1. Theorietag

Automaten und formale Sprachen

30. September 1991 - 1. Oktober 1991
in Magdeburg

1. Theorietag
Automaten und formale Sprachen

Am 30.September / 1.Oktober 1991 fand an der Technischen Universität „Otto von Guericke" Magdeburg der 1. Theorietag zu Automaten und formalen Sprachen statt. Es war das erste Treffen der Interessenten für die in Gründung befindliche Fachgruppe „Automaten und formale Sprachen" des Fachbereichs „Grundlagen der Informatik" in der Gesellschaft für Informatik.

Die inhaltlichen Schwerpunkte der Fachgruppenarbeit sollen sowohl auf den namensgebenden Gebieten Automatentheorie (z.B. Studium und Vergleich von Automatentypen, auch über komplexen Objekten wie Bäumen, Graphen, Bildern, zellulare und systolische Automaten, neuronalen und Petri-Netzen; Schaltkreise) und Theorie der formalen Sprachen (z.B. Sprachklassen und ihre Eigenschaften, Zeichenketten- und Termersetzungssysteme, Grammatiken, algebraische Methoden wie formale Potenzreihen u.ä.) als auch in den Anwendungen in Komplexitätstheorie, Rekursionstheorie, Algorithmentheorie, Programmierung, Logik, Semantik, Kombinatorik, Kodierungstheorie u.ä. liegen.
Die Vorträge des 1. Theorietages spiegelten dieses Spektrum annähernd wider.

In einer kurzen abschließenden Diskussion wurde angeregt, derartige Tage (wegen der inhaltlichen Beziehungen zu anderen Theorietagen und Workshops) jährlich einmal durchzuführen. Sie sollten so gestaltet werden, daß sie nicht nur ein Forum der Spezialisten sind, sondern auch den Nachwuchswissenschaftlern eine breite und aktive Teilnahme ermöglichen.

Dieses Heft enthält das Programm des 1. Theorietages Automaten und formale Sprachen, die nach den Autoren alphabetisch geordneten Kurzfassungen der Beiträge von jeweils ca. 3 Seiten und die Liste der Teilnehmer.

Jürgen Dassow
Örtliche Tagungsleitung

1. Theorietag
Automaten und formale Sprachen
Programm

Montag, den 30. September 1991

13.00 - 13.15	Eröffnung
13.15 - 14.15	*Matthias Jantzen und Holger Petersen (Universität Hamburg)* Erweiterung durch Reduktion - am Beispiel der kontext-freien Sprachen
14.15 - 14.45	*Hans-Jörg Kreowski (Universität Bremen)* Beschränktheitsprobleme für Kantenersetzungssysteme
14.45 - 15.15	Kaffeepause
15.15 - 15.45	*Rudolf Freund (Techn. Universität Wien)* Regulated rewriting of arrays
15.45 - 16.15	*Armin Hemmerling (Universität Greifswald)* Labyrinthprobleme und Komplexitätstheorie
16.15 - 16.45	*Ralf Stiebe (Techn. Universität Magdeburg)* Einige Bemerkungen über Matrixbildsprachen
16.45 - 17.15	*Sebastian Siebert (Universität Kiel)* Quantorenhierarchien über automaten-definierten Relationen
18.00	Geselliges Beisammensein

Dienstag, den 1. Oktober 1991

08.30 - 09.30	*Klaus-Jörn Lange (Techn. Universität München)* Formale Sprachen und Komplexitätstheorie
09.30 - 10.00	*Gerhard Buntrock und Krysztof Lorys (Universität Würzburg)* On growing context-sensitive languages
10.00 - 10.30	Kaffeepause
10.30 - 11.00	*Heiko Vogler und Armin Kühnemann (Universität Ulm)* Synthesized and inherited functions on trees - a new computational model for syntax-directed semantics
11.00 - 11.30	*Eckehart Hotzel (GMD St.Augustin)* Nichtdeterminismus, Hilfssymbole und Akzeptanzbedingungen in der Umgebung der endlichen Automaten
11.30 - 12.00	*Henning Bordihn (Techn. Universität Magdeburg)* Determiniertheitsgrad und Entscheidbarkeitsfragen

DETERMINIERTHEITSGRAD UND ENTSCHEIDBARKEITSFRAGEN

Henning Bordihn
Fakultät für Mathematik
Technische Universität Magdeburg

Der Determiniertheitsgrad ist ein Maß der syntaktischen Beschreibungskompliziertheit, das von den biologisch motivierten Lindenmayer-Systemen her bekannt ist. Erstmals wurde der Determiniertheitsgrad von ROZENBERG im Zusammenhang mit TOL-Systemen betrachtet (s. [7]). Das Maß charakterisiert die maximale Anzahl von Worten, durch die ein und dasselbe Symbol in einem Schritt des Ableitungsprozesses ersetzt werden kann. Bezeichnen wir den Determiniertheitsgrad eines TOL-Systems G mit $Det_{TOL}(G)$, dann ist für eine TOL Sprache L der Determiniertheitsgrad definiert als

$$Det_{TOL}(L) = \min \{Det_{TOL}(G) \mid L(G) = L,\ G \text{ ist ein TOL-System}\}$$

Somit sind deterministische TOL-Sprachen gerade solche, deren Determiniertheitsgrad gleich 1 ist.
ROZENBERG konnte nachweisen, daß der Determiniertheitsgrad eine unendliche Hierarchie von TOL-Sprachen induziert.

Lemma 1 [7]. *Für jede natürliche Zahl $n \geq 1$ existiert eine TOL-Sprache L mit $Det_{TOL}(L) = n$.*

Betrachtet man nun den Determiniertheitsgrad im Zusammenhang mit sequentiellen Ersetzungssystemen im Sinne von CHOMSKY, etwa mit kontextfreien Grammatiken, so findet man aufgrund der Rolle der nichtterminalen Symbole durch relativ einfache Überlegungen das folgende:

Lemma 2 [2]. *Sei $L \in \mathcal{L}(CF)$. Dann gilt:*

i) $Det_{CF}(L) = 1$ *gdw.* $card\ L \leq 1$.

ii) $Det_{CF}(L) = 2$ *gdw.* $card\ L > 1$.

Daher ist es von Interesse, den Determiniertheitsgrad in Bezug auf Grammatiken ohne Alphabetpartition in Terminale und Nichtterminale, sogenannte reine Grammatiken, zu betrachten. Diese sequentiellen Entsprechungen der L-Systeme wurden erstmals in [5] betrachtet. Reine kontextfreie, kontext-sensitive bzw. λ-freie Grammatiken werden analog zu den Begriffsbildungen bei den Chomsky-Grammatiken definiert. Die zugehörigen Sprachfamilien werden mit $\mathcal{L}(\mathrm{pCF})$, $\mathcal{L}(\mathrm{pCF}^{\lambda})$, $\mathcal{L}(\mathrm{pCS})$ bzw. $\mathcal{L}(\mathrm{pCS}^{\lambda})$ bezeichnet, wobei der obere Index λ bedeutet, daß verkürzende Regeln zur Erzeugung der Sprachen benutzt werden dürfen. Sie bilden die sogenannte *reine Chomsky-Hierarchie* (s. z.B. [3]):

$$\mathcal{L}(\mathrm{pCF}) = \mathcal{L}(\mathrm{pCF}^{\lambda}) \subset \mathcal{L}(\mathrm{pCS}) \subset \mathcal{L}(\mathrm{pCS}^{\lambda})$$

Führt man nun in naheliegender Weise den Determiniertheitsgrad für diese Klassen reiner Grammatiken und Sprachen ein (vgl. etwa [1]), so gelangt man zu den folgenden Ergebnissen.

Lemma 3 [1]. *Sei $X \in \{\mathrm{CF}, \mathrm{CF}^{\lambda}, \mathrm{CS}, \mathrm{CS}^{\lambda}\}$. Für jede natürliche Zahl $n \geq 1$ existiert eine reine Sprache $L \in \mathcal{L}(\mathrm{p}X)$ mit $Det_{\mathrm{p}X}(L) = n$.*

Lemma 4 [1]. *Es sei für $X \in \{\mathrm{CF}, \mathrm{CF}^{\lambda}, \mathrm{CS}, \mathrm{CS}^{\lambda}\}$*

$$\mathcal{L}_n(\mathrm{p}X) = \{L \in \mathcal{L}(\mathrm{p}X) \mid Det_{\mathrm{p}X}(L) \leq n\}$$

Dann gilt für jede natürliche Zahl $n \geq 1$

$$\begin{array}{ccccc} & & \mathcal{L}_n(\mathrm{pCS}) & & \\ & \subset & & \subset & \\ \mathcal{L}_n(\mathrm{pCF}) & & & & \mathcal{L}_n(\mathrm{pCS}^{\lambda}). \\ & \subset & & \subset & \\ & & \mathcal{L}_n(\mathrm{pCF}^{\lambda}) & & \end{array}$$

Da der Determiniertheitsgrad jede der betrachteten Familien reiner Sprachen genau wie die Familie $\mathcal{L}(\mathrm{TOL})$ in eine unendliche Hierarchie zerlegt, liegt es nahe, die folgenden Entscheidbarkeitsfragen für syntaktische Kompliziertheitsmaße in Bezug auf den Determiniertheitsgrad zu stellen, die von Gruska [6] in analoger Weise für die Maße Anzahl der Variablen, Anzahl der Produktionen und Anzahl der Symbole in kontextfreien Grammatiken behandelt worden sind.

Wir formulieren die Probleme hier für TOL-Systeme; für reine Grammatiken kann völlig analog gefragt werden.

P1 Gibt es einen Algorithmus, der für ein beliebiges TOL-System G den Determiniertheitsgrad $Det_{\mathrm{TOL}}(G)$ berechnet?

P2 Gibt es einen Algorithmus, der $Det_{\mathrm{TOL}}(L(G))$ für ein beliebiges TOL System G bestimmt?

P3 Ist es für ein beliebiges TOL System G und eine beliebig gegebene natürliche Zahl n algorithmisch entscheidbar, ob $Det_{\mathrm{TOL}}(L(G)) = n$ gilt?

P4 Ist es für ein beliebig gegebenes TOL System G entscheidbar, ob $Det_{\mathrm{TOL}}(L(G)) = Det_{\mathrm{TOL}}(G)$ gilt? (Ist es entscheidbar, ob G ein TOL-System mit minimalem Determiniertheitsgrad ist, die die Sprache $L(G)$ erzeugt?)

P5 Gibt es einen Algorithmus, der für ein beliebig gegebenes TOL-System G ein TOL System G' so konstruiert, daß sowohl $L(G') = L(G)$ als auch $Det_{\mathrm{TOL}}(G') = Det_{\mathrm{TOL}}(L(G))$ ist? (Gibt es einen Algorithmus, der für eine beliebig gegebene Sprache $L \in \mathcal{L}(\mathrm{TOL})$ ein erzeugendes TOL System mit minimalem Determiniertheitsgrad konstruiert?)

Satz 5. *Die Antwort auf die Fragestellung P1 ist positiv, die Antworten auf die Fragen P2 bis P5 negativ*

i) für TOL-*Systeme* ([4]),

ii) für pX-*Grammatiken*, $X \in \{\mathrm{CF}, \mathrm{CF}^{\lambda}, \mathrm{CS}, \mathrm{CS}^{\lambda}\}$ ([2]).

Literatur

[1] Bordihn, H., Pure languages and the degree of nondeterminism. *Eingereicht bei* EIK.

[2] Bordihn, H., Über den Determiniertheitsgrad reiner Versionen formaler Sprachen. *Eingereicht als* Dissertation.

[3] Dassow, J., Pure languages with regulated rewriting. Revue Roum. Math. Pures Et Appl. 31 (86), 657-666.

[4] Dassow, J., A note on the determinism in TOL systems. *Eingereicht* zur Veröffentlichung.

[5] Gabrielian, A., Pure grammars and pure languages. Techn. rep. CSRR 2027 (70), Univ. of Waterloo, Dept. comp. Sc.

[6] Gruska, J., Complexity and unambiguity of context-free grammars and languages. Inform. Control 18 (71), 502-512.

