JUSTUS-LIEBIG-
UNIVERSITÄT
GIESSEN

Institut für Informatik

18. Theorietag

Automaten und Formale Sprachen

Markus Holzer, Martin Kutrib, Andreas Malcher (Hrsg.)

Wettenberg-Launsbach bei Gießen
30. September – 2. Oktober 2008

Title: Proceedings 18. Theorietag *Automaten und Formale Sprachen*

Published by: Institut für Informatik
Universität Gießen
Arndtstraße 2
35392 Gießen
Germany

Published in September 2008

ISBN 978-3-00-025920-3

Vorwort

Der Theorietag ist die Jahrestagung der Fachgruppe *Automaten und Formale Sprachen* der Gesellschaft für Informatik. Er wird seit 1991 von Mitgliedern der Fachgruppe an wechselnden Orten in Deutschland und Österreich veranstaltet. Im Laufe des Theorietags findet auch die jährliche Fachgruppensitzung statt. Seit 1996 wird der Theorietag von einem eintägigen Workshop mit eingeladenen Vorträgen begleitet. Die bisherigen Austragungsorte waren Magdeburg (1991), Kiel (1992), Dagstuhl (1993), Herrsching (1994), Schloß Rauischholzhausen (1995), Cunnersdorf (1996), Barnstorf (1997), Riveris (1998), Schauenburg-Elmshagen (1999), Wien (2000), Wendgräben (2001), Wittenberg (2002), Herrsching (2003), Caputh (2004), Lauterbad (2005), Wien (2006) und Leipzig (2007).
Der diesjährige Theorietag wird nach 13 Jahren wieder vom Institut für Informatik der Justus-Liebig-Universität Gießen ausgerichtet. Er findet mit dem vorangestellten Workshop über *Selected Topics in Theoretical Computer Science* vom 30. September bis zum 2. Oktober 2008 in Wettenberg-Launsbach bei Gießen statt. Teilnehmer aus Belgien, Deutschland, England, Frankreich, Italien, Österreich, Tschechien und Ungarn folgten der Einladung nach Mittelhessen. Neben den vier eingeladenen Vorträgen des Workshops von

- Andreas Klein (Ghent, Belgium)
- Maurice Margenstern (Metz, France)
- Carlo Mereghetti (Milan, Italy)
- György Vaszil (Budapest, Hungary)

wurden 16 Beiträge in das Programm des Theorietags aufgenommen. Die Kurzfassungen der Beiträge sowie das Vortragsprogramm und eine Liste der Teilnehmer mit ihren Adressen sind im vorliegenden Tagungsband enthalten. Wir wünschen allen Teilnehmerinnen und Teilnehmern des 18. Theorietags eine interessante und anregende Tagung und einen schönen und angenehmen Aufenthalt in Wettenberg-Launsbach.

Gießen, im September 2008

Markus Holzer
Martin Kutrib
Andreas Malcher

Inhaltsverzeichnis

Programm

Workshop *Selected Topics in Theoretical Computer Science*

Thuesday, September 30, 2008

09.25 – 09.30	Opening
09.30 – 10.30	Maurice Margenstern (Metz, France) *About the Injectivity of the Global Function of Cellular Automata in the Hyperbolic Plane*
10.30 – 11.00	Coffee break
11.00 – 12.00	György Vaszil (Budapest, Hungary) *Symport/Antiport Systems and P Automata – Membrane Systems for the Characterization of Formal Languages*
12.00 – 14.00	Lunch
14.00 – 15.00	Viliam Geffert (Košice, Slovakia), Carlo Mereghetti, Beatrice Palano (Milan, Italy) *Descriptional Complexity Issues Concerning Regular Languages*
15.00 – 15.30	Coffee break
15.30 – 16.30	Andreas Klein (Ghent, Belgium) *Visuelle Kryptographie*
18.00	Dinner

Theorietag *Automaten und Formale Sprachen*

Mittwoch, den 1. Oktober 2008

08.55 – 09.00	Begrüßung
09.00 – 09.30	Tobias Richter, Ludwig Staiger *Topological Language Operators*
09.30 – 10.00	Rudolf Freund, Sergey Verlan (*Tissue*) *P Systems Working in the k-Restricted Minimally Parallel Mode*
10.00 – 10.30	Remco Loos, Andreas Malcher, Detlef Wotschke *Some Results on the Descriptional Complexity of Splicing Systems*
10.30 – 11.00	Kaffeepause
11.00 – 11.30	Bianca Truthe *Zu akzeptierenden Netzwerken evolutionärer Prozessoren mit zwei Knotenarten*
11.30 – 12.00	Anna Kasprzik *Making Finite-State Methods Applicable to Languages Beyond Context-Freeness via Multi-dimensional Trees*
12.00 – 13.30	Mittagessen
13.30 – 14.00	Martin Kutrib, Hartmut Messerschmidt, Friedrich Otto *On Stateless Restarting Automata*
14.00 – 14.30	Dana Pardubská, Martin Plátek *Formal Translations, Parallel Communicating Grammar Systems and Restarting Automata*
14.30 – 15.00	Hartmut Messerschmidt *On Nonforgetting Restarting Automata That Are Deterministic and/or Monotone*

15.00 – 15.30	Kaffeepause
15.30 – 16.00	Alexander Okhotin, Christian Reitwießner *Conjunctive Grammars with Restricted Disjunction*
16.00 – 16.30	Dominik D. Freydenberger, Daniel Reidenbach *Inklusionsprobleme für Patternsprachen*
16.30 – 17.00	Johannes Schneider *Eindeutige löschende Homomorphismen in freien Monoiden*
17.00 – 17.15	Pause
17.15 – 18.00	Fachgruppensitzung
18.00	Abendessen

Donnerstag, den 2. Oktober 2008

09.00 – 09.30	Henning Fernau, Ralf Stiebe *Blinde Zählerautomaten auf ω-Wörtern*
09.30 – 10.00	Henning Bordihn, Martin Kutrib, Andreas Malcher *On the Computational Capacity of Parallel Communicating Finite Automata*
10.00 – 10.30	Hermann Gruber, Jan Johannsen *Communication Complexity and Regular Expression Size*
10.30 – 11.00	Kaffeepause
11.00 – 11.30	Stefan Gulan, Henning Fernau *Konstruktion endlicher Automaten aus regulären Ausdrücken*
11.30 – 12.00	Hermann Gruber, Markus Holzer *Language Operations with Regular Expressions of Polynomial Size*
12.00 – 13.30	Mittagessen

Workshop *Selected Topics in Theoretical Computer Science*

Descriptional Complexity Issues Concerning Regular Languages

Viliam Geffert

Department of Computer Science, P. J. Šafárik University
Jesenná 5, 04154 Košice, Slovakia
viliam.geffert@upjs.sk

Carlo Mereghetti and Beatrice Palano

Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano
via Comelico 39, 20135 Milano, Italy
{mereghetti,palano}@dsi.unimi.it

Abstract

In the realm of descriptional complexity, systems are compared on the basis of their size. Here, we consider two formalisms for representing regular languages: constant height pushdown automata and straight line programs for regular expressions. We show that their sizes are polynomially related. Comparing them with the sizes of finite state automata and regular expressions, we obtain optimal exponential and double exponential gaps, i.e., a more concise representation of regular languages.

1 Introduction

In our research, we have ben investigating *regular languages* from several points of view:

- We studied conditions on the resources of formal defining systems and computational devices for generating/recognizing regular languages. We provided an almost complete account for space and input head reversal lower bounds for Turing machines recognizing nonregular languages [8]. Yet, by suitably adapting techniques, we pointed out minimal space requirements for nonregular recognition on iterative

arrays [7]. In [17], a regularity condition for context-free grammars generating regular languages is provided.

- We designed very efficient parallel algorithms for parsing and ranking of regular languages. Yet, we related parallel efficiency with the algebraic structure of the syntactic monoids of regular languages [9, 10].
- We studied the expressive power of several models of quantum automata which, due to reversibility requirement, are typically less powerful than classical automata [2]. Yet, we proposed a general model of quantum automaton capable of characterizing regular languages [11].

In particular, we spent much effort in analyzing regular languages from the viewpoint of *descriptional complexity* [5]. In the realm of descriptional complexity, roughly speaking, language defining systems are studied by comparing their *size*. A well consolidated trend in the literature deals with descriptional complexity issues concerning *regular languages*.

Several systems for representing regular languages have been considered in the literature. Representation may be much more "economical" in one system than another. The oldest and most famous result in this sense is the optimal exponential gap between the size of a deterministic and nondeterministic finite state automaton [16, 18]. We provided many contributions aiming to compare the size of several types of finite state automata and other formalisms for regular languages [4, 12, 13, 14, 15].

Here, we take under consideration *constant height pushdown automata* and *straight line programs for a regular expression*. We provide simulations in both directions showing that their sizes *are polynomially related*. Furthermore, we emphasize optimal *exponential* and *double exponential* gaps with the size of finite state automata and regular expressions.

2 Preliminaries

In this section, we present the formalism we shall be dealing with. We assume the reader is familiar with the basic notions on formal language theory (see, e.g., [6]). The set of natural numbers, including zero, is denoted here by $\mathbf{N}$.

2.1 Straight line programs for regular expressions

A *regular expression*, defined over a given alphabet Σ, is: (i) $\emptyset$, ε, or any symbol $a \in \Sigma$, (ii) r_1+r_2, $r_1 \cdot r_2$, or r_1^*, if r_1 and r_2 are regular expressions.

The language represented by a given regular expression r, denoted by $L(r)$, is defined in the usual way [6]. With a slight abuse of terminology, we often identify a regular expression with the language it represents. Thus, Σ^* is the set of words on Σ, including the empty word ε. By $|w|$, we denote the length of a word $w \in \Sigma^*$ and by Σ^i the set of words of length $i \in \mathbf{N}$, with $\Sigma^0 = \{\varepsilon\}$ and $\Sigma^{\leq m} = \bigcup_{i=0}^{m} \Sigma^i$. By $|\Sigma|$, we denote the cardinality of the set Σ.

Definition 1 *The* size of a regular expression r *on* Σ*, denoted by* size(r)*, is the number of occurrences of symbols of* $\{\emptyset, \varepsilon\} \cup \Sigma$ *plus the number of occurrences of operations* $+, \cdot$ *and* $*$ *inside* r*.*

Example 1 For $r = a \cdot (a+b)^* + (a+b)^* \cdot b \cdot a^*$, we have size$(r) = 16$.

A convenient way for representing regular expressions is given by straight line programs (see, e.g., [1]). Given a set of variables $X = \{x_1, \ldots, x_\ell\}$, a *straight line program for regular expressions* (slp) on Σ is a finite sequence of instructions

$$P \;\equiv\; \text{instr}_1\,;\; \ldots\; \text{instr}_i\,;\; \ldots\; \text{instr}_\ell\,,$$

where the i-th instruction instr_i has one of the following forms:

(i) $x_i := \emptyset$, $x_i := \varepsilon$, or $x_i := a$ for any symbol $a \in \Sigma$,

(ii) $x_i := x_j + x_k$, $x_i := x_j \cdot x_k$, or $x_i := x_j^*$, for $1 \leq j, k < i$.

Such program P expands to the regular expression reg-exp$(P) = x_\ell$, obtained by nested macro-expansion of the variables $x_1, \ldots, x_{\ell-1}$, using the right parts of their instructions. Notice that a variable may be reused several times in the right parts. Such a number of occurrences is called a *fan-out* of the variable. The fan-out of x_ℓ is 0, while the fan-out of any other variable is at least 1, since, with the exception of x_ℓ, we can remove instructions defining variables not used at least once in some right part.

Definition 2 *The* size of a straight line program P *is the ordered pair* size$(P) =$ (length(P), fan-out(P))*, where* length(P) *denotes the number of instructions in* P*, and* fan-out(P) *the maximum fan-out of its variables.*

Example 2 Let us construct an slp for the regular expression in the Example 1:

$$\begin{aligned} P \;\equiv\;\; & x_1 := a;\;\; x_2 := b;\;\; x_3 := x_1 + x_2;\;\; x_4 := x_1^*;\;\; x_5 := x_3^*; \\ & x_6 := x_2 \cdot x_4;\;\; x_7 := x_1 \cdot x_5;\;\; x_8 := x_5 \cdot x_6;\;\; x_9 := x_7 + x_8\,. \end{aligned}$$

Clearly, reg-exp$(P) = x_9 = a \cdot (a+b)^* + (a+b)^* \cdot b \cdot a^*$, and size$(P) = (9, 3)$.

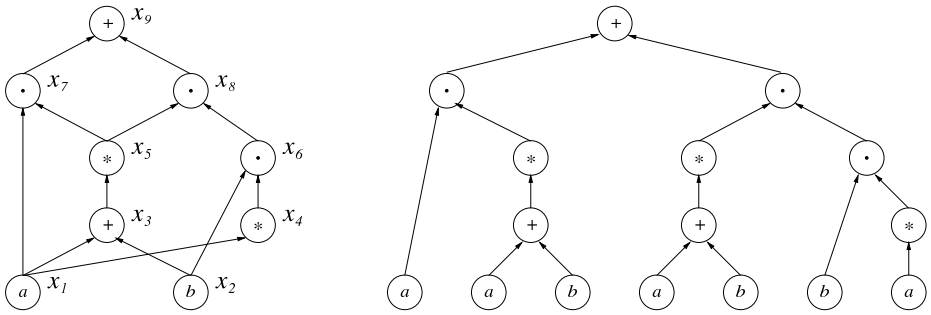


Figure 1: On the left, the directed acyclic graph D_P associated with the slp P introduced in Example 2. The corresponding classical representation of the regular expression from Example 1 as a binary tree on the right.

Any slp P can be associated with a *vertex-labeled directed acyclic graph* (dag) $D_P = (V, E)$, where the vertices in $V = \{v_1, \ldots, v_\ell\}$ correspond to the respective variables in $X = \{x_1, \ldots, x_\ell\}$. That is, a vertex v_i is labeled by $e \in \{\emptyset, \varepsilon\} \cup \Sigma$, whenever the i-th instruction is $x_i := e$, and by '$+$' or '$\cdot$', whenever this instruction is $x_i := x_j{+}x_k$ or $x_i := x_j{\cdot}x_k$, respectively. In the case of a binary operation, the directed arcs (v_j, v_i) and (v_k, v_i) are included in E, to connect v_i with its left and right sons, respectively. Similarly, v_i is labeled by '$*$', if the i-th instruction is $x_i := x_j^*$, with (v_j, v_i) included in E. (This idea is illustrated by Figure 1.)

From the definition of P, it is easy to see that D_P does not contain any directed cycle and that the fan-out of a variable establishes the out-degree of the corresponding vertex. So, there exists a unique *sink* v_ℓ (vertex without outgoing arcs) and some *sources* (vertices without ingoing arcs) labeled by $e \in \{\emptyset, \varepsilon\} \cup \Sigma$. We define the *depth* of D_P, $\mathrm{depth}(D_P)$, as the maximum length of a path from a source to the sink.

In what follows, we point out some relations between the sizes of an slp and its regular expression. Clearly, an slp with fan-out bounded by 1 is just an ordinary regular expression written down in a slightly different way.

Proposition 1 *For each slp P,* $\mathrm{length}(P) = \mathrm{size}(\text{reg-exp}(P))$ *if and only if* $\text{fan-out}(P) = 1$.

In general, however, straight line programs can be exponentially more succinct than regular expressions. The following example shows that, even with only fan-out 2, we get an exponential gap.

Example 3 Consider the slp P_ℓ on $\Sigma = \{a\}$:

$$P_\ell \;\equiv\; \quad x_1 := a;\;\; x_2 := x_1{\cdot}x_1;\;\; x_3 := x_2{\cdot}x_2;\;\; \ldots\;\; x_\ell := x_{\ell-1}{\cdot}x_{\ell-1}\,.$$

It can be immediately seen that fan-out$(P_\ell) = 2$ and reg-exp$(P_\ell) = a^{2^{\ell-1}}$. Thus, for any $\ell \geq 1$, we obtain size(reg-exp$(P_\ell)) = 2^{\text{length}(P_\ell)} - 1$.

This latter example establishes the *optimality* of the following general result:

Proposition 2 *Let P and P' be two equivalent slps such that* fan-out$(P') = 1$. *Then,* length$(P') \leq 2^{\text{depth}(P)}$.

2.2 Constant height pushdown automata

It is well known that the regular expressions (hence, slps as well) represent the class of *regular languages*. This class can also be represented by deterministic or nondeterministic finite state automata (dfas and nfas, resp.) [6]. In the literature, a *nondeterministic pushdown automaton* (npda) is usually obtained from an nfa by adding a pushdown store, containing symbols from Γ, the pushdown alphabet. For technical reasons, we shall introduce the npdas in the following form, where moves manipulating the pushdown store are clearly distinguished from those reading the input tape: a npda is a 6-tuple $A = \langle Q, \Sigma, \Gamma, H, q_0, F \rangle$, where $Q, \Sigma, \Gamma, q_0, F$ are defined as usual, while $H \subseteq Q \times (\{\varepsilon\} \cup \Sigma \cup \{+, -\} \cdot \Gamma) \times Q$ is the *transition relation* with the following meaning:

(i) $(p, \varepsilon, q) \in H$: A reaches the state q from the state p without using the input tape or the pushdown store,

(ii) $(p, a, q) \in H$: A reaches the state q from the state p by reading the symbol a from the input, not using the pushdown store,

(iii) $(p, -X, q) \in H$: if the symbol on top of the pushdown is X, A reaches the state q from the state p by popping X, not using the input tape,

(iv) $(p, +X, q) \in H$: A reaches the state q from the state p by pushing the symbol X onto the pushdown, not using the input tape.

Such machine does not use any initial pushdown symbol: an accepting computation begins in the state q_0 with the empty pushdown store and input head at the beginning, and ends in a final state $q \in F$ after reading the entire input.

A *deterministic pushdown automaton* (dpda) comes from npda by claiming it can never get into a situation in which more than one instruction can

be executed. (As an example, a dpda cannot have a pair of instructions of the form (q, ε, p_1) and (q, a, p_2).)

It is not hard to see that any npda in the "classical" form can be turned into this latter form and vice versa, preserving determinism in the case of dpdas. At the cost of one more state, we can transform our npdas so that they accept by entering a *unique* final state at the end of input processing, with *empty* pushdown store. Notice however that the following transformation does not preserve determinism.

Lemma 1 *For any npda $A = \langle Q, \Sigma, \Gamma, H, q_0, F\rangle$, there exists an equivalent npda $A' = \langle Q \cup \{q_f\}, \Sigma, \Gamma, H', q_0, \{q_f\}\rangle$, where A' accepts by entering the unique final state $q_f \notin Q$ with empty pushdown store at the end of the input.*

Given a constant $h \in \mathbf{N}$, we say that the npda A is of pushdown height h if, for any word in $L(A)$, there exists an accepting computation along which the pushdown store never contains more than h symbols. From now on, we shall consider *constant height npdas* only. Such machine will be denoted by a 7-tuple $A = \langle Q, \Sigma, \Gamma, H, q_0, F, h\rangle$, where $h \in \mathbf{N}$ is a constant denoting the pushdown height, and all other elements are defined as above. By definition, the meaning of the transitions in the form (iv) is modified as follows:

(iv') $(p, +X, q) \in H$: *if the current pushdown store height is smaller than h*, then A reaches the state q from the state p by pushing the symbol X onto the pushdown, not using the input tape.

Thus, this kind of transitions is disabled, if the current pushdown height is equal to h. A constant height npda can be replaced by an equivalent standard npda (without a built-in limit h on the pushdown) by storing, in the finite control states, a counter recording permitted pushdown heights (i.e., a number ranging within $\{0, \ldots, h\}$), hence, paying by a constant increase in the number of states.

Note that, for $h = 0$, the definition of constant height npda exactly coincides with that of an nfa, as one may easily verify. Moreover, Lemma 1 holds for constant height npdas as well, which enables us to consider acceptance by a *single final state* and, at the same time, with *empty pushdown store*. Therefore, from now on, a constant height npda will assume the form $A = \langle Q, \Sigma, \Gamma, H, q_0, \{q_f\}, h\rangle$.

Definition 3 *The* size of a constant height npda $A = \langle Q, \Sigma, \Gamma, H, q_0, \{q_f\}, h\rangle$ *is the ordered triple* $\mathrm{size}(A) = (|Q|, |\Gamma|, h)$.

Observe that this definition immediately gives that the size of an nfa is completely determined by the number of its states, since it is $(|Q|, 0, 0)$.

3 From constant height npdas to slps and back

In this section, we give the results and main ideas of the transformations from constant height npdas to slps and vice versa.

Theorem 1 *Let* $A = \langle Q, \Sigma, \Gamma, H, q_0, \{q_f\}, h\rangle$ *be a constant height npda. Then there exists an slp* P_A *such that* reg-exp(P_A) *denotes* $L(A)$*, with* length$(P_A) \leq \mathcal{O}(h \cdot |Q|^4 \cdot |\Gamma| + |Q|^2 \cdot |\Sigma|)$ *and* fan-out$(P_A) \leq |Q|^2 + 1$*. That is, for regular languages over a fixed alphabet, the size of* P_A *is polynomial in the size of* A*.*

Proof. (The main idea of the conversion) Let our constant height npda be $A = \langle \{q_1, \ldots, q_k\}, \Sigma, \Gamma, H, q_1, \{q_k\}, h\rangle$. For each $i, j \in \{1, \ldots, k\}$, $s \in \{0, \ldots, k\}$, and $t \in \{0, \ldots, h\}$, we define $[q_i, s, t, q_j]$ as the set of strings $x \in \Sigma^*$ such that, for each of them, there exists at least one computation with the following properties.

- The computation begins in the state q_i, with the pushdown empty.
- After reading the entire string x from the input, the computation ends in the state q_j, with the pushdown empty again.
- Any time the pushdown is empty during this computation, the current finite control state is from the set $\{q_1, \ldots, q_s\}$. (This restriction does not apply to q_i, q_j themselves.) For $s = 0$, the pushdown is never empty in the meantime.
- During this computation, the pushdown height never exceeds t.

An algorithm can be given which dynamically constructs regular expressions describing all sets $[q_i, s, t, q_j]$. The ultimate goal is to obtain $[q_1, k, h, q_k]$, the regular expression for $L(A)$. First, we construct $[q_i, 0, 0, q_j]$ for each q_i, q_j, basically by a direct inspection of the transitions in H. Then, we gradually increment the parameter s from 1 to k, thus obtaining $[q_i, k, 0, q_j]$ for each q_i, q_j. Second, we upgrade from the parameters $k, t-1$ to parameters $0, t$, and then from parameters s, t to $s+1, t$, which leads up to $[q_i, k, h, q_j]$ for each q_i, q_j. ∎

Theorem 2 *Let* P *be an slp. Then there exists a constant height npda* $A_P = \langle Q, \Sigma, \Gamma, H, q_0, \{q_f\}, h\rangle$ *such that* $L(A_P)$ *is denoted by* reg-exp(P) *and the size of* A_P *is linear in the size of* P*. In particular,* $|Q| < 3 \cdot$length(P)*,*

$|\Gamma| = \text{fan-out}(P)$, *and* $h < \text{length}(P)$. *More precisely,* h *is to the maximum number of vertices with fan-out greater than* 1 *along paths from sources to the sink.*

Proof. (The main idea of the conversion) Let P be our slp with variables $\{x_1, \ldots, x_\ell\}$ on Σ. We consider the associated dag D_P, as described in Section 2.1, where the vertex v_i corresponds to the variable x_i. The enumeration of the variables in P induces a topological ordering on the vertices of D_P. Now we proceed as follows.

For $i = 1, \ldots, \ell$, we construct an npda $A_i = \langle Q_i, \Sigma, \Gamma_i, H_i, q_{0,i}, \{q_{f,i}\}\rangle$ such that $L(A_i)$ is exactly the language denoted by reg-exp(x_i), a regular expression obtained by expanding the dag rooted in v_i. For a source node v_i, we define an "elementary" npda without a pushdown store — actually an nfa. An npda for an inner node is constructed inductively, using, as subprograms, npdas for vertices that are topologically smaller. The desired npda is A_ℓ. We start with the construction for sources. ∎

4 Constant height pdas vs. finite state automata

Here we compare the sizes of constant height pushdown automata and the standard finite state automata. In what follows, npdas (but not dpdas) are in the form stated in Lemma 1, i.e., they accept by entering a unique final state with empty pushdown. First of all, we point out an exponential upper bound on the size of nfas (dfas) simulating constant height npdas (dpdas, respectively).

Proposition 3 *For each constant height npda* $A = \langle Q, \Sigma, \Gamma, H, q_0, \{q_f\}, h\rangle$, *there exists an equivalent nfa* $A' = (Q', \Sigma, H', q_0', \{q_f'\})$ *with* $|Q'| \leq |Q| \cdot |\Gamma^{\leq h}|$. *If* $B = \langle Q, \Sigma, \Gamma, H, q_0, F, h\rangle$ *is a constant height dpda, we can construct an equivalent dfa with no more than* $|Q| \cdot |\Gamma^{\leq h}|$ *states.*

By Proposition 3 and the usual subset construction, one immediately gets:

Corollary 1 *Let* $A = \langle Q, \Sigma, \Gamma, H, q_0, \{q_f\}, h\rangle$ *be a constant height npda. Then there exists an equivalent dfa with no more than* $2^{|Q| \cdot |\Gamma^{\leq h}|}$ *states.*

We are now going to show that the simulation costs in Proposition 3 and Corollary 1 are *optimal* by exhibiting two witness languages with matching exponential and double exponential gaps. For a string $x = x_1 \cdots x_n$, let

$x^R = x_n \cdots x_1$ denote its reverse. Given an $h > 0$, an alphabet Γ, and two separator symbols $\sharp, \$ \notin \Gamma$, we define the language

$$L_{\Gamma,h} = \{\sharp w_1 \sharp w_2 \sharp \cdots \sharp w_m \$ w : w_1, \ldots, w_m \in \Gamma^*, w \in \Gamma^{\leq h}, \text{ and } w \in \bigcup_{i=1}^{m} \{w_i^R\}\} .$$

We begin by providing upper bounds on the size of machines accepting $L_{\Gamma,h}$:

Lemma 2 *For each $h > 0$ and each alphabet Γ: (i) The language $L_{\Gamma,h}$ can be accepted by an npda with $\mathcal{O}(1)$ states, pushdown alphabet of size $|\Gamma|$, and constant height h. (ii) The language $L_{\Gamma,h}$ can also be accepted by an nfa (or dfa) with $\mathcal{O}(|\Gamma^{\leq h}|)$ states (or $2^{\mathcal{O}(|\Gamma^{\leq h}|)}$ states, respectively).*

Now we show that the sizes for $L_{\Gamma,h}$ stated in Lemma 2 (ii) are optimal.

Lemma 3 *For each $h > 0$ and each alphabet Γ, any dfa (or nfa) accepting the language $L_{\Gamma,h}$ must use at least $2^{|\Gamma^{\leq h}|}$ states (or $|\Gamma^{\leq h}|$ states, respectively).*

Let us now show the optimality of the exponential simulation cost of constant height dpdas by dfas presented in Proposition 3. Consider the following witness language: given an $h > 0$, an alphabet Γ, and a separator symbol $\sharp \notin \Gamma$, let

$$D_{\Gamma,h} = \{w \sharp w^R : w \in \Gamma^{\leq h}\} .$$

Lemma 4 *For each $h > 0$ and each alphabet Γ: (i) The language $D_{\Gamma,h}$ is accepted by a dpda with $\mathcal{O}(1)$ states, pushdown alphabet Γ and constant height h, and also by a dfa with $2 \cdot |\Gamma^{\leq h}| + 1$ states. (ii) Any dfa accepting the language $D_{\Gamma,h}$ must have at least $|\Gamma^{\leq h}|$ states.*

5 The final picture

In conclusion, in Figure 2, we sum up the main relations on the sizes of the different types of formalisms defining regular languages we considered in this paper. Let us briefly discuss the simulation costs displayed in this figure. The costs of the following simulations are asymptotically optimal:

- h-dpda $\rightarrow$ dfa: the exponential cost comes from Proposition 3, while its optimality follows from Lemma 4.
- h-npda $\rightarrow$ nfa: exponential cost, by Proposition 3 and Lemmas 2 (i) and 3.

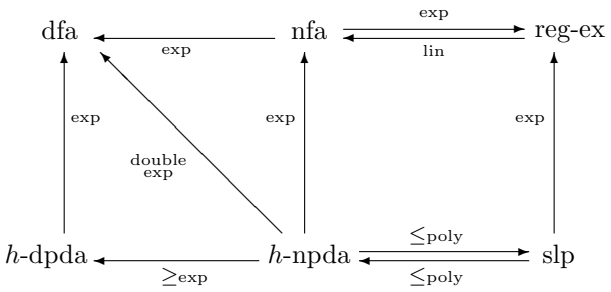


Figure 2: Costs of simulations among different types of formalisms defining regular languages. Here h-dpda (h-npda) denotes constant height dpda (npda, respectively). For clarity, some trivial linear conversions are omitted.

- slp $\to$ reg-ex: the exponential cost comes from Proposition 2, while its optimality follows, e.g., from Example 3 in Section 2.1.
- h-npda $\to$ dfa: the double exponential cost was presented by Corollary 1, its optimality follows from Lemmas 2 (i) and 3.
- nfa $\to$ dfa: the exponential cost is from [18], its optimality from [16].
- nfa $\leftrightarrow$ reg-ex: the linear cost for the "$\leftarrow$" conversion comes directly from the Kleene's Theorem, while its optimality follows trivially by considering, e.g., the regular expression a^n, for a fixed $n > 0$. The exponential cost for the converse direction and its optimality is from [3].

The costs of the following simulations are not yet known to be optimal:

- h-npda $\leftrightarrow$ slp: Theorems 1 and 2 prove polynomial upper bounds for both directions.
- h-npda $\to$ h-dpda: the exponential lower bound comes from the following consideration: a sub-exponential cost of h-npda $\to$ h-dpda conversion together with the optimal exponential cost for h-dpda $\to$ dfa would lead to a sub-double exponential cost of h-npda $\to$ dfa, thus contradicting the optimality of the double exponential cost. In general, for h-npda $\to$ h-dpda conversion, we conjecture a double exponential optimal cost.

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[2] A. Bertoni, C. Mereghetti, B. Palano. Quantum computing: 1-way quantum automata. In: *7th International Conference on Developments in Language Theory, Proceedings*, Lecture Notes in Computer Science 2710, pp. 1–20, Springer 2003.

[3] A. Ehrenfeucht, P. Zieger. Complexity measures for regular expressions. *J. Computer and System Sciences*, 12:134–146, 1976.

[4] V. Geffert, C. Mereghetti, G. Pighizzini. Converting two–way nondeterministic unary automata into simpler automata. *Theoretical Computer Science*, 295:189–203, 2003.

[5] J Goldstine, M. Kappes, C. Kintala, H. Leung, A. Malcher, D. Wotschke. Descriptional complexity of machines with limited resources. *J. Universal Computer Science*, 8:193234, 2002.

[6] J.E. Hopcroft, R. Motwani, J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001.

[7] A. Malcher, C. Mereghetti, B. Palano. Sublinearly space bounded iterative arrays. In: E. Csuhaj-Varjú, Z. Ésik (Eds.), *12th International Conference on Automata and Formal Languages (AFL'08), Proceedings,* pp. 292–301, Balatonfüred, Hungary, 2008.

[8] C. Mereghetti. Testing the descriptional power of small Turing machines on nonregular language acceptance. *Int. J. Foundations of Computer Science*, 19:827–843, 2008.

[9] C. Mereghetti, B. Palano. Threshold circuits for iterated matrix product and powering. *Theoret. Inf. Appl.*, 34:39–46, 2000.

[10] C. Mereghetti, B. Palano. The parallel complexity of deterministic and probabilistic automata. *J. Aut., Lang. Comb.,* 7:95–108, 2002.

[11] C. Mereghetti, B. Palano. Quantum finite automata with control language. *Theoretical Informatics and Applications*, 40:315–332, 2006.

[12] C. Mereghetti, B. Palano. Quantum automata for some multiperiodic languages. *Theoretical Computer Science*, 387:177–186, 2007.

[13] C. Mereghetti, B. Palano, G. Pighizzini. Note on the succinctness of deterministic, nondeterministic, probabilistic and quantum finite automata. *Theoretical Informatics and Applications*, 35:477–490: 2001.

[14] C. Mereghetti, G. Pighizzini. Two–way automata simulations and unary languages. *J. Aut., Lang. Comb.*, 5:287–300, 2000.

