T U M

I N S T I T U T F Ü R I N F O R M A T I K

Workshop Petrinetze
und
13. Theorietag Automaten und Formale Sprachen

Markus Holzer (Herausgeber)

TUM-I0322
Dezember 03

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-12-I0322-0/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

Vorwort

Die bisherigen Theorietage „Automaten und Formale Sprachen“ fanden in Magdeburg (September–Oktober 1991), Kiel (Oktober 1992), Schloß Dagstuhl (Oktober 1993), Herrsching bei München (September 1994), Schloß Rauischholzhausen (September 1995), Cunnersdorf bei Königsstein (September 1996), Barnstorf bei Bremen (September–Oktober 1997), Riveres bei Trier (September 1998), Schauenburg-Elmshagen bei Kassel (September 1999), Wien (September 2000), Wendgräben bei Magdeburg (Oktober 2001) und Wittenberg (September 2002). Vom 29. September bis 2. Oktober 2003 wurde die Tradition in der Bildungstätte des Bayerischen Bauernverbandes in Herrsching bei München fortgesetzt. Seit dem 6. Theorietag in Cunnersdorf wird der Theorietag von einem Workshop mit eingeladenen Vorträgen begleitet. Die Themen im Laufe der Jahre lauteten: „Perspektiven der Automaten und Formalen Sprachen“, „Prozesse und Formale Sprachen“, „Automaten, Formale Sprachen und Ersetzungssysteme“, „Molecular Computing, Quantum Computing“, „Coding Theory and Formal Languages“ und „Berechenbarkeit und Komplexität in der Analysis“. Das diesjährige Thema des Workshops lautet „Petrinetze“. Es nahmen 29 TeilnehmerInnen aus Australien, Deutschland, Frankreich, Kanada, Österreich, Schweden und Tschechien teil. Das wissenschaftliche Programm bestand aus angemeldeten und eingeladenen Beiträgen der TeilnehmerInnen. Die Kurzzusammenfassungen der Beiträge sind in diesem Bericht abgedruckt.

Ich danke allen TeilnehmerInnen für ihre interessanten Beiträge und die Bereitschaft zur wissenschaftlichen Diskussion. Ich danke auch der Technischen Universität München, insbesondere Herrn Brauer, für ihre Unterstützung und allen MitarbeiterInnen der Bildungsstätte des Bayerischen Bauernverbandes Herrsching für ihren Einsatz vor und während der Tagung. Ohne ihre organisatorische Hilfe wäre das Treffen nicht möglich gewesen. Auch ein recht herzliches Dankeschön an die HUK-Coburg AG für die Sachspenden zum Theorietag. Dem nächsten Theorietag in Caputh bei Potsdam wünsche ich viel Erfolg.

Ein besonderer Dank gilt Frau Erika Leber für die hilfreiche Unterstützung bei der Organisation des Theorietags und Bernd Reichel für die Email-Unterstützung und das Korrekturlesen der HTML-Seiten.

Garching, im September 2003 Markus Holzer

Inhaltsverzeichnis

Vortragsprogramm

Dienstag, der 30. September 2003

9:15–9:20 Begrüßung der TeilnehmerInnen zum Workshop „Petrinetze“

9:20–10:25 Eike Best, *Sprach- und Pomset-Äquivalenz bei beschrifteten Petrinetzen*

10:25–10:55 Kaffeepause

10:55–12:00 Ekkart Kindler, *Über den zweckmäßigen Einsatz von Petrinetzen*

12:00–14:00 Mittagspause

14:00–15:05 Karsten Schmidt, *Supporting explicit state space verification by transition invariants*

15:05–15:40 Kaffeepause

15:40–16:45 Javier Esparza, *Some applications of Petri nets to the verification of parametrized systems*

Mittwoch, der 1. Oktober 2003

8:55–9:00 Begrüßung der TeilnehmerInnen zum Theorietag

9:00–9:25 Henning Bordihn, *The degree of parallelism*

9:25–9:50 Maia Hoeberechts, *On the degree of nondeterminism of tree adjoining languages and head grammar languages*

9:50–10:15 Andreas Malcher, *Minimizing finite automata is computationally hard*

10:15–10:40 Roman König, *Eine kombinatorische Eigenschaft gewisser* $0,1$*-Matrizen*

10:40–11:10 Kaffeepause

11:10–11:35 Daniel Kirsten, *Distance automata and the star height problem*

11:35–12:00 Martin Lange, *Verification of non-regular properties*

12:00–12:25 Martin Leucker, *A hierarchy of MSC languages*

12:25–14:00 Mittagspause

14:00–14:25 Martin Kutrib, *On the descriptional power of heads, counters, and pebbles*

14:25–14:50 Jens Glöckler, *Über mehrdimensionale Rebound-Automaten*

14:50–15:15 Markus Holzer, *A common algebraic characterization of probabilistic and quantum computations*

15:15–15:45 Kaffeepause

15:45–16:10 Martin Plátek, *Restarting automata: motivations and applications*

16:10–16:35 František Mráz, *Hierarchies of weakly monotone restarting automata*

16:35–17:00 Klaus Reinhardt, *Some more regular languages that are Church Rosser congruential*

17:00–18:00 Vollversammlung der GI Fachgruppe 0.1.5

Donnerstag, der 2. Oktober 2003

9:00–9:25 Henning Fernau, *Formal language aspects of identifiable language classes*

9:25–9:50 Andreas Klein, *On the NP-completeness of the* NURIKABE *puzzle and variants thereof*

9:50–10:15 Ludwig Staiger, *Relativisations of disjunctiveness*

10:15–10:45 Kaffeepause

10:45–11:10 Marion Oswald, *P-Automaten*

11:10–11:35 Rudolf Freund, *ω-P-Automaten*

11:35–12:00 Ralf Stiebe, *On the size of hybrid networks of evolutionary processors*

12:00– Mittagspause und Abreise

SPRACH- UND POMSET-ÄQUIVALENZ BEI BESCHRIFTETEN PETRINETZEN

Eike Best
Universität Oldenburg, Fakultät II
Department für Informatik, 26111 Oldenburg, Germany
e-mail: Eike.Best@Informatik.Uni-Oldenburg.DE

KURZFASSUNG

Wir analysieren, welche Transformationen das Sprach- bzw. das Pomsetverhalten eines Petrinetzes invariant lassen. Es werden drei neuere Resultate und eine Vermutung vorgestellt:

- Jedes k-beschränkte beschriftete Petrinetz lässt sich in ein pomset-äquivalentes 1-beschränktes Petrinetz transformieren (Best/Wimmel 2000).
- Jeder k-beschränkte Synchronisationsgraph lässt sich in einen sprach-äquivalenten 1-beschränkten Synchronisationsgraphen transformieren; es gibt jedoch einen 2-beschränkten Synchronisationsgraphen, der kein pomset-äquivalentes 1-beschränktes Pendant (Synchronisationsgraph) besitzt (Darondeau/Wimmel 2002).
- Jeder 1-beschränkte Synchronisationsgraph mit stillen Transitionen τ lässt sich in einen pomset-äquivalenten 1-beschrnkten Synchronisationsgraphen ohne τ-Transitionen transformieren (Wimmel/Best 2003).
- Vermutung: die Beschränkung auf Synchronisationsgraphen kann im letzten Satz weggelassen werden.

SOME APPLICATIONS OF PETRI NETS TO THE VERIFICATION OF PARAMETRIZED SYSTEMS

Javier Esparza
Institut für Formale Methoden der Informatik
Universität Stuttgart, Universitätstr. 38
70569 Stuttgart, Germany
e-mail: esparza@informatik.uni-stuttgart.de

ABSTRACT

Many systems are designed to work for different values of a parameter. For instance, a carry look-ahead adder should work correctly for binary numbers of arbitrary length (it is then instantiated as a 32-bit, 64-bit adder etc.); multicast communication protocols should work for arbitrary numbers or receivers and routers; cryptographic protocols should be secure independently of the number of principals that use the protocol; a cahe coherence protocol should work for arbitrarily many caches. In all these cases, the verification task consists of proving that all members of an infinite family of systems, obtained by instantiating parameters with different values, behave correctly.

Petri net analysis techniques (more precisely, techniques for place/transition nets) have proved successful in attacking this problem when the parameter corresponds to the number of processes or active components of the system, and when these components have identical (or at least very similar) structure. The main idea is to interpret the places of the net as the possible local states of one component, and interpret the number of tokens on the place as the number of components that are currently in that state. This approach was first proposed by German and Sistla in [6]. They used coverability trees á la Karp-Miller to derive analysis algorithms. Another important milestone was the discovery of a surprisingly simple backwards reachability technique by Abdulla, Cerans, Jonsson, and Tsay [1].

One of the limitations of place/transition nets in the context of parametrized verification is the fact that they can only model communication by rendezvous: two processes (or, more generally, an arbitrary *but fixed* number of processes)

synchronize and change their states, while all other processes remain idle. There is no possibility to model broadcast communication, in which a process sends a message to all others, independently of how many they are (their number can change dynamically when process creation is allowed). Extensions of the analysis techniques to these cases have been studied by a number of people, see for instance [2, 3, 4, 5].

In the talk I will present the main analysis techniques together with some negative results that outline the limitations of the approach.

References

[1] Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, Yih-Kuen Tsay: Algorithmic Analysis of Programs with Well Quasi-ordered Domains. Information and Computation 160(1-2): 109-127 (2000)

[2] Giorgio Delzanno, Jean-Franois Raskin, Laurent Van Begin: Towards the Automated Verification of Multithreaded Java Programs. TACAS 2002: 173-187

[3] Giorgio Delzanno, Javier Esparza, Andreas Podelski: Constraint-Based Analysis of Broadcast Protocols. CSL 1999: 50-66

[4] Catherine Dufourd, Alain Finkel, Ph. Schnoebelen: Reset Nets Between Decidability and Undecidability. ICALP 1998: 103-115

[5] Javier Esparza, Alain Finkel, Richard Mayr: On the Verification of Broadcast Protocols. LICS 1999: 352-359

[6] Steven M. German, A. Prasad Sistla: Reasoning about Systems with Many Processes. JACM 39(3): 675-735 (1992)

ÜBER DEN ZWECKMÄSSIGEN EINSATZ VON PETRINETZEN

Ekkart Kindler
Institut für Informatik, Universität Paderborn
Warburger Straße 100, 33098 Paderborn, Germany
e-mail: kindler@upb.de

KURZFASSUNG

Petrinetze sind ein Formalismus – genauer eine Familie verwandter Formalismen – zur Modellierung des Verhaltens reaktiver und verteilter Systeme. Die Petrinetztheorie bietet viele verschiedene Techniken, um Aussagen über verschiedene Aspekte des modellierten Systems und seines Verhaltens zu treffen.

Da Petrinetze ein sehr allgemeiner und elementarer Formalismus sind, lassen sich Petrinetze in vielen verschiedenen Anwendungsgebieten, für viele verschiedene Anwendungszwecke und auf verschiedenen Abstraktionsebenen einsetzen. Je nach Gebiet und Zweck werden Petrinetze auf sehr unterschiedliche Weise zur Modellierung, Analyse und Validierung eingesetzt. Dementsprechend gibt es viele verschiedene Modellierungsparagdigmen für Petrinetze.

Im Vortrag werden anhand einiger Beispiele verschiedene Modellierungsparadigmen diskutiert. Außerdem wird gezeigt, wie verschiedene Aspekte des Systemverhaltens durch geeignete Ergänzungen des Petrinetzmodells unabhängig voneinander untersucht werden können. Als Beispiele betrachten wir die Modellierung von Algorithmen bzw. von „algorithmischen Ideen“, die Modellierung von Geschäftsprozessen und die Modellierung von Materialflußsystemen.

SUPPORTING EXPLICIT STATE SPACE VERIFICATION BY TRANSITION INVARIANTS

Karsten Schmidt
Institut für Informatik, Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
e-mail: kschmidt@informatik.hu-berlin.de

ABSTRACT

Traditionally, Petri net invariants are applied as a tool to *replace* state space verification. They are characterized as solutions of a linear system of equations derived from the incidence matrix of the given Petri net and can thus be easily computed.

In this talk, we present three examples that show how one kind of invariants—transition invariants–can be used to *support* state space verification. In all examples, we exploit the well known fact that transition invariants are closely related to cycles in the state space: if a transition sequence forms a cycle then the vector counting the occurrences of transitions in this sequence forms a transition invariant. Algebraically, a transition invariant is a linear combination of transition vectors that generate the $\underline{0}$ vector. From studying the system of equations defining transition invariants, we can partition the set of transitions into a set U and a set $\overline{U}$ where the set U contains a set of linear independent transitions of maximum size (that size is given by the rank of the incidence matrix) and the set $\overline{U}$ of all remaining transitions. Consequently,

- A sequence of transitions from U does not form cycles, or, rephrased:
- Every cycle in the state space contains an element of $\overline{U}$.

Exactly this information is applied in the following three scenarios.

First scenario (already presented at TACAS 2003). The purpose of storing states permanently in explicit state space verification is termination (through assuring that one and the same state is not explored twice). It has been observed that this condition can be weakened to: assure that on every cycle in the state space there is a state that is not explored twice (so it is sufficient to store only

those states permanently). The remaining states are re-explored upon every visit. In our approach, we store those states permanently where a transition from $\overline{U}$ is fired and cover therefore all cycles.. We show that, at least in connection with partial order reduction and a simple heuristic add-on, this method turns out to be quite useful.

Second scenario. One of the most important reduction techniques in explicit state space verification is partial order reduction. It works by exploring, in every state, only a subset of the enabled transitions. This subset satisfies a number of criteria in order to assure that the reduced state space holds the examined property if and only if the full state space does. Among the criteria are some that can be computed directly from the current state and the system structure. Yet there are other criteria requiring that some transitions must occur at least once on every cycle in the reduced system. Traditionally, these criteria are implemented using depth first search (then, every cycle contains a state that has, while visited, a successor on the depth first search stack). This implementation depends on a strictly sequential exploration of the state space. In distributed verification, it is not as easy to detect cycles. We propose to use the the same structural criterion as in the first scenario: If something needs to be done at least once on every cycle, do it whenever a transition of $\overline{U}$ is fired. This approximation leads to acceptable results.

Third scenario. The sweep-line method, recently introduced, is about removing previously computed states as soon as they are no longer possible successors of unexplored states. A progress measure (an assignment of a number to each state— in its basic shape monotonous w.r.t. the successor relation) gives the necessary information: remove states that have smaller progress values than the unexplored states. The method has been generalized to non-monotonous progress measures. Whenever a state has a smaller progress value than its predecessor, it is stored permanently, and is used as the starting point of another state space exploration. The crucial tool in this method is a suitable progress measure (one where only few transitions decrease progress values). In the original papers, construction of a progress measure is left to the user. We propose the following automated construction of a progress measure. Let the initial state have progress value 0. Let every transition in U cause an increase of the measure by 1. The increase or decrease of the remaining transitions can be determined by the fact that all of them can be expressed as linear combinations of the transitions in U. This way, we obtain a sound (not necessarily monotonous) progress measure that—when applied in combination with partial order reduction—leads to significant reduction even for reactive systems with a lot of local cycles (where one would not immediately expect a notion of "progress").

Transition invariants turn out to be a nice tool for very different applications in explicit state space verification. We assume that there are more application areas pending.

ON THE DEGREE OF NONDETERMINISM OF TREE ADJOINING LANGUAGES AND HEAD GRAMMAR LANGUAGES

Suna Bensch
Institut für Informatik, Universität Potsdam
August-Bebel-Str. 89, 14482 Potsdam, Germany
e-mail: aydin@cs.uni-potsdam.de

and

Maia Hoeberechts
Department of Computer Science
University of Western Ontario, N6A 5B7
London, Ontario, Canada
e-mail: hoebere@csd.uwo.ca

ABSTRACT

The degree of nondeterminism is a measure of syntactic complexity which was investigated for parallel rewriting systems as well as for sequential rewriting systems. In this paper, we consider the degree of nondeterminsm for tree adjoining grammars and their languages and head grammars and their languages. We show that a degree of nondeterminism of 2 suffices for both formalisms in order to generate all languages in their respective language families. Furthermore, we show that deterministic tree adjoining grammars (those with degree of nondeterminism equal to 1), can generate non-context-free languages, in contrast to deterministic head grammars which can only generate languages containing a single word.

Keywords: Syntactic complexity, degree of nondeterminism, tree adjoining grammars, head grammars.

1. Introduction

The degree of nondeterminism for tabled Lindenmayer systems and languages has been studied in [11, 10] as a measure of syntactic complexity. The degree of nondeterminism has also been considered for sequential rewriting systems [1, 2, 3]. The degree of nondeterminism is usually defined as the maximal number of production rules with the same left-hand side. In this paper we consider the degree of

nondeterminism for tree adjoining grammars and head grammars. *Tree adjoining grammars* (TAG for short) were first introduced in [5] and further investigated in [12, 6, 8, 7]. TAGs are tree-generating grammars which use an *adjoining* operation that generates new trees by joining and attaching two different trees at a particular node. *Head Grammars* (HG for short) were first introduced by Pollard in his PhD Thesis [9]. The principle feature which distinguishes a Head Grammar from a context-free grammar is that the head grammar includes a wrapping operation which allows one string to be inserted into another string at a specific point (the head). It is known that for both tree adjoining grammars and head grammars, the class of string languages generated by the grammar is larger than the class of context-free languages — for example, they are able to define the language $a^n b^n c^n d^n$ [12]. In [12] it is shown that the two formalisms generate exactly the same class of string languages, and that these languages are *mildly-context-sensitive* [12, 7, 6].

2. Degree of Nondeterminism for TAGs and HGs

TAGs generate languages by using the *adjoining* operation in which an *auxiliary tree* β is attached to an *elementary tree* γ at a specific node ν. The tree adjoining operation ∇ is illustrated in Figure 1.

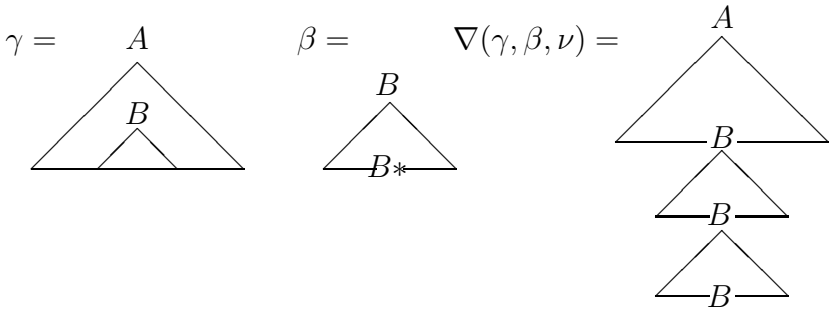


Figure 1: Tree Adjunction

The subtree in γ with the root label B and dominated by a node ν is excised and leaves a copy of ν behind. Then the auxiliary tree β is adjoined at the copy of ν and its root label is identified with the copy of ν. After that, the subtree that was excised is attached to the foot node marked with $B*$ of the auxiliary tree β and its root label is identified with the foot node of β.

The degree of nondeterminism for tree adjoining grammars will be defined as the maximal number of auxiliary trees that can be adjoined to an elementary tree at a given node. When defining the degree of nondeterminism for tree adjoining grammars an essential ambiguity has to be taken into account. There are two possible interpretations of the degree of nondeterminism for tree adjoining grammars. On the one hand, when defining the degree of nondeterminism for a given node in an elementary tree, one could consider only the auxiliary trees in

the set SA which is a set that specifies all auxiliary trees that can be adjoined at that node. On the other hand, one could consider *all* auxiliary trees for the given tree adjoining grammar (even if they are not in the set SA). We will call these views *weak degree of nondeterminism* and *strong degree of nondeterminism* respectively, and we show that the two measures are equivalent.

Head grammars were first introduced by Pollard in his PhD Thesis [9] and were compared with tree adjoining grammars in [13]. In a head grammar, a special position between two symbols, marked by $\uparrow$, is designated as the split point of a string. The split point is the location at which one string can be inserted into another during a wrapping operation. Intuitively, the degree of nondeterminism for head grammars will measure how much choice between productions rules there is when doing a derivation using a specific head grammar.

We show for TAGs and their languages and for HGs and their languages

1. that one can reduce the degree of nondeterminism of a TAG G to 2,
2. that one can reduce the degree of nondeterminism of a HG G to 2, and
3. that tree adjoining grammars with degree of nondeterminism 1 (deterministic tree adjoining grammars) can generate non-context-free languages in contrast to deterministic head grammars which can only generate languages containing a single word

References

[1] H. Bordihn, *Über den Determiniertheitsgrad reiner Versionen formaler Sprachen*. PhD thesis, Technische Universität "Otto von Guericke" Magdeburg, 1992.

[2] H. Bordihn, On The Degree Of Nondeterminism. In: *Developments in Theoretical +Computer Science* (1994), 133–140.

[3] H. Bordihn, S. Aydin, Sequential versus parallel grammar formalisms with respect to measures of descriptional complexity. To appear in Fundamenta Informaticae 2003.

[4] F. Gécseg, M. Steinby, Tree Languages. In G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages*, vol. 3, Springer, 1997.

[5] A. K. Joshi, L. S. Levi, M. Takahashi, Tree adjunct grammars. *J. Comput. System Sci.* **10** (1975), 136–163.

[6] A. K. Joshi, How much context-sensitivity is necessary for characterizing structural descriptions: Tree adjoining grammars. In D. Dowty, L. Karttunen, and A. Zwicky, eds., *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives.* Cambridge University Press, New York, 1985.

[7] A. K. Joshi, K. Vijay-Shanker, D. J. Weir, The convergence of mildly context-sensitive grammar formalism. In P. Sells, S. M. Shieber, and Th. Wasow (eds), *Foundational Issues in Natural Languuage Processing*, MIT Press, Cambridge, 1991.

[8] A. K. Joshi, Y. Schabes, Tree adjoining grammars. In G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages*, vol. 3, Springer, 1997.

[9] C. Pollard, *Generalized Phrase Structure Grammars, Head Grammars and Natural Language*. PhD thesis, Stanford University, 1984.

[10] G. Rozenberg, Extension of tabled 0L systems and languages. *Int. J. of Computer and information sciences* **2** (1973), 311–335.

[11] G. Rozenberg, TOL systems and languages. *Information and Control* **23** (1973), 357–381.

[12] K. Vijay-Shanker, D. J. Weir, The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory* **87** (1994), 511–546.

[13] D. J. Weir, K. Vijay-Shanker, A. K. Joshi, The relationship between tree adjoining grammars and head grammars. In *Proceedings of the 24th Annual Meeting of Computational Linguistics*, New York, NY, June 1986.

THE DEGREE OF PARALLELISM

Henning Bordihn
Institut für Informatik, Universität Potsdam
August-Bebel-Straße 89, D-14482 Potsdam, Germany
e-mail: henning@cs.uni-potsdam.de

and

Henning Fernau
School of Electrical Engineering and Computer Science, University of Newcastle
University Drive, NSW 2380 Callaghan, Australia

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

ABSTRACT

We study the degree of parallelism as a natural descriptional complexity measure of Lindenmayer and Bharat systems. We are interested both in static and in dynamic versions of this notion. We establish corresponding hierarchy and undecidability results. Moreover, we show that this new complexity measure in its dynamic interpretation is linked to the well-known notions of active symbols and finite index, giving some sort of alternative interpretation of these older and well-studied notions.

Keywords: Descriptional complexity, parallel derivations, Lindenmayer systems, Bharat systems.

1. Introduction and general notions

Similar to the proposal in [2], we study the (static) *degree of parallelism* in detail for Lindenmayer systems and for Bharat systems, counting the number of symbols which must be replaced non-identically in a parallel derivation step.

Likewise, one might wish to have a *dynamic* notion of the "degree of parallelism." Then, we would rather count how many symbols are actually going to be replaced non-constantly during the derivation of a word of the given language. Depending on whether we only count the number of "symbol types" which are replaced in parallel or whether we count the number of occurrences of symbols

which are non-constantly replaced, we get connections to the notions of "active symbols" and of "finite index," respectively (where the latter notion first has to be suitably but straightforwardly generalized towards pure grammars).
Notations: If $w \in \Sigma^*$, then $\alpha(w) \subseteq \Sigma$ denotes the set of symbols occurring in w. If $\mathcal{X}$ is a grammar class and Σ is an alphabet, then $\mathcal{X}_\Sigma$ is the collection of grammars from $\mathcal{X}$ with (terminal) alphabet Σ.

2. Definition of language classes

A T0L system is a triple $G = (\Sigma, H, \omega)$, where Σ is an alphabet, H is a finite set of finite substitutions from Σ into Σ^* (i.e., mappings $h : x \mapsto h(x)$, $h(x) \subset \Sigma^*$, $\#h(x) < \infty$, for all $x \in \Sigma$), and $\omega \in \Sigma^*$ is the axiom. If $y \in h(x)$, $x \in \Sigma$, we say that $x \to y$ is a rule in h. For x and y in Σ^*, we write $x \Rightarrow_h y$ for some h in H if and only if $y \in h(x)$. We call a substitution h in H a *table*.

$$L(G) = \{\, w \in \Sigma^* \mid \omega \Rightarrow_{h_{i_1}} w_1 \Rightarrow_{h_{i_2}} \cdots \Rightarrow_{h_{i_m}} w_m = w \text{ with } m \geq 0 \text{ and } h_{i_j} \in H \text{ for } 1 \leq j \leq m \,\}$$

describes the language generated by G. $\mathcal{L}$(T0L) is the class of languages generated by T0L systems. If H actually contains homomorphisms or non-erasing substitutions only, then the T0L system is said to be *deterministic* or *propagating*, and it is called DT0L or PT0L system, respectively. Moreover, a 0L system is a T0L system having only *one* table. This way, the language classes $\mathcal{L}$(DT0L), $\mathcal{L}$(PT0L), $\mathcal{L}$(PDT0L), $\mathcal{L}$(0L), $\mathcal{L}$(D0L), $\mathcal{L}$(P0L), and $\mathcal{L}$(PD0L) arise.

Basic results on the generative power of T0L systems are contained in [5].

The concept of T0S systems forms the sequential counterpart to T0L systems. A T0S system is formally defined like a T0L system, but the derivation is different. For x and y in Σ^*, we write $x \Rightarrow_h y$ for some h in H if and only if $x = z_1 a z_2$ and $y = z_1 v z_2$, for some $z_1, z_2 \in \Sigma^*$ and some $a \to v \in h$. Now, $L(G)$ is defined as in the T0L case.

For each class of languages $\mathcal{L}(Y0L)$ defined above, we obtain the corresponding class $\mathcal{L}(Y0S)$, $Y \in \{\text{PD}, \text{P}, \text{D}, \text{PDT}, \text{PT}, \text{DT}, \text{T}, \varepsilon\}$.

In [4], *pure context-free grammars* are considered which are, in fact, 0S systems having a finite set of axioms instead of a single axiom. Whenever referring to those generalized 0S systems we use the letter F as prefix in our notations, arriving at FY0S systems and languages (where Y is defined as above). The language classes are denoted accordingly. Analogously, FY0L systems/languages are defined.

Bharat systems were introduced by Kudlek in [3]. Formally again, a T0B system $G = (\Sigma, H, \omega)$ looks like a T0L system, the difference lying in the definition of the derivation relation: $x \Rightarrow_h y$ if and only if there is exactly one symbol $a \in \Sigma$ such that all occurrences of a in x are replaced by some word in $h(a)$ to obtain y from x. More precisely, $x \Rightarrow_h y$ for some $h \in H$ if and only if $x = z_0 a z_1 a z_2 \ldots z_{k-1} a z_k$ with $k \geq 0$, $z_i \in (\Sigma \setminus \{a\})^*$, for $0 \leq i \leq k$, $y =$

$z_0v_1z_1v_2z_2\dots z_{k-1}v_kz_k$, and $v_i \in h(a)$, for $1 \le i \le k$. $L(G)$ is then defined as in the T0L case. For each class of languages $\mathcal{L}(Y0\mathrm{L})$ defined above, we obtain the corresponding class $\mathcal{L}(Y0\mathrm{B})$, $Y \in \{\mathrm{F},\varepsilon\}\{\mathrm{PD},\mathrm{P},\mathrm{D},\mathrm{PDT},\mathrm{PT},\mathrm{DT},\mathrm{T},\varepsilon\}$.

3. The static notion of the degree of parallelism

In this section, we will establish the main results (for both hierarchy and decidability questions) on a static interpretation of the notion of the degree of parallelism.

Definition 1 Let $G = (\Sigma, H, \omega)$ be a T0L system. The *static(ally measured) degree of parallelism* of a table $h \in H$ is defined by $\pi^{st}(h) = \#\{\, a \in \Sigma \mid a \to a \notin h \,\}$. Correspondingly, for G we set $\pi^{st}(G) = \max\{\, \pi^{st}(h) \mid h \in H \,\}$. For a language L in $\mathcal{L}(\mathrm{T0L})$, we define

$$\pi^{st}{}_{\mathrm{T0L}}(L) = \min\{\, \pi^{st}(G) \mid G \text{ is a T0L system and } L = L(G) \,\}.$$

The notion $\pi^{st}{}_{\mathcal{X}}(L)$ for other parallel classes $\mathcal{L}(\mathcal{X})$ is defined analogously.
For a given parallel language class $\mathcal{L}(\mathcal{X})$, we will also consider the derived class

$$\mathcal{L}^{st}(\mathcal{X}, k) = \{\, L \in \mathcal{L}(\mathcal{X}) \mid \pi^{st}{}_{\mathcal{X}}(L) \le k \,\}.$$

Let us first consider a few examples to clarify these notions:

Example 1 The language $L = \{\, a^n b \mid n \ge 1 \,\}$ is generatable by each of the following 0L systems:

$$\begin{aligned}
G_1 &= (\{a,b\}, \{a \to a, b \to ab\}, && ab)\\
G_2 &= (\{a,b\}, \{a \to a, a \to aa, b \to b\}, && ab)\\
G_3 &= (\{a,b\}, \{a \to a, b \to b, b \to ab\}, && ab)\\
G_4 &= (\{a,b\}, \{a \to a, a \to aa, b \to b, b \to ab\}, && ab)
\end{aligned}$$

We can observe that $\pi^{st}(G_1) = 1$ and $\pi^{st}(G_2) = \pi^{st}(G_3) = \pi^{st}(G_4) = 0$, so that $\pi^{st}{}_{0\mathrm{L}}(L) = 0$.

This example illustrates the intuition that, the lower the degree of parallelism of a language is, the "more sequential" a system can be which generates this language.

Example 2 The language $L_n = \{\, a_1^{2^i} a_2^{2^i} \dots a_n^{2^i} \mid i \ge 0 \,\}$ over the alphabet $\Sigma_n = \{a_1, \dots, a_n\}$ is generatable by the 0L system $G_n = (\Sigma_n, \{\, a_i \to a_i^2 \mid 1 \le i \le n \,\}, a_1a_2\dots a_n)$. Hence, $\pi^{st}{}_{0\mathrm{L}}(L_n) \le n$. Actually, this assertion is not too interesting, since whenever $L \in \mathcal{L}(\mathcal{X})$ and $L \subseteq \Sigma^*$, then $\pi^{st}{}_{\mathcal{X}}(L) \le \#\Sigma$. If we consider the system G_n as a Bharat system, then L'_n is generated with

$$L'_n = \{\, a_1^{2^{i_1}} a_2^{2^{i_2}} \dots a_n^{2^{i_n}} \mid i_j \ge 0 \text{ for } 1 \le j \le n \,\}.$$

Our first main theorem is the following hierarchy result, which also justifies the definition of degree of parallelism as given. The proof is mainly based on the languages L_n of Example 2.

Theorem 1 *The measure π^{st} induces the following infinite, strict hierarchy: For $Y \in \{\mathrm{F},\varepsilon\}\{\mathrm{PD},\mathrm{P},\mathrm{D},\mathrm{PDT},\mathrm{PT},\mathrm{DT},\mathrm{T},\varepsilon\}$ and every integer $n \geq 0$, we have:*

$$\mathcal{L}^{st}(Y0\mathrm{L},n) \subset \mathcal{L}^{st}(Y0\mathrm{L},n+1)$$

If $\mathrm{D} \notin \alpha(Y)$, we have furthermore the characterization $\mathcal{L}(Y0\mathrm{S}) = \mathcal{L}^{st}(Y0\mathrm{L},0)$ for the lowest language class.[1]

Moreover, for languages over any fix alphabet Σ, we get:

$$\mathcal{L}^{st}(Y0\mathrm{L}_\Sigma,0) \subset \mathcal{L}^{st}(Y0\mathrm{L}_\Sigma,1) \subset \ldots \subset \mathcal{L}^{st}(Y0\mathrm{L}_\Sigma,\#\Sigma) = \mathcal{L}(Y0\mathrm{L}_\Sigma)$$

By taking the languages L'_n from Example 2, we can prove a completely analogous result for non-tabled Bharat systems.