[7] Rozenberg, G., TOL systems amd languages. Inform. Control 23 (73), 357-381.

On growing context-sensitive languages

(Extended Abstract, October 1991)

Gerhard Buntrock* and Krzysztof Loryś*†

Elias Dahlhaus and Manfred Warmuth in 1986 gave an efficient algorithm for the membership problem of a subclass of context-sensitive languages, namely the growing context-sensitive languages ([DW86])

Definition 1 *A context-sensitive grammar $G = (N, T, S, P)$ is growing if S does not appear on the right side and $|\alpha| < |\beta|$ for any $(\alpha \to \beta), \alpha \neq S$ from P. We denote the growing grammars by* GCSG *and the class of languages generated by such grammars by* GCSL.

Dahlhaus and Warmuth proved that the languages defined by growing production rules can be decided in polynomial time. Their proof even shows that this class is included in the closure of context-free languages (CFL) under logarithmic space-bounded reductions (LOGCFL).

By a careful examination of this proof one can get a somewhat stronger result.

Theorem 2 *The class* GCSL *is contained in the closure of* CFL *under one-way logarithmic space-bounded reductions (*1-LOGCFL*), i.e. the Turing machine computing the reduction function has access to the input only in one-way mode.*

The proof uses Clemens Lautemann's characterization of 1-LOGCFL by polynomial time and logarithmic space bounded 1-AuxPDA computations of Clemens Lautemann ([Lau88]).

Dahlhaus and Warmuth asked for the complexity of the general membership problem for GCSGs ([DW86]). The question is whether this problem can be solved in polynomial time or whether it is NP-complete. We solve this problem with the following theorem:

Theorem 3 *The language $\{(G, w) \mid G$ is a* GCSG*, w is a word of the language defined by $G\}$ is* NP*-complete.*

For the proof we reduce the satisfiability problem to this general membership problem. For each formula in conjunctive normal form we construct a GCSG and a string over a one letter alphabet. The grammar from the start symbol produces the formula and checks the satisfiability by applying growing context sensitive rules. If the formula is satisfiable we will get a polynomially long string. Otherwise the language produced by this grammar is empty.

It has been proven valuable to apply the theory of AFLs. This is not trivially possible for GCSL. Most work to show the AFL properties consists of proving the following proposition

*Institut für Informatik, Universität Würzburg, D-8700 Würzburg, Germany.

†Instytut Informatyki, Uniwersytet Wrocławski, 51-151 Wrocław, Poland (permanent address). This research was supported by Humboldt Foundation

Proposition 4 *The class* GCSL *is closed under inverse homomorphisms*

In the proof we use a new normal form theorem.

Definition 5 *We call a growing context-sensitive grammar $G = (N, T, P, S)$ k-normal if all productions have one of the following form.*

(1) $S \to \beta$, *where* $\beta \in T^*$ *and* $|\beta| \leq k$

(2) $S \to \beta$, *where* $\beta \in (T \cup N)^k$

(3) $\alpha \to \beta$, *where* $\alpha, \beta \in (T \cup N)^*$, $|\alpha| = k$ *and* $|\beta| = k + 1$

Proposition 6 *For each language $L \in$ GCSL there exists a number k_0 such that for all $k \geq k_0$ there exists a k-normal growing context-sensitive grammar G_k such that $L = L(G_k)$*

The main problem in the proof of Proposition 4 is the following: Let a be a letter of the terminal alphabet, h a homomorphism. Then the problem arises for the case $h(a) = w$ with $|w| > 1$. Here we have to show that if w is a part of a derivable word, then there exists a grammar which will produce the letter a instead of w. Now using our normal form it is possible to describe the derivation tree in such a way that there are some independent subgraphs which can be produced by longer rules; i.e. we will save some growth in a derivation by combining several rules into one rule with longer right and left sides. Unfortunately the proof is rather technical and tricky.

Now using Proposition 4 it is easy to prove the following theorem.

Theorem 7 *The class* GCSL *forms an* AFL

The closure result leads to a handy characterization by so called weighted grammars: each grammar can be weighted by giving weights to all nonterminal and terminal symbols. If a weight function exists such that all productions are growing with respect to these weights we call the grammar a quasi-growing grammar. With this characterization we can now omit the condition that each production be length increasing.

Definition 8 *A grammar $G = (N, T, S, P)$ is quasi-growing if there exists a weight function $f : (N \cup T) \to \mathbb{N}^+$ such that for all productions $(\alpha \to \beta) \in P$ we have $f(\alpha) < f(\beta)$, where $f(a_1 \ldots a_n) = \sum_{i=1}^{n} f(a_i)$ for $a_1 \ldots a_n \in (N \cup T)^*$.*[1] *We denote the quasi-growing grammars by* QGG

Theorem 9 *The languages producible by* QGG*s and* GCSG*s are the same.*

Proof. It is obvious that a GCSG forms a QGG with a constant weight function. Therefore only the forward direction needs to be shown.

Let L be generated by a quasi-growing grammar $G = (N, T, S, P)$ with weight function f. Without loss of generality we can assume that S does not appear on the right side of any production and that $f(S) = 1$.

Let $c \notin N \cup T$ and let h be a homomorphism such that $h(a) = ac^{f(a)-1}$ for each $a \in N \cup T$. Then $G' = (N \cup \{c\}, T, S, P')$, where $P' = \{h(\alpha) \to h(\beta) \mid (\alpha \to \beta) \in P\}$ is a growing context-sensitive grammar and $L(G) = h^{-1}(L(G'))$. □

[1] $\mathbb{N}^+$ denotes the positive integers

Such a weight function can easily be found, if there is one, using standard methods in linear algebra. With a weight function the maximal length of a derivation can be estimated to be linear. Gladkii ([Gla64]) and later also Book ([Boo71, Boo78]) investigated linear time bounded context-sensitive languages, i.e. languages with context-sensitive grammars where each word of a language can be produced in a linear number of derivation steps. Unfortunately this class contains NP-complete problems; moreover, in general it is impossible to decide if all words derivable by a given context-sensitive grammar can be produced in linear time.

For CSG there are some well known simple normal forms. The following form was proposed by Cremers [Cre73].

Definition 10 *A context-sensitive grammar is in Cremers' normal form if all productions are of the forms*

$$\begin{aligned} AB &\longrightarrow CD \\ A &\longrightarrow CD \\ A &\longrightarrow a \end{aligned}$$

where A, B, C, D are nonterminals and a is a terminal.

Obviously such a normal form is not growing. But it is straight forward to prove such a result for QGG.

Theorem 11 *For every* QGG *there exists an equivalent quasi-growing grammar in Cremers' normal form.*

Additionally, it is now much easier to prove that a language belongs to GCSL, since a QGG is often somewhat shorter than a GCSG. For example it can easily be seen that the members of the Morse/Thue sequence form a GCSL.

References

[Boo71] Ronald V. Book. Time-bounded grammars and their languages. *Journal of Computer and System Science*, 5:397–429, 1971.

[Boo78] Ronald V. Book. On the complexity of formal grammars. *Acta Informatica*, 9:171–181, 1978.

[Cre73] A. B. Cremers. Normal forms for context-sensitive grammars. *Actal*, 3:59–73, 1973.

[DW86] Elias Dahlhaus and Manfred K. Warmuth. Membership for growing context-sensitive grammars is polynomial. *Journal of Computer and System Science*, 33:456–472, 1986.

[Gla64] A. Gladkii. On the complexity of phrase structure grammars. *Algebri i Logika Sem.*, 3:29–44, 1964. (in Russian)

[Lau88] Clemens Lautemann. One pushdown and a small tape. In Klaus W. Wagner, editor, *Dirk Siefkes zum 50. Geburtstag*, pages 42–47. Technische Universität Berlin and Universität Augsburg, 1988.

Regulated Rewriting of Arrays

Rudolf Freund

Institut für Computersprachen
Technische Universität Wien, Resselgasse 3, 1040 Wien, AUSTRIA

Abstract. In the string case many control mechanisms have been investigated for many types of grammars (see [1]) As arrays can be regarded as a generalization of strings, these control menchanisms can also be applied to array grammars of arbitrary type. Because of some fundamental differences in the definitions of string grammars and array grammars respectively some astonishing results, concerning the generative power of certain types of array grammars provided with special mechanisms for controlling the rewriting of arrays, can be obtained.

For all types of a Chomsky-like hierarchy of array grammars (e g see [2] and [3]) we investigate the following classes of array grammars with control mechanisms: ***programmed*** array grammars, ***matrix*** array grammars, and ***ordered*** array grammars. As can be expected from the string case, the generative power of arbitrary and monotone array grammars is not increased by adding one of the control mechanisms cited above. For regular array grammars this is not true any more; regular programmed, matrix and ordered array grammars can generate array languages that not even are context-free. Again the regulated rewriting by context-free grammars yields the most interesting results: Any array language in the family of recursively enumerable array languages can be generated by a context-free array grammar provided with one of the control mechanisms cited above, the corresponding results being not true for ordered grammars in the string case.

Definition 1. An *(isometric)* ***array grammar*** is a quintuple $G = (V_N, V_T, P, S, \#)$, where V_N is a finite nonempty set of *nonterminals*, V_T is a finite nonempty set of *terminals*, $S \in V_N$ is the *starting symbol*, $\# \notin V_N \cup V_T$ is the background or *blank symbol*, and P is a finite nonempty set of rewriting rules of the form $\alpha \to \beta$, where the arrays α and β are geometrically identical over $V_N \cup V_T \cup \{\#\}$ and α contains at least one symbol in $V_N \cup V_T$

We say that the array x *directly generates* the array y, denoted $x \vdash_G y$, if there is a rewriting rule $\alpha \to \beta$ such that x contains α as a subarray and y is identical to x except that the subarray α is replaced by the corresponding symbols of the array β. Let $\vdash_G^*$ be the reflexive and transitive closure of $\vdash_G$

The language generated by G, denoted L(G), is the set of all arrays of terminal symbols and symbols # that can be generated from the starting symbol S in a field of symbols #

Definition 2. Let $G = (V_N, V_T, P, S, \#)$ be an array grammar

(0) G is of type 0 or ***unrestricted*** if there are no restrictions on P.

(1) G is of type 1 or ***monotonic*** if the image of each left-side symbol in $V_N \cup V_T$ is in $V_N \cup V_T$, i.e. symbols # cannot be generated.

(2) G is of type 2 or ***context-free*** if the lefft side of each rule contains exactly one nonterminal symbol in a field of symbols #

(3) G is of type 3 or ***regular*** if every rewriting rule in P is of the form $A \to a$ respectively $Av\# \to aB$, where $A, B \in V_N$, $a \in V_T$, and v is a two-dimensional integer vector with norm 1; e.g $A(0,-1)\# \to aB$ stands for $\begin{smallmatrix} A \\ \# \end{smallmatrix} \to \begin{smallmatrix} a \\ B \end{smallmatrix}$.

The families of array languages generated by array grammars of type i are denoted by $\mathfrak{L}(i)$, $i \in \{0,1,2,3\}$. In contrast to the usual definitions given in [3] we do not demand context-free grammars to be monotonic, too. Hence, $\mathfrak{L}(3) \subsetneq \mathfrak{L}(2) \subsetneq \mathfrak{L}(0)$ and $\mathfrak{L}(3) \subseteq \mathfrak{L}(1) \subsetneq \mathfrak{L}(0)$, but $\mathfrak{L}(1) - \mathfrak{L}(2) \neq \emptyset$ and $\mathfrak{L}(2) - \mathfrak{L}(1) \neq \emptyset$. If $\mathfrak{L}(1 \wedge 2)$ denotes the family of arrays generated by context-free monotone array grammars, then we obtain the CHOMSKY-hierarchy $\mathfrak{L}(3) \subsetneq \mathfrak{L}(1 \wedge 2) \subsetneq \mathfrak{L}(1), \subsetneq \mathfrak{L}(0)$, well-known from the string case.

