FACULTY OF MATHEMATICS AND PHYSICS
CHARLES UNIVERSITY IN PRAGUE

22nd THEORIETAG

AUTOMATA AND FORMAL LANGUAGES

OCTOBER 3–5, 2012, PRAGUE
PROCEEDINGS

František Mráz (Ed.)

matfyzpress
PRAGUE 2012

ISBN 978-80-7378-221-4

Preface

Theorietag is an annual meeting of the special interest group *Automata and Formal Languages* of German Informatics Society (*die Fachgruppe Automaten und Formale Sprachen der Gesellschaft für Informatik*). These meetings started 21 years ago. Theorietags were organized by members of the special interest group in the following places:

1991 Magdeburg	1992 Kiel
1993 Schloß Dagstuhl	1994 Herrsching near Munich
1995 Schloß Rauischholzhausen	1996 Cunnersdorf in Saxon Switzerland
1997 Barnstorf near Bremen	1998 Riveris near Trier
1999 Schauenburg-Elmshagen near Kassel	2000 Vienna (Austria)
2001 Wendgräben near Magdeburg	2002 Lutherstadt Wittenberg
2003 Herrsching near Munich	2004 Caputh near Potsdam
2005 Lauterbad near Freudenstadt	2006 Viennna (Austria)
2007 Leipzig	2008 Wettenberg-Launsbach near Gießen
2009 Lutherstadt Wittenberg	2010 Baunatal near Kassel
2011 Alrode in Harz	

Already two times Theorietag took place outside of Germany – 2000 and 2006 both times in Vienna. This year Theorietag left Germany again – it was held in Prague (Czech Republic) during October 3-5, 2012. The reason was that Prague formal languages group tightly cooperates with several German colleagues and people from Prague visited several Theorietags.

The 22nd Theorietag was organized by a group from the Faculty of Mathematics and Physics, Charles University in Prague. The members of this group work mainly in the field of formal languages, automata theory and linguistics. This fact influenced also the selection of speakers invited for an accompanying workshop on October 3, 2012:

- Jan Holub (Czech Technical University, Prague),
- Petr Jančar (Technical University of Ostrava, Ostrava),
- Karel Oliva (Academy of Sciences of the Czech republic, Prague),
- Heiko Vogler (Technische Universität Dresden, Dresden), and
- Zdeněk Žabokrtský (Charles University, Prague).

These proceedings comprise extended abstracts of 5 invited lectures and 21 contributed talks. The authors of the papers submitted to Theorietag 2012 are from 7 countries including Germany, Austria, Czech Republic, Moldova, France, United Kingdom and USA. We appreciate the contribution of all the participants to the scientific program of Theorietag 2012 as well as their preparation of the submitted up-to date high quality manuscripts.

Finally, the organizers would like to thank the German Informatics Society, the Faculty of Mathematics and Physics of Charles University in Prague and the Czech Science Foundation (grant projects No. P103/10/0783 and P202/10/1333) for their support. We would like to express our thanks also to Anna Kotěšovcová (Conforg) for local arrangements and organization of Theorietag 2012.

Prague, September 2012 Markéta Lopatková, František Mráz and Martin Plátek

Table of Contents

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 1 – 2

The Finite Automata Approaches for Bioinformatics

Jan Holub

The Prague Stringology Club
Department of Theoretical Computer Science
Faculty of Information Technology
Czech Technical University in Prague, Czech Republic

e-mail: Jan.Holub@fit.cvut.cz

Stringology is a part of computer science on string and sequence processing. Finite automata can solve efficiently many tasks in stringology. An overview of four approaches of the finite automata use in stringology as well as handling complex alphabets is presented on various bioinformatic tasks.

Deterministic Finite Automaton Deterministic finite automaton (DFA) can be directly used. Given a text t of size n and a pattern p of size m, one can build a DFA from p. The DFA runs over the t and searches for all locations p in t. This is called forward matching. In case of backward matching, the DFA is aligned to the first m symbols of t and reads symbols of t backwards (i.e., $t[m], t[m-1], \ldots$). When a mismatch is found the DFA is aligned one or more positions to the right than the previous alignment. One can also build a DFA from t (so called indexing automaton: *suffix trie*, *suffix tree*, *suffix automaton*, *compact suffix automaton*, *factor automaton*, *oracle automaton*) which provides the full index of t. The indexing automaton gets p on input and answers if p is in t in time linear with the size of pattern p.

Each depth of automaton represents a position in pattern or text. In approximate pattern matching where the found occurrences of p can be within a limited edit distance $k, 1 \leq k < m$, from p one level of states is inserted for each value of edit distance allowed $(0, 1, \ldots, k)$. Various kinds of distances can be used: Hamming, Levenshtein, Damerau, Indel, …

Deterministic Simulation of Nondeterministic Finite Automaton Nondeterministic finite automaton (NFA) cannot be directly used. It can be transformed to DFA or NFA run can be deterministically simulated. While the determinisation can lead up to very large automata, the simulation may be used in the case when the resulting DFA is too big for practical use. In simulation techniques one can control memory used at the expense of the running time. The basic simulation method is general and can be used for any NFA. Simulation techniques *dynamic programming* and *bit parallelism* improve the complexities of simulation but require some regular structure of the NFA used.

Finite Automaton as a Model of Computation Some tasks can be solved just by construction of finite automaton without the need to run it. It is even not necessary to build the whole automaton. For example let $\mathcal{M}_p$ be an NFA for approximate pattern matching for pattern p (i.e., language $\mathcal{L}_p$ accepted by $\mathcal{M}_p$ contains all patterns within a given edit distance from p)

and $\mathcal{M}_t$ be the suffix automaton for t (i.e., language $\mathcal{L}_t$ accepted by $\mathcal{M}_t$ contains all factors of t). Reaching final states during construction of intersection automaton accepting $\mathcal{L}_p \cap \mathcal{L}_t$ gives answer to approximate string matching.

Compositions of Finite Automata Solutions More complex tasks can be divided into several subtasks and their solutions put together. Parallel, serial, and cascade compositions are recognized.

Complex Alphabet The way the finite automata can process strings build over more complex alphabet than just single symbols (degenerate symbols, strings, variables) is also shown.

Bioinformatics Those approaches and complex alphabet use are demonstrated in various tasks of bioinformatics. The tasks work with sequences on various alphabets: DNA, RNA, amino acids. The searched patterns may be explicitly defined or may be defined by their properties. Some patterns may allow errors some must match exactly. Some pattern symbols may match exactly one symbol, some may match a subset of alphabet as they were not read exactly or their practical behaviour is the same.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 3 – 13

Language Equivalence of Deterministic Pushdown Automata via First-Order Grammars

Petr Jančar

Techn. Univ. Ostrava
petr.jancar@vsb.cz

Abstract

Decidability of language equivalence of deterministic pushdown automata was established by G. Sénizergues (1997), who thus solved a famous long-standing open problem. A simplified proof, also providing a primitive recursive complexity upper bound, was given by C. Stirling (2002). This text is an introduction to the talk at Theorietag 2012 which is based on [Jančar, LiCS 2012] and aims to explain the decidability in the framework of first-order terms and grammars. The proof is based on the abstract ideas used in the previous proofs, but the chosen framework seems to be more natural for the problem and allows a short presentation which should be transparent for a general computer science audience.

1. Introduction

Language equivalence of deterministic pushdown automata (DPDA) is a famous problem in language theory. The decidability question for this problem was posed in the 1960s [5], then a series of works solving various subcases followed, until the question was answered positively by Sénizergues in 1997, with the full journal version [10]. G. Sénizergues was awarded the Gödel prize in 2002 for this significant achievement.

Later Stirling [14] and also Sénizergues [11] provided simpler proofs than the original proof. A modified version, showing also a (nonelementary) primitive recursive complexity upper bound, appeared as a conference paper by Stirling in 2002 [15]; Sénizergues showed a "more reasonable" upper bound for a subclass in [12]. Sénizergues also generalised the decidability result to bisimulation equivalence over a class of nondeterministic pushdown automata [13]. Unfortunately, even the simpler proofs seem rather long and technical, which does not ease further research regarding, e.g., the complexity. (DPDA language equivalence is only known to be PTime-hard; the general bisimilarity problem is ExpTime-hard [8].)

The algorithms are based on the following key points. If two configurations are nonequivalent then there is a shortest word witnessing this fact, an *sw-word* for short. If two configurations are equivalent then any attempt to (stepwise) build a potential sw-word can be contradicted: an (algorithmically verifiable) proof of a contradiction is produced after a (sufficiently long) prefix of the potential sw-word has been constructed.

The work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), and by the Czech Grant Agency (GAČR:P202/11/0340).

One reason why the DPDA problem turned out so intricate seems to be the lack of structure of configurations (strings of symbols), which calls for a richer framework. This is also discussed by Stirling [14] who refers to the algebraic theory of linear combinations of boolean rational series built by Sénizergues, and replaces it by a process calculus whose processes are derived from determinising strict grammars. Another difficulty is that providing a proof of equivalence for a given pair of states (i.e. structured objects representing configurations) seems to require some measures and conditional rules in the respective logical systems, to keep their soundness. Stirling used (simpler) bisimulation approximants instead of Sénizergues's system of weights.

We can view the transformations T_B, T_C in [10] as two main means for contradicting that a particular word is a prefix of an sw-word; Stirling [14] uses the rules BAL and CUT. BAL ("balancing") aims at making the (structured) states in the pairs along the supposed sw-word close to each other, i.e. having "bounded (different) heads" and the same (maybe large) "tails." CUT aims at "cutting away" large tails soundly.

Here we re-prove the decidability of DPDA language equivalence in the framework of (deterministic) first-order grammars, i.e. systems of first-order terms with finitely many root-rewriting rules. The framework of *regular terms* (possibly infinite terms with only finitely many different subterms), completed *with substitutions*, seems to be a more natural and simpler substitute of the algebraic structures in the previous works. In fact, close relations between the frameworks of (D)PDA, strict deterministic grammars and first-order schemes were recognized long ago (see, e.g., references in [3] and [10]).

The strategy of the presented proof is not new, it is close to [14] in particular, but the proof is not just a translation of a previous proof into another framework. Instead of the previous logical systems we use a *Prover-Refuter game* whose soundness is obvious. The CUT rule is replaced with a *finite basis*, generating the equivalence relation in a certain sense. The chosen framework and the new ingredients allow to present a direct, short, and easily understandable proof. Moreover, the presentation is tailored so that the complexity result [15] and the generalization to bisimilarity [13] can be added smoothly.

There are comprehensive references in Sénizergues's and Stirling's papers to the prior research; recent related research (on complexity and on higher-order schemes) can be found, e.g., in [1], [2], [4], [7], [9] and the references therein.

The informal text below gives a flavour of the proof. Full details are in the paper [6] (as well as in the version on the author's web-page www.cs.vsb.cz/jancar).

2. Informal sketch of the decidability proof

Labelled Transition Systems (LTSs) and Trace Equivalence

Fig. 1 shows a (finite) *deterministic labelled transition system* $\mathcal{L} = (\mathcal{S}, \mathcal{A}, (\xrightarrow{a})_{a \in \mathcal{A}})$. E.g., we have $s_1 \xrightarrow{bab}$, i.e. the *trace bab* is *enabled in* the *state* s_1, since $s_1 \xrightarrow{bab} s_3$ (such s_3 is unique since $\mathcal{L}$ is deterministic); on the other hand, $\neg(s_1 \xrightarrow{baa})$. We are interested in the following form of language equivalence. *Trace equivalence* $\sim$ on $\mathcal{S}$, and its "strata" $\sim_0$, $\sim_1$, $\sim_2$, ..., are defined as follows:

$$s \sim t \text{ if } \forall w \in \mathcal{A}^* : s \xrightarrow{w} \Leftrightarrow t \xrightarrow{w} \text{; for } k \in \mathbb{N}\text{: } s \sim_k t \text{ if } \forall w \in \mathcal{A}^{\leq k} : s \xrightarrow{w} \Leftrightarrow t \xrightarrow{w},$$

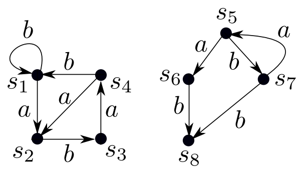


Figure 1: A (finite) deterministic labelled transition system

where $\mathcal{A}^{\leq k} = \{w \in \mathcal{A}^* \mid |w| \leq k\}$. We note that $\mathcal{S}\times\mathcal{S} = \sim_0 \supseteq \sim_1 \supseteq \sim_2 \supseteq \cdots$, and $\bigcap_{k\in\mathbb{N}} \sim_k = \sim$.

We attach the *equivalence-level* (*eq-level*) to each pair of states; e.g., $\text{EQLV}(s_1,s_2) = 0$, $\text{EQLV}(s_1,s_5) = 2$ (since $s_1 \sim_2 s_5$ and $s_1 \not\sim_3 s_5$), $\text{EQLV}(s_1,s_4) = \omega$ (i.e. $s_1 \sim s_4$). We note:

Proposition 2.1 *If* $\text{EQLV}(s,t){=}k$ *and* $\text{EQLV}(s,s'){\geq}k{+}1$ *then* $\text{EQLV}(s',t) = k$ *(since $s' \sim_k s \sim_k t$ and $s' \sim_{k+1} s \not\sim_{k+1} t$).*

Proposition 2.2 *Given a* deterministic *LTS:*
(1) If $s \xrightarrow{w} s'$, $t \xrightarrow{w} t'$ *then* $\text{EQLV}(s',t') \geq \text{EQLV}(s,t) - |w|$.
(2) If $\text{EQLV}(s,t) = k \in \mathbb{N}$ *then there is* $w = a_1a_2\ldots a_k$ *such that* $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \cdots \xrightarrow{a_k} s_k$ *and* $t \xrightarrow{a_1} t_1 \xrightarrow{a_2} t_2 \xrightarrow{a_3} \cdots \xrightarrow{a_k} t_k$ *where* $\text{EQLV}(s_j,t_j) = k-j$ *for* $j = 1,2,\ldots,k$.

In Point (2) we have $s_k \not\sim_1 t_k$, hence there is $a \in \mathcal{A}$ such that $s_k \xrightarrow{a}$, $\neg(t_k \xrightarrow{a})$ or vice versa; the word wa is then a *shortest nonequivalence-witness word* for the pair s,t.

First-Order Regular Terms and Substitutions

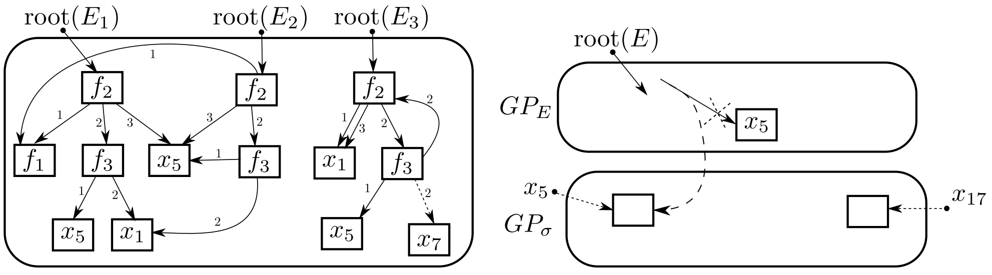


Figure 2: Graph presentations of terms (left); creating $\text{GP}_{E\sigma}$ from GP_E and GP_σ (right)

Imagine that the black dots (states) in Fig. 1 are, in fact, regular first-order terms, and we can have infinitely many such states. Fig. 2 (left) recalls the standard notion of *first-order terms* over a ranked alphabet $\mathcal{F} = \{f_1, f_2, \ldots, f_k\}$, with variables $\text{VAR} = \{x_1, x_2, x_3, \ldots\}$. By *terms* comprised in the set $\text{TERMS}_\mathcal{F}$ we mean *regular terms*; they can be infinite but have finite presentations (i.e., each regular term has finitely many subterms). Fig. 2 (right) presents a finite-support *substitution* $\sigma : \text{VAR} \to \text{TERMS}_\mathcal{F}$ (where the support $\text{SUPP}(\sigma) = \{i \mid \sigma(x_i) \neq x_i\}$ is finite), and makes clear how a graph presentation of the term $E\sigma$ arises from graph presentations of E and σ. Fig. 2 also illustrates that the (infinite regular) term E_3 arises from $E = f_2(x_1, f_3(x_5, x_7), x_1)$ by applying the substitution $\sigma = \{(x_7, E)\}$ repeatedly forever: $E_3 = E\sigma\sigma\sigma\ldots$.

(Deterministic) First-Order Grammars as Generators of (Deterministic) LTSs

$$r_1 : Ax_1 \xrightarrow{a} ABx_1 \qquad r_2 : Ax_1 \xrightarrow{b} x_1 \qquad r_3 : Bx_1 \xrightarrow{a} BAx_1 \qquad r_4 : Bx_1 \xrightarrow{b} x_1$$

Figure 3: A det-first-order grammar $\mathcal{G} = (\{A,B\},\{a,b\},\{r_1,r_2,r_3,r_4\})$

Fig. 3 gives an example of a (deterministic) *first-order grammar*, i.e. of a structure $\mathcal{G} = (\mathcal{N}, \mathcal{A}, \mathcal{R})$ where $\mathcal{N}$ is a finite set of ranked *nonterminals* (or function symbols), $\mathcal{A}$ is a finite set of *actions* (or terminals), and $\mathcal{R}$ is a finite set of *(root rewriting) rules* r of the form

$$r : Yx_1x_2\ldots x_m \xrightarrow{a} E \tag{1}$$

where $Y \in \mathcal{N}$, $arity(Y) = m$, $a \in \mathcal{A}$, and E is a *finite* term over $\mathcal{N}$ in which each occurring variable is from the set $\{x_1, x_2, \ldots, x_m\}$. ($E = x_i$, where $1 \leq i \leq m$, is an example.)
$\mathcal{G} = (\mathcal{N}, \mathcal{A}, \mathcal{R})$ is *deterministic*, a *det-first-order grammar*, if there is at most one rule (1) for each pair $Y \in \mathcal{N}$, $a \in \mathcal{A}$.

$\mathcal{G} = (\mathcal{N}, \mathcal{A}, \mathcal{R})$ generates (the *rule based*) LTS $\mathcal{L}^{\mathrm{R}}_{\mathcal{G}} = (\textsc{Terms}_{\mathcal{N}}, \mathcal{R}, (\xrightarrow{r})_{r\in\mathcal{R}})$: for each rule $r : Yx_1x_2\ldots x_m \xrightarrow{a} E$ we have

$$F \xrightarrow{r} H \text{ if there is a substitution } \sigma \text{ such that } F = (Yx_1\ldots x_m)\sigma \text{ and } H = E\sigma.$$

In (the *action-based*) LTS $\mathcal{L}^{\mathrm{A}}_{\mathcal{G}} = (\textsc{Terms}_{\mathcal{N}}, \mathcal{A}, (\xrightarrow{a})_{a\in\mathcal{A}})$ we have $F \xrightarrow{a} H$ if $F \xrightarrow{r} H$ for some $r \in \mathcal{R}$ in the form .. $\xrightarrow{a}$ ...
Convention. We use Y to range over $\mathcal{N}$, and we might also use A, B for nonterminals, but E, F, G, H and T, U, V, W will always range over $\textsc{Terms}_{\mathcal{N}}$. We implicitly assume a special nonterminal $\perp$ with $arity(\perp) = 0$ which is *dead* (not enabling any transition). In $\mathcal{L}^{\mathrm{R}}_{\mathcal{G}}$ the variables x_i are also dead terms but in $\mathcal{L}^{\mathrm{A}}_{\mathcal{G}}$ we implicitly assume $x_i \xrightarrow{a_{x_i}} x_i$ for a special action unique to x_i. In fact, we do not use these special actions, we just use the corollary that $x_i \not\sim_1 H$ if $H \neq x_i$ (in particular if $H = x_j$ for $j \neq i$).

In our example in Fig. 3 we have $arity(A) = arity(B) = 1$. $Ax_1 \xrightarrow{a} ABx_1$, $Bx_5 \xrightarrow{b} x_5$, $ABx_2 \xrightarrow{a} ABBx_2$, $BA\perp \xrightarrow{b} A\perp$ are examples of transitions in $\mathcal{L}^{\mathrm{A}}_{\mathcal{G}}$. Fig. 4 can be viewed as a proof of the fact $AB\perp \sim_2 BA\perp$ (we have performed all enabled traces of length ≤ 2 on both sides simultaneously, and we have not found a nonequivalence witness).

Fig. 5 illustrates how (rewriting) rules are applied in the general case. Fig. 6 shows a path in $\mathcal{L}^{\mathrm{A}}_{\mathcal{G}}$; note that we can start from a regular term F but the changes consist in adding and removing finite "portions", and the path can "sink back into F" after a while. If $root(F) = A$ and the depicted root-successor in F is the second one then $a_1a_2\ldots a_6$ is an $(A,2)$-*sink word* (we have $Ax_1x_2\ldots x_m \xrightarrow{a_1a_2\ldots a_6} x_2$). Later we use the fact that for $\mathcal{G}$ we can compute a number M_0 bigger than the length of the shortest (Y, j)-sink words for all pairs (Y, j).

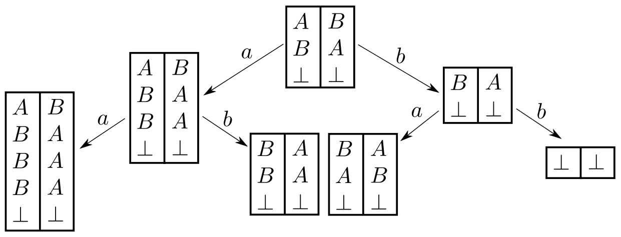


Figure 4: The 2-distance region $\text{REG}(T,U,2)$ for $(T,U) = (AB\bot, BA\bot)$

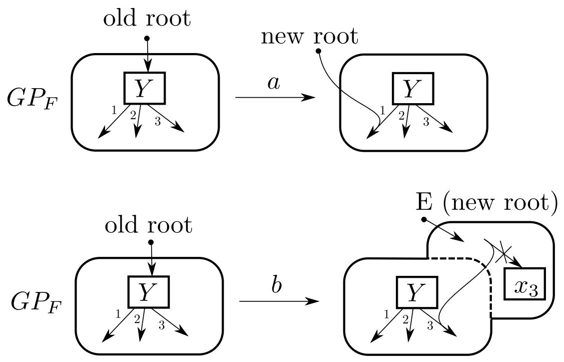


Figure 5: Applying rules $Yx_1x_2x_3 \xrightarrow{a} x_1$ and $Yx_1x_2x_3 \xrightarrow{b} E$ to GP_F

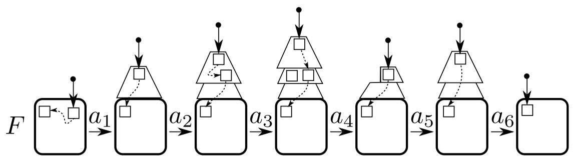


Figure 6: A path in $\mathcal{L}^{\mathbb{A}}_{\mathcal{G}}$

(D)PDA from a First-Order Term Perspective

Fig. 7 assumes a (D)PDA with the control state set $Q = \{q_1, q_2, q_3\}$, and shows how any configuration (control state and stack content) and any (push or pop) (D)PDA-rule can be naturally presented in the term-framework. The dotted arc shows that we can elegantly get rid of ε-steps in the case of DPDA: the dotted arc "swallows" the future possible step by the rule $q_2C \xrightarrow{\varepsilon} q_3$; recall that ε-steps can be assumed popping, and other steps are not available if an ε-step is enabled in a DPDA-configuration.

It is thus a simple technical routine to reduce the DPDA language equivalence problem to the following problem:

Problem TRACE-EQ-DET-G
Input: a det-first-order grammar $\mathcal{G} = (\mathcal{N}, \mathcal{A}, \mathcal{R})$, and two input (regular) terms T_{in}, U_{in}.
Question: is $T_{in} \sim U_{in}$ in $\mathcal{L}^{\mathbb{A}}_{\mathcal{G}}$?

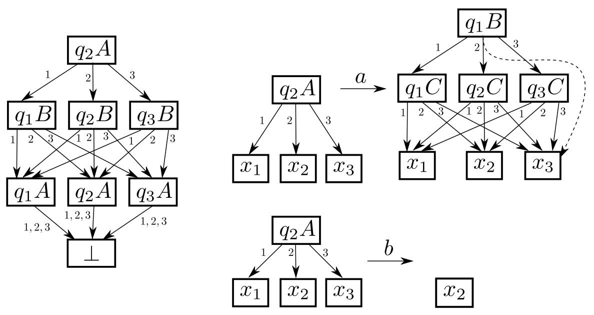


Figure 7: Term-presentations of PDA-configurations (q_2ABA) and rules ($q_2A \xrightarrow{a} q_1BC$, $q_2A \xrightarrow{b} q_2$)

Semidecidability of Trace Non-Equivalence

When discussing Fig. 4, we have implicitly touched on the following obvious fact:

Lemma 2.3 *There is an algorithm with the following property: it (halts and) computes* $\mathrm{EQLV}(T_{in}, U_{in})$ *for an instance* $\mathcal{G}, T_{in}, U_{in}$ *of* TRACE-EQ-DET-G *iff* $T_{in} \not\sim U_{in}$ *in* $\mathcal{L}^{\mathbb{A}}_{\mathcal{G}}$*. Thus the complement of* TRACE-EQ-DET-G *is semidecidable.*

An Algorithm (Semi)Deciding TRACE-EQ-DET-G

We first (easily) note the *congruence properties* in Prop. 2.4 (recall Fig. 2, 5, 6 and the fact that $x_i \not\sim_1 H$ if $H \neq x_i$). For two substitutions $\sigma, \sigma' : \mathrm{VAR} \to \mathrm{TERMS}_{\mathcal{N}}$ we define

$$\sigma \sim_k \sigma' \text{ if } \sigma(x_i) \sim_k \sigma'(x_i) \text{ for all } x_i \in \mathrm{VAR}.$$

Proposition 2.4 *(1) If* $E \sim_k F$ *then* $E\sigma \sim_k F\sigma$*. Hence* $\mathrm{EQLV}(E,F) \leq \mathrm{EQLV}(E\sigma, F\sigma)$*.*
(2) If $\sigma \sim_k \sigma'$ *then* $E\sigma \sim_k E\sigma'$*. Hence* $\mathrm{EQLV}(\sigma,\sigma') \leq \mathrm{EQLV}(E\sigma, E\sigma')$*.*

Imagine now a game between Prover (P, she) and Refuter (R, he), given our example-grammar $\mathcal{G}$ in Fig. 3 and $(T_{in}, U_{in}) = (AB\perp, BA\perp)$. Refuter claims $T_{in} \not\sim U_{in}$ but Prover aims to derive some consequences which contradict Refuter's claim. (Prover defends the claim $T_{in} \sim U_{in}$.) A particular play can run as follows.

Prover adds a finite set $\{(x_1, x_1), (Ax_1, Bx_1)\}$, which she somehow cleverly guesses and calls a BASIS, and claims that the eq-level of each pair $(AB\perp, BA\perp)$, (x_1,x_1), (Ax_1, Bx_1) is ω. Refuter must now choose a pair which he claims to have the least (and finite) eq-level; suppose he chooses $(T_0, U_0) = (AB\perp, BA\perp)$. Prover now chooses $k > 0$, say $k = 2$, and shows that $T_0 \sim_k U_0$, by constructing $\mathrm{REG}(T_0, U_0, k)$ as in Fig. 4.

Refuter then must choose a pair (T'_0, U'_0) in $\mathrm{REG}(T_0, U_0, k)$ for which he claims the least eq-level. This pair must be in the "bottom-row", in $\mathrm{REG}(T_0, U_0, k) \smallsetminus \mathrm{REG}(T_0, U_0, k-1)$ (recall Prop. 2.2); suppose he chooses $(T'_0, U'_0) = (ABBB\perp, BAAA\perp)$.

Prover can then adjust (T'_0, U'_0), using a sound subterm replacement, to get a pair (T_1, U_1) such that $\mathrm{EQLV}(T_1, U_1) = \mathrm{EQLV}(T'_0, U'_0)$ if Refuter's claims are true. E.g. she can say: by Refuter's claims and Prop. 2.2, we must have $\mathrm{EQLV}(B\perp, A\perp) >$

$\text{EQLV}(T_0',U_0')$ since the pair $(B\bot, A\bot)$ is in $\text{REG}(T_0,U_0,k-1)$ (we find it above the bottom-row). Hence $\text{EQLV}(ABBB\bot, ABBA\bot) > \text{EQLV}(T_0',U_0')$ (recall Prop. 2.4(2)), and thus $\text{EQLV}(ABBA\bot, BAAA\bot) = \text{EQLV}(ABBB\bot, BAAA\bot)$ (recall Prop. 2.1).

If Prover wishes, after creating (T_1,U_1) she can start a new phase: she chooses $k > 0$, shows $T_1 \sim_k U_1$ by constructing $\text{REG}(T_1,U_1,k)$, and lets Refuter choose $(T_1',U_1') \in \text{REG}(T_1,U_1,k) \smallsetminus \text{REG}(T_1,U_1,k-1)$ (in the new bottom row), for which he claims the least eq-level; Prover then possibly modifies (T_1',U_1') to get (T_2,U_2) and can start a new phase. (Fig. 10 shows schematically two consecutive phases.)

A play can be infinite but Prover is anytime allowed to contradict Refuter's claims by providing a verifiable proof. In our case, she can immediately continue using subterm replacements in (T_0',U_0'), using the pairs $(A\bot, B\bot)$, $(AB\bot, BA\bot)$, $(ABB\bot, BAA\bot)$, so that she gets $(AAAA\bot, BAAA\bot)$. By Refuter's claims $\text{EQLV}(AAAA\bot, BAAA\bot) = \text{EQLV}(T_0',U_0')$ and this is smaller than $\text{EQLV}(Ax_1, Bx_1)$ in particular. But $(AAAA\bot, BAAA\bot) = ((Ax_1)\sigma, (Bx_1)\sigma)$ where $\sigma(x_1) = AAA\bot$, and thus $\text{EQLV}(AAAA\bot, BAAA\bot) \geq \text{EQLV}(Ax_1, Bx_1)$ by Prop. 2.4(1). Prover has thus won this play: she has contradicted Refuter's claims; in particular, she was able to create a *basis-instance* (a pair $(T,U) = (E\sigma, F\sigma)$ where $(E,F) \in \text{BASIS}$) which has the same eq-level as (T_i,U_i) for some $i > 0$, according to Refuter's claims.

In general the game is defined as follows:

PROVER-REFUTER GAME (P-R GAME)

1. A det-first-order grammar $\mathcal{G} = (\mathcal{N}, \mathcal{A}, \mathcal{R})$ is given.

2. Prover produces (by "guessing", say) a finite set BASIS of pairs of (graph presentations of regular) terms.

3. An input pair (T_{in}, U_{in}) is given.

4. *Refuter* chooses $(T_0,U_0) \in \text{STARTSET} = \{(T_{in},U_{in})\} \cup \text{BASIS}$ and *claims* that $\text{EQLV}(T_0,U_0) = \text{MINEL}(\text{STARTSET}) < \omega$. (MINEL gives the least eq-level.)

5. For $i = 0,1,2,\ldots$, Phase i is performed, i.e.:

   (a) Prover chooses $k > 0$, and $\text{REG}(T_i,U_i,k)$ is constructed; if $T_i \not\sim_k U_i$ then Prover loses (the play ends).

   (b) Refuter chooses $(T_i',U_i') \in \text{REG}(T_i,U_i,k) \smallsetminus \text{REG}(T_i,U_i,k-1)$ and w_i, $|w_i| = k$, such that $T_i \xrightarrow{w_i} T_i'$, $U_i \xrightarrow{w_i} U_i'$; if there is no such T_i',U_i',w_i (due to dead terms, hence $T_i \sim U_i$), Prover wins. *Refuter claims* that $\text{EQLV}(T_i',U_i') = \text{MINEL}(\text{REG}(T_i,U_i,k))$.

   (c) Prover produces (T_{i+1},U_{i+1}) from (T_i',U_i') as follows:

   - either she puts $T_{i+1} = T_i'$, $U_{i+1} = U_i'$ (*no change*),
   - or she *balances* (see Fig. 8): if she finds σ,σ' such that $(\sigma(x_i),\sigma'(x_i)) \in \text{REG}(T,U,k-1)$ for all $x_i \in \text{SUPP}(\sigma) = \text{SUPP}(\sigma')$, and she presents T_i' as $G\sigma$ then she can (do a *left-balancing*, namely) put $T_{i+1} = G\sigma'$, and $U_{i+1} = U_i'$; symmetrically, if U_i' is $G\sigma'$ then she can (do a *right-balancing*, namely) put $T_{i+1} = T_i'$, and $U_{i+1} = G\sigma$.

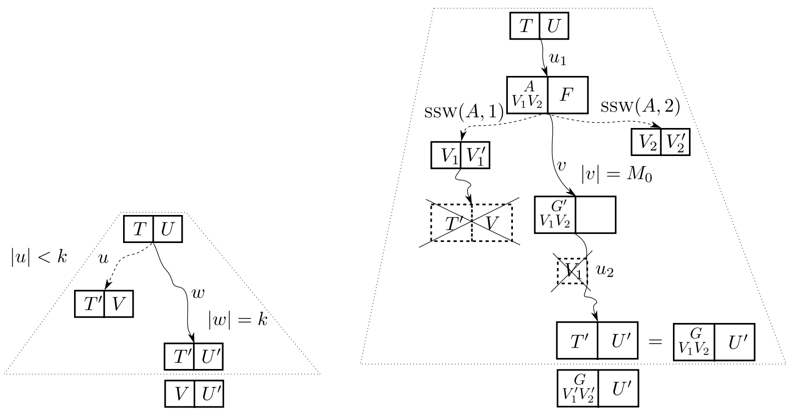


Figure 8: Left: a simplest case of left-balancing; right: another case, when no simplest exists

(Thus $\text{EQLV}(T_{i+1}, U_{i+1}) = \text{EQLV}(T'_i, U'_i)$ if Refuter's claim in 5.b is true. We have $T_{i+1} \sim U_{i+1}$ if $T_i \sim U_i$.)

(d) Prover *either derives a contradiction from Refuter's claims in 4 and 5.b*, by presenting a proof, i.e. a finite algorithmically verifiable sequence of deductions based on proved facts (like Propositions 2.1, 2.2, 2.4), in which case Prover wins, *or lets the play proceed* with Phase $i{+}1$.

By switching Points 2) and 3) we get the *weaker form of the game*; a play then starts with a given instance $\mathcal{G}, T_{in}, U_{in}$ of TRACE-EQ-DET-G. We use the above (stronger) form to stress that BASIS is related to the grammar $\mathcal{G}$ (and is independent of T_{in}, U_{in}).

If $\{(T_{in}, U_{in})\} \cup$ BASIS contains a pair of nonequivalent terms then Refuter can be choosing so that his "least eq-level claims" (in 4. and 5.b) are true; then he must win eventually (Prover cannot show $T_i \sim_k U_i$ for some i and $k > 0$). Since BASIS is finite and Refuter always has finitely many choices when there is his turn, there is an obvious algorithmic aspect which we also capture in the next (soundness) lemma.

Lemma 2.5 *There is an algorithm with the following property: given a det-first order grammar $\mathcal{G}$ and T_{in}, U_{in}, it halts iff there is some* BASIS *such that Prover can force her win for $\mathcal{G}$, T_{in}, U_{in} by using* BASIS *(in the weaker form of the game), in which case $T \sim U$ for all $(T, U) \in \{(T_{in}, U_{in})\} \cup$* BASIS.

Hence the next theorem will be proved once we show (strong) completeness, i.e., that there is some BASIS for each det-first-order grammar $\mathcal{G}$ such that Prover has a winning strategy for any $T_{in} \sim U_{in}$ when using BASIS.

Theorem 2.6 *Trace equivalence of det-first-order grammars (i.e., the problem* TRACE-EQ-DET-G*) is decidable.*

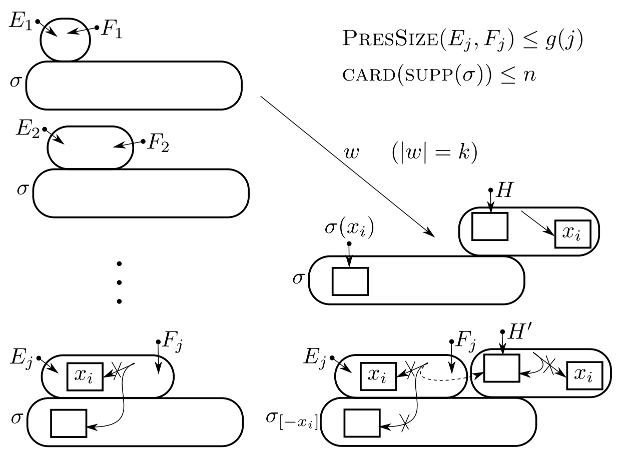


Figure 9: An (n,g)-(sub)sequence (left); decreasing $\text{SUPP}(\sigma)$ by $\{(x_i, H')\}$

Completeness of the Prover-Refuter Game

There are two steps, the first step captured by Lemma 2.7, the second by Lemma 2.9.

For $n \in \mathbb{N}$ and a (nondecreasing) function $g : \mathbb{N} \to \mathbb{N}$, a sequence of pairs of terms is an (n,g)-*sequence* if it can be presented as $(E_1\sigma, F_1\sigma)$, $(E_2\sigma, F_2\sigma)$, $(E_3\sigma, F_3\sigma)$, $\ldots$ where the "heads" satisfy $\text{PRESSIZE}(E_j, F_j) \leq g(j)$ (for $j = 1,2,3,\ldots$) and the cardinality of the support of σ satisfies $\text{CARD}(\text{SUPP}(\sigma)) \leq n$. (See the left column in Fig. 9.) Here $\text{PRESSIZE}(E_j, F_j)$ can be defined as a standard size of the smallest graph presentation of (regular terms) E_j, F_j.

We say that *Prover has an* (n,g)-*strategy* for $\mathcal{G}$ if she can force that the sequence $(T_1, U_1), (T_2, U_2), (T_3, U_3), \ldots$ arising in the phases of the P-R game has an infinite subsequence which is an (n,g)-sequence, in each play where $T_0 \sim U_0$ and the play does not finish with Prover's win in Point 5b or with a repeat. (The basis is irrelevant.) Here by a *repeat* we mean that $(T_j, U_j) = (T_i, U_i)$ for some $j > i$ (which clearly contradicts Refuter's claims).

Lemma 2.7 *If Prover has an* (n,g)*-strategy for a det-first-order grammar* $\mathcal{G}$ *then there is some* BASIS *for* $\mathcal{G}$ *which is sufficient for Prover to force her win for all* $T_{in} \sim U_{in}$.

Fig. 9 sketches the idea. When Prover has an (n,g)-strategy, she can guess a (large) bound $\mathcal{B} \in \mathbb{N}$, depending on $\mathcal{G}, n, g$, and choose $\text{BASIS} = \{(E,F) \mid E \sim F, \text{PRESSIZE}(E,F) \leq \mathcal{B}\}$. (Prover guesses all equivalent pairs upto the chosen presentation-size bound.)

Suppose Prover uses the (n,g)-strategy from $T_0 \sim U_0$ (thus we also have $T_i \sim U_i$ for all i), and a large prefix of an (n,g)-sequence has already arisen (Fig. 9-left). If $E_1 \sim F_1$ (which must be the case when $n = 0$, so when $\text{CARD}(\text{SUPP}(\sigma)) = 0$) then $(E_1\sigma, F_1\sigma)$ is a basis-instance, when $\mathcal{B}$ has been chosen so that $\mathcal{B} \geq g(1)$.

For the case $E_1 \not\sim F_1$ (while $E_1\sigma \sim F_1\sigma$) we use the next easily derivable proposition:

Proposition 2.8 *If* $\text{EQLV}(E,F) = k < \ell = \text{EQLV}(E\sigma, F\sigma)$ *(*$\ell \in \mathbb{N} \cup \{\omega\}$*) then there are some* $x_i \in \text{SUPP}(\sigma)$*,* $H \neq x_i$*, and a word* w*,* $|w| = k$*, such that* $E \xrightarrow{w} x_i$*,* $F \xrightarrow{w} H$ *or* $E \xrightarrow{w} H$*,* $F \xrightarrow{w} x_i$*; moreover,* $\sigma(x_i) \sim_{\ell-k} H\sigma$ *(by Prop. 2.2(1)).*

Fig. 9 sketches that in this case ($E_1 \not\sim F_1$) Prover can soundly use the limit of replacing subterm $\sigma(x_i)$ with $H\sigma$ forever (recall $E_3 = E\sigma\sigma\ldots$ in Fig. 2) so that she transforms a suffix $(E_j\sigma, F_j\sigma)$, $(E_{j+1}\sigma, F_{j+1}\sigma)$, $\ldots$ of the (n,g)-sequence to $(E'_1\sigma', F'_1\sigma')$, $(E'_2\sigma', F'_2\sigma')$, $\ldots$ without changing the eq-levels according to Refuter's claims. This is an $(n-1, g')$-sequence since x_i has been removed from the support of the substitution, and g' is clearly determined by $\mathcal{G}, n, g$ (being independent of T_{in}, U_{in}). Now an inductive reasoning establishes Lemma 2.7.

Lemma 2.9 *For any det-first-order grammar* $\mathcal{G}$*, Prover has an* (n,g)*-strategy (*n*,* g *being determined by* $\mathcal{G}$*).*

The proof of this final lemma is a bit technical though the idea is not much complicated. In a simplest case of left-balancing, as in Fig. 8-left, both terms in the balancing result (V,U) are reachable by at most k moves from U, which is called the (balancing) *pivot* of this concrete balancing step. In fact, we let Prover choose $k = M_1$ in each phase, where M_1 is a number sufficiently bigger than M_0 which was introduced with (shortest) (Y,j)-sink words. In the more complicated case on the right of Fig. 8, the balancing result is composed from the terms which are "shortly" (i.e., by at most M_1 moves) reachable from the pivot U, using a "resthead" G.

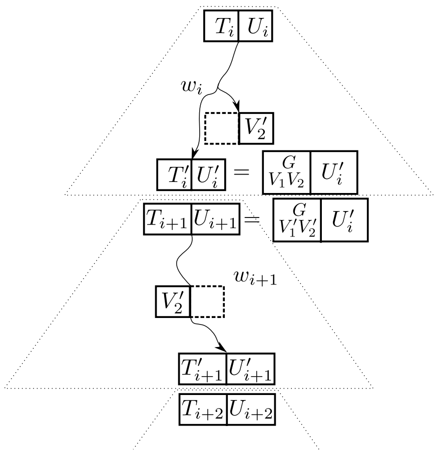


Figure 10: A left balancing phase i followed by a no-change phase $i+1$

We let Prover balance by using only *bounded finite rest-heads* G. If she makes a left-balancing in Phase i, she will not do any right-balancing in Phase $i+1$ but she tries to do a left-balancing here as well. If no left-balancing in Phase $i+1$ is possible, it must mean (the bound on G and M_1 have been chosen so) that the rest-head G gets "erased", i.e. a term which is shortly reachable from the last pivot is exposed (see Fig. 10); in Phase $i+2$ Prover can then balance on any side. (The situation with a right-balancing in Phase i is symmetrical.)

(On the right of Fig. 8, $\textsc{ssw}(A,2)$ means a shortest $(A,2)$-sink word. Using the figure, we could argue for the above "erasing G" claim; the depicted situation cannot arise in the above Phase $i{+}1$ when no left-balancing was possible.)

We can thus see that the pivot of each balancing step, except of the first one, is reachable from the pivot of the previous balancing step (no matter on which side it was). It can be also easily verified (for M_1 and the bound on G) that if only finitely many balancings happen when Prover uses the described strategy for $T_0 \sim U_0$ then a repeat must arise. Otherwise the pivots are on a special infinite path in $\mathcal{L}^{\mathrm{A}}_{\mathcal{G}}$ (recall Fig. 6), and it is a routine to show that either there is a repeat or a suffix of the sequence of balancing results is an (n,g)-sequence, where n,g can be computed from $\mathcal{G}$.

References

[1] S. Böhm, S. Göller, Language Equivalence of Deterministic Real-Time One-Counter Automata Is NL-Complete. In: *MFCS 2011*. LNCS 6907, Springer, 2011, 194–205.

[2] C. H. Broadbent, A. Carayol, C.-H. L. Ong, O. Serre, Recursion Schemes and Logical Reflection. In: *LICS 2010*. IEEE Computer Society, 2010, 120–129.

[3] B. Courcelle, Recursive applicative program schemes. In: J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science, vol. B*. Elsevier, MIT Press, 1990, 459–492.

[4] W. Czerwiński, S. Lasota, Fast equivalence-checking for normed context-free processes. In: *Proc. FSTTCS'10*. LIPIcs 8, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[5] S. Ginsburg, S. A. Greibach, Deterministic Context Free Languages. *Information and Control* **9** (1966) 6, 620–648.

[6] P. Jančar, Decidability of DPDA Language Equivalence via First-Order Grammars. In: *Proc. of Logic in Computer Science (LiCS)*. IEEE Computer Society, 2012.

[7] S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, J. Worrell, On the Complexity of the Equivalence Problem for Probabilistic Automata. In: *FoSSaCS'12*. LNCS 7213, Springer, 2012, 467–481.

[8] A. Kučera, R. Mayr, On the complexity of checking semantic equivalences between pushdown processes and finite-state processes. *Inf. Comput.* **208** (2010) 7, 772–796.

[9] S. Salvati, I. Walukiewicz, Krivine Machines and Higher-Order Schemes. In: *ICALP(2)'11*. LNCS 6756, Springer, 2011, 162–173.

[10] G. Sénizergues, L(A)=L(B)? Decidability Results from Complete Formal Systems. *Theoretical Computer Science* **251** (2001) 1–2, 1–166. (a preliminary version appeared at ICALP'97).

[11] G. Sénizergues, L(A)=L(B)? a Simplified Decidability Proof. *Theoretical Computer Science* **281** (2002) 1–2, 555–608.

[12] G. Sénizergues, The Equivalence Problem for t-Turn DPDA Is Co-NP. In: *ICALP'03*. LNCS 2719, Springer, 2003, 478–489.

[13] G. Sénizergues, The Bisimulation Problem for Equational Graphs of Finite Out-Degree. *SIAM J.Comput.* **34** (2005) 5, 1025–1106. (a preliminary version appeared at FOCS'98).

[14] C. Stirling, Decidability of DPDA Equivalence. *Theoretical Computer Science* **255** (2001) 1–2, 1–31.

[15] C. Stirling, Deciding DPDA Equivalence Is Primitive Recursive. In: *Proc. ICALP'02*. LNCS 2380, Springer, 2002, 821–832.

Grammars of Ungrammatical Strings

Karel Oliva

Institute of the Czech Language ASCR, v. v. i.
Letenská 123/4, Praha 1 – Malá Strana, CZ – 118 51, Czech Republic
oliva@ujc.cas.cz

Abstract

A natural language is usually modelled as a subset of the set T^* of strings (over some set T of terminals) generated by some grammar G. Thus, T^* is divided into two disjoint classes: into grammatical and ungrammatical strings (any string not generated by G is considered ungrammatical). This approach brings along the following problems:

- on the theoretical side, it is impossible to rule out clearly unacceptable yet "theoretically grammatical" strings (e.g., strings with multiple centre self-embeddings, cf. *The cheese the lady the mouse the cat the dog chased caught frightened bought cost 10 £*),
- on the practical side, it impedes systematic build-up of such computational linguistics applications as, e.g., grammar-checkers.

