nets.* Lect. Notes Comp. Sci. 625. Springer, 1992.

Regulär kontrollierte k-limitierte T0L-Systeme

Dietmar Wätjen

Institut für Theoretische Informatik
Technische Universität Braunschweig
Postfach 3329
38023 Braunschweig

email: waetjen@iti.cs.tu-bs.de

Ein k-limitiertes T0L-System (Σ, H, ω, k) (kurz klT0L-System genannt, eingeführt in [3]) besteht aus einem T0L-System (Σ, H, ω) sowie der Limitierung $k \in N$. Im Unterschied zur vollständig parallelen Ersetzung von T0L-Systemen werden bei klT0L-Systemen in jedem Schritt des Ersetzungsprozesses genau $\min\{k, \#_a w\}$ Vorkommen eines jeden Symbols des Alphabets Σ im gerade betrachteten Wort w ersetzt, wobei $\#_a w$ die Anzahl des Vorkommens von $a \in \Sigma$ in w ist. Für solche Systeme wurden bereits Steuerungen der Tafelanwendungen in [4] betrachtet, die entsprechend den Steuerungen bei üblichen T0L-Systemen definiert sind (siehe z.B. [1]). Es wurden periodisch zeitvariierende, Matrix-, programmierte und graph-kontrollierte klT0L-Systeme mit gewähltem Kontext (random context) mit und ohne Vorkommensprüfung betrachtet. Es zeigte sich, daß die damit erzeugten Sprachfamilien schon bei gleichem k voneinander verschieden sind und sich bei verschiedenen Limitierungen k jeweils unvergleichbare Sprachfamilien ergeben. Hier soll nun die Steuerung der Anwendung der Tafeln durch eine reguläre Steuersprache betrachtet werden, wobei wir abweichende Ergebnisse erhalten.

Es ist $G = (\Sigma, H, \omega, k, R)$ ein regulär kontrolliertes klT0L-System, wenn $R \subset H^*$ eine reguläre Menge ist. Die von G erzeugte Sprache besteht aus allen Wörtern $w \in \Sigma^*$ mit einer Ableitung

$$D : \omega \Longrightarrow_{h_{i_1}} w_1 \Longrightarrow_{h_{i_2}} \ldots \Longrightarrow_{h_{i_n}} w_n = w$$

gemäß G, so daß $h_{i_\nu} \in H, \nu = 1, \ldots, n, n \in N \cup \{0\}$, mit $h_{i_1} h_{i_2} \ldots h_{i_n} \in R$ gilt. Die entsprechende Sprachfamilie werde mit $\mathcal{L}(rc, k\text{lT0L})$ bezeichnet. Wir betrachten weiter Abbildungen $oc, noc : H \to \mathcal{P}(\Sigma)$, die jeder Tafel eine Vorkommens- bzw. Nichtvorkommensmenge zuordnen. Ein Wort w kann gemäß einer Tafel h genau

dann weiter abgeleitet werden, wenn alle Symbole aus $oc(h)$ in w vorkommen, jedoch kein Symbol aus $noc(h)$. In Verbindung mit der vorhergehenden Definition erhalten wir regulär kontrollierte klT0L-Systeme $G = (\Sigma, H, \omega, R, oc, noc)$ mit gewähltem Kontext mit oder ohne (falls $noc(h) = \emptyset$ für alle $h \in H$ gilt) Vorkommensprüfung. Die entsprechenden Sprachfamilien werden als $\mathcal{L}(rc, \textit{rand}, k\text{lT0L})$ bzw. $\mathcal{L}(rc, \textit{rand-app}, k\text{lT0L})$ geschrieben, und es gilt nach den Definitionen

$$\mathcal{L}(rc, k\text{lT0L}) \subset \mathcal{L}(rc, \textit{rand}, k\text{lT0L}) \subset \mathcal{L}(rc, \textit{rand-app}, k\text{lT0L}).$$