Applying several control mechanism - considered for the string case in [1] - to array grammars we obtain the following definitions:

Definition 3. $G = (V_N, V_T, P, S, \#)$ is called a ***programmed array grammar of type i***, $i \in \{0,1,2,3\}$, if $V_N, V_T, S, \#$ are defined as in ordinary array grammars and P is a finite set of triples $(r: \alpha \to \beta, \sigma(r), \varphi(r))$, where $r: \alpha \to \beta$ is an array production over $V_N \cup V_T \cup \{\#\}$ of type i labelled by r and $\sigma(r)$ and $\varphi(r)$ are two subsets of Lab(P) Lab(P) denoting the set of labels associated with the rules in P

For two arrays x,y and two labels $r,s \in Lab(P)$ we define $(x,r) \Rrightarrow_G (y,s)$ iff for the rule $(r: \alpha \to \beta, \sigma(r), \varphi(r))$ *either*

a) the array y is obtained from the array x by applying the production $\alpha \to \beta$ and $s \in \sigma(r)$ *or*

b) $x = y$ the production $\alpha \to \beta$ cannot be applied to x, and $s \in \varphi(r)$.

The language generated by G is

$L(G) = \{x \mid x$ is an array over V_T such that $(S,r) \Rrightarrow^*_G (x,s)$ for some $r,s \in Lab(P)\}$.

Definition 4. $G = (V_N, V_T, P, S, \#, F)$ is called a ***matrix array grammar with appearance checking of type i***, $i \in \{0,1,2,3\}$, if $V_N, V_T, S, \#$ are defined as in ordinary array grammars P is a finite set of matrices of the form $m: (r_1, \ldots, r_n)$, $n \geq 1$, with usual array productions $r_i: \alpha_i \to \beta_i$, $1 \leq i \leq n$, of type i, and F is a set of productions $\alpha_i \to \beta_i$ occuring in a matrix $m: (r_1, \ldots, r_n)$ in P

For two arrays x,y over $V_N \cup V_T \cup \{\#\}$ we define $x \Rrightarrow_G y$ if and only if there are arrays $x_0, x_1, \ldots, x_n$ with $x = x_0$, $y = x_n$ and a matrix $m: (r_1, \ldots, r_n) \in P$, such that for each i, $1 \leq i \leq n$ *either* x_i is obtained from x_{i-1} by applying the production $r_i: \alpha_i \to \beta_i$ *or* the production $\alpha_i \to \beta_i$ is not applicable to x_{i-1}, $x_i = x_{i-1}$, and the production $\alpha_i \to \beta_i$ is contained in F.

In other words, the rules occuring in F can be passed over if they cannot be applied, whereas the other rules have to be used effectively One says that the rules in F are used in the ***appearance checking mode***.

The language generated in this way is $L(G) = \{x \mid x$ is a terminal array and $S \Rrightarrow^*_G x\}$.

Definition 5. $G = (V_N, V_T, P, S, \#, <)$ is called an ***ordered array grammar*** of type i, $i \in \{0,1,2,3\}$, if $V_N, V_T, P, S, \#$ are defined as in ordinary array grammars and < is a partial order on P

For two arrays x,y over $V_N \cup V_T \cup \{\#\}$ we define $x \Rrightarrow_G y$ if and only if y can be obtained from x by applying a production $r: \alpha \to \beta$ in P such that no other production in P greater than r can be applied to x. Again we define $L(G) = \{x \mid x$ is a terminal array and $S \Rrightarrow^*_G x\}$.

Intuitively the direct derivation $x \Rrightarrow_G y$ is performed by the application of a rule which is maximal (with respect to the order relation <) in the set of applicable rules.

Definiton 6. The families of array languages generated by programmed, matrix and ordered array grammars of type i, $i \in \{0,1,2,3\}$, are denoted by $\mathfrak{L}(P,i), \mathfrak{L}(M,i)$ and $\mathfrak{L}(O,i)$ respectively

The following results are obvious from the definitions:

Theorem 1. $\mathfrak{L}(i) \subseteq \mathfrak{L}(i,Y)$, $i \in \{0,1,2,3\}$, $Y \in \{P,M,O\}$

The generative power of arbitrary or monotone array grammars is not inreased by one of the control mechanisms defined above.

Theorem 2. $\mathfrak{L}(0) = \mathfrak{L}(0,Y)$ and $\mathfrak{L}(1) = \mathfrak{L}(1,Y)$ for $Y \in \{P,M,O\}$

In contrast to the results obtained in the string case regulated rewriting increases the generative power of regular array grammars.

Example 1. Consider the array language L containing the following arrays over the terminal alphabet {b,B,}:

```
bbbbb        bbbbbb       bbbbbbb
bBBBb        bBBBBb       bBBBBBb
bBBBb        bBBBBb       bBBBBBb
bBBBbb       bBBBBb       bBBBBBb        etc
bbbbbb       bBBBBbb      bBBBBBb
             bbbbbbb      bBBBBBbb
                          bbbbbbbb
```

Then $L \in (\mathfrak{L}(2,P) \cap \mathfrak{L}(2,M) \cap \mathfrak{L}(2,O)) - \mathfrak{L}(2)$, i.e L not even is context-free.

As in the string case, the regulated rewriting by context-free grammars yields the most interesting results provided we do not demand context-free array grammars to be monotone, too.

Theorem 3. $\mathfrak{L}(0) = \mathfrak{L}(2,Y)$ for $Y \in \{P,M,O\}$.

In the string case, the corresponding results are true for programmed and matrix grammars, but not for ordered grammars.

In the following we briefly sketch the idea of the constructions to prove this result First, it is rather obvious that any language L in $\mathfrak{L}(0)$ can be generated by an unrestricted array grammar $G = (V_N, V_T, P, S)$ in *normal form*, where the productions in P are of the form $A \to a$, $A \to X$, $Av\# \to XY$, and $AvC \to XY$ with $a \in V_T \cup \{\#\}$, $A,C,X,Y \in V_N$ and v being a two-dimensional integer vector with norm 1 Then only the productions of the form $Av\# \to XY$ respectively $AvC \to XY$ are not context-free. But if we start with arrays like those shown in Example 1 with one of the symbols B replaced by the starting symbol S, then we can also simulate the productions $Av\# \to XY$ respectively $AvC \to XY$ by a sequence of context-free productions like $(B \to \#, Av\# \to XY)$ respectively $(C \to \#, Av\# \to XY)$ being applied to a subarray within the boundaries of symbols b which can be guaranteed by using the specific features of the different control mechanisms. These boundary symbols b as well as the variables B standing for the blank symbol # within these boundaries finally are eliminated in order to get the desired terminal array in L(G)

Hence, when using the context of the blank symbol # in the regulated application of context-free array productions we may obtain stronger results than in the corresponding cases of controlled applications of context-free string productions.

References:

[1] Dassow, Jürgen, Păun, Gheorghe: Regulated Rewriting in Formal Language Theory Springer-Verlag, Berlin, New York, 1989.

[2] Rosenfeld, A.: Array Grammar Normal Forms, Information and Control 23 (1973), 173 - 182.

[3] Wang, P S.-P.: Some New Results on Isotonic Array Grammars Information Processing Letters 10 (1980), 129 - 131

Labyrinthprobleme und Komplexitätstheorie

Armin Hemmerling
Fachrichtungen Mathematik/Informatik
der Ernst-Moritz-Arndt-Universität
F.-L.-Jahn-Straße 15a
0-2200 Greifswald

Abstract. After introductory remarks on aims and basic concepts of labyrinth theory, the following result <BH> is sketched.
There is no jumping multihead 1-counter automaton which is able to search all planar cubic edge-coloured (finite,connected) graphs, or all planar cubic graphs with rotation systems, or all rectilinearly embedded 3-dimensional graphs.
Finally, it is discussed whether, or why not, these proof techniques apply to prove L≠NL via Savitch's result of the NL-completeness of a certain labyrinth problem in the wider sense.

0. Labyrinth-Theorie ist ein Teilgebiet der Komplexitätstheorie, das sich mit der Charakterisierung der Komplexität von Durchmusterungs- und dazu verwandten Problemen auf nichtlinearen Strukturen, Labyrinthe genannt, beschäftigt, die vom "inneren Standpunkt" aus betrachtet werden. Letzteres bedeutet, daß die entsprechenden Automaten kurzsichtig sind, stets nur über die Umgebung ihres Standortes informiert werden und sich gemäß der Topologie des vorgegebenen Labyrinths bewegen müssen.
Diese Situation ist typisch für die vielfältigsten Problemstellungen von der Roboterbewegung bis hin zur Theorie der Wissensverarbeitung oder der Berechnungskomplexität von Turing-Maschinen. Zur historischen Entwicklung der Labyrinth-Theorie, wie auch zur ausführlicheren Diskussion der grundlegenden Begriffe und Resultate, konsultiere man die Monographie <He>. In jüngster Zeit erfährt die Thematik eine starke Belebung durch Problemstellungen zur Bewegungsplanung für Roboter.

1. Ein Labyrinthproblem ist normalerweise charakterisiert als Tripel, bestehend aus einem Objekttyp (das ist eine Klasse von Labyrinthen), einem Subjekttyp (eine Klasse von Automaten, die sich auf den Labyrinthen bewegen können) und eine Aufgabe (wie das Absuchen der Labyrinthe oder auch das Erkennen oder Entscheiden von Eigenschaften). Als Labyrinthe werden meist endliche, zusammenhängende, ungerichtete Graphen mit Rotations- oder Kompaßsystemen oder Kantenfärbungen betrachtet. Die Automaten sind entsprechend spezialisierte Mealy-Automaten, eventuell mit Zusatzspeichern oder Markierungsmöglichkeiten im Labyrinth ausgestattet. Als parallel arbeitende Modelle betrachtet man kooperierende Systeme endlich vieler Automaten (mit Informationsaustausch zwischen je zweien, wenn sie benachbarte Standorte haben) oder Mehrkopfautomaten (was einem ständigen Informationsaustausch entspricht). Kombinationen all dieser Typen sind in vielfältiger Weise möglich. Bei der wichtigsten Aufgabe, dem Absuchen, sollen die Automaten bei Start auf einem beliebigen Knoten des Labyrinths alle Knoten im Laufe ihrer Arbeit besuchen.
Durch Lösungsalgorithmen für solche Problemstellungen werden obere Schranken, durch Unlösbarkeitsresultate (Fallenkonstruktionen) untere Schranken der Komplexität markiert.

2. Das folgende Resultat liefert eine Fallenkonstruktion für einen recht komfortablen Automatentyp.

Satz <BH>. Es gibt keinen springenden Mehrkopfautomaten mit einem Zähler, der alle Labyrinthe einer der folgenden Klassen von (endlichen, zusammenhängenden) Graphen absucht:

i) planare kubische kantengefärbte Graphen;
ii) planare kubische R-Graphen (d.h. mit Rotationssystem);
iii) rektilinear eingebettete 3-dimensionale Graphen.

Es bleibt offen, ob man auch ebene kubische R-Fallen oder normierte 3D Fallen konstruieren kann.

Zunächst erhält man aus dem Resultat für kantengefärbte Graphen durch Reduktion der entsprechenden Probleme recht leicht die übrigen Behauptungen. Zum Beweis für kantengefärbte Graphen wird zunächst die Existenz universeller Fallen für alle endlichen Mengen endlicher Automaten gezeigt. Durch Aufschneiden von Kanten einer universellen Falle und geeignetes Verkleben mit einem zweiten (gespiegelten) Exemplar gewinnt man eine universelle Bisperre. Der Beweis von i) für Mehrkopfautomaten verläuft nun indirekt nach dem Prinzip der kleinsten Zahl: Sei k minimal, so daß es einen k-Kopf-Automaten A gibt, der alle planaren kubischen kantengefärbten Graphen G absucht. Durch Simulation des Verhaltens von A auf dem Graphen G<B>, erhalten aus G durch Einsetzen geeigneter universeller Bisperren B in die Kanten, zeigt man nun: Wenn der Automat A korrekt arbeitet, so sind auch schon k-1 Köpfe für das Absuchen der betrachteten Graphenklasse ausreichend. Wären nämlich zu einem Zeitpunkt, bevor das Absuchen von G im wesentlichen geleistet ist, alle k Köpfe durch Kopien von B paarweise voneinander getrennt, so würde danach kein Kopf mehr eine B-Kopie durchqueren können, d.h. alle Köpfe wären gelähmt.