In case of Bharat systems, the following deterministic T0B system $G_n = (\Sigma_n, H_n, a_1 \ldots a_n)$ generates L'_n: $H_n = \{h_1,\ldots,h_n\}$ with h_i containing the rule $a_i \to a_i^2$ as only non-identity rule. According to Definition 1, the static degree of parallelism of such a system would equal one. The possibility of distributing rules amongst (more and more) tables is inherent in the definition of Bharat systems, since only one symbols is finally picked to be replaced. Hence:

Lemma 2 $\mathcal{L}^{st}(Y\mathrm{T0B},1) = \mathcal{L}(Y\mathrm{T0B})$ *for $Y \in \{\mathrm{F},\varepsilon\}\{\mathrm{PD},\mathrm{P},\mathrm{D},\varepsilon\}$.*

Therefore, the following definition is possibly more appropriate for Bharat systems:

Definition 2 Let $G = (\Sigma, H, \omega)$ be a T0B system not containing the useless table $\{\, a \to a \mid a \in \Sigma \,\}$. The *modified static(ally measured) degree of parallelism* of a table G is defined by $\pi^{st'}(G) = \#\{\, a \in \Sigma \mid \forall h \in H : a \to a \notin h \,\}$. Corresponding to Definition 1, we can define the measures $\pi^{st'}_{\mathcal{X}}(L)$ and the derived language classes $\mathcal{L}^{st'}(\mathcal{X},k)$.

This modified definition allows us to establish a complete analogue of Theorem 1 for both Lindenmayer and Bharat systems. Observe that, for systems G with only one table, we have $\pi^{st}(G) = \pi^{st'}(G)$.

After having established the hierarchies, it is natural to ask whether there exists an algorithm which computes the static degree of parallelism of a given language.

[1] In the case of deterministic systems, the immediate connection to 0S systems is lost: $\mathcal{L}^{st}(\mathrm{D0L},0)$ (and similar classes) characterize the singleton languages, and $\mathcal{L}^{st}(\mathrm{FD0L},0)$ (etc.) are just the finite languages, because the only possible rules are of the form $a \to a$.

Definition 3 The problem of *deciding a concrete static degree of parallelism* for $\mathcal{X}$-systems is specified as follows:
Given: an $\mathcal{X}$-system G, a number k, $k \geq 0$
Question: Is ${\pi^{st}}_{\mathcal{X}}(L(G)) = k$?

The problem of *deciding the optimality of the static degree of parallelism* for $\mathcal{X}$-systems is specified as follows:
Given: an $\mathcal{X}$-system G
Question: Is ${\pi^{st}}_{\mathcal{X}}(L(G)) = \pi^{st}(G)$?

Observe that, instead of the above definition for deciding a concrete degree of parallelism, we could have also asked the following question: Is ${\pi^{st}}_{\mathcal{X}}(L(G)) \leq k$? Fortunately, both questions are (polynomial-time) equivalent. Since $\pi^{st}(G)$ is computable for every system G, the optimality question can be solved with the help of the equality question.

By using Post's correspondence problem, we obtain:

Theorem 3 *Let $Y \in \{\mathrm{F}, \varepsilon\}\{\mathrm{P}, \mathrm{T}, \mathrm{PT}, \varepsilon\}$. Both the questions of the optimality and of the concrete degree of parallelism are undecidable for Y0L and for Y0B systems.*

This statement remains valid with respect to the variant $\pi^{st'}$ for Bharat systems.

The next statement immediately follows when the connection of the classes $\mathcal{L}^{st}(X, 0)$ to sequential mechanisms is taken into consideration.

Corollary 4 *Let $X \in \{\mathrm{F}, \varepsilon\}\{\mathrm{P}, \mathrm{T}, \mathrm{PT}, \varepsilon\}\{0\mathrm{L}, 0\mathrm{B}\}$.*
Both 0S-ness problem *for X systems and* context-freeness *for X systems are undecidable.*

Corollary 5 *Let $X \in \{\mathrm{F}, \varepsilon\}\{\mathrm{P}, \mathrm{T}, \mathrm{PT}, \varepsilon\}\{0\mathrm{L}, 0\mathrm{B}\}$ and $k \geq 1$. Then, there is no algorithm which, given an X system G with $\pi^{st}(G) = k$, decides whether or not $\pi^{st}(L(G)) \leq k - 1$.*

We conclude this section by noting that there seems to be an interesting connection between the degree of parallelism of deterministic Lindenmayer systems and its growth function (as defined, e.g., in [5]), which also shows that "classical" notions in the area of Lindenmayer systems are linked to the notion of the degree of parallelism studied in this paper. Consider G_k with the alphabet $\{a_0, a_1, \ldots, a_k\}$ and rules $a_{i+1} \to a_i a_{i+1}$ for $0 \leq i < k$ and $a_0 \to a_0$ to see:

Lemma 6 *For each $k \geq 1$, there is a D0L system G_k with $\pi^{st}(G) = k$ and whose growth function is a kth order polynomial.*

We actually conjecture that ${\pi^{st}}_{\mathrm{D0L}}(L(G_k)) = k$. Moreover, we think that some kind of converse direction is also true: if $\pi^{st}(G) = k$ and the D0L system G has a polynomial growth function, then its growth function is a polynomial of degree (at most) k.

4. Dynamic notions of the degree of parallelism

In this section, we shortly present two definitions (interpretations) for measuring the degree of parallelism in a dynamic fashion.

Definition 4 Let $G = (\Sigma, H, \omega)$ be a T0L system. Consider a derivation step $x \underset{h}{\Longrightarrow} y$ according to table $h \in H$, with $x = a_1 a_2 \dots a_k$. The *dynamical(ly measured) degree of parallelism (for symbols)* in this derivation step is defined by

$$\pi^{dynsb}(x \Rightarrow y, h) = \min_{y = w_1 \dots w_k, a_i \to w_i \in h} \#\{\, a_j \mid a_j \neq w_j \,\} \tag{4.1}$$

(Possible unambiguities enforce to consider all possible ways of splitting y.)

For a derivation $D : x = x_0 \underset{h_{i_1}}{\Longrightarrow} x_1 \underset{h_{i_2}}{\Longrightarrow} x_2 \underset{h_{i_3}}{\Longrightarrow} \cdots \underset{h_{i_k}}{\Longrightarrow} x_k = y$ by G, let

$$\pi^{dynsb}(D) = \max\{\, \pi^{dynsb}(x_\ell \Rightarrow x_{\ell+1}, h_{i_{\ell+1}}) \mid 0 \leq \ell \leq k-1 \,\}. \tag{4.2}$$

For $w \in L(G)$, we define

$$\pi^{dynsb}(w, G) = \min\{\, \pi^{dynsb}(D) \mid D \text{ is a derivation } \omega \overset{*}{\Rightarrow} w \text{ by } G \,\}. \tag{4.3}$$

Now, we set

$$\pi^{dynsb}(G) = \sup(\{\, \pi^{dynsb}(w, G) \mid w \in L(G) \,\} \cup \{0\}), \tag{4.4}$$

and for a language L in $\mathcal{L}(\mathrm{T0L})$, we define

$$\pi^{dynsb}{}_{\mathrm{T0L}}(L) = \min\{\, \pi^{dynsb}(G) \mid G \text{ is a T0L system, } L = L(G) \,\}. \tag{4.5}$$

The notion $\pi^{dynsb}{}_{\mathcal{X}}(L)$ for other parallel language classes $\mathcal{L}(\mathcal{X})$ is defined analogously.

Actually, we could have used max instead of sup in Equation (4.4), but the present formulation is more suitable for the next (alternative) definition.

Observe that, apart from a small technical detail which actually should be changed in [1] (without affecting the results proved in that paper), this dynamic notion of a degree of parallelism literally coincides with the dynamic notion of the number of active symbols as introduced in [1].

There is also a static notion of active symbols (surveyed in [1]), where the symbols are counted which can be replaced non-identically, but one disregards whether or not an identical replacement is possible, too. Therefore, this notion does not coincide with the static degree of parallelism.

With some refined arguments, we are able to prove analogues of the hierarchy theorems again with the help of Example 2. Yet, for the characterization of the lowest language class, we have:

Lemma 7 *For $Y \in \{\mathrm{F}, \varepsilon\}\{\mathrm{P}, \mathrm{PT}, \mathrm{T}, \varepsilon\}$ and $X \in \{\mathrm{L}, \mathrm{B}\}$, we have:*

1. $\mathcal{L}^{dynsb}(Y0X, 0)$ *characterizes the singleton languages.*
2. $\mathcal{L}^{dynsb}(Y0X, 0) \subset \mathcal{L}(Y0S) = \mathcal{L}^{st(\prime)}(Y0X, 0) \subset \mathcal{L}^{dynsb}(Y0X, 1)$.

Analogously to the static case, we find:

Theorem 8 *The question of the optimality of the dynamic degree of parallelism (for symbols) is undecidable for P0L and for P0B systems.*

However, due to the trivial nature of systems G with $\pi^{dynsb}(G) = 0$, the question whether or not a system G obeys $\pi^{dynsb}(G) = 0$ is decidable, while being undecidable in the static case.

Corollary 9 *Let $X \in \{\mathrm{F}, \varepsilon\}\{\mathrm{P}, \mathrm{T}, \mathrm{PT}, \varepsilon\}\{\mathrm{0L}, \mathrm{0B}\}$ and $k \geq 2$. Then, there is no algorithm which, given an X system G with $\pi^{dynsb}(G) = k$, decides whether or not $\pi^{dynsb}(L(G)) \leq k - 1$.*

Corollary 10 *Let $X \in \{\mathrm{F}, \varepsilon\}\{\mathrm{P}, \mathrm{T}, \mathrm{PT}, \varepsilon\}\{\mathrm{0L}, \mathrm{0B}\}$. Then, there is no algorithm which, given an X system G decides whether or not $\pi^{st}(L(G)) = \pi^{dynsb}(L(G))$.*

Corollary 11 *Let $X \in \{\mathrm{F}, \varepsilon\}\{\mathrm{P}, \mathrm{T}, \mathrm{PT}, \varepsilon\}\{\mathrm{0L}, \mathrm{0B}\}$ and $k \geq 2$. Then, there is no algorithm which, given an X system G with $\pi^{dynsb}(G) = k$, decides* 0S*-ness or context-freeness of $L(G)$.*

Instead of giving a full definition, let us only indicate the necessary changes in Definition 4 to arrive at a notion of *dynamical(ly measured) degree of parallelism for symbol occurrences*. Equation (4.1) should now read:

$$\pi^{dynocc}(x \Rightarrow y, h) = \min_{y = w_1 \ldots w_k, a_i \to w_i \in h} \#\{\, j \mid a_j \neq w_j \,\} \tag{4.6}$$

Replacing π^{dynsb} by π^{dynocc} in the following equations (4.2) through (4.5) defines the measure π^{dynocc} finally for grammars and languages.

In fact, this measure can be interpreted as a "nonterminal-free" analogue to the well-known notion of finite index, where the number of occurrences of nonterminals in sentential forms is bounded. This notion is best explained by means of an example:

Example 3 The language $L_n = \{\, (ab^i)^n \mid i \geq 0 \,\}$ over the alphabet $\Sigma = \{a, b\}$ is generatable by the 0L system $G_n = (\Sigma, \{\, a \to ab, b \to b \,\}, a^n)$. Now, $\pi^{st}{}_{\mathrm{0L}}(G_n) = 1$ and $\pi^{dynsb}{}_{\mathrm{0L}}(G_n) = 1$, but $\pi^{dynocc}{}_{\mathrm{0L}}(G_n) = n$. The same assertion is true when considering G_n as a Bharat system.

This example shows that both notions of a dynamic degree of parallelism may deviate completely for concrete results. In fact, as the language L_1 from Example 2 shows, there are languages with $\pi^{dynocc}(L) = \infty$, while $\pi^{dynsb}(L) = 1$.

Generally speaking, we can only estabish the following (trivial) relationship between both dynamic degree notions:

Lemma 12 *If $\mathcal{X}$ is a parallel grammar class and $L \in \mathcal{L}(\mathcal{X})$, then $\pi^{dynsb}{}_{\mathcal{X}}(L) \leq \pi^{dynocc}{}_{\mathcal{X}}(L)$.*

We can also establish connections to the notion of a growth function in D0L systems:

Theorem 13 *If G is a* D0L *system with exponential growth function, then* $\pi^{dynocc}{}_{\mathrm{D0L}}(G) = \pi^{dynocc}{}_{\mathrm{D0L}}(L(G)) = \infty$.

Example 3 and Theorem 13 can be used to prove the following hierarchy result:

Theorem 14 *The measure π^{dynocc} induces the following infinite, strict hierarchies. For $Y \in \{\mathrm{F},\varepsilon\}\{\mathrm{PD},\mathrm{P},\mathrm{D},\mathrm{PDT},\mathrm{PT},\mathrm{DT},\mathrm{T},\varepsilon\}$, $X \in \{\mathrm{L},\mathrm{B}\}$ and every integer $n \geq 1$, we have:*

$$\mathcal{L}^{dynocc}(Y0X,n) \subset \mathcal{L}^{dynocc}(Y0X,n+1) \subset \mathcal{L}^{dynocc}(Y0X,\infty).$$

We have the characterization $\mathcal{L}(Y0\mathrm{S}) = \mathcal{L}^{dynocc}(Y0X,1)$ for the lowest language class.

We conclude this section by stating some further undecidability results.

Theorem 15 *For $Y \in \{\mathrm{F},\varepsilon\}\{\mathrm{P},\mathrm{PT},\mathrm{T},\varepsilon\}$, $X \in \{\mathrm{L},\mathrm{B}\}$, the following questions are undecidable:*

1. *Given a $Y0X$ system G and $n = 1,2,\ldots,\infty$, is $\pi^{dynocc}(G) = n$?*
2. *Given a $Y0X$ system G and $n = 1,2,\ldots,\infty$, is $\pi^{dynocc}(L(G)) = n$?*

Similarly, an analogue to Corollary 11 can be shown.

References

[1] H. Bordihn and M. Holzer. On the number of active symbols in L and CD grammar systems. *J. Automata, Languages and Combinatorics* 6:411–426, 2001.

[2] H. Fernau. Parallel grammars: a phenomenology. *GRAMMARS* 6:25–87, 2003.

[3] M. Kudlek. Indian parallel systems. In *Foundations of Software Technology and Theoretical Computer Science FSTTCS, 2nd conference*, pp. 283–289, 1982.

[4] H. Maurer, A. Salomaa and D. Wood. Pure grammars. *Inform. & Contr.* 44:47–72, 1980.

[5] G. Rozenberg and A. K. Salomaa. *The Mathematical Theory of L Systems.* AP, 1980.

FORMAL LANGUAGE ASPECTS OF IDENTIFIABLE LANGUAGE CLASSES

Henning Fernau
School of Electrical Engineering and Computer Science
University of Newcastle, University Drive
NSW 2380 Callaghan, Australia

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

ABSTRACT

We exhibit—by way of examples—some properties of language classes which are identifiable in the limit from positive samples, a notion coined by E. M. Gold back in 1967. Especially, we focus on closure properties and on combinatorial properties of these classes, ending with a short list of open problems in that area.

Keywords: Identification in the limit, closure properties, distinguishability.

1. Introduction and general notions

Identification in the limit from positive samples, also known as *exact learning from text* as proposed by Gold [5], is one of the oldest yet most important models of grammatical inference. Since not all regular languages can be learned exactly from text, the characterization of *identifiable* subclasses of regular languages is a popular line of research, although in practice the learning of non-regular languages is probably even more important. However, that is still an only rarely touched area, and we will focus on the regular language case in what follows.

Definition 1 Consider a language class $\mathcal{L}$ defined via a class of language describing devices $\mathcal{D}$ as, e.g., grammars or automata. $\mathcal{L}$ is said to be *identifiable* if there is a so-called *inference machine* IM to which as input an arbitrary language $L \in \mathcal{L}$ may be enumerated (possibly with repetitions) in an arbitrary order, i.e., IM receives an infinite input stream of words $E(1)$, $E(2)$, ..., where $E : \mathbb{N} \to \mathbb{L}$ is an enumeration of L, i.e., a surjection, and IM reacts with an output stream

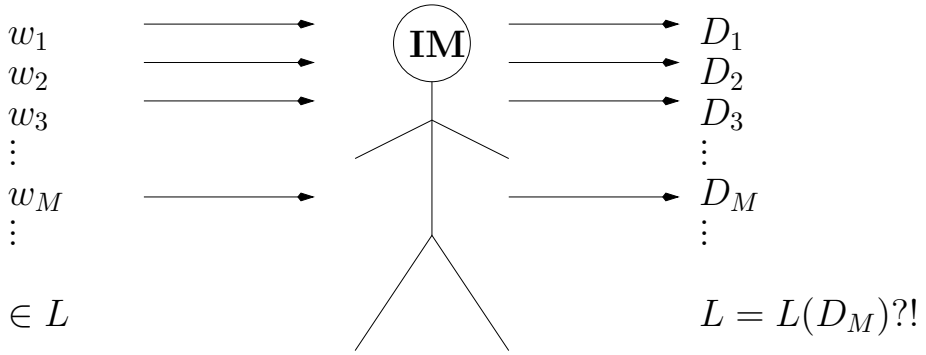


Figure 1: Gold's learning scenario

$D_i \in \mathcal{D}$ of devices such that there is an $N(E)$ so that, for all $n \geq N(E)$, we have $D_n = D_{N(E)}$ and, moreover, the language defined by $D_{N(E)}$ equals L.

Figure 1 tries to illustrate this learning scenario for a fixed language class $\mathcal{L}$ described by the device class $\mathcal{D}$. Often, it is convenient to view IM mapping a finite sample set $I_+ = \{w_1, \ldots, w_M\}$ to a hypothesis D_M. The aim is then to find algorithms which, given I_+, produce a hypothesis D_M describing a language $L_M \supseteq I_+$ such that, for any language $L \in \mathcal{L}$ which contains I_+, $L_M \subseteq L$. In other words, L_M is the smallest language in $\mathcal{L}$ extending I_+.

Gold [5] has already established:

Lemma 1 *The class of regular languages is not identifiable.*

Observe that the above result is valid for *any presentation* of the regular languages.

Let us underline the *algorithmic nature* of language classes fitting into Gold's learning model: in principle, it is possible to define language classes by giving a learning algorithm.

Example 1 (k-gram approach) Let T be an ordered alphabet; this gives a lexicographic ordering on T^*. This ordering can then (somehow) be extended to an ordering on the regular expressions of star height one.

Consider now the following k-gram learner: given a sequence of input words $I_+ = \{w_1, \ldots, w_m\} \subset T^*$, this learner will extract all k-letter subwords from I_+, yielding a set $S(k) \subseteq T^k$, and then output the smallest regular expression (according to the given ordering) which describes $T^*S(k)T^* \cup \{w_i \mid 1 \leq i \leq m\}$.

In actual fact, when looking at "practical descriptions" of learning algorithm, you would rather find the following kind of description (or probably something less formal), assuming we deal with the alphabet $T = \{a, b\}$:

```
FOR each word w ∈ I+ DO
  IF |w| < k THEN output w
  ELSE FOR each subword v of w of length k DO output (a|b)*v(a|b)*
```

The interpretation is the the output regular expressions are to be catenated. Observe that the latter description is not independent of the order in which the words are presented to the learner. Nonetheless, as explained above, it is rather straightforward in this case to translate that learning algorithm description into an algorithm fitting into Gold's paradigm. Moreover, it is pretty simple to see here that not all regular languages can be identified, but only (apart from "short words") those which can be desribed as unions of languages of the form $T^*\{v\}T^*$ for some $v \in T^k$.

In general, the (natural) question arises: what language class is actually defined by a given learning algorithm. In actual fact, most of the learning algorithms published up to now (and probably even worse: most of the algorithms applied in the "real world") are mere heuristics in the sense that nobody ever ventured to explore what kind of languages can be learnt this way. From a formal language point of view, this gives you a rich source of (small) research topics: just take any learning heuristic and try to characterize the corresponding language class. Actually, this *characterization problem* is considered to be one of the most important and challenging tasks both in the area of *grammatical inference* itself [6] and when you speak with people who build "intelligent" components into larger software packages.

2. Function distinguishability

We are now going to define in rather general terms a way to get identifiable regular language classes.

Let F be some finite set. A mapping $f : T^* \to F$ is called a *distinguishing function* if $f(w) = f(z)$ implies $f(wu) = f(zu)$ for all $u, w, z \in T^*$. Examples of distinguishing functions include the terminal function [7] $\mathrm{Ter}(x) = \{\, a \in T \mid \exists u, v \in T^* : uav = x \,\}$ and the suffix function $T_k(x)$ yielding the last $\min(k, |x|)$ letters of x, corresponding to the k-reversible languages [1]. To every distinguishing function f, a finite automaton $A_f = (F, T, \delta_f, f(\lambda), F)$ can be associated by setting $\delta_f(q, a) = f(wa)$, where $w \in f^{-1}(q)$ can be chosen arbitrarily, since f is a distinguishing function.

f-DL can be characterized by f-distinguishable automata defined in the following way:

Definition 2 Let $A = (Q, T, \delta, q_0, Q_F)$ be a finite automaton. Let $f : T^* \to F$ be a distinguishing function. A is called *f-distinguishable* if:

1. A is deterministic.
2. For all states $q \in Q$ and all $x, y \in T^*$ with $\delta^*(q_0, x) = \delta^*(q_0, y) = q$, we have $f(x) = f(y)$.
   (In other words, for $q \in Q$, $f(q) := f(x)$ for some x with $\delta^*(q_0, x) = q$ is well-defined.)

3. For all $q_1, q_2 \in Q$, $q_1 \neq q_2$, with either (a) $q_1, q_2 \in Q_F$ or (b) there exist $q_3 \in Q$ and $a \in T$ with $\delta(q_1, a) = \delta(q_2, a) = q_3$, we have $f(q_1) \neq f(q_2)$.

There is also a "normal form representation" of languages in $L \in$ (-DLf), namely the "stripped version" of $A_f \times A(L)$, denoted $A(L, f)$, where $A(L)$ is the minimal state automaton of L.

In [4], we developed an inference algorithm for f-DL based on the state-merging paradigm, similar to the inference algorithm for 0-reversible languages given by Angluin [1]. As usual, this algorithm starts with the so-called prefix tree automaton obtained from the given input sample I_+ from which an automaton generating I_+ plus possibly some other strings has to be induced. More specifically, the algorithm keeps merging states as long as there are conflicts in these states according to Def. 2. When being implemented by a union-find algorithm, the complexity of the algorithm is basically dependent on the number of union and of find operations which are triggered by merging of states plus the ones triggered in the initiation phase. More precisely, in the backward nondeterminism case, $|F|$ pairs are only created if at least some state of the automaton A_f actually has $|F|$ predecessors. For simplicity, we will call this the *indegree* of f, written $I(f)$ for short. In particular, in the automaton A_{Ter}, every state has at most $|T|$ predecessors (where T is the alphabet of the language to be inferred), and the same statement is true for the automaton A_{σ_k}.

Our observations can be summarized as follows:

Theorem 2 (Time complexity) *By using a standard union-find algorithm, the algorithm f-*`Ident` *as described in [4] can be implemented to run in time*

$$O(\alpha(2(I(f)+1)(|T|+1)n, n)(I(f)+1)(|T|+1)n),$$

where α is the inverse Ackermann function and n is the total length of all words in I_+ from language L, when L is the language presented to the learner for f-DL.

Ignoring the α-term, this means that terminal distinguishable and k-reversible languages can be inferred in time $O(|T|^2 n)$, which considerably improves the time bounds from [1, 7]. We currently don't know how to nicely upperbound $I(\Delta_k)$, where Δ_k was defined in [4] in order to generalize the piecewise k-testable languages [8].

3. An extended example

Radhakrishnan showed that the language L described by $ba^*c + d(aa)^*c$ lies in Ter-DL but its reversal does not. Consider the deterministic (minimal) automaton $A(L)$ with transition function δ (see Table 1). Is $A(L)$ Ter-distinguishable? We have still to check whether it is possible to resolve the backward nondeterminism conflicts (the state 3 occurs two times in the column labelled c). This

	a	b	c	d
$\to 0$	–	1	–	2
1	1	–	3	–
2	4	–	3	–
$3 \to$	–	–	–	–
4	2	–	–	–

Ter		a	b	c	d
$\emptyset$	$\to 0$	–	1	–	2
$\{b\}$	1	$1'$	–	3	–
$\{a,b\}$	$1'$	$1'$	–	$3'$	–
$\{d\}$	2	4	–	$3''$	–
$\{a,d\}$	$2'$	4	–	$3'''$	–
$\{b,c\}$	$3 \to$	–	–	–	–
$\{a,b,c\}$	$3' \to$	–	–	–	–
$\{c,d\}$	$3'' \to$	–	–	–	–
$\{a,c,d\}$	$3''' \to$	–	–	–	–
$\{a,d\}$	4	$2'$	–	–	–

Ter		a	b	c
$\emptyset$	$\to 0$	–	1	–
$\{b\}$	1	$1'$	–	3
$\{a,b\}$	$1'$	$1'$	–	$3'$
$\{b,c\}$	$3 \to$	–	–	–
$\{a,b,c\}$	$3' \to$	–	–	–

Table 1: The transition functions δ, δ_{Ter} and δ_{inferred}.

resolution possibility is formalized in the second and third condition in Definition 2. As to the second condition, the question is whether it is possible to assign Ter-values to states of $A(L)$ in a well-defined manner: assigning $\text{Ter}(0) = \emptyset$ and $\text{Ter}(4) = \{a, d\}$ is possible, but should we set $\text{Ter}(1) = \{b\}$ (since $\delta^*(0, b) = 1$) or $\text{Ter}(1) = \{a, b\}$ (since $\delta^*(0, ba) = 1$)?; similar problems occur with states 2 and 3. Let us therefore try another automaton accepting L, whose transition function δ_{Ter} is given by Table 1, we indicate the Ter-values of the states in the first column of the table. As the reader may verify, δ_{Ter} basically is the transition table of the stripped subautomaton of the product automaton $A(L) \times A_{\text{Ter}}$. One source of backward nondeterminism may arise from multiple final states, see condition 3.(a) of Def. 2. Since the Ter-values of all four finite states are different, this sort of nondeterminism can be resolved. Let us consider possible violations of condition 3.(b) of Def. 2. In the column labelled a, we find multiple occurrences of the same state entry:

- $\delta_{\text{Ter}}(1, a) = \delta_{\text{Ter}}(1', a) = 1'$: since $\text{Ter}(1) = \{b\} \neq \text{Ter}(1') = \{a, b\}$, this conflict is resolvable.
- $\delta_{\text{Ter}}(2, a) = \delta_{\text{Ter}}(2', a) = 4$: since $\text{Ter}(2) = \{d\} \neq \text{Ter}(2') = \{a, d\}$, this conflict is resolvable.

Observe that the distinguishing function f can be also used to design efficient "backward scanning" algorithms for languages in f-DL. The only thing one has to know are the f-values of all prefixes of the word w to be scanned. Let us try to check that $daac$ belongs to the language L in a backward fashion. For the prefixes, we compute: $\text{Ter}(d) = \{d\}$, $\text{Ter}(da) = \text{Ter}(daa) = \{a, d\}$, and $\text{Ter}(daac) = \{a, c, d\}$. Since $\text{Ter}(w_1) = \{a, c, d\}$, we have to start our backward

	a	b	c	d
$\to 0$	–	–	1	–
1	2	3	–	3
2	1	3	–	–
$3 \to$	–	–	–	–

	a	b	c	d
$\to (0, \{\lambda\})$	–	–	$(1, \{c\})$	–
$(1, \{c\})$	$(2, \{a, c\})$	$(3, \{b, c\})$	–	$(3, \{c, d\})$
$(1, \{a, c\})$	$(2, \{a, c\})$	$(3, \{a, b, c\})$	–	$(3, \{a, c, d\})$
$(2, \{a, c\})$	$(1, \{a, c\})$	$(3, \{a, b, c\})$	–	–
$(3, X) \to$	–	–	–	–

Table 2: The transition functions of A_{L^R} and $A(L^R, \mathrm{Ter})$; X is one of $\{a, b, c\}$, $\{a, c, d\}$, $\{b, c\}$ and $\{c, d\}$.

scan in state $3'''$. The column labelled c reveals that after reading the last letter c, we are in state $2'$. After reading the penultimate letter a, we are therefore in state 4. Reading the second letter a brings us into state 2, since the Ter-value of the prefix left to be read is $\{d\} = \mathrm{Ter}(2)$. Finally, reading d brings us to the initial state 0; hence, *daac* is accepted by the automaton.

Let us discuss why L^R described by $ca^*b + c(aa)^*d$ is *not* in Ter-DL, as already Radhakrishnan claimed (without proof) Table 2 shows the transition function of the minimal deterministic automaton A_{L^R} and the transition function of $A(L^R, \mathrm{Ter})$. As the reader may verify, $A(L^R, \mathrm{Ter})$ is *not* Ter-distinguishable. Our characterization theorem implies that L^R is not Ter-distinguishable either. A similar argument shows that L^R is not σ_1-distinguishable. On the contrary, L^R is σ_2-distinguishable. This can be seen by looking at $A(L^R, \sigma_2)$.

4. More formal language properties

Here, we are going to list closure properties of f-DL, referring to proofs only if they are due to the "algorithmic nature" of the language class.

1. For each f, f-DL is closed under intersection.
2. Let $w \in T^*$ and let $f : T^* \to F$ be a distinguishing function. Then, $\{w\}$ and $\{w\}^*$ are in f-DL.
3. Let $f : T^* \to F$ be a distinguishing function and let $a \in T$ be some letter. For any $k > 0$, define
$$L_k = \{a, aa, \ldots, a^k\}.$$
Then, there is some $n > 0$ such that $L_n \in f$-DL and $L_{n+1} \notin f$-DL.

Proof. By Pt. 2, both $\{a\}$ and $\{a\}^*$ are in f-DL. As we will show in this paper,[1] for each distinguishing function f, the class f-DL is learnable from

[1] We will not use the results of this subsection in order to derive the mentioned learnability results, so that there is no circularity in the argument.

text. Due to the result of Gold [5] stating that no superfinite language class is identifiable, we would arrive at a contradiction assuming that the claim of this lemma is not true. □

4. For any f, f-DL is not closed under union (and hence not under complementation).

   Proof. Let $f : T^* \to F$ be an arbitrary distinguishing function. Consider some $a \in T$. According to Pt. 3, $L_n \in f\text{-DL}$ and $L_{n+1} \notin f\text{-DL}$ for some $n > 0$. According to Pt. 2, $\{a^{n+1}\} \in f\text{-DL}$. If f-DL were closed under union, then $L_{n+1} = L_n \cup \{a^{n+1}\}$ would be in f-DL, too, which is an obvious contradiction. □

5. For any f, f-DL is not closed under intersection with finite languages.

   Proof. Let $f : T^* \to F$ be an arbitrary distinguishing function. Consider some $a \in T$. According to Pt. 2, $\{a\}^* \in f\text{-DL}$. According to Pt. 3, $L_{n+1} \notin f\text{-DL}$ for some $n > 0$. If f-DL were closed under intersection with finite languages, then $L_{n+1} = \{a\}^* \cap L_{n+1}$ would belong to f-DL, an obvious contradiction. □

Other closure properties cannot be treated in the same uniform way, e.g., there are distinguishing functions f_1 and f_2 such that f_1-DL is closed under reversals (namely, σ_k-DL, see [1]) but f_2-DL is*not* closed under reversals (see Sec. 3).

5. Formal language prospects

Observe one peculiarity in the previous examples (which is typical for language classes that can be defined by identification algorithms): it is quite easy to prove non-membership of a certain language into such a class: simply "feed" the language "in a suitable way" into the learning algorithm and observe *overgeneralization* in one point. In this way, it is often relatively easy to show non-closure properties which are rather typical for "algorithmically defined" language classes.

Furthermore, note that "on the way", many well-known (?) formal language notions are necessary to actually establish that a certain identification algorithm describes a certain language class. Of utmost importance is the "invention" of suitable normal forms. This makes it possible that the identification process always converges to the same automaton (except for state names), irrespectively of the order in which the samples are shown to the inference machine.

Using established formal language constructions, function distinguishable languages have been proposed to help infer DTDs for XML documents and in the case of tree language inference [2, 3].

Finally, identifiable language classes may give rise to new decidability questions as the suitability of a certain choice of a learnable language class. To be more

concrete, what kind of distinguishing function f should a "user" choose for his/her purpose? If f is "too simple," then f-DL might not meet the needs; if f is "too complicated," then the learning process will be rather slow, possibly yielding intermediate generalizations which are "too complex."

This gives rise to decidability questions like: Given some regular languages $L_1, \ldots, L_r$, find the "most simple" f such that $L_1, \ldots, L_r$ are all in f-DL. The complexities of these and similar questions are open (even if $r = 1$ in the example).

References

[1] D. Angluin. Inference of reversible languages. *J. ACM*, 29(3):741–765, 1982.

[2] H. Fernau. Learning XML grammars. In *Proc. Machine Learning and Data Mining in Pattern Recognition MLDM*, vol. 2123 of *LNCS/LNAI*, pp. 73—87, 2001.

[3] H. Fernau. Learning tree languages from text. In *Computational Learning Theory COLT*, vol. 2375 of *LNCS/LNAI*, pp. 153–168, 2002.

[4] H. Fernau. Identification of function distinguishable languages. *Theoret. Comp. Sci.*, 290:1679–1711, 2003.

[5] E. M. Gold. Language identification in the limit. *Inform. & Contr.* 10:447–474, 1967.

[6] C. de la Higuera. Current trends in grammatical inference. In *Advances in Pattern Recognition SSPR+SPR*, vol. 1876 of *LNCS*, pp. 28–31, 2000.

[7] V. Radhakrishnan and G. Nagaraja. Inference of regular grammars via skeletons. *IEEE Trans. Systems, Man and Cybernetics SMC*, 17(6):982–992, 1987.