In an attempt to lay a theoretical fundament enabling the solution of these problems, the paper first proposes a tripartition of the stringset into:

- clearly grammatical strings,
- clearly ungrammatical strings,
- strings with unclear ("on the verge"-) grammaticality status

and, based on this, concentrates on

- techniques for systematic discovery and description of clearly ungrammatical strings,
- the impact of the approach onto the theory of grammaticality,
- an overview of simple ideas about applications of the above in building grammar-checkers and rule-based part-of-speech taggers.

Weighted Tree Automata and Tree Transducers can help in Statistical Machine Translation of Natural Languages

Heiko Vogler$^{(A)}$

$^{(A)}$Faculty of Computer Science
Technische Universität Dresden, Germany
`Heiko.Vogler@tu-dresden.de`

Abstract

In this talk I try to illustrate that the concepts of weighted tree automata and weighted tree transducers can be useful in statistical machine translation of natural languages. My apologies: I will not provide a survey on natural language processing (NLP); also, the selection of the references is strongly biased by my personal taste and abilities.

1. Natural Languages, CF Grammars, and Tree Automata

Although N. Chomsky did not succeed to specify the complete English language by means of a formal grammar, he came up with the wonderful formal models which are known as the Chomsky grammars. Among them, the Type-2 Chomsky grammar, or context-free grammar, plays an important role in the formalization of natural languages, albeit that a context-free grammar for English can only be a rough approximation of the perfect English; but this works quite well, in particular, if the domain of discourse is restricted.

There is an old result which connects the world of strings (here: sentences of a natural language) and trees (cf. [29]): (1) the set of derivation trees of a context-free grammar is recognizable, i.e., a tree language which can be recognized by a finite-state tree automaton [8, 28], and (2) a recognizable tree language is the relabeling of the set of derivation trees of a context-free grammar. Via this connection, techniques and results from the theory of tree automata (and tree transducers) [11, 12, 7] can be employed in NLP.

2. Ambiguity and Weighted Tree Automata

Natural languages are ambiguous. Consider the sentence $w =$ *I saw her duck.* One can identify several meanings of w, a harmless one involving an animal, one which might bring pain to her back, and another cruel one. Since each meaning leads to a different derivation tree, we might view the set of all derivation trees as a representation of the meanings of w.

How to handle such ambiguities? The trick is to associate a probability with each of the possible meanings, i.e., derivation trees; then, “understanding” or: parsing the sentence *I* *saw her duck* yields a function (called *weighted tree language*) which maps every derivation tree to its probability. And, if the sentence is generated by a context-free grammar, then the weighted tree language is *recognizable* by a *weighted tree automaton* [4, 1]. In other words, such a weighted tree automaton is a finite representation of the set of all weighted meanings of the given sentence. This way of modelling languages is remarkable, because

- due to the finiteness of the representation, we can now algorithmically manipulate, transform, or process the set of meanings of sentences, e.g.,
  - (a) we can apply parsing algorithms based on tree automata [23] or
  - (b) we might select the n best (most probable) of them [14], and
- there is a rich theory of recognizable weighted tree languages [5, 10] which can be exploited or adapted appropriately, e.g., using the input and output products, see later.

3. Translation of Natural Languages

Imagine that you want to translate *I saw her duck* into French. For a long time, string- or phrase-based approaches where dominant when dealing with this problem (cf. e.g. [6, 25, 24]). It was advocated in [30] to exploit the (derivation) tree structure of the given sentence for this task, and in [17, 16] top-down finite-state tree transducers were suggested as formal model for the specification of this tree-to-tree transformation. Clearly, the transformation process usually also leads to different translations of one given derivation tree. Again we can use probabilities to disambiguate. This is captured by the concept of *weighted tree transducers* [18, 9]. Also for weighted tree transducers there is a rich theory [20, 21, 10].

4. Statistical Machine Translation

Assume that we already have a tree automaton which specifies the set of derivation trees of all English sentences. How do we obtain the probabilities for the transitions of that automaton? Or: assume that we have already a top-down tree transducer. How to obtain the probabilities for its rules? One technology which can help is *statistical machine translation* (SMT).

> *"Statistical machine translation (SMT) treats the translation of natural languages as a machine learning problem. By examining many samples of human-produced translations, SMT algorithms automatically learn how to translate."* [19]

A huge source of high-level quality samples are the Hansards of parliament speeches (e.g., of the Canadian parliament with English-French, or those of the European Union). Such samples are called (string or tree) corpora.

5. Rule Extraction

But how to get the transitions for the tree automaton and the rules for the tree transducers? Well, this is done by "reading off "context-free grammar rules from large corpora of derivation trees. The latter are either given in the form of a manually created Treebank (such as the Penn Treebank for English or the Tiger Treebank for German), or they are obtained by applying a parser to English sentences (Berkeley parser [26, 27], Stanford parser [15]); from these context-free grammars rules one can construct a finite-state tree automaton. The rules of a top-down tree transducer can be obtained by the rule extraction algorithm shown in [13].

6. Decoding using Input and Output Product

We now have a language model of English (in the form of a weighted tree automaton $\mathcal{A}$), a translation model from English to French (in the form of a weighted tree transducer $\mathcal{M}$), and an English sentence (w =*I saw her duck.*). How can we translate or: *decode* this into French? Here we can exploit the technical results from the theory of weighted tree automata and weighted tree transducers:

1. We parse w according to $\mathcal{A}$ by using the approach of [23] (the latter is based on the technique of [3, Sect.8]); this yields a weighted tree automaton $\mathcal{A}'$ which recognizes exactly the set of all derivation trees of w with their probabilites,
2. we construct the input product of $\mathcal{A}'$ and $\mathcal{M}$ using a construction of [22] (the latter is based on [2, p.195]); this yields essentially a weighted tree automaton $\mathcal{B}'$ which recognizes exactly the set of all derivations trees of all the translations of w, and
3. we might wish to apply the n best algorithm [14] to $\mathcal{B}'$ in order to obtain the n best translations of w (in the form of derivation trees), and finally consider the frontier of these derivation trees.

References

[1] A. ALEXANDRAKIS, S. BOZAPALIDIS, Weighted grammars and Kleene's theorem. *Inform. Process. Lett.* **24** (1987) 1, 1–4.

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

[3] Y. BAR–HILLEL, M. PERLES, E. SHAMIR, On formal properties of simple phrase structure grammars. *Z. Phonetik. Sprach. Komm.* **14** (1961), 143–172.

[4] J. BERSTEL, C. REUTENAUER, Recognizable formal power series on trees. *Theoret. Comput. Sci.* **18** (1982) 2, 115–148.

[5] B. BORCHARDT, *The Theory of Recognizable Tree Series*. Verlag für Wissenschaft und Forschung, 2005. (Ph.D. thesis, 2004, TU Dresden, Germany).

[6] P. BROWN, V. D. PIETRA, S. D. PIETRA, R. MERCER, The mathematics of statistical machine translation: parameter estimation. *Comput. Linguist.* **19** (1993) 2, 263–311.

[7] H. COMON, M. DAUCHET, R. GILLERON, C. LÖDING, F. JACQUEMARD, D. LUGIEZ, S. TISON, M. TOMMASI, Tree Automata Techniques and Applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007.

[8] J. DONER, Tree Acceptors and Some of Their Applications. *J. Comput. System Sci.* **4** (1970), 406–451.

[9] J. ENGELFRIET, Z. FÜLÖP, H. VOGLER, Bottom-up and Top-down Tree Series Transformations. *J. Autom. Lang. Comb.* **7** (2002), 11–70.

[10] Z. FÜLÖP, H. VOGLER, Weighted tree automata and tree transducers. In: M. DROSTE, W. KUICH, H. VOGLER (eds.), *Handbook of Weighted Automata*. chapter 9, Springer-Verlag, 2009.

[11] F. GÉCSEG, M. STEINBY, *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

[12] F. GÉCSEG, M. STEINBY, Tree Languages. In: G. ROZENBERG, A. SALOMAA (eds.), *Handbook of Formal Languages*. 3. chapter 1, Springer-Verlag, 1997, 1–68.

[13] J. GRAEHL, K. KNIGHT, Training tree transducers. In: *HLT-NAACL 2004, Boston, Massachusetts, USA, May 2 - 7. Association for Computational Linguistics*. 2004, 105–112.

[14] L. HUANG, D. CHIANG, Better k-best Parsing. In: *Proceedings of the Ninth International Workshop on Parsing Technology*. Association for Computational Linguistics, Vancouver, British Columbia, 2005, 53–64.

[15] D. KLEIN, C. MANNING, Accurate Unlexicalized Parsing. In: *ACL*. 2003.

[16] K. KNIGHT, Capturing practical natural language transformations. *Machine Translation* **21** (2007), 121–133.

[17] K. KNIGHT, J. GRAEHL, An overview of probabilistic tree transducers for natural language processing. In: *Computational Linguistics and Intelligent Text Processing, CICLing 2006*. Lecture Notes in Comput. Sci. 3406, Springer-Verlag, 2005, 1–24.

[18] W. KUICH, Tree Transducers and Formal Tree Series. *Acta Cybernet.* **14** (1999), 135–149.

[19] A. LOPEZ, Statistical Machine Translation. *ACM Computing Surveys* **40(3)** (2008), 8:1–8:9.

[20] A. MALETTI, *The Power of Tree Series Transducers*. Der Andere Verlag, Tönning, Lübeck und Marburg, 2006. (Ph.D. thesis, 2006, TU Dresden, Germany).

[21] A. MALETTI, Compositions of Extended Top-down Tree Transducers. *Inf. Comput.* **206** (2008) 9–10, 1187–1196.

[22] A. MALETTI, Input and output products for weighted extended top-down tree transducers. In: Y. GAO, S. YU (eds.), *Proc. Developments in Language Theory*. Lecture Notes in Comput. Sci. 6224, Springer-Verlag, 2010, 316–327.

[23] A. MALETTI, G. SATTA, Parsing Algorithms based on Tree Automata. In: *Proc. 11th Int. Conf. Parsing Technologies*. Association for Computational Linguistics, 2009, 1–12.

[24] M. MOHRI, Weighted automata algorithms. In: M. DROSTE, W. KUICH, H. VOGLER (eds.), *Handbook of Weighted Automata*. chapter 6, Springer-Verlag, 2009, 213–254.

[25] M. MOHRI, F. PEREIRA, M. RILEY, Weighted Automata in Text and Speech Processing. In: W. WAHLSTER (ed.), *ECAI 96. 12th European Conference on Artificial Intelligence*. John Wiley & Sons, Ltd., 1996, 1–5.

[26] S. PETROV, L. BARRETT, R. THIBAUX, D. KLEIN, Learning Accurate, Compact, and Interpretable Tree Annotation. In: *COLING-ACL*. 2006.

[27] S. PETROV, D. KLEIN, Improved Inference for Unlexicalized Parsing. In: *HLT-NAACL*. 2007.

[28] J. THATCHER, J. WRIGHT, Generalized finite automata theory with an application to a decision problem of second-order logic. *Math. Syst. Theory* **2** (1968) 1, 57–81.

[29] J. W. THATCHER, Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *J. Comput. Syst. Sci.* **1** (1967) 4, 317–322.

[30] K. YAMADA, K. KNIGHT, A syntax-based statistical translation model. In: *Proc. of 39th Annual Meeting of the Assoc. Computational Linguistics*. 2001, 523–530.

Machine Translation using Dependency Trees

Zdeněk Žabokrtský

Charles University in Prague, Faculty of Mathematics and Physics,
Institute of Formal and Applied Linguistics, Malostranské náměstí 25, 118 00, Praha 1
`zabokrtsky@ufal.mff.cuni.cz`

Abstract

The present work focuses on using tree-shaped syntactic structures as an intermediate sentence representation in an experimental English-Czech machine translation system.

1. Introduction

Natural Language Processing (NLP) is a multidisciplinary field combining computer science, mathematics and linguistics, whose main aim is to allow computers to work with information expressed in human (natural) language.

The history of NLP goes back to 1950s. Early NLP systems were based on hand-written rules founded by linguistic intuitions. However, roughly two decades ago the growing availability of language data (especially textual corpora) and increasing capabilities of computer systems lead to a revolution in NLP: the field became dominated by data-driven approaches, often based on probabilistic modeling and machine learning.

In such data-driven scenario, the role of human experts was moved from designing rules rather to (i) preparing training data enriched with linguistically relevant information (usually by manual annotation), (ii) choice of an adequate probabilistic model, proposing features (various indicators potentially useful for making the desired predictions), and (iii) specifying an objective (evaluation) function. Optimization of the decision process (such as searching for optimal feature weights) is then entirely left to the learning algorithm.

Nowadays, researched NLP tasks range from relatively simple ones (like sentence segmentation, language identification), through tasks which already need a higher level of abstraction (such as morphological analysis, part-of-speech tagging, parsing, named entity recognition, coreference resolution, word sense disambiguation, sentiment analysis, natural language generation), to highly complex systems (machine translation (MT), automatic summarization, or question answering). The importance of (and demand for) such tasks increases along with the rapidly growing amount of textual information available on the Internet.

Our experiments with MT are implemented within an in-house modular software framework for developing NLP applications, which is called Treex (formerly TectoMT, [14]) and which is being developed at our institute since 2005. The framework allows flexible integration and pipelining of NLP tools (for various purposes), as will be illustrated below on the application of MT.

1.1. Linguistic Background

Natural language is an immensely complicated phenomenon. Modeling the language in its entirety would be extremely complex, therefore its description is often decomposed into several subsequent layers (levels). There is no broadly accepted consensus on details concerning the individual levels, however, the layers typically roughly correspond to the following scale: phonetics, phonology, morphology, syntax, semantics, and pragmatics.

One of such stratificational hypotheses is Functional Generative Description (FGD), developed by Petr Sgall and his colleagues in Prague since the 1960s [11]. FGD was used with certain modifications as the theoretical framework underlying the Prague Dependency Treebank [3], which is a manually annotated corpus of Czech newspaper texts from the 1990s. PDT in version 2.0 adds three layers of linguistic annotation to the original texts:

1. **morphological layer (m-layer)** – each sentence is tokenized and each token is annotated with a lemma (e.g., nominative singular for nouns) and morphological tag (describing morphological categories such as part of speech, number, and tense).

2. **analytical layer (a-layer)** – each sentence is represented as a shallow-syntax dependency tree (a-tree). There is one-to-one correspondence between m-layer tokens and a-layer nodes (a-nodes). Each a-node is annotated with the so-called *analytical function*, which represents basic dependency roles such as subject or attribute.

3. **tectogrammatical layer (t-layer)** - each sentence is represented as a deep-syntax dependency tree (t-tree). Autosemantic (meaningful) words are represented as t-layer nodes (t-nodes). Information conveyed by functional words (such as auxiliary verbs, prepositions and subordinating conjunctions) is represented by attributes of t-nodes. Most important attributes of t-nodes are: tectogrammatical lemma, functor (which represents the semantic value of syntactic dependency relation) and a set of grammatemes (e.g. tense, number, verb modality, deontic modality, negation). Edges in t-trees represent semantic dependencies, except for special cases such as coordination.

This annotation scheme has been adopted and further modified in Treex. Treex also profits from the technology developed during the PDT project, especially from the existence of the highly customizable tree editor TrEd, which is used as the main visualization tool in Treex, and from the XML-based file format PML (Prague Markup Language, [10]), which is used as the main data format in Treex.

1.2. Contemporary Machine Translation

MT is a notoriously hard problem and it is studied by a broad research field nowadays: every year there are several conferences, workshops and tutorials dedicated to it (or even to its subfields). It goes beyond the scope of this work even to mention all the contemporary approaches to MT, but several elaborate surveys of current approaches to MT are already available to the reader elsewhere, e.g. in [7].

A distinction is usually made between two MT paradigms: rule-based MT (RBMT) and statistical MT (SMT). RBMT systems are dependent on the availability of linguistic knowledge (such as grammar rules and dictionaries), while SMT systems require human-translated parallel text, from which they extract the translation knowledge automatically.

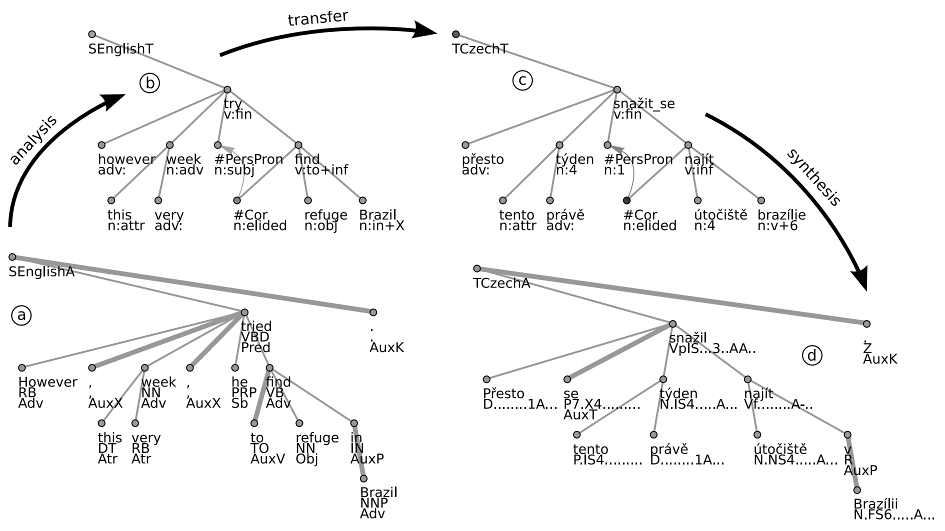


Figure 1: Analysis-transfer-synthesis translation scenario in Treex applied on the English sentence *"However, this very week, he tried to find refuge in Brazil."*, leading to the Czech translation *"Přesto se tento právě týden snažil najít útočiště v Brazílii."*. Thick edges indicate functional and autosemantic a-nodes to be merged.

Nowadays, the most popular representatives of the second group are phrase-based systems (in which the term 'phrase' stands simply for a sequence of words, not necessarily corresponding to phrases in constituent syntax), e.g. [5], derived from the IBM models [2].

Even if phrase-based systems have more or less dominated the field in the recent years, their translation quality is still far from perfect. Therefore we believe it makes sense to investigate also alternative approaches.

MT implemented in Treex lies somewhere between the two main paradigms. Like in RBMS, sentence representations used in Treex are linguistically interpretable. However, the most important decisions during the translation process are made by statistical models like in SMT, not by rules.

2. English-Czech Translation Step-by-Step

The translation scenario implemented in Treex composes of three steps: (1) analysis of the input sentences up to tectogrammatical layer of abstraction, (2) transfer of the abstract representation to the target language, and (3) synthesis (generating) of sentences in the target language. See an example in Figure 1.

Analysis. The analysis step can be decomposed into three phases corresponding to morphological, analytical and tectogrammatical analysis.

In the morphological phase, a text to be translated is segmented into sentences and each sentence is tokenized (segmented into words and punctuation marks). Tokens are tagged with part of speech and other morphological categories by the Morce tagger [12], and lemmatized.

In the analytical phase, each sentence is parsed using the dependency parser [9] based on Maximum Spanning Tree algorithm, which results in an analytical tree for each sentence. Then the analytical trees are converted to the tectogrammatical trees. Each autosemantic word with its associated functional words is collapsed into a single tectogrammatical node, labeled with lemma, functor (semantic role), formeme,[1] and semantically indispensable morphologically categories (such as tense with verbs and number with nouns, but not number with verbs as it is only imposed by subject-predicate agreement). Coreference of pronouns is also resolved and tectogrammatical nodes are enriched with information on named entities (such as the distinction between location, person and organization).

Transfer. The transfer phase follows, whose most difficult part consists in labeling the tree with target-language lemmas and formemes. Changes of tree topology and of other attributes[2] are required relatively infrequently.

Our model for choosing the right target-language lemmas and formemes in inspired by Noisy Channel Model which is the standard approach in the contemporary SMT and which combines a translation model and a language model of the target language. In other words, one should not rely only on the information on how faithfully the meaning is transfered by some translation equivalent, but also the additional model can be used which estimates how well some translation equivalent fits to the surrounding context.

Unlike in the mainstream SMT, in tectogrammatical transfer we do not use this idea for linear structures, but for tectogrammatical trees.[3] So the translation model estimates the probability of source and target lemma pair, while the language tree model estimates the probability of a lemma given its parent. The globally optimal tree labelling is then revealed by the tree-modified Viterbi algorithm [13].

Originally, we estimated the translation model simply by using pair frequencies extracted from English-Czech parallel data. A significant improvement was reached after replacing such model by Maximum Entropy model. In the model, we employed a wide range of features resulting from the source-side analysis. The weights were optimized using training data extracted from the CzEng parallel treebank [1], which contains roughly 6 million English-Czech pairs of analyzed and aligned sentences.

[1]Formemes specify how tectogrammatical nodes are realized in the surface sentence shape. For instance, `n:subj` stands for semantic noun in the subject position, `n:for+X` for semantic noun with preposition *for*, `v:because+fin` for semantic verb in a subordinating clause introduced by the conjunction *because*, `adj:attr` for semantic adjective in attributive position. Formemes do not constitute a genuine tectogrammatical component as they are not oriented semantically (but rather morphologically and syntactically). However, they have been added to t-trees in Treex as they facilitate the transfer.

[2]For instance, number of nouns must be changed to plural if the selected target Czech lemma is a plurale tantum. Similarly, verb tense must be predicted if an English infinitive or gerund verb form is translated to a finite verb form.

[3]We believe that the potential contribution of tectogrammatical layer of language representation for MT is the following: it abstracts from many language-specific phenomena (which could reduce the notorious data-sparsity problem) and offers a natural factorization of the translation task (which could be useful for formulating independence assumptions when building probabilistic models). Of course, it is not clear whether these properties can ever outbalance the disadvantages, especially cumulation and interference of errors made on different layers, considerable technical complexity, and the need for detailed linguistic insight. In our opinion, this question still remains open.

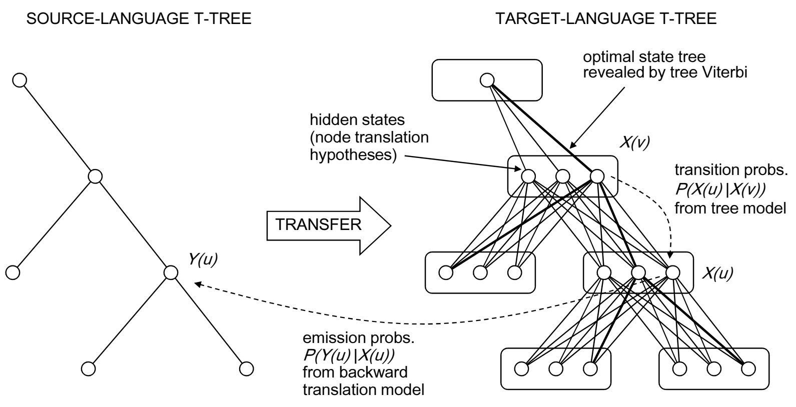

SOURCE-LANGUAGE T-TREE
TARGET-LANGUAGE T-TREE
optimal state tree revealed by tree Viterbi
hidden states (node translation hypotheses)
X(v)
transition probs. P(X(u)|X(v)) from tree model
TRANSFER
Y(u)
X(u)
emission probs. P(Y(u)|X(u)) from backward translation model

Figure 2: Tectogrammatical transfer implemented as Hidden Markov Tree Model.

Synthesis. Finally, surface sentence shape is synthesized from the tectogrammatical tree, which is basically a reverse operation for the analysis: adding punctuation and functional words, spreading morphological categories according to grammatical agreement, performing inflection (using Czech morphology database [4]), arranging word order etc.

3. Final Remarks

As for evaluating MT quality, there are two general methods: (1) the quality can be judged by humans (either using a set of criteria such as grammaticality and intelligibility, or relatively by comparing outputs of different MT systems), or (2) the quality can be estimated by automatic metrics, which usually measure some form of string-wise overlap with one or more reference (human-made) translations. Both types of evaluation are used regularly during the development of our MT system and confirm that performance increases every year.

Even if tectogrammatical translation is considered as the main application of Treex, Treex has been used for a number of other research purposes as well. For instance, it has been used for developing alternative MT quality measures in [6], for improving outputs of other MT systems by grammatical post-processing in [8], and for building linguistic data resources such as the Czech-English parallel corpus CzEng [1].

Last but not least, Treex is used for teaching purposes in our institute. Undergraduate students are supposed to develop their own modules for morphological and syntactic analysis for foreign languages of their choice. Not only that the existence of Treex enables the students to make very fast progress, but their contributions are accumulated in the Treex Subversion repository too, which enlarges the repertory of languages treatable by Treex.

When thinking about a more distant future of MT, an exciting question arises about the future relationship of linguistically interpretable approaches (like that of Treex) and purely statistical phrase-based approaches. Promising results of [8], which uses Treex for improving the output of a phrase-based system and reaches the state-of-the-art MT quality in English-Czech MT, show that combinations of both approaches might be viable.

References

[1] O. Bojar, M. Janíček, Z. Žabokrtský, P. Češka, P. Beňa, CzEng 0.7: Parallel Corpus with Community-Supplied Translations. In: *Proceedings of the Sixth International Language Resources and Evaluation*. ELRA, Marrakech, Morocco, 2008.

[2] P. E. Brown, V. J. Della Pietra, S. A. Della Pietra, R. L. Mercer, The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics* (1993).

[3] J. Hajič, E. Hajičová, J. Panevová, P. Sgall, P. Pajas, J. Štěpánek, J. Havelka, M. Mikulová, Prague Dependency Treebank 2.0. Linguistic Data Consortium, LDC Catalog No.: LDC2006T01, Philadelphia, 2006.

[4] J. Hajič, *Disambiguation of Rich Inflection – Computational Morphology of Czech*. Charles University – The Karolinum Press, Prague, 2004.

[5] P. Koehn, et al., Moses: Open Source Toolkit for Statistical Machine Translation. In: *Proceedings of the Demo and Poster Sessions, 45th Annual Meeting of ACL*. Association for Computational Linguistics, Prague, Czech Republic, 2007, 177–180.

[6] K. Kos, O. Bojar, Evaluation of Machine Translation Metrics for Czech as the Target Language. *Prague Bulletin of Mathematical Linguistics* **92** (2009).

[7] A. Lopez, *A Survey of Statistical Machine Translation*. Technical report, Institute for Advanced Computer Studies, University of Maryland, 2007.

[8] D. Mareček, R. Rosa, P. Galuščáková, O. Bojar, Two-step translation with grammatical post-processing. In: *Proceedings of the 6th Workshop on Statistical Machine Translation*. Association for Computational Linguistics, Edinburgh, Scotland, 2011, 426–432.

[9] R. McDonald, F. Pereira, K. Ribarov, J. Hajič, Non-Projective Dependency Parsing using Spanning Tree Algorithms. In: *Proceedings of Human Langauge Technology Conference and Conference on Empirical Methods in Natural Language Processing*. Vancouver, BC, Canada, 2005, 523–530.

[10] P. Pajas, J. Štěpánek, Recent Advances in a Feature-Rich Framework for Treebank Annotation. In: *Proceedings of The 22nd International Conference on Computational Linguistics*. 2, Manchester, UK, 2008, 673–680.

[11] P. Sgall, E. Hajičová, J. Panevová, *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. D. Reidel Publishing Company, Dordrecht, 1986.

[12] D. Spoustová, J. Hajič, J. Votrubec, P. Krbec, P. Květoň, The Best of Two Worlds: Cooperation of Statistical and Rule-Based Taggers for Czech. In: *Proceedings of the Workshop on Balto-Slavonic Natural Language Processing, ACL 2007*. Praha, 2007, 67–74.

[13] Z. Žabokrtský, M. Popel, Hidden Markov Tree Model in Dependency-based Machine Translation. In: *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics*. 2009.

[14] Z. Žabokrtský, J. Ptáček, P. Pajas, TectoMT: Highly Modular MT System with Tectogrammatics Used as Transfer Layer. In: *Proceedings of the 3rd Workshop on Statistical Machine Translation, ACL*. 2008.

Time-Varying Sequential P Systems

A. Alhazov$^{(B)}$ R. Freund$^{(A)}$ H. Heikenwälder$^{(A)}$ M. Oswald$^{(A)}$
Yu. Rogozhin$^{(B)}$ S. Verlan$^{(C)}$

$^{(A)}$ Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
`{rudi,hilbert,marion}@emcc.at`

$^{(B)}$ Institute of Mathematics and Computer Science, Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD-2028, Moldova
`{artiom,rogozhin}@math.md`

$^{(C)}$ LACL, Département Informatique, Université Paris Est
61, av. Général de Gaulle, 94010 Créteil, France
`verlan@univ-paris12.fr`

Abstract

In this article we introduce the regulating mechanism of control languages for the application of rules assigned to the membranes of a sequential P system. Computational completeness can only be achieved when allowing the system to have no rules applicable for a bounded number of steps; in this case we only need one membrane and periodically available sets of non-cooperative rules, i.e., time-varying sequential P systems. Only Parikh sets of matrix languages can be obtained if the terminal result has to be taken as soon as the system cannot apply any rule anymore.

1. Introduction

P systems are formal models derived from the functioning of living cells, closely related to multiset rewriting. We refer to [11], [12], and to the web page [16] for more details on P systems. In this article, we investigate the power of controlling the availability of the sets of rules assigned to the membranes of a (static) P system by a regular control language L, especially for languages L of the form $\{w\}^*$, which leads to the notion of a *time-varying* P system where the set of rules available at each membrane varies periodically with time.

The notion of the time-varying controlled application of rules comes from the area of regulated rewriting; comprehensive overviews of this area can be found in [3], [5], and [6]); periodically time-varying grammars were already mentioned in [15] following the work on time-varying automata [14]. This notion was also considered in the area of Lindenmayer systems, corresponding to controlled tabled Lindenmayer systems, with the tables being used periodically (see [9]). We can also interpret these systems as counterparts of cooperating distributed grammar systems ([2, 4]) with the order of enabling the components controlled by a graph having the shape of a ring. In the field of DNA computing several models using the variation in time of the set of available rules were considered, e.g., see [10] and [8].

2. Preliminary Definitions and Well-Known Results

For details of formal language theory the reader is referred to the monographs and handbooks in this area as [3] and [13].

A *grammar* G of type X is a construct $(O,O_T,A,P,\Longrightarrow_G)$ where O is a set of *objects*, $O_T \subseteq O$ is a set of *terminal objects*, $A \in O$ is the *axiom*, and P is a finite set of *rules* of type X. Each rule $p \in P$ induces a relation $\Longrightarrow_p \subseteq O \times O$; p is called *applicable* to an object $x \in O$ if and only if there exists at least one object $y \in O$ such that $(x,y) \in \ \Longrightarrow_p$; we also write $x \Longrightarrow_p y$. The derivation relation $\Longrightarrow_G$ is the union of all $\Longrightarrow_p$, i.e., $\Longrightarrow_G = \cup_{p\in P} \Longrightarrow_p$. The reflexive and transitive closure of $\Longrightarrow_G$ is denoted by $\overset{*}{\Longrightarrow}_G$. The *language generated by* G is the set of all terminal objects derivable from the axiom, i.e., $L(G) = \left\{ v \in O_T \mid A \overset{*}{\Longrightarrow}_G v \right\}$. The family of languages generated by grammars of type X is denoted by $\mathcal{L}(X)$.

A *string grammar* G_S is represented as $\left((N \cup T)^*, T^*, w, P, \Longrightarrow_{G_S}\right)$ where N is the alphabet of *non-terminal symbols*, T is the alphabet of *terminal symbols*, $N \cap T = \emptyset$, $w \in (N \cup T)^+$, P is a finite set of *rules* of the form $u \to v$ with $u \in V^+$ and $v \in V^*$, with $V := N \cup T$; the derivation relation for $u \to v \in P$ is defined by $xuy \Longrightarrow_{u\to v} xvy$ for all $x,y \in V^*$, thus yielding the well-known derivation relation $\Longrightarrow_{G_S}$ for the string grammar G_S. As special types of string grammars we consider string grammars with arbitrary rules, context-free rules of the form $A \to v$ with $A \in N$ and $v \in V^*$, and (right-)regular rules of the form $A \to v$ with $A \in N$ and $v \in TN \cup \{\lambda\}$. In the following, we shall also use the common notation $G_S = (N,T,w,P)$ instead, too. The corresponding types of grammars are denoted by ARB, CF, and REG, thus yielding the families of languages $\mathcal{L}(ARB)$, i.e., the family of recursively enumerable languages RE, as well as $\mathcal{L}(CF)$, and $\mathcal{L}(REG)$, i.e., the families of context-free, and regular languages (also denoted by REG), respectively.

The subfamily of REG only consisting of 1-star languages of the form W^* for some finite set of strings W is denoted by REG^{1*}; to be more specific, we also consider $REG^{1*}(k,p)$ consisting of all 1-star languages of the form W^* with k being the maximum number of strings in W and p being the maximum lengths of the strings in W. If $W = \{w\}$ for a singleton w, we call the set $\{w\}^*$ *periodic* and $|w|$ its *period*; thus, $REG^{1*}(1,p)$ denotes the family of all periodic sets with period at most p. If any of the numbers k or p may be arbitrarily large, we replace it by $*$.

A *multiset grammar [7]* G_m is of the form $\left((N \cup T)^\circ, T^\circ, w, P, \Longrightarrow_{G_m}\right)$ where N is the alphabet of *non-terminal symbols*, T is the alphabet of *terminal symbols*, $N \cap T = \emptyset$, w is a non-empty multiset over V, $V := N \cup T$, and P is a (finite) set of multiset rules yielding a derivation relation $\Longrightarrow_{G_m}$ on the multisets over V; the application of the rule $u \to v$ to a multiset x has the effect of replacing the multiset u contained in x by the multiset v. For the multiset grammar G_m we also write $(N,T,w,P,\Longrightarrow_{G_m})$.

As special types of multiset grammars we consider multiset grammars with *arbitrary* rules, *context-free* rules of the form $A \to v$ with $A \in N$ and $v \in V^\circ$, and *regular* rules of the form $A \to v$ with $A \in N$ and $v \in T^\circ N \cup T^\circ$; the corresponding types X of multiset grammars are denoted by $mARB$, mCF, and $mREG$, thus yielding the families of multiset languages $\mathcal{L}(X)$. Even with arbitrary multiset rules, it is not possible to get $Ps(\mathcal{L}(ARB))$ [7]:

$$\begin{aligned} Ps(\mathcal{L}(REG)) &= \mathcal{L}(mREG) = \mathcal{L}(mCF) = Ps(\mathcal{L}(CF)) \\ &\subsetneqq \mathcal{L}(mARB) \subsetneqq Ps(\mathcal{L}(ARB)). \end{aligned}$$

A *graph-controlled grammar* of type X is a construct $G_{GC} = (G, g, H_i, H_f, \Longrightarrow_{GC})$ where $G = (O, O_T, w, P, \Longrightarrow_G)$ is a grammar of type X; $g = (H, E, K)$ is a labeled graph where H is the set of node labels identifying the nodes of the graph in a one-to-one manner, $E \subseteq H \times \{Y, N\} \times H$ is the set of edges labeled by Y or N, $K : H \to 2^P$ is a function assigning a subset of P to each node of g; $H_i \subseteq H$ is the set of initial labels, and $H_f \subseteq H$ is the set of final labels. The derivation relation $\Longrightarrow_{GC}$ is defined based on $\Longrightarrow_G$ and the control graph g: For any $i, j \in H$ and any $u, v \in O$, $(u, i) \Longrightarrow_{GC} (v, j)$ if and only if either

- $u \Longrightarrow_p v$ by some rule $p \in K(i)$ and $(i, Y, j) \in E$ *(success case)*, or
- $u = v$, no $p \in K(i)$ is applicable to u, and $(i, N, j) \in E$ *(failure case)*.

The language generated by G_{GC} is defined by

$$L(G_{GC}) = \left\{ v \in O_T \mid (w, i) \Longrightarrow_{G_{GC}}^{*} (v, j),\ i \in H_i, j \in H_f \right\}.$$

If $H_i = H_f = H$, then G_{GC} is called a *programmed grammar*. The families of languages generated by graph-controlled and programmed grammars of type X are denoted by $\mathcal{L}(X\text{-}GC_{ac})$ and $\mathcal{L}(X\text{-}P_{ac})$, respectively. If the set E contains no edges of the form (i, N, j), then the graph-controlled grammar G_{GC} is said to be *without appearance checking*; the corresponding families of languages are denoted by $\mathcal{L}(X\text{-}GC)$ and $\mathcal{L}(X\text{-}P)$, respectively. If $(i, Y, j) \in E$ if and only if $(i, N, j) \in E$ for all $i, j \in H$, then G_{GC} is said to be a graph-controlled grammar or programmed grammar *with unconditional transfer*, the corresponding families of languages are denoted by $\mathcal{L}(X\text{-}GC_{ut})$ and $\mathcal{L}(X\text{-}P_{ut})$, respectively.

A *matrix grammar* of type X is a construct $G_M = \left(G, M, F, \Longrightarrow_{G_M}\right)$ where $G = (O, O_T, w, P, \Longrightarrow_G)$ is a grammar of type X, M is a finite set of sequences of the form $(p_1, \dots, p_n)$, $n \geq 1$, of rules in P, and $F \subseteq P$. For $w, z \in O$ we write $w \Longrightarrow_{G_M} z$ if there are a matrix $(p_1, \dots, p_n)$ in M and objects $w_i \in O$, $1 \leq i \leq n+1$, such that $w = w_1$, $z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i \Longrightarrow_G w_{i+1}$ or $w_i = w_{i+1}$, p_i is not applicable to w_i, and $p_i \in F$.

$L(G_M) = \left\{ v \in O_T \mid w \Longrightarrow_{G_M}^{*} v \right\}$ is the language generated by G_M. The family of languages generated by matrix grammars of type X is denoted by $\mathcal{L}(X\text{-}MAT_{ac})$. If the set F is empty (or if $F = P$), then the grammar is said to be *without appearance checking (with unconditional control)*; the corresponding family of languages is denoted by $\mathcal{L}(X\text{-}MAT)$ $(\mathcal{L}(X\text{-}MAT_{ut}))$.

A *grammar with regular control and appearance checking* is a construct $G_C = (G, H_C, L, F)$ where $G = (O, O_T, w, P, \Longrightarrow_G)$ is a grammar of type X and L is a regular language over H_C, where H_C is the set of labels identifying the subsets of productions from P in a one-to-one manner (H_C is a bijective function on 2^P), and $F \subseteq H_C$. The language generated by G_C consists of all terminal objects z such that there exist a string $H_C(P_1) \cdots H_C(P_n) \in L$ as well as objects $w_i \in O$, $1 \leq i \leq n+1$, such that $w = w_1$, $z = w_{n+1}$, and, for all $1 \leq i \leq n$, either

- $w_i \Longrightarrow_G w_{i+1}$ by some production from P_i or
- $w_i = w_{i+1}$, no production from P_i is applicable to w_i, and $H_C(P_i) \in F$.

The model of grammars with regular control is closely related with the model of graph-controlled grammars in the sense that the control graph corresponds to the deterministic finite automaton accepting L. Hence, we may also speak of a *grammar with regular control and without appearance checking* if $F=\emptyset$, and if $F=H_C$ then G_C is said to be a *grammar with regular control and unconditional transfer.* The corresponding families of languages are denoted by $\mathcal{L}(X\text{-}C(REG)_{ac})$, $\mathcal{L}(X\text{-}C(REG))$, and $\mathcal{L}(X\text{-}C(REG)_{ut})$.

Obviously, the control languages can also be taken from another family of languages Y, e.g., $\mathcal{L}(CF)$, thus yielding the families $\mathcal{L}(X\text{-}C(Y)_{ac})$, etc., but in this paper we shall restrict ourselves to the cases $Y=REG$ and $Y=REG^{1*}(k,p)$. For $Y=REG^{1*}(1,p)$, these grammars are also known as *(periodically) time-varying grammars*, as a control language $\{H_C(P_1)\cdots H_C(P_p)\}^*$ means that the set of productions available at a time t in a derivation is P_i if $t=kp+i$, $k\geq 0$; p is called the *period* of the time-varying system. The corresponding families of languages generated by time-varying grammars with appearance checking, without appearance checking, with unconditional transfer and with period p are denoted by $\mathcal{L}(X\text{-}TV_{ac}(p))$, $\mathcal{L}(X\text{-}TV(p))$, and $\mathcal{L}(X\text{-}TV_{ut}(p))$, respectively; if p may be arbitrarily large, p is replaced by $*$ in these notions.

In many cases it is not necessary to insist that the control string $H_C(P_1)\cdots H_C(P_n)$ of a derivation is in L, it usually also is sufficient that $H_C(P_1)\cdots H_C(P_n)$ is a prefix of some string in L. We call this control *weak* and replace C by wC and TV by wTV in the notions of the families of languages. We should like to mention that in the case of wTV the control words are just prefices of the ω-word $(H_C(P_1)\cdots H_C(P_p))^\omega$.

In the case of string grammars, for $\alpha\in\{\lambda,w\}$ we know (e.g., [6]:

$$\begin{aligned}
RE &= \mathcal{L}(CF\text{-}GC_{ac}) = \mathcal{L}(CF\text{-}P_{ac}) = \mathcal{L}(CF\text{-}MAT_{ac})\\
&= \mathcal{L}(CF\text{-}GC_{ut}) = \mathcal{L}(CF\text{-}P_{ut})\\
&= \mathcal{L}(CF\text{-}\alpha C(REG)_{ac}) = \mathcal{L}(CF\text{-}\alpha C(REG)_{ut})\\
&= \mathcal{L}(CF\text{-}\alpha TV_{ac}) = \mathcal{L}(CF\text{-}\alpha TV_{ut})\\
&\supsetneq \mathcal{L}(CF\text{-}GC) = \mathcal{L}(CF\text{-}P) = \mathcal{L}(CF\text{-}MAT).
\end{aligned}$$

3. Time-Varying P Systems – Definitions and Results

A *(sequential) P system of type X* with n membranes is a construct $\Pi=(G,\mu,R,A,f)$ where $G=(O,O_T,A',P,\Longrightarrow_G)$ is a grammar of type X and

- μ is the membrane (tree) structure of the system with n membranes (μ usually is represented by a string containing correctly nested marked parentheses); we assume the membranes, i.e., the nodes of the tree representing μ, being uniquely labeled by labels from a set H;

- R is a set of rules of the form (h,r,tar) where $h\in H$, $r\in P$, and tar, called the *target indicator*, is taken from the set $\{here,in,out\}\cup\{in_j \mid 1\leq j\leq n\}$; the rules assigned to membrane h form the set $R_h=\{(r,tar)\mid (h,r,tar)\in R\}$, i.e., R can also be represented by the vector $(R_h)_{h\in H}$; for the systems considered in this paper, we do not consider communication with the environment, i.e., no objects may be sent out from the skin membrane (the outermost membrane) or taken into the skin membrane from the environment;

- A is the initial configuration specifying the objects from O assigned to each membrane at the beginning of a computation, i.e., $A = \{(h, A_h) \mid h \in H\}$;
- f is the final membrane from where the results are taken at the end of a computation.

A configuration C of the P system Π can be represented as a set $\{(h, w_h) \mid h \in H\}$, where w_h is the current contents of objects contained in the membrane labeled by h. In a *transition step*, one rule from R is applied to the objects in the current configuration in order to obtain the next configuration. A sequence of transitions between configurations of Π, starting from the initial configuration A, is called a *computation* of Π. A *halting computation* is a computation ending with a configuration $\{(h, w_h) \mid h \in H\}$ such that no from R can be applied to the objects w_h, $h \in H$, anymore, and the object w from (f, w) then is called the result of this halting computation if $w \in O_T$.

In a similar way as for grammars themselves, we are able to consider various control mechanisms as defined in the previous section for P systems, too, e.g., using a control graph. In this paper, we are going to investigate the power of regular control.

A *(sequential) P system of type X* with n membranes *and regular control* is a construct $\Pi_C = (\Pi, H_C, L, F)$ where $\Pi = (G, \mu, R, A, f)$ is a (sequential) P system of type X, L is a regular language over H_C, where H_C is the set of labels identifying the subsets of productions from R in a one-to-one manner, and $F \subseteq H_C$. The language generated by Π_C consists of all terminal objects z obtained in membrane region f as results of a halting computation in Π. Observe that as in the case of normal grammars, the sequence of computation steps must correspond to a string $H_C(R_1) \cdots H_C(R_m) \in L$ with $R_1, \cdots, R_m$ being subsets of R. The corresponding families of languages generated by P systems with regular control Π_C (with at most n membranes) are denoted by $\mathcal{L}\left(X\text{-}\alpha C(REG)_\beta OP_n\right)$, $\alpha \in \{\lambda, w\}$, $\beta \in \{\lambda, ac, ut\}$.

Yet in contrast to the previous case, appearance checking and unconditional transfer have a special effect, as we cannot make a derivation step without applying a rule, but the derivation thus will halt immediately. In order to cope with this problem specific for P systems, we allow the system to be inactive for a bounded number of steps before it really "dies", i.e., halts. We call this specific way of terminating a computation *halting with delay d*, i.e., a computation halts if for a whole sequence of length d of production sets in a control word no rule has become applicable. In that way we obtain the language classes $\mathcal{L}\left(X\text{-}\alpha C(REG)_\beta OP_n, d\right)$, $\alpha \in \{\lambda, w\}$, $\beta \in \{\lambda, ac, ut\}$. The case $k = 0$ describes the situation with normal halting.

In the P systems area we often deal with multisets, i.e., the underlying grammar is a multiset grammar. In the following, the results obtained in [1] are listed; the types describing (non-)cooperative rules are abbreviated by *coo* (*ncoo*).

Theorem 3.1 *For all $\alpha \in \{\lambda, w\}$, $\beta \in \{ac, ut\}$, $p \geq 12$, as well as $n \geq 1$ and $d \geq 2$,*

$$\mathcal{L}(mCF\text{-}\alpha TV_\beta(p)) = \mathcal{L}(ncoo\text{-}\alpha TV_\beta OP_n(p), d) = PsRE.$$

Theorem 3.2 *For all $X \in \{ncoo, coo\}$, $\alpha \in \{\lambda, w\}$, $\beta \in \{\lambda, ac, ut\}$, and $n \geq 1$,*

$$\mathcal{L}\left(ncoo\text{-}\alpha C(REG)_\beta OP_n\right) \subseteq Ps\mathcal{L}(CF\text{-}MAT).$$

Theorem 3.3 *For all* $\alpha \in \{\lambda, w\}$*,* $\beta \in \{\lambda, ac, ut\}$*, and* $k,n,p \geq 1$*,*

$$\begin{aligned}
\mathcal{L}(mARB\text{-}MAT) &= \mathcal{L}(mCF\text{-}MAT)\\
&= Ps\mathcal{L}(CF\text{-}MAT)\\
&= \mathcal{L}(mARB)\\
&= \mathcal{L}(coo\text{-}TV(p)\,OP_n)\\
&= \mathcal{L}\left(coo\text{-}\alpha C(REG)_\beta\, OP_n\right)\\
&= \mathcal{L}\left(ncoo\text{-}\alpha C\left(REG^{1*}(*,p+1)\right)_\beta OP_n\right)\\
&= \mathcal{L}\left(ncoo\text{-}\alpha C(REG)_\beta\, OP_n\right).
\end{aligned}$$

References

[1] A. Alhazov, R. Freund, H. Heikenwälder, M. Oswald, Yurii Rogozhin, S. Verlan, Time-Varying Sequential P Systems. In: *Proceedings CMC 13*. 2012.