Zur Verallgemeinerung des Beweises auf Automaten mit springenden Köpfen und einem Zähler sind noch einige Zusatzüberlegungen nötig.

3. Das Studium von Labyrinthproblemen in den siebziger Jahren ist wesentlich durch folgendes Resultat angeregt worden.

Satz <Sa>. Es ist L=NL genau dann, wenn es einen Mehrkopfautomaten gibt, der die Durchquerbarkeit von Savitch-Labyrinthen erkennt.

Ein Savitch-Labyrinth ist dabei ein endlicher gerichteter Graph $(\{1,\ldots,n\},E)$, in dem jeder Knoten genau zwei von ihm ausgehende Kanten besitzt, die mit 0 und 1 markiert sind. Ein Automatenkopf hat hier die Möglichkeit, vom Knoten i aus durch die Kante 0 oder 1 zu laufen oder zum Knoten i+1 (modulo n) zu geben. Damit ist der innere Standpunkt der Arbeitsweise verletzt, denn die Durchquerbarkeit bezieht sich natürlich auf Kanten aus E.

Es stellt sich nun die Frage, ob man L≠NL beweisen könnte, indem man die Beweistechnik des Satzes <BH> auf ungerichtete, kantengefärbte Savitch-Labyrinthe (SEC-Labyrinthe) überträgt, um zu zeigen,daß es keinen Mehrkopfautomaten gibt, der die Durchquerbarkeit solcher Labyrinthe erkennt.

Die Anfangsschritte des Beweises, also die Konstruktion universeller Fallen und Bisperren, lassen sich noch recht gut übertragen: Endliche Automaten arbeiten in SEC-Labyrinthen autonom, so daß ihre Ausgabefolgen (der Bewegungsanweisungen) schließlich zyklisch werden. Man kann nun für endliche Mengen von schließlich zyklischen Bewegungsanweisungsfolgen, die sich bei "Reduktion modulo Baum" nicht auf Bewegungen ausschließlich entlang der linearen Knotenordnung des Graphen reduzieren, universelle SEC-Fallen konstruieren. Hieraus erhält man wieder recht leicht universelle Bisperren für solche Mengen.

Leider ist der Hauptschritt des Beweises nicht übertragbar. Sind nämlich alle Köpfe eines Automaten in einem Savitch-Labyrinth durch Bisperren voneinander getrennt, so können sie hiernach dennoch durch Verfolgung der dem Labyrinth aufgeprägten linearen Knotenordnung wieder zusammentreffen, was im Falle "eigentlicher" Labyrinthe nicht möglich ist.

Trotz dieses Mißerfolges halten wir es weiterhin für möglich, daß die Labyrinth-Theorie bisher nicht beachtete Gesichtspunkte oder Lösungsansätze für L-NL oder verwandte Probleme beitragen könnte.

Literatur:

<BH> Bull, M., A. Hemmerling. Traps for Jumping Multihead Counter Automata. Zur Veröffentlichung eingereicht. Zusammenfassung: New Results on Lower Bounds in Labyrinth Theory. in Geometrical Problems of Image Processing, Akademie-Verlag, Berlin 1991, 131-135.

<He> Hemmerling, A.: Labyrinth Problems. Labyrinth-Searching Abilities of Automata. Teubner, Leipzig 1989.

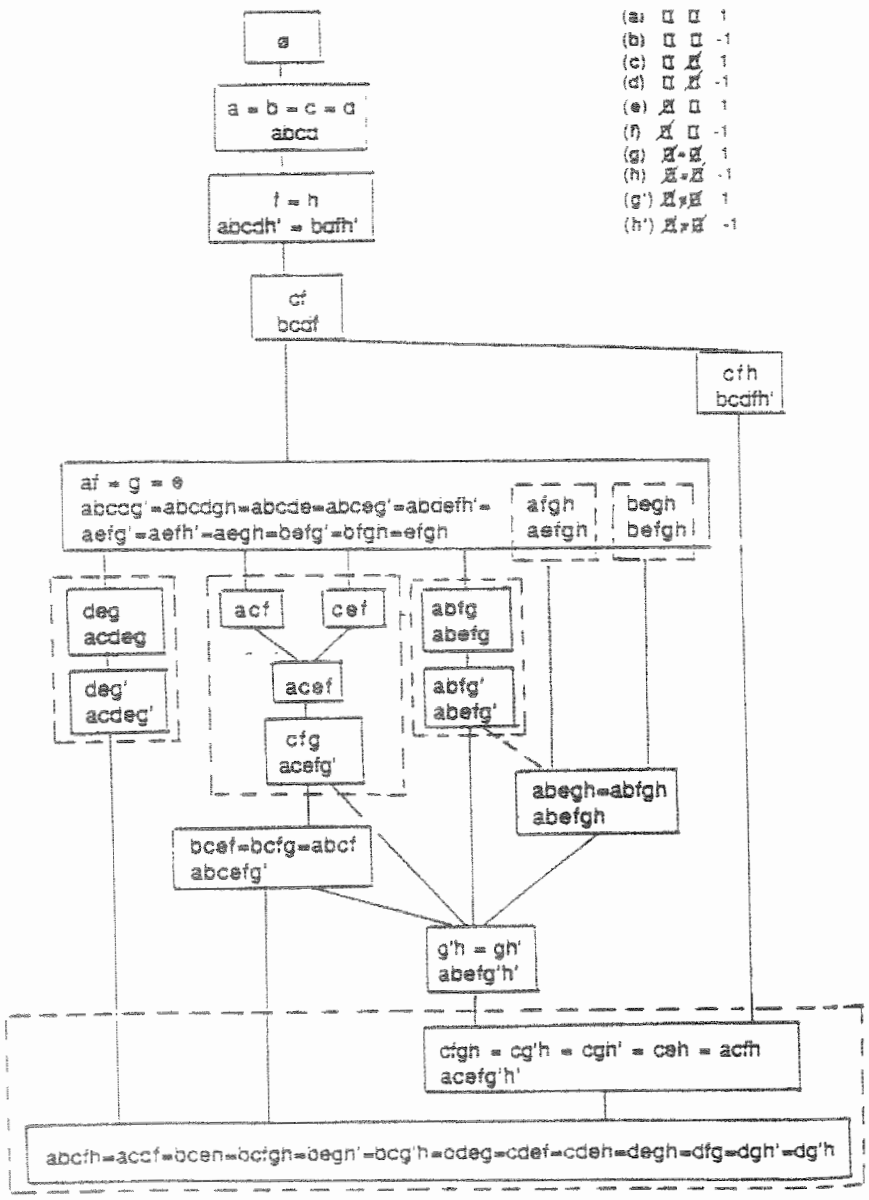
<Sa> Savitch, W.: Maze Recognizing Automata and Nondeterministic Tape Complexity. JCSS 7(4), 1973, 389-403.

E Hotzel

Nichtdeterminismus, Hilfssymbole und Akzeptanzbedingungen in der Umgebung der endlichen Automaten

Viele Automaten, bei denen Eingabe- und Arbeitsmedium zusammenfallen, lassen sich in natürlicher Weise als Ein-Band-Turingmaschinen beschreiben Eine solche Turingmaschine (TM) - nichtdeterministisch im allgemeinen - besteht 1) aus einer endlichen Menge Q von Zuständen mit einer ausgezeichneten Menge S von Anfangszuständen und einer ausgezeichneten Menge F von Endzuständen (zur Kennzeichnung akzeptierender Endsituationen), 2) einer endlichen Symbolmenge Γ mit einem ausgezeichneten Symbol $\Box$ (blank), und einer weiteren ausgezeichneten Teilmenge Σ, dem Eingabealphabet, 3) einer Menge von Rückanweisungen (wobei man sich oft auf die Elemente 1 = "nach rechts" und -1 = "nach links" beschränken kann) und 4) dem Turingprogramm, einer Menge von Tupeln der Form (q,x,x',d,q'), bestehend aus Zuständen q,q', Symbolen x,x' und einer Rückanweisung d

Man kann eine Klassifizierung dieser TM nach der Art der im Turing-programm auftretenden Befehle (q,x,x',d,q') vornehmen, im folgenden werden Befehle nur danach unterschieden, ob (a) $x=\Box$, $x'=\Box$, $d=1$, (b) $x=\Box$, $x'=\Box$, $d=-1$, (c) $x=\Box$, $x'\neq\Box$, $d=1$, (d) $x=\Box$, $x'\neq\Box$, $d=-1$, (e) $x\neq\Box$, $x'=\Box$, $d=1$, (f) $x\neq\Box$, $x'=\Box$, $d=-1$, (g) $x=x'\neq\Box$, $d=1$, (h) $x=x'\neq\Box$, $d=-1$, (g') $\Box\neq x\neq x'\neq\Box$, $d=1$ oder (h') $\Box\neq x\neq x'\neq\Box$, $d=-1$ gilt. Die Klasse aller TM, deren Befehle einer festen Auswahl dieser Bedingungen genügen, kann man als Automatentyp betrachten, z. B ist der Typ agh (d.h. jeder Befehl hat die Form (a) oder die Form (g) oder die Form (h)) eine Version des endlichen two-way-Automaten Verschiedene solche Automatentypen können unter Erkennungs-Äquivalenz zusammenfallen, d.h dieselbe Sprachenklasse festlegen (wie bekanntlich g und agh) Ähnliche Klassifikationen sind bisher von B. von Braunmühl für Zwei-Zähler-Automaten (1977) und von B. von Braunmühl und Hotzel für one-way-Automaten mit 1 Arbeitsband vorgenommen worden (1979) Dieselben Gesichtspunkte dienen bei Brandstädt als Basis von Komplexitätsuntersuchungen (1978) Der bisherige Stand der Klassifikation der Ein-Band-TM wird im umseitigen Bild angedeutet (die formal schwächsten und die formal stärksten unter Erkennungs-Äquivalenz zusammenfallenden Typen sind



jeweils in einem Kästchen zusammengefaßt, oben der Typ "leerer Automat", durch den genau die Sprachen der Form $\emptyset$ und Σ^* festgelegt werden, unten der allgemeine Typ TM, gestrichelte Umrandungen und Inklusionskanten beziehen sich auf den nichtdeterministischen Fall, sofern er vom deterministischen abweicht, weitere Inklusionsbeziehungen sind wahrscheinlich)

Die verschiedenen bekannten Formen des endlichen Automaten sind, wie die volle TM, erstaunlich unempfindlich gegenüber Determiniertheit oder Nichtdeterminiertheit, Verfügbarkeit von Hilfssymbolen (z.B als Begrenzungsmarken) und Akzeptanzfestlegungen (z.B. Akzeptanz eines Wortes bei beliebigem Stoppen, bei Durchgang durch einen Endzustand, bei Stop direkt vor oder nach dem Eingabewort usw.) Wie die folgenden Aussagen zeigen, gilt dies nicht mehr, wenn man andere Automatentypen betrachtet, die in der einen oder anderen Version zum endlichen Automaten äquivalent sind Als normale Akzeptanzbedingung wird hier Stoppen an beliebiger Stelle des Bandes (mangels passender Befehle) in einem Endzustand betrachtet (und dies für mindestens einen Lauf)

Die folgende Feststellung ist im Bild enthalten

(A) Jede von einem deterministischen aefgh oder befgh-Automat erkannte Sprache ist regulär

Jedoch werden die nichtregulären Sprachen

$$\{w_1\overleftarrow{w}_1cw_2\overleftarrow{w}_2c\ldots w_k\overleftarrow{w}_kc \mid k\geq 0,\ e_c(w_i)=w_i \text{ für } 1\leq i\leq k\} \text{ und}$$

$$\{cw_1\overleftarrow{w}_1cw_2\overleftarrow{w}_2\ldots cw_k\overleftarrow{w}_k \mid k\geq 0,\ e_c(w_i)=w_i \text{ für } 1\leq i\leq k\}$$

von afgh- bzw begh-Automaten erkannt. Hier sei e_c der durch $e_c(c) = \varepsilon$, $e_c(x) = x$ für $x \neq c$ gegebene Homomorphismus

(B) Jede bcdf-Sprache ist regulär und von der Form $K\cup R$, K endlich, $R=R\Sigma^*$, und umgekehrt.
(Zu jeder regulären Sprache $L \subseteq \Sigma^*$ gibt es einen bcdf-Automat, der L "mit Endmarke" erkennt, d h für passendes $c\notin\Sigma$ wird genau $Lc(\Sigma\cup c)^*$ erkannt.)