[8] J. Ruiz and P. García. Learning k-piecewise testable languages from positive data. In *Proc. Intern. Coll. Grammatical Inference ICGI*, vol. 1147 of *LNCS/LNAI*, pp. 203–210, 1996.

P-AUTOMATEN UND ω-P-AUTOMATEN

Rudolf Freund Marion Oswald
Institut für Computersprachen, Technische Universität Wien
Favoritenstraße 9, A-1040 Wien, Österreich
e-mail: {rudi,marion}@emcc.at

und

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

KURZFASSUNG

Wir untersuchen Varianten von akzeptierenden P-Systemen mit reinen Kommunikationsregeln und zeigen, dass diese P-Automaten selbst mit der einfachsten Membranstruktur sowohl bei der Akzeptanz von Multimengen als auch bei der Akzeptanz von Wörtern (gegeben als Folge von Terminalsymbolen, die aus der Umgebung geholt werden) die gleiche Berechnungskapazität wie Turingmaschinen besitzen. Bei der Akzeptanz von ω-Sprachen erreichen ω-P-Automaten mit zwei Membranen die gleiche Berechnungskapazität wie ω-Turingmaschinen. Überdies zeigen wir, dass P-Automaten bzw. ω-P-Automaten, welche aus nur einer Membran bestehen und Regeln einer sehr eingeschränkten speziellen Form verwenden, reguläre Sprachen bzw. ω-reguläre Sprachen charakterisieren.

1. Einleitung

Seit der Einführung von Membransystemen durch Gheorghe Păun (s. [12]) im Jahre 1998 wurden bereits viele Varianten, welche durch Merkmale biologischer Systeme inspiriert wurden, untersucht. Die Membranstruktur selbst und besondere Eigenschaften der Membranen, insbesondere für den Transport von Objekten, sind die wichtigsten Charakteristika, welche in den verschiedensten Modellen von P-Systemen untersucht werden (s. beispielsweise [2], [14]; für einen umfassenden Überblick s. [13], für den aktuellen Forschungsstand s. [18]). Eine Membranstruktur besteht aus Membranen, welche hierarchisch in die äußerste Membran (Hautmembran) eingebettet sind; jede Membran umschließt eine Region (Kompartment), welche auch andere Membranen enthalten kann. In dieser

Membranstruktur, wo die Membranen als Separatoren sowie als Kommunikationskanäle fungieren, können sich Objekte entsprechend gegebener Evolutionsregeln entwickeln. Kürzlich wurden P-Systeme mit reinen Kommunikationsregeln eingeführt (s. [11]) und weiter untersucht (s. [6], [9]); in diesen P-Systemen mit Kommunikationsregeln passieren Objekte die Membranen, ohne selbst von den Regeln verändert zu werden. Passieren Objekte die Membran in derselben Richtung, so spricht man von Symport-Regeln, passieren higegen Objekte die Membran in entgegengesetzten Richtungen, so spricht man von Antiport-Regeln.

Die Verwendung von P-Systemen mit Antiport-Regeln als Berechnungs- und Erzeugungsmechanismen wurde bereits in [6] untersucht; hier betrachten wir nun wie in [7] derartige P-Systeme als Akzeptierungsmechanismen, die eine (endliche bzw. unendliche) Eingabe-Folge von Terminalsymbolen analysieren, wie dies zum ersten Mal in [1] betrachtet wurde. Entsprechend der üblichen Notationen in der Theorie formaler Sprachen nennen wir diese Systeme P-Automaten bzw. ω-P-Automaten und zeigen, dass diese die gleiche Berechnungskapazität wie Turingmaschinen bzw. ω-Turingmaschinen erreichen können.

2. Einführende Definitionen und Resultate

Die Menge der nicht-negativen ganzen Zahlen wird mit $\mathbf{N}_0$ bezeichnet, die Menge der positiven ganzen Zahlen mit $\mathbf{N}$. Ein *Alphabet* V ist eine endliche nicht-leere Menge von abstrakten Symbolen. Das freie Monoid, das von V unter Konkatenation erzeugt wird, wird mit V^* bezeichnet; darüber hinaus definieren wir $V^+ := V^* \setminus \{\lambda\}$, wobei λ das *Leerwort* bezeichnet. Eine Multimenge über V wird als String über V (und jede seiner Permutationen) repräsentiert . Mit $\mid x \mid$ bezeichnen wir die Länge eines Wortes x über V sowie die Anzahl der Elemente in der Multimenge, die durch x repräsentiert wird. Wir betrachten zwei Sprachen $L,\ L'$ über V als gleich, wenn $L \setminus \{\lambda\} = L' \setminus \{\lambda\}$.

Ein *endlicher Automat* (EA) ist ein Quintupel $M = (Q, T, \delta, q_0, F)$, wobei Q eine endliche Menge von *Zuständen,* T das *Eingabealphabet,* $\delta : Q \times T \to 2^Q$ die *Übergangsfunktion,* $q_0 \in Q$ den *Startzustand* und $F \subseteq Q$ *die* Menge von *Endzuständen* bezeichnet. Die Übergangsfunktion δ kann auf natürliche Weise zu einer Funktion $\delta : 2^Q \times T^+ \to 2^Q$ erweitert werden. Die Sprache, welche von einem EA M akzeptiert wird, ist die Menge aller Wörter $w \in T^+$ mit $\delta(\{q_0\}, w) \cap F \neq \emptyset$ (d.h., aller Wörter, welche von M akzeptiert werden). Gilt $card(\delta(q, a)) = 1$ für alle $q \in Q$ und alle $a \in T$, so nennt man M einen *deterministischen endlichen Automaten* (DEA). Die Familie der von (deterministischen) endlichen Automaten erkannten Mengen stimmt mit der Familie der regulären Mengen überein.

Betrachten wir Multimengen von Symbolen, so stellen Registermaschinen ein einfaches universelles Berechnungsmodell dar (die ursprünglichen Definitionen sind beispielsweise in [10] zu finden, die Definitionen, wie wir sie hier verwenden, in [6] und [8]):

Eine *n-Registermaschine* ist ein Konstrukt $M = (n, R, i, h)$, wobei

- n die Anzahl der Register bezeichnet,
- R eine Menge von markierten Befehlen $j : (op(r), k, l)$ ist, wobei $op(r)$ eine Operation auf Register r von M ist und j, k, l Markierungen aus der Menge $Lab(M)$ sind,
- i die Startmarkierung ist und
- h die Endmarkierung ist.

Die Registermaschine M kennt folgende Befehle:

(A(r),k,l) Addiere eins zum Inhalt des Registers r und fahre mit Befehl k oder Befehl l fort; für die deterministischen Varianten, welche normalerweise in der Literatur in Betracht gezogen werden, wird $k = l$ gefordert.

(S(r),k,l) Ist Register r nicht leer, dann subtrahiere eins von seinem Inhalt und fahre mit Befehl k fort, sonst fahre mit Befehl l fort.

HALT Halte die Maschine an. Dieser zusätzliche Befehl kann nur der Endmarkierung h zugeordnet werden.

In der *deterministischen Variante* können solche n-Registermaschinen dazu verwendet werden, partiell rekursive Funktionen $f : \mathbf{N}_0^k \to \mathbf{N}_0^m$ zu berechnen; beginnend mit $(n_1, ..., n_k) \in \mathbf{N}_0$ in den Registern 1 bis k, hat M $f(n) = (r_1, ..., r_m)$ berechnet, wenn sie in der Endmarkierung h hält und Register 1 bis m r_1 bis r_m enthalten. Kann die Endmarkierung nicht erreicht werden, so bleibt $f(n)$ undefiniert.

Eine deterministische n-Registermaschine kann auch eine Eingabe $(n_1, ..., n_k) \in \mathbf{N}_0^k$ in den Registern 1 bis k analysieren, welche akzeptiert wird, wenn die Registermaschine in der Endmarkierung hält und alle Register leer sind. Hält die Maschine nicht, so war die Analyse nicht erfolgreich.

In der *nicht-deterministischen Variante* können n-Registermaschinen jede rekursiv aufzählbare Menge von natürlichen Zahlen (oder von Vektoren von natürlichen Zahlen) berechnen. Beginnend mit leeren Registern bezeichnen wir eine Berechnung der n-Registermaschine als erfolgreich, wenn sie hält und das Ergebnis in den ersten m Registern enthalten ist (und alle übrigen Register leer sind).

Aus Ergebnissen in [4] (welche auf wohlbekannten Ergebnissen aus [10] basieren) können wir Folgendes ableiten:

Proposition 1. *Für jede rekursiv aufzählbare Menge von Vektoren natürlicher Zahlen $L \subseteq \mathbf{N}_0^k$ existiert eine deterministische $(k+2)$-Registermaschine M, welche L erkennt.*

Darüber hinaus kennen wir ein ähnliches Ergebnis für Mengen von Wörtern (siehe auch [8]):

Proposition 2. *Für jede rekursiv aufzählbare Menge von Wörtern L über dem Alphabet T mit $card(T) = z - 1$ existiert eine deterministische 3-Registermaschine M, welche L so erkennt, dass für jedes $w \in T^*$ $w \in L$ genau dann gilt, wenn M hält, nachdem M mit $g_z(w)$ ($g_z(w)$ bezeichnet die z-äre Repräsentation des Wortes w) im ersten Register gestartet wurde.*

Bemerkung 3. *Auf Grund der Resultate in [10] wissen wir, dass die Aktionen einer Turingmaschine durch eine Registermaschine in nur zwei Registern simuliert werden können, indem eine z-äre Repräsentation ($z-1$ ist die Kardinalität des Bandalphabets) der linken und rechten Seite des Turingbandes bezogen auf die aktuelle Position des Lese-/Schreibkopfes der Turingmaschine verwendet wird. Dabei verwendet man eine Primzahlkodierung so, dass alle nötigen Operationen für die Simulation der Turingmaschine von einer Registermaschine in nur zwei Registern simuliert werden können. Hier benötigen wir nun eine etwas weniger komprimierte Repräsentation der Aktionen und Inhalte des Arbeitsbandes der Turingmaschine: Wir speichern nur die Inhalte der linken und rechten Seite des Arbeitsbandes bezogen auf die aktuelle Position des Lese-/Schreibkopfes und simulieren die Aktionen auf dem Arbeitsband in diesen zwei Registern, während der aktuelle Zustand der Turingmaschine in einem separaten zusätzlichen Register unär kodiert gespeichert wird.*

3. P-Automaten mit Antiport-Regeln

Ein *P-Automat mit Antiport-Regeln* (fortan einfach *P-Automat)* ist ein Tupel Π mit

$$\Pi = (V, T, \mu, w_1, ..., w_n, R_1, ..., R_n, F),$$

wobei

1. V ein Alphabet von *Objekten* bezeichnet,
2. $T \subseteq V$ das *Terminalalphabet* ist,
3. μ eine *Membranstruktur* ist,
4. $w_1, ..., w_n$ Multimengen über V sind, welche mit den Regionen $1, ..., n$ von μ assoziiert sind,
5. $R_1, ..., R_n$ endliche Mengen von *Antiport-Regeln* sind, welche den Membranen bzw. den von ihnen umschlossenen Regionen $1, ..., n$ zugeordnet und von der Gestalt $(x, out; y, in)$ mit $x, y \in V^+$ sind (dabei wird die Multimenge x aus der von der Membran umschlossenen Region hinausgeschickt und die Multimenge y aus der umgebenden Region hereingenommen; der *Radius* der Regel $(x, out; y, in)$ ist das Zahlenpaar $(|x|, |y|)$),
6. F eine endliche Menge von *Endzuständen* ist.

Ein *Endzustand* ist eine Funktion f, welche jeder Region eine endliche Multimenge zuordnet (s. [1]); ein spezielles Symbol Λ zeigt an, dass der Inhalt der entsprechenden Region nicht in Betracht gezogen werden muss, während für jede Multimenge $\neq \Lambda$, die einer Membranregion zugeordnet ist, deren Inhalt mit der vorgegebenen endlichen Multimenge übereinstimmen muss; in diesem Fall sagen wir, dass die zugrundeliegende Konfiguration den Endzustand f erreicht hat. Besteht die Membranstruktur μ aus nur einer Membran, so kann ein Endzustand einfach durch die der Hautmembran zugeordnete Multimenge angegeben werden.

Beginnend mit der Startkonfiguration, welche aus μ und $w_1, ..., w_n$ besteht, erfolgt der Übergang von einer Konfiguration in die nächste, indem Regeln von R_i nicht-deterministisch und in maximal paralleler Art und Weise angewendet werden. Eine *Berechnung* ist eine Folge von Transitionen. Üblicherweise wird für eine Multimenge oder ein Wort w über einem Alphabet T eine Berechnung genau dann als *erfolgreich* bezeichnet, wenn sie hält (d.h., wenn keine Regel mehr angewendet werden kann); der Idee der Endzustände folgend wie sie in [1] eingeführt wurde, werden wir hier jedoch eine Berechnung dann und nur dann als erfolgreich bezeichnen, wenn ein Endzustand f aus F erreicht wird. Eine Multimenge oder ein Wort w über T wird vom P-Automaten Π dann und nur dann erkannt, wenn es eine erfolgreiche Berechnung von Π so gibt, dass die (Folge von) Terminalsymbole(n), welche aus der Umgebung genommen wurden, genau w ist. (Falls mehr als ein Terminalsymbol in einem Schritt aus der Umgebung geholt wird, dann stellt jede Permutation dieser Symbole ein gültiges Teilwort eines Eingabewortes dar.)

Das folgende Resultat wurde bereits in [5], allerdings für haltende P-Automaten und nicht wie hier für durch Endzustand akzeptierende P-Automaten mit Antiport-Regeln, bewiesen:

Satz 4. *Sei $L \subseteq \mathbf{\Sigma}^*$ eine rekursiv aufzählbare Sprache. Dann kann L von einem P-Automaten akzeptiert werden, der aus der einfachsten Membranstruktur besteht und nur Regeln der Form $(x, out; y, in)$ mit Radius $(2, 1)$ oder $(1, 2)$ verwendet.*

Beweis. Ausgehend von Proposition 2 zeigen wir, wie wir das Eingabewort w lesen, die Kodierung $g_z(w)$ generieren und dann die Befehle der 3-Registermaschine simulieren können; dabei liegt das Hauptaugenmerk auf der Simulation einer n-Registermaschine:

- Ein Addier-Befehl $j : (A(i), k, l)$ wird durch die Regeln $(j, out; ka_i, in)$ und $(j, out; la_i, in)$ simuliert.
- Ein bedingter Subtrahier-Befehl $j : (S(i), k, l)$ wird durch die Regeln $(ja_i, out; k, in)$ für den Fall, dass ein Subtraktion möglich ist, sowie andernfalls durch die Regeln $(j, out; j'j'', in)$, $(j'a_i, out; \#, in)$, $(j'', out; \hat{j}\hat{j}', in)$, $(\hat{j}\hat{j}', out; \hat{j}'', in)$, $(j'\hat{j}'', out; l, in)$ simuliert. Falls das Register i nicht leer ist, d.h. mindestens ein Symbol a_i vorhanden ist, obwohl man die Regel

$(j, out; j'j'', in)$ angewendet hat, dann garantiert die Bedingung der maximalen Parallelität , dass im nächsten Schritt die Regel $(j'a_i, out; \#, in)$ zugleich mit $(j'', out; \hat{j}\hat{j}', in)$ angewendet wird, was zur Einführung des Fehlersymbols # führt. Nur wenn in der aktuellen Konfiguration kein Symbol a_i in der Hautmembran vorkommt, kann das Objekt j' noch zwei Schritte warten, bis es in der Regel $(j'\hat{j}'', out; l, in)$ zusammen mit dem durch die Regel $(\hat{j}\hat{j}', out; \hat{j}'', in)$ eingeführten Symbol $\hat{j}''$ verwendet wird.

- Der Halte-Befehl $h : HALT$ wird dadurch "simuliert", dass es für das Haltesymbol h keine Regel gibt; da wir davon ausgehen können, dass mit Erreichen des Haltebefehls alle Register der zu simulierenden Registermaschine leer sind, definieren wir daher den Endzustand des zu konstruierenden P-Automaten mit h.

Wir beginnen nun mit q als Axiom und nehmen $(q, out; q_a a, in)$ und $(q_a a, out; q_{a,0}, in)$ für jedes $a \in T$. Nehmen wir nun an, die Kodierung der bis zum aktuellen Zeitpunkt eingelesenen Eingabesequenz v werde durch $g_z(v)$ Symbole A repräsentiert. Ein weiteres Eingabesymbol a wird durch die Kodierung $g_z(va)$ erfasst; wegen $g_z(va) = z * g_z(v) + g_z(a)$ wird dieser Kodierungsschritt durch das folgende Unterprogramm einer Registermaschine bewerkstelligt, welches im Wesentlichen die Multiplikation mit z bewerkstelligt:

$q_{a,0}: \quad (S(1), q_{a,1}, q'_a)$
$q_{a,i}: \quad (A(2), q_{a,i+1}, q_{a,i+1})$ für $1 \leq i < z$
$q_{a,z}: \quad (A(2), q_{a,0}, q_{a,0})$
$q'_a: \quad \left(S(2), q'_{a,1}, q''_{a,1}\right)$
$q'_{a,1}: \quad (A(1), q'_a, q'_a)$

Sei nun $k = g_z(a)$; dann hören wir mit den folgenden Befehlen auf:

$q''_{a,i}: \quad \left(A(1), q''_{a,i+1}, q''_{a,i+1}\right)$ für $1 \leq i \leq k-1$

Die Eingabe des nächsten Terminalsymbols beginnt dann mit $\left(q''_{a,k} a, out; q, in\right)$.

Offensichtlich können die Befehle des obigen Unterprogramms in Antiport-Regeln übersetzt werden, wie bereits zu Beginn des Beweises ausgearbeitet wurde. Die Anzahl der Symbole A bzw. B entspricht dem Inhalt von Register 1 bzw. 2.

Wenn kein weiteres Eingabesymbol mehr eingelesen werden soll, verwenden wir die Antiport-Regeln $(q, out; q'q'', in)$, $(q'q'', out; q_0, in)$, um die Simulation der in Proposition 2 beschriebenen 3-RegisterMaschine zu starten (dabei entspricht q_0 der Startmarkierung der Registermaschine).

Offensichtlich erscheint das Haltesymbol h in der Hautmembran dann und nur dann, wenn die Registermaschine die Eingabe $g_z(w)$ akzeptiert. □

Das Wort, welches erkannt werden soll, ist durch die Folge der Terminalsymbole a gegeben, welche mittels Antiport-Regeln der Gestalt $(q, out; q_a a, in)$ aus der Umgebung geholt wurden. Offensichtlich kann dieses Wort auch als eine Repräsentation der entsprechenden Multimenge (bzw. des entsprechenden Vektors

natürlicher Zahlen) interpretiert werden, was zu einem analogen Ergebnis für rekursiv aufzählbare Multimengen über T (bzw. für die entsprechenden Mengen von Vektoren natürlicher Zahlen) führt.

3.1. Endliche P-Automaten mit Antiport-Regeln

Wenn wir P-Automaten betrachten, die aus der einfachsten Membranstruktur, d.h. nur der Hautmembran, bestehen, und nur Regeln sehr spezieller Form verwenden, erhalten wir eine Charakterisierung der Familie der regulären Sprachen, was im Folgenden gezeigt wird:

Ein *endlicher P-Automat* ist ein P-Automat

$$\Pi = (V, T, [_1]_1, w_1, R_1, F)$$

mit nur einer Membran, wobei

1. das Axiom w_1 ein Nichtterminal ist, d.h., $w_1 \in V \setminus T$ (im Fall endlicher P-Automaten bezeichnen wir ein Element aus $V \setminus T$ einfach auch als *Zustand)*;
2. die Antiport-Regeln in R_1 von der Gestalt $(q, out; pa, in)$ und $(pa, out; r, in)$ mit $a \in T$ und $p, q, r \in V \setminus T$ sind;
3. für jede Regel $(q, out; pa, in)$ in R_1 die einzigen anderen Regeln in R_1, welche p enthalten, von der Gestalt $(pa, out; r, in)$ sind;
4. $F \subseteq V \setminus T$ (jede Multimenge in F besteht aus genau einem Zustand).

Satz 5. *Sei $L \subseteq \mathbf{\Sigma}^+$. Dann ist L genau dann regulär, wenn L von einem endlichen P-Automaten akzeptiert wird.*

Beweis. Sei L eine reguläre Sprache, welche von einem EA $M = (Q, T, \delta, q_0, F)$ akzeptiert wird. Dann wird L auch von dem endlichen P-Automaten $\Pi = (V, T, [_1]_1, q_0, R_1, F)$ akzeptiert, wobei

$$\begin{aligned} V &= Q \cup \{(q, a, r) \mid r \in \delta(q, a) \text{ für } q, r \in Q, a \in T\} \cup T \text{ und} \\ R_1 &= \{(p, out; (q, a, r)\, a, in), ((q, a, r)\, a, out; r, in) \mid p, q, r \in Q, a \in T \text{ und } r \in \delta(q, a)\}. \end{aligned}$$

Jeder Übergang (q, a, r) in M wird in Π durch die Regeln $(p, out; (q, a, r)\, a, in)$ und $((q, a, r)\, a, out; r, in)$ in zwei Schritten simuliert. Dabei enthält die Hautmembran in der intermediären Konfiguration $(q, a, r)\, a$.

Auf der anderen Seite kann nun eine reguläre Sprache L, die von einem endlichen P-Automaten $\Pi = (V, T, [_1]_1, q_0, R_1, F)$ erkannt wird, auch von einem endlichen Automaten $M = (V \setminus T, T, \delta, q_0, F)$ akzeptiert werden, wobei

$$\begin{aligned} \delta = \{((q, a), \{r \in V \setminus T \mid (q, out; pa, in), (pa, out; r, in) \in R_1 \text{ für p} \in V \setminus T\}) \mid \\ q \in V \setminus T, a \in T\}. \end{aligned}$$

Jede Anwendung einer Folge $(q, out; pa, in)$, $(pa, out; r, in)$ von Regeln aus R_1 wird durch nur einen Übergang (q, a, r) in M simuliert; da die intermediären Inhalte pa der Hautmembran nicht als Endzustand fungieren können, muss in einer erfolgreichen Berechnung von M die Anwendung einer Regel $(q, out; pa, in)$

von der Anwendung von $(pa, out; r, in)$ für ein $r \in V \setminus T$ gefolgt werden, womit die Transitionen in δ die Regeln in R_1 korrekt simulieren. □

4. ω-Turingmaschinen

Wir betrachten den Raum X^ω der unendlichen Wörter (ω-Wörter) auf einem endlichen Alphabet X mit $card\,(X) \geq 2$. Sei $w \cdot b$ die Konkatenation von $w \in X^*$ und $b \in X^\omega$, welche sich auf natürliche Weise auf Teilmengen $W \subseteq X^*$ und $B \subseteq X^\omega$ erweitern läßt. Teilmengen von X^ω werden ω-Sprachen genannt. Für ein ω-Wort ξ und jedes $n \in \mathbf{N}$ bezeichnet ξ/n den Präfix von ξ der Länge n.

In den meisten Modellen, die in der Literatur gefunden werden (s. beispielsweise [3] oder [17]), wird die Akzeptanz von ω-Sprachen durch Turingmaschinen vom Verhalten auf dem Eingabeband sowie durch spezifische Endzustandsbedingungen, die von der Akzeptanz von ω-Sprachen durch endliche Automaten bekannt sind, determiniert.

Entsprechend den X-Automaten von Engelfriet und Hoogeboom (s. [3]) betrachten wir ω-Turingmaschinen als Automaten $M = (X, \Gamma, Q, q_0, P)$ mit einem separaten Eingabeband, auf welchem sich der Lesekopf nur nach rechts bewegen kann, einem Arbeitsband, einem Eingabealphabet X, dem Bandalphabet Γ, einer endlichen Menge interner Zustände Q, dem Anfangszustand q_0 und der Relation

$$P \subseteq Q \times X \times \Gamma \times Q \times \{0, +1\} \times \Gamma \times \{-1, 0, +1\},$$

welche die nächste Konfiguration definiert.

Wenn sich M im Zustand $q \in Q$ befindet sowie $x_0 \in X$ auf dem Eingabeband und $x_1 \in \Gamma$ auf dem Arbeitsband liest, so bedeutet $(q, x_0, x_1; p, y_0, y_1, y_2) \in P$, dass M in den Zustand $p \in Q$ übergeht, den Kopf auf dem Eingabeband nach rechts bewegt, wenn $y_0 = +1$ oder den Kopf nicht bewegt, wenn $y_0 = 0$, und für $y_1 \in \Gamma$ und $y_2 \in \{-1, 0, +1\}$ die Maschine y_1 an Stelle von x_1 auf das Arbeitsband schreibt und den Kopf auf dem Band nach links bewegt, wenn $y_2 = -1$, nach rechts, wenn $y_2 = +1$, oder gar nicht bewegt, wenn $y_2 = 0$.

Sei $\xi \in X^\omega$ die Eingabe der ω-Turingmaschine M. Wir nennen eine Folge $z \in Q^\omega$ von Zuständen einen *Lauf* von M auf ξ, wenn die Folge von Zuständen, welche die Turingmaschine in ihrer Berechnung mit Eingabe ξ durchläuft, z ist. Wir sagen, dass eine Eingabefolge $\xi \in X^\omega$ von M gemäß Bedingung C akzeptiert wird, wenn es einen derartigen Lauf z von M auf ξ gibt, dass z Bedingung C erfüllt. In Anlehnung an die Notationen von Engelfriet und Hoogeboom betrachten wir folgende Bedingungen: Sei $\alpha : Q^\omega \to 2^Q$ eine Abbildung, die jedem ω-Wort $\zeta \in Q^\omega$ eine Teilmenge $Q' \subseteq Q$ zuordnet, und sei $R \subseteq 2^Q \times 2^Q$ eine Relation zwischen Teilmengen von Q. Wir sagen, dass ein Paar (M, Y) für $Y \subseteq 2^Q$ ein ω-Wort $\xi \in X^\omega$ genau dann akzeptiert, wenn

$$\exists Q' \exists z \,(Q' \in Y \wedge z \text{ ist ein Lauf von } M \text{ auf } \xi \wedge (\alpha\,(z)\,, Q') \in R)\,.$$

Besteht Y nur aus einer einzigen Teilmenge von Q, d.h., $Y = \{F\}$ für ein $F \subseteq Q$, so schreiben wir üblicherweise nur (M, F) an Stelle von $(M, \{F\})$.

Für ein ω-Wort $z \in Q^\omega$ sei nun

$$ran\,(z) := \{v : v \in Q \wedge \exists i\,(i \in N \wedge z\,(i) = v)\}$$

der Wertebereich von z und

$$inf\,(z) := \left\{v : v \in Q \wedge z^{-1}\,(v) \text{ ist unendlich}\right\}.$$

Als Relationen R verwenden wir $=$, $\subseteq$ und $\sqcap$, wobei $Z' \sqcap Z'' :\Leftrightarrow Z' \cap Z'' \neq \emptyset$.

Daraus ergeben sich folgende sechs Akzeptierungsarten:

(α, R)	*Akzeptierungsart*	*bedeutet*
$(ran, \sqcap)$	1-Akzeptanz	mindestens einmal
$(ran, \subseteq)$	$1'$-Akzeptanz	überall
$(ran, =)$		
$(inf, \sqcap)$	2-Akzeptanz	unendlich oft
$(inf, \subseteq)$	$2'$-Akzeptanz	fast überall
$(inf, =)$	3-Akzeptanz	

Proposition 6 (s. beispielsweise [17])**.** Für alle $\alpha \in \{ran, inf\}$ und alle $R \in \{\subseteq, \sqcap, =\}$ stimmt die Klasse von ω-Sprachen, die im (α, R)-Modus von nicht-deterministischen ω-Turingmaschinen akzeptiert wird, mit der Klasse von Σ_1^1-definierbaren ω-Sprachen überein.

Eine ω-Sprache F wird als Σ_1^1-definierbar bezeichnet, falls

$$F = \{\xi : \exists \eta\,(\eta \in X^\omega \wedge \forall n \exists m\,((n, \eta/m, \xi/m) \in M_F))\}$$

für eine rekursive Relation $M_F \subseteq N \times X^* \times X^*$.

4.1. Endliche ω-Automaten

Eine *reguläre ω-Sprache* ist eine endliche Vereinigung von ω-Sprachen der Gestalt UV^ω, wobei U und V reguläre Sprachen sind.

Ein *endlicher ω-Automat* ist eine ω-Turingmaschine, die nur das Eingabeband verwendet. Die Klasse von ω-Sprachen, die von deterministischen endlichen ω-Automaten im $(inf, =)$-Modus akzeptiert wird (3-Akzeptanz), stimmt mit der Klasse der ω-regulären Sprachen überein, ebenso mit der Klasse jener ω-Sprachen, die von nicht-deterministischen endlichen ω-Automaten im $(inf, \sqcap)$-Modus akzeptiert werden (2-Akzeptanz). Ein derartiger (nicht-deterministischer) endlicher ω-Automat kann als (nicht-deterministischer) endlicher Automat $M = (Q, T, \delta, q_0, F)$ beschrieben werden: Für ein ω-Wort $\xi \in T_M^\omega$, $\xi = a_1 a_2 ...$, $a_i \in T$ für alle $i \geq 1$, wird ein Lauf von M auf ξ durch eine unendliche Folge $s \in Q^\omega$ mit $s = q_0 q_1 q_2 ...$ und $q_i \in \delta\,(q_{i-1}, a_i)$ für alle $i \geq 1$ gekennzeichnet; der Lauf s wird

im Sinne der 2-Akzeptanz als erfolgreich bezeichnet, wenn $inf\,(s) \cap F \neq \emptyset$. Die von M akzeptierte ω-Sprache ist die Menge aller $\xi \in T^\omega$, die einen erfolgreichen Lauf von M auf ξ erlauben.

5. ω-P-Automaten mit Antiport-Regeln

Im Fall von ω-Wörtern müssen wir nicht nur die (jetzt unendliche) Folge von Terminalsymbolen, die von der Umgebung geholt werden, beachten, sondern auch die Akzeptanzbedingung, welche über die Endzustände definiert ist.

Ein ω-P-Automat ist ein Konstrukt

$$\Pi = (V, T, \mu, w_1, ..., w_n, R_1, ..., R_n, Y)\,,$$

das wie ein P-Automat definiert ist, allerdings jetzt zur Analyse unendlicher Folgen von Terminalsymbolen und deren Akzeptanz in Übereinstimmung mit der gegebenen Akzeptanzbedingung zusammen mit einer Menge Y, $Y \subseteq 2^Q$, verwendet wird. Jede Menge $F \in Y$ stellt eine endliche Menge von Endzustnden dar.

Der Beweis des folgenden Hauptergebnisses basiert auf den Beobachtungen in Bemerkung 3:

Satz 7. *Sei $L \subseteq \mathbf{\Sigma}^\omega$ eine ω-Sprache, die von einer ω-Turingmaschine im Akzeptanzmodus (α, R), $\alpha \in \{ran, inf\}$, $R \in \{\subseteq, \sqcap, =\}$, akzeptiert wird. Dann können wir einen ω-P-Automaten mit zwei Membranen konstruieren, der die Aktionen einer Turingmaschine simuliert, L im selben Akzeptanzmodus (α, R) akzeptiert und nur Regeln der Gestalt $(x, out; y, in)$ mit Radius $(2, 1)$ bzw. $(1, 2)$, verwendet.*

Beweis. Sei $L \subseteq \mathbf{\Sigma}^\omega$ eine ω-Sprache, die von einer ω-Turingmaschine

$$M = (\Sigma, \Gamma, Q, q_0, P)$$

zusammen mit $Y \subseteq 2^Q$ im Akzeptanzmodus (α, R) akzeptiert wird.

Wir konstruieren nun einen ω-P-Automaten

$$\Pi = (V, \Sigma, [_1[_2]_2]_1, q_0', fg, R_1, R_2, Y')$$

mit zwei Membranen und Antiport-Regeln der Form $(x, out; y, in)$ mit Radius $(2, 1)$ bzw. $(1, 2)$, der die Aktionen der Turingmaschine simuliert und L im selben Akzeptanzmodus (α, R) akzeptiert. Dabei simulieren wir eine Registermaschine R_M mit nur drei Registern, welche ihrerseits wiederum die Aktionen von M dadurch simuliert, dass eine z-äre Repräsentation der linken und rechten Seite des Turingbandes in zwei Registern simuliert wird (wobei $z-1$ die Kardinalität des Bandalphabetes ist) und der aktuelle Zustand von M in einem separaten zusätzlichen Register unär kodiert gespeichert wird. Die komplexeren Schritte von M können von Π in mehreren Unterschritten simuliert werden:

Das Einlesen eines neuen Symbols $b \in \Sigma$ auf dem Eingabeband von M kann durch Regeln $(q, out; (q, b, r)\, b, in)$ und $((q, b, r)\, b, out; r, in)$, $q, r \in V \setminus T$, im ω–P-Automaten Π simuliert werden. Das Ändern der linken und rechten Seite des

Turingbandes sowie der Wechsel des aktuellen Zustands von M wird durch die Aktionen von R_M simuliert. Die Aktionen der Simulation der Registermaschine R_M selbst können durch Regeln folgenden Typs in Region 1 (d.h., durch Regeln in R_1) simuliert werden, wobei der Inhalt von Register i durch die entsprechende Anzahl von Symbolen a_i in Region 1 von Π repräsentiert wird:

- Ein Addier-Befehl $j : (A(i), k, l)$ wird durch die Regeln $(j, out; ka_i, in)$ und $(j, out; la_i, in)$ simuliert.