[15] C. Mereghetti, G. Pighizzini. Optimal simulations between unary automata. *SIAM Journal on Computing,* 30:1976–1992, 2001.

[16] A.R. Meyer, M.J. Fischer. Economy of description by automata, grammars, and formal systems. *IEEE 12th Symp. Switching and Automata Theory*, 188–191, 1971.

[17] B. Palano. A regularity condition for context-free grammars. *Int. J. Foundations of Computer Science*, 19:845–857, 2008.

[18] M. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res. Develop.,* 3:114–125, 1959.

Visuelle Kryptographie

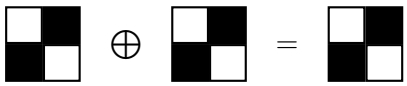
Andreas Klein

Department of Pure Mathematics and Computer Algebra, Ghent University
Krijgslaan 281-S22, 9000 Ghent, Belgium
klein@cage.ugent.be

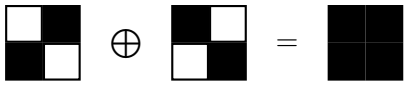
Visuelle Kryptographie ist ein 1994 von Naor und Shamir erfundenes Verschlüsselungsverfahren, bei dem die Entschlüsselung ohne Computerhilfe vorgenommen werden kann. In der einfachsten Version wird eine Schlüsselfolie und eine Nachrichtenfolie erzeugt.

Dabei wird jeder Bildpunkt (eines Schwarz-Weiß-Bilds) in 4 Subpixel zerlegt. Dies geschieht so, daß sowohl auf der Folie als auch auf dem Display des Automaten jeder Bildpunkt gleichwahrscheinlich durch eine der beiden folgenden Subpixel-Kombinationen dargestellt wird.

Die Darstellung der Pixel auf der Folie und dem Display sind jedoch nicht von einander unabhängig. Soll ein weißer Bildpunkt dargestellt werden, so wird auf beiden die gleiche Subpixel-Kombination genommen, z.B.:



Für einen schwarzen Bildpunkt müssen die Subpixel-Kombinationen verschieden sein, z.B.:



Beim Übereinanderlegen der Folien erkennt man ein Bild. Ein Angreifer der nur eine Folie kennt sieht nur eine zufällige Verteilung von den beiden möglichen Subpixel-Kombinationen. Ein Teilbild alleine liefert daher keine Informationen über das codierte Bild. Das Verfahren ist also wie auch das

bekanntere One-Time-Pad absolut sicher, und wie beim One-Time-Pad ist in diesem Fall der Schlüssel genauso groß wie die verschlüsselte Nachricht.

Wie schnell das Verfahren unsicher wird, wenn mehr als ein Bild mit derselben Folie verschlüsselt werden soll, zeigt das folgende Beispiel.

Angenommen wir wollen die beiden folgenden Bilder mit der rechts abgebildeten geheimen Folie verschlüsseln.

Wendet man das oben beschriebene Verfahren an, um die beiden Bilder zu verschlüsseln, so kann ein Angreifer aus einem geheimen Bild keine Information über das codierte Bild erhalten. Kennt der Angreifer jedoch beide geheimen Bilder (siehe unten links), so muß er sie nur auf Folien kopieren und die Folien übereinanderlegen, um den Inhalt der beiden Bilder zu erkennen (siehe unten rechts).

Das Verfahren hat unzählige Erweiterungen (mehr als zwei Folien; auf jeder Folie ist ein Bild zu sehen, dass in keinen Zusammenhang mit dem geheimen Bild steht; farbige Folien; usw.), die oft zu interessanten kombinatorischen Problem führt. Bezüge zu anderen Gebieten (Codierungstheorie, lineare Optimierung) sind ebenfalls reichlich vorhanden.

In meinem Vortag werde ich einige dieser Erweiterungen vorstellen.

References

[1] Andreas Klein Visuelle Kryptographie Springer-Verlag, 2007

[2] Moni Naor and Adi Shamir. Visual cryptography. In Alfredo De Santis, editor, *Advances in cryptology - EUROCRYPT '94*, volume 950 of *Lect. Notes Comput. Sci.*, pages 1–12. Springer-Verlag, 1995.

About the Injectivity of the Global Function of Cellular Automata in the Hyperbolic Plane

Maurice Margenstern

Laboratoire d'Informatique Théorique et Appliquée
Université de Metz, I.U.T. de Metz
Département d'Informatique
Île du Saulcy
57045 Metz Cedex, France
margens@univ-metz.fr

Summary

The global function of a cellular automaton on the line or in the Euclidean plane is an object which have been studied for a long time, already from the 60's of the previous century. This function was characterized in general terms by Hedlund and properties about the connections between injectivity and surjectivity were also studied and characterized. A bit later, in 1994, J. Kari proved that the injectivity of the global function of a cellular automaton is undecidable. This proof relies on a very sophisticated construction and makes use of the undecidability of the tiling problem in the Euclidean plane which was established in the 66's and 70's. It was used by J. Kari to derive several corollaries about the undecidability of other properties of the global funcion of a cellular automaton, namely: surjectivity, bijectivity and reversiblity as the latter notions are equivalent for these cellular automata. Cellular automata in the hyperbolic plane were introduced in 1999 by a joint work of the author with Kenichi Morita. A bit later, the author devised a technology which definitely fixed a simple way to locate tiles in a regular tiling of the hyperbolic plane, generated by reflections in the edges of a regular polygon and its replicas. Still later, the global function of a cellular automaton in the hyperbolic plane was defined and then characterized by the author. In the present work we prove that the injectivity of the global function of a cellular automaton in the hyperbolic plane is also

undecidable. This proof also makes use of the undecidability of the tiling problem in the hyperbolic plane which was established in 2007 by the author (and independently, by J. Kari). However, this result about the undecidability of the injectivity of the global function of a cellular automaton has not the impact of the corresponding result in the Euclidean plane: indeed, for cellular automata in the hyperbolic plane, the connections established between injectivity and surjectivity in the Euclidean plane no more hold.

Symport/Antiport Systems and P Automata – Membrane Systems for the Characterization of Formal Languages

György Vaszil

Computer and Automation Research Institute, Hungarian Academy of Sciences
Kende utca 13-17, 1111 Budapest, Hungary
vaszil@sztaki.hu

Abstract

We review how variants of the unconventional computational models of symport/antiport membrane systems and P automata can be used to characterize formal languages of strings, that is, structures having an essentially different nature from multisets, the mathematical objects which these systems are basically able to process.

1 Introduction

Membrane systems, or P systems were introduced in [20] as computing models inspired by the functioning of the living cell. Their main components are membrane structures consisting of membranes hierarchically embedded in the outermost skin membrane. Each membrane encloses a region containing a multiset of objects and possibly other membranes. Each region has an associated set of operators working on the objects contained by the region. Several variants of the basic notion have been introduced and studied proving the power of the framework, see the monograph [21] for a summary of notions and results of the area.

One of the most interesting variants of the model was introduced in [19] called P systems with symport/antiport. In these systems the modification of the objects present in the regions is not possible, they may only move through the membranes from one region to another. The movement is described by communication rules called symport/antiport rules associated to the regions.

2 Preliminaries

Let Σ be a set of symbols called alphabet, and let Σ^* be the set of all words over Σ, that is, the set of finite strings of symbols from Σ, and let $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ where ε denotes the empty word. The set of finite subsets of Σ is denoted by 2^Σ.

Let U be a set of objects, and let $\mathbb{N}$ denote the set of non-negative integers A multiset is a mapping $M : U \to \mathbb{N}$ which assigns to each object $a \in U$ its multiplicity $M(a)$ in M. The support of M is the set $supp(M) = \{a \mid M(a) \geq 1\}$. If $supp(M)$ is a finite set, then M is called a finite multiset. The set of all finite multisets over the set U is denoted by U°.

A multiset M over the finite set of objects V can be represented as a string w over the alphabet V with $|w|_a = M(a)$ where $a \in V$ and $|w|_a$ denotes the number of occurrences of the symbol a in the string w, and with ε representing the empty multiset. Let $|w|$ denote the length of w, that is, the cardinality of the multiset represented by w.

A symport rule is of the form (x, in) or $(x, out), x \in V^\circ$. If such a rule is present in a region i, then the objects of the multiset x can enter from the parent region or can leave to the parent region, respectively. An antiport rule is of the form $(x, in; y, out), x, y \in V^\circ$, in this case, objects of x enter from the parent region and in the same step, objects of y leave to the parent region. All types of these rules might be equipped with a promoter or inhibitor multiset, denoted as $(x, in)|_Z, (x, out)|_Z$, or $(x, in; y, out)|_Z$, with $x, y \in V^\circ, Z \in \{z, \neg z \mid z \in V^\circ\}$, where if $Z = z$ then the rules can only be applied if region i contains the objects of multiset z, or if $Z = \neg z$, then region i must not contain any of the elements of z. (For more on symport/antiport see [19], for the use of promoters see [17].)

A *P system with symport/antiport* of degree $n \geq 1$ is a construct

$$\Pi = (V, \mu, E, w_1, \ldots, w_n, R_1, \ldots, R_n, F, in)$$

where

- V is an alphabet of objects,
- μ is a membrane structure of n membranes,
- $E \subseteq V$ is a set of objects (the ones which can be found in the environment in an arbitrary number of copies),
- $w_i \in V^\circ,\ 1 \leq i \leq n$, are the initial contents of the n regions,

- R_i, $1 \leq i \leq n$, are the sets of symport/antiport rules associated to the regions,
- F is a set of final configurations, and
- $in \in \{0, 1, \ldots, n\}$ is the label of the input membrane, where if $i = 0$, the input is read from the environment.

The $n + 1$-tuple of finite multisets of objects present in finite number of copies in the environment and in the n regions of the P system Π describes a *configuration* of Π with $(\varepsilon, w_1, \ldots, w_n) \in (V^\circ)^{n+1}$ being the initial configuration.

The *transition mapping* of a symport/antiport P systems is a mapping $\delta : V^\circ \times (V^\circ)^{n+1} \to (V^\circ)^{n+1}$. For two configurations $c = (u_0, u_1, \ldots, u_n)$, $c' = (u'_0, u'_1, \ldots, u'_n)$ and a multiset $u \in V^\circ$,

$$\delta(u, (u_0, u_1, \ldots, u_n)) = (u'_0, u'_1, \ldots, u'_n)$$

holds if there exists a maximal set of rules which, when applied in a parallel and synchronous manner in the regions, transfer the system from configuration (state) c to c' with input u, that is, while the multiset u enters the system from the environment.

3 Universality

It is natural to use a symport/antiport system for describing sets of integers or vectors since there is a straightforward correspondence between numbers and the cardinalities of multisets. All we need is to specify an input region, in, which holds the input at the beginning of the computation represented as the multiplicity of a certain object, and to prescribe the set of final configurations, F.

As we might expect, this framework is quite strong, these systems are clearly universal, see [19]. There were several results, some of them optimal, obtained for bounding the number of necessary membranes and the size of antiport rules, [7, 8, 12], proving that symport rules are sufficient, [11, 1], or showing that the maximal power can also be reached with "minimal cooperation", that is, with symport and antiport rules using singleton multisets, [15, 2, 10, 16, 24, 11, 1]. The proofs of these results are usually based on the simulation of register machines, thus, the techniques can also be used to bound the number of rules by simulating universal machines, as done in [4].

To use symport/antiport systems for the characterization of string languages, we need to establish a correspondence between the object multisets that the systems are processing, and the strings over some alphabet we want to use.

The first idea we consider is introduced in [9] and called *analyzing P system*. An analyzing P system is a symport/antiport system where

- there is a set of terminal objects, $T \subseteq V$,
- $in = 0$, which means that the input is read from the environment, and
- F is the set of halting configurations, that is, configurations in which no rule can be applied.

An analyzing P system accepts the strings of terminal symbols which enter the system in the subsequent computational steps. More formally expressed, the language accepted by an analyzing P system Π is

$$L(\Pi) \quad = \quad \bigcup str_T(u_1) \cdot str_T(u_2) \cdot \ldots \cdot str_T(u_t)$$

where $c_0, c_1, \ldots, c_t$ is a sequence of configurations with $\delta(u_{i+1}, c_i) = c_{i+1}$ for all $0 \leq i \leq t-1$, $c_t \in F$ and c_0 being the initial configuration. Moreover, for a multiset $u \in V^\circ$, the set $str_T(u) \subseteq T^*$ is the set of terminal strings corresponding to the multiset $u' \in T^\circ$ of terminal symbols from u, that is, $u' \subseteq u$ and $u - u' \in (V - T)^\circ$.

Theorem 1 ([9]) *Any language recursively enumerable language can be accepted by an analyzing P system having one membrane.*

The idea of the proof of this statement is to create a numerical encoding of the input string, and then execute a simulation of a register machine.

4 Restricted variants

For the computational completeness of the model, the differentiation between terminal and nonterminal objects is essential because this way it is guaranteed that the system can work with arbitrary large multisets independent from the length of the input string. And this is necessary, because the nonterminal objects provide the "workspace" for the computation. It is interesting to consider what happens if the available resources of these

types of systems are restricted. In the following we review two possible approaches.

Consider first the model called *exponential-space symport/antiport acceptor* introduced in [13]. Such a system is a symport/antiport system with

- a set of terminal objects $T \subseteq V$ containing a distinguished symbol \$,
- $in = 0$, which means that the input is read from the environment,
- rules of the following four types in the set R_1 corresponding to the skin region:
  1. $(u, in; v, out)$, $u, v \in (V - T)^\circ$, $|v| \geq |u|$,
  2. $(ua, in; v, out)$, $u, v \in (V - T)^\circ$, $|v| \geq |u|$, and $a \in T$,
  3. $(u, in; v, out)|_a$, $u, v \in (V - T)^\circ$, $a \in T$,
  4. for every $a \in T$ there is at least one rule of the form $(u, in; a, out)$,
- rules of the form $(u, in; v, out)$, $u, v \in (V - T)^\circ$, in the regions different from the skin region.

The system accepts a string $a_1 \dots a_n\$$, $a_i \in T - \{\$\}$, $1 \leq i \leq n$, if the terminal symbols are brought into the system from the environment in the required order (by rules of type 2) and after reading the end marker \$, the computation halts. Thus, the language accepted by an exponential-space symport/antiport acceptor Π is

$$L(\Pi) = \bigcup str_T(u_1) \cdot str_T(u_2) \cdot \ldots \cdot str_T(\bar{u}_t)$$

where $c_0, c_1, \dots, c_t$ is a sequence of configurations with $\delta(u_{i+1}, c_i) = c_{i+1}$, $\$ \notin u_i$ for all $0 \leq i \leq t-1$, $\bar{u}_t = u_t - \$$, and where c_0 is the initial configuration, $c_t \in F$, and $str_T(u) \in T^*$ is the set of strings corresponding to the terminal objects of the multiset u, as defined above.

Again, the terminal-nonterminal distinction is essential, but due to the restricted form of the rules, the workspace which can be used by such systems is not arbitrary.

Theorem 2 ([13]) *A language L is accepted by an exponential-space symport/antiport acceptor if and only if L is context-sensitive.*

Moreover, as it is also shown in [13], both the number of membranes, and the number of objects in the object alphabet induce an infinite hierarchy of accepting power of exponential space symport/antiport acceptors.

By restricting further the form of the rules allowed to be used, we also obtain the following.

Theorem 3 ([13]) *A language L is regular if and only if it can be accepted by an exponential-space symport/antiport acceptor using only rules of type 1. and type 2.*

Now we continue in an other direction which was proposed by the introduction of P automata in [5].

A *P automaton* is a symport/antiport system with the following properties.

- $in = 0$, which means that the input is read from the environment,
- F defines the (not necessarily halting) final configurations, as $F = (F_1, \ldots, F_n)$ where $F_i \subseteq V^\circ,\ 1 \leq i \leq n$, are either finite sets of multisets over V, or $F_i = V^\circ$

A configuration $c = (v_0, v_1, \ldots, v_n)$ is said to be final, denoted as $c \in F = (F_1, \ldots, F_n)$, if $v_i \in F_i,\ 1 \leq i \leq n$.

Let also $f : V^\circ \to T^*$ be a mapping which maps nonempty multisets in V° to nonempty words over the alphabet T and $f(u) = \varepsilon$ if and only if u is the empty multiset.

A language $L \subseteq T^*$ is accepted by the P automaton Π if it is

$$L(\Pi, f) = \{f(u_1) \cdot f(u_2) \cdot \ldots \cdot f(u_t) \in T^* \mid \text{there is } c_t \in F \text{ and a sequence } c_i \text{ with } \delta(u_{i+1}, c_i) = c_{i+1} \text{ for all } 0 \leq i \leq t-1\},$$

where c_0 is the initial configuration, δ is the transition mapping of Π.

The requirement that a P automaton should not make any distinction between terminal and nonterminal objects, that is, that they should not completely discard any of the multisets imported in any of the steps of the computation from the accepted language, is reflected in the requirement that the mapping f only maps the empty multiset to $\{\varepsilon\}$, because it means that all nonempty input multisets are taken into account when the string of the accepted language is formed.

Of course, the mapping f should be in some sense simple if we would like to make sure that the computing power of the P automaton lies in the symport/antiport system and not in f itself. For now, let us fix the alphabet as $T = V$ and the mapping as $f_1(u) = a$ for $u = a^k,\ k \geq 1$, with $f_1(\emptyset) = \varepsilon$.

Theorem 4 ([3])

1. *For any context-sensitive language L, a P automaton Π can be constructed with object alphabet V, and with four membranes, such that $L = L(\Pi, f_1)$ for a mapping f_1 defined as above.*

2. *For any P automaton* Π *with object alphabet* V *and mapping* $f : V^\circ \to T^*$ *for some alphabet* T*, such that* f *is linear-space computable, the language* $L(\Pi, f) \subseteq T^*$ *is context-sensitive.*

We might also consider variations of P automata which restrict the forms of the rules. The notion of *P finite automaton* was defined in [6] as a P automaton where

- the object alphabet $V \cup \{a\}$ contains a distinguished symbol a,
- the set R_1 corresponding to the skin region contains rules of the form $(x, in; y, out)|_Z$ with $x \in \{a\}^\circ$, $y \in (V \cup \{a\})^\circ$, $Z \in \{z, \neg z\}$, $z \in V^\circ$, and
- if $i \neq 1$, the set R_i contains rules of the form $(x, in; y, out)|_Z$ with $Z \in \{z, \neg z\}$, $x, y, z \in V^\circ$.

As we can see, P finite automata can only input multisets of the form a^k, containing several copies of the distinguished symbol a. Therefore, it is appropriate if we define the mapping of the input multisets to the alphabet $T = \{a_1, a_2, \ldots\}$ as $f_2 : \{a\}^\circ \to T^*$ with $f_2(a^k) = \{a_k\}$, $k \geq 1$, and $f_2(\emptyset) = \varepsilon$ for the empty multiset.

As it is proved in [6] the rule restrictions introduced in the model of P finite automata also characterize the class of regular languages.

Theorem 5 ([6]) *A language* L *is regular if and only if there is a P finite automaton* Π *with object alphabet* $V \cup \{a\}$*, such that* $L = L(\Pi, f_2)$ *for a mapping* f_2 *defined as above.*

5 Unconventional aspects of P automata

In this section, we would like to propose a topic which is based on one of the unconventional aspects of membrane systems, that is, to use symport/antiport systems for the description of languages over infinite alphabets. The idea comes very naturally if we recall that that the language accepted by these systems corresponds to the sequence of multisets entering during a successful computation, and notice that the number of possible multisets which make up this sequence, that is, the number of possible symbols which make up the accepted string is not fixed in advance, but it can be arbitrary high.

If we think in terms of P automata, the set of finite multisets over V, that is, the domain of the mapping f is infinite, so its range could also

easily be defined to be infinite. This idea is explored in the case of P finite automata in [6], where the mapping producing the terminal words is defined as $f : \{a\}^\circ \to T^*$ for an infinite alphabet $T = \{a_1, a_2, \ldots\}$ as $f(a^i) = a_i$ for any $i \geq 1$.

Since P finite automata over finite alphabets accept exactly the class of regular languages, the resulting infinite alphabet language class can be considered as the extension of the class of regular languages to infinite alphabets, and this class behaves in several respects differently from infinite alphabet language classes defined using other ideas, such as, for example, the machine model called finite memory automata from [14], or the infinite alphabet regular expressions introduced in [18].

References

[1] A. Alhazov, R. Freund, Yu. Rogozhin. Computational power of symport/antiport: History, advances, and open problems. In *Membrane Computing. 6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005. Revised Selected and Invited Papers*, *Lecture Notes in Computer Science*, 3850, 1-30. Springer-Verlag, 2006.

[2] F. Bernardini, A. Păun. Universality of minimal symport/antiport: Five membranes suffice. In *Membrane Computing, International Workshop, WMC 2003, Tarragona, July 17-22, 2003, Revised Papers*, *Lecture Notes in Computer Science*, 2933, 43-54, Springer-Verlag, 2004.

[3] E. Csuhaj-Varjú, O.H. Ibarra, Gy. Vaszil. On the computational complexity of P automata. *Natural Computing*, 5, 109-126, 2006.

[4] E. Csuhaj-Varjú, M. Margenstern, Gy. Vaszil, S. Verlan. On small universal antiport P systems. *Theoretical Computer Science*, 372, 152-164, 2007.

[5] E. Csuhaj-Varjú, Gy. Vaszil. P automata, or purely communicating accepting P systems. *Membrane Computing. International Workshop WMC-CdeA, Curtea de Arges, Romania, August 19-23, 2002. Revised Papers. Lecture Notes in Computer Science*, 2597, 219-233. Springer-Verlag, 2003.

[6] J. Dassow, Gy. Vaszil. P finite automata and regular languages over countably infinite alphabets. In *Membrane Computing. 7th International Workshop, WMC 2006, Leiden, The Netherlands, July 2006.*

Revised, Selected, and Invited Papers. Lecture Notes in Computer Science, 4361, 352-366. Springer-Verlag 2006.

[7] R. Freund, M. Oswald. P systems with activated/prohibited membrane channels. In [22], 261-268.

[8] R. Freund, A. Păun. Membrane systems with symport/antiport: Universality results. In [22], 270-287.

[9] R. Freund, M. Oswald. A short note on analysing P systems with antiport rules. *Bulletin of the EATCS*, 78, 231-236, 2002.

[10] P. Frisco. About P systems with symport/antiport. In *Second Brainstorming Week on Membrane Computing*, 224-236. University of Seville, TR01, 2004.

[11] P. Frisco, H.J. Hoogeboom. P systems with symport/antiport simulating counter automata. *Acta Informatica*, 41, 145-170, 2004.

[12] P. Frisco, H.J. Hoogeboom. Simulating counter automata by P systems with symport/antiport. In [22], 288-301.

[13] O.H. Ibarra, Gh. Păun. Characterizations of context-sensitive language classes and other language classes in terms of symport/antiport P systems. *Theoretical Computer Science*, 358, 88-103, 2006.

[14] M. Kaminsky, N. Francez. Finite memory automata. *Theoretical Computer Science*, 134, 329-363, 1994.

[15] L. Kari, C. Martín-Vide, A. Păun. On the universality of P systems with minimal symport/antiport rules. In N. Jonoska, Gh. Păun, G. Rozenberg (eds.), *Aspects of Molecular Computing. Essays dedicated to Tom Head on the Occasion of His 70th Birthday*, *Lecture Notes in Computer Science*, 2950, 254-265. Springer-Verlag, 2004.

[16] M. Margenstern, V. Rogozhin, Yu. Rogozhin, S. Verlan. About P systems with minimal symport/antiport rules and four membranes. In *Pre-Proceedings of the Fifth Workshop on Membrane Computing (WMC5)*, 283-294. Universitá di Milano Bicocca, 2004.

[17] C. Martín-Vide, A. Păun, Gh. Păun. On the power of P systems with symport rules. *Journal of Universal Computer Science*, 8, 317-331, 2002.

[18] F. Otto. Classes of regular and context-free languages over countably infinite alphabets. *Discrete Applied Mathematics*, 12, 41-56, 1985.

[19] A. Păun, Gh. Păun. The power of communication: P systems with symport/antiport. *New Generation Computing*, 20, 295-305, 2002.

[20] Gh. Păun. Computing with membranes. *Journal of Computer and Systems Sciences*, 61, 108-143, 2000.

[21] Gh. Păun. *Membrane Computing. An Introduction*. Springer-Verlag, 2002.

[22] Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, (eds.). *Membrane Computing. International Workshop, WMC-CdeA'02, Curtea de Arges, Romania, August 19-23, 2002. Revised Papers*, *Lecture Notes in Computer Science*, 2597. Springer-Verlag, 2003.

[23] G. Rozenberg, A. Salomaa, (eds.). *Handbook of Formal Languages*. Springer-Verlag, 1997.

[24] Gy. Vaszil. On the size of P systems with minimal symport/antiport. *Membrane Computing, 5th International Workshop WMC 2004, Milan, Italy, June 14-16, 2004. Revised, Selected and Invited Papers. Lecture Notes in Computer Science*, 3365, 404-413. Springer-Verlag, 2005.

Theorietag *Automaten und Formale Sprachen*

On the Computational Capacity of Parallel Communicating Finite Automata

Henning Bordihn

Institut für Informatik, Universität Potsdam
August-Bebel-Straße 89, 14482 Potsdam, Germany
henning@cs.uni-potsdam.de

Martin Kutrib and Andreas Malcher

Institut für Informatik, Universität Giessen
Arndtstraße 2, 35392 Giessen, Germany
{kutrib,malcher}@informatik.uni-giessen.de

Abstract

Systems of parallel finite automata communicating by states are investigated. We consider deterministic and nondeterministic devices and distinguish four working modes. It is known that systems in the most general mode are as powerful as one-way multihead finite automata. Here we solve some open problems on the computational capacity of systems working in the remaining modes. In particular, it is shown that deterministic returning and non-returning devices are equivalent, and that there are languages which are accepted by deterministic returning and centralized systems but cannot be accepted by deterministic non-returning centralized systems. Furthermore, we show that nondeterministic centralized systems are strictly more powerful than their deterministic variants. Finally, incomparability with the class of (deterministic) (linear) context-free languages as well as the Church-Rosser languages is derived.

Introduction

The need for a fundamental understanding of parallel processes and cooperating systems is increasing more and more in today's complex world. In the classical theory of formal languages and automata mainly sequential machine models like, for example, finite automata, pushdown automata, or

Turing machines are studied. It turned out that this theory is very helpful to describe, analyze, and understand sequential processes. To obtain such a theory also for cooperating systems, it is an obvious generalization to proceed from one sequential automaton to systems of sequential automata. Some questions immediately arising are, for example, whether the input is processed in a parallel or sequential way and how the input is accepted. One may ask how the cooperation between different automata is organized and whether they work in a synchronous or an asynchronous way. One has to define in which way communication between different automata takes place and how appropriate restrictions on the amount of information communicated can be formulated. In the literature, systems of cooperating sequential automata appear in many facets. Multi-head finite automata [13] are in some sense the simplest model of cooperating automata, since a finite automaton is provided with a fixed number of reading heads. So, we have some model with one finite state control and the cooperation between the finite state control and the single components is the reading of the input and positioning the heads. This model is generalized to multi-head two-way finite automata [7] and multi-head pushdown automata [6]. Multi-processor automata [2] are in a way restricted multi-head finite automata, and the relation between both classes is investigated in [5]. Systems of different finite automata communicating by appropriate protocols are described in [1, 10], and systems of cooperating finite automata working in parallel are introduced in [11]. Apart from systems of cooperating automata there is also the broad field of systems of cooperating grammars [4].

Here, we will focus on parallel communicating finite automata systems which were introduced in [11]. In this model, several finite automata read and process the input in parallel in a synchronized way. The communication between automata is defined in such a way that an automaton can request the current state from another automaton. We distinguish four different working modes. One fundamental result shown in [11] is that nondeterministic (deterministic) non-centralized systems working in the non-returning mode are equally powerful as one-way multi-head nondeterministic (deterministic) finite automata. Recently, it has been shown in [3] that the returning and non-returning working modes coincide for nondeterministic non-centralized systems. The authors left as an open question whether the same is also true for deterministic systems. Moreover, the question whether or not centralized systems are equally powerful as non-centralized systems remained open. Here, the first question and, for deterministic systems working in the non-returning mode, the second question are answered.

Preliminaries and Definitions

A parallel communicating finite automata system of degree k is a device of k finite automata working in parallel with each other, synchronized according to a universal clock, on a common one-way read-only input tape. The k automata communicate on request by states, that is, when some automaton enters a distinguished query state q_i, it is set to the current state of automaton A_i. Concerning the next state of the sender A_i, we distinguish two modes. In *non-returning* mode the sender remains in its current state, whereas in *returning* mode the sender is set to its initial state. Moreover, we distinguish whether all automata are allowed to request communications, or whether there is just one master allowed to request communications. The latter types are called *centralized*.

Formally, a *nondeterministic parallel communicating finite automata system of degree* k (PCFA(k)) is a construct $\mathcal{A} = \langle \Sigma, A_1, A_2, \ldots, A_k, Q, \triangleleft \rangle$, where Σ is the set of *input symbols*, each $A_i = \langle S_i, \Sigma, \delta_i, s_{0,i}, F_i \rangle$, $1 \leq i \leq k$, is a *nondeterministic finite automaton* with state set S_i, initial state $s_{0,i} \in S_i$, set of accepting states $F_i \subseteq S_i$, and transition function $\delta_i : S_i \times (\Sigma \cup \{\lambda, \triangleleft\}) \to 2^{S_i}$, $Q = \{q_1, q_2, \ldots, q_k\} \subseteq \bigcup_{1 \leq i \leq k} S_i$ is the set of *query states*, and $\triangleleft \notin \Sigma$ is the *end-of-input symbol*.

The automata $A_1, A_2, \ldots, A_k$ are called *components* of the system $\mathcal{A}$. A *configuration* $(s_1, x_1, s_2, x_2, \ldots, s_k, x_k)$ of $\mathcal{A}$ represents the current states s_i as well as the still unread parts x_i of the tape inscription of all components $1 \leq i \leq k$. System $\mathcal{A}$ starts with all of its components scanning the first square of the tape in their initial states. For input word $w \in \Sigma^*$, the initial configuration is $(s_{0,1}, w\triangleleft, s_{0,2}, w\triangleleft, \ldots, s_{0,k}, w\triangleleft)$. Basically, a computation of $\mathcal{A}$ is a sequence of configurations beginning with an initial configuration and ending with a halting configuration. Each step can consist of two phases. In a first phase, all components are in non-query states and perform an ordinary (non-communicating) step independently. The second phase is the communication phase during which components in query states receive the requested states as long as the sender is not in a query state itself. This process is repeated until all requests are resolved, if possible. If the requests are cyclic, no successor configuration exists. As mentioned above, we distinguish *non-returning* communication, that is, the sender remains in its current state, and *returning* communication, that is, the sender is reset to its initial state.

For the first phase, we define the successor configuration relation $\vdash$ by

$$(s_1, a_1 y_1, s_2, a_2 y_2, \ldots, s_k, a_k y_k) \vdash (p_1, z_1, p_2, z_2, \ldots, p_k, z_k),$$

if $Q \cap \{s_1, s_2, \ldots, s_k\} = \emptyset$, $a_i \in \Sigma \cup \{\lambda, \lhd\}$, $p_i \in \delta_i(s_i, a_i)$, and $z_i = \lhd$ for $a_i = \lhd$ and $z_i = y_i$ otherwise, $1 \leq i \leq k$. For non-returning communication in the second phase, we set $(s_1, x_1, s_2, x_2, \ldots, s_k, x_k) \vdash (p_1, x_1, p_2, x_2, \ldots, p_k, x_k)$, if, for all $1 \leq i \leq k$ such that $s_i = q_j$ and $s_j \notin Q$, we have $p_i = s_j$, and $p_r = s_r$ for all the other r, $1 \leq r \leq k$. Alternatively, for returning communication in the second phase, we set $(s_1, x_1, s_2, x_2, \ldots, s_k, x_k) \vdash (p_1, x_1, p_2, x_2, \ldots, p_k, x_k)$, if, for all $1 \leq i \leq k$ such that $s_i = q_j$ and $s_j \notin Q$, we have $p_i = s_j$, $p_j = s_{0,j}$, and $p_r = s_r$ for all the other r, $1 \leq r \leq k$.

