OTTO-VON-GUERICKE-UNIVERSITÄT MAGDEBURG

11. Theorietag
der GI-Fachgruppe 0.1.5 „Automaten und Formale Sprachen“
mit dem Workshop
Coding Theory and Formal Languages
Wendgräben, 3. – 6. Oktober 2001
Proceedings

Jürgen Dassow und Bernd Reichel (Hrsg.)
Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg
Postfach 4120
39016 Magdeburg
Germany

Preprint Nr. 18
2001

Fakultät für Informatik

Impressum:

Herausgeber:
Otto-von-Guericke-Universität Magdeburg
Der Dekan
Fakultät für Informatik

V. i. S. d. P.:
Jürgen Dassow
Postfach 4120
39016 Magdeburg

Auflage: 110
Redaktionsschluß: Oktober 2001

Redaktion/Gestaltung: Abt. Publikationen und Öffentlichkeitsarbeit
Herstellung: Dezernat Allgemeine Angelegenheiten,
Sachqebiet Reproduktion

Jürgen Dassow und Bernd Reichel (Hrsg.)

11. Theorietag

der GI-Fachgruppe 0.1.5 „Automaten und Formale Sprachen“

mit dem Workshop

Coding Theory and Formal Languages

Wendgräben, 3. – 6. Oktober 2001

Proceedings

Otto-von-Guericke-Universität
Magdeburg

VORWORT

Seit 1991 wird von der GI-Fachgruppe 0.1.5 *Automaten und Formale Sprachen* jährlich der Theorietag mit der Fachgruppensitzung veranstaltet. Die Serie begann 1991 in Magdeburg, wurde 1992 in Kiel, 1993 in Dagstuhl, 1994 in Herrsching, 1995 in Schloss Rauischholzhausen, 1996 in Cunnersdorf, 1997 in Barnstorf, 1998 in Riveris, 1999 in Schauenburg-Elmshagen und 2000 in Wien fortgesetzt. Im Jahre 2001 wird der Theorietag vom 3.–6. Oktober im Bildungszentrum der Konrad-Adenauer-Stiftung in Wendgräben von der Arbeitsgruppe *Formale Sprachen* der Fakultät für Informatik der Otto-von-Guericke-Universität Magdeburg veranstaltet.

Seit Cunnersdorf 1996 gehört zum Theorietag auch ein eintägiger Workshop zu einem aktuellen Thema aus den Grenzgebieten der *Automaten und Formalen Sprachen*. Dieser Tradition folgend, findet in diesem Jahr ein Workshop zum Thema *Coding Theory and Formal Languages* mit den Vortragenden

- Véronique Bruyère (Mons, Belgien)
- Masami Ito (Kyoto, Japan)
- Helmut Jürgensen (Potsdam und London, Kanada)
- Juhani Karhumäki (Turku, Finnland)

statt.

Die Kurzfassungen dieser Vorträge als auch der Beiträge zum eigentlichen Theorietag sind in dem vorliegenden Tagungsband enthalten. Ferner finden Sie hier die Programme des Workshops und des Theorietages sowie eine Liste aller Teilnehmenden mit ihren Adressen.

Dem Kultusministerium des Landes Sachsen-Anhalt sowie der Otto-von-Guericke-Universität Magdeburg gebührt Dank für die Unterstützung des Theorietages. Wir wünschen allen Teilnehmenden einen interessanten und erfolgreichen Theorietag sowie einen angenehmen Aufenthalt in Wendgräben.

Jürgen Dassow
Bernd Reichel

Magdeburg, im Oktober 2001

INHALTSVERZEICHNIS

Workshop
„Coding Theory and Formal Languages“
Wendgräben, 4. Oktober 2001

PROGRAMM

Donnerstag, 4. Oktober 2001

08:55	Begrüßung
09:00 – 10:00	*Véronique Bruyère* (Mons, Belgien): Completion of Codes
10:00	Kaffeepause
10:30 – 11:30	*Masami Ito* (Kyoto, Japan): Set of Primitive Words and Related Languages
11:30	Mittagspause
14:00 – 15:00	*Helmut Jürgensen* (Potsdam und London, Kanada): Synchronizing Codes
15:00	Kaffeepause
15:30 – 16:30	*Juhani Karhumäki* (Turku, Finnland): From Words to Finite Sets of Words
17:00	Besichtigung der St. Laurentius-Kirche in Loburg
18:30	Abendessen

11. Theorietag
„Automaten und Formale Sprachen“
Wendgräben, 5. – 6. Oktober 2001

PROGRAMM

Freitag, 5. Oktober 2001

08:55	Eröffnung des Theorietages
09:00 – 09:25	*Oliver Boldt* (Potsdam): Solid Codes and HS-Codes
09:25 – 09:50	*Ludwig Staiger* (Halle): Decidability of Code Properties
09:50 – 10:15	*Andreas Klein* (Kassel): Faltungscodes aus Sicht der Automatentheorie
10:15	Kaffeepause
10:30 – 10:55	*Heiko Vogler* (Dresden): Tree Transducers with Costs
10:55 – 11:20	*Claus Jürgensen* (Dresden): Composition of Tree Transducers versus Categorical Deforestation
11:20 – 11:45	*Heiko Stamer* (Leipzig): Dreibeinige PCP-Biber
11:45 – 12:10	*Fritz v. Haeseler* (Leuven, Belgien): A Geometric Cobham-Semenov Theorem
12:10	Mittagspause
14:30 – 14:55	*Markus Holzer* (München): Assembling Molecules in Atomix is Hard
14:55 – 15:20	*Holger Petersen* (Stuttgart): Das Wortproblem regulärer Ausdrücke mit Durchschnitt ist vollständig für LOGCFL
15:20 – 15:45	*Klaus Reinhardt* (Tübingen): Über die Multiparty-Kommunikationskomplexität regulärer Sprachen
15:45	Kaffeepause

Freitag, 5. Oktober 2001 (Fortsetzung)

16:00 – 16:25 *Jan-Thomas Löwe* (Giessen):
Auf Zellularautomaten basierende Bilderzeugung und -kompression (Fortsetzung)

16:25 – 16:50 *Dietrich Kuske* (Leicester, England):
Welche Kommunikationsprotokolle lassen sich mit endlichen Automaten beschreiben?

16:50 – 17:15 *Martin Kutrib* (Giessen):
State Complexity of Basic Operations on Nondeterministic Finite Automata

17:15 – 17:40 *Friedrich Otto* (Kassel):
Shrinking Alternating Two-Pushdown Automata

17:50 Fachgruppensitzung

18:30 Abendbrot

Samstag, 6. Oktober 2001

09:00 – 09:25 *Benedikt Bollig* (Aachen):
Generalised Regular MSC Languages

09:25 – 09:50 *Henning Bordihn* (Potsdam):
Decision Problems on the Power of Languages

09:50 – 10:15 *Klaus Wich* (Stuttgart):
Inhärenz der Mehrdeutigkeitsfunktionen kontextfreier Grammatiken

10:15 Kaffeepause

10:30 – 10:55 *Suna Aydin* (Potsdam):
Dialogues as Cooperating Grammars

10:55 – 11:20 *Ralf Stiebe* (Magdeburg):
Positive Valence Grammars

11:20 – 11:45 *Henning Fernau* (Tübingen):
Even Linear Simple Matrix Languages: Formal Language Properties and Grammatical Inference

11:45 – 12:10 *Rudolf Freund* (Wien, Österreich):
Variablenkomplexität in graphkontrollierten, programmierten und Matrix-Grammatiken

12:10 Mittagessen und Ende des Theorietages

COMPLETION OF CODES

Véronique Bruyère
Faculty of Sciences, University de Mons-Hainaut
6 avenue du Champ de Mars, B-7000-Mons, Belgium
e-mail: veronique.bruyere@umh.ac.be

In this talk, I want to document the state of the art of the problem of *completing codes*.

All codes are subsets of maximal codes and the investigation of maximal codes is active and important in the theory of codes. A fundamental result is the equivalence for rational codes between the algebraic notion of *maximal code* and the combinatorial notion of *complete code* (Schützenberger 1955). One of the problem that gained a lot of interest over the twenty past years is, *given a code $X \subseteq A^*$, how to construct a maximal code $Y \subseteq A^*$ sharing the same properties which contains X?*

For instance, a simple construction is known to include any *rational* code into a rational maximal code (Ehrenfeucht-Rozenberg 1985). The proof uses combinatorics on words. The construction leads to a complete code which is a notion simpler to manipulate than the notion of maximal code. However, A. Restivo pointed out *finite* codes not included in any finite maximal code (Restivo 1977). It is not yet known whether the embedding of a finite code into a finite maximal one is decidable. This open problem is one among the difficult ones in the theory of codes.

A lot of completion procedures are known today. I will present some of them when the proofs are simple. Some procedures use combinatorics on words, some others are based on automata manipulations, some also use formal series. Most procedures apply to *rational* codes sharing a certain property $\mathcal{P}$ and they construct a *complete* code with the same property $\mathcal{P}$ and containing the given code. To the best of my knowledge, the list of properties $\mathcal{P}$ that have been studied is: prefix code, code with bounded deciphering delay (Schützenberger 1966, Bruyère-Wang-Zhang 1990, Bruyère 1992, Zhang-Shum-Peng 1998), code with bounded deciphering delay in both directions (Zhang-Shum-Peng 2001), bifix code (Perrin 1984, Zhang-Shen 1995, Bruyère-Perrin 1999), code with bounded synchronization delay (Montalbano 1993, Bruyère 1998), solid code (Jürgensen 1997, Lam 1998), circular code (Bassino 1996), code with finite interpreting delay (Guesnet 2001).

SET OF PRIMITIVE WORDS AND RELATED LANGUAGES

Masami Ito
Faculty of Science, Kyoto Sangyo University
Kyoto 603-8555, Japan
e-mail: ito@ksuvx0.kyoto-su.ac.jp

A word u is said to be *primitive* if u cannot be represented as the power of another word. By $Q(X)$ we denote the set of all primitive words over X. It is conjectured that $Q(X)$ is not context-free. However, this conjecture is still open. We investigate some decidability problems concerning $Q(X)$ and its related languages. Let $u \in X^+$. If $u = v^i$ for a positive integer i and a primitive word v, then we denote $root(u) = v$. For a language $L \subseteq X^+$, we define $root(L) = \bigcup_{u \in L} root(u)$. Then we have the following results:

(1) For a given regular (or context-free) language $L \subseteq X^+$, it is decidable whether $root(L)$ is finite.

(2) For a given regular language $L \subseteq X^+$, it is decidable whether $root(L)$ is regular.

(3) For a given context-free language $L \subseteq X^+$, it is undecidable whether $root(L)$ is regular (or context-free).

(4) For a given regular language $L \subseteq X^+$, it is decidable whether $L \subseteq Q(X)$ holds.

(5) For a given context-free language $L \subseteq X^+$, it is undecidable whether $L \subseteq Q(X)$ holds.

Let $L \subseteq X^+$. Then, by $deg(L)$ we mean the set $\{i : q \in Q(X),\ q^i \in L\}$. Then we have the following results:

(1) For a given regular language $L \subseteq X^+$, it is decidable whether $deg(L)$ is finite.

(2) For a given context-free language $L \subseteq X^+$, it is undecidable whether $deg(L)$ is finite.

A language $L \subseteq X^+$ is said to be *palindromic* if all words in L are palindromes. It is known that there is no dense palindromic regular language contained in $Q(X)$. For the case of context-free palindromic languages, we have the same result and moreover we can prove that $deg(L)$ is infinite (more exactly, aperiodic) if $L \subseteq X^+$ is a dense palindromic context-free language.

SYNCHRONIZING CODES [1]

Helmut Jürgensen
Department of Computer Science, The University of Western Ontario
London, Ontario, Canada, N6A 5B7
and
Insitut für Informatik, Universität Potsdam
Postfach 90 03 27, D-14439 Potsdam, Germany
e-mail: helmut@uwo.ca, helmut@cs.uni-potsdam.de

In modern communication, synchronization errors arising from various physical defects in the communication links are quite common. When information-resend is not a problem, a protocol can take care of these errors. When resend is not feasible – as in high-volume traffic or deep-space communication – coding techniques that allow for the detection and correction of synchronization errors need to be employed. Error models for the type of channels used in modern communication need to be developed and requirements for codes dealing with these error models need to be developed. We shall outline some of the current technical issues and present some techniques for coping with synchronization errors.

[1]This research was supported by the Natural Sciences and Engineering Research Council of Canada.

FROM WORDS TO FINITE SETS OF WORDS

Juhani Karhumäki
Department of Mathematics and Turku Centre for Computer Science
University of Turku, FIN-20014 Turku, Finland
e-mail: karhumak@cs.utu.fi

Words are fundamental objects of mathematics, for example, in many areas of algebra and theoretical computer science. The theory of words – often known as *Combinatorics on Words* – is well developped relatively new research area, cf. [10], [11] and [2]. Over the last few decades a number of fundamental results have been achieved, such as the decidability of the existence of solutions of word equations and the validity of the Ehrenfeucht compactness property, cf. e. g. [12] and [4]. Recently the former problem has been shown to be in *PSPACE*, see [13].

When words are replaced by finite sets of words the situation changes drastically: very little is known on many very natural problems. However, many challenging and natural questions can be stated. These are the points we want to emphasize here.

The extension from words to finite sets of words can be seen – in a natural way – as an attempt to replace deterministic problems by the corresponding nondeterministic ones. As we shall see many problems become much harder.

As an illustration we consider the *commutation problem*. For words it is folklore that two words commute if and only if they are powers of a common word. Simple examples show that it is extremely unlikely to have any similar – or even any at all – characterization for the commutation of finite languages. However, if we consider finite sets of words with multiplicities, or more precisely polynomials over a field with noncommuting unknowns, a nice characterization is available, see [1]:

Bergman's Theorem. *Two polynomials over a field and with noncommuting unknowns commute if and only if they are linear combinations of powers of a common polynomial.*

This motivates to state:

BTC-property. We say that the family $\mathcal{F}$ of languages satisfies BTC-property if the following conditions are equivalent for any $X \in \mathcal{F}$ and any set Y:

(i) $XY = YX$,

(ii) there exist V and sets I and J of indecies such that

$$X = \bigcup_{i \in I} V^i \quad \text{and} \quad Y = \bigcup_{j \in J} V^j.$$

Again simple examples show that the BTC-property does not hold for all finite sets, or even for all four element sets. However, it does hold for several nontrivial classes like that of prefix sets or two element sets, cf. e. g. [14] and [7]. The question when the BTC-property holds is very much open, even the cases when $\mathcal{F}$ is the family of three word sets or codes are unsolved.

A related, but different, problem is the one proposed by Conway 30 years ago, see [3]:

Conway's Problem. Is the maximal set commuting with a given regular set X, i. e. *the centralizer of* X, regular as well.

The problem has turned out to be surprisingly difficult: it is not even known whether the centralizer of a *finite* set is *recursive* (or equivalently *recursively enumerable*). Special cases when the answer to Conway's problem is affirmative are the cases when X contains only three elements or X is so-called ω*-code*, see [9] and [6], respectively.

The above are two simply formulated challenging problems on finite sets of words. The following two results explain – at least implicitly –why such questions seem to be so difficult. In [5] it was shown.

Theorem 1 *It is undecidable whether a given CF language and a two-element set commute.*

In order to formulate another result we need some definitions. We say that two mappings φ and ψ *are equivalent on language* L if

$$\varphi(x) = \psi(x) \quad \text{for all} \quad x \quad \text{in} \quad L.$$

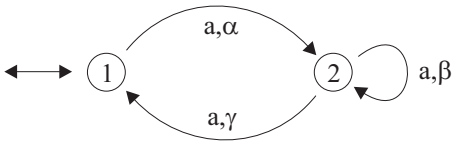
A simple decidability result corresponding to the deterministic case is the one when the mappings are morphisms and the language is regular. This is due to the pumping property of regular languages and a combinatorial lemma on words. The nondeterministic variant of the problem is surprisingly much more complicated. Indeed, in [8] it was recently proved

Theorem 2 *It is undecidable whether two finite substitutions are equivalent on the language* ab^*c.

As a consequence we obtain

Corollary 1 *The equivalence problem for two states gsm's with unary input alphabet is undecidable.*

Actually, in Corollary the machines can be assumed to be of the form



where α, β and γ are finite sets of words.

There exist many other interesting problems on finite sets of words. Even classical questions of finding a minimal grammar or automaton of certain type for a given finite set seems to be poorly understood. However, such questions might be important for many applications, for example in data compression.

References

[1] Bergman, G., Centralizers in free associative algebras, Trans. Amer. Math. Soc. 137, 327–344, 1969.

[2] Choffrut, C. and Karhumäki, J., Combinatorics of Words, In: G. Rozenberg and A. Salomaa (eds),Handbook of Formal Languages, vol. 1, 329–438, Springer, 1997.

[3] Conway, J. H., Regular Algebra and Finite Machines, Chapman Hall, 1971.

[4] Guba, V. S., The equivalence of infinite systems of equations in free groups and semigroups with finite systems (in Russian), Mat. Zametki 40, 321–324, 1986.

[5] Harju, T., Ibarra, O., Karhumäki, J. and Salomaa, A., Decision problems concerning semilinearity, morphisms and commutation of languages, Proceedings of ICALP01, LNCS 2076, 579–590.

[6] Harju, T. and Petre, I., On commutation and primitive roots of codes, manuscript.

[7] Karhumäki, J., Challenges of commutation: an advertisement, Proceedings of FCT01, LNCS 2138, 15–23.

[8] Karhumäki, J. and Lisovik, L., A surprising undecidability result: The equivalence problem for finite substitutions on ab^*c, manuscript.

[9] Karhumäki, J. and Petre, I., Conway's problem for three word sets, Theoret. Comput. Sci., to appear; preliminary version in LNCS 1853, 536–546.

[10] Lothaire, M., Combinatorics on Words, Addison-Wesley, 1983.

[11] Lothaire, M., Algebraic combinatorics on Words, Cambridge University Press, to appear.

[12] Makanin, G. S., The problem of solvability of equations in a free semigroup, Mat. Sb. 103, 147–236, 1977 (English transl. in Math. USSR Sb. 32, 129–198).

[13] Plandowski, W., Satisfiability of word equations with constants is in $PSPACE$, proceedings of FOCS 495–500, 1999.

[14] Ratoandramanana, B., Codes et motifs, RAIRO Theoret. Inform. 23, 425–444, 1989.

DIALOGUES AS COOPERATING GRAMMARS

Suna Aydin
Insitut für Informatik, Universität Potsdam
Postfach 90 03 27, D-14439 Potsdam, Germany
e-mail: aydin@cs.uni-potsdam.de

ABSTRACT

Human-machine interfaces for spoken language require a stable model of dialogue structure that captures the variability and the unpredictability occurring in ordinary dialogues. It is proposed to use cooperating grammars with memories for the dialogue modeling. With this underlying model different dialogue phenomena can be modeled in a standardized fashion. Furthermore the dialogue can be modeled as a joint activity, whereas the public and the private goals can be separated.

Keywords: dialogue processing, dialogue modeling, cooperating grammars, joint activity.

1. Introduction

The dialogue holds different dialogue phenomena, for example sudden change of opinion, disagreements, questions, misunderstandings or unexpected change of topic. Therefore a model of dialogue structure should capture sudden occurring dialogue phenomena. In the following the dialogue structure will be modeled using *cooperating distributed grammar systems with memories* (mCD grammar systems). The dialogue is regarded as a *joint activity* as in [2, 3], that is, the dialogue is modeled as a joint activity which is performed in common by all the participants involved in the dialogue. The participants share goals that they want to achieve. There are two kind of goals that are distinguished in [1]:

- *public goals* are known by all the participants - they are public information. An accepted public goal will be tried to achieve by all the participants in a cooperative fashion.
- each participants can have *private goals* (beliefs, questions etc.) of which the other participants don't know about

With this underlying dialogue model it becomes possible to represent public and private goals separately during dialogue modeling.

2. Dialogue Modeling with mCD Grammar Systems

Each participant in the dialogue will be represented by a grammar in a mCD grammar system. A dialogue usually involves several parties who *take turns*; turn-taking is realized by specific non-terminals, by rule conditions and by memory-relevant information. The definition of the mCD grammar systems in [4] is modified with the object of dialogue modeling and to the effect that no messages are sent and each memory management depends on the single grammars. The memories are used by the single grammars for storing *private notes* or removing them from the memory. The public exchange of information is by means of the current sentential form:

Definition 1 A cooperating distributed grammar system with memories (mCD grammar system) for dialogue modeling *is a construct*

$$\Gamma = (N_S, N', T, P_1, P_2, \ldots, P_n, S),$$

for $n \in N$, *with the following properties:*

1. N_S *is a finite, non-empty set, the set of the* specified nonterminal symbols*;*
2. N' *is a finite, non-empty set, the set of the* updated nonterminal symbols*;*
3. T *is a finite, non-empty set, the set of the* terminal symbols*;*
4. $S \in N_S$*;*
5. *let* $N = N_S \cup N'$*;*
6. P_i *are finite set of production rules of the form*

$$(c, (R, Q) : A \to w, d, n), \text{ where}$$

   (a) $c, d, n \in N_S^*$,
   (b) $R \subseteq N_S$,
   (c) $Q \subseteq N_S$,
   (d) $A \in N$,
   (e) $w \in (N \cup T)^*$.

Let $V = (N \cup T)$. *The* specified nonterminal symbols (N_S) *and* the updated nonterminal symbols (N') *differ in that the updated nonterminal symbols can not occur in the memory.*

A configuration of Γ is a tuple $C = (w, v_1, \ldots, v_n)$, where $w \in V^*$ and $v_i \in N_S^*$, for $1 \leq i \leq n$. In this case w is the current sentential form and v_i, $1 \leq i \leq n$ the content of the memory of G_i.

Let $\Gamma = (N_S, N', T, P_1, \ldots, P_n, S)$, $n \geq 1$, a mCD grammar system and let $C = (w, v_1, \ldots, v_n)$ and $C' = (w, v_1', \ldots, v_n')$ two configurations of Γ. One says, that C directly derives C': $C \Longrightarrow_\Gamma C'$, if the following conditions hold:

1. there exists $(c, (R, Q) : A \to z, d, n) \in P_i$;
2. $w = xAy$ and $w' = xzy$ for $x, y \in V^*$;
3. for all $r \in R$ there are words $u_1, u_2 \in V^*$ such that, $w = u_1 r u_2$;
4. for all $q \in Q$ there are no words $u_1, u_2 \in V^*$ such that, $w = u_1 q u_2$;
5. $v_i = c\overline{v}_i$, for a $\overline{v}_i \in N_S^*$;
6. $v_i' = n v_i''$, where $v_i = d v_i''$, $v_i'' \in N_S^*$;
7. $v_j' = v_j$, for all $j \neq i$.

A direct derivation step of Definition 1 consists of checking the sentential form and the context conditions, executing a context-free production and eventually updating the current contents of the memory. The cooperation protocol is defined by allowing each grammar to make as many steps as it *wants to* non-deterministically (e. g. *-mode).

Formally, let $L(\Gamma) = \{w \mid (S, \epsilon, \ldots, \epsilon) \overset{*}{\Longrightarrow}_\Gamma (w, \epsilon, \epsilon),\ w \in T^*\}$.

3. Discussion

It turns out that the application of the modified mCD grammar systems for dialogue modeling leads to a stable model of dialogue structure, which uniquely captures different dialogue phenomena. Each utterance is represented in the current sentential form. Thus the current sentential form contains the public information and the current (partial) state of the dialogue. The dialogue phenomena like change of topic, question or refusal are represented in the current sentential form with designated nonterminal symbols and regulated by means of the context conditions of each grammar. If more than two participants are involved in a dialogue the addressed participant is specified on the basis of designated nonterminal symbols belonging to the respective participant.

Because of the fact that several grammars work on one sentential form, this dialogue model handles the dialogue as a joint activity of the participants. Yet it is permitted for each grammar to have private information.

References

[1] H. Clark, *Using language*. Cambridge University Press, Cambridge, 1996.

[2] P. Cohen, H. Levesque, Teamwork. *Noûs* **25**(4), 1991, 487-512.

[3] P. Cohen, Dialogue Modeling. In: R. Cole, J. Mariani, H. Uszkoreit, A. Zaenen, V. Zue (eds.), *Survey of the State of the Art in Human Language Technology*. Volume XII·XIII, Cambridge University Press, Pisa, 1997, 204-210.

[4] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, G. Păun, *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*. Gordon and Breach Science Publishers, Amsterdam, 1994.

A GEOMETRIC COBHAM-SEMENOV THEOREM

André Barbé und Fritz v. Haeseler
K.U. Leuven
Kasteelpark Arenberg 10, B-3001 Leuven, Belgium
e-mail: Friedrich.vonHaeseler@esat.kuleuven.ac.be

ABSTRACT

We introduce sets which are generated by a k-substitution and prove a geometric Cobham-Semenov Theorem for these sets.

1. Introduction

It is a well known fact that substitutions can, after a certain graphical representation, generate compact subsets which display self-similar features. In this note we study properties of compact sets which are generated by so called l-dimensional k-substitutions, i. e., a letter is replaced by an l-dimensional array of letters of size k^l.

We study the geometric properties of a compact set X generated by a k-substitution by assigning a closed Abelian group, $\mathrm{Spec}(X)$, to X, the spectrum of X. Using the spectrum we are able to establish a geometric Cobham-Semenov Theorem.

The Cobham-Semenov Theorem, see [3] for the details and further references, is about l-dimensional sequences, i. e., maps $f : \mathbb{N}^l \to S$, with values in a finite set. A sequence is said to be k-automatic if it can be generated by a finite k-automaton. A sequence is said to be definable in $\langle \mathbb{N}, + \rangle$ if the level sets $M_s = \{\underline{i} \in \mathbb{N}^l \mid f(\underline{i}) = s\}$, $s \in S$, are semilinear sets, again see [3].

Now, the Cobham-Semenov Theorem says that a sequence f which is k_1-automatic and k_2-automatic and k_1, k_2 are multiplicatively independent, i. e., $\log k_1 / \log k_2 \notin \mathbb{Q}$, is definable in $\langle \mathbb{N}, + \rangle$. Moreover, any sequence f which is definable in $\langle \mathbb{N}, + \rangle$ is k-automatic for all $k \geq 2$.

For the geometric objects under consideration, i. e., limit sets of k-substitutions, the Theorems 4.1 and 4.2 are the geometric counterparts of the Cobham-Semenov Theorem for sequences. One should note the two analogies:

a) X is generated by a k-substitution and f is a k-automatic sequence,

b) $\mathrm{Spec}(X) = \mathbb{R}$ and f is definable in $\langle \mathbb{N}, + \rangle$.

Furthermore, it is noteworthy that these theorems can be proved without using the Cobahm-Semenov for sequences.

More details and proofs can be found in [2].

2. Words, Sequences, Substitutions

With S we always denote a finite set with distinguished element $0 \in S$. l is an integer such that $l \geq 1$. For an integer $N \geq 1$ we denote by $[N]$ the set $[N] = \{0, \ldots, N-1\}$.

The set $\mathbb{R}^l$ is equipped with the maximum norm $\| \ \|\_\infty$, i. e., $\|\underline{x}\|_\infty = \max\{|x_i| \mid i = 1, \ldots, l\}$.

A *word* ω (over S) is a map

$$\omega : \prod_{j=1}^{l} [N_j] \to S,$$

where $N_j \in \mathbb{N}$, $n_j \neq 0$, for all $j = 0, \ldots, l$. The set of words is denoted as

$$\Omega = \left\{ \omega : \prod_{j=1}^{l} [N_j] \to S \mid N_j \in \mathbb{N} \text{ for } j = 1, \ldots, l \right\}.$$

If $\omega \in \Omega$, then $\mathrm{supp}(\omega) \subset \mathbb{N}^l$ denotes the domain of definition of ω.

A *sequence* f is a map $f : \mathbb{N}^l \to S$, and the set of sequences is denoted as $S^{\mathbb{N}^l}$.

If $(\omega_n)_{n\in\mathbb{N}}$ is a sequence of words, then $(\omega_n)_{n\in\mathbb{N}}$ *converges* to $f \in S^{\mathbb{N}^l}$ if for all $M \in \mathbb{N}$ there exists an $n_0 \in \mathbb{N}$ such that

$$f(\underline{j}) = \omega_n(\underline{j})$$

holds for all $n \geq n_0$ and for all $\underline{j}$ with $\|\underline{j}\|_\infty < M$.

If $\lim_{n\to\infty} \omega_n = f$, then we say that $(\omega_n)_{n\in\mathbb{N}}$ *converges properly* to f if

$$f\,|_{\mathrm{supp}(\omega_n)} = \omega_n$$

holds for all $n \in \mathbb{N}$. In other words, the restriction of f on the domain of definition of ω_n is the word ω_n.

Let k be an integer greater than or equal to 2. Any collection of k^l maps $\sigma_{\underline{i}} : S \to S$, with $\underline{i} \in [k]^l$, induces a map Σ defined as

$$\Sigma : \Omega \cup S^{\mathbb{N}^l} \to \Omega \cup S^{\mathbb{N}^l}$$
$$\Sigma(\mathcal{F})(k\underline{j} + \underline{i}) = \sigma_{\underline{i}}(\mathcal{F}(\underline{j})).$$

The map Σ is called a *k-substitution* (over S). Note that $\Sigma(\Omega) \subset \Omega$ and $\Sigma(S^{\mathbb{N}^l}) \subset S^{\mathbb{N}^l}$.

If Σ is a k-substitution and $f \in S^{\mathbb{N}^l}$ satisfies $\Sigma(f) = f$, then f is called a *fixed point* of the substitution. Note that words are never fixed points of a substitution.

If ω is a word, then the set $G(\omega) = \{\underline{j} \mid \omega(\underline{j}) \neq 0\}$ is called the *graphical representation of words* ω. Note that $G(\omega)$ is a compact set. If f is a sequence, then we define for $N \in \mathbb{N}$, $N \geq 1$, the *graphical representation of sequences* as $X(f; N) = \{\underline{j} \mid f(\underline{j}) \neq 0 \text{ and } \|\underline{j}\|_\infty < N\}$.