- Ein bedingter Subtrahier-Befehl $j : (S(i), k, l)$ wird durch die Regeln $(ja_i, out; k, in)$ sowie $(j, out; j'j'', in)$, $(j'a_i, out; \#, in)$, $(j'', out; \hat{j}\hat{j}', in)$, $(\hat{j}\hat{j}', out; \hat{j}'', in)$, $(j'\hat{j}'', out; l, in)$ simuliert.

- Nachdem wir nur unendliche Berechnungen betrachten, sollte der Haltebefehl von R_M nie erreicht werden.

Wann immer der ω-P-Automat Π einen Schritt der ω-Turingmaschine M simuliert hat, muss Π vor der Fortsetzung der Simulation des Registermaschinenbefehls mit Markierung r eine intermediäre Prozedur starten, um kurzzeitig ein Objekt $[n]$ zu erzeugen, welches in Region 2 den aktuellen Zustand von M repräsentiert. Diese intermediäre Prozedur startet mit der Markierung $\bar{r}$ an Stelle der Markierung r - diese erscheint nun erst am Ende dieser intermediären Prozedur, für die wir die folgenden Regeln in den Regionen 1 und 2 benötigen:

Regeln in R_1	Regeln in R_2
$\left(\bar{r}, out; \bar{r}^{(0)}\bar{r}', in\right)$	
$\left(\bar{r}^{(i-1)}a_3, out; \bar{r}^{(i)}, in\right),\ 1 \leq i \leq m$	
$\left(\bar{r}^{(n)}, out; \bar{h}\,[n]\,, in\right),\ 1 \leq n \leq m$	
	$(fg, out; [n]\,, in),\ 1 \leq n \leq m$
$\left(g\bar{h}, out; \bar{h}', in\right), (fa_3, out; \#, in)$	
$\left(\bar{h}', out; h\bar{h}'', in\right)$	
	$([n]\,, out; fh, in),\ 1 \leq n \leq m$
$\left(\bar{r}'\,[n]\,, out; \tilde{r}^{(n)}, in\right)$	
$\left(\tilde{r}^{(i)}, out; \tilde{r}^{(i-1)}a_3, in\right),\ 1 \leq i \leq m$	
$\left(\tilde{r}^{(0)}\bar{h}'', out; \tilde{r}, in\right)$	
$\left(\tilde{r}, out; \tilde{r}'\tilde{h}'', in\right)$	
$\left(\tilde{h}'', out; \tilde{h}\tilde{h}', in\right)$	
	$\left(h, out; \tilde{h}\tilde{h}', in\right)$
$(\tilde{r}'h, out; \tilde{r}'', in)$	
$(\tilde{r}'', out; \bar{r}''g, in)$	
	$\left(\tilde{h}\tilde{h}', out; g, in\right)$
$\left(\bar{r}''\tilde{h}', out; \overline{\overline{r}}, in\right)$	
$\left(\overline{\overline{r}}\tilde{h}, out; r, in\right)$	

Abschließend müssen wir noch aus einer gegebenen Menge $F \in Y$ von Endzuständen für M die entsprechende Menge F' von Endzuständen für den ω-P Automaten Π konstruieren: Im ω-P-Automaten Π findet man in Region 2 nur die Multimengen fg, fh, $f\tilde{h}\tilde{h}'$ sowie die Symbole $[n]$, welche die Zustände von M repräsentieren. Für die Akzeptanzmodi $(ran, \sqcap)$ und $(inf, \sqcap)$ können wir daher einfach $F' = \{(\Lambda, l) \mid l \in F\}$ nehmen; für die anderen Akzeptierungsarten müssen wir auch die "Konstanten" fg, fh, und $f\tilde{h}\tilde{h}'$, beachten, d.h., wir müssen $F' = \left\{(\Lambda, l) \mid l \in F \cup \left\{fg, fh, f\tilde{h}\tilde{h}'\right\}\right\}$ nehmen. □

5.1. Endliche ω-P Automaten

Ein endlicher *ω-P-Automat* ist ein ω-P-Automat $\Pi = (V, T, [_1]_1, w_1, R_1, F)$ mit nur einer Membran, wobei

1. das Axiom w_1 ein Nichtterminal ist, d.h., $w_1 \in V \setminus T$;
2. die Antiport-Regeln in R_1 von der Gestalt $(q, out; pa, in)$ und $(pa, out; r, in)$ mit $a \in T$ und $p, q, r \in V \setminus T$ sind;

3. für jede Regel $(q, out; pa, in)$ in R_1 die einzigen anderen Regeln in R_1, welche p enthalten, von der Gestalt $(pa, out; r, in)$ sind;
4. $F \subseteq V \setminus T$.

Satz 8. *Sei $L \subseteq \Sigma^\omega$ eine ω-Sprache. Dann ist L genau dann ω-regulär, wenn L von einem endlichen ω-P-Automaten im 2-Akzeptanzmodus akzeptiert wird.*
Beweis. Wir verwenden die gleiche Konstruktion wie in Satz 5; interpretieren wir nun den dort betrachteten endlichen Automaten als endlichen ω-Automaten und den dort konstruierten endlichen P-Automaten als endlichen ω-P-Automaten, so erhalten wir sofort das gewünschte Ergebnis. □

Literatur

[1] Csuhaj-Varjú, E., Vaszil, G.: P automata. In: [15], 177–192

[2] Dassow, J., Păun, Gh.: On the power of membrane computing. *Journal of Universal Computer Science* **5**, 2 (1999) 33–49 (`http://www.iicm.edu/jucs`)

[3] Engelfriet, J., Hoogeboom, H.J.: X-automata on ω-words. Theoret. Comput. Sci. 110,1 (1993) 1–51

[4] Freund, R., Oswald, M.: Generalized P systems with forbidding context. *Fundamenta Informaticae* **49**, 1-3 (2002) 81–102

[5] Freund, R., Oswald, M.: A short note on analysing P systems with antiport rules. *Bulletin EATCS*, **78** (2002) 231–236

[6] Freund, R., Oswald, M.: P systems with activated/prohibited membrane channels. In: Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C. (Eds.): *Membrane Computing 2002.* Lecture Notes in Computer Science, Springer, Berlin (2002)

[7] Freund, R., Oswald, M., Staiger, L.: ω-P automata with communication rules. In: Alhazov, A., Martín-Vide, C., Păun, Gh. (Eds.): *Proceedings of Workshop on Membrane Computing (WMC-2003),* Tarragona (2003) 252–265

[8] Freund, R., Păun, Gh.: On the number of non-terminals in graph-controlled, programmed, and matrix grammars. In: Margenstern, M., Rogozhin, Y. (Eds.): Proc. Conf. Universal Machines and Computations, Chişinău (2001)

[9] Frisco, P., Hoogeboom, H. J.: Simulating counter automata by P systems with symport/antiport. In [15], 237–248

[10] Minsky, M. L.: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, New Jersey (1967)

[11] Păun, A., Păun, Gh.: The power of communication: P systems with symport/antiport. *New Generation Computing* **20,** 3 (2002) 295–306

[12] Păun, Gh.: Computing with membranes. *Journal of Computer and System Sciences* **61**, 1 (2000) 108–143

[13] Păun, Gh.: *Membrane Computing - An Introduction.* Springer-Verlag, Berlin (2002)

[14] Păun, Gh., Rozenberg, G., Salomaa, A.: Membrane computing with external output. *Fundamenta Informaticae* **41** (3) (2000) 259–266, and TUCS Research Report No. 218, 1998 (http://www.tucs.fi)

[15] Păun, Gh., Zandron, C. (Eds.): *Pre-Proceedings of Workshop on Membrane Computing (WMC-CdeA2002)*, Curtea de Argeş, Romania (2002)

[16] Rozenberg, G., Salomaa, A. (Eds.): *Handbook of Formal Languages.* Springer-Verlag, Berlin (1997)

[17] Staiger, L.: ω-languages. In: [16], Vol. 3, 339–387

[18] The P Systems Web Page: `http://psystems.disco.unimib.it`

ÜBER MEHRDIMENSIONALE REBOUND-AUTOMATEN

Jens Glöckler
Institut für Informatik
Universität Giessen
Arndtstr. 2, D-35392 Giessen
e-mail: Jens.Gloeckler@math.uni-giessen.de

KURZFASSUNG

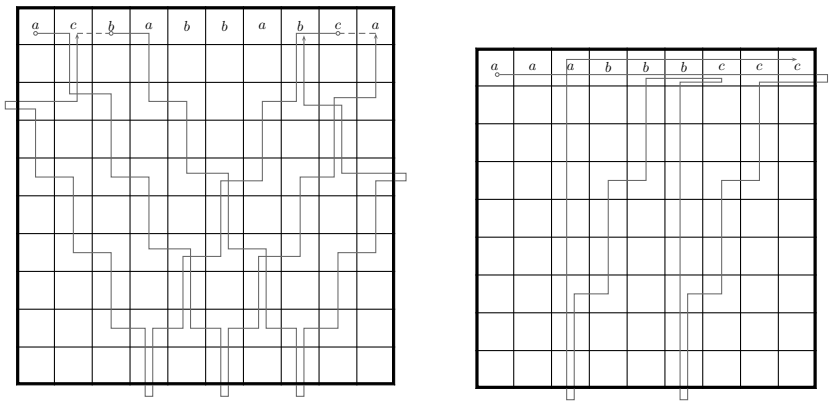
Wir betrachten mehrdimensionale Rebound-Automaten und vergleichen sie mit Mehrkopfautomaten sowie Automaten mit beschränkten Zählern. Die dichte Hierarchie für Mehrkopfautomaten (bzgl. der Anzahl der Leseköpfe) impliziert eine ebensolche Hierarchie für die Dimension der Rebound-Automaten. Weiterhin betrachten wir die Möglichkeit, auf das Abprallen des Lesekopfes an einigen der „Wände“ seines Eingabebandes zu verzichten.

Rebound-Automaten wurden 1977 von Morita, Sugata und Umeo eingeführt (s. [3, 4]) und z.B. in [1, 5, 6, 7, 8] aufgegriffen. Ein (zweidimensionaler) Rebound-Automat ist ein endlicher Automat, der sich (nur lesend) auf einem quadratischen Band bewegt, welches in der ersten Zeile das zu verarbeitende Eingabewort enthält und sonst mit Leerfeldern beschriftet ist:

a	*b*	*a*	*b*	*a*	*b*	*a*	*b*

Ein k-dimensionaler Rebound-Automat besitzt entsprechend ein Eingabeband in der Form eines k-dimensionalen Hyperwürfels, das an einer „Kante“ mit dem (eindimensionalen) Eingabewort beschriftet ist.

Rebound-Automaten können kontextfreie Sprachen wie $\{ww^R \mid w \in A^*\}$ (für ein Alphabet A) bzw. sogar kontextsensitive Sprachen wie $\{ww \mid w \in A^*\}$ oder $\{a^m b^m c^m \mid m \in \mathbb{N}\}$ erkennen, indem sie mit Hilfe des Abprallens an den Wänden Informationen über die Gesamtlänge der Eingabe ausnutzen:



(a) Erkennen von ww^R (b) Erkennen von $a^m b^m c^m$

Andererseits gibt es jedoch auch schon kontextfreie Sprachen, die von keinem (auch mehrdimensionalen) Rebound-Automaten erkannt werden können (s. [1]). Im Falle von Alphabeten mit zwei oder mehr Elementen sind Rebound-Automaten damit deutlich weniger mächtig als Mehrkopfautomaten, im unären Fall sind die beiden Modelle jedoch äquivalent; eine weitere Dimension bietet dieselben Möglichkeiten wie ein zusätzlicher Lesekopf. Mit Hilfe der dichten Hierarchie für unäre Rebound-Automaten aus [2] lassen sich dadurch dichte Hierarchien für Rebound-Automaten (unär und nicht-unär) ableiten und somit die Ergebnis aus [3] verbessern.

Eine weitere Erweiterung endlicher Automaten ist die der durch die Länge der Eingabe beschränkten Zähler. Auch diese Zähler erweisen sich im unären Fall als äquivalent zu den Dimensionen der Rebound-Automaten und sind im nicht-unären Fall mächtiger. Da sie ihrerseits durch die Köpfe eines Mehrkopfautomaten simuliert werden können, reihen sie sich mit ihren Fähigkeiten zwischen Dimension und Lesekopf ein.

Ein anderer Ansatzpunkt zur Untersuchung der Rebound-Automaten ist die Betrachtung der Bewegungen des Lesekopfes und des dabei auftretenden Anstoßens am Rand des Eingabebandes. Formal besteht das Anstoßen des Lesekopfes im Annehmen eines *Alternativzustands* beim Übergang von einer Konfiguration zur Nachfolgekonfiguration. Durch das Annehmen dieses Zustands merkt der Automat, daß die Grenze erreicht wurde; diese Tatsache ist die Grundlage für die Fähigkeiten des Automaten. Es stellt sich jedoch heraus, daß ein Rebound-Automat nicht alle seiner Wände benötigt, um seine vollen Fähigkeiten auszuschöpfen.

Im zweidimensionalen Fall kann auf die Berührung einer der vier Wände verzichtet werden, bei Hinzunahme weiterer Dimensionen kann für jede Dimension die nicht an die Eingabe grenzende Wand ausgespart werden.

Literatur

[1] Inoue, K., Takanami, I. und Taniguchi, H. A note on rebound automata. *Information Sciences 26* (1982), 87–93.

[2] Monien, B. Two-way multihead automata over a one-letter alphabet. *RAIRO – Theoretical Informatics and Applications 14* (1980), 67–82.

[3] Morita, K., Sugata, K. und Umeo, H. Computation complexity of n-bounded counter automaton and multidimensional rebound automaton. *Systems, Computers, Controls 8* (1977), 80–87. Übersetzung von [4].

[4] Morita, K., Sugata, K. und Umeo, H. Computation complexity of n-bounded counter automaton and multidimensional rebound automaton. *Denshi Tsushin Gakkai Ronbunshi 60-D* (1977), 283–290.

[5] Petersen, H. Fooling rebound automata. In *Proc. MFCS 1999* (1999), Band 1672 der *LNCS*, Springer-Verlag, S. 241–250.

[6] Petersen, H. Separation results for rebound automata. In *Proc. MFCS 2000* (2000), Band 1893 der *LNCS*, Springer-Verlag, S. 589–598.

[7] Sakamoto, M., Inoue, K. und Takanami, I. A two-way nondeterministic one-counter language not accepted by nondeterministic rebound automata. *The Transactions of the IEICE E 73* (1990), 879–881.

[8] Sugata, K., Umeo, H. und Morita, K. The language accepted by a rebound automaton and its computing ability. *Electronics and Communications in Japan 60-A* (1977), 11–18.

A COMMON ALGEBRAIC CHARACTERIZATION OF PROBABILISTIC AND QUANTUM COMPUTATIONS

Martin Beadury
Département de mathématiques et d'informatique
Université de Sherbrooke, 2500, boul. Université
Sherbrooke (Québec), J1K 2R1 Canada
e-mail: beaudry@dmi.usherb.ca

José M. Fernandez
Département d'I.R.O., Université de Montréal
C.P. 6128 succ. Centre-Ville
Montréal, (Québec), H3C 3J7 Canada
e-mail: fernandez@iro.umontreal.ca

and

Markus Holzer
Institut für Informatik, Technische Universität München
Boltzmannstraße 3, D-85748 Garching bei München, Germany
e-mail: holzer@in.tum.de

ABSTRACT

Through the study of gate arrays we develop a unified framework to deal with probabilistic and quantum computations, where the former is shown to be a natural special case of the latter. On this basis we define show how to encode a probabilistic or quantum gate array into a sum-free tensor formula which satisfies the conditions of the partial trace problem, and *vice-versa*. This allows us to define a meaningful computational problem on tensor formula, called the *partial trace tensor formula problem*, which is fundamental to our studies, which allows us to capture important complexity classes like, PP, pr-BPP (promise BPP), pr-BQP (promise BQP), and many others, as the underlying algebraic structure varies.

Keywords: Quantum computing, probabilistic computing, tensor algebra, complexity theory.

The "algebraic" approach in the theory of computational complexity consists in characterizing complexity classes within unified frameworks built around a computational model or problem involving an algebraic structure (usually finite

or finitely generated) as the main parameter. In this way, various complexity classes are seen to share the same definition, up to the choice of the underlying algebra. Successful examples of this approach include the description of NC^1 and its subclasses AC^0 and ACC^0 in terms of polynomial-size programs over finite monoids [1], and analogous results for PSPACE, the polynomial hierarchy and the polytime mod-counting classes, through the use of polytime leaf languages [5]. A more recent example is the complexity of problems whose input is a tensor formula, i.e., a fully parenthesized expression where the inputs are matrices (given in full) over some finitely generated algebra and the allowed operations are matrix addition, multiplication, and tensor product, also known as outer, or direct, or Kronecker product. Evaluating tensor formulas with explicit tensor entries is shown by Damm *et. al* [3] to be complete for $\oplus$P, for NP, and for #P as the semiring varies. Recently also other common sense computational problems on tensor formulas and tensor circuits were analyzed by Beaudry and Holzer [2]. Tensor formulas are a compact way of specifying very large matrices. As such, they immediately find a potential application in the description of the behavior of circuits, be they classical Boolean, arithmetic (tensor formulas over the appropriate semiring) or quantum (formulas over the complex field, or an adequately chosen sub-semiring thereof).

We formalize and confirm this intuition, that basic tensor calculus not only captures natural complexity classes in simple ways, but it yields a simpler and unified view on classical probabilistic and modern quantum computation, which gives probabilistic and quantum computations the exact same definition, up to the underlying algebra. Apart from offering a first application of the algebraic approach to quantum computing, our paper thus reasserts the point made by Fortnow [4], that for the classes BPP and BQP, the jump from classical to quantum polynomial-time computation consists in allowing negative matrix entries for the evolution operators, which means that different computations done in parallel may interfere destructively. Based on this unified framework, we define a meaningful computational problem on tensor formula, called the *partial trace tensor formula problem*, which is fundamental to our studies, which allows us to capture important complexity classes; that is given a tensor formula F of order $n \times 1$ over a semiring $\mathcal{S}$ plus a positive integer k, deciding whether the kth partial trace of the matrix $\mathrm{val}_{\mathcal{S}}^{n,n}(F \cdot F^{\mathsf{T}})$ fulfills a certain property. Our precise characterizations are as follows:

- We present probabilistic computation as a natural special case of quantum computation using the unified framework on gate arrays, instead of presenting quantum as a more or less artificial extension of probabilistic computation.
- The partial trace *sum-free* tensor formula problem enables us to capture the significant complexity classes (pr-)P (promise P), NP, pr-BPP (promise BPP), and PP and some of their quantum counterparts pr-EQP (promise

EQP), NQP, and pr-BQP (promise BQP), by showing completeness results of the problem under consideration.

- By giving up sum-freeness, we obtain completeness statements for further complexity classes like $\oplus$P, NP, $C_=$P, its complement co-$C_=$P, Valiant's class pr-UP (promise UP), its generalization pr-SPP (promise SPP), and unique polytime US.

Since some of these classes are "semantic" classes, i.e., the underlying machine must obey a property for all inputs, which is not obvious to check, or even undecidable. An example would be UP, since for a nondeterministic machine to define a language in UP, it must have the property that for all inputs either exactly one accepting path exists or none. Therefore, the obtained completeness results are subject to a certain promise.

References

[1] D. Mix Barrington and D. Thérien. Finite monoids and the fine structure of NC^1. *Journal of the ACM*, 35(4):941–952, October 1988.

[2] M. Beaudry and M. Holzer. The complexity of tensor circuit evaluation. In J. Sgall, A. Pultr, and P. Kolman, editors, *Proceedings of the 26th Conference on Mathematical Foundations of Computer Science*, number 2136 in LNCS, pages 173–185, Mariánské Lázně, Czech Republic, August 2001. Springer.

[3] C. Damm, M. Holzer, and P. McKenzie. The complexity of tensor calculus. *Computational Complexity*, 2003. Accepted for publication.

[4] L. Fortnow. One complexity theorist's view of quantum computing. *Theoretical Computer Science*, 292(3):597–610, 2003.

[5] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings of the 8th Annual Structure in Complexity Theory Conference*, pages 200–207, San Diego, California, May 1993. IEEE Computer Society Press.

DISTANCE AUTOMATA AND THE STAR HEIGHT PROBLEM

Daniel Kirsten[1]
LIAFA
Université Denis Diderot – Case 7014
2, place Jussieu
F-75251 Paris Cedex 05, France
e-mail: kirsten@liafa.jussieu.fr

ABSTRACT

Distance automata were introduced by K. Hashiguchi motivated by his research on the star height hierarchy in 1982 [2, 3]. Distance automata became a fruitful concept in theoretical computer science with many applications beyond their impact for the decidability of the star height hierarchy [4], e.g., they have been of crucial importance in the research on the star problem in trace monoids [6, 10], but they are also of interest in industrial applications as speech processing [11]. Consequently, distance automata and related concepts have been studied by many researchers beside K. Hashiguchi, e.g., [5, 7, 8, 9, 12, 13].

Let Σ be a finite alphabet. A *distance automaton* is a tuple $\mathcal{A} = [Q, E, I, F, \Delta]$, where

1. $[Q, E, I, F]$ is an automaton (i.e., Q is a finite set, $E \subseteq Q \times \Sigma \times Q$, and $I, F \subseteq Q$ are initial resp. accepting states) and
2. $\Delta : E \to \{0, 1\}$ is a mapping called *distance function.*

Let $\mathcal{A} = [Q, E, I, F, \Delta]$ be a distance automaton. We extend Δ to paths: we define the distance of a path π as the sum of the distances of all transitions in π and denote it by $\Delta(\pi)$. We define the distance of a word $w \in \Sigma^*$ as the minimum over the distances of all successful paths which are labeled with w. Under the convention $\min \emptyset = \infty$, $\mathcal{A}$ computes a mapping $\Delta : \Sigma^* \to \mathbb{N} \cup \infty$.

We define $L(\mathcal{A})$ as the language all words $w \in \Sigma^*$ for which $\Delta(w) \neq \infty$. This is exactly the language of the automaton $[Q, E, I, F]$.

Distance automata are more involved than classic automata, e.g., there are nondeterministic distance automata which do not admit a deterministic equivalent, and it is undecidable whether two distance automata compute the same mapping [7].

A distance automaton is *limited* if there is a bound $d \in \mathbb{N}$ such that $\Delta(w) \leq d$ for every $w \in L(\mathcal{A})$. The *limitedness problem for distance automata* is the question for an algorithm which decides whether a distance automaton is limited or not. It was already raised by C. Choffrut in the framework of recognizable power series over the tropical semiring in 1979 [1].

In 1982, K. Hashiguchi showed that the limitedness problem for distance automata is decidable. In 1987, H. Leung showed that it is PSPACE-hard [8], and very recently, H. Leung and V. Podolskiy showed that it is PSPACE-complete [9].

In the talk we introduce desert automata by associating another semantics to distance automata. Formally, desert automata are defined exactly as distance automata. Let $\mathcal{A} = [Q, E, I, F, \Delta]$ be a desert automaton. We call a transition $e \in E$ a *water transition* if $\Delta(e) = 1$.

Let π and π' paths in $\mathcal{A}$. We call π' a *subpath* of π if there are (possibly empty) paths π_1 and π_2 in $\mathcal{A}$ such that $\pi_1\pi'\pi_2 = \pi$.

Imagine that you plan to walk through a desert for a few weeks. You carry a water tank which is initially full but it not last the entire way. However, you visit several places during your journey where you can find some water and refill the tank. Clearly, the required capacity of the tank is determined by the maximal distance between two consecutive water places. If you have the choice between several paths, then you intend to carry a rather small but sufficient tank, and thus, you choose a path for which the required capacity of the tank is minimal.

Thus, we define the distance of a path π as the length of a longest subpath of π which does not contain any water transition, and denote it by $\Delta'(\pi)$. We define for every word $w \in \Sigma^*$ $\Delta'(w)$ as the minimum of the values $\Delta'(\pi)$ for all successful paths π which are labeled by w.

A desert automaton is *limited* if there is a bound $d \in \mathbb{N}$ such that $\Delta'(w) \leq d$ for every $w \in L(\mathcal{A})$. The *limitedness problem for desert automata* is the question for an algorithm which decides whether a desert automaton is limited or not.

We explain by an example that the limitedness problem for desert automata cannot be decided by a trivial pumping argument.

We define a suitable notion of transformation matrices of desert automata, and we reduce the limitedness problem for desert automata to a Burnside type problem for the semigroup of transformation matrices. We solve this Burnside type problem by applying results from finite semigroup theory and developing techniques from I. Simon's and H. Leung's approaches to the limitedness problem for distance automata.

Hence, we can show that the limitedness problem for desert automata is decidable in time and space complexity $2^{\mathcal{O}(n^2)}$ where n is the number of states.

Moreover, we show that this problem is PSPACE-hard.

Finally, we will discuss some further developments and important consequences for the star height problem.

References

[1] C. Choffrut. Series rationelles d'image finie. Technical Report 79–6, Laboratoire d'Informatique Théorique et Programmation, Paris, 1979.

[2] K. Hashiguchi. Limitedness theorem on finite automata with distance functions. *Journal of Computer and System Sciences*, 24:233–244, 1982.

[3] K. Hashiguchi. Regular languages of star heigth one. *Information and Control*, 53:199–210, 1982.

[4] K. Hashiguchi. Algorithms for determining relative star heigth and star heigth. *Information and Computation*, 78:124–169, 1988.

[5] K. Hashiguchi. New upper bounds to the limitedness of distance automata. *Theoretical Computer Science*, 233:19–32, 2000.

[6] D. Kirsten and J. Marcinkowski. Two techniques in the area of the star problem in trace monoids. *Theoretical Computer Science*, to appear in 2003.

[7] D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 4(3):405–425, 1994.

[8] H. Leung. *An Algebraic Method for Solving Decision Problems in Finite Automata Theory.* PhD thesis, Pennsylvania State University, Department of Computer Science, 1987.

[9] H. Leung and V. Podolskiy. The limitedness problem on distance automata: Hashiguchi's method revisited. *Theoretical Computer Science*, to appear in 2003.

[10] Y. Métivier and G. Richomme. New results on the star problem in trace monoids. *Information and Computation*, 119(2):240–251, 1995.

[11] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:269–311, 1997.

[12] I. Simon. On semigroups of matrices over the tropical semiring. *Informatique Théorique et Applications*, 28:277–294, 1994.

[13] A. Weber. Distance automata having large finite distance or finite ambiguity. *Mathematical Systems Theory*, 26:169–185, 1993.

ON THE NP-COMPLETENESS OF THE NURIKABE PUZZLE AND VARIANTS THEREOF

MARKUS HOLZER
Institut für Informatik, Technische Universität München,
Boltzmannstraße 3, D-85748 Garching bei München, Germany
e-mail: holzer@informatik.tu-muenchen.de

ANDREAS KLEIN
Fachbereich für Mathematik und Informatik, Universität Kassel,
Heinrich Plett Straße 40, D-34132 Kassel, Germany
e-mail: klein@mathematik.uni-kassel.de

and

MARTIN KUTRIB
Institut für Informatik, Universität Gießen,
Arndtstraße 2, D-35392 Gießen, Germany
e-mail: kutrib@informatik.uni-giessen.de

ABSTRACT

We show that the popular pencil puzzle NURIKABE is intractible from the computational complexity point of view, i.e., is NP-complete. To this end we show how to simulate Boolean gates by the puzzle under consideration. Moreover, we study some variant of NURIKABE, which remains NP-complete, too.

Keywords: Pencil puzzle game, complexity theory, intractability, completeness.

NURIKABE (engl. coating wall) is a solitaire puzzle, which was invented by *Nikoli & Time Intermedia*, a Japenese publisher. The puzzle takes place on a finite rectangular two-dimensional grid, with some cells or blocks filled with natural numbers. The goal is to fill the blocks according to the following rules:

1. You cannot fill in blocks containing numbers.
2. A number defines the number of continuous white blocks—they are linked either horizontal or vertical. Each area of white blocks contains only one number in it and they are separated by black blocks. Digonal connections do not count.

3. The black blocks are linked to be a continuous area. Diagonal connections do not count.
4. Black blocks cannot be linked to be 2×2 square.

A example of NURIKABE and a solution is shown in Figure . Implementations of

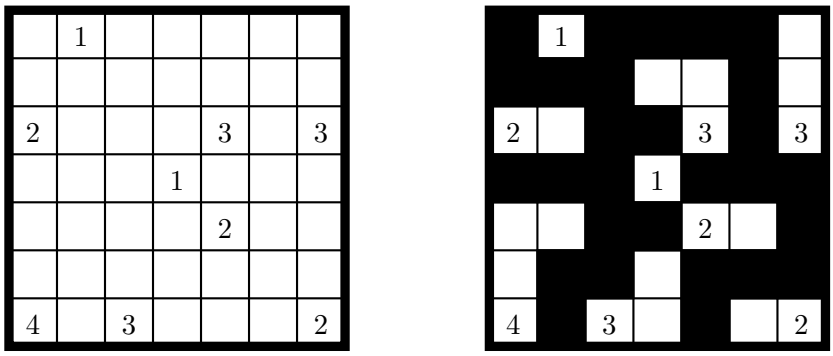


Figure 1: NURIKABE puzzle and its solution.

NURIKABE are available on the Internet; e.g., for the PALM Pilot a version can be downloaded from

`www.palmgamingworld.com/puzzle/nurikabe.shtml`

It is worth to mention that NURIKABE can be played online at `http://www.puzzle.jp`.

Formally, NURIKABE falls into the category of *pencil* puzzles. Pencil puzzles (or pencil-and-paper puzzles) are those offered as some figure on the paper and solved by drawing on the figure with the pencil. Many pencil puzzles are originated in Japan, are quite popular there, but less known outside of Japan. This is one reason, why only a few complexity results for these type of puzzles is known. Only recently, some researchers have tried to fill this gap in the literature. To our knowledge, the first result on pencil puzzles is due to Ueda and Nagao [5], which shows that Nonogram is NP-complete. Since pencil puzzles have the property that to solve them is hard, but to verify the solution is easy, most of them are contained in NP and even NP-complete. For the present the following pencil puzzles are known to be NP-complete: Slither Link [6], Cross Sum (*Kakkuro* in Japan) [4], Number Place (*Sudoku* in Japan) [7], Fillomino [7], Pearl [2], Corral [1], and Spiral Galaxies [3]. In this paper we contribute to this list, namely by showing that NURIKABE is intractable, too, by proving the following theorem:

Theorem 1 NURIKABE *on an* $n \times n$ *board is* NP*-complete.*

To this end we show how to simulate Boolean gates *via* NURIKABE puzzles. Moreover, we study also a variant of NURIKABE, where the fourth rule of the

game description is not present. In this case, it is shown, that the pencil puzzle game remains intractable, namely NP-complete.

References

[1] E. Friedman. Corral puzzles are NP-complete. Technical report, Stetson University, DeLand, FL 32723, 2002. www.stetson.edu/ efriedman/papers/corral.ps.

[2] E. Friedman. Pearl puzzles are NP-complete. Technical report, Stetson University, DeLand, FL 32723, 2002. www.stetson.edu/ efriedman/papers/pearl.ps.

[3] E. Friedman. Spiral galaxies puzzles are NP-complete. Technical report, Stetson University, DeLand, FL 32723, 2002. www.stetson.edu/ efriedman/papers/spiral.ps.

[4] T. Seta. The complexities of CROSS SUM. *IPSJ SIG Notes*, AL-84:51–58, 2002. (in Japanese).

[5] N. Ueda and T. Nagao. NP-completeness results for NONOGRAM via parsimonious reductions. Technical Report TR96-0008, Department of Computer Science, Tokyo Institute of Technology, 1996.

[6] T. Yato. On the NP-completness of the slither link puzzle. *IPSJ SIG Notes*, AL-74:25–32, 2000. (in Japanese).

[7] T. Yato. Complexity and completeness of finiding another solution and its application to puzzles. Master thesis, Graduate School of Science, University of Tokyo, January 2003.

EINE KOMBINATORISCHE EIGENSCHAFT GEWISSER $0,1$-MATRIZEN

Roman König
Universität Erlangen-Nürnberg, Institut für Informatik III
Martensstraße 3, 91058 Erlangen, Germany
e-mail: koenig@informatik.uni-erlangen.de

KURZFASSUNG

1. Einleitung

Die Datenanalyse befasst sich mit der Aufgabe, in grossen Datenbeständen Regelmäßigkeiten zu entdecken. Datenbestände stellt man sich dabei zunächst so vor, dass man eine Relation zwischen einer Menge G von Gegenständen (=Transaktionen, in der Sprache der Datenbanktheorie) und einer Menge M von Merkmalen (=Attributen) gegeben hat. Je nach Anwendung kann man sich diese Relation als ein $0,1-$Matrix oder eine Tabelle mit Kreuzen an den Stellen $(g,m) \in G \times M$, für die die Aussage: „Der Gegenstand g besitzt das Merkmal m" zutrifft, vorstellen. Solche Tabellen heißen Kontexte, die Relation wird gewöhnlich mit I bezeichnet. Bei der Betrachtung von redundanzfreien Kontexten zur Beschreibung aller endlichen Hüllensysteme treten die folgenden Matrizen auf.