A computation *halts* when the successor configuration is not defined for the current situation. In particular, this may happen when cyclic communication requests appear, or when the transition function of one component is not defined. The language $L(\mathcal{A})$ accepted by a PCFA(k) $\mathcal{A}$ is precisely the set of words w such that there is some computation beginning with $w\lhd$ on the input tape and halting with at least one component having an undefined transition function and being in an accepting state. Let $\vdash^*$ denote the reflexive and transitive closure of $\vdash$ and set $L(\mathcal{A}) = \{\, w \in \Sigma^* \mid (s_{0,1}, w\lhd, s_{0,2}, w\lhd, \ldots, s_{0,k}, w\lhd) \vdash^* (p_1, a_1y_1, p_2, a_2y_2, \ldots, p_k, a_ky_k)$, such that $p_i \in F_i$ and $\delta_i(p_i, a_i)$ is undefined, for some $1 \leq i \leq k \,\}$.

If all components A_i are deterministic finite automata, that is, for all $s \in S_i$ the transition function $\delta_i(s, a)$ maps to a set of at most one state and is undefined for all $a \in \Sigma$, whenever $\delta_i(s, \lambda)$ is defined, then the whole system is called *deterministic*, and we add the prefix D to denote it. The absence or presence of an R in the type of the system denotes whether it works in *non-returning* or *returning* mode, respectively. Finally, if there is just one component, say A_1, that is allowed to query for states, that is, $S_i \cap Q = \emptyset$, for $2 \leq i \leq k$, then the system is said to be *centralized*. We denote centralized systems by a C. Whenever the degree is missing we mean systems of arbitrary degree. The *family of languages accepted* by devices of type X (with degree k) is denoted by $\mathscr{L}(X)$ ($\mathscr{L}(X(k))$).

For example, the language $\{w\$w \mid w \in \{a, b\}^+\}$ is accepted by a DRCPCFA as well as by a DCPCFA with two components. Thus, all types of systems of parallel communicating finite automata accept more than regular languages.

Deterministic Non-Returning Versus Returning

For nondeterministic non-centralized devices it is shown in [3] that returning parallel communicating finite automata systems are neither weaker nor stronger than non-returning ones. In the same paper the question is raised whether the same equivalence is true in the deterministic case. This section is devoted to answering the question in the affirmative.

Lemma 1 *For all $k \geq 1$, the family $\mathscr{L}(\mathrm{DRPCFA}(k))$ includes the family $\mathscr{L}(\mathrm{DPCFA}(k))$.*

In [11] the equivalence between DPCFA(k) and deterministic one-way k-head finite automata (k-DFA) has been shown. The next theorem concludes the proofs of equivalence.

Theorem 2 *For all $k \geq 1$, the three families $\mathscr{L}(\mathrm{DRPCFA}(k))$ and $\mathscr{L}(\mathrm{DPCFA}(k))$ and $\mathscr{L}(k\text{-DFA})$ are equal.*

Deterministic Non-Centralized Versus Centralized

This section is devoted to comparing deterministic centralized systems with non-centralized systems. We obtain for centralized systems the surprising result that the returning mode is not weaker than the non-returning mode. Let us consider $L_{\mathrm{rc}} = \{\, uc^x v\$uv \mid u, v \in \{a, b\}^*, x \geq 0 \,\}$.

Theorem 3 *The language L_{rc} belongs to the family $\mathscr{L}(\mathrm{DRCPCFA})$ (and thus to $\mathscr{L}(\mathrm{DRPCFA}) = \mathscr{L}(\mathrm{DPCFA})$), but not to $\mathscr{L}(\mathrm{DCPCFA})$.*

Corollary 4 $\mathscr{L}(\mathrm{DCPCFA}) \subset \mathscr{L}(\mathrm{DPCFA}) = \mathscr{L}(\mathrm{DRPCFA})$.

Determinism Versus Nondeterminism

In order to show that nondeterministic centralized systems are strictly more powerful than their deterministic variants we consider the complement of the mirror language, that is, $L_{\mathrm{mi}} = \overline{\{ww^R \mid w \in \{a, b, c\}^+\}}$.

Lemma 5 *The language L_{mi} belongs to $\mathscr{L}(\mathrm{CPCFA})$, but does not belong to $\mathscr{L}(\mathrm{DPCFA})$.*

Corollary 6 $\mathscr{L}(\mathrm{DCPCFA}) \subset \mathscr{L}(\mathrm{CPCFA})$ *and* $\mathscr{L}(\mathrm{DPCFA}) \subset \mathscr{L}(\mathrm{PCFA})$.

Finally, we compare the classes under consideration with some well-known language families.

Lemma 7 *The family $\mathscr{L}(\mathrm{PCFA})$ is strictly included in the complexity class NL, hence, in the family of deterministic context-sensitive languages.*

Lemma 8 *All language classes accepted by parallel communicating finite automata systems are incomparable to the class of (deterministic) (linear) context-free languages.*

Lemma 9 *All language classes accepted by parallel communicating finite automata systems are incomparable with the class of Church-Rosser languages.*

References

[1] Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM **30** (1983) 323–342
[2] Buda, A.: Multiprocessor automata. Inform. Process. Lett. **25** (1987) 257–261
[3] Choudhary, A., Krithivasan, K., Mitrana, V.: Returning and non-returning parallel communicating finite automata are equivalent. RAIRO Inform. Théor. **41** (2007) 137–145
[4] Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, G.: Grammar Systems: A Grammatical Approach to Distribution and Cooperation. Gordon and Breach, Yverdon (1994)
[5] Ďuriš, P., Jurdziński, T., Kutyłowski, M., Loryś, K.: Power of cooperation and multihead finite systems. International Colloquium on Automata, Languages and Programming (ICALP 1998). Volume 1443 of LNCS, Springer (1998) 896–907
[6] Harrison, M.A., Ibarra, O.H.: Multi-tape and multi-head pushdown automata. Inform. Control **13** (1968) 433–470
[7] Ibarra, O.H.: On two-way multihead automata. J. Comput. System Sci. **7** (1973) 28–36
[8] Ibarra, O.H.: A note on semilinear sets and bounded-reversal multihead pushdown automata. Inform. Process. Lett. **3** (1974) 25–28
[9] Jurdziński, T.: The Boolean closure of growing context-sensitive languages. Developments in Language Theory (DLT 2006). Volume 4036 of LNCS, Springer (2006) 248–259
[10] Klemm, R.: Systems of communicating finite state machines as a distributed alternative to finite state machines. Phd thesis, Pennsylvania State University (1996)
[11] Martín-Vide, C., Mateescu, A., Mitrana, V.: Parallel finite automata systems communicating by states. Int. J. Found. Comput. Sci. **13** (2002) 733–749
[12] McNaughton, R., Narendran, P., Otto, F.: Church-Rosser Thue systems and formal languages. J. ACM **35** (1988) 324–344
[13] Rosenberg, A.L.: On multi-head finite automata. IBM J. Res. Dev. **10** (1966) 388–394
[14] Wagner, K., Wechsung, G.: Computational Complexity. Reidel, Dordrecht (1986)
[15] Yao, A.C., Rivest, R.L.: $k+1$ heads are better than k. J. ACM **25** (1978) 337–340

Blinde Zählerautomaten auf ω-Wörtern

Henning Fernau

Fachbereich IV, Abteilung Informatik, Universität Trier
54286 Trier, Germany
fernau@uni-trier.de

Ralf Stiebe

Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg
PF-4120, 339016 Magdeburg, Germany
stiebe@iws.cs.uni-magdeburg.de

1 Einführung und Definitionen

Während partiell blinde Zählerautomaten auf ω-Wörtern eingehend in der Literatur studiert wurden [1, 2, 8], ist dies nicht der Fall für blinde Zählerautomaten. Wir schließen diese Lücke (teilweise) mit dieser Arbeit, siehe [7], wo auch weitere Einzelheiten nachzulesen sind.

Blinde Zählerautomaten wurden ursprünglich (für endliche Wörter) von Greibach [9] eingeführt. Wesentlich ist, dass die Zähler nur am Schluss getestet werden können: Ein Wort wird nur akzeptiert, wenn der Automat in einem Endzustand angekommen ist und alle Zähler auf Null stehen. Stehen k Zähler zur Verfügung, so führt dies auf die Sprachklasse $\mathcal{B}_k$.

Es gibt unterschiedliche Möglichkeiten, die Wirkungsweise von blinden Zählerautomaten auf ω-Wörtern festzulegen. Wir betrachten drei Möglichkeiten in dieser Arbeit:

- $\mathcal{K}_k$ bezeichne die ω-Kleene-Hülle von $\mathcal{B}_k$. Das bedeutet, $L \subseteq \Sigma^\omega$ liegt in $\mathcal{K}_k$, wenn wir blinde k-Zählersprachen V_i und W_i finden, sodass $L = \bigcup_{i=1}^{n} V_i \cdot W_i^\omega$.

- $L^\omega(\mathfrak{M})$: Akzeptiere alle ω-Wörter, welche den Zählerautomaten $\mathfrak{M}$ unendlich oft durch einen ausgewählten Endzustand führen, während gleichzeitig alle Zähler Null enthalten. Dies führt zur Sprachfamilie $\mathcal{B}_k^\omega$.

- $L^{\omega,a}(\mathfrak{M})$: Akzeptiere alle ω-Wörter, welche den Zählerautomaten $\mathfrak{M}$ unendlich oft durch einen ausgewählten Endzustand führen und die ebenso dazu führen, dass alle Zähler unendlich of Null enthalten. Dies führt zur Sprachfamilie $\mathcal{B}_k^{\omega,a}$.

Die beiden letzteren Definitionen erweitern die klassische Büchi-Akzeptanz endlicher Automaten, im ersten Fall in synchroner, und im zweiten Fall in asynchroner Art und Weise, was die Handhabung der Zähler betrifft. Wir verwenden den Index $*$ in unserer Sprachklassenschreibweise, um beliebige Zähleranzahlen zuzulassen. Betrachten wir abschließend ein Beispiel:

Für $k \geq 1$ sei $X_k = \{a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_k\}$. Betrachte

$$\mathfrak{M} = (\{q_0\}, X_k, \{(q_0, a_i, q_0, \vec{e_i}), (q_0, b_i, q_0, -\vec{e_i}) \,:\, 1 \leq i \leq k\}, q_0, \{q_0\}).$$

Die akzeptierten (ω-) Sprachen sind:

$$\begin{aligned} L(\mathfrak{M}) &= \{w \in X_k^* \,:\, |w|_{a_i} = |w|_{b_i}, \text{ für } 1 \leq i \leq k\}, \\ L^{\omega}(\mathfrak{M}) &= \{w \in X_k^{\omega} \,:\, |\{p \sqsubset w \,:\, |p|_{a_i} = |p|_{b_i}, \text{ für } 1 \leq i \leq k\}| = \infty\}, \\ L^{\omega,a}(\mathfrak{M}) &= \{w \in X_k^{\omega} \,:\, |\{p \sqsubset w \,:\, |p|_{a_i} = |p|_{b_i}\}| = \infty, \text{ für alle } 1 \leq i \leq k\}. \end{aligned}$$

2 Ergebnisse

Lemma 1 $\mathcal{B}_0^{\omega} = \mathcal{B}_0^{\omega,a} = \mathcal{K}_0$.

Lemma 2 $\forall k \geq 0 (\mathcal{B}_k^{\omega} \subseteq \mathcal{K}_k)$.

Lemma 3 $L_1 = \{a^n b^n \mid n \geq 1\}^{\omega} \notin \mathcal{B}_*^{\omega}$.

Satz 4 $\forall k \geq 0 (\mathcal{B}_k^{\omega} \subseteq \mathcal{K}_k)$. *Gleichheit gilt genau dann, wenn* $k = 0$.

Satz 5 *Jede nicht-leere* ω*-Sprache aus* $\mathcal{K}_*$ *enthält ein* ω*-Wort der Form* st^{ω}*, wobei* s *und* t *endliche Wörter sind.*

Satz 6 *Für einen blinden Zählerautomaten* $\mathfrak{M}$ *ist entscheidbar, ob* $L^{\omega}(\mathfrak{M}) = \emptyset$ *oder nicht.*

Lemma 7 $\mathcal{B}_1^{\omega,a} \subseteq \mathcal{B}_1^{\omega}$.

Lemma 8 $\mathcal{B}_1^{\omega} \subseteq \mathcal{B}_1^{\omega,a}$.

Lemma 9 *Es gibt eine nicht-leere Sprache* $L \in \mathcal{B}_2^{\omega,a}$*, sodass* L *kein* ω*-Wort der Form* st^{ω} *enthält.*

Zum Beweis betrachte:

$$L = \{w \in \{a,b\}^\omega \quad : \quad |\{p : p \sqsubset w \wedge |p|_a = |p|_b\}| = \infty \wedge \\ |\{p : p \sqsubset w \wedge |p|_a = 2|p|_b\}| = \infty\}.$$

Satz 10 *$\mathcal{B}_2^{\omega,a} \not\subseteq \mathcal{K}_*$ (und folglich $\mathcal{B}_2^{\omega,a} \not\subseteq \mathcal{B}_*^\omega$).*

Satz 11 *$\mathcal{B}_2^\omega \not\subseteq \mathcal{B}_*^{\omega,a}$.*

Zum Beweis betrachte

$$L = \{w \in \{a_1,b_1,a_2,b_2\}^\omega : |\{p : p \sqsubset w \wedge |p|_{a_1} = |p|_{b_1} \wedge |p|_{a_2} = |p|_{b_2}\}| = \infty\}.$$

Definiere $\mathit{Infix}(L) = \{v : \exists u \exists w : uvw \in L\}$.

Lemma 12 *Für $k \geq 1$ sei $X_k = \{a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_k\}$ und*

$$\begin{aligned} L_k &= \{w \in X_k^* : |w|_{a_i} = |w|_{b_i}, \text{ für } 1 \leq i \leq k\}; \\ L_k' &= \{a_1^n \cdots a_k^n : n \geq 0\} \cup \{b_1^n \cdots b_k^n : n \geq 0\}. \end{aligned}$$

Ist L eine blinde $(k-1)$-Zählersprache, sodass $\mathit{Infix}(L)$ unendlich viele Wörter von L_k' enthält, so gilt $L \setminus L_k \neq \emptyset$.

Satz 13 *Für $k \geq 1$ ist $\mathcal{B}_k^\omega \setminus \mathcal{K}_{k-1} \neq \emptyset$. Daraus folgt:*
$\mathcal{B}_{k-1}^\omega \subset \mathcal{B}_k^\omega$ und $\mathcal{K}_{k-1} \subset \mathcal{K}_k$.

Satz 14 *Für $k \geq 1$ gilt: $\mathcal{B}_{k-1}^{\omega,a} \subset \mathcal{B}_k^{\omega,a}$.*

Satz 15 *Die Sprachfamilien $\mathcal{B}_k^\omega, \mathcal{B}_k^{\omega,a}, \mathcal{K}_k$, $1 \leq k$, sowie $\mathcal{B}_*^\omega, \mathcal{B}_*^{\omega,a}, \mathcal{K}_*$ sind abgeschlossen gegen Vereinigungsbildung.*

Satz 16 *Die Sprachfamilien $\mathcal{B}_k^\omega, \mathcal{B}_k^{\omega,a}$, $1 \leq k$, sowie $\mathcal{B}_*^\omega, \mathcal{B}_*^{\omega,a}$ sind abgeschlossen gegenüber Schnitt mit regulären ω-Sprachen.*

Satz 17 *Keine der Sprachfamilien $\mathcal{B}_k^\omega$, $\mathcal{K}_k$, $1 \leq k$, $\mathcal{B}_*^\omega$, $\mathcal{K}_*$ ist abgeschlossen gegenüber Durchschnittsbildung.*

Satz 18 *Die Sprachfamilie $\mathcal{B}_*^{\omega,a}$ ist abgeschlossen gegenüber Durchschnittsbildung.*

Satz 19 *Keine der Sprachfamilien $\mathcal{B}_i^\omega, \mathcal{B}_i^{\omega,a}, \mathcal{K}_i$, $1 \leq i$, und ebensowenig $\mathcal{B}_*^\omega, \mathcal{B}_*^{\omega,a}, \mathcal{K}_*$, ist abgeschlossen gegenüber Komplementbildung.*

Offene Fragen betreffen u.a. andere Akzeptanzmodi (außer Büchi) sowie topologische Untersuchungen. Weiter scheinen andere Kennzeichnungen der Sprachklassen möglich zu sein, z.B. solche grammatikalischer Natur (für den Fall endlicher Wörter, vergleiche [6, 5, 10]), oder auch algebraischer Natur, wie für den Fall endlicher Wörter jüngste Arbeiten von Kambites und seinen Doktoranden zeigen, wie z.B. auf der diesjährigen LATA präsentiert. Für Anwendungen im Bereich der fraktalen Geometrie scheint insbesondere die Sprachklasse $\mathcal{K}_k$ von Interesse zu sein, siehe [3, 4], vornehmlich für endliche Automaten.

Literatur

[1] J. Duparc, O. Finkel, and J.-P. Ressayre. Computer science and the fine structure of Borel sets. *Theoretical Computer Science*, 257:85–105, 2001.

[2] J. Engelfriet and H. J. Hoogeboom. X-automata on ω-words. *Theoretical Computer Science*, 110:1–51, 1993.

[3] H. Fernau and L. Staiger. Valuations and unambiguity of languages, with applications to fractal geometry. In S. Abiteboul and E. Shamir, editors, *Automata, Languages and Programming, 21st International Colloquium, ICALP 94*, volume 820 of *LNCS*, pages 11–22, July 1994. ISBN: 3-540-58201-0.

[4] H. Fernau and L. Staiger. Iterated function systems and control languages. *Information and Computation*, 168:125–143, 2001.

[5] H. Fernau and R. Stiebe. Sequential grammars and automata with valences. *Theoretical Computer Science*, 276:377–405, 2002.

[6] H. Fernau and R. Stiebe. Valuated and valence grammars: an algebraic view. In W. Kuich, G. Rozenberg, and A. Salomaa, editors, *Proceedings DLT'01*, volume 2295 of *LNCS*, pages 281–292. Springer, 2002.

[7] H. Fernau and R. Stiebe. Blind counter automata on ω-words. *Fundamenta Informaticae*, 83:51–64, 2008.

[8] O. Finkel. An effective extension of the Wagner hierarchy to blind counter automata. In L. Fribourg, editor, *Computer Science Logic CSL*, volume 2142 of *LNCS*, pages 369–383. Springer, 2001.

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

[10] Gh. Păun. A new generative device: valence grammars. *Rev. Roumaine Math. Pures Appl.*, XXV(6):911–924, 1980.

(Tissue) P Systems Working in the k-Restricted Minimally Parallel Mode

Rudolf Freund

Faculty of Informatics, Vienna University of Technology
Favoritenstraße 9, 1040 Vienna, Austria
rudi@emcc.at

Sergey Verlan

ILACL, Département Informatique
UUFR Sciences et Technologie, Université Paris XII
A61, av. Général de Gaulle, 94010 Créteil, France
verlan@univ-paris12.fr

Abstract

We consider a special variant of the minimally parallel mode – we allow only a bounded number of rules to be taken from every set of the partitioning of the whole set of rules. The 1-restricted minimally parallel mode allows us to describe the way transitions take place in spiking neural P systems without delays, i.e., in every neuron where a rule is applicable exactly one rule has to be applied. Moreover, purely catalytic P systems working in the maximally parallel mode can be described as P systems using the corresponding rules without catalysts when working in the 1-restricted minimally parallel mode.

1 Networks of Cells

For elements of formal language theory, we refer to [7]. For an introduction to the area of membrane computing we refer the interested reader to the monograph [6], the actual state of the art can be seen in the web [8]. We now consider membrane systems as a collection of interacting cells containing multisets of objects like in [2] and [5].

Definition 1.1 *A* network of cells with checking sets of degree $n \geq 1$ *is a construct* $\Pi = (n, V, w, R)$ *where*

1. *n is the number of cells;*
2. *V a* finite alphabet*;*
3. *$w = (w_1, \ldots, w_n)$ where $w_i \in \langle V, \mathbb{N}_\infty \rangle$, for all $1 \leq i \leq n$, is the* multiset initially associated to cell *i (in most of the cases, at most one cell, then being called* the environment*, will contain symbols occurring with infinite multiplicity);*
4. *R is a finite set of* interaction rules *of the form $(E : X \to Y)$ where E is a recursive condition for configurations of Π (see definition below) as well as $X = (x_1, \ldots, x_n)$, $Y = (y_1, \ldots, y_n)$, with $x_i, y_i \in \langle V, \mathbb{N} \rangle$, $1 \leq i \leq n$, are vectors of multisets over V. We will also use the notation $(E : (x_1, 1) \ldots (x_n, n) \to (y_1, 1) \ldots (y_n, n))$ for a rule $(E : X \to Y)$.*

A network of cells consists of n cells, numbered from 1 to n, that contain (possibly infinite) multisets of objects over V; initially cell i contains w_i. A *configuration* C of Π is an n-tuple of multisets over V $(u_1, \ldots, u_n)$; the *initial configuration* of Π, C_0, is described by w, i.e., $C_0 = w = (w_1, \ldots, w_n)$. Cells can interact with each other by means of the rules in R. An interaction rule $(E : (x_1, 1) \ldots (x_n, n) \to (y_1, 1) \ldots (y_n, n))$ is applicable to a configuration C if and only if C fulfills condition E; its application means rewriting objects x_i from cells i into objects y_j in cells j, $1 \leq i, j \leq n$.

The set of all multisets of rules *applicable* to C is denoted by $Appl\,(\Pi, C)$ (a procedural algorithm how to obtain $Appl\,(\Pi, C)$ is described in [5]).

For the specific *transition modes* to be defined in the following, the selection of multisets of rules applicable to a configuration C has to be a specific subset of $Appl\,(\Pi, C)$; for the transition mode ϑ, the selection of multisets of rules applicable to a configuration C is denoted by $Appl\,(\Pi, C, \vartheta)$. In the *asynchronous* mode ($asyn$), there are no particular restrictions on the multisets of rules applicable to C, i.e., $Appl\,(\Pi, C, asyn) = Appl\,(\Pi, C)$. In the *sequential* mode ($sequ$), any multiset of rules $R' \in Appl\,(\Pi, C, sequ)$ has size 1, i.e.,

$$Appl\,(\Pi, C, sequ) = \left\{ R' \mid R' \in Appl\,(\Pi, C) \text{ and } \left|R'\right| = 1 \right\}.$$

In the *maximally parallel* mode (max) we only select multisets of rules R' that are not extensible, i.e., there is no other multiset of rules $R'' \supsetneqq R'$ applicable to C:

$$\begin{aligned} Appl\,(\Pi, C, max) \;=\; & \{ R' \mid R' \in Appl\,(\Pi, C) \text{ and there is} \\ & \text{no } R'' \in Appl\,(\Pi, C) \text{ with } R'' \supsetneqq R' \}. \end{aligned}$$

For the *minimally parallel* mode, we need an additional feature for the set of rules R, i.e., we consider a partition of R into disjoint subsets R_1 to R_h. Usually, this partition of R may coincide with a specific assignment of the rules to the cells. There are several possible interpretations of this minimally parallel mode which in an informal way can be described as applying multisets such that from every set R_j, $1 \leq j \leq h$, at least one rule – if possible – has to be used (e.g., see [3]):

$$\begin{aligned} Appl\,(\Pi, C, min) \;=\; & \{R' \mid R' \in Appl\,(\Pi, C, asyn) \text{ and} \\ & \text{there is no } R'' \in Appl\,(\Pi, C, asyn) \\ & \text{with } R'' \supsetneqq R',\ (R'' - R') \cap R_j \neq \emptyset \\ & \text{and } R' \cap R_j = \emptyset \text{ for some } j,\ 1 \leq j \leq h\}\,. \end{aligned}$$

In [5], further restricting conditions on the four basic modes defined above, especially interesting for the minimally parallel mode, were considered. We now consider a restricted variant of the minimally parallel mode allowing only a bounded number of at most k rules to be taken from each set R_j, $1 \leq j \leq h$, of the partitioning into a multiset of rules applicable in the minimally parallel mode: For the *k-restricted minimally parallel* mode (min_k), we define

$$\begin{aligned} Appl\,(\Pi, C, min_k) \;=\; & \{R' \mid R' \in Appl\,(\Pi, C, min) \text{ and} \\ & |R' \cap R_j| \leq k \text{ for all } j,\ 1 \leq j \leq h\}\,. \end{aligned}$$

For all the modes defined above, we now define how to obtain a next configuration from a given one by applying an applicable multiset of rules according to the constraints of the underlying mode: Given a configuration C of Π and a mode ϑ, we may choose a multiset of rules $R' \in Appl\,(\Pi, C, \vartheta)$ in a non-deterministic way and apply it to C. A *computation* in a network of cells Π, $\Pi = (n, V, w, R)$, starts with the initial configuration $C_0 = w$ and continues with transition steps according to the chosen mode ϑ; it is called *successful* if we reach a configuration C to which no multiset of rules can be applied with respect to the mode ϑ anymore (we also say that the computation *halts*). As the *results* of a halting computation we take the number of objects in a specified output cell. We shall use the notation

$$NO_m C_n\,(\vartheta)\,[\text{parameters for rules}]$$

to denote the family of sets of natural numbers generated by networks of cells $\Pi = (n, V, w, R)$ with $m = |V|$ as well as ϑ indicating the mode; the *parameters for rules* describe the specific features of the rules in R. If any of the parameters m and n is unbounded, we replace it by $*$.

2 Specific Examples for the 1-Restricted Minimally Parallel Mode

In this section, we show how the 1-restricted minimally parallel mode may capture characteristic features of well-known models of P systems.

2.1 Extended Spiking Neural P Systems

An *extended spiking neural P system* (of degree $m \geq 1$) (*ESNP system*) is a construct $\Pi = (m, S, R)$ where

- m is the number of *neurons*; the neurons are uniquely identified by a number between 1 and m;
- S describes the *initial configuration* by assigning an initial value (of spikes) to each neuron;
- R is a finite set of *rules* of the form $\left(i, E/a^k \rightarrow P\right)$ such that $i \in [1..m]$ (specifying that this rule is assigned to neuron i), E is a regular *checking set* (the current number of spikes in the neuron has to be from E if this rule shall be executed), $k \in \mathbb{N}$ is the "number of spikes" (the energy) consumed by this rule, and P is a (possibly empty) set of *productions* of the form (l, a^w) where $l \in [1..m]$ (thus specifying the target neuron), $w \in \mathbb{N}$ is the *weight* of the energy sent along the axon from neuron i to neuron l.

A *configuration* of the ESNP system is described by specifying the actual number of spikes in every neuron. A *transition* from one configuration to another one is executed as follows: for each neuron i, we non-deterministically choose a rule $\left(i, E/a^k \rightarrow P\right)$ that can be applied, i.e., if the current value of spikes in neuron i is in E, neuron i "spikes", i.e., for every production (l, w) occurring in the set P we send w spikes along the axon from neuron i to neuron l. A *computation* is a sequence of configurations starting with the initial configuration given by S. A computation is called *successful* if it halts, i.e., if for no neuron, a rule can be activated; we then consider the contents, i.e., the number of spikes, of a specific neuron called *output neuron* in halting computations.

We now consider the ESNP system $\Pi = (m, S, R)$ as a network of cells $\Pi' = (m, \{a\}, S, R')$ working in the 1-restricted minimally parallel mode,

with

$$\begin{aligned} R' &= \big\{\big(E : \big(a^k, i\big) \to \big(a^{w_1}, l_1\big) \ldots \big(a^{w_n}, l_n\big)\big) \mid \\ &\quad \big(i, E/a^k \to \big(l_1, a^{w_1}\big) \ldots \big(l_n, a^{w_n}\big)\big) \in R\big\} \end{aligned}$$

and the partitioning R'_i, $1 \leq i \leq m$, of the rule set R' according to the set of neurons, i.e.,

$$\begin{aligned} R'_i &= \big\{\big(E : \big(a^k, i\big) \to \big(a^{w_1}, l_1\big) \ldots \big(a^{w_n}, l_n\big)\big) \mid \\ &\quad \big(E : \big(a^k, i\big) \to \big(a^{w_1}, l_1\big) \ldots \big(a^{w_n}, l_n\big)\big) \in R'\big\} . \end{aligned}$$

The 1-restricted minimally parallel mode chooses one rule – if possible – from every set R_i and then applies such a multiset of rules in parallel, which directly corresponds to applying one spiking rule in every neuron where a rule can be applied. Hence, it is easy to see that Π' and Π generate the same set if in both systems we take the same cell/neuron for extracting the output. Due to the results valid for ESNP systems, see [1], we obtain

$$NRE = NO_1C_3\,(min_1)\,[\text{ESNP}]\,.$$

2.2 Purely Catalytic P Systems

A catalytic rule is of the form $(I : (c, i)\,(a, i) \to (c, i)\,(y_1, 1) \ldots (y_n, n))$ where I denotes the condition that is always fulfilled, c is from a distinguished subset $V_C \subset V$ (catalysts), $a \in (V - V_C)$, and the y_i are from $(V - V_C)^*$. Imposing the restriction that the catalytic rules in a network of cells allow for finding a hierarchical tree structure of membranes such that symbols either stay in their membrane region or are sent out to the surrounding membrane region or sent into an inner membrane, then we get the classical *purely catalytic P systems*. As we know from [4], only three catalysts in one membrane are needed to obtain NRE with purely catalytic P systems working in the maximally parallel mode:

$$NRE = NO_*C_1\,(max)\,[\text{cat}_2] = NO_*C_1\,(max)\,[\text{pcat}_3]\,.$$

If we now partition the rule set in a purely catalytic P system according to the catalysts present in each membrane, this partitioning replaces the use of the catalysts when working in the 1-restricted minimally parallel mode, because by definition from each of these sets then – if possible – exactly one rule (as with the use of the corresponding catalyst) is chosen: from the set of purely catalytic rules R we obtain the corresponding set of noncooperative rules R' as

$$\begin{aligned} R' &= \{(I : (a, i) \to (y_1, 1) \ldots (y_n, n)) \mid \\ &\quad (I : (c, i)\,(a, i) \to (c, i)\,(y_1, 1) \ldots (y_n, n)) \in R\} \end{aligned}$$

as well as the corresponding partitioning of R' as

$$\begin{aligned} R'_{i,c} &= \{(I : (a,i) \to (y_1,1)\dots(y_n,n)) \mid \\ &\quad (I : (c,i)\,(a,i) \to (c,i)\,(y_1,1)\dots(y_n,n)) \in R\}\,. \end{aligned}$$

Considering purely catalytic P systems in one membrane, we therefore infer the following quite astonishing result that when using the 1-restricted minimally parallel mode for a suitable partitioning of rules we only need noncooperative rules:

$$NRE = NO_*C_1\,(min_1)\,[\text{noncoop}]\,.$$

References

[1] A. Alhazov, R. Freund, M. Oswald, M. Slavkovik: Extended spiking neural P systems generating strings and vectors of non-negative integers. In: H.J. Hoogeboom, Gh. Paun, and G. Rozenberg, (Eds.), *Pre-proceedings of Membrane Computing, International Workshop, WMC7*, Leiden, The Netherlands, 2006, 88–101.

[2] F. Bernardini, M. Gheorghe, M. Margenstern, S. Verlan: Networks of cells and Petri nets. In: M. A. Gutiérrez-Naranjo, Gh. Păun, A. Romero-Jiménez, A. Riscos-Núñez: *Proc. Fifth Brainstorming Week on Membrane Computing*, Sevilla, 2007, 33–62.

[3] G. Ciobanu, L. Pan, Gh. Păun, M.J. Pérez-Jiménez: P systems with minimal parallelism, *Theoretical Computer Science* **378** (1) (2007), 117–130.

[4] R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science* **330** (2005), 251–266.

[5] R. Freund, S. Verlan: A formal framework for P systems. In: G. Eleftherakis, P. Kefalas, Gh. Paun (Eds.), Pre-proceedings of Membrane Computing, International Workshop – WMC8, Thessaloniki, Greece, 2007, 317–330.

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

[7] G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages* (3 volumes), Springer-Verlag, Berlin, 1997.

[8] The P Systems Web Page: `http://ppage.psystems.eu`.