3. Limit Sets and Spectra

The set of nonempty compact subsets of $\mathbb{R}^l$ is denoted by $\mathcal{H}(\mathbb{R}^l)$. Equipped with the Hausdorff distance induced by $\| \|_\infty$, this set becomes a complete metric space, see [4].

Definition 3.1 *A compact set $X \subset \mathbb{R}^l$ is* generated *by a k-substitution if there exist a k-substitution Σ over S and an $s \in S$ such that*

$$X = \lim_{n\to\infty} \frac{1}{k^n} G(\Sigma^n(s)).$$

Among the sets generated by k-substitutions is the class of k-limit sets.

Definition 3.2 *A compact subset $X \subset \mathbb{R}^l$ is called a* k-limit set *if there exist a k-substitution Σ (over S) and a fixed point $f \in S^{\mathbb{N}^l}$ such that*

$$X = \lim_{n\to\infty} \frac{1}{k^n} X(f; k^n).$$

Every k-limit set X is also generated by a k-substitution. The converse is not true. A compact subset $X \in \mathbb{R}^l$ is called a 1-limit set, if there exists a sequence $f \in S^{\mathbb{N}^l}$ such that f is definable in $\langle \mathbb{N}, + \rangle$ and

$$X = \lim_{n\to\infty} \frac{1}{n} X(f; n).$$

Lemma 3.3 *If X, Y are generated by k-substitutions, then $X \cup Y$ and $X \cap Y$ are generated by k-substitutions.*

In the above lemma k-substitutions can be replaced by k-limit sets.

Theorem 3.4 *Let $(\omega_n)_{n\in\mathbb{N}}$ be a sequence which converges properly to $f \in S^{\mathbb{N}^l}$ and for which $\lim_{n\to\infty} \frac{1}{k^n}\mathrm{supp}(\omega_n) = \prod[0, \alpha_i] \subset [0, 1]^l$. If*

$$\lim_{n\to\infty} \frac{1}{k^n} X(f; k^n) = X(f),$$

then

$$Y = \lim_{n\to\infty} \frac{1}{k^n} G(\omega_n) = \mathrm{cl}\left(X(f) \cap \prod[0, \alpha_i[\right),$$

where cl *denotes the closure.*

We call the limit set Y of the sequence (ω_n) in Theorem 3.4 a *mutilated version* of $X(f)$.

Theorem 3.5 *If $f \in S^{\mathbb{N}^l}$ is such that $(\frac{1}{k^n} X(f; k^n))_{n\in\mathbb{N}}$ is a Cauchy sequence, then the limit is a k-limit set.*

For subsets generated by a k-substitution we have the following decomposition theorem.

Theorem 3.6 *If X is generated by a k-substitution, then X is the finite union of translated mutilated k^{s_i}-limit sets, i. e, there exists a natural number N such that*

$$X = \bigcup_{i=0}^{N} (\tau_i + Y_i),$$

where $\tau_i \in \mathbb{Q}^l$ and Y_i is a mutilated k^{s_i}-limit set.

For our further investigations of k-limit sets we introduce the spectrum of a subset $X \subseteq \mathbb{R}^l$. Note that log means the natural logarithm (actually, any logarithm is possible).

Definition 3.7 *Let $X \subseteq \mathbb{R}^l$ such that $0 \in X$. The set* $\mathrm{Spec}(X)$ *defined as*

$$\mathrm{Spec}(X) = \{\log s \mid s > 0,\ sX \subseteq X \text{or } X \subseteq sX\}$$

is called the spectrum of X *(at zero).*

Note that $0 \in \mathrm{Spec}(X)$ for all X. The important property of the spectrum is stated in the next theorem.

Theorem 3.8 *The spectrum of a subset $X \subseteq \mathbb{R}^l$ with $0 \in X$ is a closed subgroup of the group $(\mathbb{R}, +)$.*

As a consequence we have that either $\mathrm{Spec}(X) = \{0\}$ or $\mathrm{Spec}(X) = \zeta\mathbb{Z}$ for a positive real ζ or $\mathrm{Spec}(X) = \mathbb{R}$. For k-limit sets $\mathrm{Spec}(X) = \{0\}$ is not possible. Indeed, we have the following

Theorem 3.9 *If X is a k-limit set, then*

$$(\log k)\mathbb{Z} \subseteq \mathrm{Spec}(X).$$

Moreover, if $\mathrm{Spec}(X) = \zeta\mathbb{Z}$, $\zeta > 0$, *then there exists a natural number* $m \geq 1$ *such that* $\zeta = \frac{1}{m}\log k$.

Finally we denote the relation between the spectrum of a k-limit set X and the spectrum of a mutilated version.

Lemma 3.10 *If X is a k-limit set and Y a mutilated version of X, then there exists an $m \in \mathbb{N}$, $m \geq 1$ such that $m(\log k)\mathbb{Z} = \mathrm{Spec}(Y) \subset \mathrm{Spec}(X)$.*

4. Geometric Cobham-Semenov Theorems

In this section we state the geometric version of the Cobham-Semenov Theorem.

Theorem 4.1 *Let k_1, $k_2 \geq 2$ be multiplicatively independent. If X is a k_1-limit set and a k_2-limit set, then* $\mathrm{Spec}(X) = \mathbb{R}$.

The proof is based on Theorem 3.9. We have that $\log k_1$, $\log k_2 \in \mathrm{Spec}(X)$ and therefore $\{a\log k_1 + b\log k_2 \mid a, b \in \mathbb{Z}\} \subset \mathrm{Spec}(X)$. Since k_1 and k_2 are multiplicatively independent, i. e., $\log k_1$ and $\log k_2$ are linearly independent over $\mathbb{Q}$, Theorem 3.8 implies that $\mathrm{Spec}(X) = \mathbb{R}$.

The next theorem deals with the case of a k-limit set with spectrum equal to $\mathbb{R}$.

Theorem 4.2 *If $X \subset \mathbb{R}^l$ is k-limit set such that* $\mathrm{Spec}(X) = \mathbb{R}$, *then X is a 1-limit set.*

The proof is based on a geometric description of X and on an induction argument on the dimension l.

The geometric description is as follows. Since X has spectrum $\mathbb{R}$, it follows that either $X = \{0\}$ or X is a cone over $Y \subset B_1(0)$, where $B_1(0)$ is the closed ball of radius 1. In other words

$$X = \{ry \mid y \in Y,\, r \in [0,1]\}.$$

A simplex Δ is the convex hull of finitely many points. With this notion we have that a k-limit set X with spectrum equal to $\mathbb{R}$ is the cone over $Y \in B_1(0)$ and Y is a finite collection of simplices such that the vertices of the simplex have rational coordinates.

Finally, we sketch the induction argument. For $l = 1$ we obviously have that $X = [0,1]$ or $X = \{0\}$ and it is clear that X is a 1-limit set. The induction step is based on a description of the set $Y = X \cap B_1(0)$. This intersection is a finite collection of nonempty sets X_i viewed as subsets of $\mathbb{R}^{l-1}$. Any set X_i is generated by a k-substitution. Due to Theorem 3.6 each of these sets X_i is a finite union of translated mutilated k-limit sets. Furthermore, all these mutilated limit sets have spectrum equal to $\mathbb{R}$ By our induction hypothesis, we have that each of these mutilated k-limit set is a cone over finitely many simplices with rational vertices. In other words, every mutilated limit set is itself a simplex with rational coordinates. Therefore Y is the finite union of simplices with rational vertices. This proves the assertion.

As a consequence of the above theorems we note

Corollary 4.3 *Let X be a k-limit set such that* $\mathrm{Spec}(X) = \zeta\mathbb{Z}$. *If k' is such that k and k' are multiplicatively independent, then X is not generated by a k'-substitution.*

We close with an application to automatic sequences. Due to a result in [1], every k-automatic sequence f determines a k^p-limit set $X(f)$.

Corollary 4.4 *If* $f \in S^{\mathbb{N}^l}$ *is* k*-automatic such that* $\mathrm{Spec}(X) = \zeta\mathbb{Z}$*, then* f *is not definable in* $\langle \mathbb{N}, + \rangle$.

References

[1] A. Barbé, F. von Haeseler. Limit sets of automatic sequences, SISTA/COSIC report nr. 01-84, KU Leuven, 2001.

[2] A. Barbé, F. von Haeseler. A geometric Cobham-Semenov Theorem, in preparation.

[3] V. Bruyère, G. Hansel, C. Michaux, R. Villemaire. Logic and p-recognizable sets of integers, Bull. Belg. Math. Soc. **2**(1994), 191–238.

[4] C. Kuratowski. *Topology*, Acad. Press, New York, 1966.

SOLID CODES AND HS-CODES

Oliver Boldt
Universität Potsdam, Institut für Informatik
Postfach 90 03 27, D-14439 Potsdam, Germany
e-mail: boldt@cs.uni-potsdam.de

ABSTRACT

We introduce HS-codes, a class of codes less restictive than solid codes, and give first characterizations.

1. Introduction

Solid codes (s-codes), see [1, 2, 3] are codes with strong synchronization capabilities. However, the definition conditions for them, notably overlap freeness, imposes strong restrictions on the choice of possible code words. Here, we consider another, similar class of codes, namely hs-codes. With their definition the strong overlap freeness condition in the definition of solid codes is substantially loosened. By this means, the choice of possible code words increases substantially. Nevertheless, they preserve the strong synchronization capability of the s-codes.

Here, we are going to give a characterization of the subclass of hs-codes in a^+b^+. Furthermore, a necessary and sufficient condition for them to be maximal is given. Finally, it is shown that the set a^+b^+ can be partitioned with optimally maximal hs-codes.

2. Definitions and Basics

Definition 2.1 (Solid codes) *A code C is called solid, if it has the following two properties:*

1. $\mathit{Inf}(C) \cap C = \emptyset$ (*infix-freeness*),
2. $\mathit{Pref}(C) \cap \mathit{Suff}(C) = \emptyset$ (*overlap-freeness*).

Example 2.1 Each solid code C, $C \subseteq a^+b^+$, consists of a single word only.

Example 2.2 $\{aba^kb^2 \mid k \geq 1\}$ is a solid code.

Example 2.3 $\{ab^kc \mid k \geq 0\}$ is a solid code over the 3-elementary alphabet $\{a, b, c\}$.

Example 2.4 $\{aud \mid u \in \{b, c\}^*\}$ is a solid code over the 4-elementary alphabet $\{a, b, c, d\}$.

2.1. Properties of Solid Codes

Solid codes inherit all general properties of prefix codes, for example instantaneous decodability.

Now, consider a piece of transmission channel output containing a code word u of a solid code C which has actually been sent (by the sender) in order to encode a message. Let v_1, v_2, v_3 be other words occuring in the channel output. Furthermore, let v_1 be an infix of u, u be an infix of v_2, and v_3 overlap with u.

We see that, under the assumption that the output has been transmitted without errors, neither of these can be code words of C. v_1 is an infix of u; so being a code word would contradict Condition 1 of Definition 2.1. Concerning v_2, we have a similar situation, u is an infix of v_2, contradicting the same condition. Finally, v_3 overlaps with u, which is excluded for code words by Condition 2 of Definition 2.1.

This consideration essentially comprises all imaginable situations in which code words occur in correct channel output without being meant to be sent as such (by the sender). So we conclude that each code word in a piece of correct output has actually been meant to be encoding a message. No further assumption has to be made for this conclusion, especially the correct piece of output may consist of just one code word. This property is subsumed under the notion of synchronization capability.

2.2. HS-Codes

In the case of s-codes, overlap freeness imposes strong restrictions on the structure of possible codewords. As we have seen, this condition guarantees – together with infix freeness – the synchronization capability of s-codes. In this section, we are going to loosen this strong restriction substantially. Nevertheless, the synchronization capability will be preserved.

Definition 2.2 (hs-code) *A code C is called an hs-code if the following two conditions are satisfied:*

1. $\mathit{Inf}(C) \cap C = \emptyset$ (*infix-freeness*),
2. $\mathit{Suff}(C)\mathit{Pref}(C) \cap C = \emptyset$ (*overlay-freeness*).

Hs-codes differ from s-codes in the second condition. The second condition for s-codes implies the second condition for hs-codes.

Proposition 2.1 *If a code C is overlap free, then it is overlay free.*

Proof. We are going to show the 'contraposition' of the statement, which is equivalent.

If C is not overlay free, then there are a $u \in \mathrm{Pref}(C)$ and a $v \in \mathrm{Suff}(C)$, and $vu \in C$. Hence $v \in \mathrm{Pref}(C)$ (and $u \in \mathrm{Suff}(C)$) too; that is, C is not overlap free. □

The converse is not true; see the following counterexample.

Example 2.5 Let $i, j > 0$. Then $\{a^i b^k a^j \mid k \geq 1\}$ is an hs-code. It is no s-code.

We have seen that hs-codes are not necessarily overlap free. Nevertheless, they preserve the synchronization capability discussed above for s-codes. The argumentation is the same as for s-codes.

Example 2.6 Here we present an example of an hs-code out of $a^+b^+a^+b^+$, which is not an s-code.

$$\{a^k b^k a^{k+1} b^{k+1} \mid k \geq 0, k \not\equiv 0 \bmod 3\}$$

3. A Characterization of HS-Codes in a^+b^+

Theorem 3.1 *Let $C \subseteq a^+b^+$. The following two statements are equivalent:*

1. *C is an hs-code.*
2. *There are functions $f, g : \mathbb{N} \to \mathbb{N}$, f monotonically increasing, g monotonically decreasing, and an $m \in \mathbb{N}$, such that $C = \{a^{f(k)} b^{g(k)} \mid k \in [1, m]\}$.*

4. Maximal HS-Codes in a^+b^+

A code C of a certain type is called *maximal* (with respect to its type), if there is no word w, $w \notin C$, such that $C \cup \{w\}$ is another code of this type.

Theorem 4.1 *Let C be an hs-code, denoted as $C = \{a^{f(k)}b^{g(k)} \mid k \in [1,m]\}$ by means of functions f, g and a number m according to theorem 3.1.*
The following two statements are equivalent:

1. *C is maximal.*
2. $f(1) = 1$,
   $f(k+1) - f(k) = 1 \quad or \quad g(k) - g(k+1) = 1 \quad for\ all \quad k \in [1, m-1]$,
   $and \quad g(m) = 1$.

Definition 4.1 *A code C of a certain type is called optimally maximal (with respect to its type), if for any word $u \in C$, any code D of the same type containing u is itself contained in C. In other words, if $C \cap D \neq \emptyset$, then $D \subseteq C$.*

Example 4.1 Let $D_m = \{a^{m+1-l}b^l \mid m \geq l \geq 1\}$ for some $m \geq 1$. All these D_ms are hs-codes and, with the exception of $m = 1$, no s-codes. Moreover, they are optimally maximal (for all $m \geq 1$).

The codes from the previous example form a partition of a^+b^+:

Proposition 4.2

$$\bigcup_{l \geq 1} D_l = a^+b^+ \quad and \quad D_k \cap D_l = \emptyset \quad for\ k \neq l.$$

References

[1] H. Jürgensen and S. Konstantinidis. Codes. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Volume I*, pages 511–607. Berlin: Springer, 1997.

[2] H. Jürgensen and S. S. Yu. Solid codes. *J. Inform. Process. Cybernet., EIK*, pages 563–574, 1990.

[3] H. J. Shyr and S. S. Yu. Solid codes and disjunctive domains. *Semigroup Forum*, 41:23–37, 1990.

GENERALISED REGULAR MSC LANGUAGES [1]

Benedikt Bollig, Martin Leucker and Thomas Noll
Lehrstuhl für Informatik II, RWTH Aachen
Ahornstraße 55, D-52074 Aachen, Germany
e-mail: {bollig,leucker,noll}@informatik.rwth-aachen.de

1. Introduction

A *Message Sequence Chart* (*MSC*) defines a set of processes and a set of communication actions between these processes. In the visual representation of an MSC, processes are drawn as vertical lines. A labelled arrow from one line to a second corresponds to the communication event of sending the labelling value from the first process to the second. Figure 1 (a) gives an example of an MSC. Collections of MSCs are used to capture the scenarios that a designer might want the system to follow or to avoid.

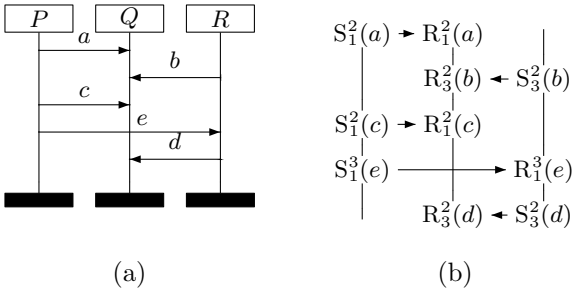


Figure 1: An MSC and its formalisation

Considering the exact behaviour of an MSC, i. e. the sequences of actions which may be observed when the system is executed, one often emanates from the so-called *visual-order semantics*. The visual order assumes that the events are ordered as shown in the MSC. That is, the events on a single process line are linearly ordered, and sending events precede their corresponding receiving events. For example, Process Q in Figure 1 (a) reads an a symbol before it can read a b. We adopt this visual-order point of view.

In a pioneering work by Henriksen et al. [4], a definition of *regularity* of MSC languages is proposed. A characterisation in terms of message passing automata and in terms of monadic second-order logic is also given. The paper explains in a convincing way the benefits of these alternative descriptions, arguing that this is the "right" notion of regularity for MSCs. For example, a characterisation in terms of finite devices (automata) gives evidence for a collection of MSCs to be *realisable*.

[1] A full version of this paper is available [2].

However, their approach has a serious limitation. So-called "MSCs with message overtaking" cannot be considered. But these are explicitly defined in the official standard [5] and must not be ignored. The limitation stems from the fact that, to establish a link between MSCs and classical language theory, the graphical representation of an MSC has somehow to be mapped to the domain of strings. The straightforward approach, enumerating the possible linearisations of the events which occur in an MSC, only works for simple types of MSCs where the correspondence between a sending event and its receiving counterpart can be derived from the order in which they occur in the string.

Our solution to this problem is to associate to every communication event in the string representation of an MSC a natural number which explicitly establishes this correspondence. As it will become clear in the next section, this allows us to drop any restriction on the set of MSCs under consideration.

Our main contribution is to develop a notion of linearisations of MSCs, a theory of regular collections of MSCs in terms of Nerode right congruences, finite automata, and models of MSO formulas for the full class of MSCs. Thus, we provide the formal basis for subsequent verification questions. Note that our approach has already turned out to be useful in the setting of LTL model checking for MSCs [1].

2. MSCs and Their Linearisations

In this section, we present our formal model for MSCs, and we establish a string representation which describes their behaviour in a linear way.

Message Sequence Charts For $N \geq 2$, let $\mathcal{P}_N := \{1, \ldots, N\}$ be a nonempty set of *processes* and Λ a *message alphabet*. Let $\Sigma_{\mathcal{S}} := \{\mathrm{S}_p^q(\lambda) \mid p, q \in \mathcal{P}_N,\ p \neq q,\ \lambda \in \Lambda\}$ and $\Sigma_{\mathcal{R}} := \{\mathrm{R}_p^q(\lambda) \mid p, q \in \mathcal{P}_N,\ p \neq q,\ \lambda \in \Lambda\}$ denote the sets of *send* and *receive actions*, respectively, $\Sigma := \Sigma_{\mathcal{S}} \cup \Sigma_{\mathcal{R}}$ their union. An action $\mathrm{S}_p^q(\lambda)$ stands for sending a message λ from process p to process q, an action $\mathrm{R}_p^q(\lambda)$ for the corresponding receive action, which is then executed by process q. $Corr := \{(\mathrm{S}_p^q(\lambda), \mathrm{R}_p^q(\lambda)) \mid p, q \in \mathcal{P}_N, p \neq q, \lambda \in \Lambda\}$ relates those actions which belong together. From now on, all premises and definitions are made wrt. a fixed set $\mathcal{P}_N$ of processes and a fixed message alphabet Λ where Σ denotes the corresponding set of actions.

A *message sequence chart* (MSC) M is a tuple $(\{E_p\}_{p \in \mathcal{P}_N}, \{\preceq_p\}_{p \in \mathcal{P}_N}, f, L)$ where $\{E_p\}_{p \in \mathcal{P}_N}$ is a family of pairwise disjoint nonempty finite sets of so-called *events* each of which is totally ordered by a well-founded relation $\preceq_p \subseteq E_p \times E_p$. (For simplicity, we consider $\preceq_p$ as a relation over $E := \bigcup_{p \in \mathcal{P}_N} E_p$, the set of all events.) Let $P : E \cup \Sigma \to \mathcal{P}_N$ yield the process an event or an action belongs to, i. e. $P(e) = p$ iff $e \in E_p$, $P(\mathrm{S}_p^q(\lambda)) = p$, and $P(\mathrm{R}_p^q(\lambda)) = q$. M is required to induce a partition $E = S \cup R$ of the events into send (S) and receive events (R) such that f is a bijective mapping from S to R satisfying the following:

- The *visual order* $\preceq \subseteq E \times E$ of M, the reflexive and transitive closure of $\bigcup_{p \in \mathcal{P}_N} \preceq_p \cup \{(e, f(e)) \mid e \in S\}$, is a partial order.
- $L : E \to \Sigma$ provides information about the messages being interchanged by communicating events, whereby $e \in S$ implies both $L(e) = \mathrm{S}_{P(e)}^{P(f(e))}(\lambda)$ and $L(f(e)) = \mathrm{R}_{P(e)}^{P(f(e))}(\lambda)$ for some $\lambda \in \Lambda$.

Figure 1 (b) presents a formal version of the MSC shown in Figure 1 (a).

Let $M = (\{E_p\}_{p \in \mathcal{P}_N}, \{\preceq_p\}_{p \in \mathcal{P}_N}, f, L)$ be an MSC. A *configuration* of M is a subset E' of E satisfying $E' = {\downarrow} E' := \{e \in E \mid \exists e' \in E' : e \preceq e'\}$. Let $Conf(M)$ denote the set of configurations of M. The execution of M may be described by a transition relation $\longrightarrow_M \subseteq Conf(M) \times \Sigma \times Conf(M)$ where $c \xrightarrow{\sigma}_M c'$ iff there exists $e \in E$ such that $L(e) = \sigma$ and $c' = c \cup \{e\}$.

MSC Words A suitable notion of regularity of a class of objects should have similarities with existing notions for regular sets of objects. We will therefore reduce regularity of collections of MSCs to regularity of word languages. Thus, we have to identify an MSC with a set of words that are called linearisations or *MSC words.* A linearisation represents a possible execution sequence of the events occurring in an MSC. To justify this view, it is necessary to guarantee that—up to isomorphism—from a set of linearisations a corresponding MSC can be unambiguously inferred and vice versa. For this purpose, each position of a word $w \in \Sigma^*$ is equipped with a natural number indicating the matching positions (namely those showing the same number). The words $\alpha_1, \alpha_2 \in (\Sigma \times \mathbb{N})^*$ from Figure 3 are such MSC words. Notice that α_1 will determine the MSC M_1, whereas M_2 will emerge from α_2. [4] do not allow an MSC like M_2 to avoid these difficulties. However, M_2 is a so-called "MSC with message overtaking" and carefully treated in the MSC standard [5].

Figure 2: MSCs generated by α_1 and α_2

We call a word $\alpha \in (\Sigma \times \mathbb{N})^*$ *proper* iff for all $(\sigma, \tau) \in \mathit{Corr}$, $\pi \in \mathbb{N}$, and prefixes α' of α, $|\alpha'|_{(\tau,\pi)} \leq |\alpha'|_{(\sigma,\pi)} \leq |\alpha'|_{(\tau,\pi)} + 1$, and we call it *complete* iff it is proper and for all $(\sigma, \tau) \in \mathit{Corr}$ and $\pi \in \mathbb{N}$, $|\alpha|_{(\sigma,\pi)} = |\alpha|_{(\tau,\pi)}$.

Definition 1 (MSC Word) *A word* $\alpha = \frac{\sigma_1}{\pi_1} \cdots \frac{\sigma_\ell}{\pi_\ell} \in (\Sigma \times \mathbb{N})^*$ *is called an* MSC word *iff it is complete. Let* MW *denote the set of all MSC words and* PW *the set of proper words.*

For examples, look at the words $\alpha_1, \alpha_2 \in \mathrm{MW}$ as given in Figure 3. We will refer to them as exemplary MSC words through the rest of the paper.

$$\alpha_1 = \begin{matrix} S_1^2(a) & S_1^2(a) & R_1^2(a) & R_1^2(a) \\ 1 & 3 & 1 & 3 \end{matrix} \qquad \alpha_2 = \begin{matrix} S_1^2(a) & S_1^2(a) & R_1^2(a) & R_1^2(a) \\ 1 & 2 & 2 & 1 \end{matrix}$$

Figure 3: Exemplary MSC words

Given a proper word $\alpha = \frac{\sigma_1}{\pi_1} \cdots \frac{\sigma_\ell}{\pi_\ell} \in \mathrm{PW}$, we determine which positions are *matching.* For $i, j \in \{1, \ldots, \ell\}$, we write $i \searrow_\alpha j$ iff $i < j$, $(\sigma_i, \sigma_j) \in \mathit{Corr}$, and $j = \min\{k \mid k > i \text{ and } \pi_k = \pi_i \text{ and } (\sigma_i, \sigma_k) \in \mathit{Corr}\}$.

From MSC Words to MSCs Given the matching relation, a word $\alpha = \frac{\sigma_1}{\pi_1} \cdots \frac{\sigma_\ell}{\pi_\ell} \in \mathrm{MW}$ generates an MSC $M(\alpha) := (\{E_p\}_{p \in \mathcal{P}_N}, \{\preceq_p\}_{p \in \mathcal{P}_N}, f, L)$ where $E_p = \{n \in \{1, \ldots, \ell\} \mid P(\sigma_n) = p\}$, $S = \{n \in \{1, \ldots, \ell\} \mid \sigma_n \in \Sigma_{\mathcal{S}}\}$, $R = \{n \in \{1, \ldots, \ell\} \mid \sigma_n \in \Sigma_{\mathcal{R}}\}$, $n \preceq_p m$ iff $n, m \in E_p$ and $n \leq m$, $f(n) = m$ iff $n \searrow_\alpha m$, and $L(n) = \sigma_n$. For example, α_1 generates the MSC M_1 illustrated in Figure 2, whereas α_2 generates M_2.

We define two equivalence relations $\approx \subseteq \mathrm{PW} \times \mathrm{PW}$ and $\sim \subseteq \mathrm{MW} \times \mathrm{MW}$. The first identifies words with equivalent projections onto the second component, and the latter allows to permute the positions of an MSC word.

Thus, for $\alpha = \frac{\sigma_1}{\pi_1} \cdots \frac{\sigma_\ell}{\pi_\ell} \in \mathrm{PW}$ and $\beta = \frac{\tau_1}{\rho_1} \cdots \frac{\tau_m}{\rho_m} \in \mathrm{PW}$, let $\alpha \approx \beta$ iff $\ell = m$, $\sigma_i = \tau_i$ for all $i \in \{1, \ldots, \ell\}$, and $i \searrow_\alpha j$ iff $i \searrow_\beta j$.

For a proper word $\alpha = \frac{\sigma_1}{\pi_1} \cdots \frac{\sigma_\ell}{\pi_\ell} \in \mathrm{PW}$, let $\mathit{open}(\alpha) \subseteq \Sigma_{\mathcal{S}} \times \mathbb{N}$ denote the set of those send events which are not followed by a matching receive event, i. e. $\mathit{open}(\alpha) := \{(\sigma_i, \pi_i) \mid \sigma_i \in \Sigma_{\mathcal{S}}$

and there is no $j > i$ such that $i \lessdot_\alpha j\}$. We call the elements of $open(\alpha)$ *open events*. A word $\alpha \in \mathrm{PW}$ is called in *normal form* iff for all prefixes $\frac{\sigma_1}{\pi_1} \cdots \frac{\sigma_k}{\pi_k}$ of α, $\sigma_k \in \Sigma_{\mathcal{S}}$ implies $\pi_k = \min\{\pi \in \mathbb{N} \mid (\sigma_k, \pi) \notin open(\frac{\sigma_1}{\pi_1} \cdots \frac{\sigma_{k-1}}{\pi_{k-1}})\}$. Thus, for every sending event, the lowest available number is chosen. Note that every equivalence class in $\mathrm{PW}/_\approx$ contains exactly one word in normal form. For $\alpha \in \mathrm{PW}$, let furthermore $nf(\alpha) = \beta$ iff $\alpha \approx \beta$ and β is in normal form. For instance, $nf(\alpha_1) = \alpha_1^2$, whereas α_2 is already in normal form so that $nf(\alpha_2) = \alpha_2$.

Definition 2 (MSC Word Language) *A set $\mathcal{L} \subseteq \mathrm{MW}$ is called an* MSC word language *iff $\mathcal{L} = \mathcal{L}^\approx$ where $\mathcal{L}^\approx$ denotes the $\approx$-closure of $\mathcal{L}$.*

For a natural number B, $\alpha \in \mathrm{MW}$ is called *B-bounded* iff for all prefixes α' of α and actions $\sigma \in \Sigma_{\mathcal{S}}$, $|open(\alpha') \cap \{(\sigma, \pi) \mid \pi \in \mathbb{N}\}| \leq B$. This means that the number of open events is bounded by B for every send action.

Linearisations of MSCs In order to finally relate MSCs to the rich theories of languages and automata over words, the concept of linearisations of an MSC is essential. We call an MSC word $\alpha = \frac{\sigma_1}{\pi_1} \cdots \frac{\sigma_\ell}{\pi_\ell} \in (\Sigma \times \mathbb{N})^*$ a *linearisation* of an MSC $M = (\{E_i\}_{i \in \mathcal{P}_N}, \{\preceq_i\}_{i \in \mathcal{P}_N}, f, L)$ with a set of events $E = \{e_1, \ldots, e_\ell\}$ iff there are configurations $c_1, \ldots, c_\ell \in Conf(M)$ with $\emptyset \xrightarrow{\sigma_1}_M c_1 \xrightarrow{\sigma_2}_M \cdots \xrightarrow{\sigma_\ell}_M c_\ell$ and there is a bijective mapping $\chi : E \to \{1, \ldots, \ell\}$ such that for all $e \in E$, $L(e) = \sigma_{\chi(e)}$, and for all $e \in S$ and $e' \in R$, $f(e) = e'$ implies $\chi(e) \lessdot_\alpha \chi(e')$. $Lin(M)$ denotes the set of linearisations of M. For a set $\mathcal{M}$ of MSCs, we canonically define $Lin(\mathcal{M}) := \bigcup\{Lin(M) \mid M \in \mathcal{M}\}$. For instance, the exemplary word α_1 is a linearisation of the MSC M_1 shown in Figure 2, and α_2 is a linearisation of M_2.