[2] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, G. Păun, *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*. Gordon and Breac, 1994.

[3] J. Dassow, G. Păun, *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, 1989.

[4] J. Dassow, G. Păun, G. Rozenberg, *Handbook of Formal Languages*, 2, chapter Grammar Systems, Springer-Verlag, 1997, 155–172.

[5] J. Dassow, G. Păun, A. Salomaa, *Handbook of Formal Languages*, 2, chapter Grammars with Controlled Derivations, Springer-Verlag, 1997, 101–154.

[6] H. Fernau, Unconditional Transfer in Regulated Rewriting. *Acta Informatica* **34** (1997) 11, 837–857.

[7] M. Kudlek, C. Martín-Vide, G. Păun, Toward a formal macroset theory. In: C. S. Calude, G. Păun, G. Rozenberg, A. Saloma (eds.), *Multiset Processing – Mathematical, Computer Science and Molecular Computing Points of View*. LNCS 2054, Springer-Verlag, 2001, 123–134.

[8] M. Margenstern, Y. Rogozhin, About time-varying distributed H systems. In: A. Condon, G. Rozenberg (eds.), *DNA Computing: 6th International Workshop on DNA-Based Computers, DNA 2000*. LNCS 2054, Springer-Verlag, 2000, 53–62.

[9] M. Nielsen, OL systems with control devices. *Acta Informatica* **4** (1975) 4, 373–386.

[10] G. Păun, DNA computing: Distributed splicing systems. In: J. Mycielski, G. Rozenberg, A. Salomaa (eds.), *Structures in Logic and Computer Science. A Selection of Essays in Honor of A. Ehrenfeucht*. LNCS 1261, Springer-Verlag Berlin, 1997, 353–370.

[11] G. Păun, *Membrane Computing. An Introduction*. Springer-Verlag, 2002.

[12] G. Păun, G. Rozenberg, A. Salomaa, *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.

[13] G. Rozenberg, A. Salomaa, *Handbook of Formal Languages, 3 volumes*. Springer-Verlag, 1997.

[14] A. Salomaa, On finite automata with a time-variant structure. *Information and Control* **13** (1968) 2, 85–98.

[15] A. Salomaa, Periodically time-variant context-free grammars. *Information and Control* **17** (1970), 294–311.

[16] The P systems Web page. `http://ppage.psystems.eu/`.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 33 – 36

Representation of ω-regular Languages by finite Automata

Stephan Barth

Ludwig-Maximilians-Universität München, Germany

stephan.barth@ifi.lmu.de

Abstract

Hugues Calbrix, Maurice Nivat and Andreas Podelski showed how to utilize finite automata for describing ω-regular languages [1].

However they developed just transformations between Büchi and finite automata.

I will present a deeper analysis of this method. New algorithms for handling these DFA allow for performing various operations directly.

Furthermore some more transformations between this kind of DFA and classic models for ω-regular lanugages have been developed.

1. Introduction

Finite automata have various advantages over the most used automata models for ω-regular languages. Most notably, DFA can be minimized in $n \cdot \log n$ whereas minimization of deterministic Büchi, parity, Muller, Rabin, Streett automata is NP-complete, minimization of the nondeterministic variants is PSPACE-complete.

Due to the fact that DFA can not be used directly for describing ω-regular languages the other automata models are used. However, Hugues Calbrix, Maurice Nivat and Andreas Podelski detected a way to do so [1]:

If L is an ω-regular language then $L_\varphi := \{u\$v | uv^\omega \in L\}$ where \$ is a new letter is regular and contains all information about the language L. Transformations between this model and Büchi automata had been given as well.

In my ongoing thesis work I am studying the regular languages of the form L_φ and the corresponding DFA. For reference, I call these "loop languages" and "loop automata". In the talk, I will present some further transformations and automata-theoretic operations on loop languages and automata.

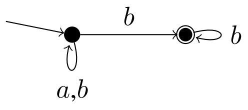
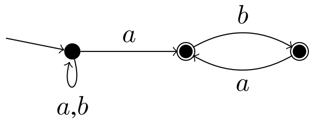
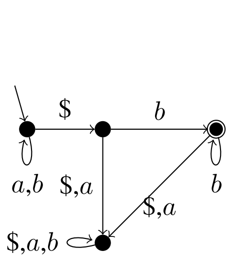
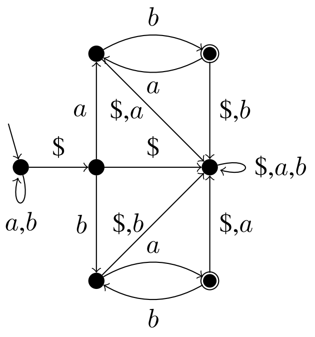
ω-regular expression	$(a\|b)^*b^\omega$	$(a\|b)^*(ab)^\omega$
Büchi automaton		
Loop automaton		

Table 1: Some examples of loop languages in comparison with Büchi automata and ω-regular expressions for the same languages

2. Properties of loop languages

As every word $u\$v$ in L_φ represents the word uv^ω in L it is clear that if $uv^\omega = u'v'^\omega$ then $u\$v \in L_\varphi \iff u'\$v' \in L_\varphi$. Together with the condition that every word in L_φ has to represent a valid word in L this leads to the following properties that every loop language fulfills:

- wellformed: $L \subset \Sigma^*\$\Sigma^+$
- duplication: $u\$v \in L \Rightarrow \forall i \in \mathbb{N}.u\$v^i \in L$
- deduplication: $(\exists i \in \mathbb{N}.u\$v^i \in L) \Rightarrow u\$v \in L$
- uprotation: $u\$av \in L \Rightarrow ua\$va \in L$
- downrotation: $ua\$va \in L \Rightarrow u\$av \in L$

These properties are sufficient for loopness and therefore the latter property can be algorithmically tested.

For a regular language L violating some of these properties, called partial loop language, it is possible that the language $\{u\$v \,|\, \exists u'\$v' \in L.uv^\omega = u'v'^\omega\}$ is not regular thus a DFA for this language can not be constructed.

Reconstruction of wellformed can be achieved by intersection with $\Sigma^*\$\Sigma^+$; the words excluded by this do not define an ω-regular word.

Reconstruction of de-/duplication does not lead to a regular language in general.

For example the regular language $\$ab^*$ would lead to the language $\{\$(ab^i)^* \,|\, i \in \mathbb{N}\}$ which is not regular.

Nevertheless extending the language only by those words that are needed for deduplication leads to a regular language, call this closure operator C_{dedup}.

As deduplication and duplication are dual to each other, that means L fulfills duplication $\iff \neg L$ fulfills deduplication, one can also define a co-closure operator $C_{dup}L = \neg C_{dedup}\neg L$ that enforces duplication by shrinking the language as little as necessary; wellformed has to be reconstructed in every step.

Note that all words that miss some bigger representative will be dropped; thus the language $\$ab^*$ would be converted to the empty language when C_{dup} is applied.

Reconstruction of up-/downrotation also does not lead to a regular language in general.

It is possible in both directions to rotate by any finite count of letters.

Arbitrary uprotation of the language $\$ab^*$ would lead to a not regular language; even when de-/duplication holds as for example $(\$ab^*)^+$ this can happen; in both cases the intersection with $ab^*\$ab^*$ would be $\{ab^i\$ab^i \,|\, i \in \mathbb{N}\}$ which is not regular.

When a language fulfills uprotation it is possible to extend it by only words that have other representatives of the same ω-regular words already in the language such that it also fulfills downroation, call this operator C_{down}.

Similar to the situation with de-/duplication up- and downrotation are again dual to each other. $C_{up}L = \neg C_{down}\neg L$ shrinks the language as little as necessary for reconstruction of uprotation.

$\$ab^*$ as well as $(\$ab^*)^+$ would return an empty language when reconstructing uprotation with this method.

3. Operations on loop languages

Various operations can be performed directly on loop languages. The necessary blowup varies on the kind of operation.

For intersection and union it is sufficient to use the corresponding constructions for deterministic finite automata.

Complementation will destroy wellformed; thus this property has to be restored.

Projection can destroy both deduplication and downrotation. After repairing these properties by increasing the language to the minimal loop language the projection is correctly performed.

4. Transformation from and to other automata models

In the original paper introducing this model [1] transformations between loop DFA and Büchi automata were given.

A. Farzan, Y. Chen et al. introduced a more efficient transformation from NBA to DFA [2]. It is done by learning a partial loop automaton.

I was able to develop new transformations, for example from DPA to a partial loop DFA, and will describe them.

5. Applications

Longer chains of operations on ω-regular languages could profit from this new model as after every operation a cheap minimization can be performed.

When applying the transformations DPA $\rightarrow$ partial loop DFA $\rightarrow$ NBA the blowup can be limited to $n \cdot k$ which is the same as when transforming DPA $\rightarrow$ NBA. Using partial loop DFA as intermediate automata has the advantage that the DFA can be minimized.

References

[1] H. CALBRIX, M. NIVAT, A. PODELSKI, Ultimately periodic words of rational omega-languages. In: S. BROOKES, M. MAIN, A. MELTON, M. MISLOVE, D. SCHMIDT (eds.), *Mathematical Foundations of Programming Semantics*. Lecture Notes in Computer Science 802, Springer Berlin / Heidelberg, 1994, 554–566. 10.1007/3-540-58027-1_27.

[2] A. FARZAN, Y.-F. CHEN, E. CLARKE, Y.-K. TSAY, B.-Y. WANG, Extending Automated Compositional Verification to the Full Class of Omega-Regular Languages. In: C. RAMAKRISHNAN, J. REHOF (eds.), *Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science 4963, Springer Berlin / Heidelberg, 2008, 2–17. 10.1007/978-3-540-78800-3_2.

Machine Translation with Multi Bottom-up Tree Transducers

Fabienne Braune[A] Nina Seemann[A] Daniel Quernheim[A]
Andreas Maletti[A]

[A]Universität Stuttgart, Institute for Natural Language Processing,
Pfaffenwaldring 5b, 70569 Stuttgart, Germany
`{braunefe, seemanna, daniel, maletti}@ims.uni-stuttgart.de`

Several tree-based formalisms such as Synchronous Context-Free Grammars [2, 8], Synchronous Tree-Substitution Grammars [5], Synchronous Tree-Adjoining Grammars [4] or Synchronous Tree-Sequence-Substitution Grammars [11] have been explored and implemented into Machine Translation systems. Furthermore, [4] and [11] show that the integration of powerful formalisms such as Synchronous Tree Adjoining Grammars (STAG) or Synchronous Tree Sequence Substitution Grammars (STSSG) lead to improvements in translation quality over systems based on less powerful models like Synchronous Context-Free Grammars (SCFG).

In the same spirit we propose to build an MT system integrating a recently developed grammar formalism, the weighted multi bottom-up tree transducer [6, 9]. Our weights currently represent probabilities. The main advantage of this formalism is its ability to model discontiguous grammar rules. This allows, first, better rule extraction, as shown in [10] as well as better translation between language pairs involving discontinuities. We expect forward application of MBOT rules to capture discontinuities on the target language side. Backward application, on the other hand, is intended to capture discontinuities on the source language side.

We implemented weighted multi bottom-up tree transducers (MBOTs) into MOSES [8] and run the standard machine translation pipeline using an MBOT as translation model. Our pipeline includes the following components:

Rule Extraction The MBOT rules that we use are obtained with the help of external tools. First, the rules are extracted from an aligned bi-parsed corpus of sentence pairs using the method of [10]. Second, a standard input and output restriction generates the regular derivation tree language for each sentence pair, which is used to determine the inside weights used in EM training [1]. The training step delivers the rule weights that best explain the training data. The obtained weighted rules are first transformed into trees of depth one and output in a format that is readable by MOSES.

All authors were financially supported by the German Research Foundation (DFG) grant MA 4959/1-1.

Translation System Several parts of the MOSES translation system have been extended in order to operate with MBOT rules. For now, only forward application has been integrated but in the long run we plan to integrate backward application as well. Our extensions include the implementation of a new rule loading procedure. The representation of grammar rules internal to the MT system has also been extended to work with MBOT rules. Because we use only forward application, the core parsing algorithm remains unchanged. However, the construction of the output translation given a derivation has been adjusted to deal with rules containing discontinuities on the target language side.

Language Model In syntax-based systems without discontinuities, language model scores are computed by memorizing partial derivations and computing n-gram scores of adjacent words. When dealing with discontinuities, this is no longer possible. We overcome this problem by considering discontiguous target sides as being independent. Hence, for each target unit, we compute the corresponding n-gram score. All obtained scores are then multiplied. Furthermore, each used rule must contain a smaller amount of discontiguous phrases as its antecedent in the derivation tree.

We will recall MBOT as the main theoretical model and then present a detailed overview of all steps (both theoretical as well as practical) that are used to obtain our final translation system. Whenever feasible this will be illustrated on examples.

We will also demonstrate the implementation and present a comparison and evaluation of our translation system using the typical synchronous context-free grammars [3] used in [7] as a baseline. We expect that MBOTs allow us to model target side discontinuities much better than SCFG based tree formalisms.

References

[1] D. B. R. A. P. DEMPSTER, N. M. LAIRD, Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B* **39** (1977) 1, 1–38.

[2] D. CHIANG, A hierarchical phrase-based model for statistical machine translation. In: *Proc. ACL*. ACL, 2005, 263–270.

[3] D. CHIANG, An Introduction to Synchronous Grammars. In: *Proc. 44th Annual Meeting of the Association for Computational Linguistics*. ACL, 2006. Part of a tutorial given with Kevin Knight.

[4] S. DENEEFE, K. KNIGHT, Synchronous Tree Adjoining Machine Translation. In: *In Proceedings of EMNLP*. 2009.

[5] J. EISNER, Learning Non-Isomorphic Tree Mappings for Machine Translation. In: *Proc. 41st Ann. Meeting ACL*. ACL, 2003, 205–208.

[6] J. ENGELFRIET, E. LILIN, A. MALETTI, Composition and Decomposition of Extended Multi Bottom-up Tree Transducers. *Acta Inform.* **46** (2009) 8, 561–590.

[7] H. HOANG, P. KOEHN, Improved Translation with Source Syntax Labels. In: *Proc. 5th Workshop Statistical Machine Translation and MetricsMATR*. ACL, 2010, 409–417.

[8] P. KOEHN, H. HOANG, A. BIRCH, C. CALLISON-BURCH, M. FEDERICO, N. BERTOLDI, B. COWAN, W. SHEN, C. MORAN, R. ZENS, C. DYER, O. BOJAR, A. CONSTANTIN,

E. HERBST, MOSES: Open Source Toolkit for Statistical Machine Translation. In: *Proc. 45th Annual Meeting of the Association for Computational Linguistics*. ACL, 2007, 177–180.

[9] A. MALETTI, An Alternative to Synchronous Tree Substitution Grammars. *Journal of Natural Language Engineering* **17** (2011) 2, 221–242.

[10] A. MALETTI, How to Train Your Multi Bottom-up Tree Transducer. In: *Proc. 49th Annual Meeting Association for Computational Linguistics*. ACL, 2011, 825–834.

[11] J. SUN, M. ZHANG, C. L. TAN, A non-contiguous Tree Sequence Alignment-based Model for Statistical Machine Translation. In: *ACL 2009, Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore*. The Association for Computer Linguistics, 2009, 914–922.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 41 – 44

As Easy As Vanda, Two, Three: Components for Machine Translation Based on Formal Grammars

Matthias Büchse[(A)]

Chair of Foundations of Programming
Technische Universität Dresden
01062 Dresden, Germany
Matthias.Buechse@tu-dresden.de

Machine Translation is the task of enabling computers to translate text from one language into another. Statistical Machine Translation (SMT), in particular, applies methods from Statistics and Machine Learning to automatically select a translation function that performs well on existing translations, with the hope that it will also perform well on new sentences.

In recent years a lot of research has focused on using formal grammars and related formalisms for specifying translation functions. Among those are synchronous context-free grammars [1, 5], synchronous tree-substitution grammars [10], synchronous tree-adjoining grammars [27, 8], synchronous tree-sequence-substitution grammars [30], extended top-down tree-to-string transducers [16, 14, 12], and multi-bottom-up tree transducers [11, 22].

In principle, these formalisms are amenable to formal treatment, just like weighted string automata and weighted string transducers. The latter possess a rich theory with results about closure properties, characterizations, complexity and decidability. Building on that strong foundation, there is a versatile algorithmic toolbox, as witnessed by [24, 25, 2]. In conjunction, the theory and the toolbox allow for effective algebraic specification and subsequent implementation of tasks in areas such as speech recognition [26] and morphology [15].

However, in the SMT realm, this kind of comprehensive formal treatment has yet to happen. Most of the formalisms named above have been defined ad-hoc, so as to build a translation system which can be evaluated.[1] Core algorithms employed in those systems are often monolithic, and they are implemented in thousands of lines of code, as witnessed by open-source systems such as Moses [19], Joshua [21], or cdec [9]. The source code of most research systems, including Hiero [6], is not available.

Being aware of weighted string automata and weighted string transducers, the community does indeed express a desire for algebraic specification [18]. In fact, it has been shown that the string devices can be employed for central algorithms [17, 7]. Moreover, May and Knight made an effort to develop a toolkit, named Tiburon [23], for extended top-down tree transducers. However, it has not been employed for building research SMT systems.

In the long run, we intend to provide components for (partial) algebraic specification of research SMT systems with our system Vanda, which shall rest on three columns:

[(A)] financially supported by DFG VO 1011/6-1.

[1] This is an enormous feat by itself, one that requires immense engineering skill, and one we do not intend to devalue.

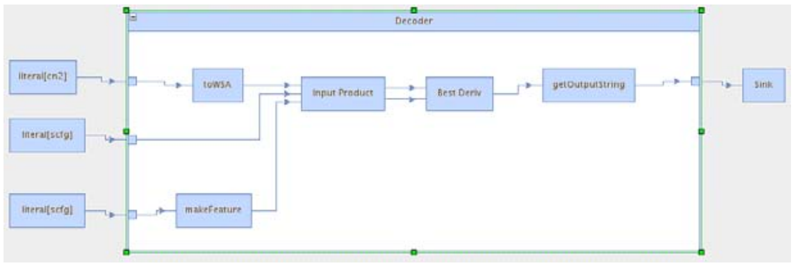


Figure 1: Representation of the decoding task as a workflow in Vanda Studio.

1. A theory and an algorithmic toolbox based on a versatile formal framework, namely interpreted regular tree grammars, or IRTGs [20]. This framework is based on the ideas of bimorphism semantics [3] and initial-algebra semantics [13], and it subsumes all of the formalisms mentioned initially.

2. Vanda Toolbox, a Haskell library that implements the algorithmic toolbox. Haskell is a clean and concise modern high-level language that is compiled into native code via the Glasgow Haskell Compiler. It features a powerful static type system, yet it can still be used for rapid prototyping because of automatic type inference at compile time.

3. Vanda Studio, a graphical (hyper)workflow management system that greatly facilitates conducting experiments by providing both a standardized, well-documented interface and the ability to specify alternatives within a single hyperworkflow. A prototype of Vanda Studio has been implemented [4].

As a proof of our concept, we[2] have implemented in Vanda Toolbox an IRTG-inspired representation of extended top-down tree-to-string transducers, along with suitable algorithms, e.g., for left/right product with regular weighted string/tree languages, binarization of rules, determining n best derivations, rule extraction, and inside-outside EM training. This implementation allows us to accomplish the following three tasks from the area of SMT:

Extraction Rule extraction from any given parallel corpus. To this end, we parse the target (English) side using the Berkeley parser [29], and we use GIZA++ [28] to obtain a word alignment for each sentence pair. We extract all rules that correspond to minimal fragments [12].

Training Estimating rule weights.

Decoding Translating any given sentence (so far without language model).

Each of the tasks can be carried out from within Vanda Studio; as an example, Figure 1 shows the workflow representation of the translation task. Each box in the outer (shaded) region corresponds to a part of a shell script to be run in a Unix environment, while the box labeled “Decoder” represents a Haskell program, and the inner boxes are Haskell functions.

[2]The author had help from his colleagues Toni Dietze, Johannes Osterholzer, and Linda Leuschner.

In my presentation, I will show how the three aforementioned tasks can be accomplished in a component-based manner using Vanda Toolbox and Vanda Studio, and I will report on the performance on medium-scale data.

References

[1] A. V. AHO, J. D. ULLMAN, Syntax directed translations and the pushdown assembler. *J. Comput. System Sci.* **3** (1969), 37–56.

[2] C. ALLAUZEN, M. RILEY, J. SCHALKWYK, W. SKUT, M. MOHRI, OpenFst: a general and efficient weighted finite-state transducer library. In: *Proceedings of the 12th international conference on Implementation and application of automata*. CIAA'07, Springer, 2007, 11–23.

[3] A. ARNOLD, M. DAUCHET, Bi-transduction de forêts. In: *Proc. 3rd Int. Coll. Automata, Languages and Programming*. Edinburgh University Press, 1976, 74–86.

[4] M. BÜCHSE, T. DIETZE, J. OSTERHOLZER, A. FISCHER, L. LEUSCHNER, Vanda: A Statistical Machine Translation Toolkit. In: M. DROSTE, H. VOGLER (eds.), *Proceedings of the Workshop Weighted Automata: Theory and Applications 2012*. 2012, 36–38.

[5] D. CHIANG, Hierarchical Phrase-Based Translation. *Comp. Ling.* **33** (2007) 2, 201–228.

[6] D. CHIANG, A. LOPEZ, N. MADNANI, C. MONZ, P. RESNIK, M. SUBOTIN, The Hiero machine translation system: extensions, evaluation, and analysis. In: *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. ACL, Morristown, NJ, USA, 2005, 779–786.

[7] A. DE GISPERT, G. IGLESIAS, G. BLACKWOOD, E. R. BANGA, W. BYRNE, Hierarchical Phrase-Based Translation with Weighted Finite-State Transducers and Shallow-n Grammars. *Computational Linguistics* **36** (2010) 3, 505–533.

[8] S. DENEEFE, K. KNIGHT, Synchronous Tree-Adjoining Machine Translation. In: *EMNLP '09: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*. ACL, Morristown, NJ, USA, 2009, 727–736.

[9] C. DYER, A. LOPEZ, J. GANITKEVITCH, J. WEESE, F. TURE, P. BLUNSOM, H. SETIAWAN, V. EIDELMAN, P. RESNIK, cdec: A Decoder, Alignment, and Learning Framework for Finite-State and Context-Free Translation Models. In: *Proceedings of the ACL 2010 System Demonstrations*. ACL, Uppsala, Sweden, 2010, 7–12.

[10] J. EISNER, Learning non-isomorphic tree mappings for machine translation. In: *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 2*. ACL '03, ACL, Stroudsburg, PA, USA, 2003, 205–208.

[11] J. ENGELFRIET, E. LILIN, A. MALETTI, Extended Multi Bottom-up Tree Transducers. In: M. ITO, F. M. TOYAMA (eds.), *Proc. 12th Int. Conf. Developments in Language Theory*. LNCS 5257, Springer, 2008, 289–300.

[12] M. GALLEY, M. HOPKINS, K. KNIGHT, D. MARCU, What's in a translation rule? In: S. DUMAIS, D. MARCU, S. ROUKOS (eds.), *HLT-NAACL 2004: Main Proceedings*. ACL, Boston, Massachusetts, USA, 2004, 273–280.

[13] J. A. GOGUEN, J. W. THATCHER, E. G. WAGNER, J. B. WRIGHT, Initial algebra semantics and continuous algebras. *J. ACM* **24** (1977), 68–95.

[14] J. GRAEHL, K. KNIGHT, J. MAY, Training Tree Transducers. *Comp. Ling.* **34** (2008) 3, 391–427.

[15] T. HANNEFORTH, *fsm2* - A Scripting Language for Creating Weighted Finite-State Morphologies. In: C. MAHLOW, M. PIOTROWSKI (eds.), *SFCM*. Communications in Computer and Information Science 41, Springer, 2009, 48–63.

[16] L. HUANG, K. KNIGHT, A. JOSHI, A syntax-directed translator with extended domain of locality. In: *Proceedings of the Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*. CHSLP '06, ACL, Stroudsburg, PA, USA, 2006, 1–8.

[17] G. IGLESIAS, A. DE GISPERT, E. R. BANGA, W. BYRNE, Hierarchical Phrase-Based Translation with Weighted Finite State Transducers. In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. ACL, Boulder, Colorado, 2009, 433–441.

[18] K. KNIGHT, Capturing practical natural language transformations. *Machine Translation* **21** (2007) 2, 121–133.

[19] P. KOEHN, H. HOANG, A. BIRCH, C. CALLISON-BURCH, M. FEDERICO, N. BERTOLDI, B. COWAN, W. SHEN, C. MORAN, R. ZENS, C. DYER, O. BOJAR, A. CONSTANTIN, E. HERBST, Moses: open source toolkit for statistical machine translation. In: *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*. ACL '07, ACL, Stroudsburg, PA, USA, 2007, 177–180.

[20] A. KOLLER, M. KUHLMANN, A Generalized View on Parsing and Translation. In: *Proceedings of the 12th International Conference on Parsing Technologies*. ACL, Dublin, Ireland, 2011, 2–13.

[21] Z. LI, C. CALLISON-BURCH, C. DYER, J. GANITKEVITCH, S. KHUDANPUR, L. SCHWARTZ, W. N. G. THORNTON, J. WEESE, O. F. ZAIDAN, Joshua: an open source toolkit for parsing-based machine translation. In: *Proceedings of the Fourth Workshop on Statistical Machine Translation*. StatMT '09, ACL, Stroudsburg, PA, USA, 2009, 135–139.

[22] A. MALETTI, Why Synchronous Tree Substitution Grammars? In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. ACL, Los Angeles, California, 2010, 876–884.

[23] J. MAY, K. KNIGHT, Tiburon: a weighted tree automata toolkit. In: O. IBARRA, H. YEN (eds.), *CIAA 2006*. Lecture Notes in Comput. Sci. 4094, Springer, 2006, 102–113.

[24] M. MOHRI, Weighted automata algorithms. In: M. DROSTE, W. KUICH, H. VOGLER (eds.), *Handbook of Weighted Automata*. chapter 6, Springer, 2009, 213–254.

[25] M. MOHRI, F. C. N. PEREIRA, M. RILEY, The design principles of a weighted finite-state transducer library. *Theoret. Comp. Science* **231** (2000), 17–32.

[26] M. MOHRI, F. C. N. PEREIRA, M. RILEY, Weighted Finite-State Transducers in Speech Recognition. *Computer Speech and Language* **16** (2002) 1, 69–88.

[27] R. NESSON, S. M. SHIEBER, A. RUSH, Induction of Probabilistic Synchronous Tree-Insertion Grammars for Machine Translation. In: *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas (AMTA 2006)*. Boston, Massachusetts, 2006, 128–137.

[28] F. J. OCH, H. NEY, A systematic comparison of various statistical alignment models. *Computational Linguistics* **29** (2003) 1, 19–51.

[29] S. PETROV, L. BARRETT, R. THIBAUX, D. KLEIN, Learning Accurate, Compact, and Interpretable Tree Annotation. In: *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*. ACL, Sydney, Australia, 2006, 433–440.

[30] M. ZHANG, H. JIANG, A. AW, H. LI, C. L. TAN, S. LI, A Tree Sequence Alignment-based Tree-to-Tree Translation Model. In: *Proceedings of ACL-08: HLT*. ACL, Columbus, Ohio, 2008, 559–567.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 45 – 50

Connecting Partial Words and Regular Languages

Jürgen Dassow$^{(A)}$ Florin Manea$^{(B)}$ Robert Mercaş$^{(A)}$

$^{(A)}$Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik,
PSF 4120, D-39016 Magdeburg, Germany,
dassow@iws.cs.uni-magdeburg.de, robertmercas@gmail.com

$^{(B)}$Christian-Albrechts-Universität zu Kiel, Institut für Informatik,
D-24098 Kiel, Germany, flm@informatik.uni-kiel.de

Abstract

We initiate a study of languages of partial words related to regular languages of full words. First, we study the possibility of expressing a regular language of full words as the image of a partial-words-language through a substitution that only replaces the hole symbols of the partial words with a finite set of letters. Results regarding the structure, uniqueness and succinctness of such a representation, as well as a series of related decidability and computational-hardness results, are presented. Finally, we define a hierarchy of classes of languages of partial words, by grouping together languages that can be connected in strong ways to regular languages, and derive their closure properties.

1. Introduction

Partial words are words that beside regular letters contain an extra "joker" symbol, also called "hole" or "do-not-know" symbol, that matches all symbols of the original alphabet, which were investigated already in the 1970s [2]. In the last decade a lot of combinatorial and algorithmic properties of partial words have been investigated (see the survey [1], and the references therein). Surprisingly, so far, the only study of classes of languages of partial words (or sets of partial words that have common features) that we are aware of, is [3]. .

In this work, we aim to establish a stronger connection between the attractive notions mentioned above: partial words, on one side, and regular languages, on the other side. First, we show how we can (non-trivially) represent every regular language as the image of a regular language of partial words through a substitution that defines the letters that may replace the hole (called $\diamond$-substitution, in the following). We show that such a representation can be useful: for some regular languages, there exist deterministic finite automata accepting languages of partial words that represent the full-word-language and are exponentially more succinct than the minimal deterministic finite automaton accepting that language. Unfortunately, it may also be the case when the minimal non-deterministic finite automaton accepting a language is exponentially more succinct than any deterministic automaton accepting a language

$^{(A)}$Work supported by the *Alexander von Humboldt Foundation*.
$^{(B)}$Work supported by the *DFG grant 582014*.

of partial words representing the same language. An automata accepting languages of partial words representing a given full-words language can be seen as intermediate between the deterministic finite automata and the non-deterministic automata accepting that language. We also present a series of algorithmic and complexity results regarding the representation of a regular language as the image of a language of partial words through a $\diamond$-substitution.

Motivated by the above results, that connect in a meaningful way languages of partial words to regular languages of full words, and by the theoretical interest of studying systematically such languages, we define a series of classes of languages of partial words. Each of these classes contains languages that can be placed in a particular strong relation with the regular languages. Further, we investigate these classes from a language theoretic point of view, show that they form a hierarchy, and establish their closure properties.[1]

We begin the paper with a series of basic definitions. For an *alphabet* V, a *full word* (or, simply, word) is a finite sequence of letters from V while a *partial word* is a finite sequence of letters from $V \cup \{\diamond\}$, the alphabet V extended with the distinguished hole symbol $\diamond$. The number of occurrences of a symbol a in a (partial) word w is denoted $|w|_a$. The *empty (partial) word* is the sequence of length zero and is denoted by λ. We denote by V^* (respectively, $(V \cup \{\diamond\})^*$) the set of words (respectively, partial words) over the alphabet V and by V^+ (respectively, $(V \cup \{\diamond\})^+$) the set of non-empty words (respectively, non-empty partial words) over V. The catenation of two (partial) words u and v is defined as the (partial) word uv. Recall that V^* (where the alphabet V may include the $\diamond$ symbol) is the free monoid generated by V, under the operation of catenation of words; the unit element in this monoid is represented by the empty word λ. A language L of full words over an alphabet V is a subset of V^*; a language of partial words L over an alphabet V (that does not contain the $\diamond$ symbol) is a subset of $(V \cup \{\diamond\})^*$. Given L we denote by **alph**(L) (the alphabet of L) the set of all the letters that occur in the words of L; for the precision of the exposure, we say that a language L of full (respectively, partial) words is over V, with $\diamond \notin V$, if and only if **alph**$(L) = V$ (respectively, **alph**$(L) = V \cup \{\diamond\}$). For instance, $L = \{abb, ab\diamond\}$ has **alph**$(L) = \{a,b,\diamond\}$, thus, is a language of partial words over $\{a,b\}$. The catenation operation can be extended to languages (for L_1, L_2 languages over V, we have $L_1L_2 = \{w_1w_2 \mid w_1 \in L_1, w_2 \in L_2\}$).

Let u and v be two partial words of equal length. Say that u is *contained* in v, denoted by $u \sqsubset v$, if $u[i] = v[i]$ for all $u[i] \in V$; moreover, u and v are *compatible*, denoted by $u \uparrow v$, if there exists w such that $u \sqsubset w$ and $v \sqsubset w$. These notions are extended to languages. Let L and L' be two languages of partial words with **alph**$(L) \cup$ **alph**$(L') = V \cup \{\diamond\}$ and $\diamond \notin V$. Say that L is *contained* in L', denoted $L \sqsubset L'$, if, for every $w \in L$, there exists $w' \in L'$ such that $w \sqsubset w'$. S ay that L is *compatible* to L', denoted $L \uparrow L'$, if, for each $w \in L$, there exists $w' \in L'$ such that $w \uparrow w'$ and, for each $v' \in L'$, there exists $v \in L$ such that $v' \uparrow v$.

A substitution is a mapping $h : V^* \to 2^{U^*}$ with $h(xy) = h(x)h(y)$, for $x, y \in V^*$, and $h(\lambda) = \{\lambda\}$. A morphism is a particular type of a substitution for which $h(a)$ contains exactly one element for all $a \in V$. A $\diamond$-substitution over V is a substitution with $h(a) = \{a\}$, for $a \in V$, and $h(\diamond) \subseteq V$. Here we assume that $\diamond$ can replace any symbol of V.

In this paper, DFA stands for deterministic finite automaton and NFA stands for non-deterministic finite automaton; the language accepted by a finite automaton M is denoted $L(M)$. Also, the set of all the regular languages is denoted by **REG**; by **REG**$_{\mathbf{full}}$, we denote

[1]A technical appendix containing full proofs: `https://www.informatik.uni-kiel.de/zs/pwords`.

the set of all the regular languages of full words. Further definitions regarding finite automata and regular languages can be found in [5], while partial words are surveyed in [1].

2. Definability by Substitutions

Let us begin our investigation by presenting several results regarding the way regular languages can be expressed as the image of a language of partial words through a substitution.

Lemma 2.1 *Let $L \subseteq (V \cup \{\diamond\})^*\{\diamond\}(V \cup \{\diamond\})^*$ be a language of partial words and let σ be a $\diamond$-substitution over V. There exists L' such that $\sigma(L) = \sigma(L')$ and $|w|_\diamond = 1$ for all $w \in L'$.*

Lemma 2.2 *Let L be a regular language over V and σ a $\diamond$-substitution over V. There exists a maximal (with respect to set inclusion) language $L' \subseteq L$ that can be written as $\sigma(L'')$, where L'' is a language of partial words such that any word in L'' has exactly one hole. Moreover, L' and L'' are regular languages and, provided that L is given by a finite automaton accepting it, one can algorithmically construct a finite automaton accepting L' and L''.*

The next relations connect partial-words-languages to full-words-languages.

Definition 2.3 *Let $L \subseteq V^*$ be a language and σ be a $\diamond$-substitution over V. We say that L is σ-defined by the language L', where $L' \subseteq (V \cup \{\diamond\})^*$ is a partial-words-language, if $L = \sigma(L')$. Moreover, we say that L is* essentially σ-defined *by L', where $L' \subseteq (V' \cup \{\diamond\})^*$, if $L = \sigma(L')$ and every word in L' contains at least a $\diamond$-symbol.*

Obviously, for any regular language L over V, there is a regular language L' of partial words and a $\diamond$-substitution σ over V such that $\sigma(L') = L$ (in L' take words from L where a symbol $a \in V$ is replaced by $\diamond$, and the $\diamond$-substitution σ that maps $\diamond$ to $\{a\}$.) We first characterize the essentially definable languages. Be Lemmas 2.1 and 2.2 we get the following:

Theorem 2.4 *For a regular language of full words L over V and σ a $\diamond$-substitution over V it is decidable whether L is essentially σ-definable.*

We also easily get the following decidability results.

Theorem 2.5 *i) Given a regular language L over V and a $\diamond$-substitution σ over V, it is decidable whether L is essentially σ-definable.*
ii) Given a regular language L over V, one can algorithmically identify all $\diamond$-substitutions σ for which L is essentially σ-definable.

The following consequence of Lemma 2.2 is worth noting, as it provides a canonical non-trivial representation of regular languages.

Theorem 2.6 *Given a regular language $L \subseteq V^*$ and a $\diamond$-substitution σ over V, there exists a unique regular language $L_\diamond$ of partial words that fulfils the following three conditions: (i) $L = \sigma(L_\diamond)$, (ii) for any language L_1 with $\sigma(L_1) = L$ we have $\{w \mid w \in L_1, |w|_\diamond \geq 1\} \sqsubset \{w \mid w \in L_\diamond, |w|_\diamond \geq 1\}$, and (iii) $(L_\diamond \cap V^*) \cap \sigma(\{w \mid w \in L_\diamond, |w|_\diamond \geq 1\}) = \emptyset$.*

Motivated by this last result, we now check if there are cases when one can describe in a more succinct way a regular language via a language of partial words and a substitution that define it? Can we decide algorithmically whether for a given regular language L there exist a language of partial words and a substitution providing a more succinct description of L? For L a regular language of full words over V denote by $\min_{DFA}(L)$ ($\min_{NFA}(L)$) the number of states of the complete minimal DFA (NFA) accepting L. Moreover, for a regular language L let $\min^{\diamond}_{DFA}(L)$ denote the minimum number of states of a (complete) DFA accepting a regular language $L' \subseteq (V \cup \{\diamond\})^*$ (where $\diamond$ is considered as an input symbol) for which there exists a $\diamond$-substitution σ over V such that $\sigma(L') = L$. We get the following relation:

Theorem 2.7 *For all regular languages we have* $\min_{DFA}(L) \geq \min^{\diamond}_{DFA}(L) \geq \min_{NFA}(L)$. *Furthermore, there exist regular languages for which the inequalities are strict.*

In fact, one can show that the differences $\min_{DFA}(L) - \min^{\diamond}_{DFA}(L)$ and $\min^{\diamond}_{DFA}(L) - \min_{NFA}(L)$ may have an exponential blow-up with respect to both relations.

Theorem 2.8 *Let n be a natural number, $n \geq 3$. There exist regular languages L and L' such that* $\min^{\diamond}_{DFA}(L) \leq n+1$ *and* $\min_{DFA}(L) = 2^n - 2^{n-2}$ *and* $\min_{NFA}(L') \leq 2n+1$ *and* $\min^{\diamond}_{DFA}(L') \geq 2^n - 2^{n-2}$.

The following remark provides an algorithmic side of the results stated above.

Remark 2.9 *Given a DFA accepting a regular language L we can construct algorithmically a DFA with* $\min^{\diamond}_{DFA}(L)$ *states, accepting a regular language of partial words L', and a $\diamond$-substitution σ over* **alph**(L)*, such that L is σ-defined by L'.*

We conclude by showing the hardness of a problem related to definability.

Theorem 2.10 *Consider the problem P: "Given a DFA accepting a language L of full words, a DFA accepting a language L' of partial words, and a $\diamond$-substitution σ over* **alph**(L)*, decide whether $\sigma(L') \neq L$." This problem is NP-hard.*

3. Languages of partial words

In this section we investigate the languages of partial words whose images through a substitution (or all possible substitutions) are regular and those compatible with at least one regular language (or only with regular languages). The definitions of the first three classes look at languages of partial words that can be transformed, via substitutions, into regular languages.

Definition 3.1 *Let L be a language of partial words over V.*
1. *We say that L is $(\forall\sigma)$-regular if $\sigma(L)$ is regular for all the $\diamond$-substitutions σ over alphabets that contain V and do not contain $\diamond$.*
2. *We say that L is* **max***-regular if $\sigma(L)$ is regular, where σ is a $\diamond$-substitution over V' with $\sigma(\diamond) = V'$, and $V' = V$ if $V \neq \emptyset$, and V' is a singleton with $\diamond \notin V'$, otherwise.*
3. *We say that L is $(\exists\sigma)$-regular if there exists a $\diamond$-substitution σ over a non-empty alphabet V', that contains V and does not contain $\diamond$, such that $\sigma(L)$ is regular.*

The classes of all $(\forall\sigma)$-regular, **max***-regular, and $(\exists\sigma)$-regular languages are denoted by* $\mathbf{REG}_{(\forall\sigma)}$, $\mathbf{REG}_{\mathbf{max}}$*, and, respectively,* $\mathbf{REG}_{(\exists\sigma)}$.

We consider, in the following, two classes of languages of partial words that are defined starting from the concept of compatibility.

Definition 3.2 *Let L be a language of partial words over V.*
4. *We say that L is $(\exists)$-regular if exists a regular language L' of full words such that $L \uparrow L'$.*
5. *We say that L is $(\forall)$-regular if every language L' of full words such that $L \uparrow L'$ is regular.*
The class of all the $(\exists)$-regular languages is denoted $\mathbf{REG}_{(\exists)}$*, while that of $(\forall)$-regular languages by* $\mathbf{REG}_{(\forall)}$.

According to the definitions from [3], the $(\exists)$-regular languages are those whose restoration contains at least a regular language, while $(\forall)$-regular languages are those whose restoration contains only regular languages. We start with the following result.

Theorem 3.3 *For every non-empty alphabet V with $\diamond \notin V$ there exist an undecidable language L of partial words over V, such that:*
i) $\sigma(L) \in$ **REG** *for all substitutions σ over V, and $\sigma'(L) \notin$* **REG** *for the $\diamond$-substitution σ' with $\sigma'(\diamond) = V \cup \{c\}$, where $c \notin V$.*
ii) every full-words-language $L' \subseteq V^*$ compatible with L is regular and there is an undecidable language $L'' \subseteq (V')^*$, where V' strictly extends V, which is compatible with L.

We can now show a first result regarding the classes previously defined.

Theorem 3.4 $\mathbf{REG} = \mathbf{REG}_{(\forall\sigma)} \subset \mathbf{REG}_{\mathbf{max}}$.

The next result gives some insight on the structure of the class $\mathbf{REG}_{\mathbf{max}}$.

Theorem 3.5 *Let $L \in \mathbf{REG}_{\mathbf{max}}$ be a language of partial words over $V \neq \emptyset$ and σ the $\diamond$-substitution used in the definition of $\mathbf{REG}_{\mathbf{max}}$. Then there exists a maximal language (with respect to set inclusion) $L_0 \in \mathbf{REG}_{\mathbf{max}}$ of partial words over V such that $\sigma(L_0) = \sigma(L)$. Moreover, given an automaton accepting L, an automaton accepting L_0 can be constructed.*

It is easy to see that any language from $\mathbf{REG}_{\mathbf{max}}$ whose words contain only holes is regular.
The following relation also holds:

Theorem 3.6 $\mathbf{REG}_{\mathbf{max}} \subset \mathbf{REG}_{(\exists\sigma)} \subset \mathbf{REG}_{(\exists)}$.

As already shown, all the languages in $\mathbf{REG}_{(\forall)}$ are in $\mathbf{REG} = \mathbf{REG}_{(\forall\sigma)}$; however, not all the languages in **REG** are in $\mathbf{REG}_{(\forall)}$. The following statement characterizes exactly the regular languages that are in $\mathbf{REG}_{(\forall)}$.

Theorem 3.7 *Let L be a regular partial-words-language over V. Then $L \in \mathbf{REG}_{(\forall)}$ if and only if the set $\{w \mid |w|_\diamond \geq 1, w \in L\}$ is finite.*

The previous result provides a simple procedure for deciding whether a regular partial-words-language is in $\mathbf{REG}_{(\forall)}$ or not (taking as input a DFA for that language check whether there are finitely many words that contain $\diamond$ or not).

Theorem 3.7 has also the following consequence.

Theorem 3.8 $\mathbf{REG}_{(\forall)} \subset \mathbf{REG}$.

In [4], partial words were defined by applying the finite transduction defined by a deterministic generalised sequential machine (DGSM) to full words, such that $\diamond$ appears in the output word. Accordingly, we can define a new class of partial-words-languages, $\mathbf{REG_{gsm}}$, using this automata-theoretic approach. Let L be a language of partial words over V, with $\diamond \in \mathbf{alph}(L)$; L is **gsm**-regular, and is in $\mathbf{REG_{gsm}}$, if there exists a DGSM M and a regular language L' such that L is obtained by applying the finite transduction defined by M to L'. It is not hard to show that $\mathbf{REG_{gsm}} = \mathbf{REG} \setminus \mathbf{REG_{full}}$.

By the Theorems 3.4,3.6,3.7, and 3.8 we get the following hierarchies:

$$\mathbf{REG_{full}} \subset \mathbf{REG}_{(\forall)} \subset \mathbf{REG} = \mathbf{REG}_{(\forall\sigma)} \subset \mathbf{REG_{max}} \subset \mathbf{REG}_{(\exists\sigma)} \subset \mathbf{REG}_{(\exists)}$$

$$\mathbf{REG} \setminus \mathbf{REG_{full}} = \mathbf{REG_{gsm}} \subset \mathbf{REG} = \mathbf{REG}_{(\forall\sigma)}$$

Finally, the closure properties of the defined classes are summarized in the following table. Note that y (respectively, n) at the intersection of the row associated with the class $\mathcal{C}$ and the column associated with the operation $\circ$ means that $\mathcal{C}$ is closed (respectively, not closed) under operation $\circ$. A special case is the closure of $\mathbf{REG_{max}}$ under union and concatenation: in general this class is not closed under these operations, but when we apply them to languages of $\mathbf{REG_{max}}$ over the same alphabet we get a language from the same class.

Class	$\cup$	$\cap$	$\cap$**REG**	$\mathbf{alph}(L)^* \setminus L$	$*$	$\cdot$	ϕ	ϕ^{-1}	σ
$\mathbf{REG}_{(\forall)}$	y	y	y	n	n	n	n	n	n
$\mathbf{REG} = \mathbf{REG}_{(\forall\sigma)}$	y	y	y	y	y	y	y	y	y
$\mathbf{REG_{max}}$	n/y	n	n	n	y	n/y	n	n	n
$\mathbf{REG}_{(\exists\sigma)}$	n	n	n	n	y	n	n	n	n
$\mathbf{REG}_{(\exists)}$	y	n	n	y	y	y	n	n	n

References

[1] F. Blanchet-Sadri, *Algorithmic Combinatorics on Partial Words*. Chapman & Hall/CRC Press, 2008.

[2] M. J. Fischer, M. S. Paterson, String matching and other products. In: *Complexity of Computation, SIAM-AMS Proceedings*. 7, 1974, 113–125.

[3] G. Lischke, Restoration of punctured languages and similarity of languages. *Mathematical Logic Quarterly* **52** (2006) 1, 20–28.

[4] F. Manea, R. Mercaş, Freeness of partial words. *Theoretical Computer Science* **389** (2007) 1-2, 265–277.