Inklusionsprobleme für Patternsprachen*

Dominik D. Freydenberger

Institut für Informatik, Goethe-Universität
60054 Frankfurt a.M., Germany
freydenberger@em.uni-frankfurt.de

Daniel Reidenbach

Dept. of Computer Science, Loughborough University
ALE11 3TU, England
D.Reidenbach@lboro.ac.uk

Zusammenfassung

Wir betrachten das Inklusionsproblem für Patternsprachen, das laut Jiang et al. (J. Comput. System Sci. 50, 1995) unentscheidbar ist. Genau genommen zeigen Jiang et al., daß keine rekursive Funktion existiert, die die Inklusion für die Klasse *aller* Patternsprachen über *allen* Alphabeten entscheidet. Die meisten Anwendungen betrachten aber Klassen von Patternsprachen über *festen* Alphabeten, so daß die Frage nach alphabetabhängigen Entscheidungsfunktionen von größerer praktischer Relevanz ist. Unser erstes Hauptresultat besagt, daß (außer in einigen sehr speziellen Ausnahmefällen) auch diese Version des Inklusionsproblems unentscheidbar ist. In der zweiten Hälfte dieser Arbeit widerlegen wir eine verbreitete Vermutung zur Inklusion *ähnlicher* E-Patternsprachen, mit verheerenden Konsequenzen für die bisher geleistete Forschung zum Äquivalenzproblem für E-Patternsprachen.

1 Einleitung

Pattern, d. h. endliche Wörter aus Variablen und Terminalsymbolen, stellen eine kompakte, elegante und natürliche Methode dar, gewisse kontextsensitive Sprachen zu repräsentieren. Ein Pattern erzeugt ein Wort durch eine Substitution, die alle Variablen im Pattern durch beliebige endliche Wörter

*Eine ausführlichere Darstellung dieser Arbeit findet sich in [3].

über einem festen Terminalalphabet ersetzt. Die *Patternsprache* eines Pattern ist somit die Menge aller Wörter, die durch die Substitution des Pattern gebildet werden können; etwas formaler ist eine Patternsprache also die Menge aller Bilder des Pattern unter beliebigen terminalerhaltenden Homomorphismen. Wenn wir beispielsweise das Pattern $\alpha := x_1\,\mathtt{a}\,x_2\,\mathtt{b}\,x_1$ (mit Variablen x_1, x_2 und Terminalsymbolen $\mathtt{a}, \mathtt{b}$) betrachten, so liegen folglich u. a. $w_1 := \mathtt{aabbba}$, $w_2 := \mathtt{abababab}$ und $w_3 := \mathtt{aaabaa}$ in der von α erzeugten Patternsprache, wohingegen die Beispielwörter $w_4 := \mathtt{ba}$, $w_5 := \mathtt{babbba}$ und $w_6 := \mathtt{abba}$ nicht von α erzeugt werden können.

In der Literatur werden grundsätzlich zwei verschiedene Arten von Patternsprachen betrachtet: NE-Patternsprachen (eingeführt von Angluin [1]), bei denen die Variablen stets mit nichtleeren Wörtern substituiert werden müssen, und E-Patternsprachen (erstmals untersucht von Shinohara [11]), bei denen auch die Substitution mit leeren Wörtern erlaubt ist. Im obigen Beispiel ist somit das Wort w_3 zwar in der E-Patternsprache, nicht jedoch in der NE-Patternsprache von α enthalten.

Patternsprachen weisen aufgrund ihrer einfachen, nur auf endlichen Wörtern und speziellen, aber dennoch sehr üblichen Homomorphismen beruhenden Definition vielfältige Verbindungen zu anderen Konzepten in der Welt der formalen Sprachen auf (s. Mateescu und Salomaa [7]). Darüber hinaus spielen sie eine zentrale Rolle bei Untersuchungen im Rahmen der Induktiven Inferenz, einem Modell der algorithmischen Lernthorie, zur Erlernbarkeit eines gemeinsamen Musters in einer Menge von Wörtern. Aufgrund der bahnbrechenden Erkenntnisse von Angluin [2] ist für das letztgenannte Themengebiet insbesondere das Inklusionsproblem für Patternsprachen von zentraler Bedeutung. Die vorliegende Arbeit untersucht daher dieses Entscheidungsproblem eingehender; indirekt befassen wir uns überdies mit dem Äquivalenzproblem für E-Patternsprachen.

2 Definitionen

Wir bezeichnen das leere Wort als λ. Sei Σ ein Alphabet sogenannter *Terminalsymbole*, und X eine unendliche und zu Σ disjunkte Menge von *Variablen*. Ein Homomorphismus $\sigma : (\Sigma \cup X)^* \to \Sigma^*$ ist eine *Substitution*, wenn $\sigma(a) = a$ für alle $a \in \Sigma$, und *nichtlöschend*, wenn $\sigma(x) \neq \lambda$ für alle $x \in \Sigma \cup X$. Wir bezeichnen Wörter aus $(\Sigma \cup X)^*$ als *Pattern* und die Menge aller Pattern über Σ als Pat_Σ. Ein Pattern in X^* heißt *terminalfrei*. Ein Pattern $\alpha \in \mathrm{Pat}_\Sigma$ erzeugt die *E-Patternsprache* $L_{\mathrm{E},\Sigma}(\alpha) := \{\sigma(\alpha) \mid \sigma : (\Sigma \cup X)^* \to \Sigma^* \text{ ist eine Substitution}\}$ und die *NE-Patternsprache* $L_{\mathrm{NE},\Sigma}(\alpha) := \{\sigma(\alpha) \mid$

$\sigma : (\Sigma \cup X)^* \to \Sigma^*$ ist eine nichtlöschende Substitution}. Die Klasse aller E-Patternsprachen (bzw. NE-Patternsprachen) über Σ wird mit ePAT_Σ (bzw. nePAT_Σ) bezeichnet. Eine Patternsprache heißt *terminalfrei*, wenn sie von einem terminalfreien Pattern erzeugt wird. Zwei Pattern $\alpha, \beta \in \mathrm{Pat}_\Sigma$ sind *ähnlich*, falls $\alpha = \alpha_0 u_1 \alpha_1 u_1 \dots \alpha_{n-1} u_n \alpha_n$ und $\beta = \beta_0 u_1 \beta_1 u_2 \dots \beta_{n-1} u_n \beta_n$ für ein $n \geq 0$ mit $\alpha_i, \beta_i \in X^+$ für alle $i \in \{1, \dots, n-1\}$, $\alpha_0, \beta_0, \alpha_n, \beta_n \in X^*$ und $u_j \in \Sigma^+$ für alle $j \in \{1, \dots, n\}$. Zwei Patternsprachen heißen *ähnlich*, wenn sie von ähnlichen Pattern erzeugt werden.

3 Das Inklusionsproblem für Patternsprachen

Laut Jiang, Salomaa, Salomaa und Yu [6] ist das *allgemeine (alphabetunabhängige) Inklusionsproblem für Patternsprachen* unentscheidbar:

Satz 1 (Jiang et al. [6]) *Sei* $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$. *Es gibt keine totale berechenbare Funktion* χ_Z, *die für jedes Alphabet* Σ *und jedes Paar von Pattern* $\alpha, \beta \in \mathrm{Pat}_\Sigma$ *entscheidet, ob* $L_{\mathrm{Z},\Sigma}(\alpha) \subseteq L_{\mathrm{Z},\Sigma}(\beta)$.

Jiang et al. zeigen dies zuerst für $\mathrm{Z} = \mathrm{E}$ durch eine aufwendige Reduktion des unentscheidbaren Leerheitsproblems für 2-Zähler-Automaten (2ZA, s. Ibarra [4]) auf das allgemeine Inklusionsproblem für E-Patternsprachen (der Fall $\mathrm{Z} = \mathrm{NE}$ läßt sich mit großem Aufwand auf $\mathrm{Z} = \mathrm{E}$ reduzieren). Aus einem gegebenen 2ZA $\mathcal{A}$ konstruieren Jiang et al. ein Alphabet Σ und Pattern $\alpha_\mathcal{A}, \beta_\mathcal{A} \in \mathrm{Pat}_\Sigma$, so daß $L_{\mathrm{E},\Sigma}(\alpha_\mathcal{A}) \subseteq L_{\mathrm{E},\Sigma}(\beta_\mathcal{A})$ genau dann gilt, wenn $\mathcal{A}$ auf keiner Eingabe terminiert. Hierbei enthält Σ einen Buchstaben für jeden Zustand von $\mathcal{A}$, sowie sechs zusätzliche Zeichen, die aus technischen Gründen benötigt werden. Da zu einer festen Zahl von Zuständen nur jeweils endlich viele 2ZA existieren, ist es essentiell für diesen Beweis, eine unendliche Anzahl von Alphabeten Σ zu betrachten. Allerdings werden in der Induktiven Inferenz stets Klassen von Patternsprachen über *festen* Alphabeten betrachtet, wobei unterschiedliche Alphabetgrößen zu großen Unterschieden in der Lernbarkeit führen können (s. Reidenbach [10]).

Es liegt also nahe, das Inklusionsproblem für Klassen von Patternsprachen über festen Alphabeten zu untersuchen. Unser erstes Hauptergebnis besagt, daß dieses Problem für E-Patternsprachen bei allen "nichttrivialen" Alphabetgrößen unentscheidbar ist:

Satz 2 *Sei* Σ *ein endliches Alphabet mit wenigstens zwei Buchstaben. Dann ist das Inklusionsproblem für* ePAT_Σ *unentscheidbar.*

Der Beweis für das wesentlich stärkere Unentscheidbarkeitsresultat in Satz 2 folgt zwar der Grundidee des Beweises von Satz 1, aber er verwendet eine

unären Codierung der Zustände der 2ZA und verzichtet auf zusätzliche Symbole. Diese Änderungen erzwingen eine deutlich andere Struktur der im Beweis konstruierten Pattern und bringen erhebliche technische Schwierigkeiten mit sich. Details müssen aus Platzgründen entfallen.

Auch auf der Grundlage von Satz 2 läßt sich für den NE-Fall die bei Satz 1 erwähnte Reduktion anwenden. Allerdings sind hierzu zwei zusätzliche Buchstaben notwendig, weswegen das Inklusionsproblem für NE-Patternsprachen bei zwei- und bei dreielementigen Alphabeten offen bleibt:

Satz 3 *Sei Σ ein endliches Alphabet mit wenigstens vier Buchstaben. Dann ist das Inklusionsproblem für* nePAT_Σ *unentscheidbar.*

4 Die Inklusion von ähnlichen E-Patternsprachen

Die in Abschnitt 3 zum Beweis der Unentscheidbarkeit des Inklusionsproblems für E-Patternsprachen verwendeten Pattern sind nicht ähnlich (siehe die Definition in Abschnitt 2). Es stellt sich also die Frage, ob sich für die natürlichen Teiklassen aller zueinander ähnlichen E-Patternsprachen jeweils die Inklusion entscheiden läßt. Für die einfachste dieser Klassen, nämlich die der terminalfreien E-Patternsprachen, kann diese Frage bejaht werden. Dies ergibt sich nahezu unmittelbar aus der folgenden Charakterisierung:

Satz 4 (Jiang et al. [6]) *Sei Σ ein Alphabet, $|\Sigma| \geq 2$, und seien $\alpha, \beta \in X^*$. Es gilt $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$ genau dann, wenn es einen Homomorphismus $\phi : X^* \to X^*$ mit $\phi(\beta) = \alpha$ gibt.*

Die Vermutung, daß sich Satz 4 kanonisch auf alle Klassen von ähnlichen E-Patternsprachen erweitern läßt, ist in der Literatur eingehend untersucht worden (z. B. von Ohlebusch und Ukkonen [8] und Reidenbach [9]), weil sich aufgrund eines Satzes von Jiang et al. [5] aus einer positiven Antwort direkt ableiten ließe, daß das wohl bekannteste offene Entscheidungsproblem für E-Patternsprachen, nämlich das Äquivalenzproblem, entscheidbar ist. Das nachfolgende Resultat zeigt jedoch, daß sich die Inklusion von allgemeinen ähnlichen E-Patternsprachen anders verhält als die von terminalfreien:

Satz 5 *Zu jedem endlichen Alphabet Σ existieren ähnliche Pattern $\alpha, \beta \in \mathrm{Pat}_\Sigma$ mit $L_{\mathrm{E},\Sigma}(\alpha) \subset L_{\mathrm{E},\Sigma}(\beta)$, für die es keinen terminalerhaltenden Homomorphismus $\phi : (\Sigma \cup X)^* \to (\Sigma \cup X)^*$ mit $\phi(\beta) = \alpha$ gibt.*

Aus Platzgründen muß der Beweis von Satz 5 entfallen; stattdessen seien lediglich die folgenden Pattern angeführt, welche seine Korrektheit für Al-

phabetgröße 5 belegen:

$$\begin{aligned}\alpha &= x_1\,\mathtt{a}\,x_2\,\mathtt{a}\,x_3\,\mathtt{b}\,x_2\,\mathtt{b}\,x_5\,\mathtt{c}\,x_2\,\mathtt{c}\,x_7\,\mathtt{d}\,x_2\,\mathtt{d}\,x_9\,\mathtt{e}\,x_2\,\mathtt{e}\,x_{11},\\ \beta &= x_1\,\mathtt{a}\,x_2\,x_4\,\mathtt{a}\,x_3\,\mathtt{b}\,x_4\,x_6\,\mathtt{b}\,x_5\,\mathtt{c}\,x_6\,x_8\,\mathtt{c}\,x_7\,\mathtt{d}\,x_8\,x_{10}\,\mathtt{d}\,x_9\,\mathtt{e}\,x_{10}\,x_2\,\mathtt{e}\,x_{11}.\end{aligned}$$

Satz 5 zeigt also, daß die zentrale, allen bislang publizierten Arbeiten zum Äquivalenzproblem zugrundeliegende Vermutung falsch ist, weswegen die Untersuchung dieses Themas mutmaßlich wieder am Anfang steht.

Literatur

[1] D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.

[2] D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.

[3] D. D. Freydenberger und D. Reidenbach. Bad news on decision problems for patterns. In *Proc. DLT 2008*, LNCS 5257, S. 327–338, 2008.

[4] O. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25:116–133, 1978.

[5] T. Jiang, E. Kinber, A. Salomaa, K. Salomaa und S. Yu. Pattern languages with and without erasing. *Int. J. Comput. Math.*, 50:147–163, 1994.

[6] T. Jiang, A. Salomaa, K. Salomaa und S. Yu. Decision problems for patterns. *Journal of Computer and System Sciences*, 50:53–63, 1995.

[7] A. Mateescu und A. Salomaa. Patterns. In G. Rozenberg und A. Salomaa (Hrsg.), *Handbook of Formal Languages*, Band 1, Kapitel 4.6, S. 230–242. Springer, 1997.

[8] E. Ohlebusch und E. Ukkonen. On the equivalence problem for E-pattern languages. *Theoretical Computer Science*, 186:231–248, 1997.

[9] D. Reidenbach. An examination of Ohlebusch and Ukkonen's conjecture on the equivalence problem for E-pattern languages. *Journal of Automata, Languages and Combinatorics*, 12:407–426, 2007.

[10] D. Reidenbach. Discontinuities in pattern inference. *Theoretical Computer Science*, 397:166–193, 2008.

[11] T. Shinohara. Polynomial time inference of extended regular pattern languages. In *Proc. RIMS Symposia on Software Sci. Eng.*, LNCS 147, S. 115–127, 1982.

Communication Complexity and Regular Expression Size*

Hermann Gruber

Institut für Informatik, Universität Giessen
Arndtstraße 2, 35392 Giessen, Germany
gruber@informatik.uni-giessen.de

Jan Johannsen

Institut für Informatik, Ludwig-Maximilians-Universität München
AOettingenstr. 67, 80538 München, Germany
jan.johannsen@informatik.uni-muenchen.de

1 Introduction

We consider the problem of converting a deterministic finite automaton (DFA) into a short regular expression (RE). Examples given by Ehrenfeucht and Zeiger in the 1970s show that the required expression size in the worst case is $2^{\Theta(n)}$ for infinite languages, and for finite languages in $n^{\Omega(\log\log n)}$ and $n^{O(\log n)}$, if the alphabet size is allowed to grow with the number of states n of the given automaton [2]. We develop a new lower bound method for regular expression size, based on communication complexity, to show that in the second case, the required size is indeed $n^{\Theta(\log n)}$, thus solving an old open problem stated in that work. Overmore, our witness languages are over a binary alphabet. For the case of infinite languages, exponential lower bounds for small alphabets have been obtained only very recently in [4, 6].

This is an abstract of the conference version [9], in which more details can be found.

*Most of the work was done while the first author was at Institut für Informatik, Ludwig-Maximilians-Universität München.

2 Preliminaries

We assume the reader to be familiar with the basic notions in formal language and automata theory as contained in [10]. In addition, we need the following notions: The *alphabetic width* (or *size*) of a regular expression E is defined as the total number of occurrences of symbols in Σ in E, and is denoted by $alph(E)$. In a similar way, for a regular language L we define $alph(L)$ as the minimum alphabetic width among all regular expressions describing L. We call a finite language L *homogeneous* if all words in L have the same length.

We will also need some notions from communication complexity theory. For a thorough treatment of that topic, the reader might want to consult [11]. Let X, Y, Z be finite sets and $R \subseteq X \times Y \times Z$ a ternary relation on them. In the search problem R, we have Alice given some input $x \in X$, Bob is given some input $y \in Y$. Initially, no party knows the other's input, and Alice and Bob both want to output some z such that (x, y, z) by communicating as few bits as possible. A *communication protocol* is a binary tree with each internal node v labeled either by a function $a_v : X \to \{0, 1\}$ if Alice transmits at this node, or $b_v : Y \to \{0, 1\}$ if Bob transmits at this node. Each leaf is labeled by an output $z \in Z$. We say that a protocol solves the search problem for relation R if for every input pair $(x, y) \in X \times Y$, walking down the tree according to the functions a_v and b_v leads to a leaf labeled with some admissible z, which satisfies $(x, y, z) \in R$. The *protocol partition number* $C^P(R)$ denotes the minimum number of leaves among all protocols solving the search problem for R. For using this notion in formal language theory, let $\Sigma = a_1, \ldots, a_k$ be an ordered alphabet. The order on Σ is extended componentwise to a partial order on Σ^n. A homogeneous language $L \subseteq \Sigma^n$ is called monotonic, if

$$v \in L \text{ and } w \geq v \text{ implies } w \in L \,.$$

For a homogeneous language $\emptyset \subset L \subset \Sigma^n$, the search problem $R_L \subseteq L \times (\Sigma^n \setminus L) \times [n]$ is defined by $(v, w, i) \in R_L$ iff $v_i \neq w_i$. If L is monotonic, then additionally the monotonic search problem R_L^m is defined by $(v, w, i) \in R_L^m$ iff $v_i > w_i$.

3 A new Lower Bound Technique for Regular Expression Size

We outline next how techniques from communication complexity can be used for proving lower bounds on the size of regular expressions for homogeneous languages.

Lemma 3.1 *For every homogeneous language L with $\emptyset \subset L \subset \Sigma^n$, $alph(L) \geq C^P(R_L)$. Moreover, if L is monotonic, then $alph(L) \geq C^P(R_L^m)$.*

The first part of the lemma was proved for the special case of the parity function in [1]—in terms of Boolean formula size instead of $C^P(R_L)$, but this is equivalent to the setup used here, see [11, Chapter 5]. We note that only the second part allows us to prove a superpolynomial lower bound on the conversion problem. To establish this bound, for a given pair of integers (ℓ, n), we define a family of graphs $\mathcal{G}_{\ell,n}$ as the set of directed graphs whose vertex set V is organized in $\ell + 2$ layers, with n vertices in each each layer. Hence we assume $V = \{ \langle i, j \rangle \mid 1 \leq i \leq n, 0 \leq j \leq \ell + 1\}$. For all graphs in $\mathcal{G}_{\ell,n}$, we require in addition that each edge connects a vertex in some layer i to a vertex in the adjacent layer $i + 1$. The following definition serves to represent subsets of $\mathcal{G}_{\ell,n}$ as homogeneous languages over the alphabet $\{0, 1\}$: Fix a graph $G \in \mathcal{G}_{\ell,n}$ for the moment. Let $e(i, j, k) = 1$ if G has an edge from vertex i in layer j to vertex k in layer $j + 1$, and let $e(i, j, k) = 0$ otherwise. Next, for vertex i in layer j, the word $f(i, j) = e(i, j, 1)e(i, j, 2) \cdots e(i, j, n)$ encodes the set of outgoing edges for this vertex. Then for layer j, the word $g(j) = f(1, j)f(2, j) \cdots f(n, j)$ encodes the set of edges connecting vertices in layer j to vertices in layer $j + 1$, for $0 \leq j \leq \ell$. Finally, the graph G is encoded by the word $w(G) = g(0)g(1) \cdots g(\ell)$. It is easy to see that each word in the set $\{0, 1\}^{n^2(\ell+1)}$ can be uniquely decoded as a graph in the set $\mathcal{G}_{\ell,n}$. Without risk of confusion, we will henceforth not distinguish between graphs and their encodings, and between sets of graphs and the corresponding languages. A graph $G \in \mathcal{G}_{\ell,n}$ belongs to the subfamily $\mathsf{fork}_{\ell,n}$, if there exists a simple path starting in $\langle 1, 1\rangle$ ending eventually in a fork, that is, a node with outdegree at least two. We prove that the language $\mathsf{fork}_{\ell,n}$ admits a small DFA but needs a large RE. More precisely, we show that $L_k = \mathsf{fork}_{\ell,n}$, for some $\ell \in \Theta(n^3)$, can be accepted by a DFA with at most $k = n^6$ states, but a large lower bound on the minimum expression size is obtained from using Lemma 3.1 by a reduction—in the communication complexity sense— from the FORK relation defined in [5].

Theorem 3.2 *There exist an infinite family of finite languages L_k over a binary alphabet such that L_k is acceptable by a DFA with at most k states, but every equivalent regular expression has alphabetic width at least*

$$k^{(1/144-\mathrm{o}(1))\log k}.$$

4 Conclusions and Further Research

In this work, we established an asymptotically tight lower bound for the problem of converting a finite automaton accepting a finite language over binary alphabet into a regular expression. As mentioned in the introduction, the case of infinite languages was also settled recently in [4, 6]. These and follow-up works [3, 7, 8] also study the effect of common language operations, such as intersection or complement, on regular expression size. A topic for further research would be a corresponding study for the case of finite languages, thus paralleling the developments in state complexity (see e.g. [12]).

References

[1] K. Ellul, B. Krawetz, J. Shallit and M. Wang. Regular Expressions: New Results and Open Problems. *Journal of Automata, Languages and Combinatorics*, 10(4):407–437, 2005.

[2] A. Ehrenfeucht and H. P. Zeiger. Complexity Measures for Regular Expressions. *Journal of Computer and System Sciences*, 12(2):134–146, 1976.

[3] W. Gelade. Succinctness of Regular Expressions with Interleaving, Intersection and Counting. In: *Proceedings of MFCS*: 363–374, LNCS 5162, Springer, 2008.

[4] W. Gelade and F. Neven. Succinctness of the Complement and Intersection of Regular Expressions. In: *Proceedings of STACS*: 325-336, Dagstuhl Seminar Proceedings 08001, IBFI Schloss Dagstuhl, 2008.

[5] M. Grigni and M. Sipser. Monotone Separation of Logarithmic Space from Logarithmic Depth. *Journal of Computer and System Sciences*, 50(3):433–437, 1995.

[6] H. Gruber and M. Holzer. Finite Automata, Digraph Connectivity, and Regular Expression Size. In: *Proceedings of ICALP*: 39–50, LNCS 5126, Springer, 2008.

[7] H. Gruber and M. Holzer. Provably Shorter Regular Expressions from Deterministic Finite Automata (Extended Abstract). In: *Proceedings of DLT*: 383–395, LNCS 5257, Springer, 2008.

[8] H. Gruber and M. Holzer. Language Operations with Regular Expressions of Polynomial Size. In: *Proceedings of DCFS*: 182–193, CSIT University of Prince Edward Island, 2008.

[9] H. Gruber and J. Johannsen. Optimal Lower bounds on Regular Expression Size using Communication Complexity. In: *Proceedings of FoSSaCS*: 273–286, LNCS 4962, Springer, 2008. Full version in preparation.

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

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

[12] S. Yu. State Complexity of Finite and Infinite Regular Languages. *Bulletin of the EATCS*, 76: 142–152, 2002.

Konstruktion endlicher Automaten aus regulären Ausdrücken

Stefan Gulan and Henning Fernau

Theoretische Informatik, FB IV, Universität Trier
Matthiasstraße 28a, 54290 Trier, Germany
{gulan, fernau}@uni-trier.de

1 Einführung und Definitionen

Betrachtet werden NFAs mit ϵ-Transitionen bzw. reguläre Ausdrücke über Produkt, Summe und Kleene-$*$. Die gegenseitige Beschreibungskomplexität dieser Formalismen ist nur wenig publiziert, wohl auch, weil keine einheitliche Definition der Größe dieser Objekte existiert. Wir verstehen hier unter der *Größe $|A|$ eines Automatens A* die gemeinsame Anzahl an Zuständen und Transitionen; die *Größe $|\alpha|$ eines Ausdrucks α* sei die Häufigkeit aller Operatoren und Terminale inklusive ϵ – entsprechend der Anzahl Knoten im Syntaxbaum von α. Bezüglich dieser Definitionen ist in [IY03] die Konstruktion eines Automatens A_α aus einem Ausdruck α mit $\frac{|A_\alpha|}{|\alpha|} \leq \frac{3}{2}$ angegeben. Weiter wird dort gezeigt, dass dieses Größenverhältnis nach unten durch $\frac{4}{3}$ beschränkt ist.

Wir geben ein α mit $\frac{|A_\alpha|}{|\alpha|} > 1,4\overline{6}$ für beliebige Konstruktion von A_α an. Es wird gezeigt, dass dieser Ausdruck den Größenquotionten für unsere Konstruktion maximiert. Die Konstruktion selbst ist eine Verfeinerung der in [OF61] vorgestellten Methode, die als Top-Down Konstruktion offenbar die bis dato einzige dieser Art ist. Diese Eigenschaft ermöglicht im wesentlichen, die Analyse des Algorithmus lokal durchzuführen. Alle hier ausgelassenen Beweise, bzw. Details davon finden sich in [GF08].

2 Eine untere Schranke

Lemma 2.1. *Seien $x_{i,j}$ paarweise verschiedene Terminale. Sei*

$$\alpha = (x_{1,1}^* + x_{1,2}^*)(x_{2,1}^* + x_{2,2}^* + x_{2,3}^*) \dots (x_{2n-1,1}^* + x_{2n-1,2}^*)(x_{2n,1}^* + x_{2n,2}^* + x_{2n,3}^*)$$

Dann gilt: Für jeden Automaten A mit $L(A) = L(\alpha)$ ist $|A| \geq 22n+1$.

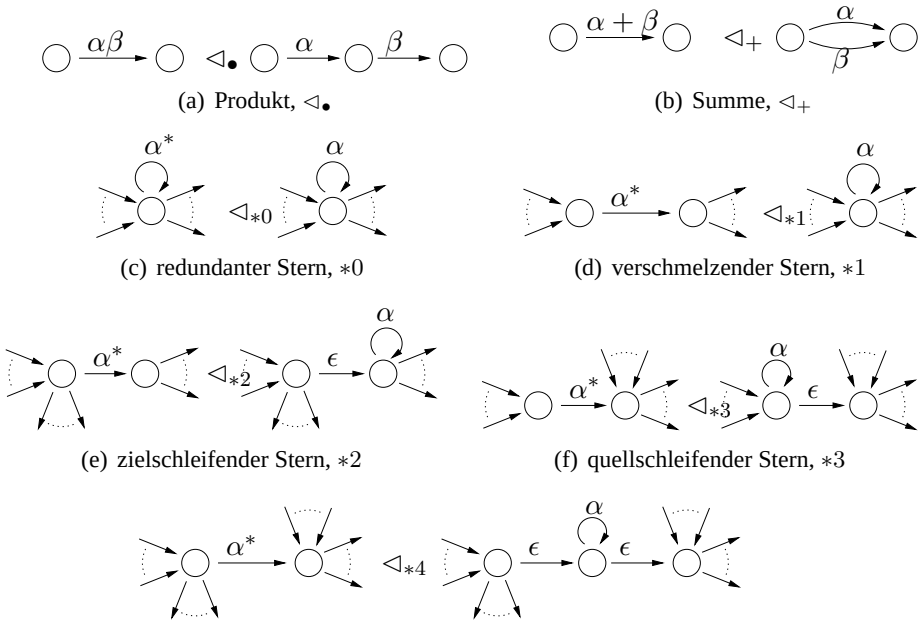
Korollar 2.1. *Für beliebige Konstruktionen $\alpha \mapsto A_\alpha$ gilt:* $\frac{|A_\alpha|}{|\alpha|} > \frac{22}{15} = 1,4\overline{6}$

Beweis. In Lem. 2.1 ist $|\alpha| = 15n-1$, also $\frac{|A_\alpha|}{|\alpha|} \geq \frac{22n+1}{15n-1} > \frac{22n}{15n-1} > \frac{22}{15}$ □

3 Konstruktion

Wesentliche Objekte der Konstruktion sind *erweiterte finite Automaten*, EFAs. Im Unterschied zum NFA sind Transitionen im EFA mit regulären Ausdrücken statt Terminalen markiert. Bei abarbeiten eines Wortes w durch einen EFA findet ein Zustandsübergang statt, wenn ein Teilwort in w gelesen werden kann, dass in der durch die Transitionsmarkierung beschriebenen Sprache liegt. NFAs sind sind spezielle EFAs; die Klasse der regulären Sprachen wird durch EFAs nicht erweitert.

Die Konstruktion formt EFAs ineinander um, bis ein NFA entsteht. Dabei werden *erweiternde Regeln* (Abb. 1), oder Expansionen, und *reduzierenden* Regeln (Abb. 2), oder Reduktionen, verwendet. Formal entsprechen die Regeln Relationen auf der Menge der EFAs.



(a) Produkt, $\lhd_\bullet$ (b) Summe, $\lhd_+$

(c) redundanter Stern, $*0$ (d) verschmelzender Stern, $*1$

(e) zielschleifender Stern, $*2$ (f) quellschleifender Stern, $*3$

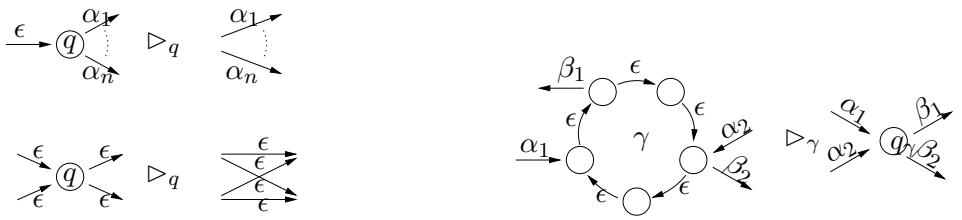
(g) erweiternder Stern, $*4$

Abbildung 1: Erweiternde Regeln gleichen die Struktur des Automatens durch lokale Expansion der des Ausdrucks an.

Relational schreiben wir $E \lhd_\bullet E'$, $E \lhd_+ E'$, $E \lhd_{*0} E'$, $\ldots$, $E \lhd_{*4} E'$, wenn E' aus E durch entsprechende Expansion hervorgeht, allgemeiner $E \lhd E'$ mit $\lhd = \lhd_\bullet \cup \lhd_+ + \bigcup_i \lhd_{*i}$. Sei $A^0_\alpha := (\{q_0, q_f\}, \mathcal{A}, (q_0, \alpha, q_f), q_0, q_f)$ der *primale* EFA zu α. Ist n die Anzahl der Operatoren in α, terminiert jede Folge von Expansionen aus A^0_α nach n Schritten, d.h. $A^0_\alpha \lhd^n A$, wobei A ein ϵNFA ist. Einen EFA der durch k Expansionen aus A^0_α hervorgeht, notieren wir als A^k_α. Ist β Markierung der bei $A^i_\alpha \lhd_{[\cdot]} A^{i+1}_\alpha$ expandierten Transition, so sagen wir, dass β $\lhd_{[\cdot]}$-expandiert wird. Jeder Teilausdruck von α wird bei vollständiger Expansion von A^0_α expandiert.

Lemma 3.1. *$\lhd$ ist konfluent.*

Beweis. $\lhd$ ist lokal konfluent. Da $\lhd$ ausserdem noethersch ist (s.o.), so nach Newman ([New42]) auch konfluent. □



(a) Zustandseliminerung (b) Zykluseliminierung

Abbildung 2: Eliminierung von Teilstrukturen, die durch ϵ-Transitionen definiert sind. Umkehren der Transitionspfeile in (a) ergibt eine weitere Regel.