An MSC is called B-bounded iff all of its linearisations are B-bounded. A collection of MSCs (a collection of MSC words, respectively) is B-bounded iff all members are B-bounded. Furthermore, we speak of *boundedness* in general iff we deal with B-boundedness for any B.

We turn towards $\sim \subseteq \mathrm{MW} \times \mathrm{MW}$, the equivalence relation that takes permutations of positions into account. For $(\sigma, \pi), (\sigma', \pi') \in \Sigma \times \mathbb{N}$, let $(\sigma, \pi) D (\sigma', \pi')$ iff $P(\sigma) = P(\sigma')$ or $((\sigma, \sigma') \in Corr$ and $\pi = \pi')$ or $((\sigma', \sigma) \in Corr$ and $\pi = \pi')$. We then define the relation $\sim$ to be the least equivalence relation satisfying the following: If $\alpha = \frac{\sigma_1}{\pi_1} \cdots \frac{\sigma_i}{\pi_i} \frac{\sigma_{i+1}}{\pi_{i+1}} \cdots \frac{\sigma_\ell}{\pi_\ell}$ and $\alpha' = \frac{\sigma_1}{\pi_1} \cdots \frac{\sigma_{i+1}}{\pi_{i+1}} \frac{\sigma_i}{\pi_i} \cdots \frac{\sigma_\ell}{\pi_\ell}$ and not $(\sigma_i, \pi_i) D (\sigma_{i+1}, \pi_{i+1})$ then $\alpha \sim \alpha'$.

Applying techniques from the theory of Mazurkiewicz traces, we obtain that for an MSC M and $\alpha \in Lin(M)$, $Lin(M) = Lin(M(\alpha))$, and that for $\alpha \in \mathrm{MW}$, $Lin(M(\alpha)) = [\alpha]_{(\approx \cup \sim)^*}$.

3. Regular MSC Languages and Their Automata

Regularity of collections of MSCs will be defined in terms of regular MSC word languages. But as MSC words are defined over the infinite alphabet $\Sigma \times \mathbb{N}$, we have to modify the usual notion of regularity. We first constitute an algebraic characterisation of regularity by means of a slightly adapted version of the Nerode right congruence which allows a straightforward extension to infinite alphabets. Then, we establish its equivalence to an automata model which has similarities with the one described in [6] but is modified to suit the requirements for MSCs and allows stronger decidability results.

Regular MSC Word Languages Given an MSC word language $\mathcal{L}$, recall the Nerode right congruence $\equiv_{\mathcal{L}} \subseteq \mathrm{PW} \times \mathrm{PW}$: $\alpha \equiv_{\mathcal{L}} \beta$ iff $(\forall \gamma \in (\Sigma \times \mathbb{N})^* : \alpha\gamma \in \mathcal{L}$ iff $\beta\gamma \in \mathcal{L}$.

Let $\mathcal{L}$ be an MSC word language. Since we want to identify $\approx$-equivalent words, we define $\mathbin{\overline{\approx}}_{\mathcal{L}} \subseteq \mathrm{PW} \times \mathrm{PW}$ as an extension of the Nerode right congruence by $\alpha \mathbin{\overline{\approx}}_{\mathcal{L}} \beta$ iff $nf(\alpha) \equiv_{\mathcal{L}} nf(\beta)$.

Definition 3 (Regular MSC Word Language) *An MSC word language $\mathcal{L}$ is called* regular *iff $\mathbin{\overline{\approx}}_{\mathcal{L}}$ has finite index.*

Theorem 1 *Let $\mathcal{L}$ be an MSC word language. $\mathcal{L}$ is regular iff $\{nf(\alpha) \mid \alpha \in \mathcal{L}\}$ is a regular word language over $\Sigma \times Q$ where Q is a finite subset of $\mathbb{N}$.*

We can finally conclude that regular MSC word languages are bounded and closed under union, intersection, concatenation, and Kleene star.

MSC Finite-Memory Automata We now present a class of automata which characterises MSC word languages.

Definition 4 *An* MSC finite-memory automaton (MFA) *is a quintuple of the form $\mathcal{A} = (S, r, \Delta, q_0, F)$, where S is a nonempty finite set of* states*, $r \geq 1$ is a natural number called* window length*, $\Delta \subseteq S \times (\Sigma \times \{1, \ldots, r\}) \times S$ is the* transition relation*, $q_0 \in S$ is the* initial state*, and $F \subseteq S$ is the set of* final states*.*

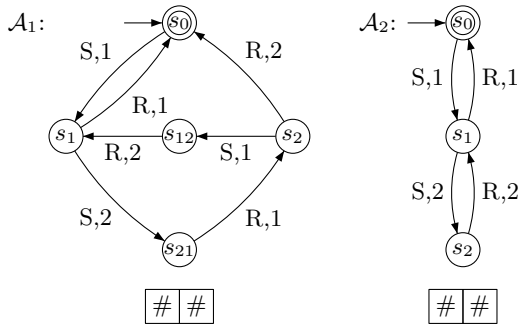


Figure 4: Two MFAs

Figure 4 shows two automata, each with a window for two elements. Let thereby S stand for $\mathrm{S}_1^2(a)$ and R for $\mathrm{R}_1^2(a)$.

Let $\mathcal{A}$ be an MFA as above. A *configuration* of $\mathcal{A}$ lists the current state and the current window entries which are either numbered send events or empty (denoted by #). The meaning of a transition $(s, (\mathrm{S}_p^q(\lambda), k), t)$ is the following (for details, see [2]): if $\mathcal{A}$ is in state s, it is able to read an input symbol $(\mathrm{S}_p^q(\lambda), \pi)$, $\pi \in \mathbb{N}$, iff the kth position of its window is currently free and, furthermore, $(\mathrm{S}_p^q(\lambda), \pi)$ does not occur elsewhere in the window, i. e. there is no further open $(\mathrm{S}_p^q(\lambda), \pi)$-labelled send event. Taking the transition, the automaton stores $(\mathrm{S}_p^q(\lambda), \pi)$ in the kth position and enters state t. If, in contrast, the automaton reads an input symbol $(\mathrm{R}_p^q(\lambda), \pi)$, there has to be a transition $(s, (\mathrm{R}_p^q(\lambda), k), t)$ such that the kth position of the window currently shows the corresponding send symbol $(\mathrm{S}_p^q(\lambda), \pi)$. Replacing this symbol with #, the automaton enters state t.

An accepting run is a corresponding sequence of configurations starting in q_0 with the empty window and ending in a final state with the empty window. Note that $\mathcal{L}(\mathcal{A})$, the language of $\mathcal{A}$, is an MSC word language.

Theorem 2 *An MSC word language $\mathcal{L}$ is regular iff there is an MFA $\mathcal{A}$ such that $\mathcal{L} = \mathcal{L}(\mathcal{A})$.*

Theorem 3 *For MFAs $\mathcal{A}_1$ and $\mathcal{A}_2$, it is decidable whether $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$ and whether $\mathcal{L}(\mathcal{A}_1)$ is empty.*

4. A Logical Characterisation

Extending our theory, a collection $\mathcal{M}$ of MSCs is called a *regular MSC language* iff $Lin(\mathcal{M})$ is a regular MSC word language. We formulate a monadic second-order logic which characterises

exactly the class of regular MSC languages applying Büchi's famous result [3]. Given a supply $\mathrm{Var} = \{x, y, \ldots\}$ of individual variables and a supply $\mathrm{VAR} = \{X, Y, \ldots\}$ of set variables, the syntax of $\mathrm{MSO}(\mathcal{P}_N, \Lambda)$ is defined according to

$$\varphi ::= L_\sigma(x) \mid x \in X \mid x \preceq y \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x\varphi \mid \exists X\varphi \in \mathrm{MSO}(\mathcal{P}_N, \Lambda)$$

where $\sigma \in \Sigma$, $x \in \mathrm{Var}$, and $X \in \mathrm{VAR}$. The satisfaction relation $M \models \varphi$ for a formula $\varphi \in \mathrm{MSO}(\mathcal{P}_N, \Lambda)$ is defined as one might expect.

Theorem 4 *Given a collection $\mathcal{M}$ of MSCs, $\mathcal{M}$ is a regular MSC language iff there exist a formula $\varphi \in \mathrm{MSO}(\mathcal{P}_N, \Lambda)$ and $B \in \mathbb{N}$ such that $Lin(\mathcal{M}) = Lin(\mathcal{M}_\varphi^B)$ where $\mathcal{M}_\varphi^B := \{M \mid M$ is B-bounded and $M \models \varphi\}$.*

References

[1] Benedikt Bollig and Martin Leucker. Modelling, specifying, and verifying message passing systems. In Claudio Bettini and Angelo Montanari, editors, *Proceedings of the Symposium on Temporal Representation and Reasoning (TIME '01)*, pages 240–248. IEEE Computer Society Press, June 2001.

[2] Benedikt Bollig, Martin Leucker, and Thomas Noll. Regular MSC Languages. Technical Report AIB-05-2001, RWTH Aachen, April 2001.

[3] J. Büchi. Weak second order logic and finite automata. *Z. Math. Logik, Grundlag. Math.*, 5:66–62, 1960.

[4] J. G. Henriksen, M. Mukund, K. Narayan Kumar, and P. S. Thiagarajan. Regular collections of message sequence charts. In *Proceedings of 25th International Symposium on Mathematical Foundations of Computer Science (MFCS '00)*, volume 1893 of *Lecture Notes in Computer Science*, pages 405–414. Springer-Verlag, 2000.

[5] ITU-TS. ITU-TS Recommendation Z.120: Message Sequence Chart 1999 (MSC '99). Technical report, ITU-TS, Geneva, 1999.

[6] Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, November 1994.

DECISION PROBLEMS ON THE POWER OF LANGUAGES

Henning Bordihn
Institut für Informatik, Universität Potsdam
Postfach 90 03 27, D-14439 Potsdam, Germany
e-mail: henning@cs.uni-potsdam.de

ABSTRACT

The power of a language L is the set of all powers of the words in L. In this paper, the following decision problem is investigated. Given a context-free language L, is the power of L context-free? We show that this problem is decidable for languages over unary alphabets, but it is undecidable whenever languages over alphabets with at least three letters are considered. The problem remains open for languages over binary alphabets.

1. Introduction and Preliminaries

An important class of decision problems in the theory of formal languages can be stated as follows. Fix a (unary) operation on languages and a family of languages, is it decidable, given a language from this family, whether or not the application of the operation to the given language leads out of this family? In this paper, we study this problem for the family of context-free languages with respect to the power operation.

For any language L, the *power of* L is the set

$$\begin{aligned}\mathrm{pow}(L) &= \{\, w^i \mid i \geq 0,\, w \in L \,\} \\ &= \bigcup_{w \in L} w^*.\end{aligned}$$

Clearly, $\mathrm{pow}(L)$ is a subset of $L^* = \bigcup_{i \geq 0} L^i$. The family of context-free languages is not closed under the power operation. Consider, e. g., the language defined by the rational expression a^+b, its power is the non-context-free language $\bigcup_{i>0}(a^ib)^*$. Therefore, it is natural to search for some algorithm deciding whether $\mathrm{pow}(L)$ is context-free if a given language L is context-free.

This problem was first mentioned for regular languages in [2] and partially answered by Cachat in [1] proving that there is an algorithm deciding whether the power of a unary regular language is regular.

Theorem 1 *For a given regular language L over a one-letter alphabet, one can decide algorithmically whether* $\mathrm{pow}(L)$ *is regular.*

The problem for regular languages over arbitrary finite alphabets is left open. For the upper classes of the Chomsky hierarchy, i. e., the family of recursively enumerable, recursive, and context-sensitive languages, the problem is quite trivial, because each of these families is closed under the power operation (what can easily be shown by appropriate machine constructions). In this paper, we aim to settle the problem for the family of context-free languages.

The reader is assumed to be familiar with basic concepts of formal language theory as contained, e. g., in [3] or [4]. Concerning our notation, we have the following conventions:

V^+ denotes the set of nonempty words over alphabet V; if the empty word λ is included, then we use the notation V^*. The mirror image of a string w is denoted by w^R, its length by $|w|$. Generally, for a singleton set $\{a\}$ we simply write a.

A context-free grammar is a four tuple $G = (N, T, P, S)$, where N and T are disjoint alphabets of nonterminals and terminals, respectively, $S \in N$ is the axiom, and P is a finite set of productions of the form $A \to u$, where $A \in N$ and $u \in (N \cup T)^*$. A production with A on its left-hand side is referred to as A-production of G.

2. Context-Free Languages

Before giving the proof of our main result that context-freeness of the power of context-free languages is undecidable, we mention a direct consequence of Theorem 1. Since any context-free language over a one-letter alphabet is regular, we immediately get the following result.

Corollary 2 *For a given context-free language L over a one-letter alphabet, one can decide algorithmically whether* $\mathrm{pow}(L)$ *is regular (and therefore context-free).*

Now, we show that there is no adequate algorithm for context-free languages over finite alphabets with at least three letters.

Theorem 3 *For a given context-free language over an alphabet with at least three letters, it is undecidable whether* $\mathrm{pow}(L)$ *is context-free.*

Proof. We proof this undecidability by reducing Post's Correspondence Problem (PCP).

Let $\{(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)\}$ be a finite set of pairs of $\{a, b\}^+ \times \{a, b\}^+$, an instance of the PCP. We consider the context-free grammar

$$G = (\{S, S', A, B, C, D, E\}, \{a, b, \#\}, P, S)\,,$$

where P is the union of the following sets of productions:

$$\begin{aligned}
P_1 &= \{S \to S', S \to E\#\#, S' \to A\#\#S', S' \to \lambda\},\\
P_2 &= \{A \to aAa, A \to bAb\},\\
P_3 &= \{A \to aBb, A \to bBa\} \cup \{\, B \to xBy \mid x, y \in \{a, b\} \,\},\\
P_4 &= \{A \to aC, A \to bC, B \to C\} \cup \{C \to aC, C \to bC\},\\
P_5 &= \{A \to Da, A \to Db, B \to D\} \cup \{D \to Da, D \to Db\},\\
P_6 &= \{B \to \#, C \to \#, D \to \#\},\\
P_7 &= \{\, E \to u_i E v_i^R \mid 1 \le i \le n \,\} \cup \{\, E \to u_i \# v_i^R \mid 1 \le i \le n \,\}.
\end{aligned}$$

The derivation process starts off with an application of P_1 generating either $(A\#\#)^j$, for some $j \ge 0$, or $E\#\#$. Any occurrence of A can be replaced by productions in P_2 followed by exactly one A-production in P_3, P_4 or P_5. Note that there is no other way to get rid of the nonterminal A. After further derivation steps using productions from P_3, P_4 or P_5, from any single A a string of one of the following forms is obtained:

- $z_1 B z_2$ with $z_1, z_2 \in \{a, b\}^+$ and $|z_1| = |z_2|$, $z_2 \neq z_1^R$,
- $z_1 C z_2$ with $z_1, z_2 \in \{a, b\}^+$ and $|z_1| > |z_2|$,
- $z_1 D z_2$ with $z_1, z_2 \in \{a, b\}^+$ and $|z_1| < |z_2|$.

Therefore, starting off the derivation with $S \to S'$, the language L_0^* with

$$L_0 = \{\, z_1 \# z_2 \# \# \mid z_1, z_2 \in \{a, b\}^+,\ z_2 \neq z_1^R \,\}$$

is obtained. On the other hand, using $S \to E\#\#$ first, the language

$$L_1 = \{\, u_{i_1} u_{i_2} \dots u_{i_m} \# v_{i_m}^R \dots v_{i_2}^R v_{i_1}^R \#\# \mid m \geq 1,\ 1 \leq i_j \leq n \text{ for } 1 \leq j \leq m \,\}$$

is generated. Thus, we have $L(G) = L_0^* \cup L_1$. Note that the grammar G' which is obtained from G by omitting the production $S \to E\#\#$ generates the language L_0^*.

Assume that the PCP does *not* have a solution for the given instance. Then $L_1 \subseteq L_0$ holds. Hence $L(G) = L_0^*$ in this case. Since $\mathrm{pow}(L_0^*) = L_0^*$ and L_0^* is context-free, $\mathrm{pow}(L(G))$ is context-free if there is no solution for the instance of the PCP.

On the other hand, if there *is* a solution, then there exist infinitely many words of the form $z\#z^R\#\#$ in $L(G)$. Therefore, we can prove $\mathrm{pow}(L(G))$ is not context-free as follows. Assume the contrary, and let k be the constant from Bar-Hillel's pumping lemma. Choose a solution $i_1 i_2 \dots i_l$ of the PCP such that $|u_{i_1} u_{i_2} \dots u_{i_l}| > k$. Let $z = u_{i_1} u_{i_2} \dots u_{i_l}$ and consider the word $\alpha = z\#z^R\#\#z\#z^R\#\#z\#z^R\#\#$, an element of the language $\mathrm{pow}(L(G))$. By standard arguments, the pumping on α can be performed only in a way such that at most two (consecutive) substrings of the form $z\#z^R$ are modified and, therefore, at least one of them remains unchanged. This yields a string of the form $w_1\#\#w_2\#\#w_3\#\#$ where, w_1 is still of the form $z\#z^R$ but w_2 or w_3 is different from w_1, or $w_3 = z\#z^R$ but w_1 or w_2 is different. Obviously, a string with this property belongs neither to L_0^* nor to $\mathrm{pow}(L_1)$, thus it does not belong to $\mathrm{pow}(L(G))$, a contradiction.

Since Post's Correspondence Problem is undecidable, the undecidability of the context-freeness of the power of context-free languages follows from our construction. □

3. Conclusion

The problem to decide whether the power of a given context-free language is context-free is shown to be undecidable in the general case, more precisely, when context-free languages over alphabets with at least three letters are considered. If one restricts to unary alphabets, the problem becomes decidable since, in this case, the question is equivalent to the problem for regular languages over unary alphabets which has been solved in [1]. The decidability status in case of context-free languages over binary alphabets remains unknown.

Since the analogous problem is trivial for the classes in the Chomsky hierarchy beyond the family of context-free languages, there remains a gap between unary regular and arbitrary context-free languages which can be expressed in the open question which of the problems summarized below is algorithmically decidable.

- Given a binary context-free language L, is $\mathrm{pow}(L)$ context-free?
- Given a non-unary regular language L, is $\mathrm{pow}(L)$ regular?
- Given a linear context-free language L, is $\mathrm{pow}(L)$ linear context-free?

References

[1] T. Cachat. The power of one-letter rational languages. In W. Kuich, editor, *Develeopments in Language Theory, 5^{th} International Conference, DLT 2001, Preproceedings*, pages 135–146. Institut für Algebra und Comptermathematik, Technische Universität Wien, 2001.

[2] H. Calbrix. *Mots ultimement périodiques des langages rationnels de mots infinis.* PhD thesis, Université Denis Diderot-Paris VII, 1996.

[3] A. Salomaa. *Formal Languages.* Springer, 1973.

[4] D. Wood. *Theory of Computation.* John Wiley & Sons, 1987.

EVEN LINEAR SIMPLE MATRIX LANGUAGES: FORMAL LANGUAGE PROPERTIES AND GRAMMATICAL INFERENCE

Henning Fernau
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
Sand 13, D-72076 Tübingen, Germany
e-mail: fernau@informatik.uni-tuebingen.de

ABSTRACT

We show that so-called deterministic even linear simple matrix grammars can be inferred in polynomial time using the query-based learner-teacher model MAT proposed by Angluin for learning deterministic regular languages in [2]. In this way, we extend the class of efficiently learnable languages beyond both the even linear languages and the even equal matrix languages proposed in [19, 20, 23, 24, 25, 26]. Moreover, we investigate formal language properties of even linear simple matrix languages and related language classes. More precisely, we discuss characterizations, (proper) inclusion relations, closure properties and decidability questions. This way, we also show that, in a certain sense, the idea of iterating the control language approach for learning purposes, as undertook by Takada [25, 26], could be seen as a special case of using deterministic even linear simple matrix grammars as basic and uniform learning target.

Keywords: learning, grammatical inference, MAT model, formal languages, language characterizations, control language, Khabbaz hierarchies, simple matrix grammars.

1. Introduction

Machine Learning and Formal Language Theory are two topics of applied and theoretical computer science whose common ground seems to be very small at first glance. Interestingly, there is indeed a very vivid common area of research, namely, what is called grammatical inference. Grammatical inference deals with the automatic learning of grammars, automata and other language describing devices. The "disadvantage" of such an interdisciplinary area is that practitioners will probably find lengthy formal language arguments boring, whilst pure formal language theorists might think that details on learning algorithms are uninteresting. Nevertheless, in the following, we try to satisfy both parts of the prospective readership of this paper, since we feel that the interdependencies between both areas are rather strong: on the one hand, we cannot formally establish the correctness of the proposed learning algorithm(s), neither can we reason about connections to other published algorithms nor can we talk about limitations of the considered methodologies without making heavy use of formal language arguments; on the other hand, the basic motivation for investigating formal language properties of even linear simple matrix languages stirs from their learnability.

The remainder of this introduction is split into two parts: one is meant for the reader who is mainly interested in the learning aspects of this paper, while the second part is the formal language theorists' introduction.

1.1. Learning Aspects

Learning languages using a teacher-learner-dialogue (also known as MAT – minimally adequate teacher – learning model or as *query learning*) has become popular since Angluin published a polynomial-time learning algorithm for regular languages [2] (or, more precisely, deterministic finite automata). The limits of the model were explored in [3, 4]. One of the questions arising from Angluin's result is whether it can be extended beyond regular sets. For example, learning algorithms for systolic automata [29] were exhibited.

Radhakrishnan and Nagaraja [19] proposed even linear languages for learning theoretical purposes by giving a skeleton-based inference algorithm and showing possible applications in the area of inference of pictures. Both the work of Takada [23] and that of Sempere and García [20] showed how the learning problem of even linear languages (introduced in [1]) could be reduced to the learning of regular languages. Moreover, Takada and his colleagues proved the usefulness of the concept of control languages (originating from [9]) in the reduction of the learning problem of languages defined via controlled fixed grammars [13, 23, 24, 25, 26]. In particular, Takada used this concept to develop an efficient learning algorithm of what he called "even equal matrix languages" [24, 26].

As mentioned above, control languages are a natural tool for transferring Angluin's learnability result to, e. g., even equal matrix languages or even linear languages. Indeed, the learning problem for those classes is reduced to the learning problem of (regular) control languages by using universal grammar normal forms. Obviously, it is now possible to use, e. g., even linear languages as control languages for universal even linear grammars, hence obtaining a whole hierarchy (similar to those of Khabbaz [12, 11]) of efficiently learnable language classes by iterating the argument sketched above. Takada's papers [25, 26] explore the learnability of levels of such hierarchies.

Here, we show how to learn what we call even linear simple matrix languages of arbitrary degree. In doing this, we extend Takada's previous results in two ways:

- Even equal matrix languages are a subset of even linear matrix languages.
- The Khabbaz/Takada hierarchy of even linear languages controlled by even linear languages and so forth is contained in the even linear matrix languages. More precisely, we show that even linear matrix languages controlled by even linear matrix languages yield even linear matrix languages, so that a further Khabbaz/Takada-like hierarchy extension of the efficiently learnable language classes is not possible.

The MAT learnability results were also presented at the conference COCOON '99 [7].

1.2. Formal Language Aspects

Linear simple matrix languages were introduced by Păun [18]. We will investigate formal language aspects of a restricted class of linear simple matrix languages which we call *even* linear simple matrix languages; they are an important class from the viewpoint of grammatical inference [7]. Further results on (even) linear simple matrix languages can be found in [15] and [6, p. 68 ff.]. Linear simple matrix languages naturally fall inbetween right-linear and context-free simple matrix languages, see [6, 10, 17, 22]. Note that right-linear simple matrix language were introduced by Siromoney [22] as equal matrix languages. Equivalent formalizations can be found in [5] and [16]. In [27], equal matrix languages and linear languages are compared. In particular, all such languages are semilinear.

Intriguingly, there may be another source of interest in the mentioned language families: Weir showed in [28] how the Khabbaz hierarchy [12] can be generalized in order to characterize tree adjoined languages, which play a prominent role in computer linguistics. A future possibility might be that those (formal) connections could lead to programs which are able to assist linguists

who are designing grammars for natural languages, e. g., by creating proposals of such grammars automatically (in the sense of learning theory) or by certifying optimality conditions on these grammars, e. g., a minimal number of variables in the case of "deterministic grammars". Moreover, the class of even linear simple matrix languages we propose for learning-theoretic purposes contains both typical "pushdown languages" and typical "queue languages". This observation should be of particular interest when considering possible linguistic applications of learning theory, since natural languages typically consist of both parenthesis structures (as exemplified by relative clauses) and "copy-structures" (as can be found in Swiss German), cf. [21].

Such research might help close the "undesirable gap between the communities of linguists and computer scientists, more specifically the communities of computational linguists and formal language theoreticians" observed by Martín-Vide in [14].

The formal language part was presented at the DMTCS '01 conference [8]. The complete paper will appear in *Theoretical Computer Science*.

We only provide some definitions in the following:

2. Definitions

Definition *A* linear simple matrix grammar of degree n, $n \geq 1$, *is* (*cf.* [18]) *an* $(n+3)$-*tuple* $G = (V_1, \ldots, V_n, \Sigma, M, S)$, *where* $\{S\}, V_1, \ldots, V_n, \Sigma$ *are pairwise disjoint alphabets* ($V_N = \bigcup_{i=1}^{n} V_i \cup \{S\}$ *contains the nonterminals and* Σ *the terminals*), *and* M *is a finite set of matrices of the form*

1. $(S \to A_1 \ldots A_n)$, *for* $A_i \in V_i$, $1 \leq i \leq n$,
2. $(A_1 \to x_1, \ldots, A_n \to x_n)$, *for* $A_i \in V_i$, $x_i \in \Sigma^*$, $1 \leq i \leq n$, *or*
3. $(A_1 \to x_1 B_1 y_1, \ldots, A_n \to x_n B_n y_n)$, $A_i, B_i \in V_i$, $x_i, y_i \in \Sigma^*$, $1 \leq i \leq n$.

Matrices of the form 1.–3. *are called* initial matrices, terminal matrices, *and* nonterminal matrices, *respectively.*

We now define three restrictions on such grammars:

- G *is a* right-linear simple matrix grammar *if the nonterminal matrices satisfy*

  3′. $(A_1 \to x_1 B_1, \ldots, A_n \to x_n B_n)$, $A_i, B_i \in V_i$, $x_i \in \Sigma^*$, $1 \leq i \leq n$.

- G *is an* even linear simple matrix grammar *if the nonterminal matrices satisfy*

  3″. $(A_1 \to x_1 B_1 y_1, \ldots, A_n \to x_n B_n y_n)$, *for* $A_i, B_i \in V_i$, $x_i, y_i \in \Sigma^*$ *such that* $|x_i| = |y_j|$ *for all* $1 \leq i, j \leq n$.

- G *is an* even right-linear simple matrix grammar (*or* even equal matrix grammar *as called by Takada* [24]) *if the nonterminal matrices satisfy*

  3‴. $(A_1 \to x_1 B_1, \ldots, A_n \to x_n B_n)$, $A_i, B_i \in V_i$, $x_i \in \Sigma^*$ *such that* $|x_1| = |x_i|$ *for all* $2 \leq i \leq n$.

Let $V_G = V_N \cup \Sigma$. *For* $x, y \in V_G^*$, *we write* $x \Rightarrow y$ *iff either (i)* $x = S$, $(S \to y) \in M$, *or (ii)* $x = u_1 A_1 v_1 \ldots u_n A_n v_n$, $y = u_1 w_1 v_1 \ldots u_n w_n v_n$, *and* $(A_1 \to w_1, \ldots, A_n \to w_n) \in M$. *As usual, define* $L(G) = \{x \in \Sigma^* \mid S \overset{*}{\Rightarrow} x\}$, *where* $\overset{*}{\Rightarrow}$ *is the reflexive transitive closure of relation* $\Rightarrow$.

The families of linear simple matrix grammars of degree n, right-linear simple matrix grammars of degree n, even linear simple matrix grammars of degree n, and even right-linear simple matrix grammars of degree n, as well as the corresponding language families, are denoted by $\mathrm{SL}(n)$, $\mathrm{SRL}(n)$, $\mathrm{ESL}(n)$ *and* $\mathrm{ESRL}(n)$, *respectively. Sometimes, we use notations like* $\mathrm{ESL}(n, m)$ *in order to specify the terminal alphabet* Σ_m *explicitly. When not needed, we sometimes omit the explicit reference to the degree, which leads to classes like* $\mathrm{SL} = \bigcup_{n \geq 1} \mathrm{SL}(n)$. *Specifically,*

ESRL(1) = SRL(1) *denotes the regular languages,* ESL(1) *the even linear languages, and* SL(1) *the linear languages.*

In the following, we give three examples (which are the common languages used by linguists to prove that natural languages are not context-free) to show the power of the mechanisms.

Example Consider $G_1 = (\{A_1\}, \{A_2\}, \{a, b\}, M_1, S)$, where M_1 contains the following matrices:

1. $(S \to A_1A_2)$,
2. $(A_1 \to \lambda, A_2 \to \lambda)$,
3. $(A_1 \to aA_1, A_2 \to aA_2), (A_1 \to bA_1, A_2 \to bA_2)$.

G_1 is an ESRL(2) grammar which generates $L(G_1) = \{ww \mid w \in \{a, b\}^*\}$.