2. Die Matrizen $\mathbb{A}_n$ und $\mathbb{C}_n$

Die Matrizen $\mathbb{A}_n$ und $\mathbb{C}_n$ sind 2^n-reihige $0,1-$Matrizen mit der Definition

$$\mathbb{C}_0 = \boxed{\times} \quad , \quad \mathbb{A}_0 = \square$$

$$\mathbb{C}_{n+1} = \begin{array}{c|c} \mathbb{C}_n & \mathbb{A}_n \\ \hline \mathbb{C}_n & \times \end{array} \text{ und } \quad \mathbb{A}_{n+1} = \begin{array}{c|c} \mathbb{A}_n & \times \\ \hline \mathbb{A}_n & \mathbb{A}_n \end{array} \quad (n \geq 0) \tag{2.1}$$

wobei das Symbol $\times$ die Matrix passender Größe bezeichnet, die an allen Stellen den Eintrag 1 besitzt. In der Definition von $\mathbb{A}_{n+1}$ und $\mathbb{C}_{n+1}$ hat also $\times$ die

Größe $2^n \times 2^n$. $\mathbb{A}_0$ ist die 0-Matrix der Größe 1×1. Für $n > 0$ sind die $\mathbb{A}_n$ und $\mathbb{C}_n$ quadratische Blockmatrizen mit quadratischen Blöcken. Solche Blockmatrizen der Form

$$\begin{pmatrix} \mathbb{U} & \mathbb{V} \\ \mathbb{X} & \mathbb{Y} \end{pmatrix}$$

mit invertierbarem $\mathbb{U}$ lassen sich mit Hilfe des Schur-Komplements schreiben

$$\begin{pmatrix} \mathbb{U} & \mathbb{V} \\ \mathbb{X} & \mathbb{Y} \end{pmatrix} = \begin{pmatrix} \mathbf{1} & \mathbf{0} \\ \mathbb{X}\mathbb{U}^{-1} & \mathbf{1} \end{pmatrix} \begin{pmatrix} \mathbb{U} & \mathbf{0} \\ \mathbf{0} & \mathbb{Y} - \mathbb{X}\mathbb{U}^{-1}\mathbb{V} \end{pmatrix} \begin{pmatrix} \mathbf{1} & \mathbb{U}^{-1}\mathbb{V} \\ 0 & \mathbf{1} \end{pmatrix} \tag{2.2}$$

wobei $\mathbf{1}$ die passende Einheitsmatrix und $\mathbf{0}$ die passende 0-Matrix bezeichnet. Mit Hilfe dieser Identität überzeugen wir uns, dass die Matrizen $\mathbb{C}_n$ und $\mathbb{A}_n$, sowie die $0,1-$Matrizen

$$\overline{\mathbb{C}_n} = \times - \mathbb{C}_n \text{ und } \overline{\mathbb{A}_n} = \times - \mathbb{A}_n$$

ihrer komplementären Kontexte für alle $n \geq 0$ die folgende Eigenschaft haben:

Theorem *Für alle* $n \geq 0$ *gilt:*

$$\det \mathbb{A}_n = 0 = \det \overline{\mathbb{C}_n},\ \det \overline{\mathbb{A}_n} = 1 = \det \mathbb{C}_n$$

Beweis. Für $n = 0$ ist die Aussage trivialerweise wahr. Die Matrizen $\mathbb{A}_n$ haben stets als letzte Zeile eine 0-Zeile, daher ist

$$\det \mathbb{A}_n = 0$$

für alle $n \geq 0$. Die Matrizen $\mathbb{C}_n$ haben als letzte Zeile stets eine 1-Zeile, daher hat $\overline{\mathbb{C}_n}$ als letzte Zeile ebenfalls stets eine 0-Zeile und es ist auch

$$\det \overline{\mathbb{C}_n} = 0$$

Wegen der obigen Gleichung (2.2) und der Beziehung

$$\det \begin{pmatrix} \mathbf{1} & \mathbf{0} \\ \mathbb{X}\mathbb{U}^{-1} & \mathbf{1} \end{pmatrix} = 1 = \det \begin{pmatrix} \mathbf{1} & \mathbb{U}^{-1}\mathbb{V} \\ \mathbf{0} & \mathbf{1} \end{pmatrix}$$

folgt für die Determinante einer Blockmatrix

$$\det \begin{pmatrix} \mathbb{U} & \mathbb{V} \\ \mathbb{X} & \mathbb{Y} \end{pmatrix} = \det \begin{pmatrix} \mathbb{U} & \mathbf{0} \\ \mathbf{0} & \mathbb{Y} - \mathbb{X}\mathbb{U}^{-1}\mathbb{V} \end{pmatrix}$$

und falls $\mathbb{U} \cdot \mathbb{X} = \mathbb{X} \cdot \mathbb{U}$ erfüllt ist, folgt

$$\det \begin{pmatrix} \mathbb{U} & \mathbb{V} \\ \mathbb{X} & \mathbb{Y} \end{pmatrix} = \det \mathbb{U} \cdot \det(\mathbb{Y} - \mathbb{X}\mathbb{U}^{-1}\mathbb{V}) = \det(\mathbb{U} \cdot \mathbb{Y} - \mathbb{X} \cdot \mathbb{V})$$

Damit folgt aus dem Aufbau (2.1) der Matrizen $\mathbb{C}_n$ und $\mathbb{A}_n$ und der Induktionsannahme $\det \overline{\mathbb{A}_n} = 1$ zunächst:

$$\begin{aligned}
\det \overline{\mathbb{A}_{n+1}} = \det(\times - \mathbb{A}_{n+1}) &= \det \begin{pmatrix} \times - \mathbb{A}_n & \mathbf{0} \\ \times - \mathbb{A}_n & \times - \mathbb{A}_n \end{pmatrix} \\
&= \det\bigl((\times - \mathbb{A}_n) \cdot (\times - \mathbb{A}_n)\bigr) \\
&= \det(\overline{\mathbb{A}_n} \cdot \overline{\mathbb{A}_n}) \\
&= \det \overline{\mathbb{A}_n} \cdot \det \overline{\mathbb{A}_n} \\
&= 1
\end{aligned}$$

und schließlich ist, wenn $\det \mathbb{C}_n = 1$, auch

$$\begin{aligned}
\det \mathbb{C}_{n+1} = \det \begin{pmatrix} \mathbb{C}_n & \mathbb{A}_n \\ \mathbb{C}_n & \times \end{pmatrix} &= \det(\mathbb{C}_n \cdot \times - \mathbb{C}_n \cdot \mathbb{A}_n) \\
&= \det(\mathbb{C}_n \cdot (\times - \mathbb{A}_n)) \\
&= \det \mathbb{C}_n \cdot \det \overline{\mathbb{A}_n} \\
&= 1
\end{aligned}$$

Literatur

[1] B. Ganter, R. Wille: *Formale Begriffsanalyse: Mathematische Grundlagen.* Springer Verlag Heidelberg, 1996, ISBN 3-540-60868-0

[2] B. Ganter: Begriffe und Implikationen, in: Stumme, Wille (Hrsg.): *Begriffliche Wissensverarbeitung – Methoden und Anwendungen*, pp.1 - 24, Springer Verlag, 2000, ISBN 3-540-66391-6

[3] R. König: Endliche Hüllensystem und ihre Implikationenbasen (eingereicht)

ON THE DESCRIPTIONAL POWER OF HEADS, COUNTERS, AND PEBBLES

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

ABSTRACT

We investigate the descriptional complexity of deterministic two-way k-head finite automata (k-DHA). It is shown that between deterministic finite automata (DFA) and any k-DHA, $k \geq 2$, there are savings in the size of description which cannot be bounded by any recursive function. The same is true for the other end of the hierarchy. Such non-recursive trade-offs are also shown between any k-DHA and $\mathsf{DSPACE}(\log)$ = multi-DHA. We also address the particular case of unary languages. In general, it is possible that non-recursive trade-offs for arbitrary languages reduce to recursive trade-offs for unary languages. Here we present enormous lower bounds for the unary trade-offs between DFA and any k-DHA, $k \geq 2$. Furthermore, several known simulation results imply the presented trade-offs for other descriptional systems, e.g. deterministic two-way finite automata with k pebbles or with k linearly bounded counters.

Keywords: Descriptional complexity, non-recursive trade-offs, k-head finite automata.

Formal languages can have many representations in the world of automata, grammars and other rewriting systems, language equations, logical formulas etc. So it is natural to investigate the succinctness of their representation by different models in order to optimize the space requirements. The regular languages are one of the first and most intensely studied language families. It is well known that nondeterministic finite automata (NFA) can offer exponential savings in space compared with deterministic finite automata (DFA). Concerning the number of states, 2^n is a tight bound for the NFA to DFA conversion [8]. For example, asymptotically bounds are $O(n^n)$ for the two-way DFA to one-way DFA conversion [8], $2^{O(n^2)}$ for the two-way NFA to one-way DFA conversion [9], $O(\sqrt{2^n})$ for the two-way DFA to one-way NFA conversion [1], and $O(2^{n+3})$ for the two-way NFA to one-way NFA conversion [1]. The latter reference is a valuable source for further simulation results.

All mentioned trade-offs in the number of states are bounded by recursive functions. But, for example, there is no recursive function which bounds the savings in descriptional complexity between deterministic and unambiguous pushdown automata [12]. In [10] it is proved that the trade-off between unambiguous and nondeterministic pushdown automata is also non-recursive. Recent results involving the parallel model of cellular automata can be found in [5]. In particular, non-recursive trade-offs are shown there between DFA and real-time one-way cellular automata (real-time OCA), between pushdown automata and real-time OCA, and between real-time OCA and real-time two-way cellular automata.

A comprehensive survey of descriptional complexity of machines with limited resources is [3] which is a valuable source for further results and references.

Nevertheless, some challenging problems of finite automata are open. An important example is the question how many states are sufficient and necessary to simulate two-way NFA with two-way DFA. The problem has been raised in [9] and partially solved in [11].

When certain problems are difficult to resolve in general, a natural question concerns simpler versions. To this regard promising research has been done for unary languages. It turned out that this particular case is essentially different from the general case. The problem of evaluating the costs of unary automata simulations has been raised in [11]. In [2] it has been shown that the unary NFA to DFA conversion takes $e^{\Theta(\sqrt{n \cdot \ln(n)})}$ states, the NFA to two-way DFA conversion has been solved with a bound of $O(n^2)$ states, and the costs of the unary two-way to one-way DFA conversion reduces to $e^{\Theta(\sqrt{n \cdot \ln(n)})}$. Several more results can be found in [6, 7]. Furthermore, in [5] it is shown for real-time OCA that non-recursive trade-offs for arbitrary languages reduce to recursive trade-offs for unary languages.

Here we investigate the descriptional complexity of deterministic two-way k-head finite automata (k-DHA). In particular, we consider the trade-offs between DFA and k-DHA for any $k \geq 2$, and the trade-offs between any k-DHA and the deterministic log-space bounded Turing machines whose languages are exactly the languages accepted by the union of all k-DHA. All these trade-offs are shown to be non-recursive. For unary languages it is not known whether the trade-offs are recursive or not. Here we present enormous lower bounds between DFA and any k-DHA. Furthermore, these lower bounds increase with the number of heads in a nice way. Provided minimality can be shown, these bounds can also serve as lower bounds between k-DHA and $(k+1)$-DHA.

The results are adapted to other types of acceptors, e.g. deterministic two-way finite automata with k pebbles or with k linearly bounded counters. Some concerned and related open questions are discussed.

References

[1] Birget, J.-C. *State-complexity of finite-state devices, state compressibility and incompressibility*. Math. Systems Theory 26 (1993), 237–269.

[2] Chrobak, M. *Finite automata and unary languages*. Theoret. Comput. Sci. 47 (1986), 149–158.

[3] Goldstine, J., Kappes, M., Kintala, C. M. R., Leung, H., Malcher, A., and Wotschke, D. *Descriptional complexity of machines with limited resources*. J. UCS 8 (2002), 193–234.

[4] Hartmanis, J. *On the succinctness of different representations of languages*. International Colloquium on Automata, Languages and Programming (ICALP 1979), LNCS 71, 1979, pp. 282–288.

[5] Malcher, A. *Descriptional complexity of cellular automata and decidability questions*. J. Autom. Lang. Comb. 7 (2002), 549–560.

[6] Mereghetti, C. and Pighizzini, G. *Two-way automata simulations and unary languages*. J. Autom. Lang. Comb. 5 (2000), 287–300.

[7] Mereghetti, C. and Pighizzini, G. *Optimal simulations between unary automata*. SIAM J. Comput. 30 (2001), 1976–1992.

[8] 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.

[9] Sakoda, W. J. and Sipser, M. *Nondeterminism and the size of two way finite automata*. ACM Symposium on Theory of Computing (STOC 1978), 1978, pp. 275–286.

[10] Schmidt, E. M. and Szymanski, T. G. *Succinctness of descriptions of unambiguous context-free languages*. SIAM J. Comput. 6 (1977), 547–553.

[11] Sipser, M. *Lower bounds on the size of sweeping automata*. J. Comput. System Sci. 21 (1980), 195–202.

[12] Valiant, L. G. *A note on the succinctness of descriptions of deterministic languages*. Inform. Control 32 (1976), 139–145.

VERIFICATION OF NON-REGULAR PROPERTIES

Martin Lange
Institut für Informatik
Ludwig-Maximilians-Universität München
e-mail: mlange@informatik.uni-muenchen.de

ABSTRACT

We present a game-based formalism that can be used to do local model checking for FLC, a modal fixed point logic that extends the μ-calculus with a sequential composition operator. This logic is capable of expressing non-regular properties which are interesting for verification purposes.

1. Introduction

The modal μ-calculus [2] is an important specification language for the verification of temporal properties. This is mainly due to the fact that it subsumes most temporal, dynamic or description logics.

On the other hand, the modal μ-calculus is equi-expressive to Monadic Second Order Logic [1], i.e. it can only describe "regular" properties. In [7], Müller-Olm has introduced FLC, *Fixed Point Logic with Chop*, which extends the modal μ-calculus with a sequential composition operator, and has shown that its expressive power is not limited by regularity. In fact, it is not hard to define the language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ in FLC on words for example.

On finite structures, FLC's model checking problem is decidable [7]. In [5], a tableau formalism was introduced that can be used to do *global* model checking. Here we present a game-based framework [4] for *local* model checking. These games cannot deny a certain similarity with alternating push-down automata over trees. Just as FLC extends the modal μ-calculus, they extend its model checking games [8].

We also give examples of FLC-definable properties that are interesting for verification purposes and present the most important results on FLC.

2. Preliminaries

A *transition system* $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \mathcal{A}\}, L)$ over a set $\mathcal{P} = \{\mathtt{tt}, \mathtt{ff}, q, \overline{q}, \ldots\}$ of propositional constants consists of *states*, *transition relations* and a *labelling function* from states to sets of propositions. Formulas of FLC are given by

$$\varphi \quad ::= \quad q \mid Z \mid \tau \mid \langle a \rangle \mid [a] \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mu Z.\varphi \mid \nu Z.\varphi \mid \varphi;\varphi$$

where $q \in \mathcal{P}$, $a \in \mathcal{A}$, and $Z \in \mathcal{V}$, a set of propositional variables. We will write σ for μ or ν. Formulas are assumed to be well named in the sense that each binder variable is distinct. Our main interest is with closed formulas, that do not have free variables, in which case there is a function $fp : \mathcal{V} \to \text{FLC}$ that maps each variable to its defining fixed point formula. The set $Sub(\varphi)$ of *subformulas* of φ and its *alternation depth* $ad(\varphi)$ are defined as usual.
The *tail* of a variable Z in a formula φ, tl_Z is a set consisting of those formulas that occur "behind" Z in $fp(Z)$ in φ. We use sequential composition for sets of formulas in a straightforward way: $\{\varphi_0, \ldots, \varphi_n\}; \psi := \{\varphi_0; \psi, \ldots, \varphi_n; \psi\}$. Let

$$tl_Z(\psi) := \{\psi\} \text{ if } \psi \in \mathcal{P} \cup \{\tau, \langle a \rangle, [a] \mid a \in \mathcal{A}\} \qquad tl_Z(\sigma Y.\psi) := tl_Z(\psi)$$

$$tl_Z(\varphi \vee \psi) = tl_Z(\varphi \wedge \psi) := tl_Z(\varphi) \cup tl_Z(\psi) \qquad tl_Z(Y) := \begin{cases} \{Y\} & \text{if } Y \neq Z \\ \{\tau\} & \text{o.w.} \end{cases}$$

$$tl_Z(\varphi;\psi) := \left\{ \begin{cases} tl_Z(\varphi);\psi & \text{if } Z \in Sub(\varphi) \\ \{\tau\} & \text{o.w.} \end{cases} \right\} \cup \left\{ \begin{cases} tl_Z(\psi) & \text{if } Z \in Sub(\psi) \\ \{\tau\} & \text{o.w.} \end{cases} \right\}$$

The tail of Z in φ is simply calculated as $tl_Z := tl_Z(fp(Z))$.

An important factor in the complexity of FLC's model checking problem is the *sequential depth* $sd(\varphi)$ of a formula. Informally the sequential depth of a formula is the maximal number of times a variable is sequentially composed with itself. It is defined as $sd(\varphi) := max\{sd_Z(fp(Z)) \mid Z \in Sub(\varphi)\} - 1$ where

$$sd_Z(\varphi \vee \psi) := max\{sd_Z(\varphi), sd_Z(\psi)\} \qquad sd_Z(\varphi \wedge \psi) := max\{sd_Z(\varphi), sd_Z(\psi)\}$$

$$sd_Z(\varphi;\psi) := sd_Z(\varphi) + sd_Z(\psi) \qquad sd_Z(\sigma Y.\varphi) := sd_Z(\varphi)$$

$$sd_Z(\psi) := 0 \quad \text{if } \psi \in \{q, \tau, \langle a \rangle, [a]\} \qquad sd_Z(Y) := \begin{cases} 1 & \text{if } Y = Z \\ 0 & \text{o.w.} \end{cases}$$

To simplify the notation of a formula's semantics we assume a transition system $\mathcal{T}$ to be fixed for the remainder of the paper. In order to handle open formulas we need environments $\rho : \mathcal{V} \to (2^{\mathcal{S}} \to 2^{\mathcal{S}})$. $\rho[Z \mapsto f]$ is the function that maps Z to f and agrees with ρ on all other arguments.

The semantics $[\![\cdot]\!]^{\mathcal{T}}_{\rho} : 2^{\mathcal{S}} \to 2^{\mathcal{S}}$ of an FLC formula, relative to $\mathcal{T}$ and ρ, is a monotone function on subsets of states with respect to the inclusion ordering on $2^{\mathcal{S}}$. These functions together with the partial order given by

$$f \sqsubseteq g \text{ iff } \forall X \subseteq \mathcal{S} : f(X) \subseteq g(X)$$

form a complete lattice with joins $\sqcup$ and meets $\sqcap$. By the Tarski-Knaster Theorem [9] the least and greatest fixed points of functionals $F : (2^{\mathcal{S}} \to 2^{\mathcal{S}}) \to (2^{\mathcal{S}} \to 2^{\mathcal{S}})$ exist. They are used to interpret fixed point formulas of FLC.

$$\begin{aligned}
[\![q]\!]_\rho &= \lambda X.\{s \in \mathcal{S} \mid q \in L(s)\} & [\![Z]\!]_\rho &= \rho(Z)\\
[\![\varphi;\psi]\!]_\rho &= [\![\varphi]\!]_\rho \circ [\![\psi]\!]_\rho & [\![\tau]\!]_\rho &= \lambda X.X\\
[\![\varphi \vee \psi]\!]_\rho &= \lambda X.[\![\varphi]\!]_\rho(X) \cup [\![\psi]\!]_\rho(X) & [\![\varphi \wedge \psi]\!]_\rho &= \lambda X.[\![\varphi]\!]_\rho(X) \cap [\![\psi]\!]_\rho(X)\\
[\![\langle a\rangle]\!]_\rho &= \lambda X.\{s \in \mathcal{S} \mid \exists t \in X, \text{ s.t. } s \xrightarrow{a} t\}\\
[\![[a]]\!]_\rho &= \lambda X.\{s \in \mathcal{S} \mid \forall t \in \mathcal{S}, s \xrightarrow{a} t \Rightarrow t \in X\}\\
[\![\mu Z.\varphi]\!]_\rho &= \bigsqcap\{f : 2^{\mathcal{S}} \to 2^{\mathcal{S}} \mid f \text{ monotone}, [\![\varphi]\!]_{\rho[Z\mapsto f]} \sqsubseteq f\}\\
[\![\nu Z.\varphi]\!]_\rho &= \bigsqcup\{f : 2^{\mathcal{S}} \to 2^{\mathcal{S}} \mid f \text{ monotone}, f \sqsubseteq [\![\varphi]\!]_{\rho[Z\mapsto f]}\}
\end{aligned}$$

A state s satisfies a formula φ under ρ, written $s \models_\rho \varphi$, iff $s \in [\![\varphi]\!]_\rho(\mathcal{S})$.

Example Let $\mathcal{A} = \{a, b\}$ and $\varphi = \nu Y.[b]\mathtt{ff} \wedge [a](\nu Z.[b] \wedge [a](Z;Z)); (([a]\mathtt{ff} \wedge [b]\mathtt{ff}) \vee Y)$. It says "the number of bs never exceeds the number of as" which is non-regular and, therefore, is not expressible in $\mathcal{L}_\mu$. This is an interesting property of protocols when a and b are the actions *send* and *receive*.

The subformula $\psi = \nu Z.[b] \wedge [a](Z;Z)$ expresses "there can be at most one b more than there are as". This can be understood best by unfolding the fixed point formula and thus obtaining sequences of modalities and variables. It is easy to see that replacing a Z with a $[b]$ reduces the number of Zs whereas replacing it with the other conjunct adds a new Z to the sequence.

Then, $[b]\mathtt{ff} \wedge [a]\psi$ postulates that at the beginning no b is possible and for every n as there can be at most n bs. Finally, the Y in φ allows such sequences to be composed or finished in a deadlock state.

Example The property of *repeating words on every path* is expressible in FLC, if there is an end marker #. The following formula says that every sequence of actions w ending in a # is followed by another w.

$$\begin{aligned}
\varphi &:= \# \vee \bigwedge_{a\in\Sigma} \psi_a\\
\psi_a &:= (\mu Z.\#; \mathtt{tt} \vee ([-]; Z \wedge [a]; (\mu Y.\# \vee [-]; Y)); [-]); \langle a\rangle
\end{aligned}$$

Each ψ_a checks for the letter $a \in \Sigma$: if a occurs before # on a path at position n then there is an a-action n steps after the later #.

3. Local model checking games for FLC

Model checking games are played by two players, called $\exists$ and $\forall$, on a transition system $\mathcal{T}$ and an FLC formula φ. Player $\exists$ tries to establish that a given state s of $\mathcal{T}$ satisfies φ, whereas $\forall$ tries to show that $s \not\models \varphi$.

A play is a (possibly infinite) sequence $C_0, C_1, \ldots$ of configurations, and a configuration is an element of $Conf = \mathcal{S} \times Sub(\varphi)^* \times Sub(\varphi)$. It is written $s, \delta \vdash \psi$ where δ is interpreted as a stack of subformulas with its top on the left. The empty stack is denoted by ϵ. With a stack $\delta = \varphi_0 \ldots \varphi_k$ we associate the formula $\delta := \varphi_0; \ldots; \varphi_k$ while ϵ is associated with the formula τ.

$$(\vee): \ \frac{s, \delta \vdash \varphi_0 \vee \varphi_1}{s, \delta \vdash \varphi_i} \ \exists i \qquad (\wedge): \ \frac{s, \delta \vdash \varphi_0 \wedge \varphi_1}{s, \delta \vdash \varphi_i} \ \forall i \qquad (\texttt{FP}): \ \frac{s, \delta \vdash \sigma Z.\varphi}{s, \delta \vdash Z}$$

$$(\texttt{VAR}): \ \frac{s, \delta \vdash Z}{s, \delta \vdash \varphi} \ \text{if } fp(Z) = \sigma Z.\varphi \qquad (;): \ \frac{s, \delta \vdash \varphi_0; \varphi_1}{s, \varphi_1 \delta \vdash \varphi_0} \qquad (\texttt{TERM}): \ \frac{s, \psi\delta \vdash \tau}{s, \delta \vdash \psi}$$

$$(\texttt{DIAM}): \ \frac{s, \psi\delta \vdash \langle a \rangle}{t, \delta \vdash \psi} \ \exists\, s \xrightarrow{a} t \qquad (\texttt{BOX}): \ \frac{s, \psi\delta \vdash [a]}{t, \delta \vdash \psi} \ \forall\, s \xrightarrow{a} t$$

Figure 1: The model checking game rules.

Each play for s_0 of $\mathcal{T}$ and φ begins with $C_0 = s_0, \epsilon \vdash \varphi$. A play proceeds according to the rules depicted in Figure 1. Some of them require one of the players to choose a subformula or a state. This is indicated at the right side of a rule. Rules $(\vee)$ and $(\wedge)$ are straightforward. Rules (VAR) and (FP) are justified by the unfolding characterisations of fixed points: $\sigma Z.\varphi \equiv \varphi[\sigma Z.\varphi/Z]$. If a formula $\varphi; \psi$ is encountered then ψ is stored on the stack with rule $(;)$ to be dealt with later on while the players try to prove or refute φ. Modalities cause either of the players to choose a successor state. After that rules, (DIAM) and (BOX) pop the top formula from the stack into the right side of the actual configuration. Rule (TERM) does the same without a choice of one of the players. The winning conditions are not straight-forward but require another definition.

Definition *A variable Z is called* stack-increasing *in a play $C_0, C_1, \ldots$ if there are infinitely many configurations $C_{i_0}, C_{i_1}, \ldots$, s.t.*

- $i_j < i_{j+1}$ *for all* $j \in \mathbb{N}$
- $C_{i_j} = s_j, \delta_j \vdash Z$ *for some* s_j *and* δ_j,
- *for all* $j \in \mathbb{N}$ *exists* $\gamma \in tl_Z \cup \{\epsilon\}$ *s.t.* $\delta_{j+1} = \gamma\delta_j$, *and* $\gamma = \epsilon$ *iff* $tl_Z = \emptyset$.

Player $\exists$ wins a play $C_0, \ldots, C_n, \ldots$ iff

1. $C_n = s, \delta \vdash q$ and $q \in L(s)$, or
2. $C_n = s, \epsilon \vdash \tau$, or
3. $C_n = s, \epsilon \vdash \langle a \rangle$ and there is a $t \in \mathcal{S}$, s.t. $s \xrightarrow{a} t$, or
4. $C_n = s, \delta \vdash [a]$, and $\delta = \epsilon$ or $\nexists t \in \mathcal{S}$, s.t. $s \xrightarrow{a} t$, or

5. the play is infinite, and there is a $Z \in \mathcal{V}$ s.t. Z is the greatest, w.r.t. $<_\varphi$, stack-increasing variable and $fp(Z) = \nu Z.\psi$ for some ψ.

Player $\forall$ wins such a play iff

6. $C_n = s, \delta \vdash q$ and $q \notin L(s)$, or
7. $C_n = s, \delta \vdash \langle a \rangle$, and $\nexists t \in \mathcal{S}$, s.t. $s \xrightarrow{a} t$, or
8. the play is infinite, and there is a $Z \in \mathcal{V}$ s.t. Z is the greatest, w.r.t. $<_\varphi$, stack-increasing variable and $fp(Z) = \mu Z.\psi$ for some ψ.

A player has a winning strategy, or simply wins the game, for $s, \delta \vdash \varphi$ if she can enforce a winning play for herself, starting with this configuration.

The following example illustrates the importance of being stack-increasing. Note that in a $\mathcal{L}_\mu$ model checking game the winner is determined by the outermost variable that occurs infinitely often. There, if two variables occur infinitely often then one of them is outer and the inner one's defining fixed point formula, say $fp(Y)$, occurs infinitely often, too. Thus two occurrences of Y cannot be related to each other in terms of approximants indices. FLC only has this property for stack-increasing variables.

Example Let $\varphi = \mu Y.\langle b \rangle \vee \langle a \rangle \nu Z.Y; Z; Y$. $ad(\varphi) = 1$ and $sd(\varphi) = 2$. Let $\mathcal{T}$ be the transition system consisting of states $\{s, t\}$ and transitions $s \xrightarrow{a} t$ and $t \xrightarrow{b} t$. $s \models \varphi$. The game tree for player $\exists$ is shown in Figure 2. Since φ does not contain any $\wedge$, $[a]$ or $[b]$, player $\forall$ does not make any choices and the tree is in fact a single play.

$$
\begin{array}{ll}
\underline{s, \epsilon \vdash \mu Y.\langle b \rangle \vee \langle a \rangle \nu Z.Y; Z; Y} & \\
\underline{s, \epsilon \vdash Y} & \\
\underline{s, \epsilon \vdash \langle b \rangle \vee \langle a \rangle \nu Z.Y; Z; Y} & \exists \langle a \rangle \nu Z.Y; Z; Y \\
\underline{s, \epsilon \vdash \langle a \rangle \nu Z.Y; Z; Y} & \\
\underline{s, \nu Z.Y; Z; Y \vdash \langle a \rangle} & \exists s \xrightarrow{a} t \\
\underline{t, \epsilon \vdash \nu Z.Y; Z; Y} & \\
\underline{t, \epsilon \vdash Z} & \\
\underline{t, \epsilon \vdash Y; Z; Y} & \\
\underline{t, Z; Y \vdash Y} & \\
\underline{t, Z; Y \vdash \langle b \rangle \vee \langle a \rangle \nu Z.Y; Z; Y} & \exists \langle b \rangle \\
\underline{t, Z; Y \vdash \langle b \rangle} & \exists t \xrightarrow{b} t \\
\underline{t, Y \vdash Z} & \\
\underline{t, Y \vdash Y; Z; Y} & \\
\underline{t, Z; Y; Y \vdash Y} & \\
\vdots &
\end{array}
$$

Figure 2: $\exists$'s winning play from the example.

Both Y and Z occur infinitely often in the play. However, neither $fp(Y)$ nor $fp(Z)$ does. Note that $Z <_\varphi Y$. Y gets "fulfilled" each time it is replaced by its defining fixed point formula, but reproduced by Z. On the other hand, Y does not start a new computation of $fp(Z)$ each time it is reproduced. But Y is not stack-increasing whereas Z is. And Z denotes a greatest fixed point, therefore player $\exists$ wins this play.

4. Results on FLC('s model checking problem)

Theorem ***(Decidability)*** *FLC's satisfiability checking problem is undecidable; FLC does not have the finite model property; its model checking problem is decidable for finite structures* [7]*; but undecidable for normed deterministic BPA already* [4].

Theorem ***(Expressiveness)*** *On linear structures (infinite words) the class of FLC-definable languages coincides with the class of alternating context-free grammars (with a parity acceptance condition)* [3].

Theorem ***(Complexity)*** *FLC's model checking problem is in EXPTIME and PSPACE-hard* [5]*, even for a fixed formula* [6]*. For formulas with fixed alternation (and sequential) depth, the model checking problem is in PSPACE.*

Obviously, the complexity results refer to finite structures only.

References

[1] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional μ-calculus with respect to monadic second order logic. In U. Montanari and V. Sassone, editors, *Proc. 7th Conf. on Concurrency Theory, CONCUR'96*, volume 1119 of *LNCS*, pages 263–277, Pisa, Italy, August 1996. Springer.

[2] D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, December 1983.

[3] M. Lange. Alternating context-free languages and linear time μ-calculus with sequential composition. In P. Panangaden and U. Nestmann, editors, *Proc. 9th Workshop on Expressiveness in Concurrency, EXPRESS'02*, volume 68.2 of *ENTCS*, pages 71–87, Brno, Czech Republic, August 2002. Elsevier.

[4] M. Lange. Local model checking games for fixed point logic with chop. In L. Brim, P. Jančar, M. Křetínský, and A. Kučera, editors, *Proc. 13th Conf. on Concurrency Theory, CONCUR'02*, volume 2421 of *LNCS*, pages 240–254, Brno, Czech Republic, August 2002. Springer.

[5] M. Lange and C. Stirling. Model checking fixed point logic with chop. In M. Nielsen and U. H. Engberg, editors, *Proc. 5th Conf. on Foundations of*

Software Science and Computation Structures, FOSSACS'02, volume 2303 of *LNCS*, pages 250–263, Grenoble, France, April 2002. Springer.

[6] M. Müller-Olm. private communication.

[7] M. Müller-Olm. A modal fixpoint logic with chop. In C. Meinel and S. Tison, editors, *Proc. 16th Symp. on Theoretical Aspects of Computer Science, STACS'99*, volume 1563 of *LNCS*, pages 510–520, Trier, Germany, 1999. Springer.

[8] C. Stirling. Local model checking games. In I. Lee and S. A. Smolka, editors, *Proc. 6th Conf. on Concurrency Theory, CONCUR'95*, volume 962 of *LNCS*, pages 1–11, Berlin, Germany, August 1995. Springer.

[9] A. Tarski. A lattice-theoretical fixpoint theorem and its application. *Pacific J.Math.*, 5:285–309, 1955.

A HIERARCHY OF MSC LANGUAGES

Benedikt Bollig
Lehrstuhl für Informatik II
Aachen University of Technology
Germany
e-mail: bollig@informatik.rwth-aachen.de

and

Martin Leucker
IT department
Uppsala University
Sweden
e-mail: Martin.Leucker@it.uu.se

ABSTRACT

Introduction A common design practice when developing communicating systems is to start with drawing scenarios showing the intended interaction of the system to be. The standardized notion of *message sequence charts* (MSCs, [3]) is widely used in industry to formalize such typical behaviours.