Erweiterende und reduzierende Regeln werden in der Konstruktion zusammengefasst, $\lhd \cup \rhd_q \cup \rhd_\gamma$ ist jedoch nicht mehr konfluent. Genauer ist $\lhd \cup \rhd_\gamma$ konfluent, was durch Hinzunahme von $\rhd_q$ zerstört wird. $\rhd_q$ bereits für sich genommen nicht konfluent.

4 Analyse

Die Umformungen führen neue Zustände und Transitionen ein, bzw. eliminieren diese. Änderungen der Zustands- und Transitionsmengen sind in Tab. 1 quantifiziert. Die Größe eines Ausdrucks ist die Summe der Operatorenhäufigkeiten: $|\alpha| = |\alpha|_\bullet + |\alpha|_+ + |\alpha|_* + |\alpha|_\mathcal{A}$. Sei $|\alpha|_{*i}$ die Anzahl der $\lhd_{*i}$-Expansionen bei Umformung von A^0_α in einen NFA, dann ist $|\alpha|_* = \sum_i |\alpha|_{i*}$.

Lemma 4.1. *Sei A_α ein aus A^0_α konstruierter ϵ-NFA. Dann gilt*

$$|A_\alpha| \leq |\alpha| + 2|\alpha|_{*4} - |\alpha|_+ + 2$$

	$\lhd_\bullet$	$\lhd_+$	$\lhd_{*0}$	$\lhd_{*1}$	$\lhd_{*2}$, $\lhd_{*3}$	$\lhd_{*4}$	$\rhd_\gamma$	$\rhd_q$
$\Delta(\|Q\|)$	1	0	0	-1	0	1	-($\|\gamma\| - 1$)	-1
$\Delta(\|\delta\|)$	1	1	0	0	1	2	-$\|\gamma\|$	-1 oder 0

Tabelle 1: Anzahl neu eingeführter / entfernter Elemente bei Expansion und Reduktion.

Beweis. Zu $|A^0_\alpha|$=3 wird die Anzahl der durch Expansion eingeführten / entfernten Elemente addiert; durch Reduktion werden Elemente ausschliesslich entfernt, was zu einer Ungleichung führt

$$|A_\alpha| \leq 2|\alpha|_\bullet + |\alpha|_+ - |\alpha|_{*1} + |\alpha|_{*2} + |\alpha|_{*3} + 3|\alpha|_{*4} + 3$$

Mit $|\alpha|_\mathcal{A} = |\alpha|_\bullet + |\alpha|_+ + 1$ und $|\alpha| = |\alpha|_\bullet + |\alpha|_+ + |\alpha|_{*0} \ldots + |\alpha|_\mathcal{A}$, ergibt sich durch Umschreiben der rechten Seite

$$\begin{aligned} |A_\alpha| &\leq |\alpha| + |\alpha|_\bullet - |\alpha|_{*0} - 2|\alpha|_{*1} + 2|\alpha|_{*4} - |\alpha|_\mathcal{A} + 3 \\ &\leq |\alpha| + |\alpha|_\bullet + 2|\alpha|_{*4} - |\alpha|_\mathcal{A} + 3 \\ &= |\alpha| + 2|\alpha|_{*4} - |\alpha|_+ + 2 \end{aligned}$$

□

Ein Ausdruck, der den Quotienten aus Automaten- und Ausdrucksgröße maximiert, wird *maximal* genannt.

Korollar 4.1. *Sei α maximal. Dann gilt:*

1. $\frac{|A_\alpha|}{|\alpha|} = 1 + \frac{2|\alpha|_{*4} - |\alpha|_+ + 2}{|\alpha|}$.
2. *Jede Folge von A^i_α besteht nur aus Expansionsschritten.*
3. *A_α ist eindeutig.*

Beweis.

1. Aus Lem. 4.1 und Maximalität.
2. Reduktionen senken den Quotienten unter das Maximum, siehe Beweis zu Lem. 4.1.
3. Da keine Reduktionen in der Umformung vorkommen und $\lhd$ nach Lem. 3.1 konfluent ist.

□

Im Folgenden wird gezeigt, dass die Struktur eines maximalen Ausdrucks $\alpha_{\max}$ bis auf endliche Wiederholung eindeutig bestimmt ist.

Proposition 4.1. *In $\alpha_{\max}$ treten Sterne auf*

Beweis. Sei α maximal und $|\alpha|_*{=}0$. Kor. 4.1 ergibt $\frac{|A_\alpha|}{|\alpha|} \leq 1{+}\frac{2}{|\alpha|} \leq 1,4$ sobald $|\alpha| \geq 5$. In Abschnitt 2 wurde jedoch gezeigt, dass der maximale Größenquotient nach unten durch $1,4\overline{6}$ begrenzt ist. Also kann α nicht maximal sein. □

Lemma 4.2. *Sei γ^* ein Teilausdruck von α, der bei vollständiger Expansion von A^0_α $*4$-expandiert wird. Dann ist γ^**

- *Operand einer Summe, oder*
- *o.B.d.A letzter Faktor eines Produkts $\beta_1^* \ldots \beta_n^* \gamma^*$.*

Beweis. Ist γ^* selbst gesternt, wird es trivialerweise $*0$-expandiert. Ist γ^* Summand, wird zunächst die Summe expandiert, was die Bedingungen der $*4$-Expansion erfüllt (siehe Abb. 1). Sei also γ^* ein Faktor. Wegen Konfluenz kann angenommen werden, dass Produkte zunächst vollständig durch $\lhd_\bullet$, und im Anschluss die Faktoren von links nach rechts durch $\lhd_+ \cup \lhd_{*i}$ expandiert werden. □

Für $\alpha_{\max}$ wird Lem. 4.2 wieder eingeschränkt:

Lemma 4.3. *Ein Teilausdruck in $\alpha_{\max}$, der $*4$-expandiert wird, ist ein Summand.*

Beweis. Angenommen γ^* ist Faktor in $\alpha_{\max}$, der $*4$-expandiert wird, dann hat $\alpha_{\max}$ ein Produkt $\pi = \beta_1^* \ldots \beta_n^* \gamma^*$ als Teilausdruck. Sei α' der Ausdruck, der entsteht, wenn π in $\alpha_{\max}$ durch $\pi' = \beta_1^* + \ldots + \beta_n^* + \gamma^*$ ersetzt wird. Dann gilt $\frac{|A_{\alpha'}|}{|\alpha'|} > \frac{|A_{\alpha_{\max}}|}{|\alpha_{\max}|}$, im Widerspruch zur Maximalität von $\alpha_{\max}$. □

Lemma 4.4.

- *Jeder gesternte Teilausdruck in $\alpha_{\max}$ ist Operand einer Summe.*
- *Jede Summe in $\alpha_{\max}$ ist von der Form $\gamma_1^* + \gamma_2^* + \ldots + \gamma_n^*$.*

Lemma 4.5. *Die Struktur von $\alpha_{\max}$ ist*

$$\alpha_{\max} = \prod_{i=1}^{k} \sum_{j=1}^{l} x_{i,j}^*$$

wobei $x_{i,j} \in \mathcal{A}$.

Zuletzt sind die Obergrenzen der Indizes, k und l, in Thm. 4.5 zu untersuchen. Die Tatsache, dass bei Konstruktion von $A_{\alpha_{\max}}$ keine Reduktionen, also auch keine Zustandseliminerung auftritt, ist für die Analyse ausschlaggebend. Sie führt zu

Theorem 4.1.

$$\alpha_{\max} = \prod_{i=1}^{k} \sum_{j=1}^{(i \bmod 2)+1} x_{i,j}$$

Literatur

[GF08] Stefan Gulan and Henning Fernau. Top-down construction of finite automata from regular expressions. Technical Report 08-7, Universität Trier, 2008.

[IY03] Lucian Ilie and Sheng Yu. Follow automata. *Information and Computation*, (186):140–162, 2003.

[New42] M.H.A. Newman. On theories with a combinatorial definition of ”equivalence”. *Annals of Mathematics*, 43(2), 1942.

[OF61] Gene Ott and Neil H. Feinstein. Design of sequential machines from their regular expressions. *Journal of the ACM*, 8(4):585–600, 1961.

Language Operations with Regular Expressions of Polynomial Size*

Hermann Gruber and Markus Holzer

Institut für Informatik, Universität Giessen
Arndtstraße 2, 35392 Giessen, Germany
{gruber, holzer}@informatik.uni-giessen.de

In the last 20 years, a large body of research on the descriptional complexity of finite automata has been developed. To the authors' knowledge, the first systematic attempt to start a parallel development for the descriptional complexity of regular expressions was presented by Ellul et al. [4] at the workshop "Descriptional Complexity of Formal Systems" (DCFS), in 2002. In particular, they raised the question of determining how basic language operations such as complementation and intersection affect the required regular expression size. For the intersection and shuffle operation, exponential lower bounds are known, and complementation can even incur a doubly-exponential blow-up [5, 6]. In [6] it was shown that the star height of a regular language is at most logarithmic in the minimum regular expression size, and lower bounds are proved by finding families of languages for which the respective language operations give rise to a dramatic increase in star height. In contrast, it is well known that taking language quotients does not increase the star height [3]. This and similar language operations appear to be a natural testing ground for deepening our understanding of the descriptional complexity of regular expressions: Either one has to find some new lower bound techniques, or one has to find a nontrivial implementation of these operations on regular expressions, or both—a straightforward procedure would be to convert the expression into a finite automaton, implement the operation on a finite automaton, and convert back to a regular expression using state elimination. Yet that last step can incur an exponential

*Most of the work was done while the first author was at Institut für Informatik, Ludwig-Maximilians-Universität München, Oettingenstraße 67, 80538 München, Germany, and the second author was at Institut für Informatik, Technische Universität München, Boltzmannstraße 3, 85748 Garching bei München, Germany.

blow-up in general, even over binary alphabets [6].

Here, we give upper bounds for the required expression size resulting from taking language quotients and circular shift. The *(left) quotient* of L with respect to a set of words W, denoted by $W^{-1}L$, is defined as $\bigcup_{w\in W} w^{-1}L$, where $w^{-1}L = \{\, x \mid wx \in L \,\}$, for some word w. Moreover, the circular (or cyclic) shift of a language, denoted by $\circlearrowleft(L)$, is given by $\{\, xw \mid wx \in L \,\}$. Descriptional complexity aspects of these operations were already studied in [1, 11] for the circular shift and [8, 9] for language quotients—the latter two references consider deterministic finite automata with multiple start states, but the results easily translate to state complexity results for (left) quotients. The basic idea is to implement the operation for the special case of linear expressions [2] called single-occurrence regular expressions in [5]. These are expressions in which every alphabetic symbol occurs exactly once, which makes it easier to deal with as they can describe only local languages. To cover the general case, we study the interplay of the operations with length-preserving homomorphisms. The main result reads as follows—the size of a regular expression is defined as the total number of occurrences of letters from the underlying alphabet:

Theorem 1 *Let r be a regular expression of size n denoting the language $L \subseteq \Sigma^*$, and let $W \subseteq \Sigma^*$. Then there is a regular expression of size $O(n^2)$ denoting $W^{-1}L$ and a regular expression of size $O(n^3)$ denoting $\circlearrowleft(L)$.*

Currently, we do not know whether these upper bounds have the right order of magnitude. Nevertheless, it is worth mentioning that for the latter operation, i.e., the circular shift operation, at least an almost quadratic blow-up can be necessary in the worst case.

Theorem 2 *There exist infinitely many regular languages L_m over a binary alphabet such that L_m admits a regular expression of alphabetic width m, but every regular expression describing $\circlearrowleft(L_m)$ has size at least $\Omega\left(\frac{m^2}{\log^2 m}\right)$.*

One task for further research is to find other regularity preserving operations for which this or similar approaches might work. For instance, for the language of scattered substrings (superstrings, respectively) of the language described by a regular expression over Σ, we simply replace every position a with a subexpression $\lambda + a$ (with a subexpression describing $\Sigma^* a \Sigma^*$, respectively) to obtain a regular expression denoting that language. Both operations can be thus performed with only linear increase in expression size provided Σ is fixed. Issues on the state complexity of these operations were studied recently in [7] and [10].

References

[1] P. R. J. Asveld. Generating all circular shifts by context-free grammars in Greibach normal form. *International Journal of Foundations of Computer Science*, 18(6):1139–1149, 2007.

[2] G. Berry and R. Sethi. From regular expressions to deterministic automata. *Theoretical Computer Science*, 48:117–126, 1986.

[3] R. S. Cohen and J. A. Brzozowski. General properties of star height of regular events. *Journal of Computer and System Sciences*, 4(3):260–280, 1970.

[4] K. Ellul, B. Krawetz, J. Shallit, and M. Wang. Regular expressions: New results and open problems. *Journal of Automata, Languages and Combinatorics*, 10(4):407–437, 2005.

[5] W. Gelade and F. Neven. Succinctness of the complement and intersection of regular expressions. In S. Albers and P. Weil, editors, *Proceedings of the* 25*th Symposium on Theoretical Aspects of Computer Science*, volume 08001 of *Dagstuhl Seminar Proceedings*, pages 325–336, Bordeaux, France, February 2008. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany.

[6] H. Gruber and M. Holzer. Finite automata, digraph connectivity, and regular expression size. In L. Aceto, I. Damgaard, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walkuwiewicz, editors, *Proceedings of the* 35*th International Colloquium on Automata, Languages and Programming*, volume 5126 of *LNCS*, pages 39–50, Reykjavik, Iceland, July 2008. Springer.

[7] H. Gruber, M. Holzer, and M. Kutrib. More on the size of Higman-Haines sets: Effective constructions. In J. O. Durand-Lose and M. Margenstern, editors, *Proceedings of the* 5*th International Conference Machines, Computations, and Universality*, volume 4664 of *LNCS*, pages 193–204, Orléans, France, September 2007. Springer.

[8] M. Holzer, K. Salomaa, and S. Yu. On the state complexity of k-entry deterministic finite automata. *Journal of Automata, Languages and Combinatorics*, 6(4):453–466, 2001.

[9] M. Kappes. Descriptional complexity of deterministic finite automata with multiple initial states. *Journal of Automata, Languages and Combinatorics*, 5(3):269–278, 2000.

[10] A. Okhotin. On the state complexity of scattered substrings and superstrings. Technical Report TUCS Technical Report No.849, University of Turku - Department of Mathematics and Turku Centre for Computer Science and Academy of Finland, October 2007.

[11] A. Okhotin and G. Jirásková. State complexity of cyclic shift. *RAIRO—Theoretical Informatics and Applications*, 42(2):335–360, 2008.

M. Holzer, M. Kutrib, A. Malcher (Hrsg.):
18. Theorietag *Automaten und Formale Sprachen* 2008

Making Finite-State Methods Applicable to Languages Beyond Context-Freeness via Multi-dimensional Trees

Anna Kasprzik

Fachbereich IV, Abteilung Informatik, Universität Trier
54286 Trier, Germany
kasprzik@uni-trier.de

Abstract

We provide a new term-like representation for multi-dimensional trees as defined by Rogers [1, 2] which establishes them as a direct generalization of classical trees. As a consequence these structures can be used as input for finite-state applications based on classical term-based tree language theory. Via the correspondence between string and tree languages these applications can then be conceived to be able to process even some language classes beyond context-freeness. **Keywords:** Finite-state methods, Multi-dimensional Trees, Regularization

It is well known that string languages that are recognizable by finite-state automata, the so-called regular languages, have a whole range of advantageous mathematical properties. However, due to the relatively restricted character of the latter, classes that lie beyond regularity are often more interesting for applications based on formal language theory, even if the devices processing these languages (e.g., grammars or automata) are significantly more complex. Obviously, it would be of considerable use if one could reunite the advantages of regular and less restricted language classes by finding a way to handle these processes via regular mechanisms without giving up any of the expressive power.

Several such regularization methods have indeed been formulated, and at least two of them have been shown to be of use in the field of linguistics. A very prominent linguistic application of formal language theory is the area of natural language processing, i.e., conceiving the strings formed by a natural

language as a formal language in order to treat them automatically. Unfortunately, the study of certain phenomena (e.g., cross-serial dependencies in Dutch or Swiss German) showed that some of these string sets are not context-free. Joshi [3] claimed the least class of formal languages containing all natural language string sets to be situated between the context-free and the context-sensitive languages, and named it the class of *mildly context-sensitive languages*. The string sets generated by the grammar formalism defined by Joshi [3] himself, Tree Adjoining Grammar, prototypically fulfil all the necessary conditions for this class. TAG is considered the standard model for mild context-sensitivity and is the foundation of a considerable amount of current work in applied computational linguistics.

There are two methods of regularization for TAG (see [4]). Both methods are two-step approaches, i.e., they transform a TAG into a regular device (grammar or automaton) of some sort by representing its components in another shape and then reconstruct the intended objects from the objects generated or licensed by these devices via a simple process that can be carried out with regular means as well. One is based on an algebraic operation called Lifting (see [5]) which could be described as a way to write terms in a form that makes their internal structure more explicit, which, if the term is noted as a tree, has the side effect that all inner nodes are turned into leaves and thus become rewritable by substitution, which is a regular mechanism, and the other method (described by Rogers [1, 2]) makes use of an additional dimension in space by representing the components of a TAG as three-dimensional trees which likewise has the consequence that all inner nodes are turned into leaves and can be expanded by substitution.

The theoretical foundation of the second method are *multi-dimensional trees*, which are structures built over tree domains of arbitrarily many dimensions. Just like ordinary two-dimensional trees, every multi-dimensional tree has a string associated with it, which is obtained by reducing the dimensions of the tree step by step to its leaves. The classes of string languages associated with the recognizable multi-dimensional tree languages ordered by number of dimensions form a (proper) infinite hierarchy properly contained in the context-sensitive class, with the classes of finite languages (associated with zero-dimensional point sets), regular languages (one-dimensional string sets), context-free languages (two-dimensional tree sets, as for the classical definition) and the mildly context-sensitive string languages generated by (non-strict) TAGs (three-dimensional tree sets, see [1, 2]) as the first four steps. According to Rogers [1], this hierarchy coincides with Weir's Control Language Hierarchy [6].

It follows from these correspondences that by processing recognizable

higher-dimensional descriptions of non-regular string languages instead of the string sets themselves, finite-state methods become applicable again, and with them all the advantages and results pertaining to regularity. This can be a valuable insight in the area of natural language processing, but also in other areas based on formal language theory, e.g., grammatical inference: For instance, just as Angluin's learning algorithm for regular string languages [7] has been adapted to regular tree languages [8, 9], thereby making context-free string languages learnable (wrt the underlying learning model), this algorithm can be generalized to recognizable tree languages of arbitrarily many dimensions, making even string languages beyond context-freeness learnable in polynomial time (see [11]).

However, before such applications founded on formal tree languages can be generalized to arbitrarily many dimensions, there is a missing link to be provided: Most of them are not based on tree domains, as is Rogers' definition of multi-dimensional trees, but on the concept of trees as terms over a partitioned alphabet (the partitioning being induced by rank in the traditional case). In my paper [10] I give a new term-like representation for multi-dimensional trees, along with an adapted definition of finite-state automata for these structures, and prove the equivalence of the two notations. As a consequence multi-dimensional trees can be seen and used as a direct generalization of classical trees, and the full range of beneficial results for regular (tree) languages as known from formal language theory in the spirit of the Chomsky Hierarchy can be exploited.

References

[1] Rogers, J.: Syntactic Structures as Multi-dimensional Trees. Research on Language and Computation 1, 265–305 (2003)

[2] Rogers, J.: wMSO Theories as Grammar Formalisms. Theoretical Computer Science 293, 291–320 (2003)

[3] Joshi, A.K.: Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural description. In: Dowty, D., Karttunen, L., Zwicky, A. (eds.) Natural Language Processing. Cambridge University Press (1985)

[4] Kasprzik, A.: Two Equivalent Regularizations of Tree Adjoining Grammars. Tech. Report 08-1, University of Trier. Available on: urts117.uni-trier.de/cms/index.php?id=15939 (2008)

[5] Morawietz, F.: Two-Step Approaches to Natural Language Formalisms. Studies in Generative Grammar, vol. 64. Mouton de Gruyter (2003)

[6] Weir, D.J.: A geometric hierarchy beyond context-free languages. Theoretical Computer Science 104(2), 235–261 (1992)

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

[8] Sakakibara, Y.: Learning context-free grammars from structural data in polynomial time. Theoretical Computer Science 76(2–3), 223–242 (1990)

[9] Drewes, F., Högberg, J.: Learning a Regular Tree Language from a Teacher. In: Ésik, Z., Fülöp, Z. (eds.) Developments in Language Theory 2003. LNCS, vol. 2710, pp. 279–291. Springer (2003)

[10] Kasprzik, A.: Making Finite-State Methods Applicable to Languages Beyond Context-Freeness via Multi-dimensional Trees. Technical Report 08-3, University of Trier. Available on: urts117.uni-trier.de/cms/index.php?id=15939 (2008). To be published in the Proceedings of FSMNLP 2008.

[11] Kasprzik, A.: A learning algorithm for multi-dimensional trees, or: Learning beyond context-freeness. Tech. Report 08-2, University of Trier. Available on: urts117.uni-trier.de/cms/index.php?id=15939 (2008). To be published in the Proceedings of ICGI 2008.

Some Results on the Descriptional Complexity of Splicing Systems*

Remco Loos†

EMBL Outstation – Hinxton, European Bioinformatics Institute
Wellcome Trust Genome Campus, Hinxton, Cambridge, CB10 1SD, U. K.
remco@ebi.ac.uk

Andreas Malcher‡

Institut für Informatik, Universität Giessen
Arndtstraße 2, 35392 Giessen, Germany
malcher@informatik.uni-giessen.de

Detlef Wotschke

Institut für Informatik, Goethe-Universität
60054 Frankfurt a.M., Germany
wotschke@em.uni-frankfurt.de

Abstract

In this paper, the descriptional complexity of extended finite splicing systems is studied. These systems are known to generate exactly the class of regular languages. Upper and lower bounds are shown relating the size of these splicing systems, defined as the total length of the rules and the initial language of the system, to the size of their equivalent minimal nondeterministic finite automata (NFA). In addition, an accepting model of extended finite splicing systems is studied. Using this variant one can obtain systems which are more than polynomially more succinct than the equivalent NFA or generating extended finite splicing system.

*Summary of a paper presented at DCFS 2007, Nový Smokovec, Slovakia.

†The work of the first author was supported by Research Grant BES-2004-6316 of the Spanish Ministry of Education and Science. The work was done while the author was at Research Group on Math. Linguistics, Rovira i Virgili University, Tarragona, Spain.

‡The work was done while the author was at Institut für Informatik, Johann Wolfgang Goethe-Universität, Frankfurt, Germany.

Introduction

The splicing system (also known as H system) [3] is a model of molecular computation based on the cutting and recombination of DNA strands induced by restriction enzymes. Formalizing this process as a string rewriting operation, it can be used to define computational systems. As an operation on strings, splicing works as follows. First, two strings are cut into two parts at specific subsequences. Then, two new strings are created by concatenating the first part of the first string with the second part of the second string, and the first part of the second string with the second part of the first string. A *splicing rule* of the form $u_1\#u_2\$u_3\#u_4$ specifies the subsequences for which this process can take place. A *splicing system* starts out with a set of initial words and repeatedly applies a set of splicing rules to produce more words, thus generating a language.

Descriptional complexity issues of splicing systems have been previously studied, but the existing work has been mainly focused on limiting a specific resource or "structural" parameter. This typically involved increasing other resources or parameters. For instance, in [8], Păun showed that extended splicing systems with regular rules are still universal with only one initial word, or with initial words of length at most 1, but that these measures cannot be simultaneously minimized. Also the *radius* (the biggest u_i for a rule $u_1\#u_2\$u_3\#u_4$) of splicing rules has been studied in the context of H systems with additional control mechanisms, e.g. [6]. Finally, in distributed H systems, the question of reducing the number of components is well studied, see for instance [1, 5], sometimes together with radius considerations [7].

While measuring specific resources or parameters provides important insights into the role or the weight which these parameters play or carry, it does not necessarily say that much about the overall description necessary to specify a particular system, nor does it always allow to realistically compare the descriptional complexity of splicing systems with other specification methods, at least not when the various description methods use different and incompatible resources or parameters. We therefore take a different approach here by studying the total size of the system. This will allow us to compare H systems to other formalisms in terms of descriptional complexity and shed some light on the behavior of splicing systems when faced with specific tasks or problems. As a first step in this direction, we will study extended finite splicing systems, i.e., systems having a finite number of rules and starting with a finite number of initial words. It is known that non-extended finite splicing systems are sub-regular in power, while

extended finite splicing systems systems generate exactly the class of regular languages. This means that, modulo a terminal alphabet, all regular languages can be obtained by recombining finite sets while applying a finite number of different rules only.

Thus, it is interesting from a descriptional complexity perspective to compare the descriptive power of such systems with the standard model for regular languages, namely deterministic and nondeterministic finite automata (henceforth abbreviated by DFA and NFA, respectively). Many succinctness results between several models describing regular languages are known in the literature and are summarized in [2]. There are, e.g., exponential trade-offs between NFA and DFA, doubly exponential trade-offs between alternating finite automata and DFA, and polynomial trade-offs between NFA with very small finite amounts of nondeterminism and DFA.

Here, we will complement the above list by investigating the relative succinctness of representations between extended finite splicing systems and NFA. The size of a splicing system is defined as the sum of the total length of its rules and the total length of the words of its initial language. We prove upper and lower bounds for passing from a splicing system to an equivalent NFA and vice versa, to show that the size needed is similar for both systems. However, we show that using an accepting model of extended finite splicing systems, one can obtain systems which are more than polynomially more succinct than the equivalent NFA or generating extended finite splicing system.

Definitions

A splicing rule over an alphabet V is defined as a string $u_1\#u_2\$u_3\#u_4$, with $u_1, u_2, u_3, u_4 \in V^*$, and $\$, \#$ special symbols not in V. We always assume that $u_1u_2, u_3u_4 \neq \lambda$. The length $|r|$ of a splicing rule $r = u_1\#u_2\$u_3\#u_4$ is defined as $|u_1| + |u_2| + |u_3| + |u_4|$. For a splicing rule $r = u_1\#u_2\$u_3\#u_4$ and $x, y, w, z \in V^*$, we write $(x, y) \vdash_r (w, z)$ if and only if $x = x_1u_1u_2x_2$, $y = y_1u_3u_4y_2$, $z = x_1u_1u_4y_2$, $w = y_1u_3u_2x_2$, for some $x_1, x_2, y_1, y_2 \in V^*$.

A *splicing scheme* is a pair (V, R), with V an alphabet and R a set of splicing rules over V. For a splicing scheme $h = (V, R)$ and a language L over V we define $\sigma_h(L) = \{w, w' \in V^* \mid (w_1, w_2) \vdash_r (w, w') \text{ for some } w_1, w_2 \in L \text{ and some rule } r \in R\}$. Given a splicing scheme h and an initial language L, the splicing language $\sigma_h^*(L)$ is defined as $\sigma_h^0(L) = L$, $\sigma_h^{i+1}(L) = \sigma_h^i(L) \cup \sigma_h(\sigma_h^i(L)), i \geq 0$, and $\sigma_h^*(L) = \bigcup_{i\geq 0} \sigma_h^i(L)$.

A *splicing system* or *H system* is a triple $H = (V, A, R)$, where V is an alphabet, $A \subseteq V^*$ is the initial language, and R is a set of splicing rules

over V. The generated language is defined as $L(H) = \sigma^*(A)$. An *extended* H system $H = (V, T, A, R)$ has a terminal alphabet T and generates the language $L(H) = \sigma^*(A) \cap T^*$. We say that an H system is finite when it has a finite set of rules and a finite initial language, i.e., A and R are finite sets.

An *accepting* splicing system [4] is a quadruple $\Gamma = (V, A, R, \underline{YES}, \langle, \rangle)$ where $\underline{YES}, \langle, \rangle \in V$ and $H_\Gamma = (V, A, R)$ is a splicing system. Let $\Gamma = (V, A, R, \underline{YES}, \langle, \rangle)$ be an accepting splicing system. We say that Γ *accepts* a word $w \in V^*$ if and only if the following condition holds:

$$\underline{YES} \in \sigma^k(A \cup \{\langle w \rangle\}) \text{ for some integer } k.$$

Thus the language accepted by Γ is $L(\Gamma) = \{w \in V^* \mid \Gamma \text{ accepts } w\}$.

An *extended* accepting splicing system $\Gamma = (V, T, A, R, \underline{YES}, \langle, \rangle)$ is defined similarly as in the generating case. The language accepted by Γ is defined as $L(\Gamma) = \{w \in T^* \mid \Gamma \text{ accepts } w\}$. In what follows, we use the abbreviations EH(FIN) for (generating) extended splicing systems with finite rules and AEH(FIN) for accepting extended splicing systems with finite rules.

We define the following complexity measures. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA. Then we denote $\mathcal{Q}(M) = Card(Q)$ and $\mathcal{T}(M) = Card(\delta)$. Since the states are implicit in the description of the transitions, we define the total size of M as $\mathsf{Size}(M) = \mathcal{T}(M)$. Given an (A)EH(FIN) $\Gamma = (V, T, A, R)$ we define the measures $\mathcal{R}(\Gamma) = \sum_{r \in R} |r|$ and $\mathcal{A}(\Gamma) = \sum_{w \in A} |w|$. The total size of Γ is defined as $\mathsf{Size}(\Gamma) = \mathcal{A}(\Gamma) + \mathcal{R}(\Gamma)$.

Extended Splicing Systems

EH(FIN) systems describe exactly the regular languages. Concerning the descriptional complexity we obtain the following upper and lower bounds for some constant c.

Theorem 1 *When converting an NFA M to an equivalent EH(FIN) Γ:*

$$\mathsf{Size}(\Gamma) \leq c \cdot \mathsf{Size}(M) \text{ and } \mathsf{Size}(\Gamma) \geq \mathsf{Size}(M)$$

When converting an EH(FIN) Γ to an equivalent NFA M:

$$\mathsf{Size}(M) \leq c \cdot \mathsf{Size}(\Gamma)^2 \text{ and } \mathsf{Size}(M) \geq \mathsf{Size}(\Gamma) - 9$$

Theorem 2 *The problems of membership, emptiness, and finiteness are solvable in polynomial time for a given EH(FIN) system. The problems of equivalence, inclusion, and universality are not solvable in polynomial time for EH(FIN) systems unless* $\mathsf{P} = \mathsf{PSPACE}$.

Accepting Extended Splicing Systems

Conjecture 3 *$\mathcal{L}(AEH(FIN)) = REG$, i.e., the class of languages accepted by accepting extended splicing systems is exactly the class of regular languages.*

Theorem 4 *When converting an NFA M to an equivalent AEH(FIN) Γ, we obtain the following upper and lower bounds for some constants c, c':*

$$\mathsf{Size}(\Gamma) \leq c \cdot \mathsf{Size}(M) \text{ and } \mathsf{Size}(\Gamma) \geq \mathsf{Size}(M) - c'$$

Theorem 5 *There is no polynomial function f such that for any AEH(FIN) Γ there exists an equivalent NFA M such that $f(\mathsf{Size}(\Gamma)) \geq \mathsf{Size}(M)$.*

References

[1] R. Freund and F. Freund. Test tube systems: when two tubes are enough. *Preproceedings of DLT99, Developments in Language Theory*, World Scientific Publishing Company, 275–286, 2000.

[2] J. Goldstine, M. Kappes, C.M.R. Kintala, H. Leung, A. Malcher and D. Wotschke, Descriptional complexity of machines with limited resources, *J. Universal Computer Science*, 8(2):193–234, 2002.

[3] T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, *Bull. Math. Biology*, 49:737–759, 1987.

[4] R. Loos, C. Martín-Vide and V. Mitrana, Solving SAT and HPP with accepting splicing systems, *PPSN IX, Lecture Notes in Computer Science* 4193, 771–777, 2006.

[5] M. Margenstern, Y. Rogozhin and S. Verlan, Time-varying distributed H systems of degree 2 can carry out parallel computations, *DNA8, Lecture Notes in Computer Science* 2568, 326–336, 2003.

[6] A. Păun, Extended H systems with permitting contexts of small radius, *Fundamenta Informaticae*, 31:185–193, 1997.

[7] A. Păun, On time-varying H systems, *Bulletin of the EATCS*, 67:157–164, 1999.

[8] Gh. Păun, Regular extended H systems are computationally universal, *J. Automata, Languages, Combinatorics*, 1(1):27–36, 1996.