[5] G. Rozenberg, A. Salomaa, *Handbook of Formal Languages*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 51 – 56

Fast Descriptive Generalization of Restricted Regular Expressions and DTDs

Dominik D. Freydenberger$^{(A)}$ Timo Kötzing$^{(B)}$

$^{(A)}$Institut für Informatik, Goethe-Universität, Frankfurt am Main
freydenberger@em.uni-frankfurt.de

$^{(B)}$Max-Planck-Institute for Informatics, Saarbrücken
koetzing@mpi-inf.mpg.de

Abstract

We study the problem of generalizing from a finite sample to an infinite language taken from a predefined language class. The two language classes we consider are subsets of the regular languages and have significance in the specification of XML documents (the classes corresponding to so called *chain regular expressions*, CHAREs, and to *single occurrence regular expressions*, SOREs).

The previous literature gave a number of algorithms for generalizing to SOREs providing a trade off between generalization speed and quality of the solution. Furthermore, a fast but non-optimal algorithm for generalizing to CHAREs is known.

For each of the two language classes we give an *efficient* algorithm returning a *minimal* generalization from the given finite sample to an element of the fixed language class; such generalizations are called *descriptive*. In this sense, both our algorithms are optimal.

1. SOREs, CHAREs, and Descriptive Generalization

The present paper provides a short overview of [5], which refines an approach for XML schema inference from positive examples that was introduced by Bex et al. [3]. The basic problem setting is as follows: Given a set of XML documents, generate a schema that describes these documents, while being compact and preferably human readable.

Bex et al. approach this problem by learning deterministic regular expressions from positive examples; i. e., they consider the following problem: Given a finite set S of positive examples from an unknown target language L, find a deterministic regular expression for L. These regular expressions can immediately be used as DTDs (Document Type Definitions), and while XSDs (XML Schema Documents) require additional effort, algorithms that infer regular expressions can also be used as a component of XSD inference algorithms (see [3, 4] for further explanations). In particular, as argued in [3], the results in [11] show that XSD inference requires deep insights into regular expression inference – as Bex et al. put it, "one cannot hope to successfully infer XSDs without good algorithms for inferring regular expressions".

$^{(A)}$This work was done while this author was visiting the Max-Planck-Institute for Informatics in Saarbrücken.

Using a classical technique from Gold [9], Bex et al. prove in [2] that even the class of deterministic regular expressions is too rich to be learnable from positive data. While, strictly speaking, the learnability criterion of *Gold style learning* as defined in [9] (which is also called *learning in the limit from positive data* or *explanatory learning*) is different from the setting in [2, 3],[1] its non-learnability results still provide valuable insights into necessary restrictions.

In particular, Gold style learning shows that, when learning from positive data, one has to balance the need for generalization (as in most cases, a regular expression that generates exactly the example is not considered a good hypothesis) with the need to avoid overgeneralization.

While there are numerous papers on restrictions on the class of regular languages that lead to learnability, apart from a few exceptions (e. g. [6]), most of these restrictions prior to [3] have been based on properties of automata. As explained in [3], this is problematic, as even under those restrictions, converting the inferred automaton to a regular expression can lead to an exponential size increase.

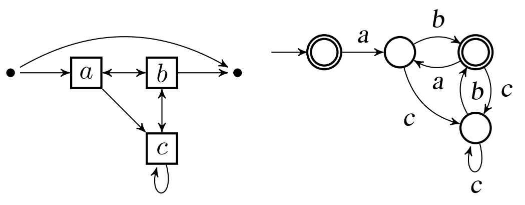
In order to achieve learnability of concise deterministic regular expression, Bex et al. propose *single occurrence regular expressions* (short SOREs), regular expressions where each *terminal letter* (or *element name*) occurs at most once.

These SOREs are deterministic by definition, and as an additional benefit, this restriction ensures that the length of the inferred expressions is at most linear in the number of different terminal letters.

The corresponding SORE-inference algorithm RWR from [3] works as follows: First, it constructs a so-called *single occurrence automaton* (short SOA, as introduced by García and Vidal [8]).

Basically, SOAs are a subclass of DFAs; namely those DFAs where for each $a \in \Sigma$, there exists a characteristic state q_a such that $\delta(q, a) = q_a$ for all states $q \in Q$. This is illustrated by the following example:

Example 1.1 *In the picture below, we have a SOA in SOA notation on the left side, and the corresponding DFA to the right side.*



As every input letter leads to a characteristic successor state, we can represent SOAs more compactly by moving the letters from the edges into the states. Both automata generate the same language as the regular expression $\alpha = ((ac^+?b)((ac^+?b) \mid (c^+b))^+?)?$. Following a notation from [3] (which is common when working with XML), we use $?$ to denote $\mid \epsilon$.

[1]Gold style learning uses a growing set of samples and requires that the learner converges toward a correct hypothesis in finite time, while this setting uses only a single finite set for each inference instance.

Note that α is not a SORE*. In fact, $L(\alpha)$ is not a* SORE*-language, but proving this using elementary techniques requires considerable effort. (The most straightforward way to prove this is to use the conversion algorithm* `Soa2Sore` *from [5] on the* SOA*, which returns the* SORE $(ab?c^+?)^+?$*, which is not equivalent to α.)*

`RWR` then attempts to convert the SOA step by step into a SORE. As the class of SORE-languages is a proper subset of the class of SOA-languages, this conversion is not always possible. In these cases, `RWR` attempts to repair the SOA, and constructs a SORE that generates a generalization of the language of the SOA. In order to generalize as little as possible, [3] suggests different orderings on the set of repair rules, as well as the variant $\mathtt{RWR}^2_\ell$, which uses additional heuristics and can have an exponential running time. Nonetheless, these variants may still infer SOREs that are not inclusion-minimal generalizations of the input sample (within the class of all SOREs).

In order to deal with insufficient data, Bex et al. proposed a further restriction on SOREs, the so-called *chain regular expressions* (short: CHAREs):

Definition 1.2 *A* chain regular expression *(or* CHARE*) is a* SORE *is of the form $f_1 \cdot \ldots \cdot f_n$ ($n \geq 0$), where each f_i is a* chain factor*, i. e., a* SORE *of the form $(a_1 \mid \cdots \mid a_k)$, $(a_1 \mid \cdots \mid a_k)?$, $(a_1 \mid \cdots \mid a_k)^+$, or $(a_1 \mid \cdots \mid a_k)^+?$, where $k \geq 1$, and each a_j is a terminal letter.*

Bex et al. introduced the corresponding inference algorithm `CRX`. Analogously to `RWR`, `CRX` may infer CHAREs that are not inclusion-minimal generalizations.

The paper [5] focuses on inferring SOREs and CHAREs that are inclusion-minimal generalizations. This approach to regular expression inference is based on a slightly different angle than Gold style learning, namely on the learning paradigm of *descriptive generalization* that was introduced by Freydenberger and Reidenbach [7].

While Gold style learning assumes that an exact representation of the target language is present in the hypothesis space, and that the learner is provided with sufficient positive information to correctly recognize the target language, descriptive generalization views the hypothesis space and the space of target languages as distinct.

Definition 1.3 *For a class $\mathcal{D}$ of language representation mechanisms (e. g., a class of automata, regular expressions, or grammars[2]), a language representation $\delta \in \mathcal{D}$ is called $\mathcal{D}$-*descriptive *of a language L if 1. $L \subseteq L(\delta)$, and 2. there is no $\gamma \in \mathcal{D}$ with $L \subseteq L(\gamma) \subset L(\delta)$.*

This concept allows us to define $\mathcal{D}$-descriptive generalization as a natural extension of Gold style learning: Instead of attempting to learn an exact representation of the target language L from a sample S, the learner has to infer a representation $\delta \in \mathcal{D}$ that is $\mathcal{D}$-descriptive of L. In other words, δ is a generalization of S that is as *inclusion-minimal* as possible within $\mathcal{D}$.

Descriptive generalization explicitly separates the hypothesis space from the class of target languages, while still providing a natural quality criterion for generalization from positive examples. In the approach presented in this paper, we consider the class of SOREs and the

[2]The canonical class $\mathcal{D}$ is the class of *NE-patterns*, where *descriptive patterns* were introduced by Angluin [1] in the context of exact learning from positive data. See [12] for a survey on the influence of pattern languages in this area.

class of CHAREs as hypothesis spaces $\mathcal{D}$, and examine the problem of inferring $\mathcal{D}$-descriptive generalizations from finite samples.

We approach this problem by first computing a SOA-descriptive SOA. As proven in [5], this approach has the advantages that the descriptive SOA is uniquely defined, can be computed efficiently, and its language is included in the language of every descriptive SORE or CHARE.

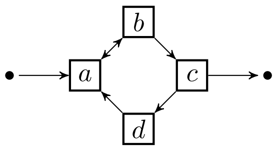
The main contribution of [5] are two algorithms, `Soa2Sore` and `Soa2Chare`, that can be used to transform any given SOA A into a SORE (resp. CHARE) that is SORE-descriptive (resp. CHARE-descriptive) of the language of $L(A)$. That is, given a sample S, these algorithms can be used to compute a generalization of S that is inclusion-minimal (or, in the terminology of [3], *optimal*) within the class of SOREs or CHAREs (respectively).

In addition to this, `Soa2Chare` and `Soa2Sore` are efficient: `Soa2Chare` runs in time $O(m)$ (compared to $O(m+n^3)$ for `CRX`), `Soa2Sore` in time $O(nm)$ (compared to $O(n^5)$ for `RWR`), where m is the number of edges and n the number of nodes in the SOA.

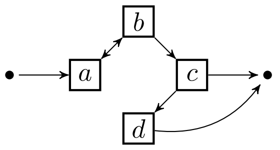
In [3], Bex et al. state that their schema inference algorithms "outperform existing algorithms in accuracy, conciseness, and speed". Considering the results presented in [5], the authors of the present paper feel confident to suggest that their new strategies outperform the algorithms from [3] with respect to both accuracy and speed, and are on par with respect to conciseness. An experimental evaluation of implementations of the algorithms is planned for the near future.

2. An Exemplary Run of `Soa2Sore`

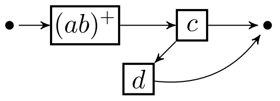
This section demonstrates how the algorithm `Soa2Sore` turns a SOA into a descriptive SORE. As the algorithm had to be omitted for space reasons, this example can only provide a very general feel of its behavior. Nonetheless, the reader might find it illuminating (and the pictures are quite elegant). Consider the sample $S = \{ababc, abcdabc\}$. The following SOA is the only SOA that is SOA-descriptive of S.



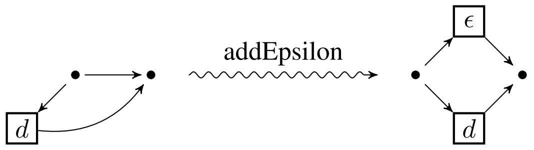
The labeled vertices of this SOA consist of a single strongly connected looped component, an application of "bend" computes the set $W = \{a, b\}$, which leads to the following SOA.



After resolving the strongly connected looped component containing a and b (all other are not "looped") and contract, we get the following.



We can split off the first node twice now, recursing finally on the remaining SOA as follows.



This results in $d \,|\, \epsilon$, or, equivalently, $d?$. Going back through the recursions, we get

$$((ab)^+cd?)^+.$$

This SORE is SORE-descriptive for the language of the input SOA, and for the sample S.

3. Beyond SOREs and CHAREs

From the authors' point of view, the following problem is probably the most interesting: In [2], Bex et al. examine the inference of *k-occurence regular expressions* (short k-OREs); regular expressions where each terminal letter occurs at most k times. (Hence, SOREs are 1-OREs). Is it possible to extend Soa2Sore to deterministic k-OREs for some $k \geq 2$, or Soa2Chare to the corresponding extension of CHAREs (where letters are allowed to occur up to k times)?

It seems that one would need to develop not only a good generalization of SOAs, but also a "good" inclusion criterion, preferably syntactic. This conjecture is based on the following observation: While the results in [5] make no direct use of the results and techniques that Freydenberger and Reidenbach [7] developed for descriptive generalization of pattern languages, both papers rely heavily on the fact that the inclusion problem for the respective language classes has a syntactic criterion for inclusion.

The proofs on descriptive generalization of pattern languages in [7] rely on the fact that inclusion for terminal-free E-pattern languages is characterized by the existence of a morphism which maps the pattern that generates the superlanguage to the pattern that generates the sublanguage. This criterion is a versatile tool to prove the nonexistence of a (pattern) language between the target language and the language of a descriptive pattern. While the proofs in [5] cannot make any *direct* use of the proofs from [7], the approaches are similar *conceptually*. In particular, the line of reasoning in which the correctness proofs of Soa2Chare and Soa2Sore use the fact that the inclusion problem for SOREs (and CHAREs) is characterized by the covering of the respective SOAs is structurally similar to the proofs for pattern languages.

Moreover, although deciding whether such a pattern morphism exists is an NP-complete problem, the techniques in [7] are not affected by the computational hardness. Hence, the hardness results on the decidability of the k-ORE-inclusion problem presented by Martens et al. [10] do not exclude the existence of such a criterion. This leaves room for hope that Soa2Sore can be extended to k-OREs with $k \geq 2$.

References

[1] D. ANGLUIN, Finding Patterns Common to a Set of Strings. *Journal of Computer and System Sciences* **21** (1980) 1, 46–62.

[2] G. J. BEX, W. GELADE, F. NEVEN, S. VANSUMMEREN, Learning Deterministic Regular Expressions for the Inference of Schemas from XML Data. *ACM Transactions on the Web* **4** (2010) 4, 14:1–14:32.

[3] G. J. BEX, F. NEVEN, T. SCHWENTICK, S. VANSUMMEREN, Inference of concise regular expressions and DTDs. *ACM Transactions on Database Systems* **35** (2010) 2, 11:1–11:47.

[4] G. J. BEX, F. NEVEN, S. VANSUMMEREN, Inferring XML Schema Definitions from XML Data. In: *Proc. VLDB 2007*. 2007, 998–1009.

[5] T. K. D. D. FREYDENBERGER, Fast Learning of Restricted Regular Expressions and DTDs. Submitted.

[6] H. FERNAU, Algorithms for learning regular expressions from positive data. *Information and Computation* **207** (2009) 4, 521–541.

[7] D. D. FREYDENBERGER, D. REIDENBACH, Inferring Descriptive Generalisations of Formal Languages. In: *Proc. COLT 2010*. 2010, 194–206.

[8] P. GARCÍA, E. VIDAL, Inference of k-Testable Languages in the Strict Sense and Application to Syntactic Pattern Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12** (1990) 9, 920–925.

[9] E. M. GOLD, Language Identification in the Limit. *Information and Control* **10** (1967) 5, 447–474.

[10] W. MARTENS, F. NEVEN, T. SCHWENTICK, Complexity of Decision Problems for XML Schemas and Chain Regular Expressions. *SIAM Journal on Computing* **39** (2009) 4, 1486–1530.

[11] W. MARTENS, F. NEVEN, T. SCHWENTICK, G. J. BEX, Expressiveness and complexity of XML Schema. *ACM Transactions on Database Systems* **31** (2006) 3, 770–813.

[12] Y. K. NG, T. SHINOHARA, Developments from enquiries into the learnability of the pattern languages from positive data. *Theoretical Computer Science* **397** (2008) 1–3, 150–165.

Finding Pseudo-Repetitions

Paweł Gawrychowski[A] Florin Manea[B] Robert Mercaş[C]
Dirk Nowotka[B] Cătălin Tiseanu[D]

[A]Max-Planck-Institute für Informatik,
Saarbrücken, Germany, gawry@cs.uni.wroc.pl

[B]Christian-Albrechts-Universität zu Kiel, Institut für Informatik,
D-24098 Kiel, Germany, {flm,dn}@informatik.uni-kiel.de

[C]Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik,
PSF 4120, D-39016 Magdeburg, Germany, robertmercas@gmail.com

[D]University of Maryland at College Park, Computer Science Department,
A.V. Williams Bldg., College Park, MD 20742, USA, ctiseanu@umd.edu

Abstract

Pseudo-repetitions are a generalization of the fundamental notion of repetitions in sequences, considered initially in the framework of DNA computing and bioinformatics. We develop the algorithmic foundations for questions on pseudo-repetitions by nontrivial application of combinatorial results on words.

1. Introduction

The notions of repetition and primitivity are fundamental concepts on sequences used in a number of fields, among them being stringology and algebraic coding theory. A word is a repetition (or power) if it can be expressed as a repeated catenation of one of its prefixes. We consider a more general concept here, namely *pseudo-repetitions in words*. A word w is a pseudo-repetition if it can be written as a repeated catenation of one of its prefixes t and its image $f(t)$ under some morphism or antimorphism (for short "anti-/morphism") f, thus $w \in t\{t, f(t)\}^+$.

Pseudo-repetitions, introduced in a restricted form by Kari et al.[2], lacked so far a developed algorithmic part, something usually quite important in the field this theory originates from – bioinformatics. This work is aimed to fill this gap. We investigate the following two basic algorithmic problems: decide whether a word w is a pseudo-repetition for some given morphism f and find all k-powers of pseudo-repetitions occurring as factors in a word, for some given f. We establish algorithms and complexity bounds for these problems for various types of morphisms thereby improving significantly the results from [1]. Apart from the application of standard tools of string algorithms, like suffix arrays, we extend the toolbox by nontrivial applications of results from combinatorics on words.

[B]The work of Florin Manea is supported by the *DFG* grant 582014. The work of Dirk Nowotka is supported by the *DFG Heisenberg* grant 590179.

[C]The work of Robert Mercaş is supported by the *Alexander von Humboldt Foundation*.

1.1. Background and Motivation

The motivation of introducing pseudo-repetition and pseudo-primitivity in [2] originated from the field of computational biology, namely the facts that the Watson-Crick complement can be formalized as an antimorphic involution and both a single-stranded DNA and its complement (or its image through such an involution) basically encode the same information. Until now, pseudo-repetitions were considered only in the cases of anti-/morphic involutions, following the original motivation, and the results obtained were mostly of combinatoric nature (e.g., generalizations of the Fine and Wilf theorem).

A natural extension of these concepts is to consider antimorphisms and morphisms in general, which is done in this paper. Considering that the notion of repetition is central in the study of combinatorics of words, and the plethora of applications that this concept has, the study of pseudo-repetitions seems even more attractive, at least from a theoretical point of view. While the biological motivation seems appropriate only for the case when f is an antimorphic involution, one can imagine a series of real-life scenarios where we are interested in identifying factors that can be written as an iterated catenation of a word and its encoding through some simple function f. Indeed, pseudo-repetitions can be seen as strings that have an intrinsic repetitive structure, hidden by rewriting some of the factors that define it through some anti-/morphism.

1.2. Some Basic Concepts

For more detailed definitions we refer to the handbook [4].

Let V be a finite alphabet. We denote by V^* the set of all words over V and by V^k the set of all words of length k. The *length* of a word $w \in V^*$ is denoted by $|w|$. The *empty word* is denoted by λ. Moreover, we denote by alph(w) the alphabet of all letters that occur in w.

A function $f : V^* \to V^*$ is a morphism if $f(xy) = f(x)f(y)$ for any words x and y over V. Further, f is an antimorphism if $f(xy) = f(y)f(x)$ for all $x, y \in V^*$. Note that, when we define a morphism or an antimorphism it is enough to give the definitions of $f(a)$, for all $a \in V$. An anti-/morphism $f : V^* \to V^*$ is an involution if $f^2(a) = a$ for all $a \in V$. We say that f is *uniform* if there exists a number k with $f(a) \in V^k$, for all $a \in V$; if $k = 1$ then f is called *literal*. If $f(a) = \lambda$ for some $a \in V$, then f is called *erasing*, otherwise *non-erasing*.

The powers of a word w are defined recursively by $w^0 = \lambda$ and $w^n = ww^{n-1}$ for $n \geq 1$. If w cannot be expressed as a nontrivial power of another word, then w is *primitive*. We say that a word w is an *f-repetition*, or, alternatively, an f-power, if w is in $t\{t, f(t)\}^+$, for some prefix t of w. If w is not an f-power, then w is *f-primitive*.

As an example, the word $abcaab$ is primitive from the classical point of view (i.e., **1**-primitive, where **1** is the identical morphism) as well as f-primitive, for the morphism f defined by $f(a) = b$, $f(b) = a$ and $f(c) = c$. However, when considering the morphism $f(a) = c$, $f(b) = a$ and $f(c) = b$, we get that $abcaab$ is the catenation of ab, $ca = f(ab)$, and ab, thus, being an f-repetition.

Finally, we stress out that the computational model we use to design and analysis of our algorithms is the unit-cost RAM (Random Access Machine) with logarithmic word size.

2. Algorithmic problems and results

In the upcoming algorithmic problems, when we are given as input a word w of length n we assume that the symbols of w are in fact integers from $\{1,\ldots,n\}$ (i.e., alph$(w) \subseteq \{1,\ldots,n\}$), and w is seen as a sequence of integers. This is a common assumption in algorithmic on words (see, e.g., the discussion in [3]).

In the first one, which is probably the most interesting in the general context of pseudo-repetitions, we are interested in deciding whether a word is an f-repetition, for some given anti-/morphism f whose size is assumed to be constant.

Problem 1 *Let $f: V^* \to V^*$ be an anti-/morphism. Given $w \in V^*$, decide whether for some word t we have $w \in t\{t, f(t)\}^+$.*

We solve this problem in the general case in time $\mathcal{O}(n \lg n)$. However, in the particular case of uniform anti-/morphisms we obtain an optimal solution running in linear time. The latter includes the biologically motivated case of involutions from [2]. Further, we extend our results to a more general form of Problem 1, testing whether $w \in \{t, f(t)\}\{t, f(t)\}^+$. Except for the most general case (of anti-/morphisms that can also erase letters), when we solve this problem in $\mathcal{O}(n^{1+\frac{1}{\lg\lg n}} \lg n)$ time , in the rest of the cases we are able to preserve the same time complexity as for the particular form of the problem.

Two other natural problems are related to identifying the factors of a word which are pseudo-repetitions. The first one was originally considered in [1]; the second generalizes it. Both these problems are related to testing the fundamental combinatorial property of freeness of words, in the context of pseudo-repetitions.

Problem 2 *Let $f: V^* \to V^*$ be an anti-/morphism and $w \in V^*$ a word.*
(1) Given the number k enumerate all pairs (i,j) such that $w[i..j] \in \{t, f(t)\}^k$.
(2) Enumerate all triples (i,j,ℓ) such that there exists t with $w[i..j] \in \{t, f(t)\}^\ell$.

Our approach to the second question of the problem is based on constructing data structures that enable us to obtain in constant time the answer to queries $rep(i,j,\ell)$: "Is there $t \in V^*$ such that $w[i..j] \in \{t, f(t)\}^\ell$?", for $1 \leq i \leq j \leq |w|$ and $1 \leq \ell \leq |w|$. In the general case, one can produce in $\mathcal{O}(n^3\sqrt{n})$ time such data structures. When f is non-erasing, the time needed to construct such data structures is $\mathcal{O}(n^3)$, while when f is a literal anti-/morphism we can do it in time $\Theta(n^2)$. In every case, once we have these structures, we can identify in $\Theta(n^3)$ time the triples (i,j,ℓ) such that $w[i..j] \in \{t, f(t)\}^\ell$, answering question (2) in Problem 2. Using these data structure we obtain, when f is non-erasing (respectively, literal), an algorithm that solves this part of the problem in $\Theta(n^3)$ (respectively, $\Theta(n^2 \lg n)$) time and show that there are input words on which every algorithm solving question (2) has a running time asymptotically equal to ours (this time, including the preprocessing time). Unfortunately, the time bound obtained for most general case is not tight.

The same data structures can be used in the simplest case, of literal anti-/morphisms, to answer question (1) of Problem 2. We obtain an algorithm that outputs in quadratic time, for a given w and k, all pairs (i,j) such that $w[i..j] \in \{t, f(t)\}^k$; again, this time bound is shown to be tight as we exhibit words w and all sorts of (general, non-erasing uniform, or, respectively, literal) anti-/morphisms f for which w has $\Theta(n^2)$ factors from $\{t, f(t)\}^k$. By taking advantage of the fact that k is given as input (so fixed throughout the algorithm) we can refine our approach to solve question (2) in order to obtain a quadratic solution of question (1) for f non-erasing, which is a tight time bound, and a solution running in $\mathcal{O}(n^2k)$ for the general case.

The results reported in this abstract improve significantly the algorithmic results from in [1, 5].

References

[1] E. CHINIFOROOSHAN, L. KARI, Z. XU, Pseudopower avoidance. *Fundamenta Informaticae* **114** (2012) 1, 55–72.

[2] E. CZEIZLER, L. KARI, S. SEKI, On a special class of primitive words. *Theoretical Computer Science* **411** (2010), 617–630.

[3] J. KÄRKKÄINEN, P. SANDERS, S. BURKHARDT, Linear work suffix array construction. *J. ACM* **53** (2006), 918–936.

[4] M. LOTHAIRE, *Combinatorics on Words*. Cambridge University Press, 1997.

[5] F. MANEA, R. MERCAŞ, C. TISEANU, Algorithms and Pseudo-Periodicity in Words. *Proc. Der 21. Theorietag der Fachgruppe Automaten und Formale Sprachen* (2011).

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 61 – 64

From Equivalence to Almost-Equivalence, and Beyond—Minimizing Automata With Errors

Markus Holzer[(A)] Sebastian Jakobi[(A)]

[(A)]Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
{holzer,jakobi}@informatik.uni-giessen.de

The study of the minimization problem for finite automata dates back to the early beginnings of automata theory. It is well known that for a given n-state deterministic finite automaton (DFA) one can efficiently compute an equivalent minimal automaton in $O(n \log n)$ time [9]. More precisely, the DFA-to-DFA minimization problem is complete for NL, even for DFAs without inaccessible states [4]. This is contrary to the nondeterministic case since the nondeterministic finite automaton (NFA) minimization problem is known to be computationally hard [10]. Minimization remains intractable even if either the input or the output automaton is deterministic [10, 12]. Recently another form of minimization for DFAs, namely hyper-minimization, was considered in the literature [2, 3, 6, 8]. While minimization aims to find an equivalent automaton that is as small as possible, hyper-minimization intends to find an almost-equivalent automaton that is as small as possible. Here two languages are considered to be *almost-equivalent*, if they are equivalent up to a finite number of exceptions. Thus, an automaton is hyper-minimal if every other automaton with fewer states disagrees on acceptance for an *infinite* number of inputs. Hence, equivalence or almost-equivalence can be interpreted as an "*error profile*:" minimization becomes exact compression and hyper-minimization is a sort of lossy compression. We provide a general framework for error profiles of automata. To this end we introduce the concept of E-equivalence. Two languages L_1 and L_2 are E-*equivalent* ($\sim_E$) for some language E, if their symmetric difference lies in E, i.e., $L_1 \triangle L_2 \subseteq E$. Here E is called the *error* language. A close inspection shows that E-equivalence allows us to cover a lot of prominent "equivalence" concepts from the literature such as, e.g., equivalence ($\equiv$)—set $E = \emptyset$ or almost-equivalence ($\sim$)—E is finite. A detailed discussion on this subject is given in the full version of the paper. We study the computational complexity of minimizing finite automata with respect to the language relations of almost-equivalence, and E equivalence, as well as the complexity of some related decision and counting problems.

The decision version of the classical DFA-to-DFA minimization problem is defined as follows: given a DFA A and an integer n, does there exist an *equivalent* n-state DFA B? This notation naturally generalizes to other types of finite automata. The DFA-to-DFA minimization problem is complete for NL, even for DFAs without inaccessible states [4]. This is contrary

This is an extended abstract of: M. Holzer, S. Jakobi. *From Equivalence to Almost-Equivalence, and Beyond—Minimizing Automata With Errors (Extended Abstract).* In H.-C. Yen, O. H. Ibarra (eds.), *Proceedings of the 16th International Conference on Developments in Language Theory*. Number 7410 in LNCS, Springer, Taipei, Taiwan, 2012, 190–201.

Equivalence relation	Minimization problem			
	DFA-to-...		NFA-to-...	
	DFA	NFA	DFA	NFA
$\equiv$	NL	PSPACE	PSPACE	
$\sim$	NL	PSPACE	PSPACE	
$\sim_E$, for DFA A_E with $E = L(A_E)$	NP	PSPACE	PSPACE	

Table 1: Results on the computational complexity of minimizing finite automata with respect to different equivalence relations. The input to all problems is a finite automaton A and an integer n, and the question is, whether there exists an n-state finite automaton B, that is in the corresponding relation to A. For the problems on E-minimization, a DFA A_E specifying the error language E is given as additional input.

to the nondeterministic case since the NFA minimization problem is known to be PSPACE-complete [10], even if the input is given as a DFA. When turning to minimization with respect to almost-equivalence, or E-equivalence, the results mirror those for ordinary DFA and NFA minimization, with some notable exceptions. For instance, hyper-minimizing deterministic machines, that is the DFA-to-DFA minimization problem w.r.t. almost-equivalence, is shown to be NL-complete while E-minimization of DFAs in general turns out to be NP-complete, even for some finite E. Our results on the complexity of minimization w.r.t. different equivalence relations are summarized, and compared to the classical results in Table 1. In the following we present our result on the E-minimization problem.

Theorem 1.1 (E-Minimization) *The problem of deciding for two given DFAs A and A_E, and an integer n, whether there exists a DFA B with n states, such that $A \sim_E B$, for $E = L(A_E)$, is* NP*-complete. This even holds, if the language E is finite. The problem becomes* PSPACE*-complete for NFAs, even if the input is given as a DFA.*

Proof. [Sketch] We only sketch the proof for NP-completeness of the DFA-to-DFA E-minimization. Since $A \sim_E B$ can be verified for DFAs in deterministic polynomial time, by constructing a DFA for the language $(L(A) \triangle L(B)) \cap \overline{E}$, and checking this language for emptiness, the problem description gives rise to a straightforward guess-and-check algorithm on a nondeterministic polynomial time bounded Turing machine.

For NP-hardness we use a reduction from MONOTONE 3SAT [5]. Given a Boolean formula $\varphi = c_0 \wedge c_1 \wedge \cdots \wedge c_{k-1}$ with variables $X = \{x_0, x_1, \ldots, x_{n-1}\}$, where each c_i is either a positive clause of the form $c_i = (x_{i_1} \vee x_{i_2} \vee x_{i_3})$ or a negative clause of the form $c_i = (\neg x_{i_1} \vee \neg x_{i_2} \vee \neg x_{i_3})$, we construct a DFA $A = (Q \cup P \cup \{r, f, s\}, \{a, b, c\}, \delta, q_0, \{f\})$, where $Q = \{q_0, q_1, \ldots, q_{k-1}\}$, and $P = \{p_0, p_1, \ldots, p_{n-1}\}$. Its transition function δ is depicted in Figure 1. The integer for the E-minimization instance is set to $n+k+2$, which is exactly one less than the number of states in A. Finally, the finite error language is

$$E = \{\, a^i b a^{n-j} \mid 0 \leq i \leq k-1,\ c_i \text{ contains } x_j \text{ or } \neg x_j \,\} \cup \{\, a^i b a^j b, a^i b a^j c \mid 0 \leq i \leq k-1,\ 1 \leq j \leq n-1 \,\} \cup \{\, a^{k+j} b \mid 0 \leq j \leq n-1 \,\}.$$

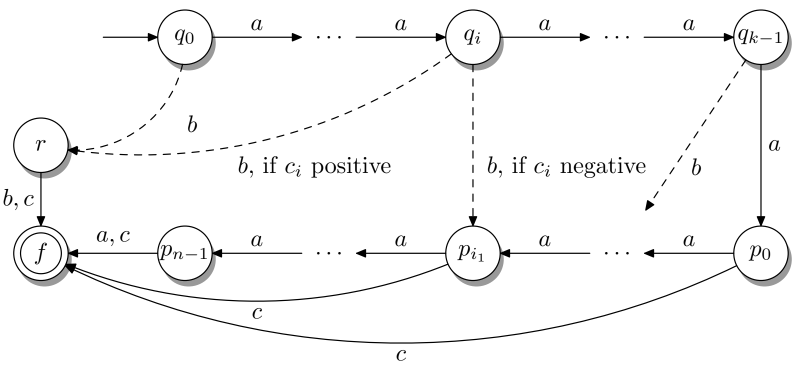


Figure 1: The DFA A constructed from the Boolean formula φ. The b-transitions from states $q_0, q_1, \ldots, q_{k-1}$ are only sketched—it is $\delta(q_i, b) = p_{i_1}$, if $c_i = (\neg x_{i_1} \vee \neg x_{i_2} \vee \neg x_{i_3})$, and $\delta(q_i, b) = r$ otherwise. All undefined transitions go to the sink state s, which is not shown.

A DFA A_E accepting this language can easily be constructed in polynomial time.

One can show that φ is satisfiable if and only if there exists a DFA B, with $A \sim_E B$, that has $n+k+2$ states—in this case, only state r is missing. The overall idea is the following. Since every word in E contains at least one b symbol, the error set does not allow E-equivalent automata to differ on inputs a or c. Further, since words $a^i bb$ and $a^i bc$ with $0 \leq i \leq k-1$ do not belong to E, the b-transitions from states q_i must end in states p_j, and the b-transitions from p_j must end in state f or the sink state s. The connection to φ is the following: a state p_i, corresponding to variable x_i, goes to state f on input b if and only if the variable x_i should be assigned the Boolean value 1. And a state q_i, corresponding to a clause c_i, goes to state p_j on input b if and only if the clause c_i gets satisfied by the variable x_j. In this way, any E-minimal DFA B, with $A \sim_E B$, corresponds to a satisfying truth assignment for φ, and *vice versa*. $\square$

We also consider problems that are related to minimization, such as deciding equivalence, minimality, or canonicity for given automata. The latter problem results from the fact that, although in general a hyper-minimal, or E-minimal automaton can be smaller than the classical minimal automaton, there are also languages, whose minimal automaton is already hyper-minimal, or E-minimal. If this is the case, the automaton (or rather its accepted language) is called canonical, or E-canonical. Although E-equivalence is a generalization of equivalence, and almost-equivalence, the problems to decide whether two languages given by finite automata are equivalent, almost-equivalent, or E-equivalent, respectively, are all of same complexity. To be more precise, whenever NFAs are involved in the language specification the decision problem is PSPACE-complete, while for DFAs it is NL-complete. In contrast to this, the problems of deciding, whether a given automaton is hyper-minimal, or E-minimal, are more closely related to minimization, which shows up in their complexity: deciding hyper-minimality of DFAs is NL-complete, just as deciding minimality, while deciding E-minimality is coNP-complete. This asymmetry can also be observed for canonicity.

Finally we focus on counting problems on finite automata; see, e.g., [1, 7, 11]. It is well known that for each regular language there is a unique minimal DFA (up to isomorphism) accepting this language. Since hyper-minimal and E-minimal DFAs are not necessarily unique anymore, we are led with the following counting problem: given a DFA A, what is the number of hyper-minimal DFAs B, with $A \sim B$? Naturally, this generalizes to determine the number of E-minimal automata. We show that there is again a significant difference between the computational complexities of questions concerning almost- and E-equivalence.

Theorem 1.2 (Counting minimal DFAs) *Given a DFA A, then the number of hyper-minimal DFAs B with $A \sim B$ can be done in* FP*, while determining the number of E-minimal DFAs B, satisfying $A \sim_E B$ and $E = L(A_E)$, here the DFA A_E is given as additional input, is* $\#\mathsf{P}$*-hard and can be computed in* $\# \cdot \mathsf{coNP}$.

One could also count minimal, hyper-minimal, or E-minimal NFAs instead. These three counting problems belong to $\#\mathsf{PSPACE} = \mathsf{FPSPACE}$ [11]. We have to leave open the lower bound for the complexity of counting minimal and hyper-minimal NFAs, but for counting the number of E-minimal NFAs, we can prove $\#\mathsf{P}$-hardness.

References

[1] C. ÀLVAREZ, B. JENNER, A very hard log-space counting class. *Theoret. Comput. Sci.* **107** (1993) 1, 3–30.

[2] A. BADR, Hyper-Minimization in $O(n^2)$. *Internat. J. Found. Comput. Sci.* **20** (2009) 4, 735–746.

[3] A. BADR, V. GEFFERT, I. SHIPMAN, Hyper-Minimizing Minimized Deterministic Finite State Automata. *RAIRO–Informatique théorique et Applications / Theoretical Informatics and Applications* **43** (2009) 1, 69–94.

[4] S. CHO, D. T. HUYNH, The Parallel Complexity of Finite-State Automata Problems. *Inform. Comput.* **97** (1992), 1–22.

[5] M. R. GAREY, D. S. JOHNSON, *Computers and Intractability, A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[6] P. GAWRYCHOWSKI, A. JÉZ, Hyper-Minimization Made Efficient. In: R. KRÁLOVIC, D. NIWINSKI (eds.), *Proceedings of the* 34*th Conference on Mathematical Foundations of Computer Science*. Number 5734 in LNCS, Springer, Novy Smokovec, High Tatras, Slovakia, 2011, 356–368.

[7] M. HOLZER, On emptiness and counting for alternating finite automata. In: J. DASSOW, G. ROZENBERG, A. SALOMAA (eds.), *Developments in Language Theory II; at the Crossroads of Mathematics, Computer Science and Biology*. World Scientific, 1996, 88–97.

[8] M. HOLZER, A. MALETTI, An $n \log n$ Algorithm for Hyper-Minimizing a (Minimized) Deterministic Automaton. *Theoret. Comput. Sci.* **411** (2010) 38–39, 3404–3413.

[9] J. HOPCROFT, An $n \log n$ algorithm for minimizing the state in a finite automaton. In: Z. KOHAVI (ed.), *The Theory of Machines and Computations*. Academic Press, New York, 1971, 189–196.

[10] T. JIANG, B. RAVIKUMAR, Minimal NFA problems are hard. *SIAM J. Comput.* **22** (1993) 6, 1117–1141.

[11] R. E. LADNER, Polynomial Space Counting Problems. *SIAM J. Comput.* **18** (1989) 6, 1087–1097.

[12] A. R. MEYER, L. J. STOCKMEYER, The equivalence problem for regular expressions with squaring requires exponential time. In: *Proceedings of the* 13*th Annual Symposium on Switching and Automata Theory*. IEEE Society Press, 1972, 125–129.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 65 – 70

State Complexity of Chop Operations on Unary and Finite Languages

Markus Holzer[(A)] Sebastian Jakobi[(A)]

[(A)]Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
{holzer,jakobi}@informatik.uni-giessen.de

Abstract

We continue our research on the descriptional complexity of chop operations. Informally, the chop of two words is like their concatenation with the touching letters merged if they are equal, otherwise their chop is undefined. The iterated variants chop-star and chop-plus are defined similar as the classical operations Kleene star and plus. We investigate the state complexity of chop operations on unary languages and finite languages, and obtain similar bounds as for the classical operations.

1. Introduction

An interesting field of descriptional complexity of formal languages is the state complexity of regular languages. Given a regular language L, its state complexity is the minimum number of states that are sufficient and necessary for a finite automaton to accept L. This can be adopted to operations on languages. Given a (regularity preserving) k-nary operation $\circ$ and regular languages $L_1, L_2, \ldots, L_k$, the state complexity of $\circ$ is the minimum number of states that are sufficient and necessary for a finite automaton to accept $\circ(L_1, L_2, \ldots, L_k)$, as a function depending on the state complexities of the input languages. First results on the state complexity of operations on regular languages were obtained about more than three decades ago in [10] and [11]. Later in [13], besides some other operations, the state complexity of concatenation and Kleene star, which are basic operations for describing regular languages, was studied. Also the special case of unary input languages was investigated there, for which significantly different bounds than in the general case were obtained. Similarly, research on these operation problems on finite languages was done in [2]. All these results concentrated on deterministic finite automata, but one can study the same problems on nondeterministic finite automata. Research on the nondeterministic state complexity of concatenation, Kleene star and Kleene plus was done in [7], where it turned out that, unlike in the deterministic case, the bounds for general regular input languages and those for unary or finite input languages do not differ much.

This is an extended abstract of: M. Holzer, S. Jakobi. *State Complexity of Chop Operations on Unary and Finite Languages.* In M. Kutrib, N. Moreira, R. Reis (eds.): Proceedings of the 14th International Workshop on Descriptional Complexity of Formal Systems, number 7386 in LNCS, pages 169–182, Braga, Portugal, July 2012. Springer.

Recently in [1], the chop operation and its iterated variant were introduced as alternatives for concatenation and Kleene star. Substituting these operations, so called chop expressions, which are defined similarly to regular expressions, can be used to describe exactly the family of (λ-free) regular languages—here λ is the empty word. Various other operations that are more or less closely related to the herein studied chop operations can be found in, e.g., [3, 4, 6, 9, 12]. Descriptional complexity of chop expressions and chop operations on regular expressions and finite automata, as introduced in [1], was studied in [5]. There, tight bounds of $m+n$, $n+1$, and $n+2$ for the nondeterministic state complexity of, respectively, chop, chop-plus, and chop-star were obtained. This should be compared to the results for the classical operations concatenation, Kleene plus, and Kleene star, which yield the tight bounds $m+n$, n, and $n+1$, respectively, on the number of states. When considering the deterministic state complexity instead, the bounds for (iterated) chop differ from those for (iterated) concatenation, since the size of the alphabet appears as a parameter.

Following the results for (iterated) concatenation on unary and/or finite languages, we investigate the corresponding operation problems for chop, chop-star, and chop-plus on deterministic and nondeterministic finite automata accepting unary and/or finite languages. The situation will be very similar to concatenation, which at first may seem to be expected—but in the light of [5], where it was shown that chop expressions can be exponentially more succinct than regular expressions, this is also quite surprising.

2. Definitions

We investigate the descriptional complexity of the chop operation, which was recently introduced in [1], and its iterated variants. The *chop* or *fusion* of two words u and w in Σ^* is defined as

$$u \odot v = \begin{cases} u'av' & \text{if } u = u'a \text{ and } v = av', \text{ with } u', v' \in \Sigma^* \text{ and } a \in \Sigma \\ \text{undefined} & \text{otherwise,} \end{cases}$$

which is extended to languages as $L_1 \odot L_2 = \{\, u \odot v \mid u \in L_1 \text{ and } v \in L_2 \,\}$. Note that in the case of a unary alphabet, the chop of two non-empty words is always defined, in particular, $a^m \odot a^n = a^{m+n-1}$, for all $m, n \geq 1$. For the chop iteration, we define $L^{\otimes_0} = \Sigma$ and $L^{\otimes_i} = L \odot L^{\otimes_{i-1}}$, for $i \geq 1$, and the *iterated chop* or *chop-star* of a language L is defined as $L^{\otimes} = \bigcup_{i \geq 0} L^{\otimes_i}$. Moreover the *chop-plus* is denoted by $L^{\oplus} = \bigcup_{i \geq 1} L^{\otimes_i}$. It is easy to see that the chop operation $\odot$ is associative, and by definition the set Σ acts as the neutral element on all languages L from Σ^+. This is compatible with the definition of chop-star, because $\emptyset^{\otimes} = \Sigma$. In general, an application of the chop operation with Σ will cancel λ from the language L. Therefore, we have $\Sigma \odot L = L \odot \Sigma = L \setminus \{\lambda\}$, for every $L \subseteq \Sigma^*$.

For the notions of *deterministic* (DFA) and *nondeterministic finite automata* (NFA) and their *accepted languages*, we refer standard literature, e.g. to [8]. We only like to stress out here, that we assume any deterministic automaton to be complete, that is, its transition function is total. Thus, when counting states of automata, a potential sink state is counted for DFAs, but not for NFAs.

Nondeterministic state complexity				
	Language family			
	Regular	Unary	Finite unary	Finite
$\cdot$	$m+n$	$m+n-1 \leq \cdot \leq m+n$	$m+n-1$	$m+n-1$
$\odot$	$m+n$	$m+n$	$m+n-2$	$m+n-2$
$*$	$n+1$	$n+1$	$n-1$	$n-1$
$\otimes$	$n+2$	$n+2$	$n-1$	n
$+$	n	n	n	n
$\oplus$	$n+1$	$n+1$	n	n

Table 1: Nondeterministic state complexities of chop $\odot$, chop-star $\otimes$, and chop-plus $\oplus$, compared to their classical counterparts concatenation $\cdot$, Kleene star $*$, and Kleene plus $+$ on different language families. Again, m and n are the number of states of the input automata. Concerning the nondeterministic state complexity of the concatenation of (infinite) unary languages, an $m+n$ upper bound an $m+n-1$ lower bound was shown in [7]. Whether the upper bound can be lowered or not is still open.

3. Nondeterministic State Complexity

In this section we investigate the nondeterministic state complexity of chop operations applied to unary and/or finite languages. For the convenience of the reader we restate the constructions from [5] of nondeterministic finite automata for the operations under consideration on arbitrary regular languages.

Theorem 3.1 *Let $A_i = (Q_i, \Sigma, \delta_i, s_i, F_i)$, for $i = 1,2$, be two nondeterministic finite automata with $|Q_1| = m$ and $|Q_2| = n$. Then the following holds:*

1. *Let $A^{\odot} = (Q_1 \cup Q_2, \Sigma, \delta, s_1, F_2)$ such that for all states $p,q \in Q_1 \cup Q_2$ and $a \in \Sigma$ we have $p \in \delta(q,a)$ if either $p,q \in Q_i$ and $p \in \delta_i(q,a)$, for $i \in \{1,2\}$, or if $q \in Q_1$ and $p \in Q_2$, such that $\delta_1(q,a) \cap F_1 \neq \emptyset$ and $p \in \delta_2(s_2,a)$. Then $L(A^{\odot}) = L(A_1) \odot L(A_2)$.*
2. *Let $A^{\oplus} = (Q_1 \cup \{s\}, \Sigma, \delta^{\oplus}, s, F_1)$ such that for all $a \in \Sigma$ we have $\delta^{\oplus}(s,a) = \delta_1(s_1,a)$, and for all states $p,q \in Q_1$ we have $p \in \delta^{\oplus}(q,a)$ if $p \in \delta_1(q,a)$, or if $\delta_1(q,a) \cap F_1 \neq \emptyset$ and $p \in \delta_1(s_1,a)$. Then $L(A^{\oplus}) = L(A_1)^{\oplus}$.*
3. *Let $A^{\otimes} = (Q_1 \cup \{s,f\}, \Sigma, \delta^{\otimes}, s, F_1 \cup \{f\})$ such that for all $a \in \Sigma$ we have $f \in \delta^{\otimes}(s,a)$ if $\delta_1(s_1,a) \cap F_1 = \emptyset$, and further, for all states $p \in Q_1$ and $q \in Q_1 \cup \{s\}$ we have $p \in \delta^{\otimes}(q,a)$ if $p \in \delta^{\oplus}(q,a)$. Then $L(A^{\otimes}) = L(A_1)^{\otimes}$.*

Our results on the nondeterministic state complexity of chop operations on unary and/or finite languages are summarized in Table 1. One can see from the table that the upper bounds, which are implied by the constructions from Theorem 3.1, are tight already for a unary alphabet. Exemplarily we present our result on the chop of two languages.