(C) Jede Sprache $L \subseteq \{a\}^*$, die von einem acegh-Automaten ohne Hilfssymbole erkannt wird (d h $\Gamma = \{\square,a\}$), ist regulär

Es läßt sich zeigen, daß die deterministischen acegh-Automaten unter den tally-Sprachen (d.h Σ einelementig, aber eventuell mit

Hilfssymbolen, d.h. $\Gamma \neq \Sigma \cup \{\Pi\}$) genau die rekursiven erkennen, im allgemeinen jedoch eine echte Zwischenklasse zwischen den rekursiven und den berechenbaren Sprachen (vgl. Bild).

Die Beweise von (A), (B), (C) beruhen auf Simulationen durch Automaten einfacheren Typs (g bzw. adgh), wobei Rechts-Links- bzw. Links-Rechts-Schleifen ausgelassen und die entsprechenden Zustandswechsel vorher bzw. nachträglich verifiziert werden.

Üblicherweise besteht die Menge der Anfangszustände S aus einem einzigen Element q_0 aus der Zustandsmenge Q, und es ist leicht zu sehen, daß man bei einigen Versionen des endlichen Automaten auf die Auszeichnung einer Menge F von Endzuständen verzichten kann (d.h. F = Q). Bei einigen Typen kann man sogar auf die Auszeichnung von Anfangszuständen verzichten.

(D) Jede reguläre Sprache wird von einem deterministischen bcgh- sowie von einem deterministischen cdgh-Automaten "ohne Anfangs- und Endzustände" (d.h. S = F = Q) erkannt.

Diese Automaten lassen sich mittels eines Hilfssymbols konstruieren, auf dem allein gestoppt werden kann. Man erkennt leicht, daß für die Erkennung einer regulären Sprache wie in (D) im allgemeinen ein Hilfssymbol erforderlich ist. Die Klasse der regulären Sprachen, die durch Automaten des einen oder anderen Typs ohne Hilfssymbole und ohne Anfangs- und Endzustände erkannt werden, ist noch nicht genau bekannt.

A. Brandstädt, On a family of complexity measures on Turing machines, defined by predicates, EIK 14, 1978, 331-339

B. von Braunmühl, Zwei-Zähler-Automaten mit gekoppelten Bewegungen, Berichte der Gesellschaft für Mathematik und Datenverarbeitung Nr. 116, 1977

B. von Braunmühl and E. Hotzel, Supercounter machines, Lecture Notes in Computer Science 71, 1979, 247-275 (ICALP 79)

Dr. Eckehart Hotzel
GMD
Postfach 1240
W-5205 St. Augustin 1

Erweiterung durch Reduktion - am Beispiel der kontextfreien Sprachen

Matthias Jantzen and Holger Petersen
Fachbereich Informatik
Universität Hamburg
Vogt-Kölln-Straße 30
D-2000 Hamburg 54

e-mail: jantzen @ rz . informatik . uni-hamburg . dbp . de

(Kurzfassung zum Vortrag auf dem ersten Theorietag in Magdeburg)

Schon frühzeitig wurden Kontrollmechanismen benutzt, um die Mächtigkeit kontextfreier Grammatiken zu erhöhen: Matrixgrammatiken, programmierte Grammatiken und, auf etwas andere Weise, Indexgrammatiken. Im Buch von Dassow und Paun, [4], finden sich alle wesentlichen Definitionen und Resultate. Daß nun auch *Reduktionen* die Sprachfamilie der kontextfreien Sprachen erweitern, war schon aus Arbeiten von Griffiths, 1968 [6], Stanat, 1971 [15], und Savitch, 1975 [13], bekannt. Recht viel später tauchten solche Reduktionen erneut auf: in anderem Zusammenhang bei Book et al., 1982 [1], Geffert, 1986 [5], Maurer et al., 1982 [12], und Brandenburg, 1989 [2, 3].

Ausgehend von den 'context-synchronized grammars' von Geffert, [5], untersuchten Jantzen et al., 1990 [10], die Dyck-Reduktionen von kontextfreien Sprachen mit dem vollständigen semi-Thue (Zeichenkettenersetzungs-) system $x\bar{x} \longrightarrow \lambda$, wobei x die einzige öffnende und $\bar{x}$ die schließende Klammer ist. In [6] und [13] war gezeigt worden, daß Dyck-Reduktionen mit zwei Klammerpaaren ausreichen, um jede aufzählbare Menge aus den kontextfreien Sprachen zu generieren. Dieses Ergebnis wurde von Brandenburg verschärft, der in [2] zeigte, daß lineare kontextfreie Sprachen und symmetrische Dyck-Sprachen mit zwei Klammerpaaren dazu ausreichen. Wir werden zeigen, welche Ergebnisse man erhält, wenn nur noch ein einziges Klammerpaar zugelassen ist.

Nach den nötigsten, unvermeidbaren Definitionen schildern wir im Folgenden nur kurz die neuen Ergebnisse.

Die noch in [10] von uns benutzte Notation unter Verwendung des vollständigen semi-Thue Systems: $x\bar{x} \longrightarrow \lambda$ ist leider nur für Dyck-Reduktionen tauglich. Da wir aber auch mit anderen als nur Dyck-Sprachen reduzieren wollen, bietet sich die von Kimura, [11], verwendete Operation % an. Mit ihr wird das vollständige Löschen von Teilworten einer anderen Menge notiert.

<u>Definition</u> 1:

Sei $V_k := \{x_i, \bar{x}_i \mid 1 \le i \le k\}$ ein Alphabet von Klammersymbolen, wobei x_i die öffnende und $\bar{x}_i$ die schließende Klammer ist.

Für Sprachen $K \subseteq V_1^*$ und $L \subseteq V_2^*$ über nicht notwendig disjunkten Alphabeten sei:

$K \% L :=$

$\{w \in (V_1 \setminus V_2)^* \mid \exists w_1 \in K\ \exists n \in \mathbb{N}\ \forall i \le n\ \exists v_i \in V_1^*\ \forall 0 \le i \le n\ \exists u_i \in L : w_1 = u_0 v_1 u_1 \cdots v_n u_n \wedge w = v_1 v_2 \cdots v_n\}$

$K \% L$ wird dann <u>Dyck$_k$-Reduktion</u> genannt, wenn L eine Dyck-Sprache über k Klammerpaaren ist

Der französischen Schreibweise folgend notieren wir die <u>einseitige</u> (oder semi-) <u>Dyck-Sprache</u> über k Klammerpaaren als

$$D_k^{*} := \{ w \in V_k^{*} \mid \forall u \in V_k^{*}\ \forall i \le k : w = uv \text{ impliziert } |u|_{x_i} \ge |u|_{\bar{x}_i} \text{ und } |w|_{x_i} = |w|_{\bar{x}_i} \}$$

Hierbei bezeichnet $|w|_x$ die Anzahl des Vorkommens des Symbols x in der Zeichenkette w

$$D_k^{=} := \{ w \in V_k^{*} \mid \forall i \le k : |w|_{x_i} = |w|_{\bar{x}_i} \}$$

bezeichne die <u>symmetrische</u> <u>Dyck-Sprache</u> und es sei für jedes $i \le k$ folgende lineare, kontextfreie Sprache definiert:

$$\mathrm{EQUAL}_i := \{ x_i^n\, \bar{x}_i^n \mid n \in \mathbb{N} \}$$

Mit $\mathcal{R}eg$ (bzw $\mathcal{L}in$, $\mathcal{C}f$, $\mathcal{R}ec$, $\mathcal{R}e$) seien die bekannten Sprachfamilien der regulären (bzw linearen, kontextfreien, entscheidbaren, aufzählbaren) Mengen notiert

Die folgenden Ergebnisse sind zum Teil veröffentlicht oder noch nicht einmal eingereicht. Wir geben hier nur kurz die Ergebnisse ohne Beweis bekannt und diskutieren begonnene und geplante weitere Untersuchungen.

Die Sprachklassen $\mathcal{C}f \,\%\, D_k^{=}$ wie auch $\mathcal{C}f \,\%\, D_k^{*}$ enthalten beide die Durchschnitte von beliebigen kontextfreien Sprachen, mehr noch, sogar jede Sprache L aus $(\mathcal{C}f \wedge \mathcal{C}f)^{\nabla}$, dem algebraischen Abschluß dieser Familie von Durchschnitten.

<u>Satz</u> 1.

(a) $(\mathcal{C}f \wedge \mathcal{C}f)^{\nabla} \subseteq \mathcal{C}f \,\%\, D_k^{*}$

(b) $(\mathcal{C}f \wedge \mathcal{C}f)^{\nabla} \subseteq \mathcal{C}f \,\%\, D_k^{=}$

(c) $(\mathcal{C}f \wedge \mathcal{C}f)^{\nabla} \subseteq \mathcal{C}f \,\%\, \mathrm{EQUAL}_1$

Wiewohl $\mathrm{EQUAL}_k \subseteq D_k^{=} \subseteq D_k^{*}$ offensichtlich ist, mußte folgendes noch detailliert bewiesen werden:

<u>Satz</u> 2:

(a) $\mathcal{C}f \,\%\, D_1^{*} \subseteq \mathcal{C}f \,\%\, \mathrm{EQUAL}_1$

(b) $\mathcal{C}f \,\%\, D_1^{=} \subseteq \mathcal{C}f \,\%\, \mathrm{EQUAL}_1$

Die Beziehung zwischen den Familien $\mathcal{C}f \,\%\, D_k^{*}$ und $\mathcal{C}f \,\%\, D_k^{=}$ ist noch unklar, denn weder Gleichheit, noch die vermutete Inklusion $\mathcal{C}f \,\%\, D_k^{=} \subseteq \mathcal{C}f \,\%\, D_k^{*}$ konnten bewiesen werden. Diese Vermutung wird gespeist aus der Tatsache, daß die erste Familie beweisbar nur entscheidbare Sprachen enthält, während dies bei der zweiten Familie immer noch nicht bewiesen oder widerlegt werden konnte. Die Familie $\mathcal{C}f \,\%\, \mathrm{EQUAL}_1$ enthält jedoch nicht mehr nur entscheidbare Mengen:

<u>Satz</u> 3:

(a) $\mathcal{C}f \,\%\, D_k^{=} \subsetneq \mathcal{R}ec$

(b) $\mathcal{C}f \,\%\, \mathrm{EQUAL}_1 = \mathcal{R}e$

Aus Satz 3 folgt natürlich Satz 2 (b) sofort Satz 3 (b) ließ sich aber noch deutlich verschärfen und ergänzt so die Resultate aus [2, 3]:

<u>Satz</u> 4:

$$\mathcal{L}in \,\%\, \mathrm{EQUAL}_1 = \mathcal{R}e$$

Schon die einfachsten Dyck_1-Reduktionen mit der symmetrischen Dyck-Sprache ist so mächtig, daß das <u>Leerheitsproblem</u> der Familie $\mathcal{L}in \,\%\, D_1^{=}$ nicht mehr entscheidbar ist. Dies war eines der ersten Ergebnisse in [10].