Example Consider $G_2 = (\{A_1, B_1\}, \{A_2, B_2\}, \{a, b\}, M_2, S)$, where M_2 contains the following matrices:

1. $(S \to A_1A_2)$,
2. $(A_1 \to \lambda, A_2 \to \lambda)$, $(B_1 \to \lambda, B_2 \to \lambda)$,
3. $(A_1 \to aA_1, A_2 \to aA_2), (A_1 \to bB_1, A_2 \to bB_2), (B_1 \to bB_1, B_2 \to bB_2)$.

$G_2 \in$ ESRL(2) generates $L(G_2) = \{a^nb^ma^nb^m \mid n, m \geq 0\}$.

Example Consider $G_3 = (\{A_1\}, \{A_2\}, \{A_3\}, \{a, b\}, M_3, S)$, where M_3 contains the following matrices:

1. $(S \to A_1A_2A_3)$,
2. $(A_1 \to \lambda, A_2 \to \lambda, A_3 \to \lambda)$,
3. $(A_1 \to aA_1, A_2 \to bA_2, A_3 \to aA_3)$.

G_3 is an ESRL(3) grammar which generates $L(G_3) = \{a^nb^na^n \mid n \geq 0\}$.

References

[1] V. Amar and G. Putzolu. On a family of linear grammars. *Information and Control*, 7:283–291, 1964.

[2] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[3] D. Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990.

[4] D. Angluin and M. Kharitonov. When won't membership queries help? In *Proc. 23rd ACM Symp. Theory of Computing STOC*, pages 444–454, 1991.

[5] J. A. Brzozowski. Regular-like expressions for some irregular languages. In *IEEE Conf. Record of 9th Ann. Symp. on Switching and Automata Theory SWAT*, pages 278–280, 1968.

[6] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *EATCS Monographs in Theoretical Computer Science*. Berlin: Springer, 1989.

[7] H. Fernau. Efficient learning of some linear matrix languages. In T. Asano et al., editors, *COCOON '99*, volume 1627 of *LNCS*, pages 221–230, 1999.

[8] H. Fernau. Even linear simple matrix languages: formal language aspects. In C.S. Calude, M.J. Dinneen, and S. Sburlan, editors, *Discrete Mathematics and Theoretical Computer Science , DMTCS '01*, pages 81–96, 2001.

[9] S. Ginsburg and E. H. Spanier. Control sets on grammars. *Mathematical Systems Theory*, 2:159–177, 1968.

[10] O. Ibarra. Simple matrix languages. *Information and Control*, 17:359–394, 1970.

[11] N. A. Khabbaz. Control sets on linear grammars. *Information and Control*, 25:206–221, 1974.

[12] N. A. Khabbaz. A geometric hierarchy of languages. *Journal of Computer and System Sciences*, 8:142–157, 1974.

[13] E. Mäkinen. A note on the grammatical inference problem for even linear languages. *Fundamenta Informaticae*, 25:175–181, 1996.

[14] C. Martin-Vide. Natural language understanding: a new challenge for grammar systems. *Acta Cybernetica*, 12:461–472, 1996.

[15] A. Mateescu. Special families of matrix languages and decidable problems. *Acta Cybernetica*, 10:45–52, 1991.

[16] H. A. Maurer and W. Kuich. Tuple languages. In W. D. Itzfeld, editor, *Proc. of the ACM International Computing Symposium*, pages 882–891. German Chapter of the ACM, 1970.

[17] A. Pascu and Gh. Păun. On simple matrix grammars. *Bull[etin] Math. de la Soc. Sci. Math. [de la R. S.] de Roumanie*, 20:333–340, 1976.

[18] Gh. Păun. Linear simple matrix languages. *Elektronische Informationsverarbeitung und Kybernetik* (*EIK*), 14:377–384, 1978.

[19] V. Radhakrishnan and G. Nagaraja. Inference of even linear grammars and its application to picture description languages. *Pattern Recognition*, 21:55–62, 1988.

[20] J. M. Sempere and P. García. A characterization of even linear languages and its application to the learning problem. In R. C. Carrasco and J. Oncina, editors, *Proceedings of the Second International Colloquium on Grammatical Inference* (*ICGI-94*)*: Grammatical Inference and Applications*, volume 862 of *LNCS/LNAI*, pages 38–44, Berlin, 1994. Springer.

[21] S. M. Shieber. Evidence against the context-freeness of natural languages. *Linguistics and Philosophy*, 8:333–343, 1985.

[22] R. Siromoney. On equal matrix languages. *Information and Control*, 14:133–151, 1969.

[23] Y. Takada. Grammatical inference of even linear languages based on control sets. *Information Processing Letters*, 28:193–199, 1988.

[24] Y. Takada. Learning even equal matrix languages based on control sets. In A. Nakamura et al., editors, *Parallel Image Analysis, ICPIA '92*, volume 652 of *LNCS*, pages 274–289, 1992.

[25] Y. Takada. A hierarchy of language families learnable by regular language learning. *Information and Computation*, 123:138–145, 1995.

[26] Y. Takada. Learning formal languages based on control sets. In K. P. Jantke and S. Lange, editors, *Algorithmic Learning for Knowledge-Based Systems*, volume 961 of *LNCS/LNAI*, pages 317–339, 1995.

[27] K. W. Wagner. On the intersection of the class of linear context-free languages and the class of single-reset languages. *Information Processing Letters*, 23:143–146, 1986.

[28] D. J. Weir. A geometric hierarchy beyond context-free languages. *Theoretical Computer Science*, 104:235–261, 1992.

[29] T. Yokomori. On learning systolic languages. In K. P. Jantke S. Doshita, K. Furukawa and T. Nishida, editors, *Proceedings of the 3rd Workshop on Algorithmic Learning Theory* (*ALT '92*), volume 743 of *LNCS/LNAI*, pages 41–52, Tokyo, Japan, October 1992. Springer.

DECIDABILITY OF CODE PROPERTIES

Henning Fernau, Klaus Reinhardt
Wilhelm-Schickhard-Institut für Informatik, Universität Tübingen
Sand 13, D-72076 Tübingen, Germany
e-mail: {fernau,reinhard}@informatik.uni-tuebingen.de

and

Ludwig Staiger
Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg
Kurt-Mothes-Str. 1, D-06120 Halle, Germany
e-mail: staiger@informatik.uni-halle.de

We explore the borderline between decidability and undecidability of the following question: "Let $\mathcal{C}$ be a class of codes. Given a machine $\mathfrak{M}$ of type X, is it decidable whether the language $L(\mathfrak{M})$ lies in $\mathcal{C}$ or not?" for codes in general, ω-codes, codes of finite and bounded deciphering delay, prefix, suffix and bi(pre)fix codes, and for finite automata equipped with different versions of push-down stores and counters.

Variablenkomplexität in graphkontrollierten, programmierten und Matrix-Grammatiken

Rudolf Freund
Institut für Computersprachen, Technische Universität Wien
Favoritenstraße 9, A-1040 Wien, Austria
e-mail: rudi@logic.at, rudi@emcc.at

und

Gheorghe Păun
Institut für Mathematik, Rumänische Akademie der Wissenschaften
PO Box 1-764, RO-70700 Bukarest, Rumänien
e-mail: gpaun@imar.ro, g_paun@hotmail.com, gp@astor.urv.es

ABSTRACT

Wir zeigen, dass das Resultat aus [7] betreffend die Anzahl der Variablen in graphkontrollierten, programmierten und Matrix-Grammatiken (mit Vorkommenstest) noch wesentlich verbessert werden kann: Um jede beliebige rekursiv aufzählbare Sprache erzeugen zu können, braucht man bei graphkontrollierten (bzw. programmierten) Grammatiken nur 3 Variablen (bzw. 4), davon nur 2 verwendet mit Vorkommenstest, bei Matrixgrammatiken 4 Variablen, davon 3 mit Vorkommenstest. Hingegen genügen 2 Variablen, die mit Vorkommenstest verwendet werden, in Matrixgrammatiken nur dann, wenn man die Gesamtzahl der Variablen nicht beschränkt.

Schlagwörter: graphkontrollierte Grammatiken, Matrix-Grammatiken, programmierte Grammatiken, Variablenkomplexität.

1. Einleitung

Die in dieser Arbeit (einer Kurzfassung des heuer bei MCU 2001 präsentierten Artikels, s. [4]) vorgestellten Ergebnisse verbessern nicht nur Ergebnisse (s. [7]) aus dem Bereich von Grammatiken mit Kontrollmechanismen, sondern sind im Speziellen auch für das neue Gebiet der Membransysteme (P-Systeme, s. [1, 8]) von Bedeutung; in vielen Fällen geht der Beweis der Universalität dieser Systeme von Matrix-Grammatiken aus, wobei die Variablenkomplexität der Matrix-Grammatiken (im Speziellen ist meist die Anzahl der mit Vorkommenstest verwendeten Variablen von Bedeutung) oft direkt mit der Anzahl der benötigten Membranen korreliert (s. [5]).

In [7] wurde gezeigt, dass zur Erzeugung einer rekursiv aufzählbaren Sprache Matrix-Grammatiken mit kontextfreien Produktionen und 6 Variablen genügen, die allerdings auch alle mit Vorkommenstest verwendet wurden. Wir beweisen nun, dass 4 Variablen, davon drei verwendet mit Vorkommenstest, genügen. Verwendet man graphkontrollierte Grammatiken, so genügen sogar 3 Variablen, 2 davon verwendet mit Vorkommenstest. Die Beweise beruhen auf einem seit langem bekannten Ergebnis, z. B. wurde bereits in [6] gezeigt, dass die Aktionen einer Turingmaschine durch eine Registermaschine mit nur zwei Registern simuliert werden können; die Aktionen von Registermaschinen können ihrerseits sehr leicht durch graphkontrollierte Grammatiken simuliert werden.

Etwa zur gleichen Zeit, als diese Arbeit enstand, kam Henning Fernau mittels einer ganz anderen Beweistechnik (einer trickreichen Turingmaschinen-Simulation) zu den gleichen Ergebnissen, was die Gesamtanzahl der Variablen betrifft (s. [3], ebenfalls präsentiert bei MCU 2001), wobei jedoch stets alle Variablen auch mit Vorkommenstest verwendet wurden. Für programmierte Grammatiken zeigt Henning Fernau, dass wie bei graphkontrollierten Grammatiken 3 Variablen genügen, während wir, um nur 2 Variablen mit Vorkommenstest verwenden zu müssen, in diesem Falle 4 Variablen benötigen (obwohl zukünftige Untersuchungen durchaus ergeben könnten, dass ebenfalls eine Gesamtzahl von 3 Variablen reicht).

2. Definitionen

Die Menge der nichtnegativen ganzen Zahlen wird mit $\mathbb{N}_0$ bezeichnet.

Wie in [2] gezeigt, erlaubt jede der im Folgenden definierten Varianten von Grammatiken mit Kontrollmechanismen, mit kontextfreien Produktionen jede beliebige rekursiv aufzählbare formale Sprache zu erhalten.

Eine *graphkontrollierte Grammatik* ist ein Konstrukt $G_C = (N, T, (R, L_{in}, L_{fin}), S)$; N ist das Variablenalphabet, T das Terminalalphabet, $N \cap T = \emptyset$; $S \in N$ ist das Startsymbol; R ist eine endliche Menge von Regeln r der Gestalt $(l(r) : p(l(r)), \sigma(l(r)), \varphi(l(r)))$, wobei $l(r) \in Lab(G_C)$; $Lab(G_C)$ ist eine Menge von Markierungen, die in einer Eins-zu-Eins-Relation den Regeln r in R zugeordnet sind; $p(l(r))$ ist eine kontextfreie Produktion über $N \cup T$, $\sigma(l(r)) \subseteq Lab(G_C)$ ist das *Erfolgsfeld* der Regel r, und $\varphi(l(r))$ ist das *Misserfolgsfeld* der Regel r; $L_{in} \subseteq Lab(G_C)$ ist die Menge der *Anfangsmarkierungen* und $L_{fin} \subseteq Lab(G_C)$ ist die Menge der *Endmarkierungen*. Für $r = (l(r) : p(l(r)), \sigma(l(r)), \varphi(l(r)))$ und $v, w \in (N \cup T)^*$ definieren wir $(v, l(r)) \Longrightarrow_{G_C} (w, k)$ genau dann, wenn

- **entweder** $p(l(r))$ auf v anwendbar und das Ergebnis der Anwendung der Produktion $p(l(r))$ auf v dann w ist sowie $k \in \sigma(l(r))$,
- **oder** $p(l(r))$ nicht auf v anwendbar ist, $w = v$ sowie $k \in \varphi(l(r))$.

Die von G_C erzeugte Sprache ist

$$L(G_C) = \{w \in T^* \mid (S, l_0) \Longrightarrow_{G_C} (w_1, l_1) \Longrightarrow_{G_C} \cdots \Longrightarrow_{G_C} (w_k, l_k),\ k \geq 1, \\ w_j \in (N \cup T)^* \text{ und } l_j \in Lab(G_C) \text{ für } 0 \leq j \leq k, \\ w_0 = S,\ w_k = w,\ l_0 \in L_{in},\ l_k \in L_{fin}\}.$$

Eine Variable $A \in N$ wird *mit Vorkommenstest verwendet,* wenn zumindest eine Regel $(l : p, \sigma(l), \varphi(l)) \in R$ existiert, sodass p von der Form $A \to \alpha$ mit $\alpha \in (N \cup T)^*$ ist und $\varphi(l) \neq \emptyset$.

Eine *programmierte Grammatik* ist ein Konstrukt $G_P = (N, T, R, S)$, sodass $G_C = (N, T, (R, Lab(G_C), Lab(G_C)), S)$ eine graphkontrollierte Grammatik ist, i. e., in einer programmierten Grammatik werden weder Anfangs- noch Endregeln spezifiziert.

Eine *Matrix-Grammatik* ist ein Konstrukt $G_M = (N, T, (M, F), S)$; N ist das Variablenalphabet, T das Terminalalphabet, $N \cap T = \emptyset$; $S \in N$ ist das Startsymbol; M ist eine endliche Menge von Matrizen, $M = \{m_i \mid 1 \leq i \leq n\}$, wobei die Matrizen m_i endliche Folgen der Gestalt $m_i = (m_{i,1}, \ldots, m_{i,n_i})$, $n_i \geq 1$, $1 \leq i \leq n$, von (kontextfreien) Produktionen $m_{i,j}$, $1 \leq j \leq n_i$, $1 \leq i \leq n$, über $N \cup T$ sind; $F \subseteq \bigcup_{1 \leq i \leq n,\, 1 \leq j \leq n_i} \{m_{i,j}\}$.

Für $m_i = (m_{i,1}, \ldots, m_{i,n_i})$ und $v, w \in (N \cup T)^*$ definieren wir $v \Longrightarrow_{m_i} w$ genau dann, wenn es $w_0, w_1, \ldots, w_{n_i} \in (N \cup T)^*$ gibt, sodass $w_0 = v$, $w_{n_i} = w$, und für alle $j, 1 \leq j \leq n_i$,

- **entweder** w_j das Resultat der Anwendung von $m_{i,j}$ auf w_{j-1} ist

- **oder** $m_{i,j}$ nicht auf w_{j-1} anwendbar ist, $w_j = w_{j-1}$, und $m_{i,j} \in F$.

Die von G_M erzeugte Sprache ist

$$L(G_M) = \{w \in T^* \mid S \Longrightarrow_{m_{i_1}} w_1 \Longrightarrow_{m_{i_2}} \cdots \Longrightarrow_{m_{i_k}} w_k,\ w_k = w,\\ w_j \in (N \cup T)^*,\ m_{i_j} \in M \ \text{ für } 1 \leq j \leq k,\ k \geq 1\}.$$

Eine Variable $A \in N$ wird *mit Vorkommenstest verwendet,* wenn zumindest eine Produktion der Gestalt $A \to \alpha$, $\alpha \in (N \cup T)^*$, in F vorkommt.

Eine graphkontrollierte, programmierte bzw. Matrix-Grammatik ist vom Typ (n, m), wenn sie maximal n Variable enthält, von denen maximal m mit Vorkommenstest verwendet werden.

Wir definieren nun ein Modell von Registermaschinen, welches die in [6] verwendeten Modelle beinhaltet, i. e., alle dort beschriebenen Resultate gelten auch für das im Folgenden definierte Modell:

Eine *n-Registermaschine* ist ein Konstrukt $M = (n, R, i, f)$, wobei n eine natürliche Zahl ist, welche die Anzahl der Register angibt; R ist eine endliche Menge von *markierten Programminstruktionen* der Gestalt $k : (op(i), l, m)$, sodass $op(i)$ eine Operation auf Register i von M ist und k, l, m Markierungen aus einer Menge von Markierungen $Lab(M)$ sind, ($Lab(m)$ markiert die Programminstruktionen von M in einer Eins-zu-eins-Relation), $k \neq f$; l ist die Markierung für die Fortsetzung des Programms nach Anwendung der Instruktion $op(i)$ auf Register i; m ist die Markierung für die Fortsetzung des Programms, wenn $op(i)$ nicht auf Register i angewendet werden kann; der Endmarkierung f wird die Instruktion *end* zugeordnet, welche die Registermaschine M anhält; i ist die Anfangsmarkierung, bei der das Programm startet.

Programminstruktionen (op, l, m) :

- $(S(i), l, m)$: Ist der Inhalt von Register i größer als 0, so subtrahiere 1 von Register i und gehe zu l, andernfalls führe keine Registeroperation aus und gehe zu m.
- $(A(i), h, h)$: Addiere 1 zu Register i und gehe zu h.

Eine n-Registermaschine M berechnet eine partiell rekursive Funktion $f : \mathbb{N}_0 \to \mathbb{N}_0$ folgendermaßen:

M beginnt mit $m \in \mathbb{N}_0$ in Register 1; hält M in f mit r in Register 1, dann hat M den Wert $f(m) = r$ berechnet, andernfalls, wenn M nicht in f hält, bleibt $f(m)$ undefiniert.

3. Simulation von Registermaschinen durch graphkontrollierte Grammatiken

In [6] wurde gezeigt, wie die Aktionen einer deterministischen Turingmaschine von einer 2-Registermaschine simuliert werden können:

Ist T ein Terminalalphabet mit $card(T) = z - 1$ und $T = \{a_i \mid 1 \leq i \leq z - 1\}$, dann kann jedes Symbol a_i in T als Ziffer i zur Basis z interpretiert werden, i. e., jedes Wort in T^* kann als nichtnegative ganze Zahl dargestellt werde, beispielsweise mittels der Funktion $g_z : T^* \to \mathbb{N}_0$ definiert durch $g_z(\lambda) = 0$, $g_z(a_i) = i$ für $1 \leq i \leq z - 1$, und $g_z(wa) = g_z(w) * z + g_z(a)$ für $a \in T$ und $w \in T^*$.

Sei nun $L \subseteq T^*$ die von der deterministischen Turingmaschine M_T akzeptierte formale Sprache derart, dass M_T für jedes $w \in T^*$ genau dann hält, wenn $w \in L$. Dann kann man eine 2-Registermaschine M_L derart konstruieren, dass – für jedes $w \in T^*$ – M_L mit $2^{g_z(w)}$ im ersten Register beginnend genau dann hält (mit leeren Registern), wenn M_T auf w hält.

Basierend auf diesem Resultat aus [6] kann nun das Hauptresultat dieser Arbeit bewiesen werden:

Theorem *Jede rekursiv aufzählbare Sprache kann von einer graphkontrollierten Grammatik G_C vom Typ $(3,2)$ erzeugt werden.*

Beweis. Die wesentlichen Ableitungsschritte in G_C können folgendermaßen beschrieben werden:

- Anfangsregel: $(i : A \to A, \sigma(i), \emptyset)$
- Wir erzeugen schrittweise ein Terminalwort $w = a_1...a_k$ am Anfang der Satzform und gleichzeitig am Ende dessen Codierung $g_z(w)$; schließlich erhalten wir $wAB^{g_z(w)}$.

  Für $T = \{a\}$, also $z = 2$ erfolgt das Hinzufügen eines Terminalsymbols a durch die folgende Sequenz von Regeln:

  $(k_1 : A \to A, \{k_1+1\}, \emptyset)$
  $(k_1+1 : A \to aA, \{k_1+2\}, \emptyset)$
  $(k_1+2 : B \to \lambda, \{k_1+3\}, \{k_1+4\})$
  $(k_1+3 : A \to ACC, \{k_1+2\}, \emptyset)$
  $(k_1+4 : C \to \lambda, \{k_1+5\}, \{k_1+6\})$
  $(k_1+5 : A \to AB, \{k_1+4\}, \emptyset)$
  $(k_1+6 : A \to AB, \{k_1+7\}, \emptyset)$
  $(k_1+7 : A \to A, \{k_1, k_2\}, \emptyset)$

  In diesem Fall können wir einfach $\sigma\{i\} = \{k_1\}$ nehmen, für $z > 2$ müssen adäquate Module für das Hinzufügen eines neuen Terminalsymbols nichtdeterministisch gewählt werden.
- Ersetze alle Symbole B durch Symbole A, i. e., aus $wAB^{g_z(w)}$ erhält man $wA^{g_z(w)+1}$:

  $(k_2 : A \to A, \{k_2+1\}, \emptyset)$
  $(k_2+1 : B \to \lambda, \{k_2+2\}, \{k_2+3\})$
  $(k_2+2 : A \to AA, \{k_2+1\}, \emptyset)$
  $(k_2+3 : A \to A, \{k_3\}, \emptyset)$
- Erzeuge $2^{g_z(w)}$ aus $g_z(w)$ so, dass man schließlich wy mit $|y|_B = 2^{g_z(w)-\rho+1}$, $|y|_C = 0$, $|y|_A = \rho$ erhält. Eine erfolgreich terminierende Ableitung erhält man allerdings nur für $\rho = 1$.

  $(k_3 : A \to A, \{k_3+1\}, \emptyset)$
  $(k_3+1 : A \to AB, \{k_3+2\}, \emptyset)$
  $(k_3+2 : A \to A, \{k_3+3, k_3+8\}, \emptyset)$
  $(k_3+3 : A \to \lambda, \{k_3+4\}, \emptyset)$
  $(k_3+4 : B \to \lambda, \{k_3+5\}, \{k_3+6\})$
  $(k_3+5 : A \to ACC, \{k_3+4\}, \emptyset)$
  $(k_3+6 : C \to \lambda, \{k_3+7\}, \{k_3+2\})$
  $(k_3+7 : A \to AB, \{k_3+6\}, \emptyset)$
  $(k_3+8 : A \to A, \{k_4\}, \emptyset)$
- Simuliere die 2-Registermaschine M_L (die Anfangsmarkierung für diese Simulation von M_L in G_C ist k_4, die Endmarkierung von M_L ist mit der Markierung $f-1$ in G_C zu

identifizieren):

$k : (S(1), l, m)$ wird simuliert durch $(k : B \to \lambda, \{l\}, \{m\})$;
$k : (S(2), l, m)$ wird simuliert durch $(k : C \to \lambda, \{l\}, \{m\})$;
$k : (A(1), l, l)$ wird simuliert durch $(k : A \to AB, \{l\}, \emptyset)$;
$k : (A(2), l, l)$ wird simuliert durch $(k : A \to AC, \{l\}, \emptyset)$.

Die Anzahl der Variablen B bzw. C repräsentiert den Inhalt von Register 1 bzw. Register 2 von M_L.

- Falls M_L hält, endet G_C mit

  $(f - 1 : A \to \lambda, \{f\}, \emptyset)$ und $(f : B \to \lambda, \emptyset, \emptyset)$.

  Die schließlich so erhaltene Satzform ist von der Gestalt $wA^{\rho-1}$, i. e., nur für $\rho = 1$ erhalten wir das Terminalwort $w \in L$ (was bedeutet, dass M_L auf $2^{g_z(w)}$ hält). □

Aus dem oben gezeigten Resultat können die folgenden Ergebnisse abgeleitet werden (die Beweise sind in [4] zu finden):

Korollar *Jede rekursiv aufzählbare Sprache kann von einer*

- *programmierten Grammatik vom Typ* $(4, 2)$
- *Matrix-Grammatik vom Typ* $(4, 3)$
- *Matrix-Grammatik vom Typ* $(n, 2)$ *für ein* $n \geq 2$

erzeugt werden.

Ob auch programmierte Grammatiken vom Typ $(3, 2)$ genügen, um jede rekursiv aufzählbare Sprache zu erzeugen, ist noch eingehender zu untersuchen.

Literatur

[1] C. Calude, Gh. Păun. *Computing with Cells and Atoms.* Taylor and Francis, London, 2000.

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

[3] H. Fernau. Nonterminal Complexity of Programmed Grammars. *Machines, Computations, and Universality* (M. Margenstern, Y. Rogozhin, eds.), MCU 2001 Proceedings, LNCS 2055, Springer, Berlin, 2001, pp. 202–213.

[4] R. Freund, Gh. Păun. On the Number of Non-terminal Symbols in Graph-Controlled, Programmed and Matrix Grammars. *Machines, Computations, and Universality* (M. Margenstern, Y. Rogozhin, eds.), MCU 2001 Proceedings, LNCS 2055, Springer, Berlin, 2001, pp. 214–225.

[5] R. Freund, Gh. Păun. Computing with Membranes: Three More Collapsing Hierarchies. EMCC Meeting, Milano, Italy, November, 2000.

[6] M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, New Jersey, USA.

[7] Gh. Păun. Six Nonterminals are Enough for Generating Each R.E. Language by a Matrix Grammar. *Intern. J. Computer Math.* **15** (1984), pp. 23–37.

[8] Gh. Păun. Computing with Membranes. *Journal of Computer and System Sciences* **1**, 1 (2000), pp. 108–143.

TREE TRANSDUCERS WITH COSTS

Zoltán Fülöp
Department of Computer Science, University of Szeged
Árpád tér 2, H-6720 Szeged, Hungary
e-mail: fulop@inf.u-szeged.hu

and

Heiko Vogler
Faculty of Computer Science, Dresden University of Technology
Mommsenstraße 13, D-01062 Dresden, Germany
e-mail: vogler@inf.tu-dresden.de

In this talk we discuss the generalization of bottom-up and top-down tree transducers by adding costs from a semiring to every transition. We show some of the problems while trying to transfer the Theory of Tree Transducers to the generalized model.

STATE COMPLEXITY OF BASIC OPERATIONS ON NONDETERMINISTIC FINITE AUTOMATA

Markus Holzer
Institut für Informatik, Technische Universität München
Arcisstr. 21, D-80290 München, Germany
e-mail: holzer@informatik.tu-muenchen.de

and

Martin Kutrib
Institut für Informatik, Universität Giessen
Arndtstr. 2, D-35392 Giessen, Germany
e-mail: kutrib@informatik.uni-giessen.de

ABSTRACT

The state complexities of basic operations on nondeterministic finite automata (NFA) are investigated. In particular, we consider Boolean operations, catenation operations – concatenation, iteration, λ-free iteration – and the reversal on NFAs that accept finite and infinite languages over arbitrary alphabets. Most of the shown bounds are tight in the exact number of states, i.e. the number is sufficient and necessary in the worst case. For the intersection of finite languages and the complementation tight bounds in the order of magnitude are proved.
It turns out that the state complexities of operations on NFAs and deterministic finite automata (DFA) are quite different. For example, the reversal and concatenation have exponential state complexity on DFAs but linear complexity on NFAs. Conversely, the complementation can be done with linear complexity on DFAs but needs exponentially many states on NFAs.

Keywords: state complexity, finite automata, nondeterminism, finite and infinite regular languages.

1. Introduction

Motivated by several applications and implementations of finite automata in software engineering, programming languages and other practical areas in computer science, the state complexity of deterministic finite automata has been studied in recent years. For example, the state complexity of the intersection of DFAs has been studied in [15]. A tight bound of 2^n states for the reversal has been shown in [8], whereas catenations and other operations are the main topic of [16]. For the important case of finite languages results have been obtained in [1]. A state-of-the-art survey can be found in [14]. Related to the problem of finding upper bounds for the state complexity is the problem of efficiently simulating nondeterministic automata by deterministic ones. For example, transforming a certain type of NFA to a DFA gives an upper bound for the corresponding NFA state complexity of complementation. Results concerning the simulation problems have been shown in [3, 9, 10, 12].

As pointed out in [14] there are several good reasons why the size of DFAs is a natural and objective measure for regular languages. On the other hand, the influence of the degree of nondeterminism on the power and limitations of certain devices is an important question in descriptional complexity theory. Finite automata with limited nondeterminism have been

considered in [7] where an infinite nondeterministic hierarchy of regular languages has been proved. In [4] it is dealt with the quantification of inherent nondeterminism in regular languages, and in [5] with the relation between ambiguity and the amount of nondeterminism.

We expect that examining the state complexity of basic operations on NFAs will enhance the understanding of the relations between nondeterminism, ambiguity and the power of finite automata.

2. Boolean Operations

We start our investigations with Boolean operations on NFAs that accept languages over arbitrary alphabets. In the case when the finite automaton is deterministic it is well-known that in the worst case the Boolean operations union, intersection and complementation have a state complexity of $m \cdot n$, $m \cdot n$ and m, respectively. (m and n denote the number of states of the automata on which the operations are performed.) However, the state complexity of NFA operations is essentially different. At first we consider the union.

Theorem 1 *For any integers $m, n \geq 1$ let $\mathcal{A}$ be an m-state and $\mathcal{B}$ be an n-state* NFA. *Then $m + n + 1$ states are sufficient and necessary in the worst case for an* NFA $\mathcal{C}$ *to accept the language $L(\mathcal{A}) \cup L(\mathcal{B})$.*

When we are concerned with finite languages the state complexity of the union can be reduced by three states. In the deterministic case some more states can be saved. For these upper bounds see [2].