Im Gegensatz zu den Sprachfamilien in [4] erhalten wir hier:

Satz 1: Für alle $k \in N$ gilt

$$\mathcal{L}(rc, \textit{rand-app}, k\text{lT0L}) = \mathcal{L}(rc, \textit{rand-app}, 1\text{lT0L}) \text{ und}$$
$$\mathcal{L}(rc, k\text{lT0L}) \subset \mathcal{L}(rc, 1\text{lT0L}),\ \mathcal{L}(rc, \textit{rand}, k\text{lT0L}) \subset \mathcal{L}(rc, \textit{rand}, 1\text{lT0L}).$$

Beweis: Der Beweis der angegebenen Inklusionen sowie von

$$\mathcal{L}(rc, \textit{rand-app}, k\text{lT0L}) \subset \mathcal{L}(rc, \textit{rand-app}, 1\text{lT0L})$$

ist recht einfach und soll hier nicht angegeben werden. Wir zeigen

$$\mathcal{L}(rc, \textit{rand-app}, 1\text{lT0L}) \subset \mathcal{L}(rc, \textit{rand-app}, k\text{lT0L}),$$

weil dabei das Zusammenspiel der verschiedenen Parameter der Systeme sichtbar wird. Es sei $G = (\Sigma, H, \omega, 1, R, oc, noc)$ ein regulär kontrolliertes 1lT0L-System mit gewähltem Kontext und Vorkommensprüfung. Wir definieren $G' = (\bar{\Sigma}, H', \omega, k, R', oc', noc')$ mit

$$\Sigma' = \{a' \mid a \in \Sigma\},\ \Sigma_s = \{s_a \mid a \in \Sigma\},\ \bar{\Sigma} = \Sigma \cup \Sigma' \cup \Sigma_s \text{ und}$$
$$H' = \{h_1, h_3 \mid h \in H\} \cup \{h_2^A \mid \emptyset \neq A \subset \Sigma, h \in H\}.$$

Wir zeigen, daß mit der Folge $h_1 h_2^A h_3$ von Tafeln ein Ableitungsschritt gemäß h simuliert werden kann. Es ist

$$h_1(a) = \{a, a's_a^k\},\ h_1(a') = \{a'\},\ h_1(s_a) = \{s_a\} \text{ für } a \in \Sigma,$$
$$oc'(h_1) = oc(h),\ noc'(h_1) = noc(h)$$

definiert. Unter Berücksichtigung der Vorkommens- und Nichtvorkommensmengen von h wird wahlweise a durch $a's_a^k$ ersetzt oder bleibt unverändert. Für k Symbole a, falls vorhanden, wird diese Wahl durchgeführt. Die Tafel h_2 mit

$$h_2^A(a) = \{a\},\ h_2^A(a') = \{a'\},\ h_2^A(s_a) = \varepsilon \text{ für } a \in \Sigma,$$
$$oc'(h_2^A) = \{s_a \mid a \in A\},\ noc'(h_2^A) = (\Sigma - A) \cup (\Sigma' - \{a' \mid a \in A\})$$

ist nur anwendbar, wenn k Symbole s_a für $a \in A$ vorhanden sind, von denen dann genau k gelöscht werden. Die Tafel ist nur anwendbar für Wörter $w \in A^*$,

bei denen jedes Symbol aus A in w vorkommt. Verschiedene Teilmengen $A \subset \Sigma$ sind nötig, da in einem Wort nicht jedes Symbol von Σ vorkommen muß.

Wenn nach Anwendung von h_2^A kein Symbol s_a übrigbleibt, hat man anfangs genau ein a in $a's_a^k$ überführt. Eine Simulation eines korrekten Ableitungsschritts des 1-limitierten Systems G wird nun durch die Tafel h_3 mit

$$h_3(a) = \{a\},\ h_3(a') = h(a),\ h_3(s_a) = \{s_a\} \text{ für } a \in \Sigma,$$
$$oc'(h_3) = \emptyset,\ noc'(h_3) = \Sigma_s$$

abgeschlossen. Hätte man anfangs mehr als ein Symbol a in $a's_a^k$ überführt, so wären jetzt noch Symbole aus Σ_s vorhanden und h_3 nicht anwendbar.