In [9] konnten wir zeigen, daß jede terminale Petrinetzsprache der Familie $\mathcal{L}_0 = \mathcal{M}_{\cap}(D_1^{=})$ durch Dyck-Reduktion aus den linearen Sprachen gewonnen werden kann:

<u>Satz</u> 5:

(a) $\mathcal{L}_0 \subseteq \mathcal{L}in \,\%\, D_1^{*}$

(b) $\mathcal{L}_0 \subseteq \mathcal{L}in \,\%\, D_1^{=}$

(c) $(\mathcal{C}f \wedge \mathcal{L}_0)^{\nabla} \subseteq \mathcal{C}f \,\%\, D_k^{*}$

(d) $(\mathcal{C}f \wedge \mathcal{L}_0)^{\nabla} \subseteq \mathcal{C}f \,\%\, D_k^{=}$

Von Savitch, [13, 14], wissen wir, daß jede EOL-Sprache als Dyck_1-Reduktion aus den kontextfreien Sprachen erhalten werden kann. Beginnen wir jedoch bei den ETOL-Sprachen mit der Reduktion, so erzeugen wir wieder alle aufzählbaren Mengen:

<u>Satz</u> 6:

(a) $\underline{\mathrm{ETOL}} \,\%\, D_k^{=} = \mathcal{R}e$

(b) $\underline{\mathrm{ETOL}} \,\%\, D_k^{*} = \mathcal{R}e$

Über Dyck_1-Reduktionen von EOL- oder Petrinetz-Sprachen sind uns noch nicht genügend Ergebnisse bekannt.

References

[1] R. V Book, M. Jantzen, C. Wrathall: Monadic Thue system. Theoret. Comput. Sci. 19 (1982) 231-251

[2] F. J. Brandenburg, : Cancellations in linear context-free languages. Techn. report MIP-8904, Univ Passau (1989).

[3] F. J. Brandenburg, J. Dassow: Reductions of picture words. Techn. report MIP-8905, Univ. Passau (1989).

[4] J. Dassow, G. Paun: Regulated Rewriting in Formal Language Theory, EATCS Monographs, 18 Springer-Verlag (1989).

[5] V Geffert: Grammars with context dependency restricted to synchronization. Lecture Notes in Comput. Sci. vol. 233, Springer-Verlag (1986) 370 - 378.

[6] T.V. Griffiths: Some remarks on derivations in general rewriting systems. Information and Control 12 (1968) 27-45.

[7] M. Hack: Petri net languages. C. S. G. Memo 124, Project MAC, MIT (1975).

[8] M. Jantzen, H. Petersen: Petri net languages and one-sided $Dyck_1$-reductions of context-free set, in K. Voss, H. Genrich, G. Rozenberg (eds): Concurrency and nets, Springer-Verlag (1987) 245 - 252.

[9] M. Jantzen, H. Petersen: Twisting Petri net languages and how to obtain them by reducing linear context-free sets. In: Proc. 12th Intern. Conf. on Petri nets, Gjern, (1991) 228-236.

[10] M. Jantzen, M. Kudlek, K.-L. Lange, H. Petersen: $Dyck_1$-reductions of context-free languages. Computers and Artificial Intelligence, 9 (1990) 3 - 18.

[11] T. Kimura: Formal description of communication behaviour. Proc. Johns Hopkins Conf. on Information Sciences and Systems (1979).

[12] H.R. Maurer, G. Rozenberg, E. Welzel; Using string languages to describe picture languages,. Inform. Control, 54 (1982) 155-185.

[13] W. J. Savitch: Some characterizations of Lindenmayer systems in terms of Chomsky-type grammars and stack machines, Inform. Contr, 27 (1975) 37-60.

[14] W. J. Savitch: Parenthesis grammars and Lindenmayer sytems. In: G. Rozenberg - A. Salomaa (eds.): The Book of L. Springer-Verlag (1986) 403 - 411.

[15] D Stanat: Formal languages and power series 3rd ACM Sympos. Theory of Computing (1971) 1 - 11

Komplexität und Formale Sprachen

Klaus-Jörn Lange, Technische Universität München

Untersuchungsgebiet sind die vielfältigen Verbindungen zwischen der Theorie der Formalen Sprachen und der Komplexitätstheorie. So führt die Anwendung formalsprachlicher Methoden und Begriffe innerhalb der Komplexitätstheorie zu einer Fülle von Charakterisierungen und Vollständigkeitsresultaten. Auf der anderen Seite ermöglicht die komplexitätstheoretische Betrachtung von Entscheidbarkeitsresultaten eine feinere Einordnung von Klassen und Problemen.

Der Vortrag gliedert sich in zwei Teile. Zunächst wird ein Überblick über die sequentielle Komplexität von Wort- und Leerheitsproblemen sowie von formalsprachlichen Operationen gegeben. Im Rahmen der parallelen Komplexitätstheorie werden die wichtigsten Modelle und ihre Beziehungen zu sequentiellen Klassen beleuchtet. Abschließend werden einige neuere Anwendungen formalsprachlicher Methoden in diesem Bereich vorgestellt.

Sequentielle Komplexitätstheorie: Ausgehend von den Kontextfreien Sprachen werden einige typische Eigenschaften formaler Sprachklassen dargestellt. Während sich rein formalsprachliche Beziehungen generell im Rahmen der AFA- und Balloon-Automata-Theorie leicht verallgemeinern lassen, scheint dies bezüglich komplexitätstheoretischer Beziehungen erst mit Hilfe der Engelfriet'schen *Automaten mit abstraktem Speicher* in befriedigender Weise möglich zu sein. So konnten komplexitätsmäßige Beziehungen zwischen Leerheits- und Wortproblemen bei kontextfreien Sprachen allgemein für Automaten mit abstraktem Speicher hergeleitet werden. Zur Verallgemeinerung von Charakterisierungen des Wortproblems durch *härteste* Sprachen scheinen jedoch zusätzliche Annahmen an das zugrundelegende Speichermodell notwendig zu sein. Die eng damit zusammenhängende Frage nach vollständigen Problemen führt direkt auf die Frage nach der Komplexität formalsprachlicher Operationen, also auf die Frage nach Abschlußeigenschaften von Komplexitätsklassen

Parallele Komplexitätstheorie: Die wohl wichtigsten Modelle der parallelen Komplexitätstheorie sind die parallelen Registermaschinen (PRAMs) und die Boole'schen Schaltnetze. Angesichts der großen Unterschiedlichkeit dieser Modelle weisen die erzielten Ergebnisse eine überraschende Einheitlichkeit auf. Dieses sowie die engen Beziehungen zur sequentiellen Komplexitätstheorie führten zur Formulierung der *Parallelen Berechnungshypothese*. Die Bemühungen zur Klassifikation effizienter Parallelisierbarkeit führten zur Betrachtung der Klasse NC, die sich durch ihre grosse Robustheit gegenüber Modellabwandlungen auszeichnet. Bei Verfeinerung zerfällt diese recht grobe Klasse in mehrere, ineinander verzahnte Hierarchien, die durch ihre wechselseitigen Entsprechungen aufschlußreiche Charakterisierungen verschiedener paralleler Modelle liefern. Leider blieb im Zuge dieser Untersuchungen das häufig benutzte Konzept des *exklusiven* Zugriffs auf gemeinsam benutzte Speicher isoliert. Dieses Problem konnte jüngst unter Verwendung des formalsprachlichen Begriffes der Eindeutigkeit gelöst werden. Die dabei gewonnenen Ergebnisse legen den Schluss nahe, Eindeutigkeit verhaltensmäßig näher bei Nichtdeterminismus, leistungsmäßig jedoch näher bei Determinismus einzuordnen. Ein anderer formalsprachlicher Begriff mit Anwendungen in der Komplexitätstheorie ist die *Datenunabhängigkeit (Obliviousness)*. Hiermit scheint eine Charakterisierung derjenigen Problem möglich zu sein, die sich effizient auf existierenden Parallelrechnern mit verteilten Speichern lösen lassen

Quantorenhierarchien über Automaten-definierbaren Relationen

S. Seibert
Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
W-2300 Kiel 1
e-mail ss@informatik.uni-kiel.dbp.de

Wir befassen uns mit der Einstufung von Relationen (über endlichen Worten) mit Hilfe von alternierenden Quantoren-Blöcken nach dem Muster der arithmetischen Hierarchie (gemäß z.B. [Hi 78])

Die Klasse RCS der rekursiven Relationen – als „Kern" der arithmetischen Hierarchie – läßt sich als Klasse aller Relationen auffassen, die von einer Turing-Maschine entschieden werden. Einer Idee aus [Th 89] folgend untersuchen wir hier Hierarchien, die sich ergeben, indem wir in der Konstruktion der arithmetischen Hierarchie RCS durch andere Klassen C ersetzen, definiert durch Automatenmodelle, die man als Spezialisierung von Turing-Maschinen ansehen kann. Wir sprechen dann von der „arithmetischen Hierarchie über C" oder der „C-Hierarchie".

Betrachtet werden verschiedene Modelle von endlichen Automaten und Kellerautomaten. Die Erweiterung dieser Automatenmodelle auf mehrere Bänder – zum Bearbeiten von Wort-Tupeln statt einzelner Wörter – zeigt, daß wir zwischen asynchronen und synchronen Versionen unterscheiden müssen, wobei die letzteren eine Einschränkung in der Hinsicht darstellen, daß sie ihre Leseköpfe nur im Gleichschritt auf der Eingabe bewegen dürfen. Für jedes dieser Modelle gibt es eine deterministische und eine nichtdeterministische Version. Diese beiden Versionen fallen nur bei den synchronen endlichen Automaten zur Klasse SRC (synchronous recognizable) zusammen, welches zur Folge hat, daß alle Klassen der SRC-Hierarchie mit ihrem Kern übereinstimmen. Die nichtdeterministischen (bzw. deterministischen) asynchronen endlichen Automaten erkennen gerade die Klasse RAT (bzw. DRAT) der rationalen (bzw. deterministisch rationalen) Relationen (vgl. [Be 79, EM 65, FR 68, RS 59]). Wir definieren darüberhinaus die Klassen der durch synchrone bzw. asynchrone, deterministische bzw. nichtderministische Kellerautomaten (pushdown automata) erkannten Relationen als DPS, NPS, DPA und NPA.

Für diese sechs Relationenklassen ergeben sich unendliche Hierarchien nach dem folgenden Muster, wobei jedes der Paare (DRAT, RAT), (DPS, NPS) und (DPA, NPA) für (D, N) eingesetzt werden kann.

Satz:

Wenn (D, N) ein Paar von Relationenklassen ist, sodaß D unter Komplement abgeschlossen ist (d.h. D=co-D), $N=\sum_1^0(D)$ und die D-Hierarchie (oder äquivalent die N-Hierarchie) unendlich ist, so lassen sich die D- und N-Hierarchie gemeinsam in folgendem Diagramm darstellen, wobei jede Linie für eine echte Inklusion steht:

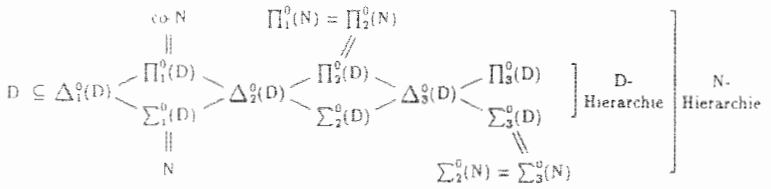


Abbildung 1

□

Dabei ist nur die Echtheit der Inklusion $D \subseteq \Delta_1^0(D)$ offen. Für die o.a. Paare von Klassen ist sie echt, für das Paar (RCS, R.E.), das auch für (D, N) eingesetzt werden kann, gilt bekanntlich $RCS = \Delta_1^0(RCS)$. (R.E. ist die Klasse der rekursiv aufzählbaren Relationen.)

Die Unendlichkeit der arithmetischen Hierarchien über DRAT, DPS und DPA folgt aus unserem Hauptergebnis, welches besagt, daß diese Hierarchien alle die RCS-Hierarchie ausschöpfen, genauer gesagt kann man sie als Verlängerung dieser nach unten bezeichnen.