Corollary 2 *For any integers $m, n \geq 1$ let $\mathcal{A}$ be an m-state* NFA *and $\mathcal{B}$ be an n-state* NFA. *If $L(\mathcal{A})$ and $L(\mathcal{B})$ are finite, then $m + n - 2$ states are sufficient and necessary in the worst case for an* NFA $\mathcal{C}$ *to accept the language $L(\mathcal{A}) \cup L(\mathcal{B})$.*

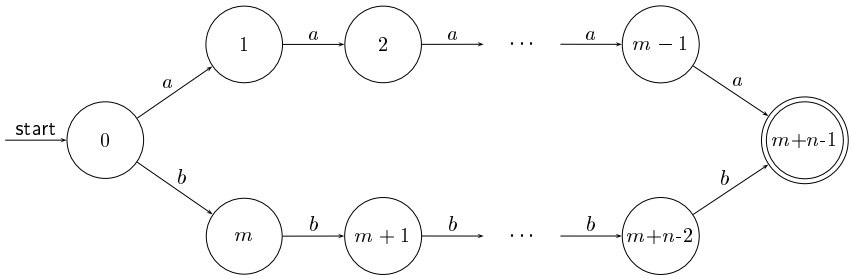


Figure 1: Minimal NFA accepting $\{a^m\} \cup \{b^n\}$.

The complementation on nondeterministic devices is often a difficult problem. In case of regular languages it is an expensive task at any rate. It is well known [11] that 2^n is the tight upper bound on the number of states necessary for a deterministic finite automaton to accept an (infinite) n-state NFA language. Since the complementation operation on deterministic finite automata neither increases nor decreases the number of states (simply exchange final and non-final states) we obtain an upper bound for the state complexity of the complementation on NFAs.

Corollary 3 *For any integer $n \geq 1$ the complement of an n-state* NFA *language is accepted by a 2^n-state* NFA.

Unfortunately, this expensive upper bound is tight in the order of magnitude. Basically, the idea is to construct an efficiently acceptable language such that nondeterminism cannot do anything for a cheap and efficient acceptance of its complement.

Theorem 4 *For any integer $n > 2$ there exists an n-state* NFA $\mathcal{A}$ *such that any* NFA *that accepts the complement of $L(\mathcal{A})$ needs at least 2^{n-2} states.*

The situation for finite languages over an ℓ-letter alphabet, $\ell \geq 2$, is quite different, since the upper bound of the transformation to a deterministic finite automaton is different. In [12] it has been shown that $O(\ell^{\sqrt[n]{\log_2 \ell+1}})$ states are an upper bound for deterministic finite automata accepting a finite n-state NFA language.

Corollary 5 *For any integers $\ell, n > 1$ the complement of a finite n-state* NFA *language over an ℓ-letter alphabet is accepted by an $O(\ell^{\sqrt[n]{\log_2 \ell+1}})$-state* NFA.

Note, that for $\ell = 2$ the upper bound is $O(2^{\frac{n}{2}})$.

Theorem 6 *For any integers $\ell > 1$ and $n > 2$ there exists a finite n-state* NFA *language L over an ℓ-letter alphabet such that any* NFA *that accepts the complement of L needs at least $\Omega(\ell^{\sqrt[n]{2 \cdot \log_2 \ell}})$ states.*

Next we are going to prove a tight bound for the remaining Boolean operation, the intersection. The upper bound is obtained by the somehow old-fashioned cross-product construction.

Theorem 7 *For any integers $n, m \geq 1$ let $\mathcal{A}$ be an m-state and $\mathcal{B}$ be an n-state* NFA. *Then $m \cdot n$ states are sufficient and necessary in the worst case for an* NFA *to accept the language $L(\mathcal{A}) \cap L(\mathcal{B})$.*

3. Catenation Operations

Now we turn to the catenation operations. In particular, tight bounds for concatenation, iteration and λ-free iteration will be shown. Roughly speaking, in terms of state complexity these are cheap operations for NFAs. Again, this is essentially different when deterministic finite automata come to play. For example, in [16] a bound of $(2m-1) \cdot 2^{n-1}$ states has been shown for the DFA-concatenation, and in [13] a bound of $2^{n-1} + 2^{n-2}$ states for the iteration.

Theorem 8 *For any integers $m, n \geq 1$ let $\mathcal{A}$ be an m-state* NFA *and $\mathcal{B}$ be an n-state* NFA. *Then $m + n$ states are sufficient and necessary in the worst case for an* NFA $\mathcal{C}$ *to accept the language $L(\mathcal{A})L(\mathcal{B})$.*

In case of finite languages the concatenation is one state cheaper.

Lemma 9 *For any integers $m, n \geq 1$ let $\mathcal{A}$ be an m-state* NFA *and $\mathcal{B}$ be an n-state* NFA. *If $L(\mathcal{A})$ and $L(\mathcal{B})$ are finite, then $m + n - 1$ states are sufficient and necessary in the worst case for an* NFA $\mathcal{C}$ *to accept the language $L(\mathcal{A})L(\mathcal{B})$.*

The constructions yielding the upper bounds for the iteration and λ-free iteration are similarly. The trivial difference between both operations concerns the empty word only. Moreover, the difference does not appear for languages containing the empty word. Nevertheless, in the worst case the difference costs one state.

Theorem 10 *For any integer $n > 2$ let $\mathcal{A}$ be an n-state* NFA. *Then $n + 1$ resp. n states are sufficient and necessary in the worst case for an* NFA *to accept the language $L(\mathcal{A})^*$ resp. $L(\mathcal{A})^+$.*

The state complexity for the iterations in the finite language case is n resp. $n - 1$.

Lemma 11 *For any integer $n > 1$ let $\mathcal{A}$ be an n-state* NFA. *If $L(\mathcal{A})$ is finite, then $n - 1$ resp. n states are sufficient and necessary in the worst case for an* NFA *to accept the language $L(\mathcal{A})^*$ resp. $L(\mathcal{A})^+$.*

4. Reversal

The last operation under consideration is the reversal. For deterministic automata one may expect that the state complexity is linear. But it is not. In [16] for infinite languages a tight bound of 2^n has been shown. A proof of a tight bound for finite languages can be found in [1]. It is of order $O(2^{\frac{n}{2}})$ for a two-letter alphabet. From the following cheap bounds for NFAs it follows once more that nondeterminism is a powerful concept.

Theorem 12 *For any integer $n > 3$ let $\mathcal{A}$ be an n-state* NFA*. Then $n+1$ states are sufficient and necessary in the worst case for an* NFA *$\mathcal{C}$ to accept the language $L(\mathcal{A})^R$.*

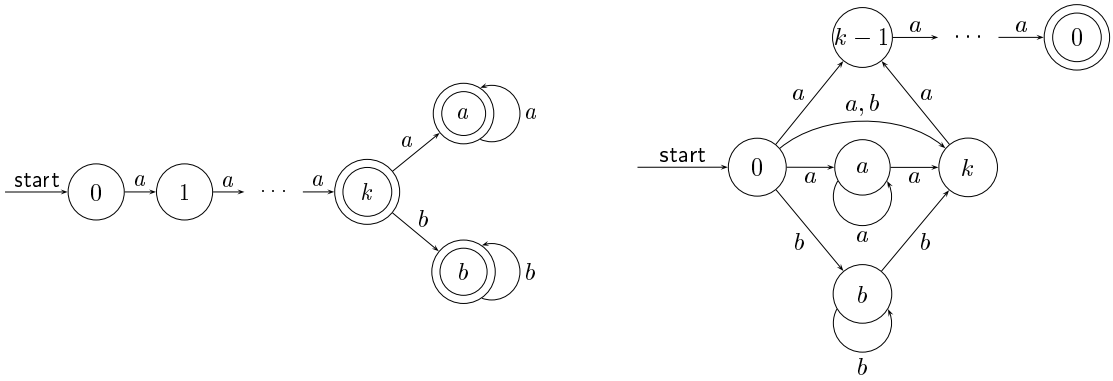


Figure 2: A $(k+3)$-state and a $(k+4)$-state NFA accepting languages L_k and L_k^R.

The fact that NFAs for finite languages do not have any cycle leads once more to the possibility of saving one state compared with the infinite case.

Lemma 13 *For any integer $n \geq 1$ let $\mathcal{A}$ be an n-state* NFA*. If $L(\mathcal{A})$ is finite, then n states are sufficient and necessary in the worst case for an* NFA *to accept the language $L(\mathcal{A})^R$.*

The bound for the reversal of finite NFA languages is in some sense strong. It is sufficient and reached for all finite languages. It holds also for the empty language.

Finally, Table 1 summarizes the state complexity bounds for NFAs and DFAs.

	NFA		DFA	
	finite	infinite	finite	infinite
$\cup$	$m+n-2$	$m+n+1$	$O(mn)$	mn
$\sim$	$O(\ell^{\frac{n}{\log_2 \ell+1}})$	$O(2^{n-2})$	n	n
$\cap$	$O(mn)$	mn	$O(mn)$	mn
R	n	$n+1$	$O(2^{\frac{n}{2}})$	2^n
$\cdot$	$m+n-1$	$m+n$	$O(mn^{t-1}+n^t)$	$(2m-1)2^{n-1}$
$*$	$n-1$	$n+1$	$2^{n-3}+2^{n-4}$	$2^{n-1}+2^{n-2}$
$+$	n	n		

Table 1: Comparison of the NFA and DFA state complexities (ℓ is the number of states, t is the number of final states of the 'left' automaton).

References

[1] Câmpeanu, C., Čulik, K., Salomaa, K., and Yu, S. *State complexity of basic operations on finite languages*. International Workshop on Implementing Automata, 1999, to appear.

[2] Câmpeanu, C., Čulik, K., Sântean, N., and Yu, S. *Finite languages and cover-automata*. Theoretical Computer Science, to appear.

[3] Chrobak, M. *Finite automata and unary languages*. Theoretical Computer Science 47 (1986), 149–158.

[4] Goldstine, J., Kintala, C., and Wotschke, D. *On measuring nondeterminism in regular languages*. Information and Computation 86 (1990), 179–194.

[5] Goldstine, J., Leung, H., and Wotschke, D. *On the relation between ambiguity and nondeterminism in finite automata*. Information and Computation 100 (1992), 261–270.

[6] Hopcroft, J. E. and Ullman, J. D. *Introduction to Automata Theory, Language, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

[7] Kintala, C. M. and Wotschke, D. *Amounts of nondeterminism in finite automata*. Acta Informatica 13 (1980), 199–204.

[8] Leiss, E. *Succinct representation of regular languages by Boolean automata*. Theoretical Computer Science 13 (1981), 323–330.

[9] Mereghetti, C. and Pighizzini, G. *Optimal simulations between unary automata*. Symposium on Theoretical Aspects of Computer Science, LNCS 1373, 1998, pp. 139–149.

[10] Mereghetti, C. and Pighizzini, G. *Unary automata simulations and cyclic languages*. International Workshop on Descriptional Complexity of Automata, Grammars and Related Structures, 1999, pp. 145–153.

[11] Meyer, A. R. and Fischer, M. J. *Economy of description by automata, grammars, and formal systems*. IEEE Symposium on Switching and Automata Theory, 1971, pp. 188–191.

[12] Salomaa, K. and Yu, S. *NFA to DFA transformation for finite languages over arbitrary alphabets*. Journal of Automata, Languages and Combinatorics 2 (1997), 177–186.

[13] Yu, S. *Regular languages*. In Rozenberg, G. and Salomaa, A. (eds.), *Handbook of Formal Languages I*. Springer, 1997, chapter 2, pp. 41–110.

[14] Yu, S. *State complexity of regular languages*. International Workshop on Descriptional Complexity of Automata, Grammars and Related Structures, 1999, pp. 77–88.

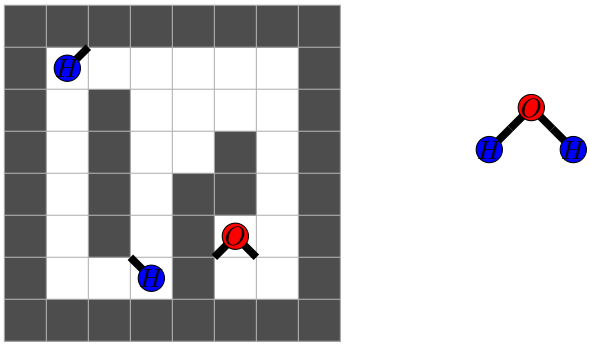
[15] Yu, S. and Zhuang, Q. *On the state complexity of intersection of regular languages*. SIGACT News 22.3 (1991), 52–54.

[16] Yu, S., Zhuang, Q., and Salomaa, K. *The state complexities of some basic operations on regular languages*. Theoretical Computer Science 125 (1994), 315–328.

ASSEMBLING MOLECULES IN ATOMIX IS HARD

Markus Holzer and Stefan Schwoon
Institut für Informatik, Technische Universität München
Arcisstraße 21, D-80290 München, Germany
e-mail: {holzer,schwoon}@in.tum.de

Atomix is a solitaire game invented by Günter Krämer in 1990 and first published by Thalion Software. The game takes place on a rectangular finite two-dimensional grid, the board. Every cell of the board is either a wall, contains an atom, or is free. Walls cannot be changed, and atoms can be of different types. A move consists of pushing an atom along the x-axis or the y-axis. When an atom is pushed, it continues moving until it reaches a wall or another atom. The game is won when all atoms are arranged in a given "molecule" goal pattern. An instance of an Atomix game assembling the water molecule is depicted in Figure 1 – walls are represented by black squares, free space by white squares, and atoms by labelled circles with connections. Implementations of certain Atomix variations are available on the Internet; e. g.,



(a) Atomix board. (b) Molecule.

Figure 1: An Atomix problem assembling the water molecule.

for the X Window System a version under the terms of the General Public License (GPL) can be downloaded from www.informatik.uni-oldenburg.de/~pearl/gnome/atomix.html. It is worth mentioning that this program also contains a level editor. A JavaScript version can be played online at www.sect.mce.hw.ac.uk/~peteri/atomix/.

Formally the Atomix problem is defined as follows: Given an Atomix board and a molecule, is there a sequence of moves to assemble the atoms on the board to form the given molecule? Obviously, this problem can be formalized as a state space search problem, which recently was undertaken by Hüffner *et al.* [9]. There different heuristic search methods were presented. Atomix falls into the category of sliding block puzzles as, e. g., PushPush [3], Sokoban [2, 4], or 15-Puzzle [12], where time and space complexity was, and still is, subject of intense research. Though seemingly trivial, most variations are at least NP-hard, and contained in PSPACE; some are even PSPACE-complete – we refer the reader to, e. g., Balcázar *et al.* [1] for further details

on computational complexity. Hüffner *et al.* [9] actually have shown that Atomix is NP-hard and contained in PSPACE, while the exact complexity was stated as an open problem. In this paper we solve this open problem and improve their result showing the following theorem:

Theorem 1 *Atomix on an $n \times n$ board is PSPACE-complete.*

To this end we show that Atomix game puzzles can simulate deterministic or nondeterministic finite automata. In particular, we construct an Atomix instance that is solvable if and only if the non-emptiness intersection problem for finite automata, a problem known to be PSPACE-complete [6, 10], has a solution. Observe that most importantly the above given theorem shows that there are Atomix instances which have superpolynomially long optimal solutions.

Although most constructions used in the PSPACE-completeness proof are much like those in the popular game, it is a tedious exercise to construct a particular Atomix instance having an exponentially long optimal solution. We give an easy construction of Atomix game levels whose optimal solutions meet the worst case. To this end we realize a pseudo n-bit counter in Atomix. The main part of the construction consists of the simulation of a single bit, i. e., a device that stores a bit, changes it accordingly, and produces a carry bit if triggered by an increment event. In order to obtain an easy Atomix game level, we implement a pseudo-counting process. This means that the stored 0 bit can, but need not, be changed to 1 and producing no-carry by an increment event, while a 1 bit must be changed to 0 and produces a carry bit. This slight difference to ordinary counting will allow us to construct simple Atomix game levels.

Finally we summarize some complexity results on block sliding puzzles, into which Atomix falls. As already mentioned most of these problems are NP-hard and contained in PSPACE; some of them are even PSPACE-complete. Table 1 is taken from Demaine *et al.* [3], extended

Game	1. Robot	2. Pull	3. Blocks	4. Fixed	5. #	6. Path	7. Slide	8. Dim.	9. Complexity
PushPush3D	+	−	unit	−	1	+	+	3D	NP-hard [11]
PushPush	+	−	unit	−	1	+	+	2D	NP-hard [3]
Push-*	+	−	unit	−	k	−	−	2D	NP-hard [7]
Sokoban+	+	−	1×2	+	2	−	−	2D	PSPACE-compl. [4]
Sokoban	+	−	unit	+	1	−	−	2D	PSPACE-compl. [2]
15-Puzzle	−		unit	−	1	−	−	2D	NP-hard [12]
RushHour	−		1×{2,3}	−	1	+	−	2D	PSPACE-compl. [5]
Atomix	−		unit	+	1	−	+	2D	PSPACE-compl.
Bricks	−		variable	+	1	+	−	2D	PSPACE-compl. [8]

Table 1: Computational complexity of some block sliding puzzles summarized.

by the category of games where blocks are pushed by an external agent not presented on the board. The columns mean:

1. Are the moves done by a robot on the board, or by an outside agent?
2. Can the robot pull as well as push?
3. Are all block unit squares, or may the have different shapes?
4. Are there fixed blocks, or are all blocks movable?
5. How many blocks can by pushed at a time?

6. Does it suffice to move the robot/a special block from s to t, instead of pushing all blocks into storage location?
7. Will the blocks "keep sliding" when pushed till they hit an obstacle?
8. The Dimension of the puzzle: Is it 2D or 3D?

References

[1] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1988.

[2] J. Culberson. Sokoban is PSPACE-complete. In *International Conference on Fun with Algorithms*, Proceedings in Informatics 4, pages 65–76, Elba, Italy, June 1999. Carleton Scientific, Waterloo, Canada.

[3] E. D. Demaine, M. L. Demaine, and J. O'Rourke. PushPush and Push-1 are NP-hard in 2D. In *Proceedings of the 12th Annual Canadian Conference on Computational Geometry*, pages 17–20, Fredericton, New Brunswick, Canada, August 2000.

[4] D. Dor and U. Zwick. Sokoban and other motion planning problems. *Computational Geometry: Theory and Applications*, 13(4):215–228, 1999.

[5] G. W. Flake and E. B. Baum. RushHour is PSPACE-complete, or "Why you should generously tip parking lot attendants". Submitted for publication, 2001.

[6] Z. Galil. Hierarchies of complete problems. *Acta Informatica*, 6:77–88, 1976.

[7] M. Hoffmann. Motion planning amidst movable square blocks: Push-* is NP-hard. In *Proceedings of the 12th Annual Canadian Conference on Computational Geometry*, pages 205–209, Fredericton, New Brunswick, Canada, August 2000.

[8] M. Holzer and S. Schwoon. Personal communication. April 2001.

[9] F. Hüffner, S. Edelkamp, H. Fernau, and R. Niedermeier. Finding optimal solutions to Atomix. Submitted for publication, May 2001.

[10] D. Kozen. Lower bounds for natural proof systems. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 254–266, 1977.

[11] J. O'Rourke and the Smith Problem Solving Group. PushPush is NP-hard in 3D. Technical Report 064, Smith College, Northampton, MA, November 1999.

[12] D. Ratner and M. K. Warmuth. The $(n^2 - 1)$-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, 1990.

COMPOSITION OF TREE TRANSDUCERS VERSUS CATEGORICAL DEFORESTATION

Claus Jürgensen [1]
Faculty of Computer Science, Dresden University of Technology
D-01062 Dresden, Germany
e-mail: Claus.Juergensen@inf.tu-dresden.de

1. Overview

We compare short cut deforestation formalized in category theory (which we call fusion) with the syntactic composition of tree transducers. The former strongly depends on types and uses the parametricity property [16] whereas the latter makes no use of types at all and allows more general compositions. We introduce the notion of a categorical transducer which is a generalization of a catamorphism and show a respective fusion result which is a generalization of the 'acid rain theorem'. We prove the following main theorems:

- The class of all categorical transducers builds a category where composition is fusion.
- The semantics of categorical transducers is a functor.
- The subclass of top-down categorical transducers is a subcategory.
- Syntactic composition of top-down tree transducers is equivalent to the fusion of top-down categorical transducers.

2. Deforestation

Deforestation is a program transformation on functional programs to eliminate intermediate data structures. Various techniques to achieve this are known: classical deforestation [17], cheap deforestation [9], short cut deforestation [10] and short cut deforestation using the 'acid rain theorem' [15], warm fusion [13], hylo-fusion [11], symbolic composition of attributed grammars [8, 3], and syntactic composition of tree transducers [14, 2].

The 'acid rain theorem' is expressed in terms of category theory, where recursive functions are represented by catamorphisms. A catamorphism $(\!|\varphi|\!)_{\mathsf{F}}$ is the unique morphism which satisfies the equation $(\!|\varphi|\!)_{\mathsf{F}} \cdot in_{\mathsf{F}} = \varphi \cdot \mathsf{F}(\!|\varphi|\!)_{\mathsf{F}}$, where F is an endofunctor, φ is an F-algebra, and in_{F} is an initial F-algebra.

We generalize the well known 'acid rain theorem' from [15] where we use a *concrete functor* H (cf. [1]):

[1]Supported by the postgraduate program 'Specification of discrete processes and systems of processes by operational models and logics' (GRK 334/2) of the German Research Community (DFG)

Theorem 1 (generalized 'acid rain theorem', [12]) *Let $\mathcal{C}$ be a category and $\mathsf{F}, \mathsf{G}, \mathsf{U} : \mathcal{C} \leftarrow \mathcal{C}$ endofunctors where F and G have initial algebras and U is faithful. For every $\varphi \in \mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}}\mathsf{G})$ holds:*

$$\frac{\mathsf{H} : (\mathcal{A}lg_{\mathcal{C}}\mathsf{F}, |\bullet|_{\mathsf{F}}) \leftarrow (\mathcal{A}lg_{\mathcal{C}}\mathsf{G}, \mathsf{U} \cdot |\bullet|_{\mathsf{G}})}{\mathsf{U}(\!|\varphi|\!)_{\mathsf{G}} \cdot (\!|\mathsf{H}in_G|\!)_{\mathsf{F}} = (\!|\mathsf{H}\varphi|\!)_{\mathsf{F}}}$$

□

Thus we expressed the composition of two catamorphisms as a single catamorphism. We call this and similar results *fusion*. This theorem is the basis for the following fusion of categorical transducers.

3. Tree Transducers

A tree transducer is a specific term rewrite system. A specific tree transducer called *top-down tree transducer* can be viewed as a functional program with a strongly restricted syntax. We denote the function which a tree transducer T computes by τT. An interesting question is whether a given function can be computed by a tree transducer. As a special case we want to know whether the composition of two functions, where each can be computed by a tree transducer, can be computed by a single tree transducer itself. This question has been answered by [14] for top-down tree transducers:

Theorem 2 *Let T_2 and T_1 be top-down tree transducers. There exist a top-down tree-transducer T, such that*

$$\tau T_2 \cdot \tau T_1 = \tau T.$$

□

The proof (cf. [6]) uses an explicit construction of the top-down tree transducer T from T_2 and T_1. We call this the *syntactic composition* of T_2 and T_1 and denote it by $T_2 \cdot T_1$. Thus the proposition of the last theorem may be written in the form $\tau T_2 \cdot \tau T_1 = \tau(T_2 \cdot T_1)$. Besides top-down tree transducers, various classes of tree transducers, e. g. ((weakly) single-use) macro tree transducers or attributed tree transducers, have been investigated [5, 8]. We are interested in the similarities and differences of fusion and syntactic composition.

4. Categorical Transducers

We tried to find something like the 'greatest common divisor' of fusion and syntactic composition of tree transducers:

Definition (categorical transducer) *Let $\mathcal{C}$ be a category and $\mathsf{F}, \mathsf{G} : \mathcal{C} \leftarrow \mathcal{C}$ be endofunctors which have an initial algebra. A* categorical transducer *to G from F over $\mathcal{C}$*

$$(\mathsf{H}, \mathsf{U}, \pi) : \mathsf{G} \leftarrow \mathsf{F}$$

consists of

- *a faithful endofunctor $\mathsf{U} : \mathcal{C} \leftarrow \mathcal{C}$,*
- *a natural transformation $\pi : \mathsf{Id} \stackrel{\cdot}{\leftarrow} \mathsf{U}$, and*
- *a concrete functor $\mathsf{H} : (\mathcal{A}lg_{\mathcal{C}}\mathsf{F}, |\bullet|_{\mathsf{F}}) \leftarrow (\mathcal{A}lg_{\mathcal{C}}\mathsf{G}, \mathsf{U} \cdot |\bullet|_{\mathsf{G}})$.*

The semantics S of the categorical transducer $(\mathsf{H}, \mathsf{U}, \pi) : \mathsf{G} \leftarrow \mathsf{F}$ is defined by

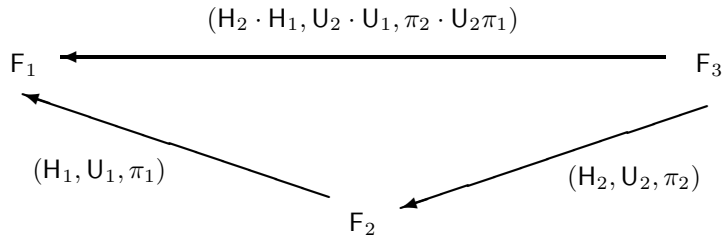
$$\mathsf{S}(\mathsf{H}, \mathsf{U}, \pi) = \pi \cdot (\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}} : \mu\mathsf{G} \leftarrow \mu\mathsf{F}.$$

The following theorem is a version of our generalized 'acid rain theorem' for categorical transducers:

Theorem 3 (composition of categorical transducers) *The class of all categorical transducers over $\mathcal{C}$ is a category with identity*

$$id_{\mathsf{F}} = (\mathsf{Id}_{(\mathcal{A}lg_{\mathcal{C}}\mathsf{F},|\bullet|_{\mathsf{F}})^{(\mathcal{A}lg_{\mathcal{C}}\mathsf{F},|\bullet|_{\mathsf{F}})}}, \mathsf{Id}_{\mathcal{C}}, id)$$

and composition



□

We denote this category by $cat\mathcal{T}_{\mathcal{C}}$.

Now we have to reinvent both theories (from Section 2 and Section 3) on this basis and then we can compare them:

First, we give an 'acid rain theorem' for categorical transducers:

Theorem 4 *The semantics* S *is a* functor $\mathsf{S} : \mathcal{C} \leftarrow cat\mathcal{T}_{\mathcal{C}}$. □

Second, we define a *top-down categorical transducer* and a respective composition result:

Theorem 5 *The subclass of all top-down categorical transducers is a category.* □

Third, we compare top-down categorical transducers with top-down tree transducers:

5. Relation

We define a relation between a top-down tree transducer T and a top-down categorical transducers C, which we denote $T \approx C$ and investigate some of its properties:

Theorem 6 *For every top-down tree transducer T exist a related top-down categorical transducer C, i. e. $T \approx C$.* □

Theorem 7 *For every top-down categorical transducer C exist a related top-down tree transducer T, i. e. $T \approx C$.* □

The main necessary condition that the relation has to satisfy is that it implies semantic equivalence:

Theorem 8 *Let T be a top-down tree transducer and C a top-down categorical transducer.*

$$T \approx C \implies \tau T = \mathsf{S}C.$$ □

Finally we are able to compare fusion and syntactic composition:

Theorem 9

$$T_2 \approx C_2 \wedge T_1 \approx C_1 \implies T_2 \cdot T_1 \approx C_2 \cdot C_1$$ □

6. Open Questions

- Is our concept of a categorical transducer general enough? Until now we could not express a macro tree transducer as a categorical transducer (but nearly).
- Why can we compose two functions with context parameters using macro tree transducers but not with the 'acid rain theorem'?
- When is the composition better than two consecutive functions? What is 'better'?
- What is the relation between the 'acid rain theorem' and cut-elimination? [4]
- When is a morphism a catamorphism? In $\mathcal{S}et$: [7].

References

[1] J. Adámek, H. Herrlich, and G. E. Strecker. *Abstract and Concrete Categories*. Pure and Applied Mathematics. John Wiley & Sons, 1990.

[2] B. S. Baker. Composition of top-down and bottom-up tree transductions. *Inform. and Control*, 41:186–213, 1979.

[3] L. Correnson, E. Duris, D. Parigot, and G. Roussel. Symbolic composition. Technical Report 3348, INRIA, January 1997.

[4] R. Cockett. Deforestation, program transformation, and cut-elimination. In *Proceedings of the 4th Workshop on Coalgebraic Methods in Computer Science (CMCS 2001)*, pages 27–46, Genova, Italy, April 2001. Elsevier Science Publishers.

[5] J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. System Sci.*, 31:71–146, 1985.

[6] Z. Fülöp and H. Vogler. *Syntax-directed semantics – Formal models based on tree transducers*. Monographs in Theoretical Computer Science, An EATCS Series. Springer-Verlag, 1998.

[7] J. Gibbons, G. Hutton, and T. Altenkirch. When is a function a fold or an unfold. In *Proceedings of the 4th Workshop on Coalgebraic Methods in Computer Science (CMCS 2001)*, pages 27–46, Genova, Italy, April 2001. Elsevier Science Publishers.

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

[9] A. Gill. *Cheap Deforestation for Non-strict Functional Languages*. PhD thesis, Department of Computing Science, Glasgow University, January 1996.

[10] A. Gill, J. Launchburry, and S. L. Peyton-Jones. A short cut to deforestation. In *Proceedings of Functional Programming Languages an Computer Architecture (FPCA '93)*, pages 223–232, Copenhagen, Denmark, June 1993. ACM Press.

[11] Z. Hu, H. Iwasaki, and M. Takeichi. Deriving structural hylomorphisms from recursive definitions. In *Proceedings of the 1st International Conference on Functional Programming*, pages 73–82, Philadelphia, PA, May 1996. ACM Press.

[12] C. Jürgensen. A formalization of hylomorphism based deforestation with an application to an extended typed λ-calculus. Technical Report TUD-FI00-13, Technische Universität Dresden, Fakultät Informatik, D-01062 Dresden, Germany, November 2000.

[13] J. Launchburry and T. Sheard. Warm fusion: Deriving build-catas from recursive definitions. In *Proceedings of Functional Programming Languages an Computer Architecture (FPCA '95)*, pages 314–323, La Jolla, San Diego, CA, USA, June 1995. ACM Press.