Theorem 3.2 *Let A be an m-state and B be an n-state nondeterministic finite automaton for any integers $m,n \geq 1$. Then $f(m,n)$ states are sufficient and necessary in the worst case for any nondeterministic finite automaton to accept the language $L(A) \odot L(B)$, where $f(m,n) = m+n$ if $L(A)$ and $L(B)$ are unary, and $f(m,n) = m+n-2$ if $L(A)$ and $L(B)$ are finite, or finite and unary.*

Proof. Let $L_i = L(A_i)$ for some NFAs $A_i = (Q_i, \Sigma, \delta_i, s_i, F_i)$, for $i = 1,2$, and $A^{\odot}$ be the NFA constructed as described in Theorem 3.1 such that $L(A^{\odot}) = L(A_1) \odot L(A_2)$. We begin with the unary case. The upper bound is trivial and for the lower bound consider the languages $(a^m)^*$ and $(a^n)^*$, that can be accepted by minimal m-state and n-state NFAs, respectively. Since a^{m+n-1} is the shortest word in the language $L = L_1 \odot L_2$, any NFA accepting L needs at least $m+n$ states.

We now turn to finite languages. Here the state graphs of A_1 and A_2 are acyclic, from which immediately follows that s_2 is not reachable in $A^{\odot}$. Further, since we may assume A_1 to be minimal and L_1 not to be empty, there must be some state $f \in F_1$, such that $\delta_1(f,a) = \emptyset$ for all $a \in \Sigma$. This state is not useful in $A^{\odot}$ and may be eliminated. So $A^{\odot}$ needs at most $m+n-2$ states. The lower bound for finite unary languages is easily verified with the languages $\{a^{m-1}\}$ and $\{a^{n-1}\}$. □

4. Deterministic State Complexity

Now we come to the deterministic state complexity of chop operations. For unary languages, there obviously cannot be any dependency on the size of the alphabet, and in fact the corresponding bounds and proofs are almost the same as in [13] for concatenation. But also for finite languages, where the alphabet size could play an important role, there is hardly a difference between the bounds for chop operations and the bounds for the corresponding classical operations based on concatenation, as studied in [2]. This contrasts the situation for general regular languages, where the size of the alphabet matters for chop operations [5], but is mostly irrelevant for concatenation and related operations. Our findings are summarized in Table 2, again compared to the results for the classical concatenation operations.

The presented bounds for chop-star and chop-plus of (non-unary) finite languages hold for languages accept by DFAs that have at least two accepting states. If the input language is accepted by a DFA with a single accepting state, then we get the following result.

Theorem 4.1 *Let A be an n-state deterministic finite automaton with one final state and $n \geq 4$, such that $L(A) \subseteq \Sigma^*$ is finite. Then $n-2+\min(|\Sigma|, n-2)$ states are sufficient and necessary in the worst case for any deterministic finite automaton to accept $L(A)^{\otimes}$. A similar statement also holds for $L(A)^{\oplus}$, where the corresponding bound is $n-1+\min(|\Sigma|, n-2)$.*

Proof. We first consider the chop-star operation. Let $A = (Q, \Sigma, \delta, q_0, \{q_f\})$ be a DFA accepting a finite language. Then there is a non-accepting sink state $q_s \in Q$ such that $\delta(q_s, a) = \delta(q_f, a) = q_s$, for all $a \in \Sigma$. Further there must be some state $q_1 \in Q \setminus \{q_f, q_s\}$ that is only reachable with words of length one. For constructing the NFA $A^{\otimes}$, there is no need of adding new states s and f, since these can be identified with q_0 and q_f: the initial state q_0 has no ingoing transitions and q_f only leads to the sink state. So we have an

Deterministic state complexity				
	Language family			
	Regular	Unary	Finite unary	Finite
$\cdot$	$m \cdot 2^n - t \cdot 2^{n-1}$	$m \cdot n$	$m+n-2$	$(m-n+3)2^{n-2}-1$
$\odot$	$m \cdot 2^n - t \cdot 2^{n-\min(k,n)}+1$	$m \cdot n+1$	$m+n-3$	$(m-n+2)2^{n-2}-1$
$*$	$2^{n-1}+2^{n-2}$	$(n-1)^2+1$	$n^2-7n+13$	$2^{n-3}+2^{n-4}$
$\otimes$	$2^n-1+\min(k,n)$	$(n-1)^2+2$	$n^2-9n+22$	$2^{n-3}+2^{n-3-t}+2$
$+$	$2^{n-1}+2^{n-2}+1$	$(n-1)^2+1$	$n^2-7n+13$	$2^{n-3}+2^{n-4}+1$
$\oplus$	2^n	$(n-1)^2+2$	$n^2-9n+22$	$2^{n-3}+2^{n-3-t}+1$

Table 2: Deterministic state complexities of chop $\odot$, chop-star $\otimes$, and chop-plus $\oplus$, compared to their classical counterparts concatenation $\cdot$, Kleene star $*$, and Kleene plus $+$ on different language families. Here m and n are the number of states of the input automata (where m corresponds to the "left" automaton in the case of $\cdot$ and $\odot$), k is the alphabet size, and t is the number of accepting states (of the "left" automaton).

NFA $A^{\otimes} = (Q,\Sigma,\delta^{\otimes},q_0,\{q_f\})$ such that for all $p,q \in Q$ and $a \in \Sigma$ we have $p \in \delta^{\otimes}(q,a)$ if $p = \delta(q,a)$, and additionally, $q_f \in \delta^{\otimes}(q_0,a)$ if $\delta(q_0,a) \neq q_f$, and $\delta(q_0,a) \in \delta^{\otimes}(q,a)$ if $\delta(q,a) = q_f$.

In the corresponding powerset automaton, only singleton sets and sets of the form $\{q,q_f\}$ for some $q \in Q \setminus \{q_f\}$ are reachable—note that all states $P \neq \{n-1\}$ are equivalent to $P \setminus \{n-1\}$. In particular, states $\{q,q_f\}$ are actually of the form $\{\delta(q_0,a),q_f\}$ for some $a \in \Sigma$. For the singleton sets note that $\{q_1\}$ is not reachable, since states containing q_1 must also contain q_f. Further, the number of states P with $q_f \in P$ is bounded by $\min(|\Sigma|,n-2)$, which can be seen as follows. If $\{q_f\}$ is reachable, then there must be some $a \in \Sigma$ with $\delta(q_0,a) = q_f$. Then there are at most $|\Sigma \setminus \{a\}|$ other states of the form $\{q,q_f\}$. Otherwise, if $\{q_f\}$ is not reachable, there are at most $|\Sigma|$ many states $\{q,q_f\}$. But even for large alphabets we cannot reach more than $n-2$ states of size two: the set $\{q_0,q_f\}$ is not reachable, and $\{q_s,q_f\}$ is equivalent to $\{q_s\}$. So we get the stated upper bound of $n-2+\min(|\Sigma|,n-2)$ states.

It suffices to prove this bound to be optimal for $\Sigma = \{a_1,a_2,\ldots,a_k\}$ with $2 \leq k \leq n-2$. Let $A = (Q,\Sigma,\delta,0,\{n-2\})$ for $Q = \{0,1,\ldots,n-1\}$, and

$$\delta(0,a_i) = \begin{cases} i & \text{for } i < k, \\ n-2 & \text{for } i = k, \end{cases} \qquad \delta(n-3,a_i) = \begin{cases} n-2 & \text{for } i < k, \\ n-1 & \text{for } i = k, \end{cases}$$

$$\delta(q,a_i) = q+1 \text{ for } 1 \leq q \leq n-4, \qquad \delta(n-2,a_i) = \delta(n-1,a_i) = n-1.$$

The construction of the NFA $A^{\otimes}$ as described above gives rise to the additional transitions $(0,a_i,n-2)$ and $(n-3,a_i,i)$ for $1 \leq i \leq k-1$. In the powerset automaton, the singletons $\{q\}$, for $2 \leq q \leq n-3$, are reachable from $\{0\}$ by reading the word $a_1a_k^{q-1}$, the state $\{n-2\}$ by reading a_k, and the sink state $\{n-1\}$ by reading $a_1a_k^{n-3}$. Further, the sets $\{p,n-2\}$, for $1 \leq p \leq k-1$, are reachable from $\{n-3\}$ by reading a_p, and no other states can be reached.

The inequivalence of distinct states P and P' is easy to see: it suffices to proof that singletons $\{p\}$ and $\{q\}$ with $0 \leq p < q \leq n-2$ are inequivalent, because this generalizes to the states $\{q, n-2\}$, and obviously $\{n-1\}$ cannot be equivalent to any other state. So let $p < q$. The states $\{0\}$ and $\{n-3\}$ are distinguished by a_k and if $p \neq 0$ or $q \neq n-3$, then the word a_1^{n-2-q} is accepted from $\{q\}$ but not from $\{p\}$.

The bounds for chop-plus can be proven with nearly the same argumentation as above. The only difference is that no additional transitions get defined for the initial state, and so $\{q_1\}$ and, respectively, $\{1\}$ are reachable in the resulting automata, so we need one more state. □

References

[1] S. A. BABU, P. K. PANDYA, Chop Expressions and Discrete Duration Calculus. In: D. D'SOUZA, P. SHANKAR (eds.), *Modern Applications of Automata Theory*. IISc research Monographs Series 2, World Scientific, 2010.

[2] C. CÂMPEANU, K. C. II, K. SALOMAA, S. YU, State complexity of basic operations on finite languages. In: O. BOLDT, H. JÜRGENSEN (eds.), *Proceedings of the 4th International Workshop on Implementing Automata*. Number 2214 in LNCS, Springer, Potsdam, Germany, 1999, 60–70.

[3] A. CĂRĂUŞU, G. PĂUN, String intersection and short concatenation. *Revue Roumaine de Mathématiques Pures et Appliquées* **26** (1981), 713–726.

[4] M. DOMARATZKI, Minimality in Template-Guided Recombination. *Information and Computation* **207** (2009), 1209–1220.

[5] M. HOLZER, S. JAKOBI, Chop Operations and Expressions: Descriptional Complexity Considerations. In: G. MAURI, A. LEPORATI (eds.), *Proceedings of the 15th International Conference Developments in Language Theory*. Number 6795 in LNCS, Springer, Milan, Italy, 2011, 264–275.

[6] M. HOLZER, S. JAKOBI, M. KUTRIB, The Chop of Languages. In: P. DÖMÖSI, S. IVÁN (eds.), *Proceedings of the 13th International Conference Automata and Formal Languages*. Debrecen, Hungary, 2011, 197–210.

[7] M. HOLZER, M. KUTRIB, Nondeterministic Descriptional Complexity of Regular Languages. *International Journal of Foundations of Computer Science* **14** (2003) 6, 1087–1102.

[8] J. E. HOPCROFT, J. D. ULLMAN, *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[9] M. ITO, G. LISCHKE, Generalized periodicity and primitivity. *Mathematical Logic Quarterly* **53** (2007), 91–106.

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

[11] A. N. MASLOV, Estimates of the Number of States of Finite Automata. *Soviet Mathematics Doklady* **11** (1970), 1373–1375.

[12] A. MATEESCU, A. SALOMAA, Parallel composition of words with re-entrant symbols. *Analele Universităţii Bucureşti Matematică-Informatică* **45** (1996), 71–80.

[13] S. YU, Q. ZHUANG, K. SALOMAA, The state complexity of some basic operations on regular languages. *Theoretical Computer Science* **125** (1994), 315–328.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 71 – 76

How to Measure Word Order Freedom for Natural Languages?

Vladislav Kuboň Markéta Lopatková$^{(A)}$ Martin Plátek$^{(A)}$

Charles University in Prague, Faculty of Mathematics and Physics
{vk,lopatkova}@ufal.mff.cuni.cz, martin.platek@.mff.cuni.cz

1. Introduction

In this paper we would like to clarify some basic features and notions which may play a key role in the investigations of the word order freedom. For this purpose we are going to exploit the elementary method of *analysis by reduction* (AR) and the formal data type derived from this method, so-called D-trees. A complete description of both the method and the data type can be found for example in ([7]). Let us remind that the analysis by reduction has served as a motivation for a family of so called restarting automata, see ([6]). The first step in the direction of more formal treatment of the word order freedom has been done in ([2]), where the authors discussed it without the exploitation of the analysis by reduction and without setting the constraints on unchanged morphological and syntactic properties of individual words. We will focus on some examples cited there and modify them according to the methods mentioned in ([7]). We also exploit sample sentences from the Prague Dependency Treebank (PDT),[1] a large-scale treebank of Czech ([1]).

2. The Background of our Experiments

We would like to introduce the notion of a shift operation in the course of the AR, a key notion for the investigation of a measure of word order freedom. In order to be able to define the shift operation, it is useful to introduce the data structure we are exploiting.

The D-tree (Delete or Dependency trees), see e.g. [7], is a rooted ordered tree with edges oriented from its leaves to its root. Nodes of each tree correspond to individual occurrences of word forms in a sentence. Moreover, we suppose a total ordering on the nodes that reflects word order in a sentence.

Let us remind that the concept of D-tree reflects the analysis by reduction (without rewriting) – its structure corresponds to a way how individual words of a sentence are deleted in the course of the corresponding steps of the analysis by reduction. (Informally, each edge of a D-tree connects a word form to some other word form if the latter cannot be deleted earlier then the first word form in (any branch of) analysis by reduction of the same sentence.)

$^{(A)}$The paper reports on the research within the NoSCoM project, supported by the grant of the Grant Agency of the Czech Republic No. P202/10/1333.

[1]http://ufal.mff.cuni.cz/pdt.html

Measures of Non-projectivity and Shift Operation

Non-projectivity. When considering word order freedom, we have to take into account one phenomenon which is common in languages with higher degree of word order freedom, namely non-projective constructions (for previous usage of this term see esp. [5, 4]). In order to classify this phenomenon, it is useful to define certain notions allowing for an easy definition of projectivity/non-projectivity and also for the introduction of measures of non-projectivity (these notions are formally defined in [2]).

The *coverage of a node u of a* D*-tree* identifies nodes from which there is a path to u in the D-tree (including empty path). It is expressed as a set of horizontal positions/indices (expressing total ordering on nodes in a D-tree, see above) of nodes directly or indirectly dependent upon a particular node.

The notion of a coverage leads directly to a notion of *a hole in a subtree*. Such a hole exists if the set of indices in the coverage is not a continuous sequence.

We say that D-tree T is *projective* if none of its subtrees contains a hole; otherwise, T is *non-projective*.

Shift operation. In order to be able to describe necessary word order shifts in the course of AR, we need to define a notion of equivalence for D-trees. Such equivalence (denoted as DP-equivalence) is defined as follows: DP*-equivalent trees* are those D-trees which have (i) the 'same' sets of nodes, i.e., the nodes describing the same set of lexical bundles, and (ii) their edges always connect 'identical' pairs of nodes (nodes with identical lexical bundles). It actually means that a particular set of DP-equivalent trees contains the D-trees representing sentences created by a permutation of the words of the original sentence but having the same dependency relations.

Let T be a D-tree; the set of D-trees which are DP-equivalent to T will be denoted DPE(T). In other words, DPE(T) is a set of D-trees which differ only in the word order of their characteristic sentence.

The previous concepts allow us to introduce a new feature, a *number of reduction steps enforcing a shift* in a single branch of AR. Shifts make it possible to change word order and thus 'recover' from incorrect word order that may be incurred by an AR deleting step. The *shift operation* is such a change in a D-tree when (i) the ordering of all nodes except for one is preserved, and (ii) the edges are preserved (connecting 'identical' pairs of nodes with respect to described lexical bundles). It means that both the original D-tree T and the modified one belong to the same set DPE(T).

Let T be a D-tree, $T \notin$ CT. Our goal is to find – if possible – a modified D-tree T' such that T' is a correct surface tree (i.e., $T' \in$ CT) and T' is DP-equivalent to T (i.e., $T' \in$ DPE(T)) by applying as small number of shift operations as possible.

3. Towards a Measure of Word Order Freedom

3.1. Data

The investigation focuses upon an interplay of two phenomena related to word order: the *non-projectivity* of a sentence and the *number of word order shifts* within the analysis by reduction. This interplay is exemplified on a set of 'suspicious' types of sentences identified in previous work on Czech word order freedom [2]. The sample set was enriched with sentences from the Prague Dependency Treebank (PDT), a large-scale treebank of Czech [1], namely the sentences with a non-projectivity given by a modal verb (typically in combination with clitics [3]). These sentences were manually annotated using the method of analysis by reduction.

3.2. Principles of Data Analysis

The following principles are applied during the analysis of sample data.

Principle 1: *'Preprocessing' – we simplify the input sentences using* AR *in such a way that only words related to these phenomena are preserved.*
In other words, we focus on those branches of AR where the words which do not contribute to the examined structures are already processed and thus deleted (if it is possible without shifting). Let us exemplify this on sentence (3) (shortened sentence from PDT) and its initial simplification:

Example:
(2) *Naše firma by se možná mohla tvářit, že se jí premiérova slova netýkají (. . .).*
'Perhaps our firm might pretend that the prime minister's words do not apply to it (. . .).'
$\rightarrow$ *Firma by se mohla tvářit.*
'The firm might pretend.'

Principle 2: *Minimality – we focus especially on those branches of* AR *that allows us to reduce a sentence with minimal number of shifts.*
Typically, there are several possibilities how to analyze a simplified sentence. In our example (2), we can start with reducing the noun *firma* 'firm'. This results in the string starting with clitics *by* and *se* – thus a shift in word order positions must by applied to ensure the correctness of the simplified sentence. We have two possibilities of shifting:
(a) We can SHIFT the verb *tvářit* 'to pretend' to the first position, which results in the correct sentence *Tvářit by se mohla.* However, the only possible subsequent reduction step means deleting the pair *tvářit se* 'to pretend + REFL', which requires another SHIFT *By mohla.* $\rightarrow_{SHIFT}$ *Mohla by.*
Or, (b) we can SHIFT the verb *mohla* 'may' to the first position *Mohla by se tvářit.* The subsequent reduction of the pair *se tvářit* 'REFL + to pretend' does not require another shifting.

This example shows that if we aim at the minimal necessary number of shifts then we must apply the second type of shifting.

Principle 3. *Restriction on the shift operation – the application of the shift operation is limited to cases where it is enforced by the correctness preserving principle of* AR (i.e., to cases where a simple deletion would violate the principle of correctness imposed on AR).

Principle 4: *Non-projectivity – we allow for non-projective reductions.*
Long distance dependencies are allowed, i.e., depending word in a distant (non-projective) position may be deleted.

Example:
(3) *Marii se Petr tu knihu rozhodl nekoupit.*
'to-Mary – REFL – Peter – that/the book – decided – not-to-buy'
'To Mary, Peter decided not to buy the book.'

The word *Marii* (indirect object of the verb *nekoupit* 'not-to-buy') is reduced even though it is 'separated' from its governing verb by the main predicate *rozhodl* 'decided' (i.e., by the root of the dependency tree); the relation *Marii – nekoupit* 'to-Mary – not-to buy' creates a non-projective edge, [2].

Principle 5: *Locality – we limit our observations to simple sentences/clauses containing interesting phenomena.*
This principle allows us to focus on an interplay of several phenomena affecting a single surface syntactic construction (based on principle 1, all 'uninteresting' words are already processed, coordination is simplified etc.); in case of more than one interesting construction in a sentence (prototypically a complex sentence consisting of several clauses), they are processed separately. The reason is simple – if we want to achieve results reflecting the properties of the investigated phenomenon, we have to eliminate chances to construct a complex sentence with an arbitrary number of shifts simply by coordinating a desired number of clauses requiring one shift each (or by inserting a relative clause with a shift).

3.3. How to Measure Word Order Freedom?

The previous work led to the proposal of a measure based on (minimal) number of shifts within an analysis by reduction of a given sentence, see esp. [3]. We can characterize this approach by principles 1-5 mentioned above, i.e., as an analysis by reduction enhanced with the possibility of word order changes. The results proved that the number of shifts is an important factor providing different information than already existing measures reflecting the complexity of word order of individual sentences.

However, the granularity of the proposed measure seemed to be too low as all the inspected sentences from PDT were analyzed with at most one shift operation, regardless the number of holes and number of clitics in a sentence. This result was improved when we subsequently inspected 'suspicious' sentences analyzed in [2]. We have found a construction where at least two shifts are necessary (even when the principle of non-projectivity is applied, i.e. we allow for non-projective reductions).

Example:
(4) *S těžkým se mu bála pomoci úkolem.*
'with – difficult – REFL – him – (she) was afraid – to help – task'
'With a *difficult* task, she was afraid to help him.'
This sentence is rather special Czech surface construction when – due to the stress on the adjectival attribute *těžkým* 'difficult' – the prepositional group *s těžkým úkolem* 'with difficult task' is split and the preposition and adjective are moved to the beginning of the sentence.

The only possible correctness preserving reduction lies in deleting the pronoun *mu* 'him'. With respect to the dependency relations in the sentence, the subsequent reduction step must delete the adjective, but this step results in an ill-formed sentence:
$\rightarrow_{DEL}$ ** S se bála pomoci úkolem.*
Thus a word order correction is enforced:
(a) We can SHIFT the noun *úkolem* 'task' to obtain the (correct) continuous noun group *s úkolem* 'with (the) task'. The reduction of this noun group is then the only reduction possibility, again resulting in the sentence with an incorrect word order. Now, the 'optimal' SHIFT of the main predicate is enforced; the final deletion results in a correct simplified sentence:
$\rightarrow_{SHIFT}$ *S úkolem se bála pomoci.* $\rightarrow_{DEL}$ ** Se bála pomoci.* $\rightarrow_{SHIFT}$ *Bála se pomoci.* $\rightarrow_{DEL}$ *Bála se.*

(b) Alternatively, the preposition *s* ''with' is shifted to create continuous noun group *s úkolem* 'with (the) task', followed by the 'optimal' shift of the main predicate; then the sentence can be reduced by applying simple delete operations:
$\rightarrow_{SHIFT}$ ** Se bála pomoci s úkolem.* $\rightarrow_{SHIFT}$ *Bála se pomoci s úkolem.* $\rightarrow_{DEL} \ldots \rightarrow$ *Bála se.*

In both branches of AR, (at least) two shift operations are necessary to analyze sentence (4) (contrary to the hypothesis made on the basis of corpus data [3]).

This observation, however, does not refine the measure itself, it only increases the range of its values for Czech.

3.4. A proposed refinement of the original measure

It is quite obvious that applying stricter constraints on the delete or shift operations would bring a more refined measure. There are actually at least two possible ways – (i) we can distinguish a type of necessary shifts (e.g., a shift of a verb / a shift across a verb), (ii) the deletion can be limited to adjacent word forms, or (iii) the deletion can be limited to projective reductions (i.e., dependent and governing words may be 'separated' only by word forms (transitively) dependent on the latter one, contrary to Principle 4). So far, we have focused on the third restriction.

Example:
(5) *Pomocí může být systém ECM.*
'help – can – to be – system – ECM'
'The ECM system may be a help.'
The first two steps are easy, we will get rid of a subject (the ECM system) by a stepwise deletion:
$\rightarrow$ *Pomocí může být.*

The remaining three words constitute a non-projective 'core' of the original sentence providing the following options typical for such a case:
(a) We can make the sentence PROJECTIVE by shifting the **dependent** word $\rightarrow$ *Může pomocí být.*
(b) We can also make it PROJECTIVE by shifting the **governing** word $\rightarrow$ *Pomocí být může.*
(c) The projectivization mentioned in (a) and (b) can also be achieved by means of a shift of the main verb, in (a) it would represent a shift of the main verb *Může* to the first position in the sentence, in b) to the last one. This option actually only increases the number of possibilities without bringing anything really new. Even worse, shifting the main verb of the sentence may bring additional complications in case that the non-projective core of the input sentence is bigger and more complex than in our example. Therefore it is better to avoid this type of a shift entirely and to concentrate on the shifts under options (a) and (b).

Let us now look at a more complicated example with a clitic combined with a non-projectivity.
Example:
(6) *Tu knihu se rozhodl věnovat nadaci.*
'This – book – REFL – decided – donate – to a foundation'
'(He) decided to donate this book to a foundation.'
The first two deletions are obvious, the words *tu* 'this' and *nadaci* 'foundation' can be reduced in an arbitrary order: $\rightarrow$ *Knihu se rozhodl věnovat.*
Let us now perform the two variants of further reduction according to the options mentioned above:
(a) Let us make the sentence PROJECTIVE by means of shifting the **dependent** word *knihu*
$\rightarrow_{SHIFT}$ ** Se rozhodl knihu věnovat.*
This shift results in an ungrammatical sentence, therefore it is necessary to perform a shift operation again, this time by shifting the predicate of the sentence to the sentence first position, thus eliminating the ungrammaticality caused by the clitic in the first position.
$\rightarrow_{SHIFT}$ *Rozhodl se knihu věnovat.*
The remaining reductions are then obvious:
$\rightarrow_{DEL}$ *Rozhodl se věnovat.* $\rightarrow_{DEL}$ *Rozhodl se.*

(b) In a similar way we can make the sentence PROJECTIVE by means of shifting the **governing** word *věnovat* $\rightarrow_{SHIFT}$ *Knihu věnovat se rozhodl.* This shift results in a sentence which looks like syntactically incorrect one due to the clitic becoming a third word in a sentence, not a second one. However, in

this case, the group *věnovat knihu* may be understood as a single unit and thus the clitic still occupies the sentence second position and we may proceed with a simple reduction:
$\rightarrow_{DEL}$ *Věnovat se rozhodl.* $\rightarrow_{DEL}$ ** Se rozhodl.* This reduction is the only possible and the ungrammaticality of the resulting sentence has to be corrected by a second shift:
$\rightarrow_{SHIFT}$ *Rozhodl se.*

So, again, regardless of the option used, we are arriving at a score of 2 shifts. This actually indicates that the refined measure captures the interplay of clitics and non-projectivities in a more subtle way than the original measure.

4. Conclusion and Future Work

In this paper we have presented the results of a detailed investigation of the phenomenon of word order freedom for a particular natural language, Czech. We have shown, on the basis of several examples, that if we leave the safe grounds of data contained in a syntactically annotated corpus of Czech, we may found sentences exemplifying the complexity of the problem. We have shown that the range of values of the original measure of word order freedom presented in previous papers may be bigger in certain cases, we have also discussed the method how to obtain an exact value for this measure, and, last but not least, we have suggested a refinement of the original measure wchich more adequately captures the interplay of various phenomena.

In the future we would like to continue the research in this direction by examining more linguistic phenomena, by testing the measure on other languages with various degree of word order freedom and by experimenting with a different or modified set of constraints applied on the shift operation.

References

[1] J. HAJIČ, J. PANEVOVÁ, E. HAJIČOVÁ, P. SGALL, P. PAJAS, J. ŠTĚPÁNEK, J. HAVELKA, M. MIKULOVÁ, Z. ŽABOKRTSKÝ, M. ŠEVČÍKOVÁ-RAZÍMOVÁ, *Prague Dependency Treebank 2.0*. Linguistic Data Consortium, Philadelphia, PA, USA, 2006.

[2] T. HOLAN, V. KUBOŇ, K. OLIVA, M. PLÁTEK, On Complexity of Word Order. *Les grammaires de dépendance – Traitement automatique des langues (TAL)* **41** (2000) 1, 273–300.

[3] V. KUBOŇ, M. LOPATKOVÁ, M. PLÁTEK, Studying Formal Properties of a Free Word Order Language. In: G. YOUNGBLOOD, P. MCCARTHY (eds.), *Proceedings of the Twenty-Fifth International Florida Artificial Intelligence Research Society Conference*. AAAI Press, Palo Alto, California, 2012, 300–3005.

[4] J. KUNZE, *Die Auslassbarkeit von Satzteilen bei koordinativen Verbindungen im Deutschen*. Number 14 in Schriften zur Phonetik, Sprachwissenschaft und Kommunikationsforschung, Akademie-Verlag, Berlin, 1972.

[5] S. MARCUS, Sur la notion de projectivité. *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik* **11** (1965) 1, 181–192.

[6] F. OTTO, Restarting Automata and Their Relation to the Chomsky Hierarchy. In: Z. ÉSIK, Z. FÜLÖP (eds.), *Proceedings of DLT 2003*. LNCS 2710, Springer-Verlag, Berlin, 2003, 55–74.

[7] M. PLÁTEK, F. MRÁZ, M. LOPATKOVÁ, (In)Dependencies in Functional Generative Description by Restarting Automata. In: H. BORDIHN, R. FREUND, T. HINZE, M. HOLZER, M. KUTRIB, F. OTTO (eds.), *Proceedings of NCMA 2010*. books@ocg.at 263, Österreichische Computer Gesellschaft, Wien, Austria, 2010, 155–170.

Isomorphisms of Linear Orders Given by Automata

Dietrich Kuske

Institut für Theoretische Informatik, Technische Universität Ilmenau, Germany

This talk considers the following decision problem:

input: two languages L_1 and L_2 over an alphabet Σ and a linear order $\leq$ on Σ^*
question: Does $(L_1, \leq) \cong (L_2, \leq)$ hold?

We ask this question for fixed linear orders like the lexicographic order and variants thereoff. Furthermore, the languages will be given by finite automata, by (deterministic) pushdown automata, and by finite tree automata.

References

[1] S.L. Bloom and Z. Ésik. The equational theory of regular words. *Information and Computation*, 197:55–89, 2005.

[2] Z. Ésik. An undecidable property of context-free linear orders. *Inform. Processing Letters*, 111(3):107–109, 2011.

[3] D. Kuske. Isomorphisms of scattered automatic linear orders. In *CSL'12*, 2012. Accepted.

[4] D. Kuske, J. Liu, and M. Lohrey. The isomorphism problem on classes of automatic structures. In *LICS 2010*, pages 160–169. IEEE Computer Society, 2010.

[5] D. Kuske, J. Liu, and M. Lohrey. The isomorphism problem on classes of automatic structures with transitive relations. *Transactions of the AMS*, 2011. Accepted.

[6] M. Lohrey and Ch. Mathissen. Isomorphism of regular trees and words. In *ICALP'11*, Lecture Notes in Comp. Science vol. 6756, pages 210–221. Springer, 2011.

[7] W. Thomas. On frontiers of regular trees. *RAIRO – Theoretical Informatics*, 20(4):371–381, 1986.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 79 – 84

States and Heads Do Count For Unary Multi-Head Finite Automata

Martin Kutrib Andreas Malcher Matthias Wendlandt

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

Abstract

Unary deterministic one-way multi-head finite automata characterize the unary regular languages. Here they are studied with respect to the existence of head and state hierarchies. It turns out that for any fixed number of states, there is an infinite proper head hierarchy. In particular, the head hierarchy for stateless deterministic one-way multi-head finite automata is obtained using unary languages. On the other hand, it is shown that for a fixed number of heads, $m + 1$ states are more powerful than m states. Finally, the open question of whether emptiness is undecidable for stateless one-way two-head finite automata is addressed and, as a partial answer, undecidability can be shown if at least four states are provided.

1. Introduction

Finite automata enhanced with multiple one-way reading heads, so-called *one-way multi-head finite automata*, can be considered as one of the oldest models for parallel computation. The idea behind is to have a common finite state control which processes in parallel several parts of the input that are read by different heads. First investigations on the computational capacity of such devices date back to [8, 9]. Since that time many extensions of the model including, for example, two-way head motion and nondeterministic behavior have been studied, and many results on the computational and descriptional complexity have been obtained. This documents the importance of such devices. A recent survey on these topics can be found in [3].

An important question raised already in [9] asks for the power of the heads, that is, whether additional heads can strengthen the computational capacity of multi-head finite automata. For one-way devices the question has been answered in the affirmative in [12], where the witness languages are defined over a ternary alphabet and are not bounded. A reduction of the size of the underlying alphabet has been obtained in [1, 7], where languages of the form a^*b^* are used.

Recently, *stateless* multi-head finite automata have been introduced as an interesting subclass with a biological motivation [11]. For stateless automata the finite state control is restricted to have one state only. Although this seems to be a strong restriction, stateless multi-head finite automata are still quite powerful. For example, it is shown in [11] that the emptiness problem is undecidable for deterministic stateless one-way three-head finite automata.

In [6] this undecidability result is extended to stateless deterministic two-way two-head finite automata. It is an open question whether emptiness remains undecidable for deterministic stateless one-way two-head finite automata. Also in [6], a proper head hierarchy for deterministic stateless one-way multi-head finite automata is shown by a suitable translation of the languages used in [12]. However, the languages used are not bounded and require a growing alphabet whose size depends on the number of heads.

An obvious generalization of the concept of stateless automata is to consider whether in some automata model $m+1$ states are more powerful than m states. Such *state hierarchies* exist, for example, for deterministic and nondeterministic finite automata and for deterministic pushdown automata [2]. On the other hand, for nondeterministic pushdown automata a state hierarchy does not exist, since every context-free language can be accepted by a stateless nondeterministic pushdown automaton (see [4]).

It is known that every unary language accepted by a one-way multi-head finite automaton is semilinear and hence regular [5, 10]. Thus, disregarding the number of states, one head will always suffice to accept these languages. But it turns out that this situation changes drastically if the number of states is fixed. In this paper, we study these deterministic unary one-way multi-head finite automata with respect to the number of heads and states. In Section 3., as a main result a double hierarchy concerning states and heads is established. On the one hand, we obtain for every number $m \geq 2$ of states that k' heads are more powerful than k heads, where $k' \geq k\left(1+\frac{1}{\log_2(m)}\right)$. On the other hand, we show for every number $k \geq 1$ of heads that $m+1$ states are more powerful than m states. Thus, we have state hierarchies for a fixed number of heads and head hierarchies for a fixed number of states. In Section 4., we prove a proper head hierarchy for deterministic stateless one-way multi-head finite automata using unary witness languages. This improves the result in [6] with respect to the alphabet size best possible. Finally, we address the open question of whether emptiness is undecidable for deterministic stateless one-way two-head finite automata and show that emptiness is undecidable for deterministic one-way two-head finite automata having four states.

2. Preliminaries and Definitions

Let $k \geq 1$ be a natural number. A one-way k-head finite automaton is a finite automaton having a single read-only input tape whose inscription is the input word in between two endmarkers (we provide two endmarkers in order to have a definition consistent with two-way finite automata). The k heads of the automaton can move to the right or stay on the current tape square but not beyond the endmarkers. Formally, a *deterministic one-way k-head finite automaton* (*1DFA*(k)) is a system $M = \langle S, A, k, \delta, \triangleright, \triangleleft, s_0 \rangle$, where S is the finite set of *internal states*, A is the finite set of *input symbols*, $k \geq 1$ is the *number of heads*, $\triangleright \notin A$ is the *left* and $\triangleleft \notin A$ is the *right endmarker*, $s_0 \in S$ is the *initial state*, $\delta : S \times (A \cup \{\triangleright, \triangleleft\})^k \to S \times \{0,1\}^k$ is the partial transition function, where 1 means to move the head one square to the right, and 0 means to keep the head on the current square. Whenever $(s', d_1 d_2 \cdots d_k) = \delta(s, a_1 a_2 \cdots a_k)$ is defined, then $d_i = 0$ if $a_i = \triangleleft$, for $1 \leq i \leq k$.

A 1DFA(k) starts with all of its heads on the left endmarker. Since we are going to limit the number of states of the automata, for convenience m-state 1DFA(k) are denoted by 1DFA$_m(k)$, for $k, m \geq 1$. The most restricted version are stateless 1DFA(k), that is, automata having exactly one state. Therefore, non-trivial acceptance cannot be defined by accepting states. Instead, we follow the definition in [6] and say that an input is accepted if and only if the computation ends in an infinite state loop in which the heads are necessarily stationary, since they are one-way. A $1\mathrm{DFA}_m(k)$ blocks and rejects when the transition function is not defined for the current situation.

A *configuration* of a 1DFA(k) $M = \langle S, A, k, \delta, \rhd, \lhd, s_0 \rangle$ at some time $t \geq 0$ is a triple $c_t = (w, s, p)$, where $w \in A^*$ is the input, $s \in S$ is the current state, and $p = (p_1, p_2, \ldots, p_k) \in \{0, 1, \ldots, |w|+1\}^k$ gives the current head positions. If a position p_i is 0, then head i is scanning the symbol $\rhd$, if it satisfies $1 \leq p_i \leq |w|$, then the head is scanning the p_ith letter of w, and if it is $|w|+1$, then the head is scanning the symbol $\lhd$. The *initial configuration* for input w is set to $(w, s_0, (0, \ldots, 0))$. During its course of computation, M runs through a sequence of configurations. One step from a configuration to its successor configuration is denoted by $\vdash$. Let $w = a_1 a_2 \ldots a_n$ be the input, $a_0 = \rhd$, and $a_{n+1} = \lhd$, then we set $(w, s, (p_1, p_2, \ldots, p_k)) \vdash (w, s', (p_1 + d_1, p_2 + d_2, \ldots, p_k + d_k))$ if and only if $(s', d_1 d_2 \cdots d_k) = \delta(s, a_{p_1} a_{p_2} \cdots a_{p_k})$. As usual we define the reflexive, transitive closure of $\vdash$ by $\vdash^*$, and its transitive closure by $\vdash^+$. Note, that due to the restriction of the transition function, the heads cannot move beyond the right endmarker. Whenever we consider an accepting computation it is understood that we mean the finite initial part of the computation up to but not including the first state loop at the end. The language accepted by a 1DFA(k) M is

$$L(M) = \{\, w \in A^* \mid \text{there are } s \in S, 0 \leq p_i \leq |w|+1, 1 \leq i \leq k \text{ such that}$$
$$(w, s_0, (0, \ldots, 0)) \vdash^* (w, s, (p_1, p_2, \ldots, p_k)) \vdash^+ (w, s, (p_1, p_2, \ldots, p_k)) \,\}.$$

The family of all languages accepted by a device of some type X is denoted by $\mathscr{L}(X)$.

Example 2.1 *For each $k, m \geq 2$, the unary singleton language $L_{k,m} = \{\, a^{(k-1)m^k} \}$ is accepted by some 1DFA$_m(k)$.*

3. State and Head Double Hierarchy

In this section we are going to show the double hierarchy on the number of states and heads. We start with the head hierarchy. The approach is to consider a fixed accepting computation of some *unary* $1\mathrm{DFA}_m(k)$, and to show that either there are infinitely many different accepting computations or the length of the longest word accepted is at most $2^{k-1}km^k$. So, to some extent the result can be seen as a *pumping argument for unary one-way multi-head finite automata*.

In the following, we say that a computation contains a *cycle* if it contains at least two configurations that coincide with the state and the input symbols scanned, that is, the actual head positions do not matter. The length of every cycle is at most m. Considering only the state and input symbols scanned, we divide a computation into at most $(2k+1)m$ phases. A new phase is entered when the automaton changes its state, or moves one or more heads from the left endmarker, or moves one or more heads onto the right endmarker.

Lemma 3.1 *Let $k, m \geq 1$ and M be a unary 1DFA$_m(k)$ accepting a nonempty language. Then $L(M)$ is either infinite or contains only words strictly shorter than $2^{k-1}km^k$.*

Now we are prepared to show the head hierarchy. To this end, we use the unary singleton language $L_{k,m} = \{a^{(k-1)m^k}\}$ of Example 2.1 as witness.

Theorem 3.2 *Let $m \geq 2$ and $k \geq 1$. For all $k' \geq k(1+\frac{1}{\log_2(m)})$, the family $\mathscr{L}$(1DFA$_m(k)$) is properly included in $\mathscr{L}$(1DFA$_m(k')$).*

In particular, the head hierarchy is strict and tight when the number of states is at least 2^{k-1}.

Theorem 3.3 *Let $k \geq 1$. For all $m \geq 2^{k-1}$, the family $\mathscr{L}$(1DFA$_m(k)$) is properly included in $\mathscr{L}$(1DFA$_m(k+1)$).*

Proof. Similarly as in the proof of Theorem 3.2 we use $L_{k+1,m} \in \mathscr{L}(\text{1DFA}_m(k+1))$ as witness. Since $km^{k+1} = kmm^k \geq k2^{k-1}m^k$, we derive from Lemma 3.1 that $L_{k+1,m}$ is not accepted by any 1DFA$_m(k)$). □

The proof of Lemma 3.1 reveals an interesting property of unary languages accepted by *stateless* 1DFA(k).

Theorem 3.4 *Every unary language accepted by some stateless 1DFA$_1(k)$ is either finite or cofinite.*

Now we turn to the state hierarchy. It is strict and tight for any number of heads.

Theorem 3.5 *Let $k \geq 1$. For all $m \geq 1$, there is a finite unary language belonging to the family $\mathscr{L}$(1DFA$_{m+1}(k)$) but not to $\mathscr{L}$(1DFA$_m(k)$). Therefore, the family $\mathscr{L}$(1DFA$_m(k)$) is properly included in $\mathscr{L}$(1DFA$_{m+1}(k)$).*

The previous theorem can be strengthened in the sense that there is a unary language accepted by some one-head $(m+1)$-state automaton that cannot be accepted by any m-state automaton having an arbitrary number of heads. Clearly, this language cannot be finite.

Theorem 3.6 *Let m be a prime number. There is a unary language belonging to the family $\mathscr{L}$(1DFA$_m(1)$) but not to any family $\mathscr{L}$(1DFA$_{m-1}(k)$), $k \geq 1$.*

4. Head Hierarchy for Stateless Finite Automata

In this section we show an infinite strict and tight head hierarchy for stateless automata using unary languages. The head hierarchy obtained in [6] is based on languages over a growing alphabet, that is, the number of symbols increases with the number of heads. We continue with an example that gives an almost trivial lower bound for the lengths of longest words in finite unary languages accepted by stateless 1DFA(k). However it is best possible for very few heads and we need it in the following. It is worth mentioning that there are also examples showing that the lower bound grows exponentially with k. In order to increase the readability, we use the following short notation. A transition $\delta(s_i, a^k) = (s_{i+1}, 1^k)$ means that the automaton is in state s_i and each of the k heads reads an a. Then the automaton changes its state to s_{i+1} and all k heads move one step to the right.

Example 4.1 *For each $k \geq 1$, the unary singleton language $\{a^{k-1}\}$ is accepted by the 1DFA$_1(k)$ $M = \langle \{s_0\}, \{a\}, k, \delta, \rhd, \lhd, s_0 \rangle$, where the transition function δ is specified as $\delta(s_0, \rhd^{k-j}a^j) = (s_0, 0^{k-(j+1)}1^{(j+1)})$ and $\delta(s_0, a^{k-1}\lhd) = (s_0, 0^k)$, for $0 \leq j \leq k-1$.*

Theorem 4.2 *For all $k \geq 1$, there is a finite unary language belonging to the family $\mathscr{L}$(1DFA$_1(k+1)$) but not to the family $\mathscr{L}$(1DFA$_1(k)$). Therefore, the family $\mathscr{L}$(1DFA$_1(k)$) is properly included in $\mathscr{L}$(1DFA$_1(k+1)$).*

5. Four States are Too Much For Two-Head Automata

In this section, we investigate the emptiness problem for 1DFA(2). It has been shown in [11] that the emptiness problem is undecidable for stateless 1DFA(k) where k is at least three. In [6], the emptiness problem is again studied for stateless multi-head automata. It turned out that the problem is undecidable for stateless two-way DFA(2). The problem has been left open for 1DFA(2). Here, we obtain a first result in this direction and show the undecidability of the problem for 1DFA$_4(2)$ having at least four states. The problem remains open for stateless 1DFA$_1(2)$ and 1DFA(2) with two or three states.

The undecidability of the problem is shown by reduction of the emptiness problem for deterministic linearly space bounded one-tape, one-head Turing machines, so-called linear bounded automata (LBA). Basically, histories of LBA computations are encoded in single words that are called *valid computations* (see, for example, [4]). We may assume that LBAs get their input in between two endmarkers, make no stationary moves, accept by halting in some unique state f on the leftmost input symbol, and are sweeping, that is, the read-write head changes its direction at endmarkers only. Let Q be the state set of some LBA M, where q_0 is the initial state, $T \cap Q = \emptyset$ is the tape alphabet containing the endmarkers $\rhd$ and $\lhd$, and $\Sigma \subset T$ is the input alphabet. Since M is sweeping, the set of states can be partitioned into Q_R and Q_L of states appearing in right-to-left and in left-to-right moves. A configuration of M can be written as a string of the form $\rhd T^*QT^*\lhd$ such that, $\rhd t_1t_2\cdots t_ist_{i+1}\cdots t_n\lhd$ is used to express that $\rhd t_1t_2\cdots t_n\lhd$ is the tape inscription, M is in state s, for $s \in Q_R$ scans tape symbol t_{i+1}, and for $s \in Q_L$ scans tape symbol t_i. Now we consider words of the form $\$w_0\$w_1\$\cdots\$w_m$, where $\$ \notin T \cup Q$, $w_i \in T^*QT^*$ are configurations of M with endmarkers chopped off, w_0 is an initial configuration of the form $q_0\Sigma^*$, $w_m \in \{f\}T^*$ is a halting, that is, accepting configuration, and w_{i+1} is the successor configuration of w_i. These words are encoded so that every state symbol is merged together with its both adjacent symbols into a metasymbol. To this end, we assume that the LBA input is nonempty, and rewrite every substring of $\$w_0\$\cdots\$w_m$ having the form tqt' to $[t,q,t']$, where $q \in Q$, $t,t' \in T \cup \{\$\}$. The set of these encodings is defined to be the set of valid computations of M. We denote it by VALC(M).

Lemma 5.1 *Let M be an LBA. Then a 1DFA$_4(2)$ accepting VALC(M) can effectively be constructed.*

Theorem 5.2 *Emptiness is undecidable for 1DFA$_n(2)$ with $n \geq 4$.*

Proof. Let M be an LBA. According to Lemma 5.1 we can effectively construct a 1DFA$_4(2)$ M' accepting VALC(M). Clearly, $L(M') = $ VALC(M) is empty if and only if $L(M)$ is either $\{\lambda\}$ or empty. Since the word problem is decidable and emptiness is undecidable for LBAs, the theorem follows. □

References

[1] Chrobak, M.: Hierarchies of one-way multihead automata languages. Theoret. Comput. Sci. 48, 153–181 (1986)

[2] Harrison, M.A.: Introduction to Formal Language Theory. Addison-Wesley, Reading (1978)

[3] Holzer, M., Kutrib, M., Malcher, A.: Complexity of multi-head finite automata: Origins and directions. Theoret. Comput. Sci. 412, 83–96 (2011)

[4] Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)

[5] Ibarra, O.H.: A note on semilinear sets and bounded-reversal multihead pushdown automata. Inform. Process. Lett. 3, 25–28 (1974)

[6] Ibarra, O.H., Karhumäki, J., Okhotin, A.: On stateless multihead automata: Hierarchies and the emptiness problem. Theoret. Comput. Sci. 411, 581–593 (2010)

[7] Kutyłowski, M.: One-way multihead finite automata and 2-bounded languages. Math. Systems Theory 23, 107–139 (1990)

[8] Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. 3, 114–125 (1959)

[9] Rosenberg, A.L.: On multi-head finite automata. IBM J. Res. Dev. 10, 388–394 (1966)

[10] Sudborough, I.H.: Bounded-reversal multihead finite automata languages. Inform. Control 25, 317–328 (1974)

[11] Yang, L., Dang, Z., Ibarra, O.H.: On stateless automata and P systems. Int. J. Found. Comput. Sci. 19, 1259–1276 (2008)

[12] Yao, A.C., Rivest, R.L.: $k+1$ heads are better than k. J. ACM 25, 337–340 (1978)

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 85 – 89

Manipulation of Finite Automata A Small Leak Will Sink a Great Ship

Martin Kutrib Katja Meckel Matthias Wendlandt

Institut für Informatik, Justus-Liebig-Universität Giessen
Arndtstr. 2, 35392 Giessen
{kutrib,meckel,wendlandt}@informatik.uni-giessen.de

Abstract

We investigate the state complexity of NFA that are obtained by manipulations of minimal DFA that may occur when transmitting this automaton to a receiver. Further, we try to fix the error that occurred and construct minimal DFA for subsets of the languages of the manipulated and fixed automata.