Satz:

(a). $RCS = \Delta_2^0(DRAT) = \Delta_2^0(DPS) = \Delta_2^0(DPA)$,

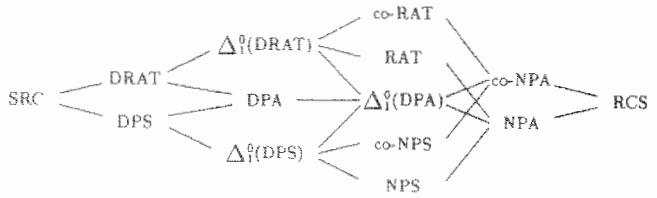
(b). $R.E. = \sum_2^0(DRAT) = \sum_2^0(DPS) = \sum_2^0(DPA)$

□

Dies hat zur Folge, daß $\Delta_n^0(RCS) = \Delta_{n+1}^0(DRAT) = \Delta_{n+1}^0(DPS) = \Delta_{n+1}^0(DPA)$ für alle n aus $\mathbb{N}$ gilt (und analog für die Klassen $\sum_n^0()$ und $\prod_n^0()$). Unterhalb von RCS fallen die betrachteten Hierarchien jedoch nicht zusammen. Wir erhalten die folgende vollständige Übersicht über die Teilmengenbeziehungen derjenigen Klassen der neuen Hierarchien, die unterhalb von RCS und damit nicht in der üblichen arithmetischen Hierarchie liegen.

Satz:

Im folgenden Diagramm steht jede Linie für eine echte Inklusion; außerdem sind je zwei Klassen hinsichtlich der Teilmengenbeziehung unvergleichbar, wenn sie in dem Diagramm nicht durch einen Pfad von links nach rechts verbunden sind.



Dieses Ergebnis beruht wesentlich auf dem Nachweis, daß bestimmte Relationen die Striktheit der Inklusionen bzw. die Unvergleichbarkeit belegen. Im einzelnen gilt:

- Die Suffix-Relation ($Suf(x, y) :\Leftrightarrow \exists z(y = zx)$) ist in $\Delta_1^0(DRAT) \setminus DPA$,
- $\{ww^R | w \in \Sigma^*\} \in \Delta_1^0(DPS) \setminus DPA$,
- $\{a^i b^i | i \in \mathbb{N}\} \in DPS \setminus RAT$ und
- $\{(a^i, a^{2i}, a^{3i}) | i \in \mathbb{N}\} \in DRAT \setminus NPS$.

□

Wir haben damit gesehen, wie wir „Verlängerungen der arithmetischen Hierarchie nach unten" erhalten, indem wir bekannte Automatenmodelle zur Charakterisierung von Relationen verwenden. Ein anderer Ansatz, der aus der Theorie formaler Sprachen motiviert wird, besteht darin als Kern einer weiteren Hierarchie die Klasse aller Relationen zu nehmen, die sich durch eine Boolesche Kombination von „Konkatenations-Gleichungen" beschreiben läßt. (Das sind Gleichungen, deren beide Seiten jeweils Verkettungen von Worten und Wort-Variablen sind.) In [Se 91] (ausfürlich: [Se 90]) wird gezeigt, daß auch die Hierarchie über dieser Klasse eine „Verlängerung der arithmetischen Hierarchie nach unten" ist, und daß sie unterhalb von RCS zu den hier betrachteten Hierarchien unvergleichbar ist. In diesen beiden Arbeiten sind auch Beweise zu den hier aufgeführten Ergebnissen angegeben.

Literatur

[Be 79] J.Berstel: Transductions and Context-free Languages, Teubner, Stuttgart 1979

[EM 65] C.C.Elgot, G.Mezei: On Relations Defined by Generalized Finite Automata, IBM J. of Res. and Dev. 9 (1965), S. 47-68

[FR 68] P.C.Fischer, A.L.Rosenberg: Multitape One-Way Nonwriting Automata, J. of Computer and System Sci. 2 (1968), S. 88-101

[Hi 78] P.G.Hinman: Recursion Theoretic Hierarchies, Springer, Berlin-Heidelberg New York 1978

[RS 59] M.O.Rabin, D.Scott: Finite Automata and their Decision Problems, IBM J of Res and Dev 3 (1959), S. 114-125 (nachgedruckt in: Sequential Machines: Selected Papers, (E.Moore, Hrsg.), Addison-Wesley, Reading, Mass. 1965, S. 63-91)

[Se 90] S.Seibert: Quantorenhierarchien über sprachtheoretischen Relationenklassen. Diplomarbeit, RWTH Aachen 1990

[Se 91] S.Seibert: Quantifier Hierarchies over Word Relations, in: CSL '91 proceedings, erscheint als LNCS-Band

[Th 89] W.Thomas: Automata and Quantifier Hierarchies, in: Formal Properties of Finite Automata and Applications (J.E. Pin, Hrsg.), LNCS 386 (1989), S. 104-119

Synthesized and inherited functions on trees - a new computational model for syntax-directed semantics

Heiko Vogler
Abt. Theoretische Informatik
Universität Ulm

Abstract: The concept of syntax-directed semantics is a well-accepted principle to describe the meaning of a tree-strutured object. In such a description, the meaning of a node of a given tree is described in a local fashion. More precisely, there is a finite set of meaning names which can be partitioned into synthesized meaning names and inherited meaning names. The two types of meaning names differ in the way in which their semantic values can be described, as follows. Consider a tree t and a node n of t If s is a synthesized meaning name, then the semantic value of s can be described in terms of inherited meaning names of n and synthesized meaning names of n's sons. If i is an inherited meaning name, then the semantic value of i can be described in terms of synthesized meaning names of n or of brothers of n and inherited meaning names of the father of n.

The concept of syntax-directed semantics was first formalized by E.T Irons [Iro61] in 1961 Since then many metalanguages have been considered as a formalization of the concept, e.g., attribute grammars [Knu68], top-down tree transducers [Rou70, Tha70], generalized sd translation schemes [AU71, AU73], denotational semantics [SS71, Gor79], affix grammars [Kos71], macro tree transducers [Eng80, Cou81], attributed tree transducers [Fül81], graph translation schemes [Son87], attribute coupled grammars [Gie88], context-free hypergraph-based sd translation schemes [EH89], and top-down tree-to-graph transducers [EV91]. A detailled study of the concept of sd semantics is given in [Eng81] where special attention is paid to the description and comparison of attribute grammars and macro tree transducers.

In order to compare the expressive (or computational) power of the various formalizations, one has to fix one semantic domain. Here we consider the free term algebra which on the one hand is a special domain, but on the other hand it is also a general domain, because every concrete semantics can be obtained by applying the unique intial homomorphism to the semantic value in the free term algebra.

In this paper we introduce a new formalization of the concept of sd semantics, called *macro attributed tree transducer* (for short: matt tree transducer) which integrates the features of (noncircular) attributed tree transducers and macro tree transducers. Let us briefly recall these two types of transducers. In an attribute tree transducer, meanings names are called attributes; attributes have trees as semantic values. Typical equations for the description of a synthesized attribute s and an inherited attribute $\imath$ are the following two, where σ, δ, and γ are basic symbols of rank 2, 2, and 1, respectively:

$$(s,\varepsilon)(\sigma(x_1,x_2)) = \delta((\imath,\varepsilon)(\sigma(x_1,x_2)),\gamma((s,2)(\sigma(x_1,x_2))))$$

$$(\imath,2)(\sigma(x_1,x_2)) = \sigma((\imath,\varepsilon)(\sigma(x_1,x_2)),(s,1)(\sigma(x_1,x_2))).$$

Clearly, in order to serve as computational model for syntax-directed semantics, such systems of equations have to be deterministic and complete in an obvious sense. In a macro tree transducer, there are no inherited meaning names. The synthesized meaning names are called states. A state represents a tree function, i.e., a function with trees as arguments and as result. Hence, since we consider the free term algebra as the only semantic domain, states themselves do not have a semantic value. Rather in every instance of a syntax-directed semantics description they occur in applied form. A typical example of the description of a state q is the following:

$$(q,\varepsilon)(\sigma(x_1,x_2),y_1,y_2) = \delta(y_2,\sigma((q',2)(\sigma(x_1,x_2),\gamma((q'',2)(\sigma(x_1,x_2)))),y_1))$$

where q, q', and q'' represent functions with arity 3, 2, and 1, respectively, and y_1 and y_2 are formal parameters of state q. Obviously, every occurrence of a state in the left-hand side and in the right-hand side is applied to the appropriate number of arguments such that to every subtree of both sides a semantic value in the free tree algebra can be associated.

As a consequence of this discussion, we can define another partitioning of meaning names which is orthogonal to the partitioning into synthesized and inherited meaning names: a meaning name can either represent a tree or a function. In the first case we call the meaning name an attribute, in the second case a function. Using this terminology, an attributed tree transducers uses synthesized and inherited attributes (as in the old terminology), and a macro tree transducer uses synthesized functions.

Now it is easy to describe the new formalization: a macro attribute tree transducer uses synthesized and inherited functions. A typical equation which describes an inherited function $\imath$ of arity 3 looks as follows:

$$(\imath,2)(\sigma(x_1,x_2),y_1,y_2) = \delta(y_2,\sigma((q',2)(\sigma(x_1,x_2),\gamma((\imath',\varepsilon)(\sigma(x_1,x_2)))),y_1))$$

where $\imath'$ is an inherited function with arity 1, and q' is a synthesized function with arity 2.

In order to have a uniform nomenclatura for tree transducers, we introduce the notion of X-tree transducer, where $X \in \{s, s_f, \imath, \imath_f\}$ and the modifiers s, s_f, $\imath$, $\imath_f$ means that the tree transducer may use synthesized attributes, synthesized functions, inherited attributes, and inherited functions, respectively. Thus, e.g., an an $s\imath$-tree transducer is an attributed tree transducer. The classes of tree functions computed by X-tree transducers are denoted by the corresponding capitals of the modifiers followed by a T, thus, e.g., SIT denotes the class of tree functions computed by $s\imath$-tree transducers, respectively. If we consider unary tree functions only, then the corresponding denotation of tree function classes is proceeded by a 'u'.

We investigate the relationships between the classes of tree functions which are computed by X-tree transducers. The following results are known:

- $ST \subset SIT \subset S_fT$
- $uSIT^n \subset uS_fT^n \subset uSIT^{n+1}$
- $S_fT = HOM \circ SIT$ where HOM denotes the class of tree homomorphisms

New results are:

- $S_fI_fT = SIT \circ SIT$, i.e. macro attributed tree transducers are equivalent to the two-fold composition of attributed tree transducers.
- $S_fI_fT \circ ST \subseteq S_fI_fT$, i.e., macro attribute tree transducers are closed under right composition with top-down tree transducers.
- $S_fIT = S_fT$, i.e., in the presence of synthesized functions, the addition of inherited attributes does not increase the computational power.
- $S_fI_fT(PREC) \subseteq PREC$, i.e., the class $PREC$ of primitive recursive tree functions is closed under s_fi_f-tree transducers (cf. [FHVV90] for the concept of tree transducers as operators on classes of functions).

We conjecture that the classes uS_fIT and uSI_fT are incomparable.

References

[AU71] A.V Aho and J.D. Ullman. Translations on a context free grammar. *Inform. and Control*, 19:439–475, 1971.

[AU73] A.V Aho and J.D. Ullman. *The Theory of Parsing, Translation and Compiling, Vol. I and Vol. II.* Prentice-Hall, 1973.

[Cou81] B. Courcelle. Attribute grammars: Theory and applications. In J. Dias and I. Ramos, editors, *Proc. Colloq. on Formalization of Programming Concepts*, pages 75–95. Springer-Verlag, Berlin, 1981.

[EH89] J. Engelfriet and L. Heyker. The term-generating power of context-free hypergraph grammars and attribute grammars. Technical Report 89-17, Rijksuniversiteit te Leiden, 1989.

[Eng80] J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. In R.V. Book, editor, *Formal language theory; perspectives and open problems.* New York, Academic Press, 1980.

[Eng81] J Engelfriet. Tree transducers and syntax directed semantics. Technical Report Memorandum 363, Technische Hogeschool Twente, 1981.