A message sequence chart 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 and interpreted as time 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. Collections of MSCs are used to capture the scenarios that a designer might want the system to follow or to avoid. Figure 1 shows three MSCs.

To support the detection of errors and to validate the intended design, formal analysis techniques for MSCs are of great value. One important question that has been studied recently is model checking MSC specifications. The motivation is that a designer could specify the scenarios using a collection of MSCs and verify them against certain requirements.

Another goal is to derive an automatic implementation of the system to develop. While this goal might be too ambitious in general, at least a generation of its design is valuable. In other words, we are interested in generating a distributed automaton *realizing* the behaviour given in form of scenarios.

To attack this problem, several questions have to be discussed. What is the *right* automaton model? What class of languages are we interested in realizing? The latter question has a standard answer which is that we want to realize *regular* languages, which, of course, just defers the question to the one "What is a regular MSC language?"

The answer to this question is not unique but debatable. We do not intend to give a final answer to this question in this abstract. Instead we recall existing notions of regularity for these systems, propose alternative ones, and compare the expressiveness of certain sets of MSC languages and message-passing automata. Hence, our work is another step towards a theory of regular MSC languages by exhibiting the domain of MSC languages and distributed notions of automata. Our work can be summarized in two pictures, Figure 2 and Figure 3, which we explain in the rest of this abstract.

An extended version of the paper is available at `http://aib.informatik.rwth-aachen.de/2003/`.

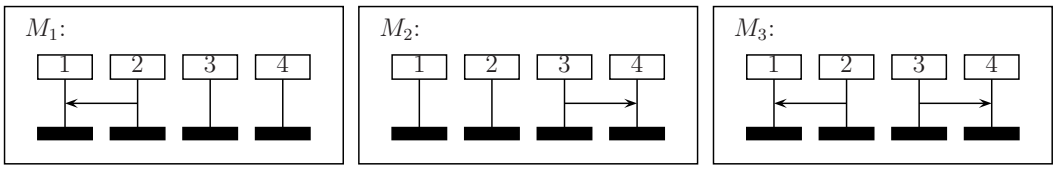


Figure 1: The language of a locally-accepting finite MPA

Regular word languages? There has been several proposals for the *right* notion of regularity for MSC languages. Henriksen et. al. started in [2], proposing that an MSC language is regular when its set of linearizations is regular. We denote this class of languages by $\mathcal{R}$ (see Figure 2).

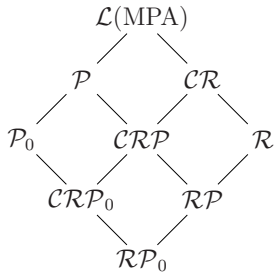


Figure 2: The hierarchy of regular and product MSC languages

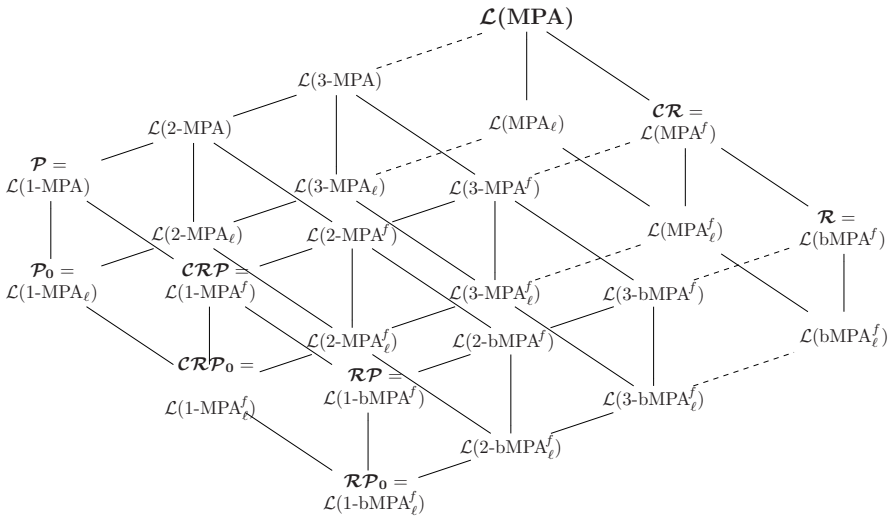


Figure 3: A Hierarchy of MSC Languages

One could likewise argue to call a set of MSCs regular, when it is definable by monadic second order logic (adjusted to certain MSCs), since this important property for verification holds for regular sets of words. This view was also pursued in [2] and it is shown that this class coincides with $\mathcal{R}$.

It was shown that languages in $\mathcal{R}$ are *bounded*. Intuitively, an MSC language is bounded iff their structure exhibits a bound on the number of send events that are not yet received by the receiving process. For example, the MSC language $\mathcal{M}_{\text{cf}}$ induced by the language $(p!q, p?q)^*$ is not bounded and hence not regular. It induces the set of MSCs that send from process p to process q, denoted by a send event $(p!q)$ and its corresponding receive event $(p?q)$. The set of all its linearizations gives rise to the set of all words that have same number of send and receive events, where for every prefix the number of send events is larger or equal to the number of receive events. This language is not regular, since, intuitively, we would need an unbounded counter for the send events.

In simple words, *regularity* aims at *finiteness.*

Product languages? The underlying systems exemplified by MSCs are distributed in nature and the notion of *process* is central. It is therefore natural to consider every process to be represented as a single state machine or transition system. Furthermore, one defines a notion of communication, describing how these parallel systems work together.

Languages defined by finite transition systems working in parallel are known as *product languages* and were initially studied in [4]. That paper discusses finite state systems. There is a common initial state and two different notions of accepting condition. Either the acceptance condition is *local*, i.e. every process

decides on his own when to stop, or, it is *global*, which allows to select certain combinations of local final states to be a valid point for terminating. It is clear that (provided the system has a single initial state) the latter notion of acceptance is more expressive than local acceptance.

In the context of (linearizations of) MSC languages, [1] studied classes of MSCs taking up the idea of product behaviour. Languages of MSCs must be closed under *inference*. The idea can be described looking at the setting in Figure 1. When realizing the behaviour described in M_1 and in M_2, it is argued that behaviour of M_3 is also possible. Since Processes 1 and 2 as well as 3 and 4 do not synchronize, the four processes do not know whether behaviour M_1 should be realized or that of M_2. We call this class of MSCs *weak product MSC languages* and denote it by $\mathcal{P}_0$ (see Figure 2). Note that in [1] no finiteness condition was studied. It is possible to realize the non-regular language $\mathcal{M}_{\text{cf}}$.

In simple words, *product* respects *independence.*

What else? Let us study extensions of the two existing approaches. When thinking about an automaton model realizing MSC languages, the allowance of different initial states or global final states will give us classes of languages closed under finite unions. For example, one could realize exactly the set consisting of M_1 and M_2. Thus, when considering finite unions of sets of $\mathcal{P}_0$ languages, one obtains the richer class of *product MSC languages*, denoted by $\mathcal{P}$. Combining the ideas of *independence* and *finiteness*, we get $\mathcal{RP}_0$ or $\mathcal{RP}$.

The drawback of the regularity notion used for $\mathcal{R}$ is that the simple language $\mathcal{M}_{\text{cf}}$ is not regular. We introduce *regularly-represented languages* ($\mathcal{CR}$) which follows the idea that the MSC language is *obtained* by a regular language. More specifically, we call an MSC language regularly-represented iff there is regular language that, restricted to linearizations of MSCs, induces this language. Together with independence, we obtain the class $\mathcal{CRP}$, or, when starting from $\mathcal{RP}_0$, the class, $\mathcal{CRP}_0$.

Automata characterizations? Instead of following the semantic approach in looking for notions of MSC languages, one could exhibit machine models. Let us turn to Figure 3. We study message-passing automata (MPA). They consist of several components that communicate using a channel. The systems have a single initial state. Dropping the requirement of finiteness (regularity or regularly represented), one needs an infinite state space for the machines, which is denoted by the missing superscript f. When considering finite unions of languages, we have to move towards global acceptance conditions, denoted by a missing ℓ.

When extending the expressiveness from regularity to regularly-represented, we drop the boundedness condition of MSCs. This is represented in Figure 3 by a missing b. It can be shown, that when realizing regular languages, one needs so-called *synchronization messages* that are used to tell other components which

transition was taken. We show that the more of these messages are allowed, the more expressive power we have. The restriction to n synchronization messages is described by a preceding n- in Figure 3.

References

[1] R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *22nd International Conference on Software Engineering*, pages 304–313, 2000.

[2] 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'2000)*, volume 1893 of *Lecture Notes in Computer Science*, pages 405–414. Springer, 2000.

[3] ITU-TS. ITU-TS Recommendation Z.120: Message Sequence Chart 1999 (MSC99). Technical report, ITU-TS, Geneva, 1999.

[4] P. S. Thiagarajan. A trace consistent subset of PTL. In Insup Lee and Scott A. Smolka, editors, *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95)*, volume 962 of *Lecture Notes in Computer Science*, pages 438–452, Philadelphia, Pennsylvania, 21–24 August 1995. Springer.

MINIMIZING FINITE AUTOMATA IS COMPUTATIONALLY HARD

Andreas Malcher
Institut für Informatik, Johann Wolfgang Goethe-Universität
D-60054 Frankfurt am Main, Germany
e-mail: malcher@psc.informatik.uni-frankfurt.de

ABSTRACT

It is known that deterministic finite automata (DFAs) can be algorithmically minimized, i.e., a DFA M can be converted to an equivalent DFA M' which has a minimal number of states. The minimization can be done efficiently [5]. On the other hand, it is known that unambiguous finite automata (UFAs) and nondeterministic finite automata (NFAs) can be algorithmically minimized too, but their minimization problems turn out to be NP-complete and PSPACE-complete, respectively [7]. In this paper, the time complexity of the minimization problem for two restricted types of finite automata is investigated. These automata are nearly deterministic, since they only allow a small amount of nondeterminism to be used. The main result is that the minimization problems for these models are computationally hard, namely NP-complete. Hence, even the slightest extension of the deterministic model towards a nondeterministic one, e.g., allowing at most one nondeterministic move in every accepting computation or allowing two initial states instead of one, results in computationally intractable minimization problems.

Keywords: Finite automata, limited nondeterminism, minimization, NP-complete.

1. Introduction

Finite automata are a well-investigated concept in theoretical computer science with a wide range of applications such as lexical analysis, pattern matching, or protocol specification in distributed systems. Due to time and space constraints it is often very useful to provide minimal or at least succinct descriptions of such automata. Deterministic finite automata (DFAs) and their corresponding language class, the set of regular languages, possess many nice properties such as, for example, closure under many language operations and many decidable questions. In addition, most of the decidability questions for DFAs, such as membership, emptiness, or equivalence, are efficiently solvable (cf. Sect. 5.2 in

[12]). Furthermore, in [5] a minimization algorithm for DFAs is provided working in time $O(n \log n)$, where n denotes the number of states of the given DFA.

It is known that both nondeterministic finite automata (NFAs) and DFAs accept the set of regular languages, but NFAs can achieve exponentially savings in size when compared to DFAs [10]. Unfortunately, certain decidability questions, which are solvable in polynomial time for DFAs, are computationally hard for NFAs such as equivalence, inclusion, or universality [11, 12]. Furthermore, the minimization of NFAs is proven to be PSPACE-complete in [7]. In the latter paper, it is additionally shown that unambiguous finite automata (UFAs) have an NP-complete minimization problem.

Therefore, we can summarize that determinism permits efficient solutions whereas the use of nondeterminism often makes solutions computationally intractable. Thus, one might ask what amount of nondeterminism is necessary to make things computationally hard, or, in other words, what amount of nondeterminism may be allowed so that efficiency is preserved.

Measures of nondeterminism in finite automata were first considered in [9] and [1] where the relation between the amount of nondeterminism of an NFA and the succinctness of its description is studied. Here, we look at computational complexity aspects of NFAs with a fixed finite amount of nondeterminism. In particular, these NFAs are restricted such that within every accepting computation at most a fixed number of nondeterministic moves is allowed to be chosen. It is easily observed that certain decidability questions then become solvable in polynomial time in contrast to arbitrary NFAs. However, the minimization problem for such NFAs is proven to be NP-complete.

We further investigate a model where the nondeterminism used is not only restricted to a fixed finite number of nondeterministic moves, but is additionally cut down such that only the first move is allowed to be a nondeterministic one. Hence we come to DFAs with multiple initial states (MDFAs) which were introduced in [4] and recently studied in [8] and [2]. We will show that the minimization problem of such MDFAs is NP-complete even if only two initial states are given. In analogy to NFAs with fixed finite branching, certain decidability questions can be shown to be efficiently solvable.

2. Definitions

Concerning the definitions of NFAs with finite branching and MDFAs we follow the notations introduced in [1] and [8].

A nondeterministic finite automaton over Σ is a tuple $M = (Q, \Sigma, \delta, q_0, F)$, with Q a finite set of states, $q_0 \in Q$ the initial state, $F \subseteq Q$ the set of accepting states, and δ a function from $Q \times \Sigma$ to 2^Q. A move of M is a triple $\mu = (p, a, q) \in Q \times \Sigma \times Q$ with $q \in \delta(p, a)$. A computation for $w = w_1 w_2 \dots w_n \in \Sigma^*$ is a sequence of moves $\mu_1 \mu_2 \dots \mu_n$ where $\mu_i = (q_{i-1}, w_i, q_i)$ with $1 \leq i \leq n$. It is an accepting computation if $q_n \in F$. The language accepted by M is $T(M) = \{w \in$

$\Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\}$. M is an (incomplete) deterministic finite automaton if $|\delta(q, a)| \leq 1$ for all pairs (q, a). The branching $\beta_M(\mu)$ of a move $\mu = (q, a, p)$ is defined to be $\beta_M(\mu) = |\delta(q, a)|$. The branching is extended to computations $\mu_1\mu_2 \dots \mu_n$, $n \geq 0$, by setting $\beta_M(\mu_1\mu_2 \dots \mu_n) = \beta_M(\mu_1) \cdot \beta_M(\mu_2) \cdot \dots \cdot \beta_M(\mu_n)$. For each word $w \in T(M)$, let $\beta_M(w) = \min \beta_M(\mu_1\mu_2 \dots \mu_n)$ where $\mu_1\mu_2 \dots \mu_n$ ranges over all accepting computations of M with input w. The branching β_M of the automaton M is $\beta_M = \sup \{\beta_M(w) \mid w \in T(M)\}$. The set of all NFAs with branching $\beta = k$ is defined as $\text{NFA}(\beta = k) = \{M \mid M \text{ is NFA and } \beta_M = k\}$.

A DFA with multiple initial states (MDFA) is a tuple $M = (Q, \Sigma, \delta, Q_0, F)$ and M is identical to a DFA except that there is a set of initial states Q_0. The language accepted by an MDFA M is $T(M) = \{w \in \Sigma^* \mid \delta(Q_0, w) \cap F \neq \emptyset\}$. An MDFA with $k = |Q_0|$ initial states is denoted by k-MDFA.

Let $\mathcal{A}, \mathcal{B}$ be two classes of finite automata. Following the notation of [7], we say that $\mathcal{A} \longrightarrow \mathcal{B}$ denotes the problem of converting a type-$\mathcal{A}$ finite automaton to a minimal type-$\mathcal{B}$ finite automaton. Formally:

PROBLEM	$\mathcal{A} \longrightarrow \mathcal{B}$
INSTANCE	A type-$\mathcal{A}$ finite automaton M and an integer l.
QUESTION	Is there an l-state type-$\mathcal{B}$ finite automaton M' such that $T(M') = T(M)$?

3. Minimizing Finite Automata Is Computationally Hard

3.1. Results for MDFAs

Theorem 1 *k-MDFA $\longrightarrow$ k-MDFA is* NP*-complete.*

Corollary 2 *Let $k, k' \geq 2$ be two constant numbers. Then the problems DFA $\longrightarrow$ k-MDFA and k-MDFA $\longrightarrow$ k'-MDFA are* NP*-complete.*

Theorem 3 *Let M be a k-MDFA and M' be a k'-MDFA. Then the following problems are solvable in polynomial time. Is $T(M) = T(M')$? Is $T(M) \subseteq T(M')$? Is $T(M) \subset T(M')$? Is $T(M) = \Sigma^*$?*

3.2. Results for NFAs with Fixed Finite Branching

Lemma 4 *Let M be an NFA and $k \geq 2$ be a constant integer. Then the question whether M has branching k can be solved in polynomial time.*

Theorem 5 *NFA($\beta = k$) $\longrightarrow$ NFA($\beta = k$) is* NP*-complete for $k \geq 3$.*

Corollary 6 *Let $k \geq 2$ and $k' \geq 3$ be constant integers. Then DFA $\longrightarrow$ NFA($\beta = k'$) and NFA($\beta = k$) $\longrightarrow$ NFA($\beta = k'$) are* NP*-complete.*

Theorem 7 *The following problems, which are* PSPACE*-complete when arbitrary NFAs are considered, are solvable in polynomial time.*

1. *Given two NFAs M, M' with $\beta_M = k$ and $\beta_{M'} = k'$. Is $T(M) = T(M')$? Is $T(M) \subseteq T(M')$? Is $T(M) \subset T(M')$? Is $T(M) = \Sigma^*$?*
2. *Given any NFA M and an NFA M' with $\beta_{M'} = k$. Is $T(M) \subseteq T(M')$?*

The results are summarized in Table 1.

	DFA	UFA	k-MDFA	NFA($\beta = k$)	NFA
emptiness	P	P	P	P	P
equivalence	P	P	P	P	PSPACE-complete
inclusion	P	P	P	P	PSPACE-complete
universality	P	P	P	P	PSPACE-complete
minimization	P	NP-complete	NP-complete	NP-complete	PSPACE-complete

Table 1: Computational complexity results

References

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

[2] M. Holzer, K. Salomaa, S. Yu, On the state complexity of k-entry deterministic finite automata, *Journal of Automata, Languages and Combinatorics* **6** (2001), 453–466.

[3] M. R. Garey, D. S. Johnson, *Computers and Intractability*. W.H. Freeman and Co., San Francisco, 1979.

[4] A. Gill, L. Kou, Multiple-entry finite automata, *Journal of Computer and System Sciences* **9** (1974), 1–19.

[5] J. E. Hopcroft, An n log n algorithm for minimizing states in a finite automaton. In: Z. Kohavi (ed.), *Theory of machines and computations*. Academic Press, New York, 1971, 189–196.

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

[7] T. Jiang, B. Ravikumar, Minimal NFA problems are hard, *SIAM Journal on Computing* **22** (1993), 1117–1141.

[8] M. Kappes, Descriptional complexity of deterministic finite automata with multiple initial states, *Journal of Automata, Languages and Combinatorics* **5** (2000), 269–278.

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

[10] A. R. Meyer, M. J. Fischer, Economy of descriptions by automata, grammars, and formal systems. In: *IEEE Symposium on Foundations of Computer Science*, 1971, 188–191.

[11] L. Stockmeyer, A. R. Meyer, Word problems requiring exponential time: preliminary report. In: *Fifth Annual ACM Symposium on Theory of Computing*, 1973, 1–9.

[12] S. Yu, Regular languages. In: G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages Vol. 1.* Springer-Verlag, Berlin, 1997, 41–110.

HIERARCHIES OF WEAKLY MONOTONE RESTARTING AUTOMATA

František Mráz[1]
Department of Computer Science, Charles University
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
e-mail: mraz@ksvi.ms.mff.cuni.cz

and

Friedrich Otto[2]
Fachbereich Mathematik/Informatik, Universität Kassel
34109 Kassel, Germany
e-mail: otto@theory.informatik.uni-kassel.de

ABSTRACT

It is known that the weakly monotone restarting automata accept exactly the growing context-sensitive langauges. We introduce a measure on the degree of weak monotonicity and show that the language classes obtained in this way form strict hierarchies for the various types of deterministic and nondeterministic restarting automata without auxiliary symbols.

Keywords: Restarting automata, growing context-sensitive languages, degree of weak monotonicity.

1. Introduction

The motivation for introducing the restarting automaton in [2] was the desire to model the so-called *analysis by reduction* of natural languages [12, 13]. This analysis consists of a stepwise simplification of a sentence in such a way that the syntactical correctness or incorrectness of the sentence is not affected. After a finite number of steps either a correct simple sentence is obtained, or an error is

[2]The first author was partially supported by the Grant Agency of the Czech Republic, Grant-No. 201/02/1456, and the Grant Agency of Charles University, Grant No. 300/2002/A-INF/MFF.

[2]The work of the second author was partially supported by a grant from the Deutsche Forschungsgemeinschaft (DFG).

detected. It turned out that the restarting automaton thus obtained can handle a class of languages that is much broader than the class CFL of context-free languages.

A restarting automaton, or RRWW-automaton, M is a device with a finite state control and a read/write window of fixed size. This window moves from left to right along a tape containing a string delimited by sentinels until M's control decides (nondeterministically) that the contents of the window should be rewritten by some shorter string. After a rewrite, M continues to move its window to the right until it either halts and accepts, or halts and rejects, which means that it has reached a configuration from which it cannot continue, or restarts, that is, it moves its window to the leftmost position, enters the initial state, and continues with the computation.

Also some restricted variants of the restarting automaton have been considered. First, there are the deterministic variants. Then there are those variants that are allowed to only use the letters from the input alphabet, while in general a restarting automaton can use a finite number of auxiliary symbols in its rewrite steps. Further, a monotonicity property was introduced for RRWW-automata, and it was shown that the monotone RRWW-automata (with auxiliary symbols) characterize the class CFL, and that various restricted versions of deterministic monotone RRWW-automata (with or without auxiliary symbols) characterize the class DCFL of deterministic context-free languages [3].

In [5, 7] the class CRL of Church-Rosser languages was introduced motivated by the fact that membership for these languages is decidable in linear time. In [10] it was then shown that the deterministic RRWW-automata (with auxiliary symbols) give another characterization of CRL. For the growing context-sensitive languages GCSL [1], which can be seen as the nondeterministic variants of the Church-Rosser languages [8], it was observed in [9] that they form a proper subclass of the class $\mathcal{L}$(RRWW) of languages that are accepted by RRWW-automata.

In order to obtain a characterization of GCSL in terms of restarting automata, a relaxation of the monotonicity condition for RRWW-automata was introduced in [4]. Let c be a non-negative integer. An RRWW-automaton M is called *weakly c-monotone* if in any two consecutive rewrite steps of any computation of M the places of rewriting can increase their distances from the right end of the tape by at most c positions. An RRWW-automaton M is called *weakly monotone* if there exists a non-negative integer c such that M is weakly c-monotone. The weakly monotone RRWW-automata (with auxiliary symbols) recognize exactly GCSL [4]. For a recent survey on the restarting automata see [11].

Here we study the influence of the degree of weak monotonicity on the expressive power of the various models of restarting automata. We focus on the restarting automata with the most transparent computations – those without auxiliary symbols. Further motivation for studying weak monotonicity comes from linguistics. The degree of weak monotonicity is a measure of across how many symbols (words of a sentence) already read before a reader must step back

to the left during an analysis of a sentence in a certain (natural) language.

2. Definitions

A *restarting automaton*, RRWW-automaton for short, is a one-tape machine that is described by an 8-tuple $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$, where Q is the finite set of states, Σ is the finite input alphabet, Γ is the finite tape alphabet containing Σ, $¢, \$ \notin \Gamma$ are the markers for the left and right border of the work space, respectively, $q_0 \in Q$ is the initial state, $k \geq 1$ is the size of the *read/write window*, and

$$\delta : Q \times \mathcal{PC}^{(k)} \to \mathcal{P}((Q \times (\{\mathsf{MVR}\} \cup \mathcal{PC}^{(k-1)})) \cup \{\mathsf{Restart}, \mathsf{Accept}\})$$

is the *transition relation*. Here $\mathcal{P}(\mathcal{S})$ denotes the powerset of the set S, and

$$\mathcal{PC}^{(k)} := (¢ \cdot \Gamma^{k-1}) \cup \Gamma^k \cup (\Gamma^{\leq k-1} \cdot \$) \cup (¢ \cdot \Gamma^{\leq k-2} \cdot \$)$$

is the set of *possible contents* of the read/write-window of M, where $\Gamma^{\leq n} := \bigcup_{i=0}^{n} \Gamma^i$, and $\mathcal{PC}^{(k-1)}$ is defined analogously.

The transition relation describes four different types of transition steps:

1. A *move-right step* is of the form $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u \in \mathcal{PC}^{(k)}$, $u \neq \$$. If M is in state q and sees the string u in its read/write-window, then this move-right step causes M to shift the read/write-window one position to the right and to enter state q'.
2. A *rewrite step* is of the form $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u \in \mathcal{PC}^{(k)}$, $u \neq \$$, and $v \in \mathcal{PC}^{(k-1)}$ such that $|v| < |u|$. It causes M to replace the contents u of the read/write-window by the string v and to enter state q'. Further, the read/write-window is placed immediately to the right of the string v. However, some additional restrictions apply in that the border markers $¢$ and \$ must not disappear from the tape nor that new occurrences of these markers are created. Further, the read/write-window must not move across the right border marker \$.
3. A *restart step* is of the form $\mathsf{Restart} \in \delta(q, u)$, where $q \in Q$ and $u \in \mathcal{PC}^{(k)}$. It causes M to move its read/write-window to the left end of the tape, so that the first symbol it sees is the left border marker $¢$, and to reenter the initial state q_0.
4. An *accept step* is of the form $\mathsf{Accept} \in \delta(q, u)$, where $q \in Q$ and $u \in \mathcal{PC}^{(k)}$. It causes M to halt and accept.

Obviously, each computation of M proceeds in cycles. Starting from an initial configuration $q_0 ¢ w \$$, the head moves right, while move-right and rewrite steps are executed until finally a restart step takes M back into a configuration of the

form $q_0 \cent w_1 \$$. It is required that in each such cycle *exactly one* rewrite step is executed. By $\vdash_M^c$ we denote the execution of a complete cycle. As by a rewrite step the contents of the tape is shortened, only a linear number of cycles can be executed within any computation. That part of a computation of M that follows after the execution of the last restart is called the *tail* of the computation. It contains at most a single application of a rewrite step.

An input $w \in \Sigma^*$ is accepted by M, if there exists a computation of M which starts with the initial configuration $q_0 \cent w \$$, and which finally ends with executing an accept step. By $L(M)$ we denote the *language accepted by* M, and $\mathcal{L}(\mathsf{RRWW})$ will denote the class of languages that are accepted by RRWW-automata.

In [3] a notion of monotonicity is considered for restarting automata. Let M be an RRWW-automaton. Each computation of M can be described by a sequence of cycles $C_1, C_2 \ldots, C_n$, where C_1 starts with an initial configuration of M, and C_n is the last cycle, which is followed by the tail of the computation. Each cycle C_i contains a unique configuration of the form $\cent xuy\$$ such that $u \to v$ is the rewrite step applied during this cycle. By $D_r(C_i)$ we denote the *r-distance* $|y\$|$ of this cycle. The sequence of cycles $C_1, C_2, \ldots, C_n$ is called *monotone* if $D_r(C_1) \geq D_r(C_2) \geq \ldots \geq D_r(C_n)$ holds, and the RRWW-automaton M is called *monotone* if all its computations are monotone.

In [3] it is shown that all variants of deterministic monotone restarting automata accept exactly the deterministic context-free languages. For the nondeterministic restarting automata it turned out that the use of auxiliary symbols is necessary to obtain a characterization of the class of context-free languages.

In order to derive a characterization of the class GCSL of growing context-sensitive languages in terms of restarting automata, the notion of weak monotonicity was introduced in [4]. Let M be an R(R)WW-automaton, and let $c \geq 0$ be an integer. We say that a sequence of cycles $C_1, C_2, \ldots, C_n$ is *weakly c-monotone*, if $D_r(C_{i+1}) \leq D_r(C_i) + c$ holds for all $i = 1, 2, \ldots, n-1$. An RRWW-automaton M is called *weakly c-monotone* if all its computations are weakly c-monotone. Further, we say that M is *weakly monotone*, if there exists a constant $c \geq 0$ such that M is weakly c-monotone. The prefixes w(c)mon- and wmon- are used to denote the corresponding classes of restarting automata.

For deterministic RRWW-automata we have the following observation.

Lemma 1 *Each deterministic* RRWW*-automaton with window size k is weakly j-monotone for some $j \leq k-2$.*

Thus, for deterministic R(R)WW-automata the above weak monotonicity condition is always satisfied, that is,

$$\mathsf{CRL} = \mathcal{L}(\text{det-wmon-RWW}) = \mathcal{L}(\text{det-wmon-RRWW}).$$

Hence, it is only for the various nondeterministic restarting automata that these

additional restrictions make a difference. In [4] it is shown that

$$\mathsf{GCSL} = \mathcal{L}(\mathsf{wmon\text{-}RWW}) = \mathcal{L}(\mathsf{wmon\text{-}RRWW}) \subset \mathcal{L}(\mathsf{RWW}),$$

where the inclusion is known to be proper.

3. Hierarchies with respect to the degree of weak monotonicity

Here we are interested in the influence of the degree of weak monotonicity on the expressive power of the various types of restarting automata. It can be shown by specifically chosen example languages that the expressive power of the various types of deterministic and nondeterministic restarting automata without auxiliary symbols strictly increases when the degree of weak monotonicity grows.

Theorem 2 [6] *For each* $i \geq 0$ *and for each* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}\}$,

(a) $\mathcal{L}(\mathsf{det\text{-}w}(i)\mathsf{mon}{-}\mathsf{X}) \subseteq \mathcal{L}(\mathsf{det\text{-}w}(i+1)\mathsf{mon}{-}\mathsf{X})$.

(b) $\mathcal{L}(\mathsf{w}(i)\mathsf{mon}{-}\mathsf{X}) \subseteq \mathcal{L}(\mathsf{w}(i+1)\mathsf{mon}{-}\mathsf{X})$.

Further, it can be shown that these hierarchies differ pairwise from each other. These results can best be presented by the following diagram, where an arrow $A \longrightarrow B$ indicates that the class $\mathcal{L}(A)$ is strictly included in the class $\mathcal{L}(B)$. When there is no oriented path between any two types of restarting automata in the diagram, then the corresponding language classes are incomparable under inclusion.

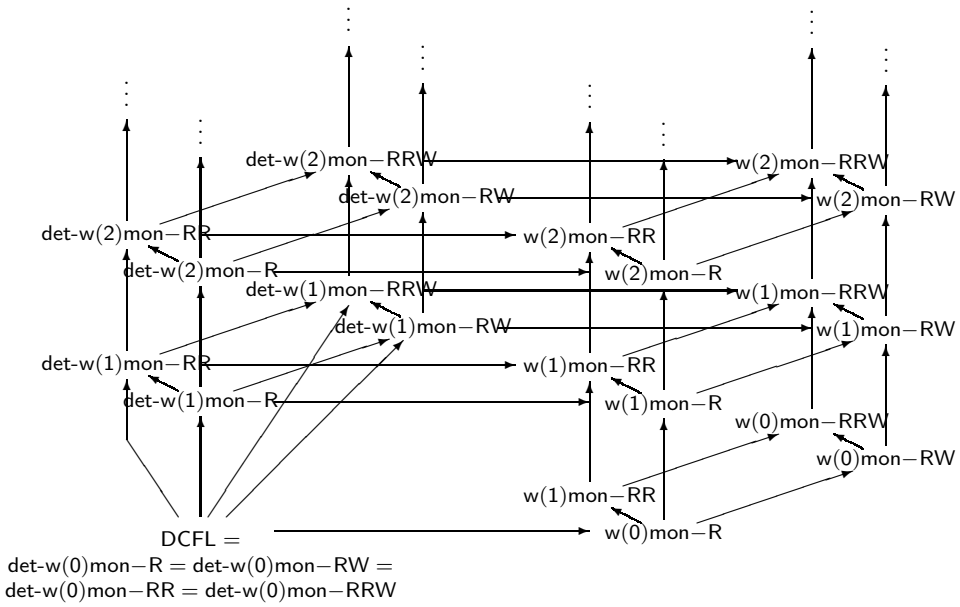


Figure 1: Hierarchies of language classes defined by the various types of weakly monotone restarting automata without auxiliary symbols.

4. Conclusions

By considering the degree of weak monotonicity we obtain infinite hierachies that lie in the gap between DCFL and CRL in the deterministic case and that lie in the gap between DCFL and GCSL in the nondeterministic case. Naturally, the question arises whether the degree of weak monotonicity induces corresponding hierarchies for the restarting automata with auxiliary symbols. We would certainly expect that. The class $\mathcal{L}(\mathsf{det\text{-}w(1)mon{-}RWW})$ is strictly larger than $\mathcal{L}(\mathsf{det\text{-}w(0)mon{-}RWW}) = \mathsf{DCFL}$, and also $\mathcal{L}(\mathsf{w(1)mon{-}RWW})$ strictly contains $\mathcal{L}(\mathsf{w(0)mon{-}RWW}) = \mathsf{CFL}$, because $\mathcal{L}(\mathsf{det\text{-}w(1)mon{-}RWW})$ contains non-context-free languages.

References

[1] E. Dahlhaus and M. K. Warmuth. Membership for growing context-sensitive grammars is polynomial. *J. Comput. Sys. Sci.* 33 (1986) 456–472.

[2] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. Restarting automata. In H. Reichel (ed.), *Proc. FCT'95*, *LNCS* 965, 283–292. Springer, Berlin, 1995.