1. Introduction

The storage of a finite automaton can be done in different ways. One is to define a set of accepting states and store the transitions in an adjacency list for every letter of the alphabet. Another way to store the transitions is to write them down as a matrix. This can be done by using a matrix for every single letter and marking the connected states by a special symbol, or by a single matrix with entries that are a list of letters of the alphabet.

Both ways of storage may lead to problems when there happens to be for example a hardware defect or when transmitting this stored automaton to another person. It is well known that by sending information it may happen that some bits get lost or get modified. Such a modification may have consequences for the transmitted automaton. It may happen that for a matrix, one entry may be deleted. This means that a complete transition gets lost. It is also possible that the adjacency list for one state gets lost. The result is the loss of a state and all its transitions. In both cases, the receiver of the automaton gets a nondeterministic finite automaton even though the sent automaton is deterministic.

It is well known that given some n-state NFA one can always construct a language equivalent DFA with at most 2^n states [6]. In fact, later it was shown independently in [3, 4, 5] that this exponential upper bound is best possible. The consequences for our problem are that if the receiver needs to work on a minimal DFA he may deal with exponentially more states than the transmitter of the automaton.

There exist a few manipulations that may be a result of a flaw in the transmission of an automaton. We concentrate on the following ones: exchange of the letter of the transition, interchange of the source and the target of a transition, deletion of a transition, insertion of a transition, deletion of a (non-)accepting state with all its incoming and outgoing transitions, and inversion of the acceptance/non-acceptance of a state. Some of these manipulations only occur if there happened more than one modification while transmitting the data.

2. Deterministic State Complexity

In this section we are interested in the number of states for a minimal DFA that accepts the language of the manipulated automaton depending on the number of states of the sent DFA, because if this number is not that much bigger than the one of the original automaton, it is not too bad with regards to the state complexity. For regular languages we summarize our results in Table 1.

Manipulation	Regular Languages
exchange letter of transition	2^n
interchange source and target of transition	2^n
delete transition	$n+1$
insert transition	2^n-1
delete accepting state	n
delete non-accepting state	n
drop acceptance for a state	n
add acceptance for a state	n

Table 1: Deterministic state complexities for manipulations of n-DFA accepting regular languages

Since some of these bounds are exponential, the question arises if we get better results when considering DFA accepting subregular language families. Prominent examples for language families that do not meet the exponential upper bound are the unary regular language family [1, 2] and the family of finite languages [8]. Our results for unary regular languages and finite languages are summarized in Table 2.

3. Error-Recognition and Error-Fixing

Apart from deterministic state complexity it is also necessary to take a look at the languages that are accepted by the NFA that are the results of the manipulations. In the usual case, the language of the received automaton differs from the language of the transmitted DFA. Thus, it is interesting to know if it is possible to recognise and perhaps even fix the flaw in the automaton. For error-recognition it is important that the receiver of the automaton knows that he should have a complete and minimal DFA at hand.

The manipulations that insert a transition, interchange the source and the target of a transition and exchange the letter of a transition introduce non-determinism to the DFA. Thus, one can recognise that an error has occurred. If there would be only one transition that is modified, it is also recognisable which error has occurred since those three manipulations result in a unique pattern. If there are at least two transitions that are manipulated it is impossible to distinguish between these errors. If the type of manipulation is recognisable it is only possible to fix the error for interchanged source and target since in all other cases it is not decidable which transition is the manipulated one and which is the one that already existed in the original automaton.

Manipulation	Finite Languages	Unary Regular Languages
exchange letter of transition	$\frac{n^2-3n}{2}+3 \leq \cdot \leq \frac{n(n+1)}{2}$	n
interchange source and target of transition	$2^{\frac{n+1}{2}}+2^{\frac{n-1}{2}}-1 \leq \cdot \leq 2^{n-1}$	$n+1$
delete transition	n	$n+1$
insert transition	$2^{\frac{n+1}{2}}+2^{\frac{n-1}{2}}-1 \leq \cdot \leq 2^{n}$	$n^2-3n+1 \leq \cdot \leq n^2$
delete accepting state	$n-1$	n
delete non-accepting state	n	n
drop acceptance for a state	n	n
add acceptance for a state	n	n

Table 2: Deterministic state complexities for manipulations of n-DFA accepting finite languages and unary regular languages

If at least one transition was deleted during the transmission it is also possible to recognise that an error occurred since then the automaton is not complete anymore. But it is not possible to fix it since even if it is known where to start the transition and on which letter it is performed, it still is impossible to recognise which state is the target for this transition. This is only decidable if the fixed automaton is complete and minimal but there may exist more than one target for the missing transition that results in such a DFA.

When a complete state with all its incoming and outgoing transitions is deleted in the automaton on its way to the receiver, it is recognisable that an error has occurred since the automaton is not complete anymore. Fixing is impossible since it is unknown which are the targets of the transitions of the missing state.

In general, it is not possible to recognise that a manipulation has occurred that dropped or added the acceptance for at least one state. In case the received automaton is not minimal anymore but still complete it is possible to recognise that an error occurred but it is impossible to fix it. There may have occurred more than one other error or, if it is known that it was an error concerning the (non-)acceptance of a state, there may exist more than one way to get a minimal and complete DFA when trying to fix the error [7].

The results from above hold true for finite and for (unary) regular languages. There is only one obvious exception for unary languages. The exchange of the letter of a transition does not change anything in the DFA since there exists only a single letter. So this manipulation does not need to be fixed for DFA accepting unary regular languages.

4. Construction of Deterministic Finite Automata for Sets Related to the Accepted Languages

Since in most cases it is impossible to fix the error we want to know which words of the accepted language L' of the manipulated automaton belong to the language L that is accepted by the original DFA. These are the words of the set $L \cap L'$. We are also interested in those words that are rejected by both automata ($\overline{L} \cap \overline{L'}$) and in those which may belong to L and are accepted by at least one of the possible fixed automata.

In a first step we assume that only one error occurred. For manipulations on the transitions we now can recognise the type of the error. We can construct three automata from the received NFA: one that recognises $L \cap L'$, another one for $\overline{L} \cap \overline{L'}$, and a third automaton that accepts all words of the fixed automata for which it is unsure if they belong to the original language. This set includes the original language.

The construction of a DFA that accepts the words of $L \cap L'$ is very simple. If one transition gets lost while transmitting the DFA, one can simply transform the NFA on hand into a deterministic automaton. For the other types of transition-errors, a DFA for this set of words is also quite simple to construct. By deleting all transitions that provide the non-determinism from the NFA that was received and determinizing this automaton, we obtain the minimal and complete DFA, that accepts exactly the "good" part of the language. All of those DFA have a deterministic state complexity of at most $n+1$, if the automaton that was transmitted is an n-state DFA since there exists no non-determinism in the constructed NFA.

To build a DFA for $\overline{L} \cap \overline{L'}$, we also delete the non-deterministic transitions (if they exist). Since this NFA is simply an incomplete DFA we can build the complement by switching the type of acceptance of all states. Thus, a minimal and complete DFA for the rejected part of the language needs at most $n+1$ states if the original DFA consists of n states.

This leaves the construction of an automaton that accepts all the words, that may belong to the language of the original DFA. For such a DFA we need to fix the error. If a transition of the n-state DFA gets lost on its way from the transmitter to the receiver, it is obvious in which state it starts and on which letter it is performed. The only problem is the target. For every possible target we construct a DFA that resembles the received NFA and add the transition with the chosen target. If we unite the languages accepted by these DFA we get the set of words that may or may not belong to the original language. A complete and minimal DFA for this language needs at most $n^3 - n$ states.

In case a transition is inserted into the n-state DFA on its way from the transmitter to the receiver, it is obvious for which state it is inserted, and on which letter it is performed. The only problem is to decide which of the transitions on the same letter in this state is the one that was inserted. Thus, we construct two DFA that resemble the received NFA and delete one of the two possible transitions. If we unite the languages accepted by these DFA we get the set of words that may or may not belong to the original language. A minimal DFA for this language needs at most n^2 states.

For the exchange of the letter on which a transition is performed, we can construct a minimal DFA for the words that may or may not belong to the accepted language of the original n-state DFA by building DFA where the exchange of the letter is corrected, building an NFA for the union of their accepted languages, and determinizing it. This DFA needs at most n^2 states.

This leaves the error that interchanges the source and target of a transition. This manipulation is easily to be fixed if only one transition is affected. By again interchanging the source and target of the transition we obtain the original DFA. Thus, n states suffice for the "may or may not"-language.

We summarize our results on the constructions for the different types of languages and the considered types of errors in Table 3.

Manipulation	Accepted Words	Rejected Words	"May Or May Not"-Words
exchange letter of transition	$n+1$	$n+1$	n^2
interchange source and target of transition	$n+1$	$n+1$	n
delete transition	$n+1$	$n+1$	n^3-n
insert transition	$n+1$	$n+1$	n^2

Table 3: Deterministic state complexities for the construction of minimal DFA from a manipulated n-DFA for subsets of the accepted language

References

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

[2] M. CHROBAK, Errata to "finite automata and unary languages". *Theoret. Comput. Sci.* **302** (2003), 497–498.

[3] O. LUPANOV, A comparison of two types of finite sources. *Problemy Kybernetiki* **9** (1963), 321–326. (in Russian), German translation: Über den Vergleich zweier Typen endlicher Quellen. Probleme der Kybernetik 6 (1966), 328-335.

[4] A. R. MEYER, M. J. FISCHER, Economy of description by automata, grammars, and formal systems. In: *Symposium on Switching and Automata Theory (SWAT 1971)*. IEEE, 1971, 188–191.

[5] F. MOORE, On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. *IEEE Trans. Comput.* **20** (1971), 1211–1214.

[6] M. RABIN, D. SCOTT, Finite automata and their decision problems. *IMB J. Res. Dev.* **3** (1959), 114–125.

[7] A. RESTIVO, R. VAGLICA, Automata with Extremal Minimality Conditions. In: *Developments in Language Theory*. LNCS 6224, Springer, 2010, 399–410.

[8] K. SALOMAA, S. YU, NFA to DFA transformation for finite languages over arbitrary alphabets. *J. Autom., Lang. Comb* **2** (1997), 177–186.

Lohmann Words and More Clusters of Words

Gerhard Lischke

Fakultät für Mathematik und Informatik,
Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 1-4, D-07743 Jena, Germany
gerhard.lischke@uni-jena.de

Six kinds of both of primitivity and periodicity of words have been introduced by Ito and Lischke [1]. These give rise to define six kinds of roots of a nonempty word, and it was the question whether there exist words u such that all the six roots of u are different each other. It was first assumed that such words do not exist, but in 2010 Georg Lohmann discovered the first of them. We will show that for such a Lohmann word u the six roots of u fulfil a fixed prefix relationship, and we give a sufficient condition for a word to be a Lohmann word. If u has exactly k different roots for $1 < k < 6$, then several prefix relationships between them are possible. With the notion of k-cluster we give an exact classification of all these possible relationships and investigate whether the appropriate words are periodic or not. Before defining the roots let us repeat the most important notions which we will use.

We restrict to the **nontrivial** alphabet $X = \{a,b\}$ with the fixed ordering $a < b$. The **empty word** is denoted by e, and $X^+ =_{Df} X^* \setminus \{e\}$. For words $p,q \in X^*$, p is a **prefix of** q, in symbols $p \sqsubseteq q$, if there exists $r \in X^*$ such that $q = pr$. p is a **strict prefix of** q, in symbols $p \sqsubset q$, if $p \sqsubseteq q$ and $p \neq q$. $Pr(q) =_{Df} \{p : p \sqsubset q\}$ is the **set of all strict prefixes of** q (including e if $q \neq e$). p is a **suffix of** q, if there exists $r \in X^*$ such that $q = rp$. Besides the usual concatenation of words we consider the following **concatenation with overlaps** or **folding operation**:

For $p,q \in X^*$, $p \otimes q =_{Df} \{w_1w_2w_3 : w_1w_3 \neq e \wedge w_1w_2 = p \wedge w_2w_3 = q\}$,
$p^{\otimes 0} =_{Df} \{e\}$, $p^{\otimes n} =_{Df} \bigcup\{w \otimes p : w \in p^{\otimes n-1}\}$ for $n \geq 1$.
For sets $A,B \subseteq X^*$, $A \otimes B =_{Df} \bigcup\{p \otimes q : p \in A \wedge q \in B\}$.

This operation is illustrated by the following example.
Let $p = aabaa$. Then $p \otimes p = p^{\otimes 2} = \{aabaaaabaa, aabaaabaa, aabaabaa\}$.

A nonempty word p is **periodic** if and only if it is a concatenation of two or more copies of the same word v, $p = v^n$, $n \geq 2$. A nonempty word is **primitive** if and only if it is not periodic.

The next definition is taken from [1].

Definition 1 *Let* $u \in X^+$.
The shortest word v *such that there exists a natural number* n *with* $u = v^n$
is called the **root** *of* u*, denoted by* $root(u)$.
The shortest word v *such that there exists a natural number* n *with* $u \in v^n \cdot Pr(v)$
is called the **strong root** *of* u*, denoted by* $sroot(u)$.
The shortest word v *such that there exists a natural number* n *with* $u \in v^{\otimes n}$
is called the **hyperroot** *of* u*, denoted by* $hroot(u)$.

The shortest word v such that there exists a natural number n with $u \in \{v^n\} \otimes Pr(v)$ is called the **super strong root** *of u, denoted by $ssroot(u)$.*
The shortest word v such that there exists a natural number n with $u \in v^{\otimes n} \cdot Pr(v)$ is called the **strong hyperroot** *of u, denoted by $shroot(u)$.*
The shortest word v such that there exists a natural number n with $u \in v^{\otimes n} \otimes Pr(v)$ is called the **hyperhyperroot** *of u, denoted by $hhroot(u)$.*

Definition 2 *For word functions f and g having the same domain $dom(f)$,*
$f \sqsubseteq g =_{Df} \forall u(u \in dom(f) \rightarrow f(u) \sqsubseteq g(u))$.

Theorem 1 *The partial ordering $\sqsubseteq$ for the functions from Definition 1 is given in Figure 1.*

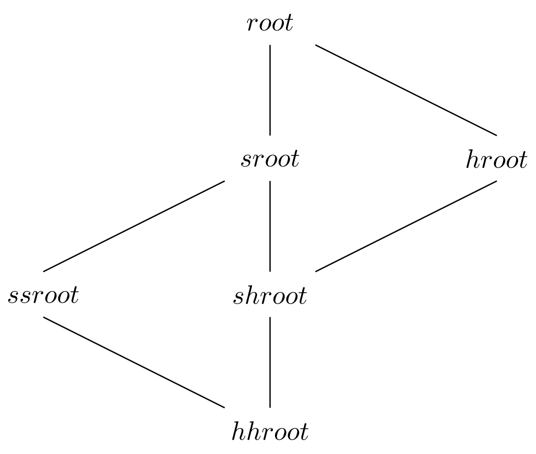


Figure 1: Partial ordering of the root-functions

The proof follows from the definition and can be done as a simple exercise. It is also contained in [1] and [2].

Definition 3 *Let $k \in \{1,2,3,4,5,6\}$. A word $u \in X^+$ is called a k*-**root word** *if $\{root(u), sroot(u), hroot(u), ssroot(u), shroot(u), hhroot(u)\}$ has exactly k elements.*
A 6-root word is also called a **Lohmann word**.

Definition 4 *Let $k \in \{1,2,3,4,5,6\}$.*
A k-**cluster** *is a set of the form $[\alpha_{11} \cdots \alpha_{1i_1} / \alpha_{21} \cdots \alpha_{2i_2} / \cdots / \alpha_{k1} \cdots \alpha_{ki_k}]$,*
where $i_1 + i_2 + \cdots + i_k = 6$,
$\{\alpha_{11}, \ldots, \alpha_{1i_1}\} \cup \{\alpha_{21}, \ldots, \alpha_{2i_2}\} \cup \cdots \cup \{\alpha_{k1}, \ldots, \alpha_{ki_k}\} = \{hh, ss, sh, h, s, r\}$ *and*
$$[\alpha_{11} \cdots \alpha_{1i_1} / \cdots / \alpha_{k1} \cdots \alpha_{ki_k}] =_{Df} \{u : u \in X^* \wedge \alpha_{11} root(u) = \cdots = \alpha_{1i_1} root(u) \sqsubset \alpha_{21} root(u) = \cdots = \alpha_{2i_2} root(u) \sqsubset \cdots \sqsubset \alpha_{k1} root(u) = \cdots = \alpha_{ki_k} root(u)\}.$$

For $\alpha = r$, $\alpha root(u)$ means $root(u)$. Thus, for instance, $[\,hh\,sh\,h\,/\,ss\,/\,s\,r\,]$ denotes the set of all words u satisfying $hhroot(u) = shroot(u) = hroot(u) \sqsubset ssroot(u) \sqsubset sroot(u) = root(u)$.

We will say, that a cluster **exists** if it is not empty.
For instance, clusters of the form $[\,r\,/\cdots]$ or $[\cdots/\,hh\cdots]$ cannot exist.

Definition 5 *A cluster $\mathcal{C}$ is called a* **primitive** *cluster if it contains only primitive words. A cluster $\mathcal{C}$ is called a* **periodic** *cluster if it contains only periodic words. A cluster is called a* **mixed** *cluster if it contains both primitive and periodic words.*

Lemma 1 *For a nonempty word u, only one of the following relationships is possible (where we will use hh, ss, sh, h, s, r, respectively, as shorthand expressions instead of $hhroot(u)$, $ssroot(u)$, $shroot(u)$, $hroot(u)$, $sroot(u)$, $root(u)$, respectively):*
(1) $hh \sqsubseteq ss \sqsubseteq sh \sqsubseteq h \sqsubseteq s \sqsubseteq r$,
(2) $hh \sqsubseteq ss \sqsubseteq sh \sqsubseteq s \sqsubseteq h \sqsubseteq r$,
(3) $hh \sqsubseteq sh \sqsubseteq h \sqsubseteq ss \sqsubseteq s \sqsubseteq r$,
(4) $hh \sqsubseteq sh \sqsubseteq ss \sqsubseteq h \sqsubseteq s \sqsubseteq r$,
(5) $hh \sqsubseteq sh \sqsubseteq ss \sqsubseteq s \sqsubseteq h \sqsubseteq r$.

Lemma 2 *If $shroot(u) = ssroot(u)$ for some $u \in X^+$, then also $sroot(u) = shroot(u) = ssroot(u)$, and therefore there is no 5-cluster with $sh = ss$.*

Lemma 3 *If $ssroot(u) = hroot(u)$ for some $u \in X^+$, then also $root(u) = sroot(u) = ssroot(u) = hroot(u)$, and therefore there is no 4-cluster and no 5-cluster with $ss = h$.*

Lemma 4 *If $u \in X^+$ is in a k-cluster with $k > 1$ and $sroot(u) = root(u)$ then u is periodic.*

Lemma 5 *If $ssroot(u) \sqsubset root(u)$ for some $u \in X^+$, then u is not periodic.*

Lemma 6 *There is no cluster with $ss \sqsubset s = r$.*

Lemma 7 *If $ssroot(u) \sqsubset hroot(u) \sqsubset root(u)$ for some $u \in X^+$, then $ssroot(u) = sroot(u)$, and therefore there is no 6-cluster with $ss \sqsubset h$.*

The proofs of the lemmas 1 to 4 are simple but not so for Lemma 5 and Lemma 7. Lemma 6 is a consequence of Lemma 4 and Lemma 5. Exemplary, we prove Lemma 7.

Proof. Let $v = ssroot(u) \sqsubset p = hroot(u) \sqsubset root(u)$ and $ssroot(u) \sqsubset sroot(u)$. Then by Lemma 5, u is not periodic and therefore $u = root(u) \in p^{\otimes m}$ and $u \in v^n \otimes v'$ for some $m \geq 2$, $n \geq 1$ and $v' \sqsubset v$. There must be an overlapping between v^n and v' since otherwise $sroot(u) \sqsubseteq v = ssroot(u)$. This means, $u = v^n v''$ for a strict subword v'' of v, more exactly, $v = v_l v'' v_r$ with $v_l, v_r \neq e$. If $n = 1$, then $u = vv''$ with $|u| \leq 2|v| - 2$ and therefore $|v| \geq \frac{|u|}{2} + 1$ and $|p| \geq \frac{|u|}{2} + 2$. Then $m = 2$ must follow and therefore $p = wqw$ and $u = wqwqw \in p^{\otimes 2}$. But then $ssroot(u) \sqsubseteq sroot(u) \sqsubseteq wq$ which means, $|v| \leq \frac{|u|}{2}$, a contradiction. Now we must have $u = v^n v'' \in p^{\otimes m}$ with $m, n \geq 2$, $v \sqsubset p$. Let $p = v^k q$ for some q with $|q| < |v|$ and $1 \leq k \leq n$. It is $q \neq e$ because otherwise $hroot(u) \sqsubseteq v$.

It is $k < n$ because otherwise $|p| > \frac{2}{3}|u|$ contradicting to the fact that the shortest word p such that $u \in p^{\otimes m}$ for some $m \geq 2$ cannot be longer than $\frac{2|u|}{3}$. Then $q \sqsubset v$ because of $p = v^k q \sqsubset u = v^n v''$. Now we have $v = qq'$ for nonempty words q, q', and $p = (qq')^k q$. p is the hyperroot of u, and therefore $k = 1$ because otherwise $qq'q$ would be a shorter candidate for the hyperroot. Hence $u = (qq')^n v'' \in (qq'q)^{\otimes m}$ and v'', which is shorter than $v = qq'$, is a suffix of $q'q$. If $|v''| \leq |q|$ then v'' is a suffix of q and $qq'v'' = v''q'q$ must follow. This means, $v'' \sqsubseteq q$ and $sroot(u) \sqsubseteq qq' = ssroot(u)$, a contradiction. If $|q| < |v''| < |qq'|$ then remember that $v = qq' = v_l v'' v_r$ with $v_l, v_r \neq e$ and v_l is a suffix of v. From $u = (v_l v'' v_r)^n v'' \in (v_l v'' v_r q)^{\otimes m}$ we get $v'' = v_1 q$ for some v_1 which is a strict nonempty suffix of q', and $v_l v_1 q v_r v_1 q = \cdots v_l v_1 q v_r q$, therefore $q v_r v_1 = v_1 q v_r$ and v_l is a suffix of $v_l v_1$. Since $q v_r v_1 = v_1 q v_r$, by the Lemma of Lyndon/Schützenberger ([3], see also [2]) $q v_r$ and v_1 must be powers of a common primitive word x, $v_1 = x^\alpha$, $q v_r = x^\beta$ for $\alpha, \beta \geq 1$. Then $v = v_l v_1 q v_r = v_l x^{\alpha+\beta}$. Since v_l is a suffix of $v_l v_1$ it follows that $v_l = y v_1^s$ for some $s \geq 0$ and a nonempty suffix y of v_1. Since $v_1 = x^\alpha$ we can assume that $v_l = y x^t$ for some $t \geq 0$ and y is a nonempty suffix of x. Now we have $v = y x^c$ with $c = t + \alpha + \beta \geq 2$ and $u = (y x^c)^n v''$ where $x^\alpha \sqsubseteq v'' \sqsubset x^{\alpha+\beta} \sqsubseteq x^c$. This means, $v'' = x^i x_1$ where $\alpha \leq i < \alpha + \beta - 1$ and $e \sqsubseteq x_1 \sqsubseteq x$. Then $u = (y x^c)^n x^i x_1$. If $x_1 = e$ or $x_1 = x$ then $hroot(u) \sqsubseteq yx \sqsubset v$, a contradiction. Otherwise, $qq'q = y x^c q$ with $e \sqsubset q \sqsubset x^\beta$ cannot be the hyperroot of u. □

Theorem 2 *All Lohmann words belong to the cluster* $[hh/sh/h/ss/s/r]$ *which is a primitive cluster. I.e., if* u *is a 6-root word then* u *is primitive and*
$hhroot(u) \sqsubset shroot(u) \sqsubset hroot(u) \sqsubset ssroot(u) \sqsubset sroot(u) \sqsubset root(u) = u$.

The proof follows with Lemmas 1, 5 and 7.

Definition 6 *For finite sequences* $(k_1,\ldots,k_r)$ *and* $(t_1,\ldots,t_m)$ *of natural numbers, let*
$(k_1,\ldots,k_r) \odot (t_1,\ldots,t_m) =_{Df} (k_1,\ldots,k_{r-1},k_r+t_1,t_2,\ldots,t_m)$ *and*
$(k_1,\ldots,k_r)^{\odot 0} =_{Df} (0)$, $(k_1,\ldots,k_r)^{\odot s} =_{Df} (k_1,\ldots,k_r)^{\odot s-1} \odot (k_1,\ldots,k_r)$ *for* $s \geq 1$.
Let $(k_1,\ldots,k_r,t)$ *be a sequence of natural numbers with* $r \geq 2$, $2 \leq k_1 \leq k_i \leq 2k_1$ *for each* $i \in \{1,\ldots,r\}$ *and* $0 \leq t \leq k_1$, *and let* $s \geq 2$ *and*
$(k_1,\ldots,k_r,t)^{\odot s} = (k_1,\ldots,k_r,k_{r+1},\ldots,k_{s\cdot r},t)$.
If $t \neq 0$ *and* $k_1 \leq k' < k_1 + t$ *then* $(k_1,\ldots,k_{s\cdot r},k')$ *is called an* L**-sequence** *with its* **producer** $(k_1,\ldots,k_r,t)$; *if* $t = 0$ *and* k' *with* $\max\{k_1,\ldots,k_r\} < k' \leq 2k_1$ *exists then* $(k_1,\ldots,k_{s\cdot r-1},k')$ *is called an* L**-sequence** *with its* **producer** $(k_1,\ldots,k_r,0)$.

Theorem 3 *Let* v *and* w *be words such that* $e \sqsubset v \sqsubset w$, $wv \not\sqsubseteq p^l$ *for some* $p \sqsubset w$ *and* $l > 1$, *let* $(k_1,\ldots,k_n)$ *be an* L*-sequence and* k^+ *the greatest number in this sequence, and let* w' *be a word such that* $w^{k-1}v \sqsubset w' \sqsubset w^k v$ *for some* $k \geq 2$ *with* $k^+ - k_1 \leq k \leq k_1$ *and* $w^2 \sqsubseteq w'$. *Then* $u = w^{k_1} v w^{k_2} v \cdots w^{k_n} v w'$ *is a Lohmann word.*

Proof. Let $(k_1,\ldots,k_r,t)$ be the shortest producer of the L-sequence $(k_1,\ldots,k_n)$. Then the proof is done by verifying the roots:

$$\begin{array}{rclrcl} hhroot(u) & = & wv, & ssroot(u) & = & w^{k_1} v w^{k_2} v \cdots w^{k_r} v w^t, \\ shroot(u) & = & w^{k_1} v, & ssroot(u) & \sqsubset & sroot(u) \sqsubseteq w^{k_1} v w^{k_2} v \cdots w^{k_{n-1}} v w^{k_n - k_1}, \\ hroot(u) & = & w^{k_1} v w', & root(u) & = & u. \end{array}$$

□

Example 1 *(2,2,0), (2,3,0), (2,2,1), (2,4,1), (2,4,2), (3,4,4,2) are producer of the L-sequences (2,2,2,3), (2,3,2,4), (2,2,3,2,2), (2,4,3,4,2), (2,4,4,4,3), (3,4,4,5,4,4,5,4,4,3), respectively. (2,3,0) is also a producer of (2,3,2,3,2,4). From (2,2,2,3) with $w = ab$, $v = a$, and $w' = (ab)^2$ we get the smallest Lohmann word $ababaababaababaabababaabab$.*

By Lemma 1, altogether 89 k-clusters are possible: 1 for $k = 1$, 8 for $k = 2$, 23 for $k = 3$, 32 for $k = 4$, 20 for $k = 5$, and 5 for $k = 6$. For $k \in \{2,3,4,5\}$ they are listed in the Figures 2 to 5. We have already seen that there exists only one 6-cluster. The only 1-cluster is $[hhssshhsr]$. The most of the remaining clusters cannot exist by our Lemmas, where the corresponding numbers are given in the tables. The sign + means that the existence of the corresponding cluster is sure and was verified by computer experiments.

$[hh/ssshhsr]$	+	$[hhshh/sssr]$	+
$[hhss/shhsr]$	6	$[hhssshh/sr]$	2,3,6
$[hhsh/sshsr]$	+	$[hhssshs/hr]$	+
$[hhsssh/hsr]$	2,6	$[hhssshhs/r]$	3

Figure 2: 2-clusters

$[hh/ss/shhsr]$	6	$[hhsssh/s/hr]$	2
$[hh/sssh/hsr]$	2,6	$[hhssshs/h/r]$	+
$[hh/ssshh/sr]$	2,3,6	$[hh/sh/hsssr]$	+
$[hh/ssshhs/r]$	3	$[hh/shh/sssr]$	+
$[hhss/sh/hsr]$	6	$[hhsh/h/sssr]$	+
$[hhss/shh/sr]$	6,7	$[hhsh/hss/sr]$	3,6
$[hhss/shhs/r]$	7	$[hhsh/hsss/r]$	3
$[hhsssh/h/sr]$	2,6,7	$[hhshh/ss/sr]$	6
$[hhsssh/hs/r]$	2,7	$[hhshh/sss/r]$	+
$[hhssshh/s/r]$	2,3	$[hhsh/ss/hsr]$	6
$[hh/ssshs/hr]$	+	$[hhsh/sss/hr]$	+
$[hhss/shs/hr]$	+		

Figure 3: 3-clusters

Theorem 4 *There exists only one mixed cluster namely the 1-cluster $[hhssshhsr]$. The clusters $[hhsh/sshsr]$, $[hhshh/sssr]$, $[hh/ssshhsr]$, $[hhsh/h/sssr]$, $[hh/sh/hsssr]$, $[hh/shh/sssr]$ and $[hh/sh/h/sssr]$ are periodic clusters, and all other existing clusters are primitive.*

The proof follows from our Figures with Lemmas 4 and 5.

$[hhss/shh/s/r]$	7	$[hhsh/ss/hs/r]$	7
$[hhss/sh/hs/r]$	7	$[hhsh/ss/h/sr]$	6,7
$[hhss/sh/h/sr]$	6,7	$[hhsh/sss/h/r]$	+
$[hh/sssh/hs/r]$	2,7	$[hhsh/ss/s/hr]$	+
$[hh/sssh/h/sr]$	2,6,7	$[hh/sh/sss/hr]$	+
$[hh/ss/shh/sr]$	6,7	$[hhsssh/h/s/r]$	2,7
$[hhss/shs/h/r]$	7	$[hh/ssshh/s/r]$	2,3
$[hhss/sh/s/hr]$	+	$[hh/ss/shhs/r]$	7
$[hh/sssh/s/hr]$	2	$[hh/ss/sh/hsr]$	6
$[hh/ss/shs/hr]$	+	$[hhsssh/s/h/r]$	2,7
$[hhsh/hss/s/r]$	3	$[hh/ssshs/h/r]$	+
$[hhsh/h/sss/r]$	+	$[hhshh/ss/s/r]$	+
$[hhsh/h/ss/sr]$	6	$[hh/sh/hsss/r]$	3
$[hh/shh/sss/r]$	+	$[hh/sh/h/sssr]$	+
$[hh/shh/ss/sr]$	6	$[hhshss/h/s/r]$	2,7
$[hh/sh/hss/sr]$	3,6	$[hh/sh/ss/hsr]$	6

Figure 4: 4-clusters

$[hhss/sh/h/s/r]$	7	$[hh/shh/ss/s/r]$	+
$[hh/sssh/h/s/r]$	2,7	$[hh/sh/hss/s/r]$	3
$[hh/ss/shh/s/r]$	7	$[hh/sh/h/sss/r]$	+
$[hh/ss/sh/hs/r]$	7	$[hh/sh/h/ss/sr]$	6
$[hh/ss/sh/h/sr]$	6,7	$[hhsh/ss/h/s/r]$	7
$[hhss/sh/s/h/r]$	7	$[hh/sh/ss/hs/r]$	7
$[hh/sssh/s/h/r]$	2,7	$[hh/sh/ss/h/sr]$	6,7
$[hh/ss/shs/h/r]$	7	$[hhsh/ss/s/h/r]$	7
$[hh/ss/sh/s/hr]$	+	$[hh/sh/sss/h/r]$	+
$[hhsh/h/ss/s/r]$	+	$[hh/sh/ss/s/hr]$	+

Figure 5: 5-clusters

Acknowledgements This work was supported by the Fellowship Program of the Japan Society for Promotion of Science (JSPS) under No. S-11717. The author is very grateful to Masami Ito for the cooperation and for the hospitality during his stay in Kyoto, and to Georg Lohmann in Apolda for his interest and the assistence with his computer programs.

References

[1] M. Ito, G. Lischke, Generalized periodicity and primitivity for words, *Math. Log. Quart.* 53 (2007), 91–106, Corrigendum in *Math. Log. Quart.* 53 (2007), 642–643.

[2] G. Lischke, Primitive words and roots of words, *Acta Univ. Sapientiae, Informatica* 3, 1 (2011), 5–34.

[3] R. C. Lyndon, M. P. Schützenberger, On the equation $a^M = b^N c^P$ in a free group, *Michigan Math. Journ.* 9 (1962), 289–298.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 97 – 102

Avoidability of Cubes under Morphic Permutations

Florin Manea[(A)] Mike Müller[(A)] Dirk Nowotka[(B)]

Institut für Informatik, Christian-Albrechts-Universität zu Kiel
D-24098 Kiel, Germany.
{flm,mimu,dn}@informatik.uni-kiel.de

Abstract

We consider the avoidance of patterns involving permutations. One of the remarkable facts is that in this setting the notion of avoidability index (the smallest alphabet size for which a pattern is avoidable) is meaningless since a pattern with permutations that is avoidable in one alphabet can be unavoidable in a larger alphabet. We characterise the (un-)avoidability of all patterns of the form $\pi^i(x)\pi^j(x)\pi^k(x)$, called cubes under morphic permutations here, for all alphabet sizes.[1]

1. Introduction

We are concerned with a generalisation of pattern avoidability [2, 4], by considering patterns with functional dependencies between variables, in particular, we investigate permutations. More precisely, we do allow function variables in the pattern that are morphic extensions of permutations on the alphabet. For example, an instance of the pattern $x\,\pi(x)\,x$ is a word uvu that consists of three parts of equal length, that is, $|u| = |v|$, and v is the image of u under any permutation on the alphabet. For example, $aab|bba|aab$ is an instance of $x\pi(x)x$ for the morphic extension of permutation $a \mapsto b$ and $b \mapsto a$.

Recently, there has been some initial work on avoidance of patterns with involutions which is a special case of the permutation setting considered in this paper (as involutions are permutations of order at most two); see [1, 3]. Since these are the very first considerations on this kind of pattern avoidance at all, we restrict ourselves to cube-like patterns with one variable, occurring three times, and only one function variable, that is, we investigate patterns of the form: $\pi^i(x)\,\pi^j(x)\,\pi^k(x)$ where $i, j, k \geq 0$.

It is worth noting that the notion of avoidability index plays no role in the setting of patterns involving permutations. Contrary to the traditional setting, where once a pattern is avoidable for some alphabet size it remains avoidable in larger alphabets, a pattern with permutations may become unavoidable in a larger alphabet. This is a new and somewhat unexpected phenomenon in the field of pattern avoidance. It does not occur, for example, in the involution setting but requires permutations of higher order.

(A) The work of Florin Manea and Mike Müller is supported by the DFG grant 582014.

(B) The work of Dirk Nowotka is supported by the DFG Heisenberg grant 590179.

[1] This an extended abstract of F. Manea, M. Müller, D. Nowotka: *The Avoidability of Cubes under Permutations*. In Hsu-Chun Yen, Oscar H. Ibarra (eds.): Proceedings of the 16th International Conference on Developments in Language Theory, LNCS 7410, pages 416-427, Taipei, Taiwan, August 14-17, 2012. Springer.

2. Preliminaries

We define $\Sigma_k = \{0, \dots, k-1\}$ to be an alphabet with k letters. For a word w and an integer i with $1 \leq i \leq |w|$ we denote the i-th letter of w by $w[i]$.

If $f : \Sigma_k \to \Sigma_k$ is a permutation, we say that the order of f, denoted $\mathbf{ord}(f)$, is the minimum value $m > 0$ such that f^m is the identity. If $a \in \Sigma_k$ is a letter, the order of a with respect to f, denoted $\mathbf{ord}_f(a)$, is the minimum number m such that $f^m(a) = a$.

A pattern which involves functional dependencies is a term over (word) variables and function variables. For example, $x\pi(y)\pi(\pi(x))y$ is a pattern involving the variables x and y and the function variable π. An instance of a pattern p in Σ_k is the result of substituting every variable by a word in Σ_k^+ and every function variable by a function over Σ_k^*. A pattern is avoidable in Σ_k if there is an infinite word over Σ_k that does not contain any instance of the pattern. Here, we consider patterns with morphic permutations, that is, all function variables are unary and substituted by permutations f where $f(uv) = f(u)f(v)$ for all words u and v.

The infinite Thue-Morse word t is defined as $t = \phi_t^\omega(0)$, for $\phi_t : \Sigma_2^* \to \Sigma_2^*$ where $\phi_t(0) = 01$ and $\phi_t(1) = 10$. It is well-known (see, for instance, [6]) that the word t avoids the patterns xxx (cubes) and $xyxyx$ (overlaps).

Let h be the infinite word defined as $h = \phi_h^\omega(0)$, where $\phi_h : \Sigma_3^* \to \Sigma_3^*$ is a morphism due to Hall [5], defined by $\phi_h(0) = 012$, $\phi_h(1) = 02$ and $\phi_h(2) = 1$. The infinite word h avoids the pattern xx (squares).

The reader is referred to [6] for further details on the concepts discussed here.

3. Main results

We begin this section by showing the avoidability of a series of basic patterns from which we derive avoidability results for more general patterns. Our first result uses the morphism $\alpha : \Sigma_2^* \to \Sigma_3^*$ that is defined by

$$0 \mapsto 02110, \qquad 1 \mapsto 02210.$$

Lemma 3.1 *The infinite word $t_\alpha = \alpha(t)$ avoids the pattern $x\pi(x)x$ in Σ_m, for all $m \geq 3$. This pattern cannot be avoided by words over smaller alphabets.* □

The following lemma is the main tool that we use to analyse the avoidability of cubes under morphic permutations. To obtain this result we apply the morphism $\beta : \Sigma_2^* \to \Sigma_4^*$ defined by

$$0 \mapsto 012013213, \qquad 1 \mapsto 012031023.$$

Lemma 3.2 *Let $t_\beta = \beta(t)$ for the morphism β defined above and let $i, j \in \mathbb{N}$ and f, g be morphic permutations of Σ_m with $m \geq 4$. The word t_β does not contain any factor of the form $uf(u)g(u)$ for any $u \in \Sigma_m^+$ with $|u| \geq 7$. Furthermore, t_β does not contain any factor $uf^i(u)f^j(u)$ with*

$$\left|\{u[\ell], f^i(u)[\ell], f^j(u)[\ell]\}\right| \leq 2,$$

for all $\ell \leq |u|$ and $|u| \leq 6$.

The next result highlights sets of patterns that cannot be simultaneously avoided.

Lemma 3.3 *There is no $w \in \Sigma_3^\omega$ that avoids the patterns $xx\pi(x)$, and $x\pi(x)x$ simultaneously. There is no $w \in \Sigma_3^\omega$ that avoids the patterns $x\pi(x)\pi(x)$, and $x\pi(x)x$ simultaneously.*

Proof. It can be easily seen (for instance, by trying all the possibilities using backtracking) that any word of length at least 9 over Σ_3 contains a word of the form uuu, $uuf(u)$, or $uf(u)u$, for some $u \in \Sigma_3^+$ and some morphic permutation f of Σ_3. Similarly, any word of length at least 10 over Σ_3 contains a word of the form uuu, $uf(u)f(u)$, or $uf(u)u$, for $u \in \Sigma_3^+$ and a morphic permutation f of Σ_3. $\square$

The following result shows the equivalence between the avoidability of several pairs of patterns.

Lemma 3.4 *Let $m \in \mathbb{N}$. A word $w \in \Sigma_m^\omega$ avoids the pattern $xx\pi(x)$ if and only if w avoids the pattern $\pi(x)\pi(x)x$. A word $w \in \Sigma_m^\omega$ avoids the pattern $x\pi(x)\pi(x)$ if and only if w avoids the pattern $\pi(x)xx$. A word $w \in \Sigma_m^\omega$ avoids the pattern $x\pi(x)x$ if and only if w avoids the pattern $\pi(x)x\pi(x)$.*

Proof. If an infinite word w has no factor $uuf(u)$, with $u \in \Sigma_m^+$ and a morphic permutation f of Σ_m, then w does not contain any factor $g(u)g(u)u$, with $u \in \Sigma_m^+$ and a morphic permutation g of Σ_m for which there exists a morphic permutation f of Σ_m such that $g(f(a)) = a$, for all $a \in \Sigma_m$. This clearly means that w avoids $\pi(x)\pi(x)x$ in Σ_m. The other conclusions follow by the same argument. $\square$

The following two remarks are immediate.

- The pattern $\pi^i(x)\pi^i(x)\pi^i(x)$ is avoidable in Σ_m for $m \geq 2$ by the word t.
- The patterns $\pi^i(x)\pi^i(x)\pi^j(x)$ and $\pi^i(x)\pi^j(x)\pi^j(x)$, $i \neq j$, are avoidable in Σ_m for $m \geq 3$ by the word h.

Another easy case of avoidable patterns is highlighted in the next lemma.

Lemma 3.5 *The pattern $\pi^i(x)\pi^j(x)\pi^i(x)$, $i \neq j$, is avoidable in Σ_m, for $m \geq 3$.*

Proof. Assume $i < j$. In this case, setting $y = \pi^i(x)$ we get that the pattern $\pi^i(x)\pi^j(x)\pi^i(x)$ is actually $y\pi^{j-i}(y)y$. We can avoid the last pattern in Σ_m if we can avoid the pattern $y\pi(y)y$ in Σ_m. This pattern is avoidable in alphabets with three or more letters, by Lemma 3.1. Also, $y\pi^{j-i}(y)y$ is avoidable in Σ_2 if and only if $j - i$ is even.

If $i > j$, we take $y = \pi^j(x)$ and we obtain that $\pi^i(x)\pi^j(x)\pi^i(x)$ equals $\pi^{i-j}(y)y\pi^{i-j}(y)$, which is avoidable if $\pi(y)y\pi(y)$ is avoidable. This latter pattern is avoidable over alphabets with three or more letters, by Lemmas 3.1 and 3.4. The pattern is also avoidable in Σ_2 if and only if $i - j$ is even. $\square$

In the next lemma we present the case of the patterns $x\pi^i(x)\pi^j(x)$, with $i \neq j$. For this we need to define the following values:

$$k_1 = \inf\{t : t \nmid |i-j|, t \nmid i, t \nmid j\} \tag{1}$$

$$k_2 = \inf\{t : t \mid |i-j|, t \nmid i, t \nmid j\} \tag{2}$$

$$k_3 = \inf\{t : t \mid i, t \nmid j\} \tag{3}$$

$$k_4 = \inf\{t : t \nmid i, t \mid j\}\,. \tag{4}$$

Remember that $\inf\varnothing = +\infty$. However, note that $\{t : t \nmid |i-j|, t \nmid i, t \nmid j\}$ is always non-empty, and that $k_1 \geq 3$ (as either $|i-j|$ is even or one of i and j is even, so $k_1 > 2$). Also, as $i \neq j$ at least one of the sets $\{t : t \mid i, t \nmid j\}$ and $\{t : t \nmid i, t \mid j\}$ is also non-empty. Further, we define

$$k = \min\{\max\{k_1,k_2\}, \max\{k_1,k_3\}, \max\{k_1,k_4\}\} \tag{5}$$

According to the remarks above, k is always defined (that is $k \neq +\infty$).

Lemma 3.6 *The pattern $x\pi^i(x)\pi^j(x)$, $i \neq j$, is unavoidable in Σ_m, for $m \geq k$.*

Proof. First, let us note that the fact that $m \geq k_1$ means that for every word $u \in \Sigma_m^+$ there exists a morphic permutation f such that $u \neq f^i(u) \neq f^j(u) \neq u$; indeed, we take f to be a permutation such that the orbit of $u[1]$ is a cycle of length k_1, which means that the first letters of u, $f^i(u)$ and $f^j(u)$ are pairwise different. Similarly, the fact that $m \geq k_2$ (when $k_2 \neq +\infty$) means that for every word $u \in \Sigma_m^+$ there exists a morphism f such that $u \neq f^i(u) = f^j(u)$. In this case, we take f to be a permutation such that $\mathbf{ord}_f(u[1]) = k_2$, and f only changes the letters from the orbit of $u[1]$ (thus, $\mathbf{ord}(f) \mid k_2$). Clearly, the first letters of $f^i(u)$ and $f^j(u)$ are not equal to $u[1]$, but $f^i(u) = f^j(u)$ as $\mathbf{ord}(f)$ divides $|i-j|$. We get that $u \neq f^i(u) = f^j(u)$, for this choice of f. Finally, one can show by an analogous reasoning that the fact that $m \geq k_3$ (when $k_3 \neq +\infty$) means that for every word $u \in \Sigma_m^+$ there exists a morphism f such that $u = f^i(u) \neq f^j(u)$ and the fact that $m \geq k_4$ (when $k_4 \neq +\infty$) means that for every word $u \in \Sigma_m^+$ there exists a morphism f such that $f^i(u) \neq u = f^j(u)$.

Further, we show that if $m \geq \max\{k_1,k_2\}$ (in the case when $k_2 \neq +\infty$) there is no infinite word over Σ_m that avoids $x\pi^i(x)\pi^j(x)$. As $k_1 \geq 3$ it follows that $m \geq 3$. One can quickly check that the longest word without an instance of this pattern has length six and is 001010 by trying to construct such a word letter by letter. So there is no infinite word over Σ_m that avoids this pattern in this case.

By similar arguments, we show that if $m \geq \max\{k_1,k_3\}$ (in the case when $k_3 \neq +\infty$) there is no infinite word over Σ_m that avoids $x\pi^i(x)\pi^j(x)$. In this case, the longest word that avoids those patterns is 01010.

If $m \geq \max\{k_1,k_4\}$ (in the case when $k_4 \neq +\infty$) we also get that there is no infinite word over Σ_m that avoids $x\pi^i(x)\pi^j(x)$. The construction ends at length six, the longest words avoiding the pattern are 011001, 011002, 011221, 011223 and 011220.

These last remarks show that the pattern $x\pi^i(x)\pi^j(x)$ is unavoidable by infinite words over Σ_m, for all $m \geq k$. □

The next result represents the main step we take towards characterising the avoidability of cubes under morphic permutations.