[EV91] J Engelfriet and H. Vogler. The translation power of top-down tree-to-graph transducers. manuscript 1991, 1991

[FHVV90] Z. Fülöp, F Herrmann, S. Vagvölgyi, and H. Vogler. Tree transducers with external functions. Technical Report 27, Aachen University of Technology, Fachgruppe Informatik, Ahornstr. 55, W-5100 Aachen, FRG, 1990. to appear in Theoret. Comput. Sci.

[Fül81] Z. Fülöp. On attributed tree transducers. *Acta Cybernetica*, 5:261–279, 1981

[Gie88] R. Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Informatica*, 25:355–423, 1988.

[Gor79] M.J.C. Gordon. *The denotational description of programming languages; an introduction.* Springer-Verlag, 1979.

[Iro61] E.T Irons. A syntax directed compiler for ALGOL 60. *Comm. Assoc. Comput. Mach.*, 4:51–55, 1961.

[Knu68] D.E. Knuth. Semantics of context-free languages. *Math. Syst. Theory*, 2:127–145, 1968.

[Kos71] C.H.A Koster Affix grammars. In *Proc. of the IFIP working conf. on ALGOL68 implementation.* North-Holland, Amsterdam, 1971.

[Rou70] W.C. Rounds. Mappings and grammars on trees. *Math. Syst. Theory*, 4:257–287, 1970.

[Son87] M. Sonnenschein. Graph translation schemes to generate compiler parts. *ACM Trans. on Progr. Languages*, 9:473–490, 1987.

[SS71] D. Scott and C. Strachey. Toward a mathematical semantics for computer languages. In J. Fox, editor, *Computers and automata*, pages 19–46. Wiley, New York, 1971

[Tha70] J.W. Thatcher. Generalized 2 sequential machine maps. *J. Comput. Syst. Sci.*, 4:339–367, 1970.

Einige Bemerkungen über Matrixbildsprachen

Ralf Stiebe
Technische Universität Magdeburg
Fakultät für Mathematik

Die Bilderzeugung mit Hilfe von Grammatiken wurde in den vergangenen Jahren intensiv erforscht Sehr bekannt sind die Kettenkodesprachen, die von Freeman [Fr] eingeführt wurden
In diesem Beitrag wird ein anderes Modell betrachtet, das der Siromoney-Matrixsprachen [Si]
Eine Matrixgrammatik ist ein Paar (G_1,G_2), wobei

$G_1 = (N_1,T_1,P_1,S_0)$, $T_1 = \{S_1, \quad ,S_k\}$ eine Grammatik ist und

$G_2 = \bigcup_{i=1}^{k} G_{2i}$, $G_{2i} = (N_{2i},T_2,P_{2i},S_i)$, $i = 1, \ldots,k$,

ein System von regulären Grammatiken ist Um eine Matrix zu erzeugen, wird zunächst mittels G_1 ein horizontales Wort über T_1 generiert. Anschließend werden mit Hilfe der Grammatiken aus G_2 (parallel) aus allen Buchstaben dieses Wortes "vertikale" Wörter gleicher Länge erzeugt

Beispiel: $G_1 = (\{S_0,A\},\{S_1,S_2\},\{S_0 \to S_1A, A \to S_2A, A \to S_1\},S_0)$,

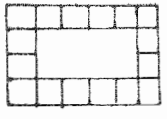
$G_{21} = (\{S_1\},\{a,b\},\{S_1 \to aS_1, S_1 \to a\},S_1)$,

$G_{22} = (\{S_2,B\},\{a,b\},\{S_2 \to aB, S_2 \to bB, B \to a\},S_2)$

Ableitung. $S \overset{*}{\Rightarrow} S_1S_2S_2S_2S_2S_1 \Downarrow$

```
aaaaaa
abbbba
abbbba
abbbba
```

Die Menge aller aus einer Matrixgrammatik erzeugbaren Matrizen heißt Matrixsprache Ist G_1 regulär (kontextfrei), so heißt die dazugehörige Sprache REG-RM-Sprache (CF-RM-Sprache)
Um einer Matrix ein Bild zuzuordnen, wird jeder Buchstabe der Matrix eindeutig durch ein Teilbild gegebener Größe $p \times q$ ersetzt. Als Bild bezeichnen wir dabei eine endliche Menge von horizontalen und vertikalen Einheitslinien
Ist z.B. pic(a) =  und pic(b) =  ,
so ist das zur obigen Matrix gehörende Bild



Die Menge Pic(G,pic) aller von der Matrixgrammatik G und der Abbildung pic erzeugten Bilder heißt Matrixbildsprache

Im weiteren untersuchen wir folgende Fragestellungen:

- Vergleich der Klassen der Matrixbildsprachen und der erweiterten Kettenkodebildsprachen,
- Abschlußeigenschaften der Matrixbildsprachen,
- Entscheidungsprobleme für Matrixbildsprachen.

<u>Satz 1.</u> Es seien $\mathcal{B}$(CF) bzw. $\mathcal{B}$(REG) die Familien der kontextfreien bzw regulären Kettenkodebildsprachen sowie $\mathcal{B}$(CF-RM) bzw $\mathcal{B}$(REG-RM) die Familien der kontextfreien bzw regulären RM-Sprachen. Dann gilt

(i) $\mathcal{B}(\text{REG-RM}) \not\subseteq \mathcal{B}(\text{CF})$,

(ii) $\mathcal{B}(\text{REG}) \not\subseteq \mathcal{B}(\text{CF-RM})$.

Ein Beispiel für (i) ist die oben erwähnte Matrixbildsprache. Ein Beispiel für (ii) ist die zu $\{(rd)^n \quad n > 0\}$ gehörige Kettenkodebildsprache.

Nachdem die Unvergleichbarkeit von Matrix- und Kettenkodebildsprachen gezeigt wurde, sollen nun Abschluß- und Entscheidungsprobleme betrachtet werden, die für Kettenkodebildsprachen bereits untersucht wurden ([DS] bzw [DH])

Neben den üblichen Eigenschaften spielen dabei die folgenden graphentheoretischen Eigenschaften eine Rolle

P_1 Der Graph ist zusammenhängend,

P_2 Der Graph ist 2-fach zusammenhängend,

P_3 Der Graph ist mit k Farben kantenfärbbar, $k \in \{1,2,3\}$,

P_4 Der Graph ist regulär vom Grad k, $k \in \{1,2\}$,

P_5 Der Graph ist kreisfrei,

P_6 Der Graph ist Eulersch,

P_7 Der Graph ist Hamiltonsch,

sowie die Eigenschaft

P_8 Das Bild enthält ein gegebenes Bild als Teilbild.

Eine Familie von Bildsprachen heißt abgeschlossen bezüglich P_i, $i \in \{1, \quad ,8\}$, falls für jede Bildsprache aus dieser Menge der Durchschnitt mit den Bildern mit der Eigenschaft P_i in der gegebenen Familie liegt

<u>Satz 2.</u> Die Familie der REG-RM-Sprachen ist abgeschlossen gegenüber Vereinigung und P_8

<u>Satz 3:</u> Die Familie der REG-RM-Sprachen ist nicht abgeschlossen gegenüber Durchschnittsbildung sowie P_i, $i \in \{1, \quad ,7\}$

Für die Eigenschaften P_i, $i \in \{1,.. ,8\}$ können folgende Entscheidungsprobleme gestellt werden

Q_1: Gibt es ein Bild mit der Eigenschaft P_i?

Q_2: Haben alle Bilder die Eigenschaft P_i?

<u>Satz 4:</u> Die folgenden Probleme sind für CF-RM-Sprachen entscheidbar:

- Leerheitsproblem, Mitgliedschaftsproblem,
- Q_1 für P_8 und Q_2 für P_4,P_5

<u>Satz 5:</u> Die folgenden Probleme sind für REG-RM-Sprachen unentscheidbar

- Q_1 für P_i, $i \in \{1,.. ,7\}$,
- Q_2 für P_i, $i \notin \{4,5\}$,
- das Äquivalenzproblem

Literatur

[DH] Dassow, Hinz: Decision problems and Regular Chain Code Picture Languages, Preprint,1989

[DS] Dassow, Stiebe: On Graph-Theoretical Closure Properties of Chain Code Picture Languages, Preprint,1990

[Fr] Freeman. On the Encoding of Arbitrary Geometric Configurations, Ire Transactions on Electronic Computers, 10 (2) 1961, 260-268

[Si] Siromoney, Siromoney, Krithivasan: Abstract Families of Matrices and Picture Languages, Computer Graphics and Image Processing 1 (1972), 284-307

1. Theorietag
Automaten und Formale Sprachen

Magdeburg, 30.09.-01.10.1991

Teilnehmerliste

Bordihn, Henning	Techn. Univ. „O. v. Guericke" Fakultät für Mathematik	DO-3010 Magdeburg Postfach 4120
Buntrock, Gerhard	Universität Würzburg Lehrstuhl f. Theor. Informatik	DW-8700 Würzburg Am Exerzierplatz 3
Carstensen, Heino	Universität Hamburg Fachbereich Informatik	DW-2000 Hamburg 13 Vogt-Kölln-Str. 30
Dassow, Jürgen	Techn. Univ. „O. v. Guericke" Fakultät für Mathematik	DO-3010 Magdeburg Postfach 4120
Enmic, Diana	Universität Würzburg Lehrstuhl f. Theor. Informatik	DW-8700 Würzburg Am Exerzierplatz 3
Freund, Rudolf	Techn. Universität Wien Institut f. Computersprachen	A-1040 Wien Resselgasse 3
Hemmerling, Armin	E.-M.-Arndt-Universität Fachrichtungen Math./Inf.	DO-2200 Greifswald F.-L.-Jahn-Str. 15a
Hinz, Friedhelm	Universität Trier Fachbereich IV	DW-5500 Trier Postfach 3825
Hotzel, Eckehart	GMD	DW-5205 St. Augustin 1 Postfach 1240
Hümpel, Claudia	Universität Hannover Institut für Informatik	DW-3000 Hannover 1 Welfengarten 1
Jantzen, Matthias	Universität Hamburg Fachbereich Informatik	DW-2000 Hamburg 54 Vogt-Kölln-Str. 30
Kreowski, Hans-Jörg	Universität Bremen FB Mathematik/Informatik	DW-2800 Bremen 33 Postfach 330440
Kühnemann, Armin	Universität Ulm Abt. Theoretische Informatik	DW-7900 Ulm Oberer Eselsberg
Kudlek, Manfred	Universität Hamburg Fachbereich Informatik	DW-2000 Hamburg 54 Vogt-Kölln-Str. 30

Lange, Klaus-Jörn	Techn. Universität München Institut für Informatik	DW-8000 München Arcisstr. 21, PF 202420
Middendorf, Martin	Universität Hannover Institut für Informatik	DW-3000 Hannover 1 Welfengarten 1
Priese, Lutz	Universität Paderborn Fachbereich 17	DW-4790 Paderborn
Reichel, Bernd	Techn. Univ. „O. v. Guericke" Fakultät für Mathematik	DO-3010 Magdeburg Postfach 4120
Roßnick, Torsten	Techn. Univ. „O. v. Guericke" Fakultät für Mathematik	DO-3010 Magdeburg Postfach 4120
Seibert, Sebastian	Universität Kiel Institut für Informatik	DW-2300 Kiel 1 Olshausenstr. 40
Skalla, Stefan	Techn. Univ. „O. v. Guericke" Fakultät für Mathematik	DO-3010 Magdeburg Postfach 4120
Stiebe, Ralf	Techn. Univ. „O. v. Guericke" Fakultät für Mathematik	DO-3010 Magdeburg Postfach 4120
Thomas, Wolfgang	Universität Kiel Institut für Informatik	DW-2300 Kiel 1 Olshausenstr. 40
Vogel, Jörg	Friedrich-Schiller-Universität Mathematische Fakultät	DO-6900 Jena Uni-Hochhaus, 17.OG
Vogler, Heiko	Universität Ulm Abt. Theoretische Informatik	DW-7900 Ulm Oberer Eselsberg
Wätjen, Dietmar	Techn. Univ. Braunschweig Institut für Theor. Informatik	DW-3300 Braunschweig Gaußstr. 11