Wir definieren also eine reguläre Substitution sb auf H^* durch

$$sb(h) = \{h_1\}\{h_2^A \mid \emptyset \neq A \subset \Sigma\}\{h_3\} \text{ für alle } h \in H.$$

Folglich ist $sb(R)$ regulär. Wir wählen $R' = sb(R)$ und schließen, daß die von G und G' erzeugten Sprachen gleich sind. □

Ob die Inklusionen echt sind oder nicht, ist offen. Der Satz zeigt, daß die Familien der regulär kontrollierten klT0L-Systeme mit gewähltem Kontext und Vorkommensprüfung für verschiedene k gleich sind. Ist $\mathcal{L}(re)$ die Familie der rekursiv-aufzählbaren Sprachen, dann erhalten wir außerdem

Satz 2: Für alle $k \in N$ gilt $\mathcal{L}(re) = \mathcal{L}(rc, \textit{rand-app}, k\text{lT0L})$.

Beweisskizze: Daß jede Sprache aus $\mathcal{L}(rc, \textit{rand-app}, k\text{lT0L})$ eine rekursiv-aufzählbare Sprache ist, ergibt sich aus der Churchschen These. Umgekehrt ist bekannt, daß jede rekusiv-aufzählbare Sprache durch eine programmierte kontextfreie Grammatik mit Vorkommensprüfung erzeugt werden kann. Zu dieser Grammatik wird zunächst ein graph-kontrolliertes 1lT0L-System mit gewähltem Kontext und Vorkommensprüfung konstruiert, das die Satzformen der gegebenen Grammatik erzeugt. Zu diesem System wird schließlich ein regulär kontrolliertes 1lT0L-System mit gewähltem Kontext und Vorkommensprüfung angegeben, das die gegebene rekusiv-aufzählbare Sprache erzeugt. Bei diesem letzten System kommt es vor allem darauf an, daß Wörter mit Nichtterminalzeichen durch eine Testtafel h_t zum Abschluß der Ableitung nicht durchgelassen werden. Dies wird dadurch erreicht, daß die Nichtvorkommensmenge $noc(h_t)$ gleich der Menge der Nichtterminalzeichen der gegebenen Grammatik gesetzt wird. □

Die Erzeugungsmächtigkeit der regulär kontrollierten klT0L-Systeme mit gewähltem Kontext und Vorkommensprüfung beruht also vor allem auf den Abbildungen $noc : H \to \mathcal{P}(H)$.

Weiter erhalten wir

Satz 3: Für alle $k \in N$ gilt $\mathcal{L}(rc, k\text{lT0L}) \subsetneq \mathcal{L}(rc, \textit{rand-app}, k\text{lT0L})$. □

Diese echte Inklusion wird durch die Sprache $\{a^{2^n} \mid n \geq 0\}$ gegeben. Die genaue Lage von $\mathcal{L}(rc, rand, k\text{lT0L})$ bezüglich der beiden Sprachfamilien ist unbekannt.

Wird mit $\mathcal{L}(metalin)$ die Familie der metalinearen Sprachen bezeichnet, also $\mathcal{L}(metalin) = \cup_{r \in N} \mathcal{L}(r\text{-}lin)$ (Vereinigung der Familien der r-linearen Sprachen), so kann der folgende Satz bewiesen werden.

Satz 4: Für alle $k \in N$ gelten die folgenden Aussagen: Es ist

$$\mathcal{L}(metalin) \subsetneq \mathcal{L}(rc, k\text{lT0L}).$$

Für $L \in \mathcal{L}(metalin)$ folgt $L^* \in \mathcal{L}(rc, k\text{lT0L})$. □

Weiter kann z.B. gezeigt werden, daß die Sprachfamilien $\mathcal{L}(rc, k\text{lT0L})$ und $\mathcal{L}(rc, rand, k\text{lT0L})$ abgeschlossen sind unter Vereinigungsbildung.