Proposition 3.7 *Given the pattern $x\pi^i(x)\pi^j(x)$ we can determine effectively the values m, such that the pattern is avoidable in Σ_m.*

Proof. Since the case $m \geq k$ is handled in Lemma 3.6, we assume $m < k$.

The cases for $m = 2$ and $m = 3$ are depicted in Table 1. Note that an entry "✓" (resp. "×") at the intersection of line (i) and column (j, Σ_m) means that the pattern $xf^i(x)f^j(x)$ is avoidable (resp. unavoidable) in Σ_m. To build the table we used the results from Lemmas 3.3 to 3.5 and the fact that $x\pi^i(x)\pi^j(x)$ is avoidable in Σ_2 if and only if $i \equiv j \equiv 0(\mathit{mod}\ 2)$ (by the Thue-Morse word). Also, for Σ_3, when $j \neq 0$, every instance of the pattern contains cubes or squares, so it can be avoided by the infinite words t (seen as a word over three letters, that just does not contain one of the letters) or h, respectively. In the case when $j = 0$, we use the word defined in Lemma 3.2 to show the avoidability of the respective patterns.

We move on to the case $m \geq 4$. In this case, we split the discussion in several further cases, depending on the minimum of k_1, k_2, k_3, and k_4.

Case 1: $k_1 = \min\{k_1, k_2, k_3, k_4\}$. This means that $k > k_1$. If $m < k_1$ it follows that $m \mid i$ and $m \mid j$ (since $k_3, k_4 > k_1$). For every $a \in \Sigma_m$ and every permutation f of Σ_m, since $\mathbf{ord}_f(a) \leq m$ we get that $\mathbf{ord}_f(a) \mid i$ and $\mathbf{ord}_f(a) \mid j$. So an instance of $x\pi^i(x)\pi^j(x)$ is actually a cube, which is avoided by the Thue-Morse word. If $k_1 \leq m < k$, then for every $a \in \Sigma_m$ and every permutation f of Σ_m we either have that $\mathbf{ord}_f(a)$ divides both i and j or that $\mathbf{ord}_f(a)$ divides neither i nor j nor $|i-j|$. If we have a letter a occurring in a word u such that the latter holds, it means that there are at least three different letters in $uf^i(u)f^j(u)$. If no such letter exists in u, then $uf^i(u)f^j(u)$ is a cube. In both cases, the Thue-Morse word avoids the pattern $x\pi^i(x)\pi^j(x)$.

Case 2: $k_2 = \min\{k_1, k_2, k_3, k_4\}$. In this case, we get that $k = k_1$. If $4 \leq m < k_2$ we have for every $a \in \Sigma_m$ and every permutation f of Σ_m that $\mathbf{ord}_f(a) \mid i$ and $\mathbf{ord}_f(a) \mid j$ (since $k_3, k_4 > k_2$). So every instance of the pattern $x\pi^i(x)\pi^j(x)$ is a cube, which is avoided by the Thue-Morse word. If $k_2 \leq m < k$, we have for each $a \in \Sigma_m$ and every permutation f of Σ_m that either $\mathbf{ord}_f(a)$ divides at least one of i and j or $\mathbf{ord}_f(a) \mid |i-j|$. In all cases, this means that for each position l of a word u, we have that at least two of the letters $u[\ell], f^i(u)[\ell]$ and $f^j(u)[\ell]$ are equal, and the word defined in Lemma 3.2 avoids such patterns.

The cases $k_3 = \min\{k_1, k_2, k_3, k_4\}$ and $k_4 = \min\{k_1, k_2, k_3, k_4\}$ are analysed using the same methods and lead to similar results, so that we can conclude: a pattern of the form $x\pi^i(x)\pi^j(x)$ is always avoidable in Σ_m for all $4 \leq m < k$. Moreover, it might also be avoidable in Σ_2 and Σ_3, or only Σ_3 but not in Σ_2, or neither in Σ_2 nor in Σ_3 (according to Table 1). Therefore, for each pair (i, j) of natural numbers, we can effectively compute the values of m such that $x\pi^i(x)\pi^j(x)$ is avoidable in Σ_m. □

Further we show the following result, as a completion of the previous one.

Proposition 3.8 *Given the pattern $\pi^i(x)\pi^j(x)x$ we can determine effectively the values m, such that the pattern is avoidable in Σ_m.*

		j(mod 6)											
		0		1		2		3		4		5	
i(mod 6)	0	✓	✓	×	✓	✓	✓	×	✓	✓	✓	×	✓
	1	×	✓	×	✓	×	×	×	×	×	×	×	×
	2	✓	✓	×	×	✓	✓	×	×	✓	✓	×	✓
	3	×	✓	×	✓	×	×	×	✓	×	×	×	✓
	4	✓	✓	×	✓	✓	✓	×	×	✓	✓	×	×
	5	×	✓	×	×	×	×	×	×	×	×	×	✓
		Σ_2	Σ_3	Σ_2	Σ_3	Σ_2	Σ_3	Σ_2	Σ_3	Σ_2	Σ_3	Σ_2	Σ_3

Table 1: Avoidability of $x\pi^i(x)\pi^j(x)$ in Σ_2 and Σ_3

Proof. To check whether $\pi^i(x)\pi^j(x)x$ is avoidable in Σ_m or not, let $M = \max\{i+1, j+1, m\}$. Then $f^{M!}$ equals the identity for all permutations f of the alphabet Σ_m. Let us take $y = \pi^i(x)$. Since the functions that substitute π are permutations, we obtain that $\pi^i(x)\pi^j(x)x$ is avoidable in Σ_m if and only if $y\pi^{M!-i+j}(y)\pi^{M!-i}(y)$ is avoidable in Σ_m. Moreover, note that:

$$\begin{aligned}
\inf\{t : t \nmid j, t \nmid M!-i, t \nmid M!-i+j\} &= \inf\{t : t \nmid |i-j|, t \nmid i, t \nmid j\}\\
\inf\{t : t \mid j, t \nmid M!-i, t \nmid M!-i+j\} &= \inf\{t : t \nmid i, t \mid j\}\\
\inf\{t : t \mid M!-i, t \nmid M!-i+j\} &= \inf\{t : t \mid i, t \nmid j\}\\
\inf\{t : t \nmid M!-i, t \mid M!-i+j\} &= \inf\{t : t \mid |i-j|, t \nmid i, t \nmid j\}
\end{aligned}$$

Therefore, $y\pi^{M!-i+j}(y)\pi^{M!-i}(y)$ is avoidable in Σ_m if $4 \leq m < k$, where k is defined using (5) for i and j. □

In the exact same manner we get the following proposition.

Proposition 3.9 *Given the pattern $\pi^i(x)x\pi^j(x)$ we can determine effectively the values m, such that the pattern is avoidable in Σ_m.* □

We can now summarise our results in the following theorem:

Theorem 3.10 *Given the pattern $\pi^i(x)\pi^j(x)\pi^k(x)$, we can determine effectively the values m such that the pattern is avoidable in Σ_m.*

Proof. If $i = \min\{i, j, k\}$, let $y = \pi^i(x)$. The pattern becomes $y\pi^\ell(y)\pi^t(y)$, and we can identify all the alphabets where this pattern is avoidable by Proposition 3.7.

If $j = \min\{i, j, k\}$ we use Proposition 3.9 to identify all the alphabets where this pattern is avoidable. Finally, if $k = \min\{i, j, k\}$, we use Proposition 3.8 to identify all the alphabets where this pattern is avoidable. □

References

[1] B. BISCHOFF, J. CURRIE, D. NOWOTKA, Unary Patterns with Involution. *International Journal of Foundations of Computer Science* (2012). To appear.

[2] J. CASSAIGNE, *Algebraic Combinatorics on Words*, chapter Unavoidable Patterns. Cambridge University Press, Cambridge, UK, 2002, 111–134.

[3] E. CHINIFOROOSHAN, L. KARI, Z. XU, Pseudopower avoidance. *Fundamenta Informaticae* **114** (2012), 1–18.

[4] J. CURRIE, Pattern avoidance: themes and variations. *Theoret. Comput. Sci.* **339** (2005) 1, 7–18.

[5] M. HALL, *Lectures on Modern Mathematics*, 2, chapter Generators and relations in groups – The Burnside problem, Wiley, New York, 1964, 42–92.

[6] M. LOTHAIRE, *Combinatorics on Words*. Cambridge University Press, 1997.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 103 – 108

On Internal Contextual Grammars with Subregular Selection Languages

Florin Manea$^{(A)}$ Bianca Truthe$^{(B)}$

$^{(A)}$Christian-Albrechts-Universität zu Kiel, Institut für Informatik
Christian-Albrechts-Platz 4, D-24098 Kiel, Germany
flm@informatik.uni-kiel.de

$^{(B)}$Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik
PSF 4120, D-39016 Magdeburg, Germany
truthe@iws.cs.uni-magdeburg.de

Abstract

In this paper, we study the power of internal contextual grammars with selection languages from subfamilies of the family of regular languages. If we consider families $\mathcal{F}_n$ which are obtained by restriction to n states or nonterminals or productions or symbols to accept or to generate regular languages, we obtain four infinite hierarchies of the corresponding families of languages generated by internal contextual grammars with selection languages in $\mathcal{F}_n$.

1. Introduction

Contextual grammars were introduced by Solomon Marcus in [5] as a formal model that might be used in the generation of natural languages. The derivation steps consist in adding contexts to given well formed sentences, starting from an initial finite basis. Formally, a context is given by a pair (u, v) of words and the external adding to a word x gives the word uxv whereas the internal adding gives all words $x_1ux_2vx_3$ when $x = x_1x_2x_3$. Obviously, by linguistic motivation, a context can only be added if the words x or x_2 satisfy some given conditions. Thus, it is natural to define contextual grammars with selection in a certain family $\mathcal{F}$ of languages, where it is required that x or x_2 have to belong to a language of the family $\mathcal{F}$ which is associated with the context. Mostly, the family $\mathcal{F}$ is taken from the families of the Chomsky hierarchy (see [3, 7, 6], and the references therein).

By Jürgen Dassow, in [1], the study of external contextual grammars with selection in special regular sets was started. Finite, combinational, definite, nilpotent, regular suffix-closed, regular commutative languages and languages of the form V^* for some alphabet V were considered. In [2], Jürgen Dassow, Florin Manea, and Bianca Truthe continued the research and new results on the effect of regular commutative, regular circular, definite, regular

This paper was presented at DCFS 2012 (see [4]).

$^{(A)}$F. Manea's work is supported by the DFG grant 582014.

suffix-closed, ordered, combinational, nilpotent, and union-free selection languages on the generative power of external contextual grammars were obtained. Furthermore, families of regular languages which are defined by restrictions on the resources used to generate or to accept them were investigated. As measures, the number of states necessary to accept the regular languages and the number of nonterminals, production rules or symbols needed to generate the regular languages have been considered. In all these cases, infinite hierarchies were obtained.

In the present paper (which is a short version of the paper [4] presented at the conference DCFS 2012), we continue this line of research and investigate the effect of the number of resources (states, nonterminals, production rules, and symbols) on the generative power of internal contextual grammars. This case seems more complicated than the case of external contextual grammars, as there are two important differences between the way a derivation is conducted in internal grammars and in an external one. First, in the case of internal contextual grammars, the insertion of a context in a sentential form can be done in more than one place, so the derivation becomes, in a sense, non-deterministic; in the case of external grammars, once a context was selected there is at most one way to insert it: wrapped around the sentential form, when this word was in the selection language of the context. Second, if a context can be added internally, then it can be added arbitrarily often (because the subword where the context is wrapped around does not change) which does not necessarily hold for external grammars. However, we are able to obtain infinite hierarchies with respect to the descriptional complexity measures we use, but with different proof techniques.

2. Definitions

Throughout the paper, we assume that the reader is familiar with the basic concepts of the theory of automata and formal languages. For details, we refer to the Handbook of Formal Languages by Grzegorz Rozenberg and Arto Salomaa ([7]). Here we only recall some notation and the definition of contextual grammars with selection which form the central notion of the paper.

Given an alphabet V, we denote by V^* and V^+ the set of all words and the set of all non-empty words over V, respectively. The empty word is denoted by λ. For a word $w \in V^*$ and a letter $a \in V$, by $|w|$ and $\#_a(w)$ we denote the length of w and the number of occurrences of a in w, respectively. The cardinality of a set A is denoted by $\#(A)$.

Let $G = (N,T,P,S)$ be a regular grammar (specified by finite sets N and T of nonterminals and terminals, respectively, a finite set of productions of the form $A \to wB$ or $A \to w$ with $A,B \in N$ and $w \in T^*$ as well as $S \in N$). Further, let $A = (X,Z,z_0,F,\delta)$ be a deterministic finite automaton (specified by sets X and Z of input symbols and states, respectively, an initial state z_0, a set F of accepting states, and a transition function δ) and L be a regular language. Then we define

$$\begin{aligned} \mathit{State}(A) &= \#(Z), \\ \mathit{Var}(G) &= \#(N),\ \mathit{Prod}(G) = \#(P),\ \mathit{Symb}(G) = \sum_{A\to w\in P}(|w|+2), \\ \mathit{State}(L) &= \min\{\,\mathit{State}(A) \mid A \text{ is a det. finite automaton accepting } L\,\}, \\ K(L) &= \min\{\,K(G) \mid G \text{ is a reg. grammar for } L\,\}\,(K \in \{\mathit{Var},\mathit{Prod},\mathit{Symb}\}), \end{aligned}$$

and, for $K \in \{\,\mathit{State},\mathit{Var},\mathit{Prod},\mathit{Symb}\,\}$, we set

$$\mathit{REG}_n^K = \{\,L \mid L \text{ is a regular language with } K(L) \le n\,\}.$$

Remark. We note that if we restricted ourselves to rules of the form $A \to aB$ and $A \to \lambda$ with $A,B \in N$ and $a \in T$, then we would have $\mathit{State}(L) = \mathit{Var}(L)$.

We now introduce the central notion of this paper.

Let $\mathcal{F}$ be a family of languages. A *contextual grammar with selection in* $\mathcal{F}$ is a triple

$$G = (V, \{\,(S_1,C_1),(S_2,C_2),\dots,(S_n,C_n)\,\}, B)$$

where

- V is an alphabet,
- for $1 \le i \le n$, S_i is a language over V in $\mathcal{F}$ and C_i is a finite set of pairs (u,v) with $u \in V^*$, $v \in V^*$,
- B is a finite subset of V^*.

The set V is called the basic alphabet; the languages S_i and the sets C_i, $1 \le i \le n$, are called the *selection languages* and the sets of *contexts* of G, respectively; the elements of B are called *axioms*.

We now define the internal derivation for contextual grammars with selection.

Let $G = (V, \{\,(S_1,C_1),(S_2,C_2),\dots,(S_n,C_n)\,\}, B)$ be a contextual grammar with selection. A direct *internal derivation step* in G is defined as follows: a word x derives a word y (written as $x \Longrightarrow y$) if and only if there are words x_1, x_2, x_3 with $x_1x_2x_3 = x$ and there is an integer i, $1 \le i \le n$, such that $x_2 \in S_i$ and $y = x_1ux_2vx_3$ for some pair $(u,v) \in C_i$. Intuitively, we can only wrap a context $(u,v) \in C_i$ around a subword x_2 of x if x_2 belongs to the corresponding language S_i. We call a word of a selection language useful, if it is a subword of a word of the generated language – if it is really selected from wrapping a context around it.

By $\Longrightarrow^*$ we denote the reflexive and transitive closure of $\Longrightarrow$. The *internal language generated by* G is defined as

$$L(G) = \{\,z \mid x \Longrightarrow^* z \text{ for some } x \in B\,\}.$$

By $\mathcal{L}(\mathit{IC},\mathcal{F})$ we denote the family of all internal languages generated by contextual grammars with selection in $\mathcal{F}$. When we speak about contextual grammars in this paper, we mean contextual grammars with internal derivation (also called internal contextual grammars).

Example 2.1 *Let $n \geq 1$ and $V = \{a\}$ be a unary alphabet. We set*

$$B_n = \{ a^i \mid 1 \leq i \leq n \}, \ U_n = \{ a^n \}^+, \text{ and } L_n = B_n \cup U_n.$$

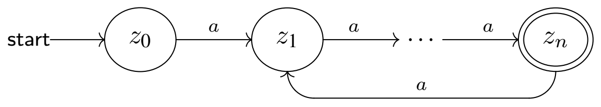
The contextual grammar $G_n = (V, \{ (U_n, \{(\lambda, a^n)\}) \}, B_n)$ generates the language L_n. This can be seen as follows. The context a^n can be added to a word w if and only if w contains at least n letters. The only axiom to which a context can be added is a^n. From this, we get the unique derivation

$$a^n \Longrightarrow a^{2n} \Longrightarrow a^{3n} \Longrightarrow \cdots.$$

It is easy to see that the set U_n is accepted by the automaton

$$(V, \{z_0, z_1, \ldots, z_n\}, z_0, \{z_n\}, \delta_n)$$

where the graph



represents the transition function δ_n.

Hence, we have $L_n \in \mathcal{L}(IC, REG_{n+1}^{State})$. ◇

3. Results

The language $L_1 = \{ \lambda \} \cup \{ w \mid w \in \{a,b\}^+, \#_b(w) = 1 \}$ and the languages L_n for $n \geq 2$ given in Example 2.1 are witnesses for the proper inclusions of the hierarchy with respect to the number of states.

Theorem 3.1 *For any natural number $n \geq 1$, we have the proper inclusion*

$$\mathcal{L}(IC, REG_n^{State}) \subset \mathcal{L}(IC, REG_{n+1}^{State}).$$

Also with respect to the number of nonterminal symbols, we obtain an infinite hierarchy.

Theorem 3.2 *For any natural number $n \geq 0$, we have the proper inclusion*

$$\mathcal{L}(IC, REG_n^{Var}) \subset \mathcal{L}(IC, REG_{n+1}^{Var}).$$

For these proper inclusions, the languages

$$L_n = \{ a^{p_1} b a^{p_2} b \ldots a^{p_n} b a^{p_1} b a^{p_2} b \ldots a^{p_n} b \mid p_i \geq 1, \ 1 \leq i \leq n \}$$

for $n \geq 1$ are witnesses.

As consequences from the previous theorem, we obtain also infinite hierarchies with respect to the number of production rules and the number of symbols. However, the properness of the inclusions $\mathcal{L}(IC, REG_n^K) \subseteq \mathcal{L}(IC, REG_{n+1}^K)$ with $K \in \{Prod, Symb\}$ and $n \geq n_0$ for some start number n_0 does not immediately follow. For the complexity measure *Prod*, we consider a generalization of the languages L_n for $n \geq 1$ used in the proof of the previous theorem.

Let $m \geq 1$, $A_m = \{ a_1, \ldots, a_m \}$, and $V = A_m \cup \{b\}$. The languages L_n consist again of words formed by $2n$ a-blocks ended by the letter b each and where the block lengths coincide in a crossed agreement manner. However, an a-block now consists of letters from the set A_m instead of the single letter a only.

Formally, we define for $n \geq 1$ the languages

$$L_n^{(m)} = \{ w_1 b w_2 b \ldots w_n b w_{n+1} b w_{n+2} b \ldots w_{2n} b \mid \\ w_i, w_{n+i} \in A_m^+,\ |w_i| = |w_{n+i}|,\ 1 \leq i \leq n \}.$$

For these languages, we obtain

$$L_n^{(m)} = \mathcal{L}(IC, REG^{Prod}_{(m+1)(n-1)+1}) \setminus \mathcal{L}(IC, REG^{Prod}_{(m+1)(n-1)}).$$

This result leads to the inclusion

$$\mathcal{L}(IC, REG^{Prod}_{k-1}) \subset \mathcal{L}(IC, REG^{Prod}_{k})$$

for $k \geq 3$, if we set $n = 2$ and $m = k - 2$.

For $k = 1$, the proper inclusion holds because the family $\mathcal{L}(IC, REG^{Prod}_0)$ only contains finite languages whereas the infinite language $L_1^{(m)}$ belongs to the family $\mathcal{L}(IC, REG^{Prod}_1)$.

For $k = 2$, the properness of the inclusion can be shown with the witness language

$$L = \left\{ w_{ab} c d^n e^m \mid m \geq 0, n \geq 0, w_{ab} \in \{ a, b \}^*, \#_a(w_{ab}) = n, \#_b(w_{ab}) = m \right\}.$$

Together, we obtain the following result.

Theorem 3.3 *The relation*

$$\mathcal{L}(IC, REG^{Prod}_n) \subset \mathcal{L}(IC, REG^{Prod}_{n+1})$$

holds for every natural number $n \geq 0$.

We now consider the complexity measure *Symb*. Both the families $\mathcal{L}(IC, REG^{Symb}_0)$ and $\mathcal{L}(IC, REG^{Symb}_1)$ are equal to the class of finite languages (every selection language is the empty set; the language generated coincides with the set of axioms). The language family $\mathcal{L}(IC, REG^{Symb}_2)$ contains infinite languages; for instance, the language $L = \{a\}^*$ is generated by the contextual grammar $G = (\{ a \}, \{ (\{\lambda\}, \{(\lambda, a)\}) \}, \{ \lambda \})$. This yields the proper inclusion

$$\mathcal{L}(IC, REG^{Symb}_1) \subset \mathcal{L}(IC, REG^{Symb}_2).$$

Also the further inclusions are proper, as can be proven with the languages

$$L_n = \{ a^m c_1 c_2 \ldots c_{n-1} b^m \mid m \geq 0 \}$$

for $n \geq 2$.

This leads to the following result.

Theorem 3.4 *We have the relations*

$$\mathcal{L}(IC, REG_0^{Symb}) = \mathcal{L}(IC, REG_1^{Symb}) = FIN$$

and

$$\mathcal{L}(IC, REG_n^{Symb}) \subset \mathcal{L}(IC, REG_{n+1}^{Symb})$$

for every natural number $n \geq 1$.

It remains open for future research to consider other families of subregular languages, defined by restrictions of combinatorial nature, like the ones studied in [1, 2] for external contextual grammars.

References

[1] J. DASSOW, Contextual grammars with subregular choice. *Fundamenta Informaticae* **64** (2005) 1–4, 109–118.

[2] J. DASSOW, F. MANEA, B. TRUTHE, On Contextual Grammars with Subregular Selection Languages. In: M. HOLZER, M. KUTRIB, G. PIGHIZZINI (eds.), *Descriptional Complexity of Formal Systems – 13th International Workshop, DCFS 2011, Gießen/Limburg, Germany, July 25 – 27, 2011. Proceedings*. LNCS 6808, Springer-Verlag, 2011, 135–146.

[3] S. ISTRAIL, Gramatici contextuale cu selectiva regulata. *Stud. Cerc. Mat.* **30** (1978), 287–294.

[4] F. MANEA, B. TRUTHE, On Internal Contextual Grammars with Subregular Selection Languages. In: M. KUTRIB, N. MOREIRA, R. REIS (eds.), *Descriptional Complexity of Formal Systems – 14th International Workshop, DCFS 2012, Braga, Portugal, July 23 – 25, 2012. Proceedings*. LNCS 7386, Springer-Verlag, 2012, 222–235.

[5] S. MARCUS, Contextual grammars. *Revue Roum. Math. Pures Appl.* **14** (1969), 1525–1534.

[6] G. PĂUN, *Marcus Contextual Grammars*. Kluwer Publ. House, Doordrecht, 1998.

[7] G. ROZENBERG, A. SALOMAA (eds.), *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 109 – 114

Limited Context Restarting Automata and McNaughton Families of Languages

Friedrich Otto$^{(A)}$ Peter Černo$^{(B)}$ František Mráz$^{(B)}$

$^{(A)}$Fachbereich Elektrotechnik/Informatik, Universität Kassel
34109 Kassel, Germany
otto@theory.informatik.uni-kassel.de

$^{(B)}$Charles University, Faculty of Mathematics and Physics
Department of Computer Science, Malostranské nám. 25
118 00 Praha 1, Czech Republic
petercerno@gmail.com,mraz@ksvi.ms.mff.cuni.cz

Abstract

In the literature various types of restarting automata have been studied that are based on contextual rewriting. A word w is accepted by such an automaton if, starting from the initial configuration that corresponds to input w, the word w is reduced to the empty word within a finite number of applications of these contextual rewritings. This approach is reminiscent of the notion of McNaughton families of languages. Here we put the aforementioned types of restarting automata into the context of McNaughton families of languages, relating the classes of languages accepted by these automata in particular to the class GCSL of growing context-sensitive languages and to the class CRL of Church-Rosser languages.

1. Introduction

Restarting automata have been introduced to model the linguistic technique of *analysis by reduction* [8]. By now many different types of restarting automata have been defined and studied intensively, see for example [13]. The deterministic context-free languages, the context-free languages, the Church-Rosser languages and the growing context-sensitive languages have all been characterized by certain types of restarting automata.

In [5] the so-called *clearing restarting automaton* was introduced. While in general a restarting automaton scans the tape contents from left to right until it detects a position to which a rewrite operation applies, the rewriting done by a clearing restarting automaton only depends on the context of a fixed size around the subword to be rewritten. In fact, a clearing restarting automaton can only delete symbols. For these automata a simple learning algorithm exists, but not surprisingly, clearing restarting automata are quite limited in their expressive power. They accept all regular languages and even some languages that are not context-free,

$^{(B)}$The second and the third author were supported by the Grant Agency of the Czech Republic under the projects P103/10/0783 and P202/10/1333 and by the Grant Agency of Charles University under project 272111/A-INF/MFF.

but they do not even accept all context-free languages. Accordingly, they were extended to the so-called Δ-*clearing restarting automata* and the Δ*-*clearing restarting automata* that can use a marker symbol Δ in their rewrite operations. It turned out that these types of restarting automata can accept all context-free languages [6].

In [2] *limited context restarting automata* were defined that can be seen as an extension of the clearing restarting automaton. These automata just apply rewrite steps based on context information, but their rewrite rules are more general. In fact, the most general form of these automata accepts exactly the growing context-sensitive languages. In [2] a special version of a genetic algorithm is proposed to learn these automata from positive and negative samples.

As a limited context restarting automaton applies its rewrite operations based on context information, it can be interpreted as executing reductions with respect to a finite string-rewriting system. This is essentially the same concept as the one that underlies the notion of *McNaughton families* of languages studied in [3]. Accordingly, it is natural to investigate the correspondence between the various types of limited context restarting automata on the one hand and the McNaughton families of languages of [3] on the other hand. Here we compare the families of languages that are accepted by the various types of limited context restarting automata to the class GCSL of *growing context-sensitive languages* [4, 7] and to the class CRL of *Church-Rosser languages* [10].

Notation. In the following all alphabets considered will be finite. For an alphabet Σ, Σ^* is used to denote the set of all words over Σ including the empty word λ. For $w \in \Sigma^*$, $|w|$ denotes the length of w. By $\mathbb{N}$ we denote the set of non-negative integers. A *weight function* is a mapping $g : \Sigma \to \mathbb{N}$ that assigns a positive weight $g(a)$ to each letter a of Σ. It is extended to arbitrary words by taking $g(\lambda) = 0$ and $g(wa) = g(w) + g(a)$ for all words $w \in \Sigma^*$ and all $a \in \Sigma$. Finally, for any type A of automaton, $\mathcal{L}(\mathsf{A})$ is used to denote the class of languages accepted by automata of this type.

2. Basic Definitions and First Results

A *string-rewriting system* S on Σ consists of (finitely many) pairs of strings from Σ^*, called *rewrite rules*, which are written as $(\ell \to r)$. By $\mathrm{dom}(S)$ we denote the set $\mathrm{dom}(S) = \{\, \ell \mid \exists r \in \Sigma^* : (\ell \to r) \in S \,\}$ of left-hand sides of rules of S. The *reduction relation* $\Rightarrow_S^*$ on Σ^* that is induced by S is the reflexive and transitive closure of the *single-step reduction relation* $\Rightarrow_S = \{\, (u\ell v, urv) \mid (\ell \to r) \in S, u, v \in \Sigma^* \,\}$. For a string $u \in \Sigma^*$, if there exists a string v such that $u \Rightarrow_S v$ holds, then u is called *reducible* modulo S. Otherwise, u is called *irreducible* modulo S. $\mathrm{IRR}(S)$ denotes the set of all irreducible strings modulo S.

Next we come to the notion of McNaughton families of languages. A language $L \subseteq \Sigma^*$ is called a *McNaughton language*, if there exist a finite alphabet Γ strictly containing Σ, a finite string-rewriting system S on Γ, strings $t_1, t_2 \in (\Gamma \smallsetminus \Sigma)^* \cap \mathrm{IRR}(S)$, and a letter $Y \in (\Gamma \smallsetminus \Sigma) \cap \mathrm{IRR}(S)$ such that, for all $w \in \Sigma^*$, $w \in L$ if and only if $t_1 w t_2 \Rightarrow_S^* Y$. Here the symbols of Σ are *terminals*, while those of $\Gamma \smallsetminus \Sigma$ can be seen as *nonterminals*. We say that the McNaughton language L is *specified* by the four-tuple (S, t_1, t_2, Y). This fact will be expressed as $L = L(S, t_1, t_2, Y)$. By placing restrictions on the finite string-rewriting systems used we obtain certain families of McNaughton languages [3].

The Δ*-clearing restarting automata were introduced together with the clearing restarting automata and the Δ-clearing restarting automata by two of the authors in [5]. Here we establish an upper bound by showing that these types of restarting automata only accept languages that are growing context-sensitive.

Definition 2.1

(a) *Let k be a nonnegative integer. A k-*context rewriting system *is a system $M = (\Sigma, \Gamma, I)$, where Σ is a finite input alphabet, Γ is a finite working alphabet containing Σ, but not the special symbols ¢ and \$, called sentinels, and I is a finite set of instructions of the form $(x, z \to t, y)$, where $x \in \{\lambda, ¢\} \cdot \Gamma^*$, $|x| \leq k$, is called a* left context, *$y \in \Gamma^* \cdot \{\lambda, \$\}$, $|y| \leq k$, is called a* right context, *and $z \to t$ is called a* rule, *where $z, t \in \Gamma^*$, $z \neq t$. If we do not specify k (e.g. if we talk about context rewriting systems) we have no upper bound on the contexts, which is equivalent to $k = +\infty$.*

A word $w = uzv$ can be rewritten into utv, denoted as $uzv \vdash_M utv$, if and only if there exists an instruction $i = (x, z \to t, y) \in I$ such that x is a suffix of $¢ \cdot u$ and y is a prefix of $v \cdot \$$. The language *associated with M is defined as $L(R) = \{w \in \Sigma^* \mid w \vdash_M^* \lambda\}$, where $\vdash_M^*$ denotes the reflexive and transitive closure of $\vdash_M$.*

(b) *A k-*clearing restarting automaton *(k-cl-RA) is a k-context rewriting system $M = (\Sigma, \Sigma, I)$ such that, for each instruction $i = (x, z \to t, y) \in I$, we have $z \in \Sigma^+$ and $t = \lambda$.*

(c) *A k-Δ-*clearing restarting automaton *(k-Δ-cl-RA) is a k-context rewriting system $M = (\Sigma, \Gamma, I)$ such that $\Gamma = \Sigma \cup \{\Delta\}$, where Δ is a new symbol, and for each instruction $i = (x, z \to t, y) \in I$, we have $z \in \Gamma^+$ and $t \in \{\lambda, \Delta\}$.*

(d) *A k-Δ^*-*clearing restarting automaton *(k-Δ^*-cl-RA) is a k-context rewriting system $M = (\Sigma, \Gamma, I)$ such that $\Gamma = \Sigma \cup \{\Delta\}$, where Δ is a new symbol, and for each instruction $i = (x, z \to t, y) \in I$, we have $z \in \Gamma^+$ and $t = \Delta^i$ for some i satisfying $0 \leq i \leq |z|$.*

Observe that $\lambda \in L(M)$ holds for each context rewriting system M. In order to simplify the discussion we will call two languages $L_1, L_2 \subseteq \Sigma^*$ *equal* if $L_1 \cap \Sigma^+ = L_2 \cap \Sigma^+$.

Theorem 2.2 $\mathcal{L}(\Delta^*\text{-cl-RA}) \subseteq \text{GCSL}$.

It remains open whether Δ^*-clearing restarting automata accept all growing context-sensitive languages.

3. Limited Context Restarting Automata

The *limited context restarting automaton*, abbreviated as lc-R-automaton, was introduced in [2] as a generalization of the clearing restarting automaton. Here we introduce a slightly generalized version which uses weight-reducing rules instead of length-reducing ones.

Definition 3.1 *A* limited context restarting automaton *is a context rewriting system $M = (\Sigma, \Gamma, I)$, such that for all $(x, z \to t, y) \in I : g(x) > g(y)$ for some weight function $g : \Sigma^* \to \mathbb{N}$. We use the notation $(x \mid z \to t \mid y)$ for instructions.*

Example 3.2 *Let* $M = (\{a,b,c\},\{a,b,c\},I)$*, where* $I = \{(\lambda \mid acbb \to c \mid \lambda), (¢ \mid c \to \lambda \mid \$)\}$*, be an* lc-R*-automaton. Then* $aaacbbbbbb \vdash_M aacbbbb \vdash_M acbb \vdash_M c \vdash_M \lambda$*, and so the word* a^3cb^6 *belongs to* $L(M)$*. It is easily seen that* $L(M) = \{a^n cb^{2n} \mid n \geq 0\}$*.*

We consider several restricted types of lc-R-automata. Recall from Definition 3.1 that all instructions of an lc-R-automaton are necessarily weight-reducing. We say that an lc-R-automaton $M = (\Sigma,\Gamma,I)$ is of type

- $\mathcal{R}'_0$, if I is an arbitrary finite set of (weight-reducing) instructions;
- $\mathcal{R}'_1$, if for all $(u|x \to y|v) \in I$: $y \in \Gamma \cup \{\lambda\}$, and $x \in \Gamma^+$;
- $\mathcal{R}'_2$, if for all $(u|x \to y|v) \in I$: $y \in \Gamma \cup \{\lambda\}$, $u \in \{\lambda, ¢\}$, $v \in \{\lambda, \$\}$, and $x \in \Gamma^+$;
- $\mathcal{R}'_3$, if for all $(u|x \to y|v) \in I$: $y \in \Gamma \cup \{\lambda\}$, $u \in \{\lambda, ¢\}$, $v = \$$, and $x \in \Gamma^+$.

We say that an lc-R-automaton $M = (\Sigma,\Gamma,I)$ is of type $\mathcal{R}_i$, if it is of type $\mathcal{R}'_i$ and, in addition, M is *length reducing*, i.e. for all $(u|x \to y|v) \in I$: $|x| > |y|$, for all $i \in \{0,1,2,3\}$.

In what follows, lc-R-automata of type $\mathcal{R}$, for $\mathcal{R} \in \{\mathcal{R}_0, \mathcal{R}'_0, \mathcal{R}_1, \mathcal{R}'_1, \mathcal{R}_2, \mathcal{R}'_2, \mathcal{R}_3, \mathcal{R}'_3\}$, will be referred to as lc-R[$\mathcal{R}$]-automata. In [1], Basovník studied the power of length-reducing lc-R-automata. Here we complete his results by also studying the other types of lc-R-automata.

Theorem 3.3 $\mathcal{L}(\text{lc-R}[\mathcal{R}'_0]) = \mathcal{L}(\text{lc-R}[\mathcal{R}_0]) = \text{GCSL}$.

Theorem 3.4 $\mathcal{L}(\text{lc-R}[\mathcal{R}'_1]) = \text{GCSL}$.

Theorem 3.5 $\text{GACSL} \subseteq \mathcal{L}(\text{lc-R}[\mathcal{R}_1]) \subseteq \text{GCSL}$*, that is, the class of languages that are accepted by* lc-R[$\mathcal{R}_1$]*-automata lies between the class* GACSL *of* growing acyclic context-sensitive languages *[11] and the class* GCSL *of growing context-sensitive languages.*

Theorem 3.6 $\mathcal{L}(\text{lc-R}[\mathcal{R}'_2]) = \mathcal{L}(\text{lc-R}[\mathcal{R}_2]) = \text{CFL}$.

Theorem 3.7 $\mathcal{L}(\text{lc-R}[\mathcal{R}'_3]) = \mathcal{L}(\text{lc-R}[\mathcal{R}_3]) = \text{REG}$.

4. Confluent Limited Context Restarting Automata

As defined in Definition 3.1, an lc-R-automaton $M = (\Sigma,\Gamma,I)$ is a nondeterministic device. This phenomenon complicates the problem of deciding membership in $L(M)$. Here we are interested in lc-R-automata for which all computations from $¢w\$$ lead to $¢\$$, if $w \in L(M)$. Unfortunately, it is undecidable in general whether a finite string-rewriting system is confluent on a given congruence class, even if the given finite system only contains length-reducing rules [12]. Therefore, we turn to lc-R-automata that are even further restricted.

Definition 4.1 *An* lc-R*-automaton* $M = (\Sigma,\Gamma,I)$ *is called* confluent *if the corresponding string-rewriting system* $R(M) = \{\, uxv \to uyv \mid (u \mid x \to y \mid v) \in I \,\}$ *is confluent.*

We will use the prefix con- to denote types of confluent lc-R-automata. Further, for each type $\mathcal{R} \in \{\mathcal{R}'_i, \mathcal{R}_i \mid i \in \{0,1,2,3\}\}$, lc-R[con-$\mathcal{R}$] will denote the class of lc-R-automata of type $\mathcal{R}$ that are confluent.

Theorem 4.2 $\mathcal{L}(\text{lc-R[con-}\mathcal{R}'_0])= \mathcal{L}(\text{lc-R[con-}\mathcal{R}_0]) = \mathsf{CRL}$.

Theorem 4.3 $\mathcal{L}(\text{lc-R[con-}\mathcal{R}'_3])= \mathcal{L}(\text{lc-R[con-}\mathcal{R}_3]) = \mathsf{REG}$.

For the classes of languages that are accepted by confluent lc-R-automata of types $\mathcal{R}'_1$, $\mathcal{R}_1$, $\mathcal{R}'_2$ or $\mathcal{R}_2$, we have no characterization results yet. However, we have some preliminary results concerning the latter classes.

Lemma 4.4 *If L is accepted by an* lc-R[con-$\mathcal{R}'_2$]*-automaton, then $L \in \mathsf{DCFL} \cap \mathsf{DCFL}^R$.*

Lemma 4.5 *The deterministic context-free language $L_u = \{\, ca^nb^nc \mid n \geq 1\,\} \cup \{\, da^mb^{2m}d \mid m \geq 1\,\}$ is not accepted by any* lc-R[con-$\mathcal{R}'_2$]*-automaton.*

Corollary 4.6
(a) *The class of languages that are accepted by* lc-R[con-$\mathcal{R}'_2$]*-automata is properly contained in the class* symDCFL.
(b) *The class of languages that are accepted by* lc-R[con-$\mathcal{R}'_2$]*-automata is incomparable to the language classes* DLIN *and* LIN.

These results also hold for the class of languages that are accepted by lc-R[con-$\mathcal{R}_2$]-automata. However, the exact relationship of these classes of languages to the class of confluent monadic McNaughton languages and to the class of confluent generalized monadic McNaughton languages of [9] remains currently open.

Finally we shortly consider lc-R[con-$\mathcal{R}_1$]-automata.

Lemma 4.7 *The language $L_{expo5} = \{\, a^{5^n} \mid n \geq 0\,\}$ is accepted by an* lc-R[con-$\mathcal{R}_1$]*-automaton.*

Corollary 4.8 *The class of languages that are accepted by* lc-R[con-$\mathcal{R}_1$]*-automata is incomparable to the class* CFL *under inclusion. In particular, it properly includes the class of languages that are accepted by* lc-R[con-$\mathcal{R}_2$]*-automata.*

5. Concluding Remarks

We have studied the relationship between various classes of limited context restarting automata on the one hand and certain McNaughton families of languages on the other hand. We have seen that the class GCSL of growing context-sensitive languages is an upper bound for all the types of limited context restarting automata considered, and that this upper bound is attained by three classes of these automata. Under the additional requirement of confluence, the Church-Rosser languages form an upper bound, which is reached by the two most general types of these automata. On the other hand, for the most restricted types of lc-R-automata, we just obtain the regular languages, both in the confluent and the non-confluent case. However, for the intermediate systems, the question for an exact characterization of the classes of languages accepted remains open.

References

[1] S. BASOVNÍK, *Learning restricted restarting automata using genetic algorithm*. Master's thesis, Charles University, Faculty of Mathematics and Physics, Prague, 2010.

[2] S. BASOVNÍK, F. MRÁZ, Learning limited context restarting automata by genetic algorithms. In: J. DASSOW, B. TRUTHE (eds.), *Theorietag 2011*. Otto-von-Guericke-Universität, Magdeburg, 2011, 1–4.

[3] M. BEAUDRY, M. HOLZER, G. NIEMANN, F. OTTO, McNaughton families of languages. Theoret. Comput. Sci. **290** (2003), 1581–1628.

[4] G. BUNTROCK, F. OTTO, Growing context-sensitive languages and Church-Rosser languages. Inform. and Comput. **141** (1998), 1–36.

[5] P. ČERNO, F. MRÁZ, Clearing restarting automata. Fund. Inf. **104** (2010), 17–54.

[6] P. ČERNO, F. MRÁZ, Δ-clearing restarting automata and CFL. In: G. MAURI, A. LEPORATI (eds.), *DLT 2011*. *LNCS 6795*, Springer, Berlin, 2011, 153–164.

[7] E. DAHLHAUS, M. WARMUTH, Membership for growing context-sensitive grammars is polynomial. J. Comput. System Sci. **33** (1986), 456–472.

[8] P. JANČAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, Restarting automata. In: H. REICHEL (ed.), *FCT'95*. *LNCS 965*, Springer, Berlin, 1995, 283–292.

[9] P. LEUPOLD, F. OTTO, On McNaughton families of languages that are specified by some variants of monadic string-rewriting systems. Fund. Inf. **112** (2011), 219–238.

[10] R. MCNAUGHTON, P. NARENDRAN, F. OTTO, Church-Rosser Thue systems and formal languages. J. Assoc. Comput. Mach. **35** (1988), 324–344.

[11] G. NIEMANN, J. WOINOWSKI, The growing context-sensitive languages are the acyclic context-sensitive languages. In: W. KUICH, G. ROZENBERG, A. SALOMAA (eds.), *DLT 2002*. *LNCS 2295*, Springer, Berlin, 2002, 197–205.

[12] F. OTTO, On deciding the confluence of a finite string-rewriting system on a given congruence class. J. Comput. System Sci. **35** (1987), 285–310.

[13] F. OTTO, Restarting automata. In: Z. ÉSIK, C. MARTIN-VIDE, V. MITRANA (eds.), *Recent Advances in Formal Languages and Applications*. Studies in Computational Intelligence 25, Springer, Berlin, 2006, 269–303.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 115 – 120

On Localization of (Post)Prefix (In)Consistencies$^{(A)}$

Martin Procházka Martin Plátek

Charles University, MFF UK, Department of Computer Science,
Malostranské náměstí 25, 118 00 Praha 1, Czech Republic
martproc@gmail.com, martin.platek@mff.cuni.cz

1. Introduction

A reducing automaton (red-automaton) is a deterministic automaton proposed for checking word and sub-word correctness by the use of analysis by reduction, see [3, 4, 5]. Its monotone version characterizes the class of deterministic context-free languages (DCFL). We propose a method for a construction of a deterministic push-down transducer for any monotone reducing automaton which is able with the help of special output symbols to localize its prefix and post-prefix (in)consistencies, and certain types of reducing conflicts.

2. Definitions and Basic Properties

The *reducing automaton* is a refinement of R-automaton from [2]. It has a finite *control unit* and a *working head* attached to a list with sentinels on both ends. It works in certain cycles called *stages*. At the beginning of each stage, the head points at the leftmost item behind the left sentinel, and the control unit is in a special *initial state*. In the process of the stage the automaton moves the head from the item it currently points to the next item on the right. During such a *transition* it changes the state of its control unit according to the current state and the currently scanned symbol. The stage ends as the control unit gets to any of special states called *operations*. There are three kinds of operations: ACC, ERR, and RED. Both ACC and ERR-operation halts the whole computation, ACC accepts and ERR rejects the word in the list. The RED-operation RED(n) determines how the list should be shortened. Its parameter n – a binary word of a limited size – specifies which item on the left of the head are to be removed from the list. Bit 1 means "remove the item from the list", bit 0 means "leave the item in the list". After all items designated for deletion are removed, the automaton resets its control unit to the initial state and places the head at the leftmost item in the list. The string $n \in (\mathtt{10}^*)^+$ determines, which items will be deleted from the list. If the i-th symbol of n from the right is equal to 1, then the automaton deletes the i-th item to the left from the position of the head. The item scanned by the head is considered as the first one.

All final states of a reducing automaton M create a finite subset F_M of the (unbounded) set $\{\mathtt{ACC}, \mathtt{ERR}\} \cup \{\mathtt{RED}(n) \mid n \in (\mathtt{1}\cdot\mathtt{0}^*)^+\}$. Now we are able to introduce reducing automata in a formal way.

$^{(A)}$This work was supported by the grant projects of the Grant Agency of the Czech Republic No. P202/10/1333 and P103/10/0783.

A *reducing automaton (red-automaton)* is a 7-tuple $M = (\Sigma_M, «, », S_M, s_M, F_M, f_M)$, where Σ_M is a finite input alphabet, $«,» \notin \Sigma_M$ are the (left and right) sentinels, S_M is the finite set of internal states, $s_M \in S_M$ is the (re)starting state, F_M is the finite set of final states (operations), $f_M : S_M \times (\Sigma_M \cup \{»\}) \longrightarrow (S_M \cup F_M)$ is the *transition function* of M, which fulfills the following condition: $\forall s \in S_M : f_M(s, ») \in F_M$.

We will describe the behavior of M in more details by two functions enhancing the transition function f_M:
$\delta_M : (S_M \cup F_M \cup \{\texttt{RED}\}) \times (\Sigma_M \cup \{«,»\}) \longrightarrow (S_M \cup F_M \cup \{\texttt{RED}\})$
$\Delta_M : (S_M \cup F^*_M) \times (\Sigma_M \cup \{«,»\}) \longrightarrow (S_M \cup F^*_M)$

RED is a new (helping) state which is different from all states from S_M, and the set F^*_M is defined in the following way: $F^*_M = F_M \cup \{\texttt{RED}(n \cdot \texttt{0}^k) \mid \texttt{RED}(n) \in F_M \text{ a } k \geq 1\}$

Both functions δ_M, Δ_M for all pairs created by a state $s \in S_M$ and by a symbol $a \in (\Sigma_M \cup \{»\}$ are equal to the function f_M. We define the new functions for the remaining relevant pairs in the following way:

$$\begin{aligned} \delta_M(s,«) &= s_M & \Delta_M(s,«) &= s_M \\ \delta_M(\texttt{ACC},a) &= \texttt{ACC} & \Delta_M(\texttt{ACC},a) &= \texttt{ACC} \\ \delta_M(\texttt{ERR},a) &= \texttt{ERR} & \Delta_M(\texttt{ERR},a) &= \texttt{ERR} \\ \delta_M(\texttt{RED}(n),a) &= \texttt{RED} & \Delta_M(\texttt{RED}(n),a) &= \texttt{RED}(n\cdot\texttt{0}) \\ \delta_M(\texttt{RED},a) &= \texttt{RED} \end{aligned}$$

The first enhancement of δ_M: $\delta^*_M(s,\lambda) = s, \quad \delta^*_M(s,ua) = \delta_M(\delta^*_M(s,u),a)$
The first enhancement of Δ_M: $\Delta^*_M(s,\lambda) = s, \quad \Delta^*_M(s,ua) = \Delta_M(\Delta^*_M(s,u),a)$
We will often use the following conventions: $\delta^*_M(s_M,w) = \delta^*_M(«w)$, $\Delta^*_M(s_M,w) = \Delta^*_M(«w)$.