On Nonforgetting Restarting Automata That Are Deterministic and/or Monotone

Hartmut Messerschmidt

Fachbereich Elektrotechnik/Informatik, Universität Kassel
34109 Kassel, Germany
hardy@theory.informatik.uni-kassel.de

Abstract

The nonforgetting restarting automaton is a restarting automaton that is not forced to reset its internal state to the initial state when executing a restart operation. We analyse the expressive power of the various deterministic and monotone variants of this model.

1 Introduction

The restarting automaton was introduced by Jančar et. al. [JMPV95] to model the *analysis by reduction*, which is a technique used in linguistics to analyse sentences of natural languages. According to this technique a sentence is processed by simplifying it stepwise, in each step preserving the correctness or incorrectness of the sentence. In Czech and German linguistics already several programs use the idea of restarting automata. Nonforgetting restarting automata were introduced in [MS04], the following results are taken from [Mes08].

A nonforgetting restarting automaton, nf-RLWW-automaton for short, is a nine tuple $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$, where

- Q is a finite set of states,
- Σ is a finite input alphabet,
- $\Gamma \supseteq \Sigma$ is a finite tape alphabet,
- $¢, \$ \notin \Gamma$ are border markers,

- $q_0 \in Q$ is the initial state
- $k \geq 1$ is the window size.

Its actions are governed by a *transition relation* δ that assigns to each pair (q, u) consisting of a state $q \in Q$ and a possible contents u of the window a finite set of transition steps, of which there are five types: *move-right* and *move-left steps*, which change the internal state of M and shift the window one position to the right or to the left, respectively, *rewrite steps* that change the internal state and replace the contents of the window by some shorter string, thereby shortening the tape, *restart steps* that place the window over the left end of the tape, and *accept steps* that cause M to halt and accept.

Thus, each computation of M proceeds in *cycles*: starting from a (restarting) configuration of the form $q\¢x\$$, M performs move-right, move-left and rewrite steps until a restart operation takes it back to a restarting configuration of the form $q'\¢y\$$. As part of the definition it is required that in each such cycle M executes *exactly one* rewrite operation. The part of a computation that follows after the last application of a restart step will be denoted as the *tail* of the computation. By now many variants of the restarting automaton have been studied, and it has been shown that many well studied language classes can be characterised by variants of the restarting automaton. Actually the main variants of the restarting automaton are obtained by combining two types of restrictions:

(a) Restrictions on the movement of the read/write window (expressed by the first part of the class name):
   - RL- means no restriction,
   - RR- means that no move-left operations are available,
   - R- means that no move-left operations are available and that each rewrite step is followed immediately by a restart step.

(b) Restrictions on the rewrite instructions (expressed by the second part of the class name, where λ denotes the empty string):
   - -WW means no restriction,
   - -W means that no auxiliary symbols are available (that is, $\Gamma = \Sigma$),
   - -λ means that each rewrite step simply deletes some symbols, that is, if $(q', v) \in \delta(q, u)$, then v is a scattered proper subword of u.

2 Deterministic Monotone Restarting Automata

Here we concentrate on nonforgetting restarting automata that are deterministic and monotone. A restarting automaton M is monotone, if the distance of the place of the rewrite step and the right border is nonincreasing in any computation of M.

It has been shown that deterministic nonforgetting restarting automata are more powerful than deterministic forgetting ones. This is as well true for monotone restarting automata that are deterministic.

Forgetting deterministic monotone restarting automata recognize exactly DCFL, and the R-model is as powerful as the RRWW-model. This is shown in [JMPV99] by a simulation of a det-mon-RRWW-automaton by a deterministic PDA. This result can be adapted to nonforgetting automata, however only in a weaker form.

Theorem 2.1. $\mathsf{DCFL} = \mathcal{L}(\text{det-mon-nf-R}) = \mathcal{L}(\text{det-mon-nf-RWW})$.

This theorem only covers nonforgetting restarting automata that restart immediately after executing a rewrite step. In the next paragraphs we show that this theorem is only valid for these restarting automata. Because already det-mon-nf-RR-automata accept languages that are not deterministic context-free.

Example 2.2. $L_{\overline{\mathsf{pal}}} = \{wc\bar{w}^R \mid w \in \{a,b,c\}^*, \bar{a} = a, \bar{b} = b, \bar{c} = \varepsilon\} \in \mathcal{L}(\text{det-mon-nf-RR})$*:*

To show that $L_{\overline{\mathsf{pal}}} \notin \mathsf{DCFL}$ is valid, we use Ogdens Lemma for deterministic context-free languages.

With small changes we can find separation languages that prove the strictness of the inclusions between $\mathcal{L}$(det-mon-nf-RR), $\mathcal{L}$(det-mon-nf-RRW) and $\mathcal{L}$(det-mon-nf-RRWW). To simplify the description of the restarting automata, we use the additional restriction $|w|_c \geq 1$ for the other languages. Without this restriction the same results hold. Here (EF), for $e, f \in \{a,b,c\}$, are extra symbols that are used to encode the tape content.

$$
\begin{aligned}
L_{\overline{\mathsf{pal}}} &= \{wc\bar{w}^R \mid w \in \{a,b,c\}^*, \bar{a} = a, \bar{b} = b, \bar{c} = \varepsilon\} \in \mathcal{L}(\text{det-mon-nf-RR}) \setminus \mathsf{DCFL} \\
L_{\mathsf{pal}''} &= \{wc\phi(w^R) \mid w \in \{a,b,c\}^*, \phi(a) = aa,\ \phi(b) = ab,\ \phi(c) = ba, |w|_c \geq 1\} \\
&\qquad \in \mathcal{L}(\text{det-mon-nf-RRWW}) \setminus \mathcal{L}(\text{det-mon-nf-RRW}) \\
L_{\overline{\mathsf{pal}}''} &= \{w(AA)(AB)(AC)(BA)(BB)(BC)(CA)(CB)(CC) \mid w \in L_{\mathsf{pal}''}\} \\
&\qquad \in \mathcal{L}(\text{det-mon-nf-RRW}) \setminus \mathcal{L}(\text{det-mon-nf-RR})
\end{aligned}
$$

In $L_{\overline{\mathsf{pal}}}$ the morphism $d \to \overline{d}$ only removes the cs. In $L_{\mathsf{pal}''}$ the morphism ϕ encodes each letter by a string of length two, and in $L_{\overline{\mathsf{pal}}''}$ each word from

$L_{\mathsf{pal}''}$ is followed by nine additional symbols that are used to encode two input symbols of $L_{\mathsf{pal}''}$.

All of these languages are in $\mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RL})$, because we always look for the last c, which is trivial for RL-automata. We will now state a result that goes much further. In this theorem shrinking nonforgetting restarting automata are used. A shrinking restarting automaton does not need to reduce the length of the tape in each rewrite step $u \to v$, it must only reduce, for a given weight function, the weight. That means the weight of v must be smaller than the weight of u.

Theorem 2.3. $\mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}sh\text{-}RLWW}) = \mathcal{L}(\mathsf{det\text{-}mon\text{-}RL})$.

The only question left to answer is the exact relationship between deterministic monotone nonforgetting RL-automata and deterministic monotone nonforgetting RRWW-automata. Here the following theorems can be achieved.

Theorem 2.4. $\mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}sh\text{-}RRWW}) = \mathcal{L}(\mathsf{det\text{-}mon\text{-}RL})$.

Theorem 2.5. $\mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RRWW}) = \mathcal{L}(\mathsf{det\text{-}mon\text{-}RL})$.

3 Concluding Remarks

This is the first time that some variants of RWW and RRWW-automata are separated. Just recently it was shown that the language class $\mathcal{L}(\mathsf{det\text{-}mon\text{-}RL})$ coincides with the class of left-to-right regular languages (LRR). For a definition of LRR grammars and languages see [CC73].

Proposition 3.1. [Ott07] $\mathsf{LRR} = \mathcal{L}(\mathsf{det\text{-}mon\text{-}RL})$.

So the classes $\mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RWW})$ and $\mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RRWW})$ are not only different, but they both describe well-known language classes. Thus we get the following taxonomy of deterministic monotone nonforgetting restarting automata:

References

[CC73] K. Culic, II and R. Cohen. LR-regular grammars - an extension of LR(k) grammars. J. Computer System Sciences, 7:66–96, 1973.

$$
\begin{array}{ccccc}
\mathrm{LRR} & = & \mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}sh\text{-}RLWW}) & = & \mathcal{L}(\mathsf{det\text{-}mon\text{-}RL}) \\
 & & {\scriptstyle L_{\mathsf{pal}''}}\uparrow & & \\
 & & \mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RRW}) & & \\
 & & {\scriptstyle L_{\overline{\mathsf{pal}}''}}\uparrow & & \\
 & & \mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RR}) & & \\
 & & {\scriptstyle L_{\overline{\mathsf{pal}}}}\uparrow & & \\
\mathsf{DCFL} & = & \mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}R}) & = & \mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RWW})
\end{array}
$$

Figure 1: The taxonomy of deterministic monotone nonforgetting restarting automata

[JMPV95] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. Restarting automata. In H. Reichel, editor, *Fundamentals of Computation Theory, Proceedings FCT'95*, *Lecture Notes in Computer Science 965*, pages 283–292. Springer-Verlag, Berlin, 1995.

[JMPV99] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. On monotonic automata with a restart operation. J. Automata, Languages and Combinatorics, 4:287–311, 1999.

[Mes08] H. Messerschmidt. CD-Systems of Restarting Automata. Dissertation, Fachbereich Elektrotechnik/Informatik, Universität Kassel, 2008.

[MS04] H. Messerschmidt and H. Stamer. Restart-automaten mit mehreren restart-zuständen. In H. Bordihn, editor, *Workshop "Formale Methoden in der Linguistik" und 14. Theorietag "Automaten und Formale Sprachen", Proc.*, pages 111–116. Institut für Informatik, Universität Potsdam, 2004.

[Ott07] F. Otto. Left-to-right regular languages and two-way restarting automata. *manuscript*, 2007.

Conjunctive Grammars with Restricted Disjunction*

Alexander Okhotin

Department of Mathematics, University of Turku
20014 Turku, Finland
alexander.okhotin@utu.fi

Christian Reitwießner

Theoretische Informatik, Universität Würzburg
Am Hubland, 97074 Würzburg, Germany
reitwiessner@informatik.uni-wuerzburg.de

Abstract

It is shown that every conjunctive language is generated by a conjunctive grammar of a special form, in which every nonterminal A has at most one rule of the general form $A \to \alpha_1 \& \ldots \& \alpha_n$, while the rest of the rules for A must be of the type $A \to w$, where w is a terminal string. For context-free grammars, a similar property does not hold (S. A. Greibach, W. Shi, S. Simonson, "Single tree grammars", 1992).

1 Introduction

As *conjunction* of syntactical conditions is not expressible in context-free grammars, they can be extended by allowing an explicit conjunction in the formalism of rules. The resulting extension is known as *conjunctive grammars* [3]. The present paper continues the investigation of the power of Boolean operations in context-free grammars with a subclass of conjunctive grammars in which each nonterminal A may have only one rule referring to other nonterminals, while the rest of its rules must be of the form $A \to w$, where w is a terminal string. As a result, the implicit disjunctions created by having multiple rules for one nonterminal are restricted. The same

*Supported by the Academy of Finland under grant 118540.

restriction on the context-free grammars has been studied by Greibach et al. [1] under the name of *single tree grammars*. These grammars have quite a limited expressive power, in particular, they cannot generate the language of all palindromes.

Contrasting this reduction of expressive power, we can show that the same restriction can be put on conjunctive grammars at no cost: Every conjunctive language can be generated by a conjunctive grammar with restricted disjunction. The form with restricted disjunction may thus be regarded as a normal form for conjunctive grammars.

The proof of this result is based upon another normal form for conjunctive grammars, the *odd normal form*, in which every nonterminal other than the start symbol generates only strings of odd length.

A full version of this paper is available as a technical report [4].

2 Conjunctive grammars

Details about conjunctive grammars can be found in a paper by Okhotin [3], we will only define them briefly.

Definition 2.1 (Okhotin [3])**.** *A conjunctive grammar is a quadruple* $G = (\Sigma, N, P, S)$ *in which* Σ *and* N *are disjoint finite nonempty sets of terminal and nonterminal symbols respectively;* P *is a finite set of grammar rules, each of the form* $A \to \alpha_1 \& \ldots \& \alpha_n$ *(with* $A \in N$*,* $n \geqslant 1$ *and* $\alpha_1, \ldots, \alpha_n \in (\Sigma \cup N)^*$*) and* $S \in N$ *is a nonterminal designated as the start symbol.*

Definition 2.2 ([3])**.** *Given a grammar* G*, consider terms over concatenation and conjunction with symbols from* $\Sigma \cup N$ *as atomic terms. The relation* $\Longrightarrow$ *of immediate derivability on the set of terms is defined as follows:*

- *Using a rule* $A \to \alpha_1 \& \ldots \& \alpha_n$*, a subterm* $A \in N$ *of any term* $\varphi(A)$ *can be rewritten as* $\varphi(A) \Longrightarrow \varphi(\alpha_1 \& \ldots \& \alpha_n)$*.*
- *A conjunction of several identical strings can be rewritten by one such string:* $\varphi(w \& \ldots \& w) \Longrightarrow \varphi(w)$*, for every* $w \in \Sigma^*$*.*

The language generated by a term φ *is* $L_G(\varphi) = \{ w \mid w \in \Sigma^*,\ \varphi \Longrightarrow^* w \}$*. The language generated by the grammar is* $L(G) = L_G(S) = \{ w \mid w \in \Sigma^*,\ S \Longrightarrow^* w \}$*.*

Let us give some examples of conjunctive grammars. Every language representable as an intersection of finitely many context-free languages, such as $\{ a^n b^n c^n \mid n \geqslant 0 \}$, can be straightforwardly specified using conjunction

for the start symbol. It is more interesting to construct a grammar for a language not in the intersection closure of the context-free languages, such as the following.

Example 2.3 (Okhotin [3])**.** *The conjunctive grammar*

$$\begin{array}{lcl} S & \to & C\&D \\ C & \to & aCa \mid aCb \mid bCa \mid bCb \mid c \\ D & \to & aA\&aD \mid bB\&bD \mid cE \\ A & \to & aAa \mid aAb \mid bAa \mid bAb \mid cEa \\ B & \to & aBa \mid aBb \mid bBa \mid bBb \mid cEb \\ E & \to & aE \mid bE \mid \varepsilon \end{array}$$

generates the language $\{ wcw \mid w \in \{a,b\}^* \}$*. In particular,* $L(D) = \{ uczu \mid u,z \in \{a,b\}^* \}$*.*

The rules for D match a single symbol in the left part to the corresponding symbol in the right part using A or B, and the recursive reference to aD or bD makes the remaining symbols be compared in the same way. The intersection with the language $\{ ucv \mid u,v \in \{a,b\}^*, |u| = |v| \}$ generated by C completes the grammar.

Example 2.4 (Jeż [2])**.** *The following conjunctive grammar with the start symbol* A_1 *generates the non regular unary language* $\{ a^{4^n} \mid n \geqslant 0 \}$*:*

$$\begin{array}{lcl} A_1 & \to & A_2A_2\&A_1A_3 \mid a \\ A_2 & \to & A_6A_2\&A_1A_1 \mid aa \\ A_3 & \to & A_6A_6\&A_1A_2 \mid aaa \\ A_6 & \to & A_3A_3\&A_1A_2 \end{array}$$

Each nonterminal A_i *generates the language* $\{ a^{i\cdot 4^n} \mid n \geqslant 0 \}$*.*

3 The odd normal form

The *odd normal form* for conjunctive grammars proposed in this section has the following main property: every nonterminal (possibly except the start symbol) may only generate strings of odd length. Let us introduce the notation $\mathrm{Odd} := \Sigma(\Sigma^2)^*$ and $\mathrm{Even} := (\Sigma^2)^*$ (where Σ is the implicitly assumed alphabet) for the sets of all strings of odd and even length, respectively.

Definition 3.1 (Odd normal form)**.** *A conjunctive grammar* $G = (\Sigma, N, P, S)$ *is said to be in* odd normal form *if all rules in* P *are of the*

form

$$\begin{array}{ll} A \to a & \text{with } A \in N,\ a \in \Sigma, \qquad \text{or} \\ A \to B_1 a_1 C_1 \,\&\, \ldots \,\&\, B_n a_n C_n & \text{with } n \geqslant 1,\ A, B_i, C_i \in N,\ a_i \in \Sigma \end{array}$$

If S does not occur in the right-hand sides of the rules, then the following two types of rules, called ***even rules****, are also allowed:*

$$\begin{array}{ll} S \to aA & \text{with } a \in \Sigma,\ A \in N \\ S \to \varepsilon & \end{array}$$

Note that if there are no even rules in a grammar in odd normal form, then it generates a subset of Odd. Thus even rules are needed for some languages, but regardless of whether are used, the main part of the grammar operates on odd strings only.

The main idea for transforming a given grammar G into odd normal form is to add nonterminals that generate the languages $x^{-1}L_G(A)y^{-1} \cap \mathrm{Odd}$ for any original nonterminal A and $x, y \in \Sigma \cup \{\varepsilon\}$. The "removed" terminals are inserted again when two nonterminals are concatenated. This can be done for any conjunctive grammar.

Theorem 3.2. *For every conjunctive grammar there exists and can be effectively constructed a conjunctive grammar in odd normal form generating the same language.*

4 Restricted conjunctive grammars

Now let us define a restricted subfamily of conjunctive grammars that will be studied in this paper.

Definition 4.1. *A* restricted conjunctive grammar *is a conjunctive grammar in which every nonterminal may have at most one rule not of the form $A \to w$, with $w \in \Sigma^*$. In other words, the rules for every nonterminal A are of the form:*

$$A \to \alpha_1 \& \ldots \& \alpha_n \mid w_1 \mid \ldots \mid w_m \quad (n \geqslant 1,\ m \geqslant 0,\ \alpha_i \in (\Sigma \cup N)^*,\ w_j \in \Sigma^*)$$

The grammar in Example 2.4 is restricted conjunctive, while the grammar in Example 2.3 is not. We will now see that there is a restricted conjunctive grammar for every conjunctive language. The key idea is to use the fact that for $A, B \subseteq \mathrm{Odd}$ it holds that $(A \cup \{\varepsilon\})(B \cup \{\varepsilon\}) \cap \mathrm{Odd} =$

$(AB \cup A \cup B \cup \{\varepsilon\}) \cap \mathrm{Odd} = A \cup B$. Since the languages generated by nonterminals of a grammars in odd normal form only consist of words of odd length, we can use this property to simulate the unions.

Lemma 4.2. *For every conjunctive grammar $G = (\Sigma, N, P, S)$ generating a subset of* Odd *there exists and can be effectively constructed a restricted conjunctive grammar generating the same language.*

For every conjunctive language $L \subseteq \Sigma^*$ one can construct restricted conjunctive grammars that generate the languages $L \cup \mathrm{Even}$ and $L \cup \overline{a \cdot \mathrm{Odd}}$ for every $a \in \Sigma$. Since the intersection of all these languages is $(L \cup \mathrm{Even}) \cap \bigcap_{a \in \Sigma}(L \cup \overline{a\mathrm{Odd}}) = L \cup \varepsilon$ we can now create a restricted conjunctive grammar for the full language. The following theorem, the main result of this paper, is then obtained.

Theorem 4.3. *Every conjunctive language is generated by a restricted conjunctive grammar.*

5 Restricted conjunctive grammars without ε

The above simulation of an arbitrary conjunctive grammar by a conjunctive grammar with restricted disjunction essentially uses rules of the form $A \to \varepsilon$, known as *ε-rules*. Since conjunctive grammars of the general form do not need ε-rules, this raises the question of whether restricted conjunctive grammars without ε-rules are as powerful as conjunctive grammars of the general form.

First of all, this stronger restriction on conjunctive grammars still gives a non-trivial family. For instance, the important unary grammar given in Example 2.4 is of this form. Grammars for interesting languages over larger alphabets (for example the palindromes but also $\{wcw \mid w \in \{a, b\}\}$) can be constructed as well. Furthermore, it can be shown that conjunctive grammars of a special form (including grammars generating all regular languages) can be simulated by restricted conjunctive grammars without ε-rules. Further details can be found in the technical report of this paper [4].

The exact expressive power of conjunctive grammars with restricted disjunction and without ε-rules is left as an open question to study.

References

[1] S. A. Greibach, W. Shi, S. Simonson, "Single tree grammars", in: J. D. Ullman (Ed.), *Theoretical Studies in Computer Science*, Academic Press, 1992, 73–99.

[2] A. Jeż, "Conjunctive grammars generate non-regular unary languages", *International Journal of Foundations of Computer Science*, 19:3 (2008), 597–615.

[3] A. Okhotin, "Conjunctive grammars", *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), 519–535.

[4] A. Okhotin, C. Reitwießner, "Conjunctive grammars with restricted disjunction", Technical Report No. 447, Institut für Informatik, Universität Würzburg, September 2008.

On Stateless Restarting Automata*

Friedrich Otto

Fachbereich Elektrotechnik/Informatik, Universität Kassel
34109 Kassel, Germany
otto@theory.informatik.uni-kassel.de

Almost all classical types of automata like Turing machines, pushdown automata, stack automata, etc., can be seen as extensions of finite automata, that is, one of their main ingredients is a finite-state control. On the other hand, the so-called P systems introduced by Păun as an unconventional computing model realizing membrane computing are stateless, because it is difficult and even unrealistic to maintain a global state for a massively parallel group of objects appearing in natural phenomena of cell evolution and chemical reactions. Accordingly, it is of interest to study stateless variants of classical models of automata to obtain insight into the real importance of the finite-state control for these devices.

A finite automaton without states (or rather, with a single state only) accepts all words that it can read completely. Thus, the language it accepts is either empty or of the form Σ^*, where Σ is a subset of the input alphabet. For a stateless pushdown automaton it only makes sense to consider acceptance by empty pushdown. It is well-known that the class of languages that are accepted by stateless pushdown automata coincides with the class CFL of context-free languages. Thus, in this case the resource 'pushdown store' can compensate for the loss of states. On the other hand, for deterministic pushdown automata accepting by empty pushdown, the expressive power is known to increase strictly with the number of states [1]. In particular, a language is accepted by a stateless deterministic pushdown automaton if and only if it is a so-called *simple language*, that is, it is generated by a context-free grammar of a very restricted form. So in this case the resource 'pushdown store' cannot compensate for the loss of states.

Recently Ibarra and his co-workers have started the investigation of stateless multihead finite automata and stateless multicounter systems [6].

*This is joint work with Martin Kutrib (Giessen) and Hartmut Messerschmidt (Kassel). Some of the results have been presented at AFL 2008 [4].

In particular, they have studied the language accepting power of such devices. Among other results they established infinite strict hierarchies depending on the number of heads [2]. Moreover, they have shown that there is no fixed integer k such that every finite unary language can be accepted by a stateless two-way nondeterministic finite automaton with k heads. Hence, the additional resource 'heads' cannot compensate for the loss of states. So, from this point of view, it is a natural and interesting question of how other additional resources given to finite automata relate to the absence or presence of states. Given a particular computational model, are states necessary at all?

Here we investigate the expressive power of stateless restarting automata. The restarting automaton was introduced by Jancar et. al. [3] as a formal tool to model the analysis by reduction, which is a technique used in linguistics to analyze sentences of natural languages. Many restricted types of restarting automata have been studied and put into correspondence to more classical types of formal languages (see, e.g., [5] for a recent survey). In particular, it has been shown that the classes DCFL, CFL, CRL, and GCSL are all characterized by certain types of restarting automata.

We introduce and study stateless variants of various types of restarting automata. How is their expressive power related to that of restarting automata of the same type with states, and what are the closure properties of the language classes characterized by the various types of stateless restarting automata?

For those types of restarting automata that perform a restart in combination with every rewrite step (the so-called RWW-*automata*) the definition of a stateless variant is straightforward [4]. However, for those types of restarting automata that may continue to scan their tape after executing a rewrite step (the so-called RRWW-*automata*) the situation is more involved. We consider two different stateless variants of RRWW-automata: for the first type we simply interpret a second attempt to execute a rewrite step within a cycle as a reject step, while for the second type we distinguish in each cycle between two phases: the phase *before* and the phase *after* the rewrite step. Accordingly, the latter are called *stateless two-phase restarting automata*. We study the expressive power of the various types of stateless restarting automata and present some (non-) closure results for language families that are specified by various deterministic types of stateless restarting automata.

References

[1] M. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, MA, 1978.

[2] O.H. Ibarra, J. Karhumäki, and A. Okhotin. On stateless multihead automata: hierarchies and the emptiness problem. *TUCS Technical Report 848*, Turku Centre for Computer Science, 2007.

[3] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. Restarting automata. In: H. Reichel (ed.), *FCT 1995, Proc.*, *Lect. Notes Comput. Sci. 965*, Springer, Berlin, 1995, 283–292.

[4] M. Kutrib, H. Messerschmidt, and F. Otto. On stateless two-pushdown automata and restarting automata. In: E. Csuhaj-Varjú and Z. Ésik (eds.), *Automata and Formal Languages, AFL 2008, Proc.*, Computer and Automation Research Institute, Hungarian Academy of Sciences, Budapest, 2008, 257–268.

[5] F. Otto. Restarting automata. In: Z. Ésik, C. Martin-Vide, and V. Mitrana (eds.), *Recent Advances in Formal Languages and Applications*, Studies in Computational Intelligence, Vol. 25, Springer, Berlin, 2006, 269–303.

[6] L. Yang, Z. Dang, and O. Ibarra. On stateless automata and P systems. In: *Intern. Workshop "Automata for Cellular and Molecular Computing," Proc.*, MTA SZTAKI, Budapest, 2007, 144–157.

Formal Translations, Parallel Communicating Grammar Systems and Restarting Automata*

Dana Pardubská

Department of Computer Science, Comenius University
842 48 Bratislava, Slovak Republic
pardubska@dcs.fmph.uniba.sk

Martin Plátek

Department of Computer Science, Charles University
11800 Prague, Czech Republic
martin.platek@mff.cuni.cz

1 Introduction

The Functional Generative Description (FGD) for the Czech language developed in Prague (see, e.g., [3]) is based on the notion of the formal translation, on the syntactico-semantic notion of valence, and on the method of analysis by reduction. The main goal of this text is to illustrate the analysis by reduction, and the above mentioned notions in a formal way. We implement this method on formal translations given by Parallel Communicating Grammar Systems (PCGS, [1, 6]). In particular, we use a special type of Freely Rewriting Restarting Automata (FRR)) for this implementation in a similar way as in [3] for FGD.

Thanks to the modelling of the relations of ambiguity and synonymy of Czech sentences FGD describes the formal translation from Czech sentences into their (formal) meanings and vice versa. Here, our effort should support

*Partially supported by the Slovak Grant Agency for Science (VEGA) under contract "Theory of Models, Complexity and Algorithms", by the Grant Agency of the Czech Republic under Grant-No. 405/08/0681, and by the program Information Society under project 1ET100300517.

the understanding of the (formal) complexity of the Functional Generative Description, and the systems with similar ambitions as well.

FRR-automata for formal translation work on so-called *characteristic languages*, that is, on languages with auxiliary symbols (categories) included in addition to the input symbols and output symbols. The *input language* is obtained from a characteristic language by removing all auxiliary and output symbols from its sentences. similarly, the *output language* is obtained from the characteristic language by removing all auxiliary and input symbols from its sentences. By requiring that the automata considered are *linearized* we restrict the number of auxiliary symbols allowed on the tape by a function linear in the number of terminals on the tape. We mainly focus on deterministic restarting automata in order to ensure the *correctness preserving property* for the analysis, i.e., after any restart within an accepting computation the content of the tape is a word from the characteristic language, and after any restart within a non-accepting computation the content of the tape is a word from the complement of the characteristic language. In fact, we mainly consider *strongly linearized* restarting automata. This additional restriction requires that all rewrite operations must be deletions.

Parallel Communicating Grammar Systems (PCGS) handle the creation of copies of generated strings and their regular mappings in a natural way. This ability has a strong similarity to the generation of valences in the Czech language (and some other natural languages). For our purposes, the most interesting property of PCGS is that they give a very natural way how to define the formal translations.

2 Basic Notions

Informally, a PCGS of degree m consists of m separate grammars—in our case *regular grammars*—generating synchronously words in derivations starting from their own axiom. The cooperation is based on some communications realized in so called *communication steps*. In one communication step one grammar G may via special nonterminals require a string generated until now by another grammar G' of PCGS. After receiving it, G includes this string in its own string and G' continues the derivation from its axiom.

Definition 1 PCGS of degree m, $m \geq 1$, *is an* $(m+3)$*-tuple* $\Pi = (G_1, \ldots, G_m, N, T, K)$, *where*

- *for all* $i \in \{1, \ldots, m\}$, $G_i{=}(N_i, T, S_i, P_i)$ *are regular grammars called* component grammars

- T, N, $N = \bigcup_{i=1}^{m} N_i$, *are the set of terminals and nonterminals respectively;* $N_i \cap T = \emptyset$
- $K \subseteq \{Q_1, \ldots, Q_m\} \bigcap N$ *is a set of special symbols, called* communication symbols.

A configuration *is an* m*-tuple* $C = (x_1, \ldots, x_m)$, $x_i = \alpha_i A_i$, $\alpha_i \in T^*, A_i \in (N_i \cup \varepsilon)$*; we call* x_i *the* i-th component of the configuration *(resp. component). We say a* configuration $X = (x_1, \ldots, x_m)$ directly derives a configuration $Y = (y_1, \ldots, y_m)$ *and write* $X \Rightarrow Y$*, if* Y *is derived from* X *by one* generative *or* communication *step:*

Let $X = (x_1, \ldots, x_m)$, $x_i = \alpha_i A_i$, $\alpha_i \in T^*, A_i \in (N_i \cup \varepsilon)$*:*

1. *(generative step) if* $A_i \notin K$ *for all* i, $1 \leq i \leq m$, *then*
$$x_i \overset{G_i}{\Rightarrow} y_i \text{ for } x_i \in T^* N_i,$$
$$y_i = x_i \text{ for } x_i \in T^+;$$
2. *(communication step) if* $A_i \in K$ *for some* i, $1 \leq i \leq m$, *then for each* k *such that* $x_k = z_k Q_{j_k}$, $z_k \in T^*, Q_{j_k} \in K$, *the following holds:*
   (a) *If* $A_{j_k} \notin K$, *then* $y_k = z_k x_{j_k}$ *and* $y_{j_k} = S_{j_k}$.
   (b) *If* $A_{j_k} \in K$, *then* $y_k = x_k$.

   For all remaining indices t*, for which* x_t *does not contain a communication symbol and* Q_t *does not occur in any of* x_i*'s, we put* $y_t = x_t$.

A derivation *of a* PCGS Π *is a sequence of configurations* $D = C_1, C_2, \ldots, C_t$*, where* $C_i \Rightarrow C_{i+1}$ *in* Π.

The language *generated by a* PCGS Π *is the set of terminal words that are generated by* Π *in its first component grammar*

$$L(\Pi) = \{w \in T \mid (S-1, \ldots, S_m) \overset{*}{\Rightarrow} (w, \alpha_2, \ldots, \alpha_m)\}.$$

Here, we use PCGS to model formal translations. Therefore, we will associate with a PCGS Π two (different) languages—*input*, *output* and *translation*. The *input*[*output*] *language* $L_i(\Pi)[L_o(\Pi)]$ generated by a PCGS Π is the set of terminal words generated by $G_1(G_2)$ (in cooperation with the other grammars). Finally, *translation* $T(\Pi)$ generated by a PCGS Π is the set of ordered pairs (w_1, w_2) of terminal words generated at the same time by G_1 and G_2, respectively. Obviously, indices $1, 2$ in G_1, G_2 are not important, we will use denotation G_{in}, G_{out} instead. Analogously, we use indices in, out whenever the same notion has the input and output variant.

Now, let us introduce the complexity measures. The *communication complexity* measure introduced in [2, 7] is the number of exchanged messages

(strings) during the generation procedure. This complexity measure is clearly a computational complexity measure which may be considered as a function of the length of the generated word. Unlike the general case, we are interested in a *constant communication complexity* where the constant does not depend on the length of the generated word but depends on a PCGS instead.

To introduce new complexity measures, let $D = D(w_{in}, w_{out}) = C_0, C_1, \ldots, C_t$ be a derivation of (w_{in}, w_{out}) by Π; $D, D(w_{in}, w_{out})$, Π, and (w_i, w_o) are fixed in what follows. In fact, the derivation is a sequence of generative and communication steps; we will call a non-empty sequence of generative steps between two consecutive communication steps in D[1], resp. before the first and/or after the last communication steps in D a *generative section.*

The *degree of generation* $DG(D)$ is the number of generative sections of D.