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

[4] T. Jurdziński, K. Loryś, G. Niemann, and F. Otto. Some results on RWW- and RRWW-automata and their relationship to the class of growing context-sensitive languages. Tech. Report 14/01, Fachb. Mathematik/Informatik, Universität Kassel, 2001.

[5] R. McNaughton, P. Narendran, and F. Otto. Church-Rosser Thue systems and formal languages. *J. Assoc. Comput. Mach.* 35 (1988) 324–344.

[6] F. Mráz and F. Otto. Hierachies of weakly monotone restarting automata. Tech. Report 8/03, Fachb. Mathematik/Informatik, Universität Kassel, 2003.

[7] Narendran, P.: Church-Rosser and related Thue systems. PhD thesis, Rensselaer Polytechnic Institute, Troy, New York, 1984.

[8] G. Niemann and F. Otto. The Church-Rosser languages are the deterministic variants of the growing context-sensitive languages. In M. Nivat (ed.), *Proc. FoSSaCS'98*, *LNCS* 1378, 243–257. Springer, Berlin, 1998.

[9] G. Niemann and F. Otto. On the power of RRWW-automata. In M. Ito, G. Păun, and S. Yu (eds.), *Words, Semigroups, and Transductions*, 341–355. World Scientific, Singapore, 2001.

[10] G. Niemann and F. Otto. Further results on restarting automata. In M. Ito and T. Imaoka (eds.), *Words, Languages and Combinatorics III, Proc.*, 352–369. World Scientific, Singapore, 2003.

[11] F. Otto. Restarting automata and their relations to the Chomsky hierarchy. In Z. Esik and Z. Fülöp (eds.), *Developments in Language Theory, Proc. DLT 2003*, *LNCS* 2710, 55–74. Springer, Berlin, 2003.

[12] M. Straňáková. Selected types of pg-ambiguity. The Prague Bulletin of Mathematical Linguistics **72** (1999) 29–57

[13] M. Straňáková. Selected types of pg-ambiguity: Processing based on analysis by reduction. In Sojka, P., Kopeček, I., Pala, K. (eds.), Text, Speech and Dialogue, 3rd Int. Workshop, Proceedings, *LNCS* 1902, 139–144. Springer, Berlin, 2000.

RESTARTING AUTOMATA: MOTIVATIONS AND APPLICATIONS

Martin Plátek
KTIML MFF, Charles University, Prague,
Malostranské nám. 25, CZ-118 00 Praha 1 - Malá Strana, Czech Republic
e-mail: platek@ksi.ms.mff.cuni.cz

Markéta Lopatková
Centre for Computational Linguistics, MFF, Charles University, Prague,
Malostranské nám. 25, CZ-118 00 Praha 1 - Malá Strana, Czech Republic
e-mail: lopatkova@ckl.ms.mff.cuni.cz

and

Karel Oliva
Austrian Research Institute for Artificial Intelligence (OeFAI),
Freyung 6, A-1010 Wien, Austria
and
Institute of the Czech Language, Academy of Sciences of the Czech Republic,
Letenská 4, CZ-110 00 Praha 1 - Malá Strana, Czech Republic
e-mail: karel@oefai.at, oliva@ujc.cas.cz

ABSTRACT

Automata with restart operation (henceforth RA) constitute an interesting class of acceptors and reduction systems at the same time, and simultaneously display a remarkable *'explicative power'* wrt. the analysis of formal and natural languages. This paper focuses on RA as a formal-theoretical tool for modelling the *analysis by reduction* of natural languages and on the role of analysis by reduction (i.e. of analysis by RA) in dependency approach to syntax, and it also outlines some practical applications.

1. Restarting automata – bottom up analysers

The aim of this contribution is to present the main motivations for study of restarting automata and outline some applications of restarting automata in linguistics in an informal way. All the formal notions can be found in [1] or in [3], which also contains the bibliographic data.

A *restarting automaton* M can be described as a non-deterministic device with a finite control unit and a head with a lookahead window attached to a linear list. The automaton works in *cycles*. Within one cycle, it performs the following sequence of actions:

- *moves* the head along the word on the list (possibly in both directions);
- *rewrites* — a limited number of times within one cycle — the contents of the list within the scope of its lookahead by another (usually shorter) string defined by instructions of the automaton;
- *continues* by scanning some further symbols;
- *"restarts"* – i.e. resets the control unit into the initial state and places the head on the left end of the (shortened) word.

The computation halts when M enters an accepting or a rejecting state.

M can be viewed as a *recognizer* and as a (regulated) *reduction system*, as well. A *reduction* by M is defined trough the rewriting operations performed during one individual cycle by M, in which M distinguishes two alphabets: the input alphabet and the working alphabet.

For any input word w, M induces a set (due to non-determinism) of sequences of reductions. We call this set the *complete analysis* of w by M ($CA(w, M)$). $CA(w, M)$ is composed of accepting sequences and rejecting sequences of reductions. If, for a given w, $CA(w, M)$ contains at least one accepting sequence, w is accepted. Otherwise, w is rejected.

¿From the techniques summarized in [1] and quoted in [3] it follows that restarting automata define a taxonomy for types of *regulation* of bottom-up syntactic analysers (parsers) for several types of (originally unregulated) grammars (e.g., CFGs, pure grammars, Marcus grammars, categorial grammars, and for variants of these allowing for discontinuous constituency).

Restarting automata enable direct modelling of correct syntactic phenomena as well as of syntactic incorrectness. This is crucial since in the current techniques of computational linguistics the positive and negative observations (constraints) play equally important roles. Let this be explained in more detail. Syntactic observations of natural languages are collected stepwise, in a slow and uneasy way. At any moment of the development of a realistic syntactic description, the collection of positive syntactic observations is far from complete and precise. The same, then, holds also for negative observations. Because of lack of initial completness and precision, both types of observations have to be collected iteratively and gradually integrated into the analysers (grammars), and their formal description then individually verified and debugged. Such an approach, in turn, can be efficiently supported by (implementing the analysers by means of) restarting automata.

2. Restarting automata – analysis by reduction

A special (restricted) type of restarting automata is the RLW-automata, cf. [3].

The operation of an RLW-automaton can be viewed as implementation of a set of meta-instructions, each of which describes the actions to be performed within an individual cycle of the automaton. Each meta-instruction has the form $(R_1, u \rightarrow v, R_2)$, where R_1, R_2 are regular expressions, u, v are strings over the input alphabet ($|v| < |u|$), and $u \rightarrow v$ is a rewriting rule. In one cycle, the RLW-automaton rewrites (according to the meta-instruction) the contents u of its lookahead by the string v (only one rewriting operation can be performed within one cycle). An important property of a meta-instruction is that it can be verified (debugged) individually.

A very important feature of operation of RLW-automata is the "error preserving property": if an input sentence is incorrect then the reduced sentence is also incorrect (after any number of cycles). In addition, deterministic RLW-automata have "correctness preserving property": if a sentence is correct then the reduced sentence is correct as well. Obviously, both these properties are important for the description (and build-up of syntactic structures) of natural languages.

Due to the above, the RLW-automata can serve as a framework for formalization of the analysis method called *analysis by reduction* (AR), which constitutes a natural and above all solid basis for building dependency grammars of natural languages.

In particular, AR helps to find (define) the relations of:

- *vertical dependency*, which holds in case one of the two words constituting the dependency pair can be omitted without harming the grammaticality of the sentence, while the other word cannot be omitted;
- *horizontal dependency*, which holds if none of the two words constituting the dependency pair can be omitted without making the result ungrammatical;
- *coordination* and other non-dependency relations;

as well as to state which pairs of words have no syntactic relation to each other.

Let it be remarked that in the visualization of dependency trees (grammars), both vertical and horizontal dependencies are usually represented as one type, e.g., as oblique dependencies (see the Figs. below for examples of dependency trees). The transformation of vertical dependencies into the oblique dependencies is trivial. On the other hand, in order to obtain the oblique dependencies from the horizontal ones (i.e. to determine the 'governors' of these dependencies) finer methods are to be used, (e.g., analogy).

Let us outline some principles of the analysis by reduction and its relation to syntactic (in)dependencies and coordinations:

- The AR consists of stepwise simplification of a complete sentence in such a way that its syntactic correctness is preserved; at the end, the 'core' of the sentence is obtained.
- The reduction is the basic operation of AR: in each step (cycle) the simplification is performed by deleting at least one word of the sentence and possibly rewriting other words in the sentence. The deletion is 'justified' by

linguistically motivated rules.

- The process is non-deterministic, mutually *independent* words (parts) can be deleted in any order (preserving syntactic correctness is the only criterion for the success of the reduction).
- All the *vertically dependent* words (parts) must be deleted before their governor (from the corresponding dependency structure) is deleted; all the *horizontally dependent* words must be deleted simultanously (i.e. within one cycle).
- Two parts of a sentence can be considered as a *coordination* only if these parts are not (syntactically) dependent on each other.

The following three examples illustrate the way how dependency structures are obtained via *AR*. Note the contrast between the (superficial) similarity of the three sample sentences and the differences among the resulting types of *AR*'s.

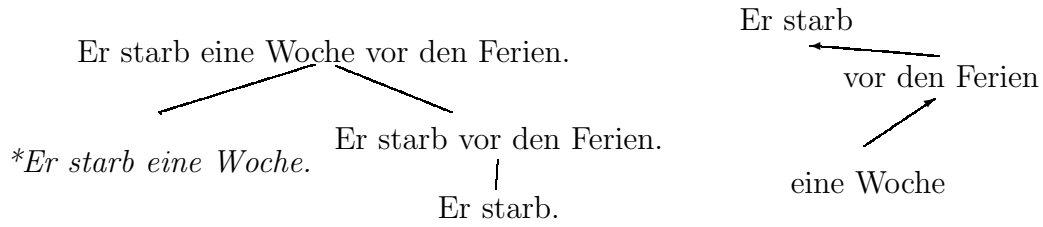


Figure 1: First example of AR and the corresponding dependency structure.

The sentence in Fig.1 can undergo two different reductions in the first reduction step (in the left part of Fig.1): either the deletion of the string of horizontally dependent words *vor den Ferien* or the deletion of the horizontally dependent pair *eine Woche*. In the first case the syntactic correctness of the sentence is violated (which is marked off by an asterisk and italics – in this case the reduction fails), in the second case the correctness is preserved. The second reduction step is then determined unambiguously: it consists in the deletion of the words *vor den Ferien*. In such a way we have obtained only one correct sequence of reductions and one incorrect reduction. This result determines unambiguously the dependency structure from the right part of Fig.1. The oblique (in this example, originally vertical) dependencies are represented by arrows, the horizontal dependencies are represented by the remaining strings of words in the 'nodes'.

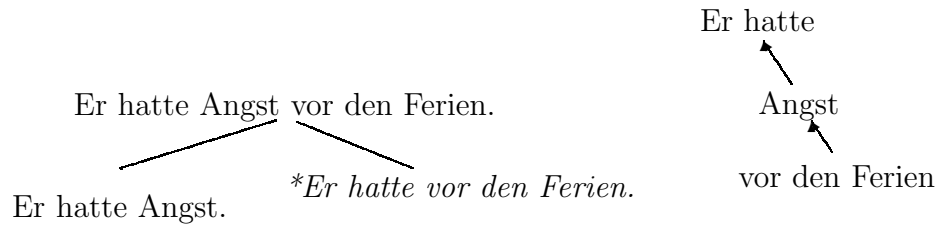


Figure 2: Second example of AR and the corresponding dependency structure.

Let us consider the second example. In this case only the first reduction is correct, while the second one yields an incorrect string. In this way we obtain only one arrow denoting a vertical dependency in the resulting dependency structure (right half of the picture). It is the one between the nodes with *Angst* and *vor den Ferien*. The other one (originally horizontal), between *Er hatte* and *Angst*, is obtained by analogy, i.e. via the observation that the parts with finite verbs occur "usually" on the top of a dependency structure, similarly as in the previous example.

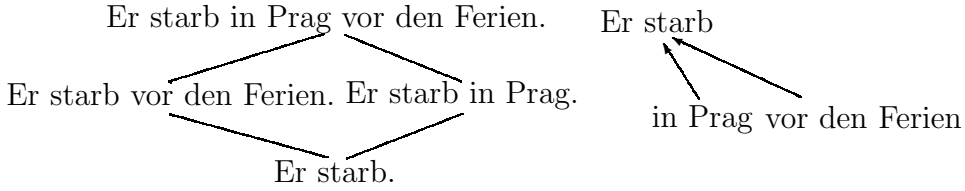


Figure 3: The example of AR for an independence.

The third example serves as an example of how to discover independence between two parts of the sentence. The deletions of the strings *in Prag* and *vor den Ferien* can be performed in both orders. It means that this two pairs are mutually (syntactically) independent (and, in this particular case, are both dependent on the finite verb).

It is the aim of the approach described here to find transformations from restarting automata modelling *AR* into dependency analyzers.

3. Two other applications: disambiguation and grammar-checking

Among the topical problems of current corpus linguistics (i.e. the branch of linguistics dealing with large bodies of running texts - so called *corpora*) is the *resolution of morphological ambiguity* of words in a text corpus - either on the level of Part-of-Speech (PoS) of a word (e.g., the German wordform *sein* has to be disambiguated between the pronominal and the verbal reading) or on some more fine-grained level (e.g., the noun *Leiter*, ambiguous between feminine and masculine gender). There exist different approaches to this *disambiguation*, including statistical ones, memory-based learning ones etc. The most accurate among them (at least up to date) is the rule-based disambiguation. This kind of disambiguation presupposes preprocessing the text by a morphological analyser, which assigns the set of all its potential morphological readings to each word of the input. In the main processing phase, the disambiguation then employs explicit rules (created by linguists) for the (stepwise) elimination of those readings which are impossible in the context of the word token. In particular, the elimination of a reading of a word relies on finding (for the very reading) a *disambiguation context*, which is to be understood as a context in which a particular

reading of a particular word is impossible (and hence the reading can be safely removed, by means of which disambiguation is performed). Thus, e.g., in the clause *weil niemand meinen Eltern einen lebenswürdigen Unterhalt versprechen konnte* the finite verb reading of *meinen* can be eliminated based on the occurrence of the unambiguous finite verb *konnte* and on a rule postulating that in any correct German sentence, a comma or a coordinating conjunction must occur somewhere between two finite verbs (for a complete disambiguation, an application of another rule, leaning on some other context, would be needed in order to remove the infinitival reading of *meinen*). While the creation of such rules is a linguistic-theoretical task, it is quite obvious that their application is an issue which is in fact identical to the idea of restarting automata. This can be seen from the following two points:

- each rule looks for the configuration it describes (i.e. for a particular reading with its context), and when such a configuration is detected, the rule serves for deleting the particular reading (leaving the context untouched); after the application of the rule, the whole process has to start again on the newly defined input (i.e. input after the deletion), using some other rule (or maybe the same one on another configuration within the sentence)

- successful application of one rule can trigger the application of another rule (the first rule disambiguates and hence gives rise to a context needed for the application of the second one); the subtle point here is that this is obviously dependent not only on the rules, but also on the sentence to be disambiguated: two rules R_1 and R_2 applied in this order can work well for the sentence S_1, while for another sentence S_2 this order does not work, but the order R_2, R_1 would work. This problem has a straightforward and easy solution, however: given a set of rules $R_1, R_2, ..., R_n$, this set should be applied such that all possible rule permutations (non-determinism) are simulated. Thus, any sentence may be processed, since also the order needed for this sentence is applied (if all are, then also the one needed) - and this is exactly what a non-deterministic restarting automaton with deletion does.

Let it be mentioned in passing that the approach described above for disambiguation can also be used for grammar-checking purposes. In particular, if the morphological analysis and the set of disambiguation rules are applied on a sentence, and if this results in deleting all readings of one word supplied by the morphological analyser, then it is obvious that the readings of this word are in disaccord with the readings of other words of the sentence, i.e. the sentence is syntactically ill-formed. Thus, the corresponding restarting automaton is in fact able to discover (at least some) cases of ill-formed sentences, and in fact also the source of the error can be located as the "last configuration before deletion" (at least in a "reasonable" number of cases). Some grammar-checking techniques based on restarting automata are outlined in [2] and some new techniques of *error recovery* are currently studied.

Acknowledgements

We are indebted to J. Panevová for stimulating discussions concerning analysis by reduction. M. Plátek and M. Lopatková are supported by grants No. 201/02/1456 of GAČR and LN00A063 of MŠMT ČR. K. Oliva is supported by Grant No. P16614-G03 of FWF. OeFAI is supported by the Austrian Federal Ministry of Education, Science and Culture.

References

[1] P. Jančar, F. Mráz, M. Plátek, J. Vogel: On Monotonic Automata with a Restart Operation. *J. Autom. Lang. Comb.* 4 (1999), 287-311.

[2] V. Kuboň, M. Plátek: A Grammar Based Approach to Grammar Checking of Free Word Order Languages, In: *Proceedings COLING 94, Vol.II,* Kyoto, August, 1994, 906 - 910.

[3] F. Otto: Restarting Automata And Their Relations to the Chomsky hierarchy. In: Z. Esik, Z. Fülöp (eds.): *Developments in Language Theory, Proceedings of DLT'2003*, LNCS 2710, Springer Verlag, Berlin, 2003, 55 - 74.

[4] K. Oliva, P. Květoň and R. Ondruška: The Computational Complexity of Rule-Based Part-of-Speech Tagging. To appear in: *Proceedings of TSD 2003*, to be published in the LNAI Series by Springer Verlag.

SOME MORE REGULAR LANGUGAES THAT ARE CHURCH ROSSER CONGRUENTIAL

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

and

Denis Thérien
School of Computer Science, McGill University
e-mail: denis@cs.mcgill.ca

ABSTRACT

We show that those languages, where the syntactic monoid is a finite group, are Church Rosser Congruential Languages (CRCL). We then extend this result to the class of regular languages whose syntactic monoid lies in the variety DO.

1. Introduction

In [1] McNaughton et al considered finite, length-reducing and confluent string-rewriting systems, which allow to find a unique irreducible shortest representing word for any given word: the Church-Rosser congruential languages (CRCL) are defined as the set of languages, which are the union of finitely many such equivalence classes. This is defined formally as follows:

A language $L \in A^*$ is in CRCL if there exists a rewriting system $R \subseteq \{l \mapsto r \mid l, r \in A^*, |l| > |r|\}$ allowing derivations

$\alpha l \beta \underset{R}{\Longrightarrow} \alpha r \beta$ with the property that if $w \overset{*}{\underset{R}{\Longrightarrow}} w'$ and $w \overset{*}{\underset{R}{\Longrightarrow}} w''$ then there is a $v \in A^*$ with $w' \overset{*}{\underset{R}{\Longrightarrow}} v$ and $w'' \overset{*}{\underset{R}{\Longrightarrow}} v$ and L is the finite union of sets $[w_e]_R = \{w \mid w \overset{*}{\underset{R}{\Longrightarrow}} w_e\}$ for finitely many w_e.

It is an open problem weather all regular languages are in CRCL.

It was shown in [2] that some shuffle languages as well as Level 1 of the Straubing-Thérien hierarchy are in CRCL. Furthermore [2] describes a solution for the language $(A^2)^*$ (words of even length on alphabet $A = \{a, b\}$), which is

the smallest nontrivial example for our result since the syntactic monoid for $(A^2)^*$ is the cyclic group $\mathbb{Z}_2$.

Our approach is as follows: We refine the *syntactic monoid* $M(L) := \Sigma^*/\equiv_L$ for the congruence relation $\equiv_L$ defined by $w \equiv_L v$ iff $\forall u, x \in \Sigma^* : uwx \in L \leftrightarrow uvx \in L$. The refinement is defined by the construction of a confluent length-reducing string-rewriting system $R \subset \equiv_L$ with the property that words exceeding some length contain an infix l with $l \mapsto r \in R$, which leads to a finite $\equiv_R \supseteq \equiv_L$ having the property that every congruence class has a unique shortest representing word.

For the example $(A^2)^*$ the string-rewriting system $R = \{aa \mapsto \lambda, bab \mapsto b, bbb \mapsto b\}$ leads to the syntactic monoid consisting of the 10 elements $[\lambda], [ab], [ba], [bb], [abba], [a], [b], [aba], [abb], [bba]$, where two words are equivalent, if their length, the number of a's before the first b and the number of a's after the last b differ by an even number and both contain or do not contain a b. The first 5 refine $[\lambda]$ and the last 5 refine $[a] = [b]$ (two minimal representing strings since $|a| = |b|$, which was the reason which made the refinement necessary).

2. The construction for a group

Theorem *If the syntactic monoid $M(L)$ is a group, then $L \in CRCL$. Furthermore the equivalence relation $\equiv_R \supseteq \equiv_L$ is finite but for any length we can choose R such that all words up to that length are irreducible.*

Proof. For $L = \emptyset$ or $L = \{\lambda\}$ this is trivial, the following is an induction over the size of the alphabet of L. Let L be a language over the alphabet $\Sigma = \{b, a_0, ..., a_{s-1}\}$, $G = M(L)$ and n be a multiple of the order of all elements $e \in G$ (Note here that n can be chosen arbitrarily large). We use sequences of consecutive powers of words of the form $\gamma_i := ba_{i \bmod s}^{n+(i \operatorname{Div} s)}$

or in other words $\gamma_{i+sj} = ba_i^{n+j}$ to build up representing words for each element in G. For example $[a_0bba_1a_2]$ can be represented as

$$(ba_0^n)^{n-1}(ba_0^{n+1})(ba_0^{2n})(ba_1^{2n+1})(ba_2^{3n})^{n-1}(ba_2^{3n+1}) = \gamma_0^{n-1}\gamma_s\gamma_{sn}\gamma_{s(n+1)+1}\gamma_{s2n+2}^{n-1}\gamma_{s(2n+1)+2}.$$

Since G is finite, we can find an m such that for every $e \in G$ there is a representing word w_e with $[w_e] = e$ and $bw_e = \gamma_0^{i_0}\gamma_1^{i_1}...\gamma_m^{i_m}$. Furthermore since $|\gamma_0| + 1 = |\gamma_s|$, we can pump i_0 and i_s by some multiples of n for each e in a way such that $l \le |w_e| < l + n$ for some l chosen big enough.

By induction over the alphabet size we have a confluent, strictly length-reducing rewriting system R_a, which reduces each word $w \in \{a_0, ..., a_{s-1}\}^*$ to an irreducible word $w_w \in \mathrm{Irr}_a$ with $[w_w] = [w]$. Let $l_a := max\{|w| \mid w \in \mathrm{Irr}_a\}$. Furthermore we may assume w.l.o.g. that $\{\gamma_0, ..., \gamma_m\} \subseteq b\mathrm{Irr}_a$ (Choosing the n in the induction large enough). Let

$$\{\alpha_1, ...\alpha_p\} := \{w \in (b\mathrm{Irr}_a)^+ \mid w \text{ primitive and } |w| \le n \text{ or } w \in bIrr_a\}.$$

In order to deal with all long cyclic repetitions of these words, we use the rewriting-rules $R_c = \{\alpha_i^{l+n}b \mapsto \alpha_i^l b \mid i \le p\}$. Relevant words not being a part of

such a cycle are in $L_{nc} := \{w \in (bIrr_a)^+b \mid \neg\exists i \leq p, x, y \in \Sigma^* \; xwy \in \alpha_i^+\}$. As marker-words we use

$$\{\beta_1, ..., \beta_q\} := L_{nc} \setminus \Sigma^+ L_{nc} \setminus L_{nc}\Sigma^+ \setminus \{w\gamma_j b \mid j \leq p \; \neg\exists i > j \; w = \gamma_i\}.$$

Observe here that each of the β's has a length of at least $n+1$. Furthermore we assume an ordering of the β's such that if $\beta_j = \gamma_k x$ and $\beta_i \neq \gamma_h y$ for any $h \leq k$ then $i > j$. Now we define the rewrite-rules

$$R'_n := \{\beta_i w \beta_i \mapsto \beta_i w_{[w]} \beta_i \mid |w_{[w]}| < |w| \leq l_{i-1}, \beta_i w \beta_i \notin \Sigma^* \beta_j \Sigma^* \text{ for any } j > i, \}$$

where the length restriction of l_{i-1} for w is constructed in the proof of Claim 2 showing that a longer w would not be reducible. To make it easier to prove the confluence, we eliminate rules, where the left side contains the left side of another rule, which does not change the reducibility of a word:

$$R_n := \{(l \mapsto r) \in R'_n \mid \neg\exists (l' \mapsto r') \in R'_n \; l' \text{ is Infix of } l\}.$$

It is clear that the rewriting system $R = R_a \cup R_c \cup R_n$ is strictly length-reducing and it's congruence refines G since left and right side of a rule are always congruent in G. It remains to show the following two claims:

□

Example: Let $G = S_3$, the group of permutations of $\{1, 2, 3\}$ be generated by the mirrorings $[a] = (12)$ and $[b] = (23)$ represented by the symbols a, b. (Choose $L \subseteq \{a, b\}^*$ to be any language accepted by this group.) The example is not trivial because of $[aba] = [bab]$. Choosing $n = 6$ according to the construction would make the example very big but the following 'handmade' construction of the representing words works already with $n = 2$: By recursion for the alphabet $\{a\}$ we obtain $R_a = \{a^5 \mapsto a^3\}$ leaving $\text{Irr}_a = \{a^i \mid i < 5\}$ and $l_a = 4$. Using $\gamma_0 = baa, \gamma_1 = baaa, \gamma_2 = baaaa$, we can find the representing words

$w_{()} = aa(baa)^2(baaaa)^2$, $w_{(12)} = aa(baa)^1(baaa)^1(baaaa)^2$, $w_{(23)} = aaaa(baaaa)^3$,

$w_{(123)} = aa(baa)^4(baaa)^1$, $w_{(132)} = aaa(baaaa)^3$, $w_{(23)} = aa(baa)^3(baaa)^2$,

all having length l =18 or 19. We get $\alpha_i = ba^{i-1}$ for $i \leq p = 5$ and obtain the rules $R_c = \{(ba^i)^{20} \mapsto (ba^i)^{18} \mid i \leq 5\}$. We get $\{\beta_1, ..., \beta_{11}\} =$

$$\{\gamma_1\gamma_0 b, \gamma_2\gamma_0 b, \gamma_2\gamma_1 b, \gamma_0 bb, \gamma_0 bab, \gamma_1 bb, \gamma_1 bab, \gamma_2 bb, \gamma_2 bab, babb, bbab\}$$

and obtain the rules $R_n := \{\beta_i w \beta_i \mapsto \beta_i w_{[w]} \beta_i \mid |w_{[w]}| < |w| \leq 338210, \beta_i w \beta_i \notin \Sigma^* \beta_j \Sigma^* \text{ for any } j > i, \}$. The length of an irreducible word can be at most $l_{11} = 676450$ which bounds the number of equivalence classes to $< 2^{676450}$.

Claim 1: R is confluent.

Proof. We have to show that for any pair of rules $l_1 \mapsto r_1, l_2 \mapsto r_2 \in R$ that if $ul_1v = xl_2y$ then there exists a $w \in \Sigma^*$ with $ur_1v \underset{R}{\overset{*}{\Longrightarrow}} w$ and $xr_2y \underset{R}{\overset{*}{\Longrightarrow}} w$.

- If both rules are in R_a this holds by induction.

- If $l_1 \mapsto r_1 \in R_a$ and $l_2 \mapsto r_2 \in R_c \cup R_n$ this holds since $l_1 \in \{a_0, ..., a_{s-1}\}^*$ and $l_2 \in (bIrr_a)^+b$ can not overlap.
- If both rules are in R_c this holds since either l_1 and l_2 overlap only in a small part z, which is not changed by the rules, that means w.l.o.g. $l_1 = \alpha_{i_1}^{l+n}b = \alpha_{i_1}^{l+n-1}z'z$ and $l_2 = \alpha_{i_2}^{l+n}b = zz''\alpha_{i_2}^{l+n-1}$ and thus $w = u\alpha_{i_1}^{l-1}z'zz''\alpha_{i_2}^{l-1}y$ or otherwise α_{i_1} is a rotation of α_{i_2} and thus $w = ur_1v = xr_2y$.
- If $l_1 \mapsto r_1 \in R_c$ and $l_2 \mapsto r_2 \in R_n$, again there can be only a short overlapping of the left sides inside a β_i, which is not changed since the appearance of α_k^{n+l} is not allowed in l_2 and β_i cannot appear in α_k^* by definition.
- If both rules are in R_n then either both rules have the same β_i in which case $w = u\beta_i w_{[z]}\beta_i y$ for $ul_1v = xl_2y = u\beta_i z\beta_i y$ since positions of repeated occurances of β_i as infix in a word must differ at least n, which garanties that w is shorter than ur_1v and xr_2y or otherwise with different β_{i_1}, β_{i_2} with $i_1 < i_2$ there is again a short unchanged overlap since in the case that $\beta_{i_2}w'\beta_{i_2} \in \Sigma^*\beta_{i_1}\Sigma^*$, we would have $\beta_{i_2}w'\beta_{i_2} \in \Sigma^*\beta_{i_1}\Sigma^*\beta_{i_1}\Sigma^*$, since no β_{i_2} can occur between two β_{i_1}'s but then l_1 would be an infix of l_2, which is not possible according to the definition of R_n.

□

Claim 2: The number of congruence classes defined by R is finite.

Proof. We show by induction on h that any corresponding to R irreducible word w, which does not contain any β_j with $j > h$ as infix, can only have some bounded length.

The case $h = 0$: Assume there would be an arbitrarily long irreducible word. After reading a prefix in Irr_a, we see the first b, which is the beginning of an infix $\alpha_{i'}$. If we continue reading, we can find $k < l + n$ consecutive repetitions of $\alpha_{i'}$, before we find a different $\alpha_{j'}$, which is not a prefix of a word in $\alpha_{i'}\Sigma^*$. Now this infix $\alpha_{i'}^k\alpha_{j'}$ must have an infix in L_{nc} and thus a minimal one in $L_{nc} \setminus \Sigma^+L_{nc} \setminus L_{nc}\Sigma^+$. Therfore the only possibility not to get a β as infix is that $\alpha_{j'} = \gamma_j$ for a $j \leq m$ and $\neg\exists i > j\ \alpha_{i'} = \gamma_i$, which can occur at most m times continuing to read the word. This restricts the length of the word to $\leq l_0 = (m + 1)(l + n)(l_a + 1)$.

Step from $h - 1$ to h: By induction, the length of the prefix before the first occurance of of β_h and the postfix after the last occurance of β_h is bounded by l_h. The same holds for the distance between two occurances of β_h, but then the reduction-rules for β_h ensure that it is even at most $l+n$ and thus this even holds for the distance between the first and the last occurance of β_h, which means we can bound the length of the word by $l_h := 2l_{h-1} + l + n + 2(l_a + 1)$.

Thus no irreducible word can be longer than l_q. □

3. Extension to DO

The variety DO consists of the closure of group and letter-testing languages under unambiguous concatenation.

Theorem *If the syntactic monoid $M(L)$ of a regular language L is in the variety DO, then $L \in CRCL$.*

References

[1] R. McNaughton, P. Narendran, F.Otto. Church-Rosser Thue systems and formal languages. Journal of the ACM, 35:324-344, 1988.

[2] G. Niemann, J. Waldmann. Some regular languages that are Church-Rosser congruential. DLT 2001.

RELATIVISATIONS OF DISJUNCTIVENESS

Cristian S. Calude
Department of Computer Science, The University of Auckland
Private Bag 92019, Auckland, New Zealand
e-mail: cristian@cs.auckland.ac.nz

and

Ludwig Staiger
Martin-Luther-Universität Halle-Wittenberg, Institut für Informatik,
von-Seckendorff-Platz 1, D - 06099 Halle, Germany
e-mail: staiger@informatik.uni-halle.de

ABSTRACT

The present paper proposes a generalisation of the notion of disjunctive (or rich) sequence, that is, of an infinite sequence of letters having each finite sequence as a subword.

Our aim is to give a reasonable notion of disjunctiveness relative to a given set of sequences F. We show that a definition like "*every subword which can occur infinitely often in F has to occur infinitely often in the sequence*" fulfils properties similar to the original unrelativised notion of disjunctiveness.

Finally, we investigate our concept of generalised disjunctiveness in spaces of Cantor expansions of reals.

Keywords: Disjunctive sequences, Cantor expansion, density, porosity.

A semi-infinite sequence is called disjunctive (or rich) if it has all finite words as subwords (infixes) (cf [8, 9]). This condition is, obviously, equivalent to having every finite word infinitely often as infix.

In connection with disjunctive sequences ξ over $\{0,\dots,r-1\}$ their real counterparts $0.\xi$ were considered. It is interesting to note that in contrast to properties like randomness or Kolmogorov complexity (cf. [1, 2, 7, 17]) disjunctiveness is not invariant under base conversion, more precisely speaking, if $\xi \in \{0,\dots,r-1\}^\omega$ and $\eta \in \{0,\dots,b-1\}^\omega$ satisfy $0.\xi = 0.\eta$ (as reals) then ξ might be disjunctive whereas η need not be so. For a more detailed treatment see [6].