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

[15] A. Takano and E. Meijer. Shortcut deforestation in calculational form. In *Proceedings of the Conference on Functional Programing Languages and Computer Architecture*, pages 306–313, La Jolla, CA, June 1995. ACM Press.

[16] P. Wadler. Theorems for free! In *The 4th International Conference on Functional Programming Languages and Computer Architecture (FPCA '89)*, pages 347–359, London, September 1989. Imperial College, ACM Press.

[17] P. Wadler. Deforestation: Transforming programs to eliminate trees. *Theoretical Computer Science*, 73(2):231–248, 1990.

FALTUNGSCODES AUS SICHT DER AUTOMATENTHEORIE

Andreas Klein
Fachbereich Mathematik / Informatik, Universität Gesamthochschule Kassel
Postfach 10 13 80, D-34109 Kassel, Germany
e-mail: klein@mathematik.uni-kassel.de

Bei Übertragung von Information über einen verrauschten Kanal stellt sich das Problem der Fehlererkennung und -korrektur. Dazu werden zusätzlich zu den Informationbits auch Kontrollbits, die sich aus den Informationbits berechnen lassen, übertragen. Die Codierungstheorie beschäftigt sich mit der Konstruktion guter Codes. Eine spezielle Klasse von Codes sind die sogenannten Faltungscodes (eng.: convolutional codes).

Ein (binärer) Faltungscode mit Rate k/n ist ein Mealy Automat mit Eingabealphabet $\{0,1\}^k$ und Ausgabealphabet $\{0,1\}^n$. Ein Hauptproblem der Codierungstheorie ist:

- Gegeben ist die Anzahl der Zustände m. Wie groß kann der Minimalabstand d_{free} der Ausgabefolgen werden?

Aus praktischen und theoretischen Gründen beschränken sich die meisten Untersuchungen auf den Spezialfall eines Schieberegisters mit linearer Ausgabefunktion. Die Gründe für diese Einschränkung sind:

1. Schieberegister lassen sich im Vergleich zu einer allgemeinen Speicherstruktur leicht herstellen, d.h. die Herstellungskosten sinken.
2. Durch die Einschränkung auf lineare Operationen kann man das Instrumentarium der linearen Algebra anwenden.

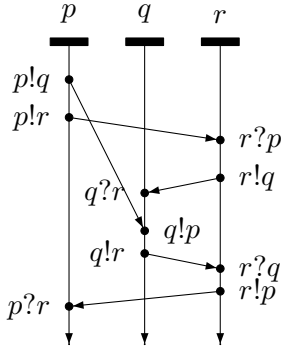
In meinen Vortrag möchte ich auf die Frage eingegen, in wie weit die Güte des Codes durch die Einschränkung auf den linearen Fall beeinflußt wird. Dazu untersuche ich für festes k und m Codes mit maximalem d_{free}.

WELCHE KOMMUNIKATIONSPROTOKOLLE LASSEN SICH MIT ENDLICHEN AUTOMATEN BESCHREIBEN?

Dietrich Kuske
Department of Mathematics and Computer Science, University of Leicester
Leicester, LE1 7RH, England
e-mail: d.kuske@mcs.le.ac.uk

Message sequence charts (MSCs) form a popular visual formalism used in the software development. In its simplest incarnation, an MSC depicts the desired exchange of messages and corresponds to a single partial-order execution of the system.

Definition by example



This MSC describes the behavior of three processes p, q, and r. The basic actions that these processes can perform are sending and receiving messages, e. g., $p!q$ means that process p sends a message to process q. The messages are sent via FIFO-channels, and there is precisely one FIFO channel for any pair of distinct processes. The messages are represented by diagonal arrows connecting the sending and the receiving event.

A communication protocol can then be described by a set of MSCs that denote the correct behaviors of the components involved. Several methods to specify communication protocols, and therefore sets of MSCs, have been considered, among them MSC-graphs or High-level MSCs (HMSCs) that generate sets of MSCs by concatenating "building blocks", Büchi automata that accept the linear extensions of MSCs, and logics. In general, these formalisms have different expressive power.

In [1], Alur & Yannakakis show that the collection of MSCs generated by a "bounded" ("locally synchronized" in the terminology of [7]) MSC-graph can be represented as a string language recognizable by a finite deterministic automaton (which is false and even undecidable in general). Based on this observation, Henriksen et al. [7] study sets of MSCs whose linear extensions form a regular string language. I will call these sets of MSCs "recognizable". Notions of recognizability has proven to be robust and fruitful in different settings like strings, trees, Mazurkiewicz traces and other classes of partial orders (both finite and infinite). The robustness is reflected by the fact that there are alternative definitions of recognizable sets of objects using combinatorial, algebraic, or logical methods: in all these settings, recognizable sets can be presented by finite-state devices, by congruences of finite index, or by sentences of monadic second order logic. Even more, natural subclasses of monadic second order logic correspond to natural classes of

congruences and finite-state devices (e. g., first-order logic corresponds to groupfree syntactic monoids and to counterfree automata). The main results in [7] show similar equivalences for bounded sets of finite MSCs. In particular, they prove the equivalence of the following three concepts for bounded sets K of finite MSCs:

1. The set of linear extensions of K can be accepted by a finite deterministic automaton.
2. There is a sentence φ of the monadic second order logic such that K is the set of bounded MSCs that satisfy φ.
3. The set K can be accepted by a nondeterministic message passing automaton.

This result was sharpened in [15] where it is shown that deterministic message passing automata suffice.

The main focus of [11] (that forms the basis of the present talk) is the extension of these results to sets of infinite MSCs. These infinite MSCs occur naturally as executions of systems that are not meant to stop like distributed operating systems or telecommunication networks. In the first part, we will extend the equivalence between the first and the second statement. We will also consider two fragments of monadic second order logic, namely first-order logic FO and its extension by modulo-counting quantifiers FO+MOD(n) [16]. We describe the expressive power of these logics in the spirit of Büchi's theorem: for a bounded set of possibly infinite MSCs K, the following statements are equivalent

1. The set of linear extensions of K is recognizable (n-solvable, aperiodic, resp.).
2. The set K is axiomatizable by a sentence of monadic second order logic (of the logic FO+MOD(n), first-order logic, resp.) relative to all possibly infinite MSCs.

In order to characterize recognizable sets of infinite MSCs in terms of message passing automata, we extend the model from [7] by a Muller-acceptance condition. It is shown that for a bounded set of possibly infinite MSCs K, the following statements are equivalent

1. The set of linear extensions of K is recognizable.
3. The set K is accepted by a finite deterministic message passing automaton with Muller-acceptance condition.

The proof of the implication 1→2 relies on a first-order interpretation of an MSC in any of its linearisations. This allows us to use results from [3, 13] and [16] that characterize the expressive power of the logics in question for infinite words. The proof 2→1 for *finite* MSCs from [7] uses a first-order interpretation of the lexicographically least linear extension of t in the finite MSC t. This proof method does not extend to the current setting since in general no linear extension of order type ω can be defined in an infinite MSC. To overcome this problem, we use ideas from [17] by choping an infinite MSC into its finite and its infinite part. It turns out that the infinite part is the disjoint union of infinite posets to which the "classical" method from [7] is applicable. The proof of the implication 3→1 is an obvious variant of similar proofs for finite automata for words (cf. [18]), asynchronous automata for traces [19], or asynchronous cellular automata for pomsets without autoconcurrency [6]. Mukund et al. proved the implication 1→3 for finite MSCs. In order to do so, they had to reprove several results from the theory of Mazurkiewicz traces in the more complex realm of MSCs. Differently, my proof for infinite MSCs refers to deep results in the theory of Mazurkiewicz traces directly (cf. [4] for surveys on this theory). These results are applicable since any recognizable set of MSCs can be represented as a set of traces up to an easy relabeling. This observation has in my opinion several nice aspects: (1) it simplifies the proof, (2) it also results in smaller message passing automata for finite MSCs, and (3) it highlights the similarity of MSCs and Mazurkiewicz traces and the unifying role that Mazurkiewicz traces can play in the theory of distributed systems. This last point is also stressed by the fact that similar proof techniques have been used, e. g., in [2, 5, 10, 6, 12, 14].

References

[1] R. Alur and M. Yannakakis. Model checking of message sequence charts. In *CONCUR '99*, Lecture Notes in Comp. Science vol. 1664, pages 114–129. Springer, 1999.

[2] A. Arnold. An extension of the notions of traces and of asynchronous automata. *Informatique Théorique et Applications*, 25:355–393, 1991.

[3] J.R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.*, 6:66–92, 1960.

[4] V. Diekert and G. Rozenberg. *The Book of Traces*. World Scientific Publ. Co., 1995.

[5] M. Droste and P. Gastin. Asynchronous cellular automata for pomsets without autoconcurrency. In *CONCUR '96*, Lecture Notes in Comp. Science vol. 1119, pages 627–638. Springer, 1996.

[6] M. Droste, P. Gastin, and D. Kuske. Asynchronous cellular automata for pomsets. *Theoretical Comp. Science*, 247:1–38, 2000. (Fundamental study).

[7] J.G. Henriksen, M. Mukund, K. Narayan Kumar, and P.S. Thiagarajan. Towards a theory of regular MSC languages. Technical report, BRICS RS-99-52, 1999. The results of this technical report appeared in the extended abstracts [9, 8].

[8] J.G. Henriksen, M. Mukund, K. Narayan Kumar, and P.S. Thiagarajan. On message sequence graphs and finitely generated regular MSC languages. In *ICALP '00*, pages 675–686. Springer, 2000.

[9] J.G. Henriksen, M. Mukund, K. Narayan Kumar, and P.S. Thiagarajan. Regular collections of message sequence charts. In *MFCS 2000*, Lecture Notes in Computer Science vol. 1893. Springer, 2000.

[10] D. Kuske. Asynchronous cellular automata and asynchronous automata for pomsets. In *CONCUR '98*, Lecture Notes in Comp. Science vol. 1466, pages 517–532. Springer, 1998.

[11] D. Kuske. Another step towards a theory for regular MSC languages. Technical Report 2001-36, Department of Mathematics and Computer Science, University of Leicester, 2001.

[12] D. Kuske and R. Morin. Pomsets for local trace languages: Recognizability, logic and Petri nets. In *CONCUR 2000*, Lecture Notes in Comp. Science vol. 1877, pages 426–411. Springer, 2000.

[13] R.E. Ladner. Application of model theoretic games to discrete linear orders and finite automata. *Information and Control*, 33:281–303, 1977.

[14] R. Morin. On regular message sequence chart languages and relationships to mazurkiewicz trace theory. In *FOSSACS01*, Lecture Notes in Comp. Science vol. 2030, pages 332–346. Springer, 2001.

[15] M. Mukund, K. Narayan Kumar, and M. Sohoni. Synthesizing distributed finite-state systems from MSCs. In C. Palamidessi, editor, *CONCUR 2000*, Lecture Notes in Computer Science vol. 1877, pages 521–535. Springer, 2000.

[16] H. Straubing, D. Thérien, and W. Thomas. Regular languages defined with generalized quantifiers. *Information and Computation*, 118:289–301, 1995.

[17] P.S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic for mazurkiewicz traces. In *LICS '97*, pages 183–194. IEEE Computer Society Press, 1997.

[18] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 133–191. Elsevier Science Publ. B.V., 1990.

[19] W. Thomas. On logical definability of trace languages. In V. Diekert, editor, *Proceedings of a workshop of the ESPRIT BRA No 3166: Algebraic and Syntactic Methods in Computer Science (ASMICS) 1989*, Report TUM-I9002, Technical University of Munich, pages 172–182, 1990.

AUF ZELLULARAUTOMATEN BASIERENDE BILDERZEUGUNG UND -KOMPRESSION (FORTSETZUNG)

Jan-Thomas Löwe
Institut für Informatik, Universität Giessen
Arndtstr. 2, D-35392 Giessen, Germany
e-mail: Jan-Thomas.Loewe@informatik.uni-giessen.de

Zur Speicherung von Bildern in Computersystemen werden diese normalerweise durch eine Sequenz von Helligkeitswerten für jedes einzelne Pixel und jede einzelne Farbe angegeben bzw. beschrieben. Ein wesentlicher Nachteil dieser Beschreibungsform besteht im hohen Speicherbedarf, der für jedes Bild benötigt wird. Interessanter sind dabei Methoden, die gewisse Eigenschaften von Bildern ausnutzen und die Informationsmenge somit auf das Wesentliche reduzieren, d. h. das Bild komprimieren.

Es gibt mehrere universelle Kompressions-Algorithmen, die auf einen Datenstrom angewendet werden können, z. B. die LZ-Kompression in [3]. Doch diese berücksichtigen nicht einige besondere Eigenschaften und Charakteristiken von Bildern.

In [1, 2] benutzen Culik und Kari einen Kompressionsalgorithmus, der auf (gewichteten) endlichen Automaten basiert. Der endliche Automaten beschreibt dabei das ursprüngliche Bild, indem Bilder als endliche Sprache aufgefaßt werden. Kompression bedeutet, einen Automaten zu konstruieren, der ein Eingabewort erhält, das der Postion eines Pixels entspricht, und eine Ausgabe produziert (einen Graustufenwert).

Zellularautomaten sind in Bezug auf Sprachverarbeitung leistungsfähiger als endliche Automaten. Aus diesem Grund sind wir daran interessiert herauszufinden, wie (eventuell auch modifizierte) Zellularautomaten zur Bilderzeugung und -kompression benutzt werden können.

Dazu betrachten wir zwei-dimensionale Zellularautomaten. Jede Zelle entspricht einem Pixel des Bildes. Das Ziel ist die Verwendung von Signalen, um eine Zelle bzw. ein Pixel einzufärben. Signale erlauben es uns Teilbilder zu kombinieren, skalieren, rotieren um daraus neue Teilbilder zu generieren. Daraus erhoffen wir uns Speicher für Teilbilder einzusparen, die aus schon gespeicherten generiert werden können.

Literatur

[1] K. Culik II, J. Kari *Image Compression Using Weighted Finite Automata.* Comput. Graphics 17 (1993), 305–313.

[2] K. Culik II, J. Kari *Inference Algorithms for WFA and Image Compression.* In Fractal Image Compression, ed. Y. Fisher, Springer, 1994, 245–262.

[3] J. Ziv, A. Lempel *A Universal Algorithms for Sequential Data Compression.* IEEE Transactions on Information Theory Vol. IT-23 No. 3 (1977) 337–343.

SHRINKING ALTERNATING TWO-PUSHDOWN AUTOMATA

Friedrich Otto
FB Mathematik/Informatik, Universität Kassel
D-34109 Kassel, Germany
e-mail: otto@theory.informatik.uni-kassel.de

and

Etsuro Moriya
Department of Mathematics, School of Education, Waseda University
Tokyo, Japan
e-mail: moriya@gauss.em.cache.waseda.ac.jp

It is well-known that the pushdown automaton with two pushdown stores, the so-called *two-pushdown automaton* (TPDA), is as powerful as the Turing machine, and hence, it is a universal model of a computing device. In [2] a restricted variant of the TPDA is considered, the so-called *shrinking* TPDA. A weight function g is chosen that assigns a positive weight to each symbol of the stack alphabet and to each state symbol. By adding up the weights this yields a weight $g(C)$ for each configuration C of the TPDA M considered. Now M is called *shrinking* if there exists a weight function g such that each step of M strictly decreases the weight of the actual configuration. It is shown in [2] that a language can be accepted by a shrinking TPDA (sTPDA) if and only if this language is growing context-sensitive. The class GCSL of growing context-sensitive languages has first been investigated by Dahlhaus and Warmuth [5], who proved that GCSL is contained in the complexity class LOG·CFL of languages that are log-space reducible to context-free languages. Buntrock and Loryś proved that GCSL is an *abstract family of languages*, that is, it is closed under union, concatenation, iteration, ε-free morphisms, inverse morphisms, and intersection with regular sets [1]. On the other hand, it has been observed that the so-called *Gladkij language*

$$L_{\mathrm{Gl}} := \{\, w\#w^R\#w \mid w \in \{a,b\}^* \,\}$$

is not growing context-sensitive [3], where w^R denotes the reversal of the string w.

The deterministic variant of the sTPDA accepts exactly the *Church-Rosser languages* CRL [9], and so this class can be interpreted as the deterministic variant of the class GCSL. However, the class CRL also admits some other interesting characterizations. In various papers Jančar, Mráz, Plátek, and Vogel present and investigate the *restarting automaton* RRWW and many variants of it (see, e. g., [7]). Niemann and Otto show that the deterministic RWW- and the deterministic RRWW-automata yield other characterizations of the class CRL [11, 10], and that GCSL is contained in the class $\mathcal{L}$(RWW) of languages accepted by RWW-automata. In [6] it is shown that L_{Gl} is accepted by some RWW-automaton, thus showing that GCSL is properly contained in $\mathcal{L}$(RWW). There it is further shown that RWW-automata accept some NP-complete languages, which shows that the closure of the class $\mathcal{L}$(RWW) under log-space reductions coincides with the complexity class NP.

Alternation is a powerful generalization of nondeterminism. In fact, alternating polynomial time is equivalent to deterministic polynomial space, and alternating polynomial space is equivalent to deterministic exponential time [4]. Also the alternating versions of pushdown automata and context-free grammars have been investigated. It turned out that the alternating pushdown automata characterize the complexity class $\bigcup_{c>0} \mathrm{DTIME}(c^n)$ [4], and in [8] the question is addressed of whether the alternating context-free grammars are as expressive as the alternating pushdown automata.

Here we will present the alternating variant of the shrinking TPDA, the *shrinking alternating* TPDA (sATPDA). We will show that the increase in computational power obtained by alternation is not as big as one might expect, as the sATPDA only accepts languages that are deterministic context-sensitive. This is done by providing a simulation of an sATPDA by a deterministic, linear space-bounded Turing machine. On the other hand, we will see that a suitably encoded version of the language QBF of quantified Boolean formulas, which is known to be PSPACE-complete, is accepted by some sATPDA, which implies that the closure of $\mathcal{L}(\mathsf{sATPDA})$ under log-space reductions yields the complexity class PSPACE. In this way we obtain the following sequence of inclusions, where DRWW and DRRWW denote the deterministic RWW- and RRWW-automata, respectively:

$$\begin{array}{lllllll}
\mathsf{CRL} & = \mathcal{L}(\mathsf{sDTPDA}) & = \mathcal{L}(\mathsf{DRWW}) & = \mathcal{L}(\mathsf{DRRWW}) & \subset \mathcal{L}(\mathsf{sTPDA}) \\
& = \mathsf{GCSL} & \subset \mathsf{LOG{\cdot}CFL} & = \mathsf{LOG{\cdot}GCSL} & \subseteq \mathrm{P} \\
& \subseteq \mathrm{NP} & = \mathsf{LOG} \cdot \mathsf{RWW} & = \mathsf{LOG} \cdot \mathsf{RRWW} & \subseteq \mathrm{PSPACE} \\
& = & \mathsf{LOG} \cdot \mathcal{L}(\mathsf{sATPDA}). & & &
\end{array}$$

Finally, it should be pointed out that the language class $\mathcal{L}(\mathsf{sATPDA})$ is an abstract family of languages that in addition is closed under complementation, and therewith under intersection, and reversal.

References

[1] G. Buntrock and K. Loryś. On growing context-sensitive languages. In W. Kuich, editor, *Proc. of 19th ICALP*, Lecture Notes in Computer Science 623, pages 77–88. Springer-Verlag, Berlin, 1992.

[2] G. Buntrock and F. Otto. Growing context-sensitive languages and Church-Rosser languages. *Information and Computation*, 141:1–36, 1998.

[3] G. Buntrock. *Wachsende kontext-sensitive Sprachen.* Habilitationsschrift, Fakultät für Mathematik und Informatik, Universität Würzburg, July 1996.

[4] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the ACM*, 28:114–133, 1981.

[5] E. Dahlhaus and M. Warmuth. Membership for growing context-sensitive grammars is polynomial. *Journal of Computer and System Sciences*, 33:456–472, 1986.

[6] T. Jurdzinski, K. Loryś, G. Niemann, and F. Otto. *Some observations concerning* RWW-*automata.* In preparation.

[7] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. On monotonic automata with a restart operation. *Journal of Automata, Languages and Combinatorics*, 4:287–311, 1999.

[8] E. Moriya. A grammatical characterization of alternating pushdown automata. *Theoretical Computer Science*, 67:75–85, 1989.

[9] G. Niemann and F. Otto. The Church-Rosser languages are the deterministic variants of the growing context-sensitive languages. In M. Nivat, editor, *Foundations of Software Science*

and Computation Structures, Proceedings FoSSaCS'98, Lecture Notes in Computer Science 1378, pages 243–257. Springer-Verlag, Berlin, 1998.

[10] G. Niemann and F. Otto. *Further results on restarting automata*. Submitted for publication, August 2000.

[11] G. Niemann and F. Otto. Restarting automata, Church-Rosser languages, and representations of r.e. languages. In G. Rozenberg and W. Thomas, editors, *Developments in Language Theory – Foundations, Applications, and Perspectives, Proceedings DLT 1999*, pages 103–114. World Scientific, Singapore, 2000.

DAS WORTPROBLEM REGULÄRER AUSDRÜCKE MIT DURCHSCHNITT IST VOLLSTÄNDIG FÜR LOGCFL

Holger Petersen
Institut für Informatik, Universität Stuttgart
Breitwiesenstr. 20–22, D-70565 Stuttgart, Germany
e-mail: petersen@informatik.uni-stuttgart.de

Wir zeigen, dass die Erkennung kontextfreier Sprachen auf das Wortproblem regulärer Ausdrücke mit Durchschnitt reduziert werden kann. Die Reduktion lässt sich deterministisch in logarithmischem Platz und linearer Zeit ausführen. Somit ist dieses Wortproblem mindestens so schwer wie die Erkennung kontextfreier Sprachen. Ein LOGCFL-Algorithmus verbessert die bekannte obere Schranke NC^2, womit das Wortproblem vollständig für LOGCFL ist. Bei Einschränkung auf einelementige Alphabete ist das Problem schwer für NL.

ÜBER DIE MULTIPARTY-KOMMUNIKATIONSKOMPLEXITÄT REGULÄRER SPRACHEN

Pavel Pudlak
Mathematical Institute, Academy of Sciences of the Czech Republic
e-mail: pudlak@math.cas.cz

Klaus Reinhardt
Institut für Informatik, University of Tübingen
e-mail: reinhard@informatik.uni-tuebingen.de

und

Pascal Tesson und Denis Thérien
School of Computer Science, McGill University
e-mail: {ptesso, denis}@cs.mcgill.ca

ABSTRACT

Gemäss dem algebraischen Ansatz von [4] untersuchen wir die Kommunikationskomplexität aperiodischer Monoide im Multiparty-model von Chandra, Furst und Lipton. Wir stellen eine Vermutung vor, die die aperiodischen Monoide, die beschränkte Kommunikationskomplexität haben, charakterisieren würde und zeigen verschiedene Ergebnisse in dieser Richtung.

1. Einleitung

1.1. Kommunikationskomplexität

Wir betrachten Kommunikationsspiele mit k-Spielern (siehe [2]), bei denen n Eingabevariablen in k Mengen $[n] = X_1 \dot\cup \cdots \dot\cup X_k$ unterteilt sind, wobei der Spieler P_i Zugang zu allen Variablen ausser denen in X_i hat.

Seien A und B endliche Mengen. Ziel der Spieler ist es, gemeinsam eine Funktion $f : A^n \to B$ zu berechnen, wobei es ihnen erlaubt ist, Bits nach einem bestimmten Protokoll an eine Tafel zu schreiben, die für jeden Spieler sichtbar ist. Das Protokoll terminiert, wenn jeder Spieler den Wert von f kennt. Die *Kosten* eines Protokolles für f sind definiert durch die maximale Anzahl von Bits, die von den Spielern für eine Eingabe bei der schlechtesten Art der Verteilung zu kommunizieren ist. Die k-Spieler *Kommunikationskomplexität* $C^{(k)}(f)$ von f sind die Kosten des billigsten k-Spieler-Protokolls, das f berechnet. Aus der Definition folgt unmittelbar $C^{(k)}(f) \geq C^{(k')}(f)$ für $k < k'$ und $C^{(k)}(f) \leq \frac{n}{k} + 1$.

1.2. Algebra

Für zwei endliche Monoide[1] M und N gelte $M \prec N$, falls M das homomorphe Bild eines Untermonoides von N ist. Eine Sprache $L \subseteq X^*$ wird durch M erkannt, wenn ein Morphismus

[1] Assoziative Halbgruppen mit neutralem Element, siehe [1], [3]

$\phi : X^* \to M$ existiert mit $L = \phi^{-1}(F)$ für eine Teilmenge $F \subseteq M$. Eine Sprache L ist regulär, wenn sie durch ein endliches M erkannt werden kann.

Für eine reguläre Sprache $L \subseteq X^*$ und ein Wort x definieren wir die Menge $Y_x = \{(u, v) : uxv \in L\}$. Durch $x \sim_L y$ genau dann wenn $Y_x = Y_y$ erhalten wir eine Kongruenz mit endlichem Index. Das syntaktische Monoid $X^*/\sim_L$ von L bezeichnen wir als $M(L)$. Dieses Monoid $M(L)$ erkennt L und für alle N, die ebenfalls L erkennen, gilt $M(L) \prec N$.

Die k-Spieler Kommunikationskomplexität $C^{(k)}(M)$ von M ist die maximale Komplexität einer regulären Sprache, die von M erkannt werden kann. Es wurde in [4] gezeigt, dass dies bis auf einen konstanten Faktor der Komplexität, ein Produkt von Elementen in M auszuwerten, entspricht. Ferner gilt $C^{(k)}(M) = \Theta(C^{(k)}(L))$ für alle L mit $M(L) = M$ vorausgesetzt L hat einen neutralen Buchstaben.

DA ist die Menge der Monoide mit $(xyz)^t y (xyz)^t = (xyz)^t$, für ein $t \geq 0$. **DA** hat viele nützliche Charakterisierungen und steht im Zentrum vieler Ergebnisse über Komplexitätsfragen über aperiodische Monoide. Insbesondere benützen wir die Eigenschaft, dass ein aperiodisches Monoid M genau dann ausserhalb von **DA** liegt, wenn es entweder von BA_2, dem syntaktischen Monoid der regulären Sprache $(ab)^*$, oder von U, dem syntaktischen Monoid von A^*aaA^* geteilt wird.

Sei A ein endliches Alphabet und, für $t \in \mathbb{N}$ und $x \in A^*$ sei $\alpha_t(x)$ die Anzahl der Vorkommen jedes Buchstaben von A im Wort x bis zum Schwellenwert t. Wir definieren induktiv über k eine Familie von Äquivalenzrelationen $\approx_{k,t}$ auf A^* für $k, t \in \mathbb{N}$. Es gilt $x \approx_{0,t} y$ für alle $x, y \in A^*$ und alle t. Rekursiv definieren wir $x \approx_{k,t} y$ genau dann wenn

1. $x \approx_{k-1,t} y$
2. $\alpha_t(x) = \alpha_t(y)$
3. Für alle $x = x_0 a x_1$ und $y = y_0 a y_1$ mit $|x_0|_a = |y_0|_a \leq t$, gilt $x_0 \approx_{k-1,t} y_0$ und $x_1 \approx_{k-1,t} y_1$
4. Für alle $x = x_0 a x_1$ und $y = y_0 a y_1$ mit $|x_1|_a = |y_1|_a \leq t$, gilt $x_0 \approx_{k-1,t} y_0$ und $x_1 \approx_{k-1,t} y_1$

Für alle k, t ist das endliche Monoid $A^*/\approx_{k,t}$ aperiodisch für alle k, t und liefert eine Parametrisierung von **DA**:

Lemma *Sei $M = A^*/\gamma$, mit $|A| = n$ so ist $M \in$ **DA** ganau dann wenn $\approx_{k,t} \subseteq \gamma$ für ein Paar k, t.*

Es sei $\mathbf{W_k}$ die Menge der Monoide $M = A^*/\gamma$ mit $\approx_{k,t} \subseteq \gamma$ für ein t.

2. Einige Schranken der Multiparty-Kommunikationskomplexität von Monoiden

Es wurde in [4] gezeigt, dass eine Gruppe G genau dann konstante k-Spieler Kommunikationskomplexität ($k \geq 2$) hat wenn sie nilpotent der Klasse $(k-1)$ ist, andernfalls hat sie lineare Komplexität.

Vermutung *Ein aperiodisches Monoid M hat konstante k-Spieler Kommunikationskomplexität genau dann wenn es zu $\mathbf{W_{k-1}}$ gehört.*

Die „wenn"-Richtung können wir durch Konstruktion eines Protokolles zeigen, bei welchem die Spieler feststellen, welche Spieler die ersten bzw. letzten t a's haben und und jeweils rekursiv mit einem Spieler weniger die Relation $\approx_{k-1,t}$ prüfen. Ein Zählen (bis t) ist bereits mit zwei Spielern möglich.

In der umgekehrten Richtung haben wir nur Teilergebnisse. Wir zeigen zunächst, dass ein Monoid, das nicht zu $\bigcup_k \mathbf{W_k} = \mathbf{DA}$ gehört, keine konstante k-Spieler Kommunikationskomplexität für irgend ein k besitzt, was ein offenes Problem in [4] löst. Nach obiger Bemerkung genügt es, untere Schranken für BA_2 und für U zu zeigen.

Theorem *$C^{(k)}(BA_2)$ is nicht konstant für jedes k.*

Es gilt $C^{(k)}(U) \in \omega(\log n)$ laut [4]. Die Vermutung in [4] dass alle aperiodischen Monoide, die nicht zu **DA** gehören, lineare Kommunikationskomplexität hätten, widerlegen wir wie folgt:

Theorem *Die Sprache $(ab)^*$ kann durch 4 Spieler mit $O(\sqrt{n}\log(n))$ Kommunikationsbits und durch 5 Spieler mit $O(\log(n))$ Kommunikationsbits erkannt werden.*

2.1. Der 3-Spieler-Fall

Unsere Vermutung ist trivial für $k = 1$. Für $k = 2$, wissen wir aus [4, 5] dass aperiodische Monoide genau dann konstante 2-Kommunikationskomplexität besitzen, wenn sie kommutativ sind, was der Definition von $\mathbf{W_1}$ entspricht. Wir können die Vermutung auch für $k = 3$ zeigen.