Denote by $g(i,j)$ $(g(i,j,D))$ the terminal part generated by G_i within the j-th generative section of D; we call it the (i,j)-(generative) factor (of D). Obviously, w_{in}, w_{out} have risen by a concatenation of some $g(i,j)$'s:

$$w_{in} = g(i_1,j_1)g(i_2,j_2)\ldots g(i_r,j_r),\ w_{out} = g(i'_1,j'_1)g(i'_2,j'_2)\ldots g(i'_{r'},j'_{r'})$$

We will denote by $n_{in}(i,j)$ $(n_{in}(i,j,D)$ the number of occurrences of $g(i,j)$ in w_{in}; analogously for w_{out}. Define $N_{in}(j,D) = \sum_i n_{in}(i,j,D)$, where sum is taken over such i for which $g(i,j)$ is a part of w_{in}; $N_{out}(j,D)$ is defined analogously.

The *degree of input distribution* $DD_{in}(D(w_{in}, w_{out}))$ of $D(w_{in}, w_{out})$ is the maximum over all (defined) $N_{in}(j, (w_{in}, w_{out}))$. The *degree of output distribution* $DD_{out}(D(w_{in}, w_{out}))$ of $D(w_{in}, w_{out})$ is defined analogously.

Now we are ready to introduce the notions of *(input and output) distribution complexity.* First, the input distribution complexity of a derivation D (denoted $DD_{in}(D)$) is the degree of distribution introduced above.

Then the input distribution complexity of the input language and the associated complexity class are defined in the usual way (always considering the corresponding maximum): input distribution complexity of a derivation $\rightsquigarrow$ input distribution complexity of the input language L_{in} (denoted $DD_{in}(L_{in})$) as a function of the length of the word $\rightsquigarrow f(n) - DD_{in}$ as the class of the input languages whose distribution complexity is bounded by $f(n)$. Analogously for the output distribution complexity.

[1]Note that if some communication cut contains more than one communication symbol, then there might be no generative step between two communication steps.

Realize that when a communication complexity of a PCGS Π is bounded by a constant depending on Π only then so are the degree of generation and the (input and output) distribution complexities. For relevant observations about the derivations of PCGS with constant communication complexity consult [5].

A *restarting automaton M of type* t-SLnRR, abbreviated as t-SLnRR-automaton (see more [4]), is a deterministic restarting automaton. Any computation of M consists of certain phases. A phase, called a *cycle*, starts in a (re)starting configuration. The window is shifted along the tape by move-right and rewrite operations until a restart operation is performed and thus a new restarting configuration is reached. If no further restart operation is performed, the computation necessarily finishes in a halting configuration – such a phase is called a *tail*. It is required that in each cycle M performs at least one rewrite step, and at most t rewrite steps. As each rewrite step is in fact a delete step, we see that each cycle reduces the length of the tape.

A t-SLnRR-automaton M determined for formal translation work on so-called *characteristic language*, that is, on language with auxiliary symbols (categories) included in addition to the input symbols and output symbols. The *input language* is obtained from a characteristic language by removing all auxiliary and output symbols from its sentences. Similarly, the *output language* is obtained from the characteristic language by removing all auxiliary and input symbols from its sentences. By requiring that the automata considered are *linearized* we restrict the number of auxiliary symbols allowed on the tape by a function linear in the number of input and output symbols on the tape.

3 Result and conjectures

By a similar simulation as in [4] we obtain the basic result stated that a translation generated by a PCGS Π with constant input and output distribution complexity can be analyzed (by reduction) by a t-SLnRR-automaton M, where t is a constant:

Theorem 1 *Let $i, o \in \mathbb{N}$ be constants. Then to every translation* PCGS Π *of degree $m > 1$ with the input distribution complexity i, and with the output distribution complexity o there is a* t-SLnRR*-automaton M such that $T(\Pi) = T(M)$, and $t \leq i + o$.*

Let (i, o)-TDG denotes the class of formal translations given by PCGS with the input input distribution i and with the output distribution o. Let

t-TSLnRR means the class of formal translations given by t-SLnRR-automata. We believe that the following conjecture hold.

Conjecture 1 *For all* $i, o \geq 1$ *the following* proper *inclusions hold:*

(a) (i,o)-TDG $\subset$ $(i+1,o)$-TDG, (b) (i,o)-TDG $\subset$ $(i,o+1)$-TDG,

(c) t-TSLnRR $\subset$ t+1-TSLnRR, (d) (i,o)-TDG $\subset$ t-TSLnRR, *where* $t = i + o$.

References

[1] E.Czuhaj-Varjú, J. Dassow, J. Kelemen and G.Paun (eds.): Grammar Systems: A Gramatical Approach to Distribution and Cooperation. Gordon and Breach Science Publishers, 1994, ISBN 2-88124-957-4

[2] J. Hromkovič, J. Kari, L. Kari. Some hierarchies for the communication complexity measures of cooperating grammar systems. *Theoretical Computer Science 127*(1994), pp. 123-147.

[3] M. Lopatková, M. Plátek, and P. Sgall. Towards a formal model for functional generative description: Analysis by reduction and restarting automata. *The Prague Bulletin of Mathematical Linguistics* 87 (2007) 7–26.

[4] Dana Pardubská, Martin Plátek, and Friedrich Otto. On the Correspondence between Parallel Communicating Grammar Systems and Restarting Automata. Technical Report of FMFI UK Bratislava, TR-2008-015, 2008, Bratislava. http://kedrigern.dcs.fmph.uniba.sk/reports/

[5] D. Pardubská. Communication complexity hierarchy of parallel communicating grammar system. In: *Developments in Theoretical Computer Science*. Yverdon: Gordon and Breach Science Publishers, 1994. - 115–122. - ISBN 2-88124-961-2.

[6] Gh. Păun and L. Santean. Parallel communicating grammar systems: the regular case. *Ann. Univ. Buc. Ser. Mat.-Inform.* 37 vol.2 (1989) 55–63.

[7] L. Santean. Parallel Communicating Systems. *EATCS Bulletin 42*(1990), pp.160-171

Topological Language Operators

Tobias Richter

tobias.richter@gmx.li

Ludwig Staiger

Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg
von-Seckendorff-Platz 1, 06099 Halle (Saale), Germany
staiger@informatik.uni-halle.de

1 Introduction

Topological methods are useful in the theory of ω-languages in connection with proving hierarchy results (e.g. [Th90, St97]). To this end one considers, for a finite alphabet X, the set of all infinite words (ω-words) over X as the infinite product space of the discrete space X.

To study infinite words as limits of finite words it seems to be providing to include both into the same space. This was done by Boasson and Nivat [BN80]. Redziejowski [Re86] observed that the limit considered in [BN80] is different from that one used in the theory of ω-automata. Therefore he proposed another topology including finite and infinite words into one space.

Both concepts seem to have several drawbacks when considering topology in connection with acceptance of ω-words. Redziejowski's topology has all sets consisting of only infinite words (ω-languages) as closed sets, thus providing no information on the difficulty of acceptance by topological means.

The topology considered by Boasson and Nivat is closely related to the product topology of X^ω, for its restriction to X^ω is nothing else that the product topology. However, all finite word languages are open in this topology, and, moreover, each finite word is an isolated point in this topology.

In this note we start from the notion of δ-limit used in the theory of ω-automata and present topologies on the space X^* of all finite words which can be carried over to the product space X^ω. Because of the countability of X^* we cannot expect to obtain a full topological correspondence between the spaces X^* and X^ω via a limit operation. In fact, the δ-limit does not

go beyond the class $\mathbf{G}_\delta$ of the Borel hierarchy in X^ω but provides a correspondence between the open, closed and also the (σ,δ)-subsets of X^* and the open, closed and $\mathbf{F}_\sigma \cap \mathbf{G}_\delta$-subsets of X^ω.

2 Topology

2.1 General topology

A topology $\mathcal{T} = (\mathcal{X}, \mathcal{O})$ on a set (space) $\mathcal{X}$ is given by a family of open sets $\mathcal{O} \subseteq 2^{\mathcal{X}}$. Here $\mathcal{O}$ is a family of subsets of $\mathcal{X}$ containing $\mathcal{X}$ and closed under arbitrary (including empty) union and finite intersection. there are also other possibilities to specify a topology on $\mathcal{X}$.

One can give a topology $\mathcal{T}$ by a sub-base $\mathcal{B}$ of open sets, that is, by an arbitrary family $\mathcal{B} \subseteq 2^{\mathcal{X}}$. Then

$$\Big\{\bigcup_{i\in I}\bigcap_{j=1}^{n_i} B_{ij} : I \subseteq \mathcal{X} \wedge n_i \in \mathbb{N} \wedge B_{ij} \in \mathcal{B}\Big\}$$

is the family of open sets.

Following Kuratowski's definition [Ku67] we define the family of closed sets (complements of open sets) by a closure operator. A mapping $h : \mathcal{X} \to \mathcal{X}$ is called a *topological closure operator* provided it satisfies the following conditions.

$$\begin{aligned} h(\emptyset) &= \emptyset && (1)\\ h(M) &\supseteq M && (2)\\ h(M_1 \cup M_2) &= h(M_1) \cup (M_2) \text{ and} && (3)\\ h(M) &= h(h(M)) && (4) \end{aligned}$$

2.2 $\mathring{A}$-topology

We start with a first well-known topology in X^* which resembles the product topology on X^ω when we consider the set of open subsets.

This topology has the base $\{w \cdot X^* : w \in X^*\}$, and its open subsets are of the form $W \cdot X^*$ where $W \subseteq X^*$. As it is easily verified the closure operator defining this topology is the initial word operator $\mathring{A}$ assigning to each language $L \subseteq X^*$ its set of prefixes $\mathring{A}(L) := \{w : \exists v(v \in L \wedge w \sqsubseteq v)\}$.

Moreover, as $w \cdot X^* = \{v : v \in X^* \wedge w \sqsubseteq v\}$, this topology is the right topology on X^* derived from the prefix ordering $\sqsubseteq$ on X^*.

3 The Operator Anf

Prodinger and Urbanek [PU79] defined a generalisation of the initial word operator $\mathring{A}$ as follows. Let $\mathcal{L} \subseteq 2^{X^*}$ be a family of languages and define, for $W \subseteq X^*$, the *left derivative* of the language W by the word w as usual as $W/w := \{v : v \in X^* \wedge w \cdot v \in W\}$. Then $\mathsf{Anf}_{\mathcal{L}}(W) := \{w : w \in X^* \wedge W/w \in \mathcal{L}\}$.

Several examples of $\mathsf{Anf}_{\mathcal{L}}$-operators are known.

Example 1

1. If $\mathcal{L}_{\mathring{A}} = \{L : L \subseteq X^* \wedge L \neq \emptyset\}$ then $\mathsf{Anf}_{\mathcal{L}_{\mathring{A}}} = \mathring{A}$.
2. If $\mathcal{L}_{\text{center}} = \{L : L \subseteq X^* \wedge L \text{ is infinite}\}$ then $\mathsf{Anf}_{\mathcal{L}_{\text{center}}} = \mathsf{center}$ is the center-operator [PU79, BN81, SN86].
3. If $\mathcal{L}_{\text{sctr}} = \{L : L \subseteq X^* \wedge L \text{ contains an infinite } \sqsubseteq\text{-chain}\}$ then $\mathsf{Anf}_{\mathcal{L}_{\text{sctr}}} = \mathsf{sctr}$ is the supercenter-operator [SN86].

3.1 General properties of $\mathsf{Anf}_{\mathcal{L}}$

The operator $\mathsf{Anf}_{\mathcal{L}}$ has the following properties (see [PU79, Pr80]):

Property 1

1. $\mathsf{Anf}_{\mathcal{L}}(W/w) = \mathsf{Anf}_{\mathcal{L}}(W)/w$
2. $\mathsf{Anf}_{\mathcal{L}}(\emptyset) = \emptyset$ *if and only if* $\emptyset \notin \mathcal{L}$
3. $\mathsf{Anf}_{\mathcal{L}}$ *is monotone if and only if* $W \in \mathcal{L}$ *and* $W \subseteq V$ *imply* $V \in \mathcal{L}$.
4. $\mathsf{Anf}_{\mathcal{L}}$ *is* $\cup$*-stable if and only if* $\mathsf{Anf}_{\mathcal{L}}$ *is monotone and* $W \cup V \in \mathcal{L}$ *implies* $W \in \mathcal{L}$ *or* $V \in \mathcal{L}$.
5. *It holds* $\mathsf{Anf}_{\mathcal{L}}(W) = W$*, for all* $W \subseteq X^*$*, if and only if* $\mathcal{L} \supseteq \{V : V \subseteq X^* \wedge e \in V\}$.[1]

As a consequence of Property 1 we have requirements under which the operator $\mathsf{Anf}_{\mathcal{L}}$ satisfies the conditions of Eqs. (1), (2) and (3) of a topological closure.

The following theorem from [Ri07] gives a relation for the superposition of Anf-operators.

Theorem 2 *Let* $\mathcal{J}, \mathcal{K}, \mathcal{L}, \mathcal{M}$ *be families of subsets of* X^**. Then the following conditions are equivalent.*

1. $\mathsf{Anf}_{\mathcal{K}}(\mathsf{Anf}_{\mathcal{L}}(W)) \subseteq \mathsf{Anf}_{\mathcal{J}}(\mathsf{Anf}_{\mathcal{M}}(W))$ *for all* $W \subseteq X^*$*, and*

[1] e is the empty word.

2. $\mathsf{Anf}_{\mathcal{L}}(V) \in \mathcal{K}$ *implies* $\mathsf{Anf}_{\mathcal{M}}(V) \in \mathcal{J}$ *for all* $V \subseteq X^*$.

The theorem yields several consequences.

Corollary 3 *1. We have* $\mathsf{Anf}_{\mathcal{K}} \circ \mathsf{Anf}_{\mathcal{L}} = \mathsf{Anf}_{\mathcal{J}}$ *if and only if* $\forall V(V \subseteq X^* \to (\mathsf{Anf}_{\mathcal{L}}(V) \in \mathcal{K} \leftrightarrow V \in \mathcal{J}))$.

2. The operator $\mathsf{Anf}_{\mathcal{L}}$ *is idempotent, that is,* $\mathsf{Anf}_{\mathcal{L}} \circ \mathsf{Anf}_{\mathcal{L}} = \mathsf{Anf}_{\mathcal{L}}$ *if and only if* $\forall V(V \subseteq X^* \to (\mathsf{Anf}_{\mathcal{L}}(V) \in \mathcal{L} \leftrightarrow V \in \mathcal{L})))$

3.2 Closure operators

Corollary 3.*2* yields the fourth condition under which an $\mathsf{Anf}_{\mathcal{L}}$-operator is a topological closure operator. Thus we obtain the following.

Theorem 4 ([Pr80]) *A mapping* $\mathsf{Anf}_{\mathcal{L}}$ *is a topological closure on* X^* *if and only if the following conditions are satisfied.*

1. $\mathcal{L} \neq \emptyset$, $\emptyset \notin \mathcal{L}$,

2. $W \in \mathcal{L}$ *and* $W \subseteq V$ *imply* $V \in \mathcal{L}$,

3. $W \cup V \in \mathcal{L}$ *implies* $W \in \mathcal{L}$ *or* $V \in \mathcal{L}$,

4. if $e \in W$ *then* $W \in \mathcal{L}$, *and*

5. $W \in \mathcal{L}$ *if and only if* $\mathsf{Anf}_{\mathcal{L}}(W) \in \mathcal{L}$.

Now, Corollary 3.*1* implies a relation to the init-operator $\mathring{A}$.

Corollary 5 *If* $\mathsf{Anf}_{\mathcal{L}}$ *is a topological closure on* X^* *then* $\mathsf{Anf}_{\mathcal{L}}(W) \subseteq \mathsf{Anf}_{\mathcal{L}}(\mathring{A}(W)) = \mathring{A}(W)$, *for all* $W \subseteq X^*$.

4 Joint Topologies

In this section we relate some of the topologies generated by closures of the form $\mathsf{Anf}_{\mathcal{L}}$ to the usual CANTOR space X^ω of infinite words (cf. [St97, Section 1.4]). To this end ewe define

Definition 1 The *δ-limit* of a language $W \subseteq X^*$ is the set $W^\delta := \{\xi : \xi \in X^\omega \wedge \mathring{A}(\xi) \cap W \text{ is infinite}\}$.

It is well-known that $\mathring{A}(F)^\delta$ is the closure $\mathcal{C}(F)$ of the set $F \subseteq X^\omega$ in CANTOR space, and that $(W \cdot X^*)^\delta = W \cdot X^\omega$. Thus, in view of Corollary 5, for every closure operator of the form $\mathsf{Anf}_\mathcal{L}$, any closed subset $F \subseteq X^\omega$ has the closed pre-image $\mathring{A}(F)$ and every open subset $W \cdot X^\omega \subseteq X^\omega$ has the open pre-image $W \cdot X^*$ under δ-limit.

Now, our approach gives rise to the following definition.

Definition 2 A closure operator $\mathsf{Anf}_\mathcal{L}$ is *compatible* w.r.t. the δ-limit with the CANTOR topology of X^ω provided, for every $W \subseteq X^*$, the set $\mathsf{Anf}_\mathcal{L}(W)^\delta$ is closed and the set $(X^* \setminus \mathsf{Anf}_\mathcal{L}(W))^\delta$ is open in CANTOR space.

The first example of a closure $\mathsf{Anf}_\mathcal{L}$ compatible with the CANTOR topology is the init-operator $\mathring{A} = \mathsf{Anf}_{\mathcal{L}_{\overset{\circ}{A}}}$. It generates, however, only a T_0-topology on X^*, that is, a topology in which not every finite set is closed.

This drawback can be circumvented using closures related to $\mathcal{L}_{\text{center}}$ and $\mathcal{L}_{\text{sctr}}$, that is, the closures $\mathsf{Anf}_{\mathcal{L}_{\text{center}} \cup \mathcal{L}_0}$ and $\mathsf{Anf}_{\mathcal{L}_{\text{sctr}} \cup \mathcal{L}_0}$ where $\mathcal{L}_0 := \{W : W \subseteq X^* \wedge e \in W\}$.

Lemma 6 *If $\mathcal{L} = \mathcal{L}_{\text{center}} \cup \mathcal{L}_0$ or $\mathcal{L} = \mathcal{L}_{\text{sctr}} \cup \mathcal{L}_0$ then $\mathsf{Anf}_\mathcal{L}(W) = W$, for every finite subset $W \subseteq X^*$.*

Moreover, it holds the following.

Theorem 7 *1. $\mathsf{Anf}_{\mathcal{L}_{\text{center}} \cup \mathcal{L}_0}(W) = \{\xi : \xi \in X^\omega \wedge \mathring{A}(\xi) \subseteq \mathring{A}(W)\}$ is the adherence[2] of the language W.*

2. $\mathsf{Anf}_{\mathcal{L}_{\text{sctr}} \cup \mathcal{L}_0}(W) = \mathcal{C}(W^\delta)$ is the closure (in CANTOR space) of the δ-limit of the language W.

Both of these closures have the property that $\mathsf{Anf}_{\mathcal{L}_{\text{center}}}(W)^\delta, \mathsf{Anf}_{\mathcal{L}_{\text{sctr}}}(W)^\delta \supseteq W^\delta$. It turns out, however, that this need not be true for all closures $\mathsf{Anf}_\mathcal{L}$ compatible with the CANTOR topology (see [Ri07]).

Theorem 8 *Define $\mathcal{L} := \{W : e \in W \vee W^\delta$ is infinite$\}$. Then $\mathsf{Anf}_\mathcal{L}$ is a closure on X^* compatible with the CANTOR topology.*

Moreover, for $\mathcal{L}_\infty := \{W : W^\delta$ is infinite$\}$, it holds $\mathsf{Anf}_{\mathcal{L}_\infty}(W)^\delta = \emptyset$ whenever W^δ is finite.

[2]The adherence or **ls**-limit of a language is defined in [LS77] or [BN80].

References

[BN80] L. Boasson and M. Nivat: Adherences of languages. *J. Comput. System Sci.* **20** (1980), 285–309.

[BN81] L. Boasson and M. Nivat: Centers of languages. In P. Deussen (editor): *Theoretical Comput. Sci.*, Lect. Notes Comput. Sci. **104**, 245–251, Springer-Verlag, Berlin, 1981.

[Ku67] K. Kuratowski: *Topology* I. Academic Press, 1967.

[LS77] R. Lindner and L. Staiger: *Algebraische Codierungstheorie; Theorie der sequentiellen Codierungen.* Akademie-Verlag, Berlin, 1977.

[Pr80] H. Prodinger: Topologies on free monoids induced by closure operators of a special type. *RAIRO Inform. Théor.* **14** (1980), 225–237.

[Pr83] H. Prodinger: Topologies on free monoids induced by families of languages. *RAIRO Inform. Théor.* **17** (1983), 285–290.

[PU79] H. Prodinger and F. J.Urbanrek: Language operators related to Init. *Theoret. Comput. Sci.* **8** (1979), 179–199.

[Re86] R. R. Redziejowski: Infinite word languages and continuous mappings. *Theoret. Comput. Sci.* **43** (1986), 59–79.

[Ri07] T. Richter: Lokalisierung in formalen Sprachen. Diplomarbeit, Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg, 2007.

[RS97] G. Rozenberg, A. Salomaa (editors): *Handbook of Formal Languages.* Springer-Verlag, Berlin, 1997.

[St97] L. Staiger: ω-languages. In Rozenberg and Salomaa [RS97],Vol. 3, 339–387.

[SN86] L. Staiger and W. Nehrlich: The centers of context-sensitive languages. In J. Gruska, B. Rovan and J. Wiedermann(editors): *Proc. MFCS'86*, Lect. Notes Comput. Sci. **233**, 594–601, Springer-Verlag, Berlin, 1986.

[Th90] W. Thomas: Automata on infinite objects. In J. van Leeuwen (editor): *Handbook of Theoretical Computer Science*, Vol. B. 135–191. North-Holland, Amsterdam, 1990.

M. Holzer, M. Kutrib, A. Malcher (Hrsg.):
18. Theorietag *Automaten und Formale Sprachen* 2008

Eindeutige löschende Homomorphismen in freien Monoiden

Johannes C. Schneider

Fachbereich Informatik, Technische Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern, Germany
jschneider@informatik.uni-kl.de

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit der grundlegenden kombinatorischen Fragestellung, ob zu einer gegeben Zeichenkette α ein Homomorphismus σ existiert, der eindeutig ist, es also keinen anderen Homomorphismus τ gibt mit $\tau(\alpha) = \sigma(\alpha)$. Während Freydenberger et al. (*Internat. Journal of Comp. Science* 17, 2006) diejenigen Zeichenketten charakterisieren, für die ein eindeutiger *nichtlöschender* Homomorphismus existiert, ist nur wenig bekannt über die Eindeutigkeit von *löschenden* Homomorphismen, also solche Homomorphismen, die gewisse Zeichen auf das leere Wort abbilden. Wir zeigen, dass – im Unterschied zum Hauptresultat von Freydenberger et al. – die Frage der Eindeutigkeit von löschenden Homomorphismen sehr stark von der Größe des Bildalphabets des Homomorphismus abhängen kann und stellen noch einige weitere Ergebnisse vor.

1 Einleitung

Diese Arbeit führt die in Freydenberger, Reidenbach und Schneider [1] begonnene explizite Untersuchung der Eindeutigkeit von Homomorphismen fort. Dabei betrachten wir Homomorphismen, die Zeichenketten über dem unendlichen Alphabet $\mathbb{N}$, im Folgenden als *Pattern* bezeichnet, auf Zeichenketten über einem Zielalphabet Σ abbilden. Für ein Pattern α bezeichnen wir mit $\mathrm{var}(\alpha)$ die Menge der Zeichen, die im Pattern vorkommen (*Variablen* genannt). Bei gegebenem Pattern $\alpha \in \mathbb{N}^+$ nennen wir einen Homomorphismus $\sigma : \mathbb{N}^* \to \Sigma^*$ mit $\sigma(\alpha) \neq \varepsilon$ genau dann *eindeutig (bezüglich α)*, wenn es keinen Homomorphismus $\tau : \mathbb{N}^* \to \Sigma^*$ gibt mit $\tau(\alpha) = \sigma(\alpha)$ und $\tau(x) \neq \sigma(x)$ für ein $x \in \mathrm{var}(\alpha)$. Falls σ nicht eindeutig bezüglich α ist, so nennen wir σ *mehrdeutig (bezüglich α)*. So ist z.B. für $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ und $\alpha = 1 \cdot 2 \cdot 2$

der Homomorphismus $\sigma : \mathbb{N}^* \to \Sigma^*$, definiert durch $\sigma(1) := \mathtt{a}$, $\sigma(2) := \mathtt{b}$ nicht eindeutig, also mehrdeutig, denn der Homomorphismus $\tau : \mathbb{N}^* \to \Sigma^*$, gegeben durch $\tau(1) := \mathtt{abb}$, $\tau(2) := \varepsilon$, erfüllt $\tau(\alpha) = \mathtt{abb} = \sigma(\alpha)$ und $\tau(1) \neq \sigma(1)$.

2 Bekanntes über nichtlöschende Homomorphismen

In Freydenberger et al. [1] werden hauptsächlich solche Homomorphismen betrachtet, die jedes Zeichen im Pattern auf ein Wort der Länge 1 oder mehr abbilden, im weiteren Verlauf der Arbeit als *nichtlöschend* bezeichnet. Dabei ist folgende Partition der Menge aller Pattern von großer Bedeutung:

Definition 1 *Sei $\alpha \in \mathbb{N}^+$. Wir nennen α genau dann* prolix*, wenn eine Faktorisierung $\alpha = \beta_0\, \gamma_1\, \beta_1\, \gamma_2\, \beta_2 \ldots \gamma_n\, \beta_n$ mit $n \geq 1$, $\beta_i \in \mathbb{N}^*$, $0 \leq i \leq n$, und $\gamma_i \in \mathbb{N}^+$, $1 \leq i \leq n$, existiert, so dass gilt:*

1. *Für alle $i \in \{1, 2, \ldots, n\}$ ist $|\gamma_i| \geq 2$,*
2. *für alle $i \in \{0, 1, \ldots, n\}$ und für alle $j \in \{1, 2, \ldots, n\}$ ist $\mathrm{var}(\beta_i) \cap \mathrm{var}(\gamma_j) = \emptyset$,*
3. *für alle $i \in \{1, 2, \ldots, n\}$ existiert ein $y_i \in \mathrm{var}(\gamma_i)$, so dass y_i genau einmal in γ_k vorkommt, und, für alle $i' \in \{1, 2, \ldots, n\}$ gilt: Falls $y_i \in \gamma_{i'}$, so $\gamma_i = \gamma_{i'}$.*

Wir nennen $\alpha \in \mathbb{N}^+$ genau dann prägnant*, wenn es nicht prolix ist.*

Diese Definition entspringt der Beschäftigung mit Patternsprachen, bei denen prägnante Pattern die kürzesten Erzeuger ihrer Patternsprache sind. Einen Überblick über Patternsprachen bietet Mateescu and Salomaa [2].

Prägnante und prolixe Pattern spielen aber nicht nur in der Theorie der Patternsprachen eine wichtige Rolle, sondern bilden auch die Menge der Fixpunkte trivialer bzw. nichttrivialer Endomorphismen und entsprechen außerdem den m-(im)primitiven Wörtern (siehe Reidenbach, Schneider [3]).

Bezüglich der Eindeutigkeit von nichtlöschenden Homomorphismen gilt nun folgende Charakterisierung:

Satz 1 (Freydenberger, Reidenbach, Schneider [1]) *Sei $\alpha \in \mathbb{N}^*$ und Σ ein Alphabet, $|\Sigma| \geq 2$. Es existiert genau dann ein eindeutiger nichtlöschender Homomorphismus $\sigma : \mathbb{N}^* \to \Sigma^*$ bzgl. α, wenn α prägnant ist.*

Eine direkte Konsequenz aus dem Satz ist, dass für prolixe Pattern kein eindeutiger nichtlöschender Homomorphismus existiert.

3 Neue Ergebnisse zu löschenden Homomorphismen

In dieser Arbeit beschäftigen wir uns nun erstmals systematisch mit *löschenden* Homomorphismen, also Homomorphismen, die gewisse Zeichen auf das leere Wort ε abbilden, und der Frage, für welche Pattern ein eindeutiger löschender Homomorphismus existiert. Hierzu stellen wir zunächst fest, dass es zum einen prolixe Pattern gibt, bezüglich derer gewisse Homomorphismen eindeutig sind (diese Homomorphismen müssen dann jedoch gewisse Zeichen im Pattern „löschen“, als eine Folgerung aus Satz 1). Zum anderen existieren prolixe Pattern, für die *jeder* Homomorphismus mehrdeutig ist. Wir verdeutlichen dies an den folgenden Beispielpattern, die mittels Definition 1 und geeigneter Wahl der γ_i ($\gamma_1 = \gamma_2 = 1 \cdot 2$) leicht als prolix identifiziert werden können: Sei $\alpha_1 = 1 \cdot 2 \cdot 1 \cdot 2 \cdot 3 \cdot 3 \cdot 3$ (wir benutzen „$\cdot$“ um zwischen einzelnen Variablen im Pattern zu trennen und so z.B. nicht $1 \cdot 2$ und 12 durcheinander zu werfen). Für dieses Pattern ist jeder Homomorphismus eindeutig, der 3 auf ein Wort der Länge 1 abbildet.

Unser nächstes Beispielpattern sei $\alpha_2 = 1{\cdot}2{\cdot}1{\cdot}2{\cdot}3{\cdot}3$. Wir behaupten, dass für dieses Pattern jeder Homomorphismus mehrdeutig ist: Angenommen, es existiert ein eindeutiger Homomorphismus σ mit $\sigma(1) \neq \varepsilon$. Dann widerlegt der Homomorphismus τ, definiert durch $\tau(1) := \varepsilon$, $\tau(2) := \sigma(1 \cdot 2)$, $\tau(x) := x$ für $x \neq 1, 2$, die Eindeutigkeit von σ, da $\tau(\alpha_2) = \sigma(\alpha_2)$ und $\tau(1) \neq \sigma(1)$ gilt. Mit $\tau(1) := \sigma(1 \cdot 2)$, $\tau(2) := \varepsilon$, $\tau(x) := x$ für $x \neq 1, 2$ kann man in gleicher Weise argumentieren, dass ein eindeutiger Homomorphismus auch 2 auf das leere Wort abbilden muss. Falls also σ eindeutig ist, gilt $\sigma(1 \cdot 2) = \varepsilon$. Nehmen wir weiterhin an, dass $\sigma(3) \neq \varepsilon$ gilt: Dann widerlegt der Homomorphismus τ, definiert als $\tau(1) := \sigma(3)$, $\tau(x) := \varepsilon$ für alle $x \neq 3$, die Eindeutigkeit von σ. Also existiert für α_2 kein eindeutiger Homomorphismus.