Eine ausführliche Darstellung dieser Ergebnisse ist in [5] erschienen. Regulär kontrollierte klET0L-Systeme wurden bereits in [2] untersucht.

Literatur

[1] Dassow, J., Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Akademie-Verlag, Berlin, 1989, Springer, Berlin, 1990.

[2] Gärtner, S., A remark on the regulation of klET0L systems, *Information Processing Letters* **48**(1993), 83–85.

[3] Wätjen, D., k-limited 0L Systems and Languages, *J. Inform. Process. Cybern. EIK* **24**(1988), 267–285.

[4] Wätjen, D., Regulation of k-limited ET0L Systems, *Intern. J. Computer Math.* **47**(1993), 29–41.

[5] Wätjen, D., On regularly controlled k-limited T0L Systems, *Intern. J. Computer Math.* **55**(1995), 57–66.

GI-Fachgruppe 0.1.5 „Automaten und Formale Sprachen“

Wahl der Fachguppenleitung

Die Wahl der Leitung der GI-Fachgruppe 0.1.5 wurde am 29. September 1994 im Rahmen der Fachgruppen-Sitzung auf dem 4. Theorietag „Automaten und Formale Sprachen“ in Herrsching durchgeführt. Anwesend waren folgende Fachgruppen-Mitglieder:

Michael Bertol (Stuttgart), Oliver Boldt (Magdeburg), Bernd Borchert (Heidelberg), Franz J. Brandenburg (Passau), Carsten Damm (Trier), Jürgen Dassow (Magdeburg), Volker Diekert (Stuttgart), Manfred Droste (Dresden), Henning Fernau (Karlsruhe), Rudolf Freund (Wien), Annegret Habel (Bremen), Christian Herzog (Frankfurt am Main), Markus Holzer (München), Klaus P. Jantke (Leipzig), Dietrich Kuske (Dresden), Martin Kutrib (Gießen), Klaus-Jörn Lange (München), Helmut Lescow (Kiel), Gundula Niemann (Würzburg), Friedrich Otto (Kassel), Holger Petersen (Hamburg), Bernd Reichel (Magdeburg), Klaus Reinhardt (Stuttgart), Peter Rossmanith (München), Sebastian Seibert (Kiel), Andreas Stübinger (Passau), Ralf Stiebe (Magdeburg), Walter Vogler (Augsburg), Dietmar Wätjen (Braunschweig).

Zum Wahlleiter wurde Herr Markus Holzer bestimmt. Der Wahlleiter stellte die 7 Kandidaten vor. Von den 29 abgegebenen Stimmzetteln waren 29 gültig. Es wurden gewählt

Jürgen Dassow (Magdeburg), Manfred Droste (Dresden), Annegret Habel (Bremen), Friedrich Otto (Kassel), Wolfgang Thomas (Kiel).

Alle Gewählten nahmen die Wahl an. Das Wahlprotokoll wurde verlesen und genehmigt.

Wahl des Fachgruppensprechers

Am 30. September 1994 einigten sich die anwesenden Mitglieder der Fachgruppenleitung darauf, daß Herr Dassow weiterhin das Amt des Sprechers und Herr Thomas das Amt des stellvertretenden Sprechers wahrnehmen werden. Dieser Vorschlag wurde einstimmig angenommen. Herr Dassow nahm die Wahl an; Herr Thomas erklärte telefonisch seine Bereitschaft zur Übernahme des Amtes.