Lemma *Ein Monoid M gehört zu $\mathbf{W_2}$ genau dann wenn ein t existiert so dass für alle $x, y, z \in M$ sowohl $x^{t+1} = x^t$ als auch $x^t y x^t z x^t = x^t y z x^t$ gilt.*

Sei $f : \{0,1\}^n \times \{0,1\}^{\log n} \times \{0,1\}^{\log n} \to \{0,1\}$ definiert durch $f(\alpha, \beta, \gamma) = 1$ genau dann wenn $\beta < \gamma$ und in dem n-Bit Wort α die Bits $\alpha_\beta, \alpha_{\beta+1}, \ldots, \alpha_\gamma$ alle gleich 0 sind.

Theorem $C^{(3)}(f) = \omega(1)$

Folgerung *Ein aperiodisches Monoid M hat konstante 3-Spieler Kommunikationskomplexität genau dann wenn es zu $\mathbf{W_2}$ gehört.*

Vermutung *Für alle k gilt: Die reguläre Sprache $A^*a_1A^*a_2 \ldots A^*a_kA^*$, mit $A = \{a_1, \ldots, a_k\}$ hat nicht-konstante k-Spieler Kommunikationskomplexität.*

Literatur

[1] S. Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, 1976.

[2] E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, 1997.

[3] J.-E. Pin. *Varieties of formal languages.* North Oxford Academic Publishers Ltd, London, 1986.

[4] J.-F. Raymond, P. Tesson, and D. Thérien. An algebraic approach to communication complexity. *Lecture Notes in Computer Science*, 1443:29–40, 1998.

[5] M. Szegedy. Functions with bounded symmetric communication complexity, programs over commutative monoids, and ACC. *J. Comput. Syst. Sci.*, 47(3):405–423, 1993.

DREIBEINIGE PCP-BIBER

Mario Schmidt, Heiko Stamer und Johannes Waldmann
Institut für Informatik, Universität Leipzig
Augustusplatz 10–11, D-04109 Leipzig, Germany
e-mail: pcp@informatik.uni-leipzig.de

ABSTRACT

We look for small PCP instances that are hard: They consist of only a few pairs of short words, but their minimal solution is very long.

In particular, we collect information on PCP instances consisting of three pairs. We list the known record holders, then describe a family of hard instances (related to D0L systems), and finally conjecture an upper bound for hardness.

Schlagwörter: Post correspondence problem, busy beaver.

1. Einleitung

Oft kann man dadurch Informationen über eine Problemklasse gewinnen, daß man nach kleinen, aber möglichst schweren Instanzen sucht. Ein klassisches Beispiel ist die Suche nach Busy-Beaver-Turingmaschinen [4].

Gelingt es zudem, die Schwere einer Instanz durch eine berechenbare Funktione ihrer Größe zu beschränken, erhält man Aussagen über Entscheidbarkeit und Komplexität.

Wir wenden diese Technik auf eingeschränkte Klassen des Postschen Korrespondenz-Problems (PCP) an.

2. Bezeichnungen und Voraussetzungen

Eine *PCP-Instanz* ist eine endliche Liste $[(u_1, v_1), \ldots, (u_n, v_n)]$ von Wörtern $u_i, v_i \in \Sigma^+$, die zwei Morphismen $\phi, \psi : \Gamma^* \to \Sigma^*$ (mit $\Gamma = \{1, \ldots, n\}$) durch $\phi(i) = u_i,\ \psi(i) = v_i$ bestimmen. Eine *Lösung* einer Instanz (ϕ, ψ) heißt jedes nichtleere Wort aus der Menge $E(\phi, \psi) = \{w : \phi(w) = \psi(w)\}$. Es ist wohlbekannt, daß die Menge der lösbaren Instanzen nicht entscheidbar ist [6].

Die *Größe* einer Instanz ist die Anzahl ihrer Wortpaare, d. h. $|\Gamma|$. Wir bezeichnen mit PCP(n) die Menge aller lösbaren Instanzen der Größe n. Die Menge PCP(2) ist entscheidbar [1], aber PCP(7) nicht [5]. Wir interessieren uns hier für PCP(3). (Im folgenden betrachten wir nur Instanzen mit Ziel-Alphabet $\Sigma = \{0, 1\}$.)

Die *Weite* einer Instanz ist die größte vorkommende Wortlänge $\max\{|u_i|, |v_i|\}$. Die Menge aller lösbaren Instanzen mit Größe n und Weite w heißt PCP(n, w). (Diese Mengen sind endlich.) Wir fragen nach den Instanzen aus PCP($3, w$), für die die Länge einer kürzesten Lösung möglichst groß wird.

Beispiel 1 *Die Instanz*

$$P_1 = \begin{pmatrix} 110 & 0 & 1 \\ 1 & 1 & 01 \end{pmatrix}$$

hat die kürzeste Lösung AACBCCBC.

Zu jeder Instanz $P = (\phi, \psi : \Gamma \to \Sigma)$ definieren wir den unendlichen, gerichteten, kantenmarkierten Graphen $G(P)$: seine Knoten sind alle möglichen *Differenzen* $\{(p, q) : p, q \in \Sigma^*, p = \epsilon \vee q = \epsilon\}$, und er enthält eine Kante $(p, q) \xrightarrow{i} (p', q')$ genau dann, wenn $\exists c \in \Sigma^* : p \cdot u_i = c \cdot p' \wedge q \cdot v_i = c \cdot q'$. Eine Lösung von P ist dann ein nichtleerer Pfad von START=(ϵ, ϵ) zurück zu START in $G(P)$.

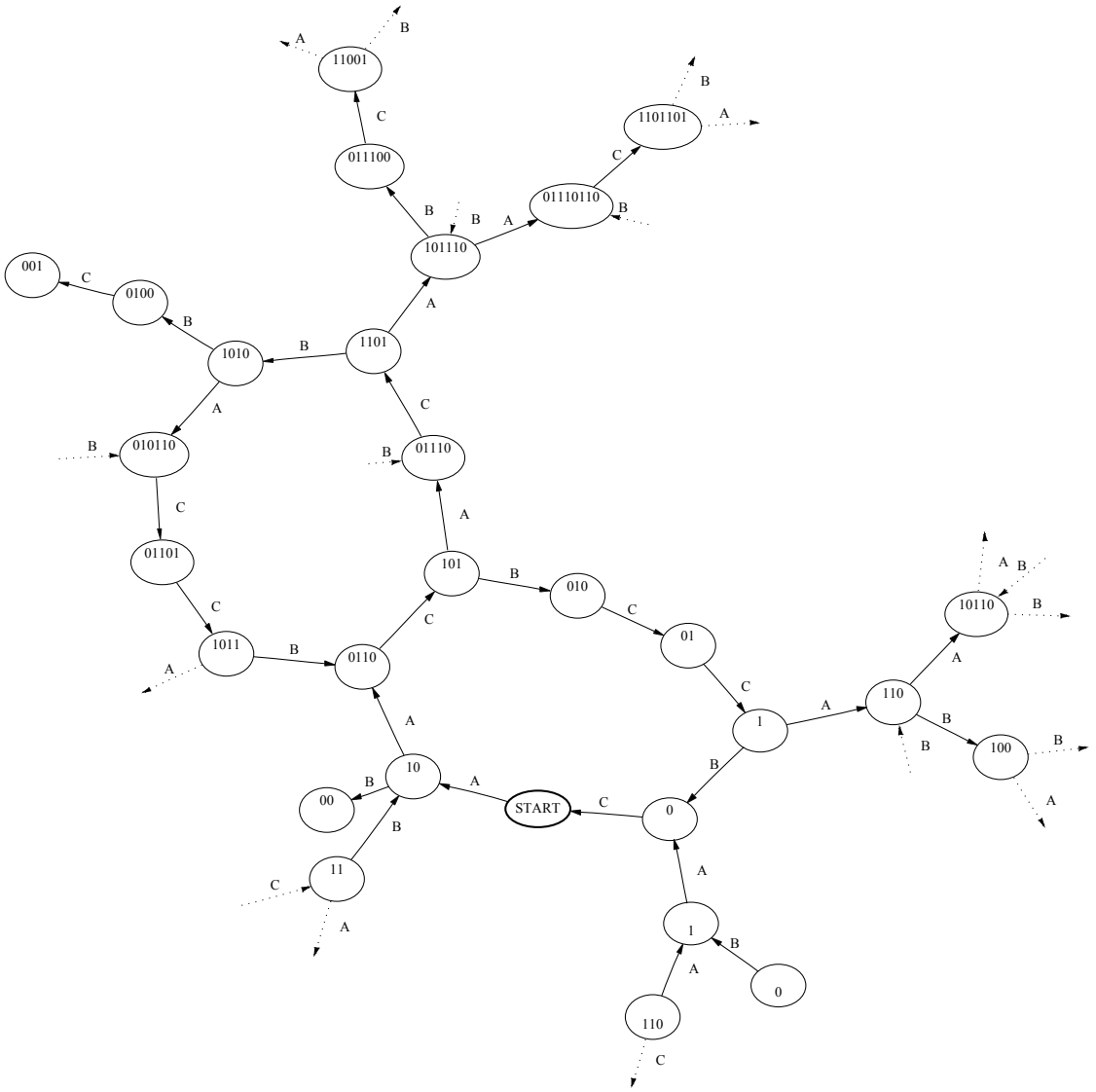


Abbildung 1: von START erreichbare Knoten (der Weite ≤ 6) in $G(P_1)$

Bei der maschinellen Suche nach Lösungen einer Instanz kommt es darauf an, die Menge der bereits betrachteten Knoten in $G(P)$ effizient zu verwalten.

3. Schwierige PCP-Instanzen

Wir listen in der Tabelle 1 die derzeitigen Rekordhalter auf. (Aktuelle Informationen sind immer auf der Webseite `http://www.informatik.uni-leipzig.de/~pcp/`)

(Größe,Weite)	Instanz	kürzeste Lösung	Autor
(3,3)	$\begin{pmatrix} 0 & 1 & 001 \\ 001 & 0 & 1 \end{pmatrix}$	75	Lorentz [3], Waldmann
(3,4)	$\begin{pmatrix} 101 & 1 & 010 \\ 1 & 01 & 1101 \end{pmatrix}$	216	Zhao [7]
(3,5)	$\begin{pmatrix} 11010 & 1 & 11111 \\ 11 & 10101 & 01 \end{pmatrix}$	189	Stamer
(4,3)	$\begin{pmatrix} 000 & 0 & 11 & 10 \\ 0 & 111 & 0 & 100 \end{pmatrix}$	204	Lorentz [3]
(4,4)	$\begin{pmatrix} 1010 & 11 & 0 & 01 \\ 100 & 1011 & 1 & 0 \end{pmatrix}$	256	Zhao [7]

Tabelle 1: Rekorde schwieriger PCP-Instanzen

4. Spezielle PCP(3)

Es ist uns aufgefallen, daß viele schwere PCP(3)-Instanzen die Form $M(u,v) = \begin{pmatrix} 0 & 1 & u \\ v & 0 & 1 \end{pmatrix}$ haben, beispielsweise das Rekord-PCP(3,3). (Dort gilt sogar $u = v$. Siehe auch [2] über eine Verallgemeinerung.)

Um eine Lösung für $M(u,v)$ zu suchen, können wir, von links beginnend, beliebig lange ausschließlich die ersten beiden Paare benutzen. Wir erhalten so Anfangsstücke des vom Morphismus $0 \mapsto v, 1 \mapsto 0$ erzeugten unendlichen D0L–Wortes. Wir können auch von rechts beginnen und ausschließlich die letzten beiden Paare benutzen, und erhalten (gespiegelt) Anfangsstücke des D0L-Wortes von $0 \mapsto 1, 1 \mapsto \overline{u}$. Die Frage dabei ist, ob und wie sich diese beiden Wörter schließlich treffen. Die Lösungen haben im Allgemeinen jedoch *nicht* die Form $\{A,B\}^*\{B,C\}^*$.

Ein Spezialfall dieses Musters sind Fibonacci-Instanzen $F(u) = M(u, 01)$. Die ersten beiden Paare ergeben das Fibonacci-Wort $f = 01001010010\ldots$ Die Instanz $F(1001)$ besitzt die minimale Lösungslänge 78, und $F(001001)$ erreicht 120. Es ist klar, daß $F(u)$ nur dann lösbar ist, wenn u ein Faktor von f ist. Diese Bedingung ist jedoch nicht ausreichend, denn beispielweise $F(1001001)$ ist unlösbar.

5. Notwendige Bedingungen für lösbare Instanzen

Wir sind dabei, maschinell *alle* Instanzen aus PCP(3,3) auf Lösbarkeit hin zu überprüfen. Dabei kommt es natürlich darauf an, Lösungen schnell zu finden, aber auch darauf, unlösbare Instanzen schnell zu erkennen.

Bei PCP(3) kann man aus den Parikh-Vektoren der u_i, v_i *zwei* homogene lineare Gleichungen für die *drei* Elemente das Parikh-Vektors jeder Lösung bestimmen. D. h. das Verhältnis der Buchstabenanzahlen A, B, C in der Lösung steht (bis auf pathologische Fälle) von vornherein fest, und kann zur Beschleunigung der Suche benutzt werden.

6. PCP-Familien und Wachtumsraten

Lorentz [3] bemerkt, daß die Familie $\begin{pmatrix} 0 & 10^n \\ 0^n 1 & 0 \end{pmatrix}$ mit minimalen Lösungen $A^n B^n$ das stärkste bekannte Wachstum (der Länge einer kürzesten Lösung, betrachtet als Funktion der Weite der Instanz) innerhalb PCP(2) erreicht. Jedoch scheint es keinen einfachen Beweis dafür zu geben,

daß minimale PCP(2)-Lösungen nur linear lang sind, denn das würde einen einfachen Beweis der Entscheidbarkeit von PCP(2) implizieren.

Welches sind schnell wachsende PCP(3)-Familien? Wir erreichen quadratisches Wachstum durch $\begin{pmatrix} 0 & 0^n & 1^n0 \\ 00 & 1 & 0 \end{pmatrix}$ mit minimalen Lösungen $A^{n^2}B^nC$, und konnten das bisher nicht übertreffen.

Es gibt einige Familien, deren Verhalten nicht so offensichtlich ist. Zum Beispiel ist die Lösungslänge von $\begin{pmatrix} 11010 & 1 & 1^n \\ 11 & 10101 & 01 \end{pmatrix}$ dann groß, wenn 2 eine primitive Wurzel modulo $4n-1$ ist. Das ist für $n = 5$ der Fall und gibt das derzeitige Rekord-PCP(3,5). Die Lösungslänge ist jedoch selbst dann quadratisch beschränkt.

7. Zusammenfassung und Ausblick

Wir vermuten, daß PCP(3) entscheidbar ist. Die bisher vorliegenden Daten legen Beweisversuche aus zwei Richtungen nahe:

1. *spezielle* PCP(3):
   (a) Zeige, daß die Lösbarkeit von $M(u,v) = \begin{pmatrix} 0 & 1 & u \\ v & 0 & 1 \end{pmatrix}$ entscheidbar ist.
   Spezialfall: zeige, daß Fibonacci-PCPs $F(u) = M(u, 01)$ entscheidbar sind.
   (b) Zeige, daß sich andere PCP(3) auf die Form $M(u,v)$ reduzieren lassen.
   Zum Beispiel: welcher Zusammenhang besteht zwischen dem Rekord-PCP(3,4) $\begin{pmatrix} 101 & 1 & 010 \\ 1 & 01 & 1101 \end{pmatrix}$ und dem Fibonacci-PCP $F(10100101) = \begin{pmatrix} 0 & 1 & 10100101 \\ 01 & 0 & 1 \end{pmatrix}$?
   Beide haben eine kürzeste Lösung der Länge 216.
2. *Wachstum*:
   (a) Zeige, daß eine kürzeste Lösung von PCP$(3, w)$ die Länge $O(w^2)$ besitzt – oder finde eine stärker wachsende Familie.

In jedem Fall würde eine vergrößerte Sammlung von Busy-Beaver-Instanzen und -Instanz-Familien weiterhelfen. Wir laden deswegen alle Interessenten herzlich ein, diese Tiere zu jagen und an unser Museum zu schicken.

Fordern Sie auch Ihre Studenten zur Mitarbeit auf! Das Thema eignet sich nach unserer Erfahrung sehr gut zum Erlernen und Ausprobieren von implementierungs-technischen, aber auch formal-sprachlichen Methoden.

Auf unserer Webseite `http://www.informatik.uni-leipzig.de/~pcp/` finden Sie neben den aktuellen Rekorden auch ein Online-PCP-Puzzlespiel, für das täglich neue (mittelschwere) Instanzen generiert werden.

Literatur

[1] A. Ehrenfeucht, J. Karhumaki, and G. Rozenberg. The (generalized) post correspondence problem with lists consisting of two words is decidable. *Theoretical Computer Science*, 21(2):119–144, November 1982.

[2] Vesa Havala and Tero Harju. Some new results on post correspondence problem and its modifications. TUCS Technical Report 338, Turku Centre for Computer Science, January 2001. `http://www.tucs.fi/publications/techreports/TR338.html`.

[3] Richard J Lorentz. Creating difficult instances of the post correspondence problem. 2nd International Conference on Computers and Games, Hamamatsu, 2000. `http://www.etl.go.jp/etl/suiron/~ianf/cg2000/`.

[4] Heiner Marxen. Busy beaver. `http://www.drb.insel.de/~heiner/BB/index.html`, 2000.

[5] Yuri Matiyasevich and Géraud Sénizergues. Decision problems for semi-Thue systems with a few rules. In *Proceedings, 11*th *Annual IEEE Symposium on Logic in Computer Science*, pages 523–531, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press.

[6] Emil Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52:264–268, 1946.

[7] Ling Zhao. Web pages on posts correspondence problem. 2001. `http://games.cs.ualberta.ca/~zhao/PCP/intro.htm`.

POSITIVE VALENCE GRAMMARS

Ralf Stiebe
Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg
PSF 4120, D-39016 Magdeburg, Germany
e-mail: stiebe@iws.cs.uni-magdeburg.de

ABSTRACT

We consider grammars whose productions are valuated by integer vectors. A derivation is legal iff the sum of the valuations is non-negative at any time and equal to zero at the end. We show that positive valence grammars with context-free rules are equivalent to matrix grammars, investigate their Parikh languages and discuss some restrictions.

1. Introduction

In [3], Greibach introduced two kinds of restricted multicounter machines. A *blind k-counter machine* is a nondeterministic finite automaton whose state transitions are equipped with k-dimensional integer vectors. An input word is accepted iff there is a run that reaches a final state such that the sum of the vectors of the applied transitions is the zero vector. A *partially blind k-counter machine* is defined in the same way; additionally, it is required that the sum of the vectors is positive at any time of an accepting run.

Valence grammars, independently introduced by Păun [4], combine the mechanisms of a grammar and a blind multicounter machine. Due to their simplicity and to several nice properties regarding closure under operations and decidability, valence grammars have found attention in several recent publications.

In this paper, we are going to discuss an analogous combination of grammars and partially blind multicounter machines, which will be called *positive valence grammars*. We shall show that context-free positive valence grammars are equivalent to context-free matrix grammars (without appearance checking). Then, we shall consider the Parikh sets of positive valence languages. Finally, we discuss two restrictions, namely grammars of finite index and grammars without zero test in the end.

Note that Brandenburg [1] discussed a similar variant of regulated rewriting. In particular, he obtained results analogous to Theorems 1 and 3.

2. Definitions

We assume that the reader is familiar with the basics of formal language theory, and give only definitions from regular rewriting.

First, let us fix some notations. Inclusion is denoted by "$\subseteq$", proper inclusion by "$\subset$". The sets of natural and integer numbers are denoted by $\mathbb{N}$ and $\mathbb{Z}$. The i-th unit vector in $\mathbb{Z}^k$ is $\vec{e}_i$. A derivation using the sequence of rules π is written $\alpha \stackrel{\pi}{\Rightarrow} \beta$.

A *grammar with control language* is a quintuple $G = (N, T, P, S, C)$, where (N, T, P, S) is a grammar (with nonterminals N, terminals T, production rules P, start symbol S), and C is a

language over P. The language generated by G is $L(G) = \{w \in T^* : S \stackrel{\pi}{\Rightarrow} w \wedge \pi \in C\}$. G is called a *grammar with regular control* if C is a regular language.

A *matrix grammar* (without appearance checking) is a quintuple $G = (N, T, M, S)$, where N, T, S are defined as in a grammar, and M is a finite set of matrices, i.e., finite sequences of rules (or a finite set of words over a set of rules P). The language generated by G is $L(G) = \{w \in T^* : S \stackrel{\pi}{\Rightarrow} w \wedge \pi \in M^*\}$.

A *valence grammar over* $\mathbb{Z}^k$ is a quintuple $G = (N, T, P, S, \mathbb{Z}^k)$, where N, T, S are defined as in a grammar, and P is a finite set of valence rules $(\alpha \to \beta, \vec{r})$, where $\alpha \to \beta$ is a rule and $\vec{r} \in \mathbb{Z}^k$. The direct derivation relation $\Rightarrow$ over $(N \cup T)^* \times \mathbb{Z}^k$ is defined by: $(\gamma, \vec{z}) \Rightarrow (\gamma', \vec{z}')$ iff $\gamma = \gamma_1\alpha\gamma_2$, $\gamma' = \gamma_1\beta\gamma_2$ and $\vec{z}' = \vec{z} + \vec{r}$, for some $(\alpha \to \beta, \vec{r}) \in P$. The language generated by G is $L(G) = \{w \in T^* : (S, \vec{0} \Rightarrow (w, \vec{0})$.

A *positive valence grammar* is formally defined like a valence grammar. However, the direct derivation relation $\Rightarrow$ for $G = (N, T, P, S, \mathbb{Z}^k)$ is only defined over $(N \cup T)^* \times \mathbb{N}^k$. The language generated by G is $L(G) = \{w \in T^* : (S, \vec{0} \Rightarrow (w, \vec{0})$.

The families of languages generated by the above regulated grammars are denoted by $\mathcal{L}(R, X)$, $R \in \{\mathrm{M}, \mathrm{rC}, \mathrm{Val}, \mathrm{PVal}\}$, $X \in \{\mathrm{CF}, \mathrm{CF} - \lambda, \mathrm{REG}\}$, where R specifies the type of regulation (matrix, regular control, valence, positive valence) and X specifies the type of the core rules (context-free, non-erasing context-free, regular). It is well-known that $\mathcal{L}(\mathrm{M}, X) = \mathcal{L}(\mathrm{rC}, X)$ [2].

The family of languages accepted by partially blind multicounter automata is denoted by *PBLIND*.

3. Generative Power

Theorem 1 $\mathcal{L}(\mathrm{PVal}, \mathrm{CF}) = \mathcal{L}(\mathrm{M}, \mathrm{CF})$.

Proof. Consider a positive valence grammar $G = (N, T, P, S, \mathbb{Z}^k)$. The idea is to introduce a new nonterminal for each dimension. The current value of a component is indicated by the number of appearances of the respective symbol. Formally, we construct a matrix grammar $G' = (N \cup N', T, M, S')$ with $N' = \{A_1, \ldots, A_k, S', X\}$, the start matrix $(S' \to SX)$ and the termination matrix $(X \to \lambda)$. For a valence rule $p = (A \to \alpha, \vec{r}) \in P$, $\vec{r} = (r_1, r_2 \ldots r_k)$, M contains a matrix $m(p)$ consisting of the rule $A \to \alpha$ and, for any $1 \leq i \leq k$, of r_i copies of $X \to A_iX$ if $r_i \geq 0$, or $|r_i|$ copies of the rule $A_i \to \lambda$ if $r_i < 0$.

On the other hand, let $G = (N, T, P, S, C)$ be a grammar with regular control. Let $\mathcal{A} = (\mathcal{Q}, \mathcal{P}, \mathrm{II}_\infty, \delta, \mathcal{F})$ be a finite automaton accepting C with $Q = \{q_1, q_2 \ldots, q_k\}$. We construct an equivalent positive valence grammar $G' = (N \cup \{S', X\}, T, P', S', \mathbb{Z}^k)$. For a transition (q_i, p, q_j), P' contains the valence rule $(p, \vec{e}_j - \vec{e}_i)$. Moreover, there are the start rule $(S' \to SX, \vec{e}_1)$ and the termination rules $(X \to \lambda, -\vec{e}_i)$, for all $q_i \in F$. □

The proof of the equivalence between regular grammars and finite nondeterministic automata with λ-moves can be extended, and we obtain:

Theorem 2 $\mathcal{L}(\mathrm{PVal}, \mathrm{REG}) = \mathit{PBLIND}$.

Theorem 3 $\mathcal{L}(\mathrm{PVal}, \mathrm{CF} - \lambda) = \mathcal{L}(\mathrm{PVal}, \mathrm{CF})$.

Proof. Let $G = (N, T, P, S, \mathbb{Z}^k)$ be a positive valence grammar. Without loss of generality, we can assume that N can be partitioned as $N = N_1 \cup N_2$ such that symbols from N_1 (N_2, respectively) can only generate nonempty words (the empty word, respectively).

The idea is to store the number of symbols from N_2 in $\mathrm{card}(N_2)$ additional dimensions. The application of a rule of the form $(A \to \alpha, \vec{r})$, $A \in N_2, \alpha \in N_2^*$, is simulated by a rule of the form $(B \to B, \Psi(\alpha) - \Psi(A) + \vec{r})$, where $\Psi(\alpha)$ is the vector associated to $\alpha \in N_2^*$. □

Note that the last result does not help to solve the open question whether matrix grammars with erasing rules are strictly stronger than those without erasing rules. The construction in the proof of Theorem 1 usually gives a matrix grammar with erasing rules, even for positive valence grammars without erasing rules.

4. Parikh Sets

It is well-known that Parikh languages of matrix grammars are closely related to reachability sets in Petri nets. In what follows, we gives some more results on these sets. Let $\mathcal{PL}$ be the family of Parikh sets of languages in $\mathcal{L}$.

Theorem 4 $\mathcal{PL}(\mathrm{PVal}, \mathrm{CF}) = \mathcal{PL}(\mathrm{PVal}, \mathrm{REG})$.

Proof. Again, the idea is to store the count of a nonterminal symbol in an additional dimension. □

Greibach [3] showed that linear time partially blind counter automata (accepting a word of length n in $O(n)$ steps, notation $PBLIND(lin)$) are strictly stronger than the real-time variant (performing $O(1)$ λ-moves in sequence, notation $PBLIND(n)$). We show that the respective families of Parikh languages are equivalent.

Theorem 5 $\mathcal{PL}(PBLIND(lin)) = \mathcal{PL}(PBLIND(n))$.

Proof. Let $\mathcal{A}$ be a linear time partially blind multicounter automaton with input alphabet T. Let the computation time for a word of length n be bounded by dn. We construct a partially blind multicounter automaton $\mathcal{B}$ with 2 additional counters for each input symbol. In every $(d+1)$-th step, $\mathcal{B}$ consumes a letter of the input word and increases the first counter corresponding to the symbol. In the other steps, $\mathcal{B}$ performs λ-moves and simulates an accepting run of $\mathcal{A}$ on some other input. Whenever a non-λ-move of $\mathcal{A}$ is simulated, the second counter corresponding to the symbol is increased. Finally, it is verified that, for any symbol in T, the counters have the same contents and thus the real input and the guessed input for $\mathcal{A}$ are Parikh equivalent. □

5. Two Restrictions

The first restriction we consider is the finite index. A context-free (matrix, regular control,valence, positive valence) grammar is of finite index iff there is a constant k such that every word of the generated language can be generated using only sentential forms with at most k nonterminal symbols. The language family generated by finite index grammars of regulation R and rule type X is denoted by $\mathcal{L}(R, X_{\mathrm{fin}})$. Note that $\mathcal{L}(\mathrm{M}, \mathrm{CF}_{\mathrm{fin}}) = \mathcal{L}(\mathrm{rC}, \mathrm{CF}_{\mathrm{fin}})$ [2].

Theorem 6 $\mathcal{L}(\mathrm{M}, \mathrm{CF}_{\mathrm{fin}}) \subset \mathcal{L}(\mathrm{PVal}, \mathrm{CF}_{\mathrm{fin}})$

Proof. The second part of the proof of Theorem 1 shows that a grammar with regular control of finite index can be transformed into an equivalent positive valence grammar of finite index. Thus, the inclusion follows. It is a proper one, as matrix grammars of finite index can only generate languages with semilinear Parikh sets [2], while the non-semilinear language $\{a^m b^n : 1 \leq m, 1 \leq n \leq 2^m\}$ can be generated by a regular positive valence grammar. □

Finally let us shortly discuss grammars without zero test, i.e., where a terminal word w is in the language if some pair $(w, \vec{r})$ can be generated. For core rules of type X, denote the corresponding language families by $\mathcal{L}(\mathrm{PVal} - 0, X)$.

Lemma 7 For an arbitrary core rules type X, $\mathcal{L}(\mathrm{PVal} - 0, X) \subseteq \mathcal{L}(\mathrm{PVal}, X)$.

Proof. Consider a positive valence grammar $G = (N, T, P, S, \mathbb{Z}^k)$ without zero tests. The equivalent positive valence grammar G' has just an extended set of valence rules. For any valence rule $(p, \vec{r}) \in P$, G' contains all valence rules $(p, \vec{s})$, where $\vec{r} = (r_1, \ldots, r_k)$, $\vec{s} = (s_1, \ldots, s_k)$, $0 \leq s_i \leq r_i$, if $0 \leq r_i$, and $s_i = r_i$, if $0 > r_i$.

It is now easily shown by induction that $(\alpha, \vec{s})$ can be generated by G' iff some $(\alpha, \vec{r})$, $\vec{s} \leq \vec{r}$, is generated by G. □

We finally show that the inclusion turns into equality in the case of general context-free rules, while it is strict for regular rules.