Along with the usual base r expansions of real numbers one can also consider so-called Cantor expansions. In general, a Cantor expansion of a real is an element

$\xi \in X^{(f)}$ where $X^{(f)}$ is a certain subset of the Baire space $\mathbb{N}^\omega$ where the set of allowed letters at position i ($i \geq 1$) is specified as $\{0, \ldots, f(i)-1\}$ (cf. [4, 5]): Let $f(1), f(2), \ldots, f(n), \ldots$ be a fixed infinite sequence of positive integers greater than 1. Using a point we form the sequence $0.x_1x_2 \ldots x_i \ldots$ such that $x_n \in \mathbb{N}$, $0 \leq x_n < f(n)$, for all $n \geq 1$. The real number

$$\alpha := \sum_{i=1}^{n} \frac{x_i}{f(1) \cdot f(2) \cdots f(i)}$$

has $0.x_1x_2 \ldots$ as (one of) its *Cantor expansion(s).*

It is easy to see that the set of subwords occurring in a sequence $\xi \in X^{(f)} := \{x_1x_2 \ldots x_i \ldots : 0 \leq x_i < f(i)\}$ depends on the function $f : \mathbb{N} \to \mathbb{N}$, and, moreover, some subword occurring once in some $\xi \in X^{(f)}$ might not occur infinitely often in any $\eta \in X^{(f)}$.

Thus the definition of disjunctiveness for Cantor expansions needs some generalisation. In this paper we propose a possible modification of the notion of disjunctive sequence in the following way[1]: *A sequence $\xi \in F \subseteq \mathbb{N}^\omega$ is F-disjunctive if and only if every infix which may occur infinitely often in some sequence $\eta \in F$ occurs infinitely often in ξ. Here, of course, the phrase* "which may occur infinitely often in some sequence" *needs further specification.*

1. Preliminaries

1.1. Notation

By $\mathbb{N} = \{0, 1, 2, \ldots\}$ we denote the set of natural numbers. In order to treat arbitrary finite alphabets we let $X_r := \{0, \ldots, r-1\}$ be an alphabet of cardinality $|X_r| = r$, $r \in \mathbb{N}, r \geq 2$. In this paper we will use finite alphabets X_r, $r \in \mathbb{N}$, and $\mathbb{N}$ as a countably infinite alphabet. By X^* we denote the set of finite strings (words) over the alphabet X, including the *empty* word e. We consider also the space X^ω of infinite sequences (ω-words) over X. For $w \in X^*$ and $\eta \in X^* \cup X^\omega$ let $w \cdot \eta$ be their *concatenation.* This concatenation product extends in an obvious way to subsets $W \subseteq X^*$ and $B \subseteq X^* \cup X^\omega$.

By "$\sqsubseteq$" we denote the prefix relation, and $\mathbf{pref}(B) := \bigcup_{\eta \in B} \mathbf{pref}(B)$ are the languages of finite prefixes of η and B, respectively. The set of subwords (infixes) of $\eta \in X^* \cup X^\omega$ will be denoted by $\mathbf{infix}(\eta) := \{w : w \in X^* \wedge \exists v(vw \sqsubseteq \eta)\}$.

In the sequel, we shall confine ourselves to sets of the form $X^{(f)} \subseteq \mathbb{N}$ as defined in the introduction.[2] Introducing a metric in $X^{(f)}$ as follows

$$\rho_f(\xi, \eta) := \inf\Big\{\prod_{i=1}^{|w|} f(i)^{-1} : w \sqsubset \xi \wedge w \sqsubset \eta\Big\} \tag{1.1}$$

[1] We base our generalisation on infinite occurrences of subwords. This proposal seems to be justified by the results of Section 2.

[2] For $f \equiv r$ this covers also the case $X^{(f)} = X_r^\omega$.

we make $(X^{(f)}, \rho_f)$ a compact metric space. It is easily verified that ρ_f is indeed a metric which, in addition, satisfies the ultra-metric inequality.

$$\rho_f(\zeta, \xi) \leq \max\{\rho_f(\zeta, \eta), \rho_f(\xi, \eta)\} \tag{1.2}$$

Open (in view of Eq. (1.2) they are simultaneously closed) balls in this space $(X^{(f)}, \rho_f)$ are the sets $X^{(f)} \cap w \cdot \mathbb{N}^\omega$. Then open sets in $X^{(f)}$ are of the form $X^{(f)} \cap W \cdot \mathbb{N}^\omega$ where $W \subseteq \mathbf{pref}(X^{(f)})$. From this it follows that a subset $F \subseteq X^{(f)}$ is *closed* if and only if $\mathbf{pref}(\xi) \subseteq \mathbf{pref}(F)$ implies $\xi \in F$.

The *closure* of a subset $F \subseteq X^{(f)}$ in the space $(X^{(f)}, \rho_f)$, that is, the smallest closed subset of X^ω containing F is denoted by $\mathcal{C}(F)$. One has $\mathcal{C}(F) = \{\xi : \mathbf{pref}(\xi) \subseteq \mathbf{pref}(F)\}$.

It should be mentioned that, due to the special choice of the metric (see Eq. (1.1)), the following uniformity property for balls is satisfied.

$$\sum_{x \in X_{f(|w|)+1}} \mathrm{diam}_f(X^{(f)} \cap w \cdot x \cdot \mathbb{N}^\omega) = \mathrm{diam}_f(X^{(f)} \cap w \cdot \mathbb{N}^\omega) \tag{1.3}$$

1.2. Density, Baire Category and Porosity

Next, we introduce the concept of topological density, Baire category and porosity for our complete metric space $(X^{(f)}, \rho_f)$. (see e.g. [10, 11, 18])

A subset $M \subseteq X^{(f)}$ is called *dense* in $X^{(f)}$ provided its closure $\mathcal{C}(F)$ is the whole space $X^{(f)}$. A set $M \subseteq X^{(f)}$ is *nowhere dense* in $(X^{(f)}, \rho_f)$ provided its closure $\mathcal{C}(M)$ does not contain a nonempty open subset.

A set F is of *first Baire category* iff it is a countable union of nowhere dense sets, otherwise it is of *second Baire category*. The complements of sets of first Baire category are called *residual*.

The concept of porous subsets in general metric spaces is introduced e.g. in [18, Section 2.C]. We explain it for the space $(X^{(f)}, \rho_f)$.

Let $\lambda(E, u) := \sup\{\mathrm{diam}_f(w \cdot \mathbb{N}^\omega \cap X^{(f)}) : u \sqsubseteq w \wedge w \cdot \mathbb{N}^\omega \cap E = \emptyset\}$ be the diameter of a largest ball contained $u \cdot \mathbb{N}^\omega \cap X^{(f)}$ but disjoint to E, and

$$\mathbf{p}(E, \xi) := \limsup_{u \to \xi} \frac{\lambda(E, u)}{\mathrm{diam}_f(u \cdot \mathbb{N}^\omega \cap X^{(f)})} \tag{1.4}$$

is the *porosity* of E at the point ξ. A set $E \subseteq X^{(f)}$ is called *porous* if $\mathbf{p}(E, \xi) > 0$ for all $\xi \in X^{(f)}$, and a set $F \subseteq X^{(f)}$ is called *σ-porous* iff it is a countable union of porous sets. It is obvious that every porous set is nowhere dense, but the converse need not be true (see [18]). It should be noted, however, that in (X_r^ω, ρ) every nowhere dense set definable by a finite automaton is porous ([12, 15, 16]).

1.3. Disjunctive sequences in X_r^ω

Finally, we list some properties of the set of disjunctive sequences $D_r \subseteq X_r^\omega$ known from [3, 16]

Theorem 1 *1. D_r is a residual set in X_r^ω.*

2. $X_r^\omega \setminus D_r = \bigcup_{w\in X_r^*}\left(X_r^\omega \setminus X_r^* \cdot w \cdot X_r^\omega\right) = \bigcup_{w\in X_r^*}(X_r^{|w|} \setminus \{w\})^\omega$

3. $X_r^\omega \setminus D_r$ is the union of all nowhere dense ω-languages definable by a finite automaton.

4. $X_r^\omega \setminus D_r$ is σ-porous.

5. $\mu(D_r) = 1$ for all non-degenerate product measures on X_r^ω.

2. Generalised Disjunctiveness

In this section we make precise what we mean when we say informally that an ω-word $\xi \in F$ should be called *disjunctive* if and only if every word $w \in \mathbb{N}^*$ which can appear infinitely often as an infix in F has to appear infinitely often as an infix of ξ. To this end we observe that a necessary condition when a word w can appear infinitely often as an infix in F is the following one.

Let $\mathbf{Infix}_\infty(F) := \{w : \exists^\infty n \exists u(|u| = n \wedge uw \in \mathbf{pref}(F)\}$. An ω-word ξ is called *F-disjunctive* provided $\xi \in F$ and $\mathbf{Infix}_\infty(F) = \mathbf{Infix}_\infty(\xi)$.

For general subsets of $\mathbb{N}^\omega$ or $X^{(f)}$ this condition is still complicated to treat. To this end we introduce the following notion which, when satisfied for $\mathbf{Infix}_\infty(F)$, will alleviate the investigation of disjunctive sequences. A set $W \subseteq \mathbb{N}^*$ is referred to as *left prolongable* if and only if for every $w \in W$ there is a $x \in \mathbb{N}$ such that $x \cdot w \in W$.

Proposition 2 *Let $\mathbf{Infix}_\infty(F)$ be left prolongable. Then $\mathbf{infix}(\xi) \supseteq \mathbf{Infix}_\infty(F)$ implies $\mathbf{Infix}_\infty(\xi) \supseteq \mathbf{Infix}_\infty(F)$ for all $\xi \in F$.*

The following example shows that prolongability is essential.

Example 1 Let $F := \prod_{i=2}^\infty \{i1, 00\}$. Then $\mathbf{Infix}_\infty(F) = 0^* \cup 1 \cdot 0^*$, and, indeed, for $\eta = 210^\omega$ we have $\mathbf{infix}(\eta) \supseteq \mathbf{Infix}_\infty(F) \supset \mathbf{Infix}_\infty(\eta) = 0^*$.

As a corollary to Proposition 2 we obtain properties of the set of all F-disjunctive ω-words similar to Property 2 in Theorem 1 for D_r.

Corollary 3 *If $\mathbf{Infix}_\infty(F)$ is left prolongable, then $D_F = \bigcap_{w\in\mathbf{Infix}_\infty(F)} (F \cap \mathbb{N}^* \cdot w \cdot \mathbb{N}^\omega)$ is the set of all F-disjunctive ω-words, and*

$$F\setminus D_F = \bigcup_{w\in\mathbf{Infix}_\infty(F)} (F \setminus \mathbb{N}^* \cdot w \cdot \mathbb{N}^\omega) = \bigcup_{w\in\mathbf{Infix}_\infty(F)} \left(F \cap \bigcap_{j=0}^{|w|-1} \mathbb{N}^j \cdot (\mathbb{N}^{|w|} \setminus \{w\})^\omega\right).$$

3. Disjunctiveness in ultimately connected sets

Here we consider the case of a subset F of a Cantor space X_r^ω having the following property.

$$\forall u(u \in \mathbf{pref}(F) \to \exists v(v \in X_r^* \wedge u \cdot v \cdot F \subseteq F)) \tag{3.5}$$

Those sets were called *ultimately connected* and can be characterised by the so-called *stabiliser* of $F \subseteq X_r^\omega$, $Stab(F)$ (cf. [13, 14]).

$$Stab(F) := \{w : w \in \mathbf{pref}(E) \setminus \{e\} \wedge w \cdot F \subseteq F\} \tag{3.6}$$

$Stab(F)$ is closed under concatenation, that is, is a subsemigroup of X_r^*.

Proposition 4 *An ω-language $F \subseteq X_r^\omega$ is ultimately connected if and only if* $\mathbf{pref}(F) \subseteq \mathbf{pref}(Stab(F))$

Examples of ultimately connected ω-languages are the so-called ω-power languages $W^\omega := \{\prod_{i=1}^\infty w_i : w_i \in W \setminus \{e\}\}$. Obviously, the stabiliser of an ω-power language W^ω satisfies $W^* \setminus \{e\} \subseteq Stab(W^\omega) \subseteq Stab(\mathcal{C}(W^\omega)) \subseteq \mathbf{pref}(W^\omega)$ and $Stab(W^\omega) \cdot W^\omega = W^\omega$.

For ultimately connected ω-languages F the language $\mathbf{Infix}_\infty(F)$ has the following properties.

Proposition 5 $\mathbf{Infix}_\infty(F) = \mathbf{infix}(Stab(F))$ *and* $\mathbf{Infix}_\infty(F)$ *is two-sided prolongable.*

Corollary 3 applies immediately to ultimately connected ω-languages.

Corollary 6 *If $F \subseteq X_r^\omega$ is ultimately connected then*

$$F \setminus D_F = \bigcup_{w \in Stab(F)} (F \setminus X_r^* \cdot w \cdot X_r^\omega) = \bigcup_{w \in Stab(F)} \left(F \cap \bigcap_{j=0}^{|w|-1} X_r^j \cdot (X_r^{|w|} \setminus \{w\})^\omega \right).$$

We conclude the part on ultimately connected ω-languages by mentioning some results from [14] similar to Theorem 1.3 concerning ω-languages definable by a finite automaton which are nowhere dense in F. To this end we mention that a set $E \subseteq X_r^\omega$ *is nowhere dense in* $F \subseteq X_r^\omega$ if and only if $\forall u(u \in \mathbf{pref}(F) \to \exists v\,(v \in X_r^* \wedge u \cdot v \in \mathbf{pref}(F) \setminus \mathbf{pref}(E)))$.

Lemma 7 *If $F \subseteq X_r^\omega$ is ultimately connected and E is definable by a finite automaton then*

1. *E is nowhere dense in F iff $\exists w(w \in Stab(F) \wedge E \cap \mathcal{C}(F) \subseteq \mathcal{C}(F) \setminus Stab(F)^* \cdot w \cdot X_r^\omega)$, and*
2. *E is nowhere dense in F iff $\exists \bar{w}(\bar{w} \in Stab(F) \wedge E \subseteq \mathcal{C}(F) \setminus X_r^* \cdot \bar{w} \cdot X_r^\omega)$ provided $E \subseteq \mathcal{C}(F)$.*

4. Disjunctiveness for Cantor expansions

In this section we derive some simple properties of the set $\mathbf{Infix}_\infty(X^{(f)})$. From these properties, in particular, we derive that $\mathbf{Infix}_\infty(X^{(f)})$ is left prolongable, whence properties like Proposition 2 and Corollary 3 hold for the set of all disjunctive ω-words in $X^{(f)}$, D_f, independently of the choice of f.

We start with a simple property.

Proposition 8 *1.* $u \cdot \mathbb{N}^\omega \subseteq \mathbb{N}^* \cdot w \cdot \mathbb{N}^\omega$ *iff* $w \in \mathbf{infix}(u)$.

2. Let $u \in \mathbf{pref}(X^{(f)})$ *and* $|u| \leq n$.
Then $0^n \cdot w \in \mathbf{pref}(X^{(f)})$ *iff* $u \cdot 0^{n-|u|} \cdot w \in \mathbf{pref}(X^{(f)})$.

3. If $w \in \mathbf{Infix}_\infty(X^{(f)})$ *then* $\{0,1\}^* \cdot w \cdot \{0,1\}^* \subseteq \mathbf{Infix}_\infty(X^{(f)})$.

Lemma 9 *Let* $\kappa_f(n) := \limsup_{i\to\infty} \min\{f(i+j) : 1 \leq j \leq n\} \geq r$.
Then $\{0,1\}^* \cdot X_r^n \cdot \{0,1\}^* \subseteq \mathbf{Infix}_\infty(X^{(f)})$.

As an immediate consequence of Proposition 8.3 we obtain the announced property.

Corollary 10 $\mathbf{Infix}_\infty(X^{(f)})$ *is left prolongable, for every function* $f : \mathbb{N} \to \mathbb{N} \setminus \{0,1\}$.

5. Topological Properties of $X^{(f)} \setminus \mathbb{N}^* \cdot w \cdot \mathbb{N}^\omega$

In this section we are going to investigate some topological properties of the set of disjunctive sequences in $X^{(f)}$. First we investigate the relationship to density and measure. We start with a simple proposition which holds for all functions $f : \mathbb{N} \to \mathbb{N} \setminus \{0,1\}$.

Lemma 11 *The set* $X^{(f)} \setminus \mathbb{N}^* \cdot w \cdot \mathbb{N}^\omega$ *is nowhere dense in* $(X^{(f)}, \rho_f)$ *whenever* $w \in \mathbf{Infix}_\infty(X^{(f)})$.

In contrast to Theorem 1.5 the measure property does not hold in general.

Example 2 Let μ be the measure on $X^{(f)}$ induced by the Lebesgue measure on $[0,1]$. For the function $f(i) := (i+1)^2$ and the set $F := X^{(f)} \setminus \mathbb{N}^* \cdot 0 \cdot \mathbb{N}^\omega = \prod_{i=1}^\infty (X_{f(i)} \setminus \{0\})$ we have $\mu(F) = \prod_{i=1}^\infty (1 - f(i)^{-1}) = \prod_{j=2}^\infty (1 - j^{-2}) > 0$.

From [18] it is known that a porous set $F \subseteq X^{(f)}$ is nowhere dense and has measure $\mu(F) = 0$. As we have seen in Example 2 the complement of the set of disjunctive sequences in $X^{(f)}$ may have positive measure. Next we investigate how this behaviour depends on the function f.

A first result and a comparison with Theorem 1 show that the case of bounded functions is similar to the case of constant alphabets.

Lemma 12 *If $f : \mathbb{N} \to \mathbb{N}$ is bounded then for every $w \in \mathbf{Infix}_\infty(X^{(f)})$ the set $X^{(f)} \setminus \mathbb{N}^* \cdot w \cdot \mathbb{N}^\omega$ is porous in $X^{(f)}$.*

Thus the case when f is unbounded needs special treatment. We obtain the following sufficient condition for the non-porosity of sets $X^{(f)} \setminus \mathbb{N}^* \cdot w \cdot \mathbb{N}^\omega$.

Theorem 13 *If $f : \mathbb{N} \to \mathbb{N}$ is unbounded and $k_f := \sup\{k : f^{-1}(k)$ is infinite$\} < \infty$ then for every $i > k_f$ and $v \in \{0,1\}^*$ the set $X^{(f)} \setminus \mathbb{N}^* \cdot vi \cdot \mathbb{N}^\omega$ is not porous in $X^{(f)}$.*

We may, however, have also sets of the form $X^{(f)} \setminus \mathbb{N}^* \cdot w \cdot \mathbb{N}^\omega$ which are porous.

Example 3 Define

$$f(i) := \begin{cases} n, \text{ if } i = n^2 \text{ for some } n \in \mathbb{N}, \text{ and} \\ 2, \text{ otherwise.} \end{cases}$$

Then the set $X^{(f)} \setminus \mathbb{N}^* \cdot w \cdot \mathbb{N}^\omega$ is porous in $X^{(f)}$ whenever $w \in \{0,1\}^*$.

In particular, the condition of Theorem 13 is satisfied when f tends to infinity.

Corollary 14 *If $\lim_{n\to\infty} f(n) = \infty$ then $\mathbf{Infix}_\infty(X^{(f)}) = \mathbb{N}^*$ and each set $X^{(f)} \setminus \mathbb{N}^* \cdot w \cdot \mathbb{N}^\omega$ is not porous in $X^{(f)}$.*

References

[1] J.-Y. Cai, J. Hartmanis: On Hausdorff and Topological Dimensions of the Kolmogorov Complexity of the Real Line. *J. Comput. System Sci.* 49 (1994) 3, 605–619.

[2] C.S. Calude, H. Jürgensen: Randomness as an Invariant for Number Representations. In: Results and Trends in Theoretical Computer Science (H. Maurer, J. Karhumäki, G. Rozenberg Eds.), Lecture Notes in Comput. Sci. 812, Springer-Verlag, Berlin 1994, 44–66.

[3] C.S. Calude, L. Priese, L. Staiger: Disjunctive Sequences: An Overview. CDMTCS Research Report 063, Auckland (1997)

[4] S. Drobot: Real Numbers. Prentice-Hall, Englewood Cliffs, New Jersey, 1964.

[5] G. H. Hardy, E. M. Wright: An Introduction to the Theory of Numbers. Cambridge Univ. Press, London 1954.

[6] P. Hertling: Disjunctive ω-words and Real Numbers. *J.UCS (Journal of Universal Computer Science)* 2 (1996), 549 – 568.

[7] P. Hertling, K. Weihrauch: Random Elements in Effective Topological Spaces with Measure. *Inform. and Comput.* 181 (2003), 32–56.

[8] H. Jürgensen, H.J. Shyr, G. Thierrin: Disjunctive ω-languages. *Elektron. Informationsverarb. Kybernetik EIK* 19 (1983), 267–278.

[9] H. Jürgensen, G. Thierrin: On ω-languages whose Syntactic Monoid is Trivial. *Intern. J. Comput. Inform Sci.* 12 (1983), 359 -365.

[10] K. Kuratowski: Topology I. Academic Press, New York 1966.

[11] J.C. Oxtoby: Measure and Category. Springer-Verlag, Berlin 1971.

[12] L. Staiger: Reguläre Nullmengen. *Elektron. Informationsverarb. Kybernetik EIK* 12 (1976), 307–311.

[13] L. Staiger, A note on connected ω-languages. *Elektron. Informationsverarb. Kybernetik EIK* 16 (1980), 245–251.

[14] L. Staiger, On ω-power languages, In: New Trends in Formal Languages, Control, Cooperation, and Combinatorics, (Gh. Păun and A. Salomaa Eds.), Lecture Notes in Comput. Sci. No. 1218, Springer–Verlag, Berlin. 377–393.

[15] L. Staiger: Rich ω-words and Monadic Second-order Arithmetic. In: Computer Science Logic, 11th International Workshop, CSL'97, Selected Papers (M. Nielsen and W. Thomas Eds.), Lecture Notes in Comput. Sci. 1414, Springer-Verlag, Berlin (1998), 478–490.

[16] L. Staiger: How Large is the Set of Disjunctive Sequences? *J.UCS (Journal of Universal Computer Science)* 8 (2002), 348–362.

[17] L. Staiger: The Kolmogorov Complexity of Real Numbers. *Theoret. Comput. Sci.* 284 (2002), 455 – 466.

[18] L. Zajìček: Porosity and σ-porosity. *Real Analysis Exch.* 13 (1987-88), 314–350.

ON THE SIZE OF HYBRID NETWORKS OF EVOLUTIONARY PROCESSORS

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

ABSTRACT

We show that any recursively enumerable language can be generated by a hybrid network of evolutionary processors (HNEP) of constant size. Moreover, we show that HNEPs of size 2 can generate non-context-free languages.

Keywords: Evolutionary processors, descriptional complexity.

In a series of papers, Mitrana and others introduced networks of evolutionary processors and variants [1, 2, 3, 4]. An*evolutionary processor* contains a set of strings and can perform the following elementary operations (*evolution steps*): insert a symbol, delete a symbol, replace a symbol by another symbol. Moreover, it can be required that insertions and deletions can only take place on the left or right end. In a *hybrid network of evolutionary processors (HNEP)*, a set of processors is connected according to an underlying graph. Moreover, the processors have input and output filters (permitting and forbidding random context conditions). In a *communication step*, any string matching the output condition leaves a processor and enters a neighboring processor if it passes the input filter. A computation of an HNEP consists of an alternating sequence of evolutionary and communication steps. The generated language is the set of all strings that reach a specified output processor.

It is shown in [2] that any recursively enumerable language $L \subseteq V^*$ can be generated by an HNEP with $26 + 3|V|$ processors. Moreover, it is proved that any HNEP with 1 processor generates a regular language and that there is a non-context-free language generated by an HNEP with 4 processors. We improve these results as follows:

Theorem 1 *There is a fixed graph G such that any recursively enumerable language L is generated by an HNEP with underlying graph G.*

Theorem 2 *There is an HNEP with* 2 *processors that generates a non-context-free language.*

References

[1] Juan Castellanos, Carlos Martín-Vide, Victor Mitrana, and José Sempere. Networks of evolutionary processors. *Acta Informatica*, 39:517–529, 2003.

[2] Erzsébet Csuhaj-Varjú, Carlos Martín-Vide, and Victor Mitrana. Hybrid networks of evolutionary processors are computationally complete. *Submitted*, 2003.

[3] Carlos Martín-Vide, Victor Mitrana, Mario Pérez-Jiménez, and Fernando Sancho-Caparrini. Hybrid networks of evolutionary processors. In *Genetic and Evolutionary Computation - GECCO 2003*, LNCS, pages 401–412, 2003.

[4] Victor Mitrana. Some complexity aspects of hybrid networks of evolutionary processors. In *Proceedings of DCFS 2003*, pages 53–65, 2003.

GI-Fachgruppe 0.1.5 „Automaten und Formale Sprachen“

Wahl der Fachruppenleitung

Die Wahl der Leitung der GI-Fachgruppe 0.1.5 wurde am 1. Oktober 2003 im Rahmen der Fachgruppen-Sitzung auf dem 13. Theorietag „Automaten und Formale Sprachen“ in Herrsching durchgeführt. Anwesend waren folgende Fachgruppen-Mitglieder:

Suna Bensch (Potsdam), Henning Bordihn (Potsdam), Jürgen Dassow (Magdeburg), Henning Fernau (Tübingen), Rudolf Freund (Wien), Jens Glöckler (Gießen), Maia Hoeberechts (London, Ontario), Markus Holzer (München), Ekkart Kindler (Paderborn), Daniel Kirsten (Paris), Andreas Klein (Kassel), Martin Kutrib (Gießen), Martin Lange (München), Martin Leucker (Uppsala), Andreas Malcher (Frankfurt), František Mráz (Prag), Marion Oswald (Wien), Friedrich Otto (Kassel), Martin Plátek (Prag), Bernd Reichel (Magdeburg), Jens Reimann (Gießen), Klaus Reinhardt (Tübingen), Ludwig Staiger (Halle) und Ralf Stiebe (Magdeburg).

Zum Wahlleiter wurde Henning Fernau bestimmt. Der Wahlleiter stellte die 6 Kandidaten vor. Von den 24 abgegebenen Stimmen waren 24 gültig. Es wurden gewählt

Henning Bordihn (Potsdam), Jürgen Dassow (Magdeburg), Markus Holzer (München), Martin Kutrib (Gießen) und Friedrich Otto (Kassel).

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

Wahl des Fachgruppensprechers

Am 1. Oktober 2003 einigten sich die anwesenden Mitglieder der Fachgruppenleitung darauf, daß Herr Otto das Amt des Sprechers und Herr Kutrib das Amt des stellvertretenden Sprechers wahrnehmen werden. Dieser Vorschlag wurde einstimmig angenommen. Herr Dassow und Herr Kutrib nahmen die Wahl an.

TeilnehmerInnenliste

- **Suna Bensch**
  Institut für Informatik
  Universität Potsdam
  August-Bebel-Straße 89
  14482 Potsdam

  Tel.: +49-331-977-3028
  Fax: +49-331-977-3022

  e-mail: aydin@cs.uni-potsdam.de

- **Eike Best**
  Department für Informatik
  Universität Oldenburg
  Fakultät II
  26111 Oldenburg

  Tel.: +49-441-798-2973
  Fax: +49-441-798-2965

  e-mail: Eike.Best@Informatik.Uni-Oldenburg.DE

- **Henning Bordihn**
  Institut für Informatik
  Universität Potsdam
  August-Bebel-Straße 89
  14482 Potsdam

  Tel.: +49-331-977-3027
  Fax: +49-331-977-3022

  e-mail: henning@cs.uni-potsdam.de

- **Jürgen Dassow**
  Fakultät für Informatik

Otto-von-Guericke-Universität Magdeburg
Postfach 4120
39016 Magdeburg

Tel.: +49-391-67-18853
Fax: +49-391-67-12018

e-mail: dassow@iws.cs.uni-magdeburg.de

- **Javier Esparza**
Institut für Formale Methoden der Informatik
Universität Stuttgart
Universitätsstr. 38
70569 Stuttgart

Tel.: +49-711-7816-455
Fax: +49-711-7816-370

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

- **Henning Fernau**
School of Electrical Engineering and Computer Science
University of Newcastle
University Drive
Callaghan, NSW 2308
Australia

Tel.: +61-2-4921-6076
Fax: +61-2-4921-6929

e-mail: fernau@cs.newcastle.edu.au

- **Rudolf Freund**
Institut für Computersprachen
Technische Universität Wien
Favoritenstr. 9
A-1040 Wien
Österreich

Tel.: +43-1-58801-18542
Fax: +43-1-58801-18599

e-mail: rudi@emcc.at

- **Jens Glöckler**
  Institut für Informatik
  Universität Gießen
  Arndtstr. 2
  35392 Gießen

  Tel.: +49-641-99-32142
  Fax: +49-641-99-32149

  e-mail: Jens.Gloeckler@math.uni-giessen.de

- **Maia Hoeberechts**
  Department of Computer Science
  University of Western Ontario
  N6A 5B7 London
  Ontario, Canada

  Tel.:
  Fax:

  e-mail: hoebere@csd.uwo.ca

- **Markus Holzer**
  Institut für Informatik
  Technische Universität München
  Boltzmannstraße 3
  85748 Garching bei München

  Tel.: +49-89-289-17230
  Fax: +49-89-289-17207

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

- **Roman König**
  Institut für Informatik III
  Universität Erlangen-Nürnberg
  Martensstraße 3
  91058 Erlangen

  Tel.: +49-9131-8527921
  Fax: +49-9131-39388

  e-mail: koenig@informatik.uni-erlangen.de

- **Ekkart Kindler**
  Institut für Informatik
  Universität Paderborn
  33098 Paderborn

  Tel.: +49-5251-60-3320
  Fax: +49-5251-60-3530

  e-mail: kindler@uni-paderborn.de

- **Daniel Kirsten**
  LIAFA
  Université Denis Diderot – Case 7014
  2, place Jussieu
  F-75251 Paris Cedex 05
  France

  Tel.:
  Fax: +33-1-44276849

  e-mail: kirsten@liafa.jussieu.fr

- **Andreas Klein**
  Fachbereich für Mathematik und Informatik
  Universität Kassel
  Heinrich-Plett-Straße 40
  34132 Kassel

  Tel.: +49-561-804-4367
  Fax:

  e-mail: klein@mathematik.uni-kassel.de

- **Martin Kutrib**
  Institut für Informatik
  Universität Gießen
  Arndtstr. 2
  35392 Gießen

  Tel.: +49-641-99-32144
  Fax: +49-641-99-32149

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

- **Martin Lange**
  LFE Theoretische Informatik
  Institut für Informatik
  Ludwig-Maximilians-Universität
  Oettingenstr. 67
  80538 München

  Tel.: +49-89-2180 9313
  Fax: +49-89-2180 9338

  e-mail: mlange@informatik.uni-muenchen.de

- **Martin Leucker**
  Department of Computer Science
  Uppsala University
  Box 337
  SE-75105 Uppsala
  Sweden

  Tel.: +46-18-471-5779
  Fax:

  e-mail: leucker@docs.uu.se

- **Andreas Malcher**
  FB Informatik
  Johann Wolfgang Goethe-Universität
  Postfach 11 19 32
  60054 Frankfurt am Main

  Tel.: +49-69-798-28413
  Fax: +49-69-798-28334

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

- **František Mráz**
  Department of Computer Science
  Charles University
  Malostranské nám. 25
  CZ-118 00 Praha 1
  Czech Republic

  Tel.: +42-2-21914219
  Fax: +42-2-21914281

e-mail: mraz@ksvi.ms.mff.cuni.cz

- **Marion Oswald**
  Institut für Computersprachen
  Technische Universität Wien
  Favoritenstr. 9
  A-1040 Wien
  Österreich

  Tel.:
  Fax:

  e-mail: marion@emcc.at

- **Friedrich Otto**
  Arbeitsgruppe Theoretische Informatik
  Fachbereich Mathematik/Informatik
  Universität Gesamthochschule Kassel
  Heinrich-Plett-Straße 40
  34132 Kassel

  Tel.: +49-561-804-4573
  Fax: +49-561-804-4008

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

- **Martin Plátek**
  KTIML MFF
  Charles University
  Malostranské nám. 25
  CZ-118 00 Praha 1
  Czech Republic

  Tel.:
  Fax:

  e-mail: platek@ksi.ms.mff.cuni.cz

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

Tel.: +49-391-67-12851
Fax: +49-391-67-11250

e-mail: reichel@iws.cs.uni-magdeburg.de

- **Jens Reimann**
Institut für Informatik
Universität Gießen
Arndtstr. 2
35392 Gießen

Tel.: +49-641-99-32153
Fax: +49-641-99-32149

e-mail: Jens.Reimann@informatik.uni-giessen.de

- **Klaus Reinhardt**
Wilhelm-Schickard Institut für Informatik
Universität Tübingen
Sand 13
72076 Tübingen

Tel.: +49-7071-29-77566
Fax: +49-7071-29-5061

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

- **Karsten Schmidt**
Institut für Informatik
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin

Tel.: +49-30-2093-3083
Fax:

e-mail: kschmidt@informatik.hu-berlin.de

- **Ludwig Staiger**
Institut für Informatik
Martin-Luther-Universtität Halle-Wittenberg
Von-Seckendorff-Platz 1
06099 Halle an der Saale

Tel.: +49-345-5524714
Fax: +49-345-5527009

e-mail: staiger@informatik.uni-halle.de

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

Tel.: +49-391-67-12457
Fax: +49-391-67-12018

e-mail: stiebe@iws.cs.uni-magdeburg.de

- **Bettina Sunckel**
Institut für Informatik
Johann Wolfgang Goethe-Universität
Postfach 11 19 32
60054 Frankfurt am Main

Tel.: +49- 69-7982 8419
Fax: +49-69-7982 8334

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