Teilnehmerliste

Michael Bertol
Institut für Informatik
Universität Stuttgart
Breitwiesenstraße 20–22
70565 Stuttgart

Tel.: +49-711-7816-344
Fax: +49-711-7816-310

e-mail:
bertol@informatik.uni-stuttgart.de

Oliver Boldt
Fakultät für Informatik
Otto-von-Guericke-Universität
Postfach 4120
39016 Magdeburg

Tel.:
Fax:

e-mail:

Bernd Borchert
Mathematisches Institut
Im Neuenheimer Feld 294
69121 Heidelberg

Tel.: +49-6221-56-3282
Fax:

e-mail:
bb@math.uni-heidelberg.de

Franz J. Brandenburg
Universität Passau
Lehrstuhl für Informatik
Innstraße 33
94030 Passau

Tel.: +49-851-509-343
Fax: +49-851-509-374

e-mail:
brandenb@fmi.uni-passau.de

Carsten Damm
Universität Trier
FB IV — Informatik
54286 Trier

Tel.: +49-651-201-2831
Fax: +49-651-201-3954

e-mail:
damm@uni-trier.de

Jürgen Dassow
Fakultät für Informatik
Otto-von-Guericke-Universität
Postfach 4120
39016 Magdeburg

Tel.: +49-391-5592-3543
Fax: +49-391-5592-157

e-mail:
dassow@irb.cs.tu-magdeburg.de

Volker Diekert
Institut für Informatik
Universität Stuttgart
Breitwiesenstraße 20–22
70565 Stuttgart

Tel.: +49-711-7816-329
Fax: +49-711-7816-310

e-mail:
diekert@informatik.uni-stuttgart.de

Manfred Droste
Institut für Algebra
Technische Universität Dresden
Mommsenstr. 13
01062 Dresden

Tel.: +49-351-463-3908
Fax:

e-mail:
droste@nalw01.math.tu-dresden.de

Jürgen Duske
Institut für Informatik
Universität Hannover
Welfengarten 1
30167 Hannover

Tel.: +49-511-762-5184
Fax: +49-511-762-3675

e-mail:
jd@informatik.uni-hannover.de

Gudrun Falkner
FH Stralsund
FB Elektrotechnik
Große Parower Str. 145
18435 Stralsund

Tel.: +49-3831-367-596
Fax: +49-3831-367-680

e-mail:

Henning Fernau
Lehrstuhl Informatik
für Ingenieure und Naturwissenschaftler
Universität Karlsruhe
Am Fasangarten 5
76128 Karlsruhe

Tel.: +49-721-608-4336
Fax: +49-721-698675

e-mail:
fernau@ira.uka.de

Rudolf Freund
Technische Universität Wien
Institut für Computersprachen
Resselg. 3
A-1040 Wien
Österreich

Tel.: +43-1-58801-4084
Fax: +43-1-5041589

e-mail:
freund@csdec1.tuwien.ac.at

Annegret Habel
Universität Bremen
Fachbreich Mathematik und Informatik
Postfach 330 440
28334 Bremen

Tel.: +49-421-218-3489
Fax: +49-421-218-4322

e-mail:
habel@Informatik.Uni-Bremen.DE

Christian Herzog
Fachbereich Informatik
Johann-Wolfgang-Goethe-Universität
Postfach 11 19 32
60054 Frankfurt am Main

Tel.: +49-69-798-8413
Fax: +49-69-798-8353

e-mail:
herzog@psc.informatik.uni-frankfurt.de

Markus Holzer
Fakultät für Informatik
Technische Universität München
Arcisstr. 21
80290 München

Tel.: +49-89-2105-2397
Fax: +49-89-2105-8207

e-mail:
holzer@informatik.tu-muenchen.de

Klaus P. Jantke
HTWK Leipzig
FB IMN
Postfach 66
04251 Leipzig

Tel.: +49-341-3928-426
Fax: +49-341-3928-426

e-mail:
jantke@informatik.th-leipzig.de

Dietrich Kuske
Institut für Algebra
Technische Universität Dresden
Mommsenstr. 13
01062 Dresden