Theorem 8 $\mathcal{L}(\mathrm{PVal} - 0, \mathrm{CF}) = \mathcal{L}(\mathrm{PVal}, \mathrm{CF})$.

Proof. Given a context-free positive valence grammar $G = (N, T, P, S, \mathbb{Z}^k)$, using the constructions in the proof of Theorem 1, we can find an equivalent context-free positive valence grammar $G = (N', T, P', S', \mathbb{Z}^{k'})$ which can only generate pairs of the form $(\alpha X, \vec{e}_i)$ or $(\alpha, \vec{0})$ where X is a specific symbol in N', $\alpha' \in (N' \setminus \{X\} \cup T)^*$, and $1 \leq i \leq k'$. This means that G' without zero test generates the same language as with zero test. □

Theorem 9 $\mathcal{L}(\mathrm{PVal} - 0, \mathrm{REG}) \subset \mathcal{L}(\mathrm{PVal}, \mathrm{REG})$.

Proof. We show that the language $L = \{a^n b^n : n \geq 1\}$ is not in $\mathcal{L}(\mathrm{PVal} - 0, \mathrm{REG})$. Let $G = (N, T, P, S, \mathbb{Z}^k)$ be a regular positive valence grammar without zero test such that $L \subseteq L(G)$. Then there is a nonterminal A such that $(S, \vec{0}) \stackrel{*}{\Rightarrow} (a^n A, \vec{r}(n)) \stackrel{*}{\Rightarrow} (a^n b^n, \vec{s}(n))$, for infinitely many n. By Dickson's Lemma, there are $n_1 < n_2$ such that $\vec{r}(n_1) \leq \vec{r}(n_2)$. Hence, there is a derivation

$$(S, \vec{0}) \stackrel{*}{\Rightarrow} (a^{n_2} A, \vec{r}(n_2)) \stackrel{*}{\Rightarrow} (a^{n_2} b^{n_1}, \vec{s}(n_2) + \vec{r}(n_2) - \vec{r}(n_1)),$$

and thus $L \subset L(G)$. □

References

[1] Franz-Josef Brandenburg. On context-free grammars with regular control sets. In: Volker Dieckert (editor), *3. Theorietag Automaten und formale Sprachen*, Dagstuhl, 1993.

[2] Jürgen Dassow and Gheorghe Păun. *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, Berlin, 1989.

[3] Sheila A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7:311–324, 1978.

[4] Gheorghe Păun. A new generative device: Valence grammars. *Rev. Roum. Math. Pures Applic.*, 1980.

INHÄRENZ DER MEHRDEUTIGKEITSFUNKTIONEN KONTEXTFREIER GRAMMATIKEN

Klaus Wich
Institut für Informatik, Universität Stuttgart
Breitwiesenstr. 20–22, D-70565 Stuttgart, Germany
e-mail: Klaus.Wich@informatik.uni-stuttgart.de

ABSTRACT

Es wird gezeigt, dass die Menge der Mehrdeutigkeitsfunktionen zykelfreier kontextfreier Grammatiken mit der Menge der inhärenten Mehrdeutigkeitsfunktionen kontextfreier Sprachen übereinstimmt.

1. Einleitung

Eine kontextfreie Grammatik heißt eindeutig, wenn jedes Wort höchstens einen Ableitungsbaum besitzt. Eine kontextfreie Sprache heißt (inhärent) mehrdeutig, wenn sie von keiner eindeutigen kontextfreien Grammatik erzeugt werden kann. Die Existenz von mehrdeutigen kontextfreien Sprachen wurde in [5] nachgewiesen. Man kann mehrdeutige kontextfreie Grammatiken und Sprachen nach der maximalen Anzahl von Ableitungsbäumen auffächern. Eine kontextfreie Grammatik heißt k-deutig, wenn es ein Wort mit k Ableitungsbäumen gibt, aber kein Wort mehr als k Ableitungsbäume besitzt. Eine kontextfreie Sprache heißt k-deutig, wenn sie von einer k-deutigen kontextfreien Grammatik aber von keiner $(k-1)$-deutigen kontextfreien Grammatik erzeugt wird. In [3] wurde die Existenz k-deutiger kontextfreier Sprachen für alle $k \in \mathbb{N}$ nachgewiesen. Darüberhinaus gibt es aber auch kontextfreie Sprachen mit unendlichem Mehrdeutigkeitsgrad [2]. Diese Sprachen können nach dem Wachstumsverhalten ihrer Mehrdeutigkeitsfunktion in Abhängigkeit von der Wortlänge weiter aufgefächert werden. Die Mehrdeutigkeitsfunktion $\hat{d}_G(n)$ einer zykelfreien kontextfreien Grammatik ist eine totale Funktion, die zu jedem n die maximale Anzahl der Ableitungsbäume angibt, die ein Wort bis zur Länge n haben kann. In [6, 7] wurde die Mehrdeutigkeitsfunktion eingeführt und gezeigt, dass es zu jeder zykelfreien kontextfreien Grammatik G ein $k \in \mathbb{N}$ gibt so, dass $\hat{d}_G(n) \in 2^{\Theta(n)} \cup \mathcal{O}(n^k)$ gilt. Der Wert von k kann dabei effizient bestimmt werden [9]. In [4] wurde gezeigt, dass es für jedes $k \in \mathbb{N}$ eine kontextfreie Sprache mit inhärenter Mehrdeutigkeit $\Theta(n^k)$, sowie exponentiell mehrdeutige kontextfreie Sprachen gibt. Es wurden auch bereits einige kontextfreie Sprachen mit sublinearer Mehrdeutigkeit gefunden [8]. Bei allen Mehrdeutigkeitsfunktionen wurde bisher getrennt die Frage betrachtet, ob es eine kontextfreie Grammatik mit der entsprechenden Mehrdeutigkeit gibt oder ob es sogar kontextfreie Sprachen gibt, die genau diese Mehrdeutigkeit erfordern. Hier wird gezeigt, dass sich aus der Angabe einer zykelfreien kontextfreien Grammatik bereits die Existenz einer entsprechend mehrdeutigen kontextfreien Sprache ergibt. Dies vereinfacht zum einen bestehende Beweise, zum anderen ergibt sich bei der Entdeckung neuer Mehrdeutigkeitsfunktionen für zykelfreie kontextfreie Grammatiken unmittelbar deren Inhärenz.

2. Definitionen

Sei Σ ein endliches Alphabet, $w \in \Sigma^*$, $\Gamma \subseteq \Sigma$ und $n \in \mathbb{N}$. Die Länge von w schreiben wir $|w|$. Wir bezeichnen das leere Wort mit ε und definieren $\Sigma^{\leq n} := \{w \in \Sigma^* \mid |w| \leq n\}$. Die *Projektion* auf Γ ist ein Homomorphismus π_Γ vermöge $\pi_\Gamma(a) = a$ für alle $a \in \Gamma$ und $\pi_\Gamma(a) = \varepsilon$ für alle $a \in \Sigma \setminus \Gamma$. Wir erweitern π_Γ in natürlicher Weise auf Sprachen, also $\pi_\Gamma(L) := \{\pi_\Gamma(w) \mid w \in L\}$.

Eine *kontextfreie Grammatik* ist ein Viertupel $G = (N, \Sigma, P, S)$. Dabei ist N eine endliche Menge von *Nichtterminalen*, Σ eine endliche Menge von *Terminalen*, $P \subseteq N \times (N \cup \Sigma)^*$ eine endliche Menge von *Produktionen* und $S \in N$ ist das *Startsymbol*. Wir schreiben $A \to \alpha$ oder $(A \to \alpha)$ für die Produktion (A, α). Wir bezeichnen $A \to \alpha \in P$ als *ε-Produktion* wenn $\alpha = \varepsilon$ ist. Anstatt die von G erzeugte Sprache wie üblich über eine Ableitungsrelation zu definieren, verwenden wir eine Stringrepräsentation von Ableitungsbäumen, die sich aus deren Traversierung in der Reihenfolge einer Tiefensuche ergibt. Ein Vorteil dieser Darstellung ist, dass in ihr die Sprache der Ableitungsbäume einer beliebigen kontextfreien Grammatik eindeutig kontextfreie ist.

Sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik. Die Menge $T_G := N \cup \Sigma \cup P$ bezeichnen wir als *Baumalphabet* von G. Die Menge der *partiellen Ableitungsbäume* von G wird induktiv definiert:

(i) Jedes Symbol aus $N \cup \Sigma$ ist ein partieller Ableitungsbaum.

(ii) Wenn $\tau_1 A \tau_2$ ein partieller Ableitungsbaum ist, wobei $\tau_1, \tau_2 \in T_G^*$, $A \in N$ und $p = (A \to \gamma) \in P$, dann ist auch $\tau_1 p \gamma \tau_2$ ein partieller Ableitungsbaum.

Die Menge der partiellen Ableitungsbäume von G wird mit $\triangle_p(G)$ bezeichnet. Sei $X\rho$ für ein $X \in T_G$ und ein $\rho \in T_G^*$ ein partieller Ableitungsbaum. Die *Wurzel* von $X\rho$ ist definiert durch $\uparrow X\rho := A$ falls $X = (A \to \alpha) \in P$ für ein $\alpha \in (N \cup \Sigma)^*$ und $\uparrow X\rho := X$ falls $X \in N \cup \Sigma$. Man beachte, dass in letzterem Fall notwendigerweise $\rho = \varepsilon$ gilt. Man sieht leicht, dass sich zwei partielle Ableitungsbäume $\tau_1 A \tau_2$ und ρ mit $\tau_1, \tau_2 \in T_G^*$ und $A = \uparrow\rho$ zu einem partiellen Ableitungsbaum $\tau_1 \rho \tau_2$ zusammensetzen lassen. Die Front eines partiellen Ableitungsbaums ρ ist definiert durch $\downarrow\rho := \pi_{N \cup \Sigma}(\rho)$. Man erhält also die Front, indem man die Produktionen aus ρ löscht. (Die Pfeile für die Wurzel und die Front deuten jeweils in die Richtung, in der man Wurzel bzw. Front in einer graphischen Darstellung eines Ableitungsbaums findet.) Ein partieller Ableitungsbaum ρ heißt *Ableitungsbaum*, wenn $\uparrow\rho = S$ und $\downarrow\rho \in \Sigma^*$, d.h. das Startsymbol ist die Wurzel von ρ und die Front von ρ enthält nur Terminale. Die Menge der Ableitungsbäume von G schreiben wir $\triangle(G)$.

Die von G *erzeugte Sprache* ist $L(G) := \{w \in \Sigma^* \mid \exists \rho \in \triangle(G) : w = \downarrow\rho\}$. Eine Sprache heißt kontextfrei, wenn sie von einer kontextfreien Grammatik erzeugt wird.

Die kontextfreie Grammatik G ist zykelfrei, wenn alle partiellen Ableitungsbäume, bei denen die Wurzel gleich der Front ist, nur einen Knoten besitzen, d.h. $\forall \rho \in \triangle_p(G) : \uparrow\rho = \downarrow\rho \Rightarrow |\rho| = 1$. Die kontextfreie Grammatik G ist in Greibachnormalform, wenn $P \subseteq N \times \Sigma N^*$. Man beachte, dass dann $\triangle(G) \subseteq (P\Sigma)^*$.

Sei $w \in \Sigma^*$. Die *Mehrdeutigkeit von w* ist $d_G(w) := |\{\rho \in \triangle(G) \mid w = \downarrow\rho\}|$. Die zugehörige Abbildung $d_G : \Sigma^* \to \mathbb{N}$ heißt *charakteristische Mehrdeutigkeitspotenzreihe* von G. Die *Mehrdeutigkeitsfunktion von G* ist definiert durch $\hat{d}_G(n) := \max\{d_G(w) \mid w \in \Sigma^{\leq n}\}$ für alle $n \in \mathbb{N}$.

Sei $f : \mathbb{N} \to \mathbb{N}$ eine totale Funktion. Eine kontextfreie Sprache L heißt inhärent f-deutig wenn

(i) es eine kontextfreie Grammatik G mit $L = L(G)$ und $f = \hat{d}_G$ gibt und

(ii) für jede kontextfreie Grammatik G' mit $L = L(G')$ ein $k \in \mathbb{N}$ existiert so, dass $\hat{d}_{G'}(k \cdot n) \geq \hat{d}_G(n)$ für alle $n \in \mathbb{N} \setminus \{0\}$.

Eine Funktion f ist eine *inhärente Mehrdeutigkeitsfunktion*, wenn es eine inhärent f-deutige kontextfreie Sprache gibt.

3. Von Mehrdeutigkeitsfunktionen zur Inhärenz

Aus dem nachfolgenden Theorem ergibt sich unmittelbar, dass die Menge der Mehrdeutigkeitsfunktionen zykelfreier kontextfreier Grammatiken mit der Menge der inhärenten Mehrdeutigkeitsfunktionen übereinstimmt.

Theorem 1 *Für eine beliebige zykelfreie kontextfreien Grammatik G gibt es eine kontextfreie Sprache L, die inhärent $\hat{d}_G$-deutig ist. L ist konstruierbar.*

Der Beweis von Theorem 1 ergibt sich leicht aus dem nachfolgenden Lemma 3. Dieses Lemma ist einerseits schwächer als Theorem 1, weil es die Greibachnormalform erfordert, andererseits stärker, weil es eine Aussage über die, im Vergleich zur Mehrdeutigkeitsfunktion feiner auflösende, Mehrdeutigkeitspotenzreihe macht.

Definition 2 *Ein Homomorphismus $h : \Sigma^* \to \Gamma^*$ heißt* konstant streckend*, wenn für alle Symbole $a, b \in \Sigma$ die Beziehung $|h(a)| = |h(b)| > 0$ gilt.*

Lemma 3 *Sei G eine kontextfreie Grammatik in Greibachnormalform mit $L(G) \subseteq \Sigma^*$ und $\{0,1\} \cap \Sigma = \emptyset$, dann gibt es eine kontextfreie Sprache $L \subseteq (\Sigma \cup \{0,1\})^*$ mit den folgenden Eigenschaften:*

(i) $L(G) \subseteq L$.

(ii) $L(G) = \pi_\Sigma(L)$

(iii) $\exists G'$ *kontextfrei :* $L = L(G')$ *und* $\forall w \in (\Sigma \cup \{0,1\})^* : d_G(\pi_\Sigma(w)) = d_{G'}(\pi_\Sigma(w)) \geq d_{G'}(w)$.

(iv) $\forall G''$ *kontextfrei mit* $L = L(G'') : \exists h : \Sigma^* \to (\Sigma \cup \Gamma)^*$ *konstant streckend:* $\forall w \in \Sigma^* :$ $d_G(w) \leq d_{G''}(h(w))$.

(v) L *kann aus G konstruiert werden.*

Beweisskizze. Sei $G = (N, \Sigma, P, S)$. Wir konstruieren die kontextfreie Grammatik $G_\triangle = (N, \Sigma \cup P, P_\triangle, S)$ mit $P_\triangle := \{A \to p\alpha \mid p = (A \to \alpha) \in P\}$. Da man allein anhand des ersten Zeichens der rechten Regelseiten eindeutig die angewendete Regel identifizieren kann ist $G_\triangle$ eindeutig. Man sieht leicht, dass $L(G_\triangle) = \triangle(G)$. Also wird die Menge der Ableitungsbäume von G durch die eindeutige kontextfreie Grammatik $G_\triangle$ generiert.

Sei $k := |P|$ und $j \in \mathbb{N}$ mit $1 \leq j \leq k$. Wir definieren $L_j := \{0^{i_0}10^{i_1}1 \ldots 0^{i_k}1 \mid i_0, \ldots, i_k \in \mathbb{N} \wedge i_0 = i_j\} \cup \{\varepsilon\}$. Diese Sprachen haben $k+1$ Blöcke von 0-en, die von 1-en abgeschlossen werden. Wir beginnen die Zählung bei 0. In der Sprache L_j muss der 0-te Block mit dem j-ten Block übereinstimmen. Man beachte, dass $(0^q1)^{k+1}$ für alle $q \in \mathbb{N}$ im Schnitt aller L_j liegt. Die Vereinigung $\cup_{j \in M} L_j$ ist für beliebige $M \subseteq \{1, \ldots, k\}$ inhärent $|M|$-deutig.

Wir substituieren jedes Element aus P durch genau eine Sprache L_j aus unserem System eindeutiger Sprachen. D.h. wir wählen eine Bijektion $\delta : P \to \{1, \ldots, k\}$ und definieren damit eine Substitution $\sigma : (\Sigma \cup P)^* \to 2^{(\Sigma \cup \{0,1\})^*}$ vermöge $\sigma(X) = \{X\}$ für $X \in \Sigma$ und $\sigma(X) = L_{\delta(X)}$ für $X \in P$. Nun definieren wir die Sprache $L := \sigma(\triangle(G))$ und zeigen im folgenden, dass L die geforderten Eigenschaften besitzt.

Da $L(G) = \{\downarrow\rho \mid \rho \in \triangle(G)\} = \pi_{\Sigma \cup N}(\triangle(G)) = \pi_{\Sigma \cup N}(L(G_\triangle)) = \pi_\Sigma(L(G_\triangle))$, gilt entsteht $L(G)$ durch Löschung der Elemente aus P in den von $G_\triangle$ erzeugten Wörtern. Da $\varepsilon \in L_j$ für alle $j \in \{1, \ldots, k\}$ ergibt sich daraus, dass $L \subseteq L(G)$ ist (i). Da bei der Substitution nur Wörter über $\{0,1\}^*$ für Elemente aus P eingesetzt werden und $\Sigma \cap \{0,1\} = \emptyset$ ist folgt (ii). Für alle $p \in P$ wählen wir eindeutig kontextfreie Grammatiken $G_p(N_p, \{0,1\}, P_p, p)$ so, dass $L(G_p) = L_{\delta(p)}$, $\forall p, p' \in P : N_p \cap N_{p'} = \emptyset$ und $N_p \cap N = \emptyset$. Wir konstruieren aus diesen Grammatiken und $G_\triangle$ eine kontextfreie Grammatik G' für L in der kanonischen Weise, also $G' = (N \cup P \cup (\cup_{p \in P} N_p), \Sigma \cup \{0,1\}, P_\triangle \cup (\cup_{p \in P} P_p), S)$, womit (v) folgt. Man kann zeigen, dass G' die in (iii) geforderten

Bedingungen erfüllt. Sei G'' eine beliebige kontextfreie Grammatik mit $L = L(G'')$ und sei r die zu G'' gehörige Konstante von Ogdens Lemma für kontextfreie Grammatiken [1, Lemma 2.5]. Sei $s := r + r!$. Wir definieren die Homomorphismen $h_u, h : (\Sigma \cup P)^* \to (\Sigma \cup \{0,1\})^*$ vermöge $h(X) = h_u(X) = X$ für $X \in \Sigma$ und $h_u(X) = 0^r 1(0^s 1)^{\delta(X)-1} 0^r 1(0^s 1)^{k-\delta(X)}$ bzw. $h(X) = (0^s 1)^{k+1}$ für $X \in P$. Nach Definition von L gilt $h_u(\triangle(G)) \subseteq L$ bzw. $h(\triangle(G)) \subseteq L$. Sei $w \in \Sigma^*$ ein beliebiges Wort und $R_w := h_u(\triangle(G)) \cap \pi_\Sigma^{-1}(w)$. Man beachte das es zu jedem Wort $v \in R_w$ einen eindeutig bestimmten Ableitungsbaum $\rho \in \triangle(G)$ gibt so, dass $v \in \sigma(\rho)$ ist. Daraus ergibt sich $|R_w| = d_G(w)$. Da G in Greibachnormalform vorliegt folgt, dass $R_w \subseteq (\Sigma(0^*1)^{k+1})^*$ und $h(w) \in (\Sigma(0^*1)^{k+1})^*$. Man kann dies benutzen, um unter Anwendung von Ogdens Lemma zu zeigen, dass alle Wörter aus R_w zu dem Wort $h(w)$ aufgepumpt werden können. Die Annahme das die zugehörigen Bäume nicht paarweise verschieden sind führt dann zu einem Widerspruch. Da h offenbar konstant streckend ist erhält man schließlich auch (iv). □

Es folgt der Beweis von Theorem 1.

Beweisskizze. Da G zykelfrei ist, ist $\hat{d}_G$ wohldefiniert. Man kann zeigen, dass es eine äquivalente kontextfreie Grammatik G_0 in Greibachnormalform mit $d_{G_0} = d_G$ gibt. Zu dieser können wir nach Lemma 3 eine Sprache L mit den dort genannten Eigenschaften konstruieren. Sei G' eine kontextfreie Grammatik für L, welche die Bedingung (iii) in Lemma 3 erfüllt. Der Mehrdeutigkeitsgrad eines Wortes ist für den Verlauf der Mehrdeutigkeitsfunktion nur dann ausschlaggebend, wenn es kein kürzeres Wort mit mindestens gleicher Mehrdeutigkeit gibt. Somit folgt aus (iii), dass Wörter, die 0-en und 1-en enthalten, für den Verlauf der Mehrdeutigkeitsfunktion unerheblich sind. Die übrigen Wörter bilden nach (i) und (ii) aber die Sprache $L(G)$. Damit ergibt sich wiederum unter Anwendung von (iii), dass $\hat{d}_{G'} = \hat{d}_G$. Aus Lemma 3(iv) ergibt sich nun unmittelbar, dass L inhärent $\hat{d}_G$-deutig ist. □

4. Zusammenfassung

Es wurde gezeigt, dass man zu jeder zykelfreien kontextfreien Grammatik G eine Sprache L konstruieren kann so, dass L inhärent $\hat{d}_G$-deutig ist. Damit genügt die Angabe einer zykelfreien kontextfreien Grammatik, um die Existenz kontextfreier Sprachen mit der entsprechenden Mehrdeutigkeit nachzuweisen. Dies vereinfacht und vereinheitlicht solche Existenzbeweise.

Mit Lemma 3 überträgt sich sogar die Struktur der charakteristischen Mehrdeutigkeitspotenzreihen inhärent auf L. Durch eine geringfügige Modifikationen von Lemma 3 lassen sich deshalb die Mehrdeutigkeitsauswirkungen von einigen Grammatikkonstruktionen inhärent auf die kontextfreien Sprachen übertragen. Seien G_1, G_2 und G_3 zykelfreie kontextfreie Grammatiken so, dass G_3 durch die kanonischen Konstruktion für die Konkatenation aus G_1 und G_2 hervorgeht, dann gibt es kontextfreie Sprachen L_1, L_2 und L_3, so dass $L_3 = L_1 L_2$ und für jedes $i \in \{1,2,3\}$ die Sprache L_i inhärent $\hat{d}_{G_i}$-deutig ist. Analoges gilt für die Vereinigung.

Es stellen sich die folgenden Fragen:

- Welche Operationen, außer den genannten übertragen noch die Mehrdeutigkeitsauswirkungen von zykelfreien kontextfreien Grammatiken auf kontextfreie Sprachen?
- Wie kann man die Menge der Mehrdeutigkeitsfunktionen charakterisieren?

Literatur

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

[2] J. Crestin. Un langage non ambigu dont le carré est d'ambiguité non bornée. In M. Nivat, editor, *Automata, Languages and Programming*, pages 377–390. Amsterdam, North-Holland, 1973.

[3] H. Maurer. The existence of context-free languages which are inherently ambiguous of any degree. Research series, Department of Mathematics, University of Calgary, 1968.

[4] M. Naji. Grad der Mehrdeutigkeit kontextfreier Grammatiken und Sprachen, 1998. Diplomarbeit, FB Informatik, Johann–Wolfgang–Goethe–Universität.

[5] R. J. Parikh. Language–generating devices. In *Quarterly Progress Report*, volume 60, pages 199–212. Research Laboratory of Electronics, M.I.T, 1961.

[6] K. Wich. Kriterien für die Mehrdeutigkeit kontextfreier Grammatiken, 1997. Diplomarbeit, FB Informatik, Johann–Wolfgang–Goethe–Universität.

[7] K. Wich. Exponential ambiguity of context-free grammars. In G. Rozenberg and W. Thomas, editors, *Proceedings of the 4th International Conference on Developments in Language Theory, July 1999*, pages 125–138. World Scientific, Singapore, 2000.

[8] K. Wich. Sublinear ambiguity. In M. Nielsen and B. Rovan, editors, *Proceedings of the MFCS 2000*, number 1893 in Lecture Notes in Computer Science, pages 690–698, Berlin-Heidelberg-New York, 2000. Springer.

[9] K. Wich. Characterization of context-free languages with polynomially bounded ambiguity. In J. Sgall, A. Pultr, and P. Kolman, editors, *Proceedings of the MFCS 2001*, number 2136 in Lecture Notes in Computer Science, pages 703–714, Berlin-Heidelberg-New York, 2001. Springer.

AUTORENVERZEICHNIS

TEILNEHMERVERZEICHNIS

Suna Aydin
Institut für Informatik, Universität Potsdam
Postfach 90 03 27, 14439 Potsdam
aydin@cs.uni-potsdam.de

Oliver Boldt
Institut für Informatik, Universität Potsdam
Postfach 90 03 27, 14439 Potsdam
boldt@cs.uni-potsdam.de

Benedikt Bollig
Lehrstuhl für Informatik II, RWTH Aachen
Ahornstraße 55, 52074 Aachen
bollig@informatik.rwth-aachen.de

Björn Borchardt
Institut für Theoretische Informatik, Technische Universität Dresden
Cottaer Straße 8, 01159 Dresden
borchard@tcs.inf.tu-dresden.de

Henning Bordihn
Institut für Informatik, Universität Potsdam
Postfach 90 03 27, 14439 Potsdam
henning@cs.uni-potsdam.de

Véronique Bruyère
Faculty of Sciences, Université de Mons-Hainaut
Le Pentagone, 6 Avenue du Champ de Mars, B-7000 Mons, Belgium
Veronique.Bruyere@umh.ac.be

Jürgen Dassow
Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg
Postfach 41 20, 39016 Magdeburg
dassow@iws.cs.uni-magdeburg.de

Henning Fernau
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
Sand 13, 72076 Tübingen
fernau@informatik.uni-tuebingen.de

Rudolf Freund
Institut für Computersprachen, Technische Universität Wien
Favoritenstraße 9, A-1040 Wien, Austria
rudi@logic.at

Fritz von Haeseler
Elektrotechniek, Katholieke Universiteit Leuven
Kastelpark Arenberg 10, B-3001 Leuven, Belgium
fvanhaes@esat.kuleuven.ac.be

Maia Hoeberechts
Department of Computer Science, The University of Western Ontario
Middlesex College, London, Ontario, Canada N6A 5B7
hoebere@csd.uwo.ca

Markus Holzer
Institut für Informatik, Technische Universität München
Arcisstraße 21, 80290 München
holzer@in.tum.de

Masami Ito
Faculty of Science, Kyoto Sangyo University
Kyoto 603-8555, Japan
ito@ksuvx0.kyoto-su.ac.jp

Claus Jürgensen
Institut für Theoretische Informatik, Technische Universität Dresden
Cottaer Straße 8, 01159 Dresden
claus.juergensen@inf.tu-dresden.de

Helmut Jürgensen
Institut für Informtik, Universität Potsdam
Postfach 90 03 27, 14439 Potsdam
helmut@cs.uni-potsdam.de
und
Department of Computer Science, The University of Western Ontario
Middlesex College, London, Ontario, Canada N6A 5B7
helmut@uwo.ca

Juhani Karhumäki
Department of Mathematics, University of Turku
FIN-20014 Turku, Finland
karhumak@cs.utu.fi

Andreas Klein
Fachbereich Mathematik/Informatik, Universität Gesamthochschule Kassel
Heinrich Plett Straße 40 (AVZ), 34132 Kassel
klein@mathematik.uni-kassel.de

Roman König
Lehrstuhl für Informatik II, Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstraße 3, 91058 Erlangen
koenig@informatik.uni-erlangen.de

Dietrich Kuske
Department of Mathematics and Computer Science, University of Leicester
University Road, Leicester LE1 7RH, United Kingdom
d.kuske@mcs.le.ac.uk

Martin Kutrib
Institut für Informatik, Universität Gießen
Arndtstraße 2, 35392 Gießen
kutrib@informatik.uni-giessen.de

Jan-Thomas Löwe
Institut für Informatik, Universität Gießen
Arndtstraße 2, 35392 Gießen
jan-thomas.loewe@informatik.uni-giessen.de

Charlotte Miller
Department of Computer Science, The University of Western Ontario
Middlesex College, London, Ontario, Canada N6A 5B7
cmiller@csd.uwo.ca

Gundula Niemann
Fachbereich Mathematik/Informatik, Universität Gesamthochschule Kassel
Heinrich Plett Straße 40 (AVZ), 34132 Kassel
niemann@theory.informatik.uni-kassel.de

Friedrich Otto
Fachbereich Mathematik/Informatik, Universität GH Kassel
Heinrich-Plett-Straße 40, 34109 Kassel
otto@theory.informatik.uni-kassel.de

Holger Petersen
Institut für Informatik, Universität Stuttgart
Breitwiesenstraße 20–22, 70565 Stuttgart
petersen@informatik.uni-stuttgart.de

Bernd Reichel
Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg
Postfach 41 20, 39016 Magdeburg
reichel@iws.cs.uni-magdeburg.de

Klaus Reinhardt
Wilhelm-Schickard-Institut für Informatik, Universtität Tübingen
Sand 13, 72076 Tübingen
reinhard@informatik.uni-tuebingen.de

Ludwig Staiger
Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg
von-Seckendorff-Platz 1, 06099 Halle (Saale)
staiger@informatik.uni-halle.de

Heiko Stamer
Institut für Informatik, Universität Leipzig
Augustusplatz 10–11, 04109 Leipzig
stamer@informatik.uni-leipzig.de

Ralf Stiebe
Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg
Postfach 41 20, 39016 Magdeburg
stiebe@iws.cs.uni-magdeburg.de

Heiko Vogler
Institut für Theoretische Informatik, Technische Universität Dresden
Mommsenstraße 13, 01062 Dresden
vogler@inf.tu-dresden.de

Klaus Wich
Institut für Informatik, Universität Stuttgart
Breitwiesenstraße 20–22, 70565 Stuttgart
wich@informatik.uni-frankfurt.de