Die Notwendigkeit eines eindeutigen Homomorphismus, in einem prolixen Pattern gewisse Variablen auf das leere Wort abbilden, kann sich also wie ein „Dominoeffekt“ fortsetzen, was die folgende Definition aufgreift – dabei sei für ein $V \subseteq \mathbb{N}$ der Homomorphismus $\delta_V : \mathbb{N}^* \to \mathbb{N}^*$ definiert durch $\delta_V(x) := x$, falls $x \in V$, $\delta_V(x) := \varepsilon$,falls $x \notin V$:

Definition 2 *Sei $\alpha \in \mathbb{N}^+$. Wir definieren induktiv eine* Mehrdeutigkeitspartition (bezüglich α)*:*

(i) $(\emptyset, \mathrm{var}(\alpha))$ *ist eine Mehrdeutigkeitspartition bezüglich α.*

(ii) *Falls (E, N) eine Mehrdeutigkeitspartition bezüglich α ist und ein Homomorphismus $h : \mathbb{N}^* \to \mathbb{N}^*$ existiert, der $h(i) \neq (i)$ für ein $i \in N$ und $h(\alpha) = \delta_N(\alpha)$ erfüllt, dann ist auch (E', N') eine Mehrdeutigkeitspar-*

tition bezüglich α mit

$$\begin{aligned} E' &:= E \cup \{x \in N \mid h(x) = \varepsilon\}, \\ N' &:= \{x \in N \mid h(x) \neq x\}. \end{aligned}$$

Mittels Punkt (ii) der Definition und den Homomorphismen $h_1, h_2 : \mathbb{N}^* \to \mathbb{N}^*$, gegeben durch $h_1(1) = h_2(2) = 1 \cdot 2$, $h_1(2) = h_2(1) = \varepsilon$ und $h_1(3) = h_2(3) = 3$, können wir leicht verifizieren, dass $(E, N) = (\{1, 2\}, \{3\})$ eine Mehrdeutigkeitspartition bzgl. sowohl α_1 als auch α_2 ist. Ferner können wir formal zeigen, dass Variablen aus E von eindeutigen Homomorphismen auf das leere Wort abgebildet werden müssen:

Satz 2 *Sei Σ ein Alphabet. Sei $\alpha \in \mathbb{N}^+$ und (E, N) eine Mehrdeutigkeitspartition bzgl. α. Dann ist jeder Homomorphismus $\sigma : \mathbb{N}^* \to \Sigma^*$ mit $\sigma(x) \neq \varepsilon$ für ein $x \in E$ mehrdeutig bzgl. α.*

Damit erhält man sofort ein hinreichendes Kriterium dafür, dass zu einem Pattern *kein* eindeutiger Homomorphismus existiert:

Folgerung 1 *Sei Σ ein Alphabet und $\alpha \in \mathbb{N}^+$. Falls $(\mathrm{var}(\alpha), \emptyset)$ eine Mehrdeutigkeitspartition bzgl. α ist, so ist kein Homomorphismus $\sigma : \mathbb{N}^* \to \Sigma^*$ eindeutig bzgl. α.*

Diese Folgerung passt auf das Beispielpattern α_2, da es $(\{1, 2, 3\}, \emptyset)$ als Mehrdeutigkeitspartition besitzt.

Bisher haben wir das Bildalphabet Σ des Homomorphismus nicht weiter spezifiziert. Wenn wir hierfür ein unendliches Alphabet wählen, ist die Bedingung aus Folgerung 1 sogar charakteristisch:

Satz 3 *Sei Σ_∞ ein unendliches Alphabet und sei $\alpha \in \mathbb{N}^+$. Es gibt genau dann keinen eindeutigen Homomorphismus $\sigma : \mathbb{N}^* \to \Sigma_\infty^*$ bzgl. α, wenn $(\mathrm{var}(\alpha), \emptyset)$ eine Mehrdeutigkeitspartition bzgl. α ist.*

Diese Charakterisierung ermöglicht das folgende Resultat über die Komplexität einer Entscheidungsprozedur für $\mathrm{AMB}_{\Sigma_\infty} := \{\alpha \in \mathbb{N}^+ \mid$ es existiert kein eindeutiger Homomorphismus bzgl. $\alpha\}$ für unendliche Alphabete Σ_∞:

Satz 4 *Sei Σ_∞ ein unendliches Alphabet. Das Problem, $\mathrm{AMB}_{\Sigma_\infty}$ zu entscheiden, ist NP-vollständig.*

Im Folgenden wenden wir uns der Betrachtung *endlicher* Alphabete zu. In diesem Fall gilt Satz 3 nicht:

Satz 5 *Sei $k \in \mathbb{N}$ und seien Σ_k, Σ_{k+1} endliche Alphabete mit k bzw. $k+1$ Buchstaben. Es existiert ein Pattern $\alpha \in \mathbb{N}^+$, so dass*

(i) $(\mathrm{var}(\alpha), \emptyset)$ *keine Mehrdeutigkeitspartition bzgl. α ist,*
(ii) *kein Homomorphismus $\sigma : \mathbb{N}^* \to \Sigma_k^*$ ist eindeutig bzgl. α,*
(iii) *ein eindeutiger Homomorphismus $\sigma' : \mathbb{N}^* \to \Sigma_{k+1}^*$ bzgl. α existiert.*

Dies offenbart einen neuartigen und unerwarteten Aspekt in der Mehdeutigkeitsforschung von Homomorphismen. Fragt man nämlich nach der Existenz eines eindeutigen *nichtlöschenden* Homomorphismus für ein Pattern, so hängt die Antwort dieser Frage gemäß Satz 1 lediglich davon ab, ob das Pattern prägnant oder prolix ist (solange mindestens zwei Buchstaben im Zielalphabet des Homomorphismus sind). Betrachtet man hingegen *löschende* Homomorphismen, so spielt die Größe das Zielalphabets des Homomorphismus eine entscheidende Rolle. Man könnte also im letzteren Fall die bisher verfolgte Fragestellung „Gegeben ein Alphabet, welche Pattern lassen sich über diesem Alphabet eindeutig abbilden?“ auch wie folgt umdrehen: Sei $\alpha \in \mathbb{N}^*$ ein Pattern, was ist die minimale Größe von Σ, so dass ein eindeutiger Homomorphismus $\sigma : \mathbb{N}^* \to \Sigma^*$ existiert, falls es einen solchen Homomorphismus überhaupt gibt?

In jedem Fall muss man bei der weiteren Beschäftigung mit eindeutigen löschenden Homomorphismen $\sigma : \mathbb{N}^* \to \Sigma^*$ alphabetspezifische Phänomene in Abhängigkeit von Σ miteinbeziehen.

Literatur

[1] D.D. Freydenberger, D. Reidenbach, und J.C. Schneider. Unambiguous morphic images of strings. *International Journal of Foundations of Computer Science*, 17:601–628, 2006.

[2] A. Mateescu und A. Salomaa. Patterns. In G. Rozenberg and A. Salomaa, Herausgeber, *Handbook of Formal Languages*, Band 1, Kapitel 4.6, Seiten 230–242. Springer, 1997.

[3] D. Reidenbach und J. C. Schneider. Morphically primitive words. In *Proc. 6th International Conference on Words, WORDS 2007*, Seiten 262–272, 2007.

Zu akzeptierenden Netzwerken evolutionärer Prozessoren mit zwei Knotenarten

Bianca Truthe

Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg
Universitätsplatz 2, 39106 Magdeburg, Germany
truthe@iws.cs.uni-magdeburg.de

Zusammenfassung

In dieser Arbeit wird untersucht, welche Sprachen von Netzwerken evolutionärer Prozessoren, bei denen die Anzahl der vorkommenden Knotenarten auf zwei beschränkt ist, akzeptiert werden.

Wir zeigen, dass jede kontextabhängige Sprache von einem Netzwerk mit einem ersetzenden Prozessor und einem Ausgabeknoten akzeptiert wird. Jede rekursiv aufzählbare Sprache wird von einem Netzwerk mit drei evolutionären Prozessoren (einem ersetzenden, einem einfügenden und einem Ausgabe-Prozessor) akzeptiert. Auch Netzwerke mit einfügenden und löschenden Prozessoren aber ohne ersetzende Prozessoren akzeptieren alle rekursiv aufzählbaren Sprachen. Dabei sind zwei einfügende, ein löschender und ein Ausgabe-Prozessor ausreichend.

1. Einleitung

Netzwerke von Sprachprozessoren wurden von E. CSUHAJ-VARJÚ und A. SALOMAA eingeführt ([3]). Solch ein Netzwerk kann als Graph angesehen werden, bei dem jeder Knoten Regeln und Wörter hat, die er entsprechend den Regeln ableitet, und die nach dem Passieren gewisser Filter über die Kanten zu anderen Knoten gelangen. Die von einem Netzwerk erzeugte Sprache besteht aus allen Wörtern, die irgendwann in einem festgelegten Knoten auftreten.

Von Punktmutationen in der Biologie inspiriert, haben J. CASTELLANOS, C. MARTIN-VIDE, V. MITRANA und J. SEMPERE in [2] Netzwerke evolutionärer Prozessoren eingeführt. Dabei sind die verwendeten Regeln Ersetzen eines Buchstabens durch einen anderen, Einfügen eines Buchstabens und Löschen eines Buchstabens. In [6] wurde eine Charakterisierung der Komplexitätsklasse $\mathbb{NP}$ basierend auf akzeptierenden Netzwerken evolutionärer Prozessoren präsentiert. Von

J. DASSOW und V. MITRANA wurden in [4] erstmals akzeptierende Netzwerke evolutionärer Prozessoren untersucht, bei denen die Kommunikation der Prozessoren durch reguläre Filter gesteuert wird.

In [5] wurde die Erzeugungskraft von Netzwerken evolutionärer Prozessoren untersucht, bei denen nur zwei Knoten-Arten vorkommen. Insbesondere wurden folgende Resultate bewiesen:

- Netzwerke ohne löschende Knoten erzeugen alle kontextabhängigen Sprachen; jeweils ein ersetzender und ein einfügender Knoten sind ausreichend.
- Netzwerke ohne ersetzende Knoten erzeugen alle rekursivaufzählbaren Sprachen; jeweils zwei einfügende und ein löschender Knoten sind ausreichend.

In der vorliegenden Arbeit werden die dualen Aussagen bewiesen:

- Jede kontextabhängige Sprache wird von einem Netzwerk evolutionärer Prozessoren mit einem ersetzenden, einem löschenden und einem Ausgabe-Knoten akzeptiert.
- Jede rekursiv aufzählbare Sprache wird von einem Netzwerk evolutionärer Prozessoren mit zwei einfügenden, einem löschenden und einem Ausgabe-Knoten akzeptiert.

Netzwerke, die nur aus ersetzenden Prozessoren bestehen, können ausschließlich endliche Sprachen erzeugen, aber sie können unendliche Sprachen akzeptieren. Der Grund dafür liegt darin, dass erzeugende Netzwerke mit einer endlichen Wortmenge beginnen und ersetzende Prozessoren die Wortlänge nicht vergrößern können, wogegen akzeptierende Netzwerke unendlich viele Eingabewörter erhalten.

Wir zeigen, dass zum Akzeptieren einer jeden kontextabhängigen Sprache ein ersetzender Prozessor und ein Ausgabeknoten genügen. Außerdem weisen wir nach, dass jede rekursiv aufzählbare Sprache von einem Netzwerk mit einem ersetzenden, einem einfügenden und einem Ausgabe-Knoten akzeptiert wird.

Wir geben im Folgenden einige der in dieser Arbeit verwendeten Begriffe und Notationen an. Für weitere Definitionen sei auf die Literatur verwiesen (z. B. [7]).

Zu einem Alphabet V bezeichnen wir mit V^* die Menge aller Wörter über V einschließlich dem Leerwort λ. Eine Grammatik ist ein Quadrupel $G = (N, T, P, S)$ mit einem Alphabet N von Nichtterminalen, einem Alphabet T von Terminalen, einer endlichen und nicht-leeren Menge P von Ersetzungsregeln der Form $\alpha \to \beta$ mit $\alpha \in (N \cup T)^* \setminus T^*$ und $\beta \in (N \cup T)^*$ und einem Startsymbol $S \in N$. Eine Grammatik ist in Kuroda-Normalform, wenn alle ihre Ersetzungsregeln eine der folgenden Formen haben: $AB \to CD$, $A \to CD$, $A \to x$, $A \to \lambda$ mit $A, B, C, D \in N$ und $x \in N \cup T$.

Eine Regel $\alpha \to \beta$ heißt ersetzend, wenn $|\alpha| = |\beta| = 1$ gilt, und löschend, wenn $|\alpha| = 1$ und $\beta = \lambda$ gelten. Wir betrachten Einfügen als Gegenstück zu Löschen, schreiben $\lambda \to a$ für einen Buchstaben a und bezeichnen es ebenfalls als Regel. Das Einfügen $\lambda \to a$ liefert zu einem Wort w ein Wort $w_1 a w_2$ mit $w = w_1 w_2$ für zwei

(möglicherweise leere) Wörter w_1 und w_2. Ersetzende, löschende und einfügende Regeln werden auch Evolutionsregeln genannt.

Wir definieren nun akzeptierende Netzwerke evolutionärer Prozessoren.

Definition 1.1

(i) *Ein akzeptierendes Netzwerk evolutionärer Prozessoren der Größe n ist ein $(n+3)$-Tupel*

$$\mathcal{N}^{(n)} = (U, V, N_1, N_2, \ldots, N_n, E, j, O)$$

mit

- *zwei endlichen Mengen U (dem Eingabe-Alphabet) und V (dem Arbeitsalphabet, $U \subseteq V$),*
- *n Knoten $N_i = (M_i, I_i, O_i)$ (für $1 \leq i \leq n$), wobei*
  - *M_i eine sortenreine Menge von Evolutionsregeln ist sowie*
  - *I_i und O_i reguläre Sprachen über V sind,*
- *einer Teilmenge E von $\{1,2,\ldots,n\} \times \{1,2,\ldots,n\}$,*
- *einer natürlichen Zahl j mit $1 \leq j \leq n$ und*
- *einer nicht-leeren Teilmenge O von $\{1,2,\ldots,n\}$.*

(ii) *Eine Konfiguration eines Netzwerkes $\mathcal{N}^{(n)}$ ist ein n-Tupel $C = (C(1), C(2), \ldots, C(n))$ wobei $C(i) \subseteq V^*$ für $1 \leq i \leq n$ gilt.*

(iii) *Es seien $C = (C(1), C(2), \ldots, C(n))$ und $C' = (C'(1), C'(2), \ldots, C'(n))$ zwei Konfigurationen eines Netzwerkes $\mathcal{N}^{(n)}$. Wir sagen, dass C zu C' in einem*

- *Evolutionsschritt abgeleitet wird (geschrieben $C \Longrightarrow C'$), wenn für $1 \leq i \leq n$ die Menge $C'(i)$ aus allen Wörtern $w \in C(i)$, auf die keine Regel aus M_i anwendbar ist, und aus allen Wörtern w, zu denen ein Wort $v \in C(i)$ und eine Regel $p \in M_i$ so existieren, dass $v \Longrightarrow_p w$ gilt, besteht,*
- *Kommunikationsschritt abgeleitet wird (geschrieben $C \vdash C'$), wenn für $1 \leq i \leq n$*

$$C'(i) = (C(i) \setminus O_i) \cup \bigcup_{(k,i) \in E} C(k) \cap O_k \cap I_i$$

gilt.

Die Berechnung eines Netzwerkes $\mathcal{N}^{(n)}$ auf einem Eingabewort $w \in U^*$ ist eine Folge von Konfigurationen $C_t^w = (C_t^w(1), C_t^w(2), \ldots, C_t^w(n))$, $t \geq 0$, derart, dass

- $C_0^w = (C_0^w(1), C_0^w(2), \ldots, C_0^w(n))$ *mit* $C_0^w(j) = \{w\}$ *und* $C_0^w(i) = \emptyset$ *für* $1 \leq i \neq j \leq n$ *gilt,*
- *für alle* $t \geq 0$ *die Konfiguration* C_{2t}^w *zu* C_{2t+1}^w *in einem Evolutionsschritt führt:* $C_{2t}^w \Longrightarrow C_{2t+1}^w$,
- *für alle* $t \geq 0$ *die Konfiguration* C_{2t+1}^w *zu* C_{2t+2}^w *in einem Kommunikationsschritt führt:* $C_{2t+1}^w \vdash C_{2t+2}^w$.

(iv) *Die von einem Netzwerk* $\mathcal{N}^{(n)}$ *schwach akzeptierte Sprache* $L_{\mathrm{w}}(\mathcal{N})$ *und stark akzeptierte Sprache* $L_{\mathrm{s}}(\mathcal{N})$ *sind als*

$$L_{\mathrm{w}}(\mathcal{N}^{(n)}) = \{\, w \in U^* \mid \exists t \geq 0 \, \exists o \in O : C_t^w(o) \neq \emptyset \,\} \text{ und}$$
$$L_{\mathrm{s}}(\mathcal{N}^{(n)}) = \{\, w \in U^* \mid \exists t \geq 0 \, \forall o \in O : C_t^w(o) \neq \emptyset \,\}$$

definiert, wobei $C_t^w = (C_t^w(1), C_t^w(2), \ldots, C_t^w(n))$, $t \geq 0$ *die Berechnung von* $\mathcal{N}^{(n)}$ *auf* w *ist.*

Man stelle sich ein Netzwerk evolutionärer Prozessoren als einen gerichteten Graphen vor, dessen Knoten N_i ($1 \leq i \leq n$) Prozessoren sind, die über die Kanten (angegeben durch die Menge E) Wörter austauschen. Jeder Prozessor N_i hat eine Menge M_i von Evolutionsregeln, einen Eingangsfilter I_i und einen Ausgangsfilter O_i. Der Prozessor heißt

- ersetzend, wenn $M_i \subseteq \{a \to b \mid a, b \in V\}$ gilt,
- einfügend, wenn $M_i \subseteq \{\lambda \to b \mid b \in V\}$ gilt, und
- löschend, wenn $M_i \subseteq \{a \to \lambda \mid a \in V\}$ gilt.

Mit einem Knoten N_i und einer Zeit $t \geq 0$ verbinden wir eine Wortmenge $C_t(i)$. Zu Beginn enthält der ausgewiesene Knoten N_j das Eingabewort; alle anderen Knoten enthalten keine Wörter.

In einem Evolutionsschritt leitet jeder Prozessor seine Wortmenge entsprechend seiner Regeln ab. Die neue Wortmenge besteht dabei aus allen jenen Wörtern, die dadurch entstehen, dass eine Regel auf ein Wort der ursprünglichen Menge an einer möglichen Stelle angewendet wird, und jenen Wörten, auf die keine Regel anwendbar ist.

In einem Kommunikationsschritt sendet jeder Prozessor N_i alle Wörter, die seinen Ausgangsfilter passieren, zu allen benachbarten Prozessoren (zu denen eine Kante hinführt). Die Wörter, die der Ausgangsfilter nicht durchlässt, bleiben im Prozessor. Außerdem nimmt jeder Prozessor alle Wörter auf, die ihn auf einer ankommenden Kante erreichen und seinen Eingangsfilter passieren. Wörter, die einen Knoten verlassen und von keinem Knoten aufgenommen werden, gehen verloren (verschwinden aus dem Netzwerk).

Die Arbeit eines Netzwerkes beginnt mit einem Evolutionsschritt; danach wechseln sich Kommunikations- und Evolutionsschritte ab. Wenn zu einem Zeitpunkt ein Ausgabeknoten ein Wort enthält, so wird das Eingabewort schwach akzeptiert. Wenn zu einem Zeitpunkt alle Ausgabeknoten ein Wort enthalten, so wird das Eingabewort stark akzeptiert. Die schwach oder stark akzeptierte Sprache eines Netzwerkes ist die Menge aller Wörter, die schwach bzw. stark akzeptiert werden.

2. Netzwerke ohne einfügende Prozessoren

In [5] wurde gezeigt, dass Netzwerke ohne löschende Prozessoren die kontextabhängigen Sprachen erzeugen. In diesem Abschnitt betrachten wir den dualen Fall: akzeptierende Netzwerke ohne einfügende Prozessoren. In [4] wurde gezeigt, dass jede von einem solchen Netzwerk akzeptierte Sprache kontextabhängig ist und zu jeder kontextabhängigen Sprache ein solches Netzwerk existiert, das die Sprache akzeptiert. Die Anzahl der Prozessoren ist bei dem konstruierten Netzwerk linear in der Anzahl der Regeln der zugrunde liegenden Grammatik.

In diesem Abschnitt wird gezeigt, dass die Anzahl der Prozessoren nicht von der Grammatik abhängt.

Satz 2.1 *Zu jeder kontextabhängigen Sprache L gibt es ein Netzwerk $\mathcal{N}$ evolutionärer Prozessoren mit jeweils genau einem ersetzenden, einem löschenden und einem Ausgabe-Knoten ohne Regeln, das schwach und stark die Sprache L akzeptiert: $L = L_{\mathrm{w}}(\mathcal{N}) = L_{\mathrm{s}}(\mathcal{N})$.*

Beweis. Zu jeder kontextabhängigen Sprache L gibt es eine monotone Grammatik $G = (N, T, P, S)$ in Kuroda-Normalform, die L erzeugt. Ähnlich zum Beweis in [5], dass zum Erzeugen ein ersetzender und ein einfügender Knoten ausreichen, kann ein akzeptierendes Netzwerk zu L konstruiert werden. Dabei werden die Regeln der zugehörigen Grammatik G rückwärts simuliert. In dem ersetzenden Knoten werden die Regeln mit längengleicher rechter und linker Seite simuliert ($A \to x$ für $A \in N$ und $x \in N \cup T$ sowie $AB \to CD$ für $A, B, C, D \in N$). Der Ausgangsfilter sorgt dafür, dass die Wörter in einem erfolgreichen Ableitungsprozess den Knoten nicht verlassen. Alle anderen Wörter verlassen den Knoten, passieren aber nicht die Eingangsfilter der anderen Knoten und verschwinden somit aus dem Netzwerk. Zum Rückwärtssimulieren von Regeln der Form $A \to BC$ (für $A, B, C \in N$) werden der ersetzende und der löschende Knoten gebraucht. Ein Wort w gehört genau dann zur Sprache L, wenn es eine „Rückwärtsableitung“ zum Axiom S gibt. Dieses Wort ist das einzige, das den Eingangsfilter des Ausgabe-Knotens passiert. Somit erhält der Ausgabe-Knoten genau dann ein Wort, wenn das Eingabewort

zur Sprache gehört. Das Netzwerk akzeptiert also die Sprache L. Da es nur einen Ausgabe-Knoten gibt, stimmen schwache und starke Akzeptanz überein. □

Wenn wir nur löschende Prozessoren erlauben, können nicht alle kontextabhängigen Sprachen akzeptiert werden, auch nicht alle linearen Sprachen. Zum Beispiel kann die lineare Sprache $\{ a^n b a^n \mid n \geq 1 \}$ nicht akzeptiert werden. Zu einem Eingabewort $a^n b a^m$ mit $n \geq 1$, $m \geq 1$ kann ein Netzwerk nur durch Löschen und die regulären Filter nicht entscheiden, ob $n = m$ gilt (wenn ein a gelöscht wird, sieht das Netzwerk nicht, auf welcher Seite vom b es war).

Folglich ist das beschriebene Netzwerk bezüglich der Anzahl der ersetzenden Knoten optimal. Aber wir können zeigen, dass löschende Prozessoren nicht nötig sind.

Satz 2.2 *Zu jeder kontextabhängigen Sprache L gibt es ein Netzwerk $\mathcal{S}$ evolutionärer Prozessoren mit jeweils genau einem ersetzenden Knoten und einem Ausgabe-Knoten ohne Regeln, das schwach und stark die Sprache L akzeptiert: $L = L_{\mathrm{w}}(\mathcal{S}) = L_{\mathrm{s}}(\mathcal{S})$.*

Beweis. Hierzu wird das Netzwerk aus dem vorhergehenden Beweis so modifiziert, dass der ersetzende Knoten die Aufgabe des löschenden Knotens übernimmt. Allerdings wird ein Zeichen nicht wirklich gelöscht, sondern durch eine Löschmarkierung $\lrcorner$ ersetzt. Die Filter müssen dann so beschaffen sein, dass alle Löschmarkierungen in einem Wort ignoriert werden. Der Ausgabe-Knoten lässt alle Wörter zu, die das Axiom S enthalten und ansonsten nur aus Löschmarkierungen bestehen. □

Diese Anzahl von Prozessoren ist optimal, da Eingabe- und Ausgabe-Knoten verschieden sein müssen (sonst würde jedes Eingabewort akzeptiert werden).

3. Netzwerke mit einfügenden Prozessoren

Der wesentliche Unterschied zwischen kontextabhängigen und nicht kontextabhängigen Grammatiken in Kuroda-Normalform besteht darin, dass in der Normalform einer beliebigen Regelgrammatik löschende Regeln (λ-Regeln) erlaubt sind. Um eine λ-Regel rückwärts zu simulieren, verwenden wir einen einfügenden Knoten.

Satz 3.1 *Zu jeder rekursiv aufzählbaren Sprache L gibt es ein Netzwerk $\mathcal{N}$ evolutionärer Prozessoren mit jeweils genau einem ersetzenden, einem einfügenden und einem Ausgabe-Knoten ohne Regeln, das schwach und stark die Sprache L akzeptiert: $L = L_{\mathrm{w}}(\mathcal{N}) = L_{\mathrm{s}}(\mathcal{N})$.*

Beweis. Die Idee ist, das Netzwerk $\mathcal{S}$ aus dem Beweis zu Satz 2.2 um einen einfügenden Prozessor zu erweitern, der die Rückwärtssimulation der λ-Regeln übernimmt.

Zwischen den Rückwärtssimulationen zweier Regeln kann der ersetzende Knoten ein Symbol markieren, damit das Wort den Knoten verlassen und zum einfügenden Knoten wechseln darf. Der Prozessor fügt dann ein Nichtterminal ein, das zu einer λ-Regel der zugrunde liegenden Grammatik G gehört, und sendet das Wort an den ersetzenden Knoten zurück. Dieser Prozessor muss zunächst die Markierung aufheben. Falls eine Markierung nicht im richtigen Moment gesetzt oder aufgehoben wird, geht das Wort verloren. Auch hier wird ein Eingabewort genau dann akzeptiert, wenn es auf das Startwort S der Grammatik zurückgeführt werden kann (als gelöscht markierte Stellen werden ignoriert). □

In [1] wurde gezeigt, dass jede rekursiv aufzählbare Sprache von einem Netzwerk mit einem einfügenden und einem löschenden Prozessor erzeugt werden kann. Analog dazu kann die folgende Aussage bewiesen werden.

Satz 3.2 *Zu jeder rekursiv aufzählbaren Sprache L gibt es ein Netzwerk $\mathcal{N}$ evolutionärer Prozessoren mit zwei einfügenden, einem löschenden und einem Ausgabe-Knoten, das schwach und stark die Sprache L akzeptiert: $L = L_{\mathrm{w}}(\mathcal{N}) = L_{\mathrm{s}}(\mathcal{N})$.*

Beweis. Es sei $w = a_1 a_2 \cdots a_n$ das Eingabewort. Einer der beiden einfügenden Knoten wandelt zunächst das Wort zu $a_1 \textvisiblespace a_2 \textvisiblespace \cdots a_n \textvisiblespace$ um. Die anderen beiden Knoten simulieren dann die Ableitung rückwärts analog zum Verfahren aus [1]. □

Würde das Eingabewort bereits in der Weise geliefert werden, dass die Buchstaben durch einzelne Leerzeichen getrennt sind, wären ein einfügender, ein löschener und ein Ausgabe-Knoten ausreichend.

Literatur

[1] A. ALHAZOV, J. DASSOW, C. MARTIN-VIDE, YU. ROGOZHIN und B. TRUTHE, On Networks of Evolutionary Processors with Nodes of Two Types. Eingereicht.

[2] J. CASTELLANOS, C. MARTIN-VIDE, V. MITRANA und J. SEMPERE, Solving NP-complete problems with networks of evolutionary processors. In: *Proc. IWANN*, LNCS 2084, Springer-Verlag, Berlin, 2001, 621–628.

[3] E. CSUHAJ-VARJÚ und A. SALOMAA, Networks of parallel language processors. In: GH. PĂUN und A. SALOMAA (Hrsg.), *New Trends in formal Language Theory*. LNCS 1218, Springer-Verlag, Berlin, 1997, 299–318.

[4] J. DASSOW und V. MITRANA, Accepting networks of non-inserting evolutionary processors. In: I. PETRE und G. ROZENBERG (Hrsg.), *Proceedings of NCGT 2008 – Workshop on Natural Computing and Graph Transformations, Leicester, United Kingdom, September 8, 2008*. University of Leicester, 2008, 29–41.

[5] J. DASSOW und B. TRUTHE, On the Power of Networks of Evolutionary Processors. In: J. DURAND-LOSE und M. MARGENSTERN (Hrsg.), *Machines, Computations and Universality, MCU 2007, Orléans, France, September 10–13, 2007, Proc*. LNCS 4664, Springer, 2007, 158–169.

[6] M. MARGENSTERN, V. MITRANA und M. J. PÉREZ-JIMÉNEZ, Accepting Hybrid Networks of Evolutionary Processors. In: C. FERRETTI, G. MAURI und C. ZANDRON (Hrsg.), *10th International Workshop on DNA Computing*. LNCS 3384, Springer, 2005, 235–246.

[7] G. ROZENBERG und A. SALOMAA, *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.

Teilnehmerverzeichnis

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

Sviatlana Danilava
Goethe-Universität
Institut für Informatik
Postfach 11 19 32
60054 Frankfurt am Main, Germany
danilava@em.uni-frankfurt.de

Jürgen Dassow
Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Postfach 4120
39016 Magdeburg, Germany
dassow@iws.cs.uni-magdeburg.de

Henning Fernau
Universität Trier
Fachbereich IV – Informatik
54286 Trier, Germany
fernau@uni-trier.de

Rudolf Freund
Technische Universität Wien
Institut für Computersprachen
Favoritenstraße 9
1040 Wien, Austria
rudi@logic.at

Dominik D. Freydenberger
Goethe-Universität
Institut für Informatik
Postfach 11 19 32
60054 Frankfurt am Main, Germany
freydenberger@em.uni-frankfurt.de

Hermann Gruber
Universität Gießen
Institut für Informatik
Arndtstraße 2
35392 Gießen, Germany
gruber@informatik.uni-giessen.de

Stefan Gulan
Universität Trier
Fachbereich IV – Informatik
54286 Trier, Germany
gulan@informatik.uni-trier.de

Markus Holzer
Universität Gießen
Institut für Informatik
Arndtstraße 2
35392 Gießen, Germany
holzer@informatik.uni-giessen.de

Norbert Hundeshagen
Universität Kassel
Fachbereich Elektrotechnik/Informatik
Wilhelmshöher Allee 71–73
34121 Kassel, Germany
norbert.hundeshagen@yahoo.de

Anna Kasprzik
Universität Trier
Fachbereich IV – Informatik
54286 Trier, Germany
kasprzik@uni-trier.de

Andreas Klein
Ghent University
Department of Pure Mathematics and Computer Algebra
Krijgslaan 281-S22
9000 Gent, Belgium
klein@cage.ugent.be

Martin Kutrib
Universität Gießen
Institut für Informatik
Arndtstraße 2
35392 Gießen, Germany
kutrib@informatik.uni-giessen.de

Andreas Malcher
Universität Gießen
Institut für Informatik
Arndtstraße 2
35392 Gießen, Germany
malcher@informatik.uni-giessen.de

Maurice Margenstern
Université Paul Verlaine – Metz
LITA
Île du Saulcy
57045 Metz cedex 1, France
margens@univ-metz.fr

Carlo Mereghetti
Università degli Studi di Milano
Dipartimento di Scienze dell'Informazione
via Comelico 39
20135 Milano, Italy
mereghetti@dsi.unimi.it

Hartmut Messerschmidt
Universität Kassel
Fachbereich Elektrotechnik/Informatik
Wilhelmshöher Allee 71–73
34121 Kassel, Germany
hardy@theory.informatik.uni-kassel.de

Friedrich Otto
Universität Kassel
Fachbereich Elektrotechnik/Informatik
Wilhelmshöher Allee 71–73
34121 Kassel, Germany
otto@theory.informatik.uni-kassel.de

Martin Plátek
Charles University
Malostranské náměsti 25
11800 Praha 1, Czech Republic
martin.platek@mff.cuni.cz

Daniel Reidenbach
Loughborough University
Department of Computer Science
Loughborough Leicestershire LE11 3TU, Great Britain
d.reidenbach@lboro.ac.uk

Christian Reitwießner
Universität Würzburg
Institut für Informatik
Am Hubland
97074 Würzburg, Germany
reitwiessner@informatik.uni-wuerzburg.de

Johannes C. Schneider
Technische Universität Kaiserslautern
Fachbereich Informatik
Postfach 3049
67653 Kaiserslautern, Germany
jschneider@informatik.uni-kl.de

Ludwig Staiger
Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik
Von-Seckendorff-Platz 1
06120 Halle an der Saale, Germany
staiger@informatik.uni-halle.de

Heiko Stamer
Universität Kassel
Fachbereich Elektrotechnik/Informatik
Wilhelmshöher Allee 71–73
34121 Kassel, Germany
stamer@theory.informatik.uni-kassel.de

Bianca Truthe
Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Postfach 4120
39016 Magdeburg, Germany
truthe@iws.cs.uni-magdeburg.de

György Vaszil
MTA SZTAKI
Kende utca 13–17
1111 Budapest, Hungary
vaszil@sztaki.hu

Detlef Wotschke
Goethe-Universität
Institut für Informatik
Postfach 11 19 32
60054 Frankfurt am Main, Germany
wotschke@em.uni-frankfurt.de