Tel.: +49-351-463-3642
Fax: +49-531-463-4235

e-mail:
kuske@nalw01.math.tu-dresden.de

Martin Kutrib
Universität Gießen
AG Informatik
Arndtstr. 2
35392 Gießen

Tel.: +49-641-702-2540
Fax: +49-641-702-2548

e-mail:
kutrib@informatik.uni-giessen.de

Klaus-Jörn Lange
Fakultät für Informatik
Technische Universität München
Arcisstr. 21
80290 München

Tel.: +49-89-2105-2403
Fax: +49-89-2105-8207

e-mail:
lange@informatik.tu-muenchen.de

Hans Leiß
Universität München
Centrum für Informations- und
Sprachverarbeitung
Wagmüllerstr. 23
80538 München

Tel.: +49-89-21106-73
Fax: +49-9-21106-74

e-mail:
leiss@cis.uni-muenchen.de

Helmut Lescow
Institut für Informatik und Praktische
Mathematik
Universität Kiel
24098 Kiel

Tel.: +49-431-880-4483
Fax:

e-mail:
hel@informatik.uni-kiel.d400.de

Gundula Niemann
Theoretische Informatik
Uni Würzburg
Am Exerzierplatz 3
97072 Würzburg

Tel.: +49-931-79621-14
Fax: +49-931-79621-20

e-mail:
niemann@informatik.uni-wuerzburg.de

Friedrich Otto
Fachbereich Mathematik/Informatik
Universität GH Kassel
Postfach 101380
34109 Kassel

Tel.: +49-561-804-4573
Fax: +49-561-8044199

e-mail:
otto@theory.informatik.uni-kassel.de

Holger Petersen
Fachbereich Informatik
Vogt-Kölln-Str. 30
22527 Hamburg

Tel.: +49-40-54715-237
Fax: +49-40-54715-246

e-mail:
petersen@rzdspc1.informatik.uni-hamburg.de

Bernd Reichel
Fakultät für Informatik
Otto-von-Guericke-Universität
Postfach 4120
39016 Magdeburg

Tel.: +49-391-5592-2851
Fax: +49-391-5592-2810

e-mail:
reichel@irb.cs.tu-magdeburg.de

Klaus Reinhardt
Institut für Informatik
Universität Stuttgart
Breitwiesenstraße 20–22
70565 Stuttgart

Tel.: +49-711-7816-410
Fax: +49-711-7816-310

e-mail:
reinhard@informatik.uni-stuttgart.de

Peter Rossmanith
Fakultät für Informatik
Technische Universität München
Arcisstr. 21
80290 München

Tel.: +49-89-2105-2397
Fax: +49-89-2105-8207

e-mail:
rossmani@informatik.tu-muenchen.de

Sebastian Seibert
Institut für Informatik und Praktische
Mathematik
Universität Kiel
24098 Kiel

Tel.: +49-431-880-4483
Fax:

e-mail:
ss@informatik.uni-kiel.d400.de

Andreas Stübinger
Universität Passau
Lehrstuhl für Informatik
Innstraße 33
94030 Passau

Tel.: +49-851-509-779
Fax: +49-851-509-374

e-mail:
stuebing@fmi.uni-passau.de

Ralf Stiebe
Fakultät für Informatik
Otto-von-Guericke-Universität
Postfach 4120
39016 Magdeburg

Tel.:
Fax:

e-mail:
stiebe@irb.cs.tu-magdeburg.de

Walter Vogler
Institut für Mathematik
Universität Augsburg
Universitätsstr. 14
86135 Augsburg

Tel.: +49-821-598-2120
Fax: +49-821-598-2200

e-mail:
vogler@uni-augsburg.de

Dietmar Wätjen
Institut für Theoretische Informatik
TU Braunschweig
Postfach 3329
38023 Braunschweig

Tel.: +49-531-3919520
Fax: +49-531-3919529

e-mail:
waetjen@iti.cs.tu-bs.de