We define for the both function a further important enhancement, namely for the final subsets S of the set $S_M \cup F_M \cup \{\texttt{RED}\}$ resp. $S_M \cup F^*_M$: $\delta^*_M(S,u) = \{\delta^*_M(s,u) \mid s \in S\}$, $\Delta^*_M(S,u) = \{\Delta^*_M(s,u) \mid s \in S\}$.

We will consider in the following only the reducing automata which fulfills the following natural condition: $\delta^*_M(s_M,u) = \texttt{RED}(n) \quad \Longrightarrow \quad |u| \geq |n|$.
Simple language by M**:** $L_0(M) = \{w \in \Sigma^*_M \mid \Delta^*_M(«w») = \texttt{ACC}\}$.
Characteristic constant k_M**:** $k_M = \max\left\{ |n| \,\middle|\, \texttt{RED}(n) \in F_M \right\}$.
The operation of reduction. We will exactly describe a reduction of a word by a binary sequence with the help of the following operation $/$: $a/\texttt{0} = a$, $a/\texttt{1} = \lambda$, $\lambda/n = \lambda$, $u/\lambda = u$, $(u \cdot a)/(n \cdot i) = (u/n)\cdot(a/i)$, where $u \in \Sigma^*$, $a \in \Sigma$, $n \in (\texttt{10}^*)^+$ and $i \in \{\texttt{0},\texttt{1}\}$. The size of the strings u, n is here unbounded, moreover u can be longer then n, and vice versa. The reduction of the word (a) by the sequence $\texttt{1}\cdot\texttt{0}\cdot\texttt{1}$ is given in the following way: $\texttt{(a)}/\texttt{101} = \texttt{a}$. We can describe the way how the red-automaton M reduces a word $w \in \Sigma^*_M$.
The relation of reduction denoted by $\Rightarrow_M$ is introduced in the following way: $«w» \Rightarrow_M «w'»$, if $\Delta^*_M(«w») = \texttt{RED}(n)$, and $«w»/n = «w'»$. If $«w» \Rightarrow_M «w'»$ holds, we say, that the automaton M *reduces* the word w into the word w'. We can see that $|w| > |w'|$. The relation $\Rightarrow^+$ is the transitive closure of $\Rightarrow$; $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$.
Analysis by reduction by M is any sequence of reductions $«w_1» \Rightarrow «w_2» \Rightarrow \ldots \Rightarrow «w_n»$, which cannot be further prolonged. If $w_n \in L_0(M)$, we speak about *accepting* analysis by reduction, in the other case we speak about *rejecting* analysis by reduction. Often we will speak about analysis instead of analysis by reduction.

Stages. Each computation of a red-automaton is divided in stages. At the beginning of each stage the head points at the leftmost item behind the left sentinel, and the control unit is in the (re)starting state. The stage ends as the control unit gets to any final state (operation) from F_M. There are three kinds of operations: `ACC`, `ERR`, and `RED`. Accordingly, we have *accepting* (`ACC`-), *rejecting* (`ERR`-), and *reducing* (`RED`-)*stages*.
Recognized language by M**:** $L(M) = \{w \mid «w» \Rightarrow^*_M «w'», \ \ and \ \ w' \in L_0(M)\}$.
Error and correctness preserving property. We can see the following usefull property: If $«w_1» \Rightarrow_M «w_2»$, then $w_1 \in L(M)$, exactly if $w_2 \in L(M)$.
Monotony. Monotony is an important property that enables to characterize the class of DCFL in terms of monotonic reducing automata. Informally a red-automaton M is monotonic if the size of sequences of non-visited items (symbols) in individual stages of any analysis by reduction by M is non-increasing.

2.1. Prefix and post-prefix (in)consistencies

Assumption. We assume in the following that $L \subseteq \Sigma^*$, and any symbol of Σ is a symbol of some word from L.We call a word v *inconsistent (incorrect)* with respect to the language $L \subseteq \Sigma^*$, if for any u, $w \in \Sigma^*$ is $uvw \notin L$. We can see that incorrect words can obtain proper incorrect sub-words. This fact lead us to the following notion. We say that a word v is an *incorrect core* of the word w with respect to the language L, if it is a subword of w, if it is incorrect with respect to the language L, and if it is minimal by the ordering "to be a sub-word". On the other hand, a word v is a *correct* sub-word of a word a w with respect to the language L, if $w = xvy$, and for some x', y' is $x'vy' \in L$. We say that v is a *correct core* of a word w with respect to the language L, if it is a correct sub-word of w with respect to L, and it is maximal by the ordering "to be a sub-word". The assumption that each symbol of Σ is a symbol of some word of the language L ensures that each symbol of any word $w \in \Sigma^*$ is contained in some correct core of this word.

Prefix consistence is the longest correct prefix v of the analyzed word w. *Prefix inconsistence* is the shortest incorrect prefix of the analyzed word w, i.e., it is the prefix va of w, where $a \in \Sigma$. *Post-prefix consistence* is a suffix x of a correct core behind (to the right of) the prefix consistence, or behind some of the previous post-prefix consistencies. We assign to the post-prefix consistency x the incorrect sub-word xa of w. We say that xa is a post-prefix inconsistence of w (with respect to L).

Our effort in the following is to deterministically, in a monotonic way to localize the prefix and post-prefix (in)consistencies in the analyzed words from DCFL.

3. Post-prefix robust analyzer A

Prefix consistence. A red-automaton M is *prefix-consistent* when for each word u and each symbol a (including the right sentinel) it holds the following: if $\Delta_M(s_M, u) \in S_M$ and $\Delta_M(s_M, ua) \neq$ `ERR`, then ua is a prefix of some word from $L(M) \cdot \{»\}$.

Let us note that monotone, prefix consistent, red-automata characterize the class of DCFL. We will show informally in the next part a method how construct for a given monotone, prefix-correct, state-minimal reducing automaton M a robust analyzer A which determines in any word $w \in$ «Σ_M^*» the prefix-(in)consistence, and (not obligatory all) post-prefix (in)consistencies. We suppose for the construction that $L(M) \neq \emptyset$.

At first A will use the prefix-consistency of M for the finding of the prefix-(in)consistency of the analyzed word w.

Such a situation can occur after one, or after more stages if M will be transfered into the final rejecting state `ERR`. The computation (analysis) of M on the word w until this moment we describe in the following way:

1) At first M (possibly) gradually reduces the word w into the word w', i.e., $w \Rightarrow_M^* w'$.

2) Then in the next stage M transfers over some prefix x of the word w' into some non-final state $s \in S_M$, i.e., $\delta_M^*(s_M, x) = s \in S_M$,

3) Finally from the state s transfers over the next symbol a into the final state `ERR`, i.e., $\delta_M(s, a) =$ `ERR`.

We can see that A has founded by the previous simulation of M the prefix inconsistency of w. For marking of the prefix inconsistency A inserts the sign `!` between the correct prefix «x and the symbol a.

The prefix-consistency of M ensures that M has visited in the last step described above the symbol a at the first time. Therefore if $w' =$ «xay for some y then ay is a suffix of the original input word w.

Let us now informaly describe how A continues in the robust analysis over the mentioned suffix ay of the word w.

We will use the function δ_M for this aim. This function was introduced as an enhancement of the transition function f_M. It describes not only the transfers between the individual states, but also the tranfers between the indiviual subsets of the set $S_M \cup F_M \cup \{\texttt{RED}\}$, i.e., of the set of all final and non-final states, and of a special state `RED`. We will use it in the following in order to describe the all possible computation of M over the suffix ay at the same time.

We let A to compute the function δ_M over the suffix $ay = a_0 a_1 \dots a_{|y|}$ starting from the set S_M of all non-final states of M. A will control the computation in the following way. Let us initially take the set S_M as a set further denoted as S_I.

Let us denote the following part of the computation of A as a cycle C_1. The cycle C_1 is performed until for the set $S = \delta_M^*(S_I, a_0 \dots a_i)$, where $0 \leq i \leq |y|$, holds that $\emptyset \subset S \subseteq S_M \cup \{ERR\}$, and S contains some non-final state. Then A performs the following action: the head of A will be placed to the next item to the right, and as (the current value of) the set S will be taken the set $\delta_M(S, a_{i+1})$. Here ends the description of C_1.

The core of the post-prefix analysis by A are the following four cases where is not fulfilled the condition for the remaining word in the cycle C_1.

Correct suffix. The set S contains the accepting state `ACC`; i.e., `ACC` $\in S$. If `ACC` $\in S$, then the current suffix of the analyzed word w by A is a suffix of some word from $L(M)$. Therefore, the current suffix cannot contain any further inconsistency. The work of A on w is finished at this moment.

An unambiguous inconsistency. The set S contains a single state – the rejecting state `ERR`; i.e., $S = \{\texttt{ERR}\}$. All the possible computations of M over the word w behind the previous inconsistency has ended at the same time in the state `ERR`. We have found a suffix of a correct core of the analyzed word, i.e., one of its post-prefix (in)consistency. At this moment A inserts the sign `!` immediately before the position of its working head. The automaton A will look for a new post-prefix (in)consistency behind (to the right from) the currently inserted sign `!`. A will take instead of the set $\{\mathtt{ERR}\}$ as the current value of the set S the set $\delta_M(S_M, a)$, where a is the symbol scanned by the working head. A will continue in the robust analysis of the remaining suffix by the schema of the cycle C_1.

An ambiguous reduction. S does not contain `ACC`, and either does contain two different reducing states of M, or does contain at least one non-final state, and at least one reducing state; i.e., $\mathtt{ACC} \notin S$, and $\exists n : \mathtt{RED}(n) \in S \not\subseteq \{\mathtt{RED}(n), \mathtt{ERR}\}$. We say that S fulfilling the condition above is an *ambiguous* set. The task for A is to work without false inconsistency messages. From that reason A separates the ambiguous part from the remaining suffix. It inserts the sign for the ambiguity `?` in the place of the current ambiguity, i.e., immediately to the left from the position of the working head (if the sign is not already placed there in some of the previous stages). At this moment A takes for the set S the complete set S_M, and continues in the robust analysis behind the sign `?` by the scheme of the cycle C_1.

An unambiguous reduction. The set S contains exactly one reducing operation, and possibly beside it the final state `ERR`; i.e., $\exists n : \mathtt{RED}(n) \in S \subseteq \{\mathtt{RED}(n), \mathtt{ERR}\}$.

Let us denote as u the sub-word which is created by the input symbols positioned between the last sign `!` or `?`, and the position of the working head including the scanned symbol. The sub-word u is because of the prefix-consistency, and because of the state minimality of M a sub-word of some word from $L(M)$. Moreover, the u is reduced in any word $w \in L(M)$ of the form $w = vux$ by the reducing seguence n, i.e, the reducing sequence and the position of the reduction are determined unambiguously. A will reduce also by n, but only the symbols from u if we consider the case that n can be longer then u.

Observation. The reducing sequence n deletes at least one symbol from u. This observation follows from the unambiguity of the reduction of u. Apart from the reduction of u by n, A will insert into the list a new item with an auxiliary symbol – the set U of pairs of an internal state and a word over an input alphabet of the length k_M at most. This auxiliary symbol will be used in the next stage to adjust the set of states computed by the function δ_M. Our goal is to avoid situation when $\delta^*_M(s, u) = \mathtt{ERR} \neq \delta^*_M(s, u/n)$ for some $s \in S_M$. Such internal states s must be eliminated. The set U is defined by the following way:

> If $|u| \geq |n|$, then A reduces the working list of items by n and A puts a new item with the auxiliary symbol U just in front of the leftmost deleted item. $U = \{(s, \lambda) \mid \exists s' \in S_M : \delta^*_M(s', u_1) = s$ and $\delta^*_M(s, u_2) = \mathtt{RED}(n)\}$ where $u = u_1u_2$ and $|u_2| = |n|$.
>
> If $|u| < |n|$, then A cannot reduce by the whole n as such a reduction would impact a part of the working list in front of u; this part would be reduced by n_1 such that $n = n_1n_2$ and $|n_2| = |u|$. But a part of the working list in front of the rightmost marker `!` or `?` can be reduced in some word of $L(M)$ in other way or even not at all. So, A will reduce items behind the rightmost marker `!` or `?` by the reducing sequence n_2 and it will insert a new item containing an auxiliary symbol U just to the right of the rightmost marker. $U = \{(s, x) \mid \exists v \in \Sigma^*_M : x = v/n_1$ a $|v| = |n_1|$ a $\delta^*_M(s, vu) = \mathtt{RED}(n)\}$.

Insertion of the set U into the working list is important as it ensures the continuity of subsequent stages of computation. In next stage, A will use this set to adjust the set S of internal states computed by function δ_M. As soon as A reach the item with U, it substitute S by $S' = \{\delta^*_M(s, v) \mid (s, v) \in U\}$. It guarantees that A enter a part of the list impacted by the last reduction in such states only that led to the last reduction of u by n resp. n_2.

A uses just defined set U in such a case only when the new item with U is inserted just behind an item containing a symbol of the input alphabet or a marker. Otherwise, when this item contains an auxiliary symbol U' different from both markers, then (instead of insertion of U) A replaces U' with U computed in the following way:
$U = \{(s,x) \mid \exists y, (s',x') \in U' : \ x = yx'/n_1$, and $\delta^*_M(s,y) = s'$, and $\delta^*_M(s',x'u) = \texttt{RED}(n)$, and $|y| = \max\{0, |n_1| - |x'|\}\}$, where n_1 is a prefix of n of the length $|n| - |u|$. In all cases, the length of the word x contained in any pair of inserted set U is bounded by the characteristic constant k_M which ensures that U is finite.

Now we have outlined the behavior of A in the first stage after the localization of the prefix-inconsistence. In the next stages we need also to consider the signs and the other auxiliary symbols inserted in the previous stages.

In [3] is in detail described the construction of the robust analyzer A and the transformation of it into a deterministic push-down transducer with the properties summarized in the following theorem.

Theorem 3.1 *Let M be a mon-red-automaton which is prefix-consistent, and state-minimal. Then there is a deterministic push-down transducer T which translates any word w from Σ^*_M on a word $p_A(w) \in (\Sigma_M \cup \{!, ?\})^*$ with the following properties:*
1. If $p_A(«w»)$ does not contain any sign $!$, then $p_A(«w») = «w»$ and w is from $L(M)$.
2. If $«w» \in «L(M)»$ then $p_A(«w») = «w»$.
3. If $«u!$ is a prefix of $p_A(«w»)$ and u does not contain the sign $!$ then u does not contain the sign $?$ as well, and $«u$ is the longest correct-prefix of the word $«w»$ with respect to the language $«L(M)»$.
4. If $!u!$ or $?u!$ is a sub-word of the word $p_A(«w»)$ and u does not contain any sign $!$ or $?$ then u is a suffix of some corect core of the word $«w»$ with respect to the language $«L(M)»$.
5. If $!u»$ or $?u»$ is a suffix of the word $p_A(«w»)$, and u does not contain any sign $!$ or $?$ then $u»$ is a sufix of some word from $«L(M)»$.
6. If $!u?$ or $?u?$ is a sub-word of $p_A(«w»)$ and u does not contain any sign $!$ or $?$ then u is a sub-word of some word from $«L(M)»$.

References

[1] Cormack G.V., *An LR Substring Parser for Noncorrecting Syntax Error Recovery*, in: Proc. of PLDI '89, 1989, 161-169.

[2] Jančar P., Mráz F., Plátek M., Vogel J.: *Restarting Automata*; in Proc. FCT'95, Dresden, Germany, August 1995, LNCS 965, Springer Verlag 1995, pp. 283 - 292

[3] Procházka M.: *Redukční automaty a syntaktické chyby*; (in Czech) text for PhD dissertation, submitted in July 2012.

[4] Procházka M.: *Concepts of Syntax Error Recovery for Monotonic Reducing Automata*, MIS 2004, pp. 94–103, http://ulita.ms.mff.cuni.cz/pub/MIS/MIS2004.pdf

[5] Procházka M., Plátek M.: *Localization of (In)Consistencies by Monotone Reducing Automata.*, Accepted for ITAT 2012.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 121 – 126

New Model for Picture Languages Recognition: Two-dimensional Sgraffito Automaton

Daniel Průša[(A)] František Mráz[(B)]

[(A)]Czech Technical University, Faculty of Electrical Engineering
Karlovo náměstí 13, 121 35 Prague 2, Czech Republic
prusapa1@cmp.felk.cvut.cz

[(B)]Charles University, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 25 Prague 1, Czech Republic
frantisek.mraz@mff.cuni.cz

Abstract

We present a new model of a two-dimensional computing device called sgraffito automaton and demonstrate its significance. In general, the model is simple, allows a clear design of important computations and defines families exhibiting good properties. It does not exceed the power of finite-state automata when working over one-dimensional inputs. On the other hand, it induces a family of picture languages that strictly includes REC and the deterministic variant recognizes languages in DREC as well as those accepted by four-way automata.

1. Introduction

The theory of two-dimensional languages generalizes concepts and techniques from the theory of formal languages. The basic element, which is a string, is extended to a matrix of symbols, usually called a picture. Various classes of picture languages can be formed, especially by generalizing one-dimensional computational or generative models, which possibly leads to some two-dimensional variant of the Chomsky hierarchy. Naturally we can ask, whether the induced families inherit properties of their one-dimensional counterparts. The answer is typically negative. A more complex topology of pictures causes that families of picture languages are of a different founding.

A four-way finite automaton (4FA) [2] is a good example. It is a finite-state device composed of a control unit equipped with a head moving over an input picture in four directions: left, right, up and down. Even if the automaton is a simple extension of the two-way finite automaton, the formed family of languages shows properties different from those of regular languages [4].

In 1991, Giammaresi and Restivo proposed the family of recognizable languages (REC) [3]. The languages in REC are defined using tiling systems. They also coincide with the

[(A)]The authors were supported by the Grant Agency of the Czech Republic: the first author under the project P103/10/0783 and the second author under the projects P103/10/0783 and P202/10/1333.

languages recognizable by the two-dimensional on-line tessellation automata (2OTA) [6] or definable using existential monadic second order logic. The family is well established, has many remarkable properties and the defined recognizability is a robust notion. It is even presented as the ground-level class among the families of two-dimensional languages. However, this vision is somehow contradicted by the fact that the non-determinism exhibited by REC makes it quite powerful. Even some NP-complete problems belong to REC [8]. This fact has inspired the further proposal of DREC [1] – the family of deterministically recognizable languages. It coincides with the closure under rotation of $\mathcal{L}(\mathsf{2DOTA})$, where 2DOTA denotes deterministic 2OTA.

In this abstract we present a new two-dimensional computing device called *sgraffito automaton* (2SA). We introduced it at DLT 2012 [9].

> Sgraffito (Italian: "scratched"), in the visual arts, a technique used in painting, pottery, and glass, which consists of putting down a preliminary surface, covering it with another, and then scratching the superficial layer in such a way that the pattern or shape that emerges is of the lower colour. (Encyclopædia Britannica Online. Retrieved 20 March, 2012)

The automaton has a finite state control and works on a picture consisting of symbols with different weights (as if they were put on its background in order from the lightest to the heaviest). 2SA can move its head over the picture in four directions. It must rewrite scanned symbol in each step and the symbol can be rewritten by a lighter symbol only (this corresponds to scratching some of the top layers). The automaton accepts by entering an accepting state.

A formal definition of the automaton is given in Section 2 Section 3 shows several closure properties for languages accepted by nondeterministic and deterministic 2SAs. The recognition power of the model is compared to other models in Section 4 Finally, concluding remarks are presented in Section 5

2. Sgraffito Automata

The sgraffito automaton is a special instance of two-dimensional Turing machine. For a picture $P \in \Sigma^{*,*}$, $\widehat{P}$ denotes the boundary picture which extends P by two border rows and columns formed of special markers from $\mathcal{S} = \{\vdash, \dashv, \top, \bot, \#\}$, called *sentinels*.

$$\begin{array}{|c|c|c|}
\hline
\# & \top\ \top\ \cdots\ \top\ \top & \# \\
\hline
\vdash & & \dashv \\
\vdots & P & \vdots \\
\vdash & & \dashv \\
\hline
\# & \bot\ \bot\ \cdots\ \bot\ \bot & \# \\
\hline
\end{array}$$

Furthermore, let $\mathcal{H} = \{\mathrm{R}, \mathrm{L}, \mathrm{D}, \mathrm{U}, \mathrm{Z}\}$ be the set of the *head movements* (right, left, down, up and none movement respectively). Define a mapping $\nu : \mathcal{S} \to \mathcal{H}$ where

$$\nu(\vdash) = \mathrm{R}, \quad \nu(\dashv) = \mathrm{L}, \quad \nu(\top) = \mathrm{D}, \quad \nu(\bot) = \mathrm{U} \text{ and } \nu(\#) = \mathrm{Z}.$$

Definition 2.1 *A (nondeterministic)* two-dimensional bounded Turing machine (2BTM) *is a tuple* $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Q_F)$ *where*

- Σ *is an input alphabet,*
- Γ *is a working alphabet such that* $\Sigma \subseteq \Gamma$,
- Q *is a finite, nonempty set of states,*
- $q_0 \in Q$ *is the initial state,*
- $Q_F \subseteq Q$ *is the set of final states, and*
- $\delta : (Q \setminus Q_F) \times (\Gamma \cup \mathcal{S}) \to 2^{Q \times (\Gamma \cup \mathcal{S}) \times \mathcal{H}}$ *is a transition relation.*

Moreover, for any pair $(q, a) \in Q \times (\Gamma \cup \mathcal{S})$, *every* $(q', a', d) \in \delta(q, a)$ *fulfils*

- $a \in \mathcal{S}$ *implies* $d = \nu(a) \wedge a' = a$, *and*
- $a \notin \mathcal{S}$ *implies* $a' \notin \mathcal{S}$.

If $\forall q \in Q, \forall a \in \Gamma \cup \mathcal{S} : |\delta(q, a)| \leq 1$, *we say* $\mathcal{M}$ *is a* deterministic 2BTM.

The notions like configuration and computation of the machine $\mathcal{M}$ are easily defined as usual. Let $P \in \Sigma^{*,*}$ be an input. In the initial configuration of $\mathcal{M}$ on P, its working tape contains $\widehat{P}$, its control unit is in state q_0 and the head scans the top-left corner of P. When P is the empty picture, the head scans the bottom-right corner of $\widehat{P}$ containing #. The machine accepts P iff there is a computation of $\mathcal{M}$ starting in the initial configuration on P and finishing in a state from Q_F.

Definition 2.2 A two-dimensional sgraffito automaton (2SA) *is a tuple* $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Q_F, \mu)$ *where*

- $(Q, \Sigma, \Gamma, \delta, q_0, Q_F)$ *is a* 2BTM,
- $\mu : \Gamma \to \mathbb{N}$ *is a weight function and the transition relation satisfies*

$$(q', a', d) \in \delta(q, a) \Rightarrow \mu(a') < \mu(a) \quad \textit{for all } q, q' \in Q, d \in \mathcal{H}, a, a' \in \Gamma.$$

$\mathcal{A}$ *is a* deterministic 2SA (2DSA) *if the underlying* 2BTM *is deterministic.*

Lemma 2.3 *Let* $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Q_F)$ *be a* 2BTM. *Let* $k \in \mathbb{N}$ *be an integer such that during each computation of* $\mathcal{M}$ *over any picture from* $\Sigma^{*,*}$, *each tape field is scanned by the head of* $\mathcal{M}$ *in at most* k *configurations. Then, there is a* 2SA $\mathcal{A}$ *such that* $L(\mathcal{A}) = L(\mathcal{M})$. *Moreover, if* $\mathcal{M}$ *is deterministic,* $\mathcal{A}$ *is deterministic too.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \Gamma_2, \delta_2, q_0, Q_F, \mu)$ be a 2SA, where $\Gamma_2 = \Sigma \cup (\Gamma \times \{1, \ldots, k\})$ and each instruction $(q, a) \to (q', a', d)$ from δ is represented in δ_2 by the following set of instructions:

$$\begin{aligned} (q, a) &\to (q', (a', 1), d), \\ (q, (a, i)) &\to (q', (a', i+1), d) \quad \forall i \in \{1, \ldots, k-1\}. \end{aligned}$$

Finally, we define

$$\begin{aligned} \mu(a) &= k + 1 & \forall a \in \Sigma, \\ \mu((a, i)) &= k + 1 - i & \forall (a, i) \in \Gamma \times \{1, \ldots, k\}. \end{aligned}$$

We shall see that $L(\mathcal{A}) = L(\mathcal{M})$ and that deterministic δ produces deterministic δ_2. □

Lemma 2.3 says that, instead of designing a 2SA, it is sufficient to describe a 2BTM for which the number of transitions over each tape field is bounded by a constant. One-dimensional constant-visit machines were already studied by Hennie [5]. He proved that such machines recognize only the regular languages. This implies that sgraffito automata restricted to work over one-row (or one-column) pictures recognize just the regular languages as well.

3. Closure Properties

The common closure properties of the families induced by sgraffito automata are identical to the closure properties of REC and DREC, as it is demonstrated in this section.

Theorem 3.1
- $\mathcal{L}(\mathsf{2SA})$ *and* $\mathcal{L}(\mathsf{2DSA})$ *are closed under union, intersection, rotation and mirroring.*
- $\mathcal{L}(\mathsf{2SA})$ *is closed under row and column concatenation and projection.*
- $\mathcal{L}(\mathsf{2DSA})$ *is closed under complement.*

Proof. A clear constructive proof can be given for each particular claim. Let us demonstrate this for the column concatenation of picture languages L_1, L_2, denoted by $L_1 \obar L_2$.

Let $\mathcal{A}_1$, $\mathcal{A}_2$ be 2SAs such that $L_1 = L(\mathcal{A}_1)$, $L_2 = L(\mathcal{A}_2)$. We construct a 2SA $\mathcal{A}$ which nondeterministically chooses a column in the input $\widehat{P}$ and marks it. Then it simulates $\mathcal{A}_1$ over the left part (including the marked column) and, after that, $\mathcal{A}_2$ over the right part (excluding the marked column). $\mathcal{A}$ accepts if both simulations finished with accepting. □

To reveal more properties of sgraffito automata, we utilize two languages over $\Sigma = \{0,1\}$. The language of "duplicates" L_{dup} consists of all pictures $Q \obar Q$, where Q is a nonempty square over Σ. The language of "permutations" L_{perm} is a subset of L_{dup} and consists of those pictures $Q \obar Q$, where each row and each column of Q contains symbol 1 exactly once. Examples are shown in Figure 1.

$$
(a)\ \begin{array}{|cccc|cccc|}
\hline
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\hline
\end{array}
\qquad
(b)\ \begin{array}{|cccc|cccc|}
\hline
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\hline
\end{array}
$$

Figure 1: Sample pictures from (a) L_{dup} and (b) L_{perm}.

Proposition 3.2 ([4, 7]) L_{dup} *and* L_{perm} *are not in* REC, *while their complements are in* REC.

An analogous result can be obtained for L_{dup} when considering sgraffito automata.

Lemma 3.3 L_{dup} *is not accepted by any* 2SA. $\overline{L}_{\text{dup}}$ *is not accepted by any* 2DSA.

Utilizing Lemma 3.3, the following theorem is proved.

Theorem 3.4 $\mathcal{L}(\mathsf{2SA})$ *is not closed under complement.* $\mathcal{L}(\mathsf{2DSA})$ *is not closed under projection and row, neither column concatenation.*

The studied closure properties are summarized in Figure 2. Their coincidence with the properties of REC and DREC is emphasized.

	$\cup$	$\cap$	$\setminus$	$\ominus, \oplus$	π	R,VM,HM
REC	yes	yes	no	yes	yes	yes
$\mathcal{L}(\mathsf{2SA})$	yes	yes	no	yes	yes	yes
DREC	yes	yes	yes	no	no	yes
$\mathcal{L}(\mathsf{2DSA})$	yes	yes	yes	no	no	yes

Figure 2: Overview of closure properties.

4. A Taxonomy of Picture Languages

This section compares the power of sgraffito automata with four-way automata and on-line tessellation automata.

It is a known fact that DREC and $\mathcal{L}(\mathsf{4FA})$ are incomparable. We showed that 2DSAs accept all languages recognized by 4FAs. The proof is based on two facts - an oriented graph G expressing transitions of a 4FA can be represented by a 2DSA $\mathcal{A}$ and $\mathcal{A}$ can perform a depth-first search in G determining accessibility of a final state from the initial configuration.

Theorem 4.1 $\mathcal{L}(\mathsf{4FA})$ *is included in* $\mathcal{L}(\mathsf{2DSA})$.

Furthermore, a straightforward simulation of 2OTA, resp. 2DOTA can be easily designed.

Theorem 4.2 REC *is included in* $\mathcal{L}(\mathsf{2SA})$, DREC *is included in* $\mathcal{L}(\mathsf{2DSA})$.

While L_{dup} cannot be recognized by any 2SA, there is a 2DSA recognizing L_{perm}. It is possible to check if a pair of symbols 1 in a row is located in columns with the same index in each half of the picture. Moreover, the whole verification can be done by a constant number of visits in each tape field – a unique row, column and diagonals are used to define a bouncing style traversal based localization for each pair. For more details, see [9].

Since we have shown $\overline{L}_{\text{dup}} \in (\mathsf{REC} \setminus \mathcal{L}(\mathsf{2DSA}))$ and $L_{\text{perm}} \in (\mathcal{L}(\mathsf{2DSA}) \setminus \mathsf{REC})$, families $\mathcal{L}(\mathsf{2DSA})$ and REC are incomparable.

All the mentioned relationships are summarized in Figure 3.

5. Conclusion

We have introduced a new computational model called sgraffito automaton and investigated its properties. If the automaton is restricted to work over one-row pictures only, the recognition power degenerates to the power of finite-state automaton. Such results give the families a

$$\begin{tikzcd}
 & \mathcal{L}(\mathsf{2SA}) \\
\mathsf{REC} \arrow[ur] \arrow[r, dash, dashed] & \mathcal{L}(\mathsf{2DSA}) \arrow[u] \\
\mathsf{DREC} \arrow[u] \arrow[ur] \arrow[r, dash, dashed] & \mathcal{L}(\mathsf{4FA}) \arrow[u] \arrow[ul]
\end{tikzcd}$$

Figure 3: Relationships between studied families. Arrows represent proper inclusions while dashed lines connect incomparable classes.

great importance and entitle us to see them as alternative ground levels in the two-dimensional hierarchy. This is also well justified by the results on closure properties. We think that sgraffito automata deserve to be the subject of further research. A special attention should be paid to 2DSAs, since they simulate 4FAs and define thus an interesting deterministic family. The study of the automata can provide additional insight on the fundamental differences between one-dimensional and two-dimensional languages.

References

[1] M. ANSELMO, D. GIAMMARRESI, M. MADONIA, From Determinism to Non-determinism in Recognizable Two-Dimensional Languages. In: T. HARJU, J. KARHUMÄKI, A. LEPISTÖ (eds.), *Developments in Language Theory*. LNCS 4588, Springer, 2007, 36–47.

[2] M. BLUM, C. HEWITT, Automata on a 2-dimensional tape. In: *Proceedings of the 8th Annual Symposium on Switching and Automata Theory (SWAT 1967)*. FOCS '67, IEEE Computer Society, Washington, DC, USA, 1967, 155–160.

[3] D. GIAMMARRESI, A. RESTIVO, Recognizable picture languages. *International Journal of Pattern Recognition and Artificial Intelligence* **6** (1992) 2-3, 32–45.

[4] D. GIAMMARRESI, A. RESTIVO, Two-dimensional languages. In: G. ROZENBERG, A. SALOMAA (eds.), *Handbook of formal languages, Vol. 3*. Springer-Verlag New York, Inc., New York, NY, USA, 1997, 215–267.

[5] F. HENNIE, One-tape, off-line Turing machine computations. *Information and Control* **8** (1965) 6, 553–578.

[6] K. INOUE, A. NAKAMURA, Some properties of two-dimensional on-line tessellation acceptors. In: *Information Sciences*. 13, 1977, 95–121.

[7] J. KARI, C. MOORE, New Results on Alternating and Non-deterministic Two-Dimensional Finite-State Automata. In: A. FERREIRA, H. REICHEL (eds.), *STACS 2001*. LNCS 2010, Springer Berlin / Heidelberg, 2001, 396–406.

[8] K. LINDGREN, C. MOORE, M. NORDAHL, Complexity of Two-Dimensional Patterns. *Journal of Statistical Physics* **91** (1998) 5-6, 909–951.

[9] D. PRŮŠA, F. MRÁZ, Two-Dimensional Sgraffito Automata. In: H.-C. YEN, O. IBARRA (eds.), *Developments in Language Theory*. Lecture Notes in Computer Science 7410, Springer Berlin / Heidelberg, 2012, 251–262.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 127 – 132

Inside the Class of REGEX Languages

Markus L. Schmid

Department of Computer Science, Loughborough University,
Loughborough, Leicestershire, LE11 3TU, United Kingdom
M.Schmid@lboro.ac.uk

Abstract

We study different possibilities of combining the concept of homomorphic replacement with regular expressions in order to investigate the class of languages given by extended regular expressions with backreferences (REGEX). It is shown in which regard existing and natural ways to do this fail to reach the expressive power of REGEX.

1. Introduction

Since their introduction by Kleene in 1956 [7], *regular expressions* have not only constantly challenged researchers in formal language theory, they also attracted pioneers of applied computer science as, e. g., Thompson [9], who developed one of the first implementations of regular expressions, marking the beginning of a long and successful tradition of their practical application (see Friedl [5] for an overview). In order to suit practical requirements, regular expressions have undergone various modifications and extensions which lead to so-called *extended regular expressions with backreferences* (*REGEX* for short), nowadays a standard element of most text editors and programming languages (cf. Friedl [5]).

The main difference between REGEX and *classical* regular expressions is the concept of backreferences. Intuitively speaking, a backreference points back to an earlier subexpression, meaning that it has to be matched to the same word the earlier subexpression has been matched to. For example, $r := (_1\ (\mathtt{a} \mid \mathtt{b})^*\ )_1 \cdot \mathtt{c} \cdot \backslash 1$ is a REGEX, where $\backslash 1$ is a *backreference* to the *referenced subexpression* in between the parentheses $(_1$ and $)_1$. The language described by r, denoted by $\mathcal{L}(r)$, is the set of all words wcw, $w \in \{\mathtt{a}, \mathtt{b}\}^*$; a non-regular language.

A suitable language theoretical approach to these backreferences is the concept of *homomorphic replacement*. For example, the REGEX r can also be given as a string $x\mathtt{b}x$, where the symbol x can be homomorphically replaced by words from $\{\mathtt{a}, \mathtt{b}\}^*$, i. e., both occurrences of x must be replaced by the same word. Numerous language generating devices can be found that use various kinds of homomorphic replacement. The most prominent example are probably the well-known L systems (see Kari et al. [6] for a survey), but also many types of grammars as, e. g., Wijngaarden grammars, macro grammars, Indian parallel grammars or deterministic iteration grammars, use homomorphic replacement as a central concept (cf. Albert and Wegner [1] and Bordihn et al. [3] and the references therein). Albert and Wegner [1] and Angluin [2] introduced H-systems and pattern languages, respectively, which both use homomorphic replacement in a more puristic way, without any grammar like mechanisms. More recent models like pattern expressions (Câmpeanu and Yu [4]), synchronized regular expressions (Della Penna et al. [8]) and EH-expressions (Bordihn et al. [3]) are mainly inspired directly by REGEX.

The contribution of this paper is to investigate alternative possibilities to combine the two most elementary components of REGEX, i. e., regular expressions and homomorphic replacement, with the objective of reaching the expressive power of REGEX as close as possible, without exceeding it. Particularly challenging about REGEX is that due to the possible nesting of referenced subexpression the concepts of regular expressions and homomorphic replacement seem to be inherently entangled and there is no easy way to treat them separately. We illustrate this with the example $t := (_1\ \mathtt{a}^*\ )_1 \cdot (_2\ (\mathtt{b} \cdot \backslash 1)^*\ )_2 \cdot \backslash 2 \cdot \backslash 1$. The language $\mathcal{L}(t) := \{\mathtt{a}^n(\mathtt{ba}^n)^m(\mathtt{ba}^n)^m\mathtt{a}^n \mid n,m \geq 0\}$ cannot that easily be described in terms of a single string with a homomorphic replacement rule, e. g., by the string $xyyx$, where x can be replaced by words from $\{\mathtt{a}^n \mid n \geq 0\}$, and y by words of form $\{(\mathtt{ba}^n)^m \mid n,m \geq 0\}$, since then we can obtain words $\mathtt{a}^n(\mathtt{ba}^{n'})^m(\mathtt{ba}^{n'})^m\mathtt{a}^n$ with $n \neq n'$. In fact, two steps of homomorphic replacement seem necessary, i. e., we first replace y by words from $\{(\mathtt{b}z)^n \mid n \geq 0\}$ and after that we replace x and z by words from $\{\mathtt{a}^n \mid n \geq 0\}$, with the additional requirement that x and z are substituted by the same word. More intuitively speaking, the nesting of referenced subexpressions require *iterated* homomorphic replacement, but we also need to carry on information from one step of replacement to the next one.

The concept of homomorphic replacement is covered best by *pattern languages* (see Angluin [2]). We combine patterns with regular expressions by first adding the alternation and star operator to patterns and, furthermore, by letting their variables be typed by regular languages, i. e., the words variables are replaced with are from given regular sets. Then we iterate this step by using this new class of languages again as types for variables and so on. We also take a closer look at *pattern expressions* (see Câmpeanu and Yu [4]). In [4], many examples are provided that show how to translate a REGEX into an equivalent pattern expression and vice versa. It is also stated that this is possible in general, but a formal proof for this statement is not provided. In the present work we show that pattern expressions are in fact much weaker than REGEX and they describe a proper subset of the class of REGEX languages. These limits in expressive power are caused by the above described difficulties due to the nesting of referenced subexpressions.

On the other hand, pattern expressions still describe an important and natural subclass of REGEX languages, that has been independently defined in terms of other models and, as shown in this work, also coincides with the class of languages resulting from the modification of patterns described above. We then refine the way of how pattern expressions define languages in order to accommodate the nesting of referenced subexpressions and we show that the thus obtained class of languages coincides with the class of languages given by REGEX that do not contain a referenced subexpression under a star.

2. General Definitions

Let $\mathbb{N} := \{1,2,3,\ldots\}$ and let $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. For an arbitrary alphabet A, a *word* (*over* A) is a finite sequence of symbols from A, and ε stands for the *empty word*. The notation A^+ denotes the set of all nonempty words over A, and $A^* := A^+ \cup \{\varepsilon\}$. For the *concatenation* of two words w_1, w_2 we write $w_1 \cdot w_2$ or simply $w_1 w_2$. We say that a word $v \in A^*$ is a *factor* of a word $w \in A^*$ if there are $u_1, u_2 \in A^*$ such that $w = u_1 \cdot v \cdot u_2$. The notation $|K|$ stands for the size of a set K or the length of a word K.

We use regular expression as they are commonly defined (see, e. g., Yu [10]). For the alternation operations we use the symbol "|". For any regular expression r, $\mathcal{L}(r)$ denotes the language described by r and REG denotes the set of regular languages. Let Σ be a finite alphabet of *terminal symbols* and let $X := \{x_1, x_2, x_3, \ldots\}$ be a countably infinite set of *variables* with $\Sigma \cap X = \emptyset$. For any word $w \in (\Sigma \cup X)^*$, var(w) denotes the set of variables that occur in w.

3. Patterns with Regular Operators and Types

In this section, we combine the pattern languages mentioned in Section 1. with regular languages and regular expressions.

Let PAT $:= \{\alpha \mid \alpha \in (\Sigma \cup X)^+\}$ and every $\alpha \in$ PAT is called a *pattern*. We always assume that, for every $i \in \mathbb{N}$, $x_i \in \text{var}(\alpha)$ implies $\{x_1, x_2, \ldots, x_{i-1}\} \subseteq \text{var}(\alpha)$. For any alphabets A, B, a *morphism* is a function $h : A^* \to B^*$ that satisfies $h(vw) = h(v)h(w)$ for all $v, w \in A^*$. A morphism $h : (\Sigma \cup X)^* \to \Sigma^*$ is called a *substitution* if $h(a) = a$ for every $a \in \Sigma$. For an arbitrary class of languages $\mathfrak{L}$ and a pattern α with $|\text{var}(\alpha)| = m$, an *$\mathfrak{L}$-type for α* is a tuple $\mathcal{T} := (T_{x_1}, T_{x_2}, \ldots, T_{x_m})$, where, for every i, $1 \leq i \leq m$, $T_{x_i} \in \mathfrak{L}$ and T_{x_i} is called the *type language of (variable) x_i*. A substitution h *satisfies $\mathcal{T}$* if and only if, for every i, $1 \leq i \leq m$, $h(x_i) \in T_{x_i}$.

Definition 3.1 *Let $\alpha \in$ PAT, let $\mathfrak{L}$ be a class of languages and let $\mathcal{T}$ be an $\mathfrak{L}$-type for α. The set $\mathcal{L}_{\mathcal{T}}(\alpha) := \{h(\alpha) \mid h$ is a substitution that satisfies $\mathcal{T}\}$ is the* $\mathcal{T}$-typed pattern language of α. *For any class of languages $\mathfrak{L}$, $\mathcal{L}_{\mathfrak{L}}(\text{PAT}) := \{\mathcal{L}_{\mathcal{T}}(\alpha) \mid \alpha \in \text{PAT}, \mathcal{T}$ is an $\mathfrak{L}$-type for $\alpha\}$ is the* class of $\mathfrak{L}$-typed pattern languages.

In order to describe larger classes of REGEX languages by means of the pattern-based formalism given in Definition 3.1, the next step could be to type the variables of patterns with languages from $\mathcal{L}_{\text{REG}}(\text{PAT})$ instead of REG and then using the thus obtained languages again as type languages and so on. However, this approach leads to a dead end:

Proposition 3.2 *For any class of languages $\mathfrak{L}$, $\mathcal{L}_{\mathfrak{L}}(\text{PAT}) = \mathcal{L}_{\mathcal{L}_{\mathfrak{L}}(\text{PAT})}(\text{PAT})$.*

This observation brings us to the definition of the set of *patterns with regular operators*: $\text{PAT}_{\text{ro}} := \{\alpha \mid \alpha$ is a regular expression over $(\Sigma \cup X)\}$. In order to define the language given by a pattern with regular operators, we extend the definition of types to patterns with regular operators in the obvious way.

Definition 3.3 *Let $\alpha \in \text{PAT}_{\text{ro}}$ and let $\mathcal{T}$ be a type for α. The* $\mathcal{T}$-typed pattern language of α *is defined by $\mathcal{L}_{\mathcal{T}}(\alpha) := \bigcup_{\beta \in \mathcal{L}(\alpha)} \mathcal{L}_{\mathcal{T}}(\beta)$. For any class of languages $\mathfrak{L}$, we define $\mathcal{L}_{\mathfrak{L}}(\text{PAT}_{\text{ro}}) := \{\mathcal{L}_{\mathcal{T}}(\alpha) \mid \alpha \in \text{PAT}_{\text{ro}}, \mathcal{T}$ is an $\mathfrak{L}$-type for $\alpha\}$.*

Patterns with regular operators are also used in the definition of pattern expressions (see [4] and Section 4.) and have been called *regular patterns* in [3].

It seems reasonable to assume that REG-typed patterns with regular operators are strictly more powerful than REG-typed patterns without regular operators. In the following proposition, we formally prove this intuition.

Proposition 3.4 $\mathcal{L}_{\{\Sigma^*\}}(\mathrm{PAT}) \subset \mathcal{L}_{\mathrm{REG}}(\mathrm{PAT}) \subset \mathcal{L}_{\mathrm{REG}}(\mathrm{PAT}_{\mathrm{ro}})$.

The invariance of typed patterns – represented by Proposition 3.2 – does not hold anymore with respect to patterns with regular operators. Before we formally prove this claim, we shall define an infinite hierarchy of classes of languages given by typed patterns with regular operators. Let $\mathfrak{L}_{\mathrm{ro},0} := \mathrm{REG}$ and, for every $i \in \mathbb{N}$, we define $\mathfrak{L}_{\mathrm{ro},i} := \mathcal{L}_{\mathfrak{L}_{\mathrm{ro},i-1}}(\mathrm{PAT}_{\mathrm{ro}})$. Furthermore, we define $\mathfrak{L}_{\mathrm{ro},\infty} = \bigcup_{i=0}^{\infty} \mathfrak{L}_{\mathrm{ro},i}$.

It follows by definition, that the classes $\mathfrak{L}_{\mathrm{ro},i}$, $i \in \mathbb{N}_0$, form a hierarchy and we strongly conjecture that it is proper. However, here we only separate the first three levels of that hierarchy.

Theorem 3.5 $\mathfrak{L}_{\mathrm{ro},0} \subset \mathfrak{L}_{\mathrm{ro},1} \subset \mathfrak{L}_{\mathrm{ro},2} \subseteq \mathfrak{L}_{\mathrm{ro},3} \subseteq \mathfrak{L}_{\mathrm{ro},4} \subseteq \ldots$.

4. Pattern Expressions

We now define pattern expressions as introduced by Câmpeanu and Yu [4], but we use a slightly different notation.

Definition 4.1 *A* pattern expression *is a tuple* $(x_1 \to r_1, x_2 \to r_2, \ldots, x_n \to r_n)$*, where, for every* i*,* $1 \leq i \leq n$*,* $r_i \in \mathrm{PAT}_{\mathrm{ro}}$ *and* $\mathrm{var}(r_i) \subseteq \{x_1, x_2, \ldots, x_{i-1}\}$*. The set of all pattern expressions is denoted by* PE*.*

In [4], the language of a pattern expression is defined in the following way.

Definition 4.2 *Let* $p := (x_1 \to r_1, x_2 \to r_2, \ldots, x_n \to r_n)$ *be a pattern expression. We define* $L_{p,x_1} := \mathcal{L}(r_1)$ *and, for every* i*,* $2 \leq i \leq n$*, we define* $L_{p,x_i} := \mathcal{L}_{\mathcal{T}_i}(r_i)$*, where* $\mathcal{T}_i := (L_{p,x_1}, L_{p,x_2}, \ldots, L_{p,x_{i-1}})$ *is a type for* r_i*. The* language generated by p with respect to iterated substitution *is defined by* $\mathcal{L}_{\mathrm{it}}(p) := L_{p,x_n}$ *and* $\mathcal{L}_{\mathrm{it}}(\mathrm{PE}) := \{\mathcal{L}_{\mathrm{it}}(p) \mid p \in \mathrm{PE}\}$*.*

The class of languages described by pattern expressions with respect to iterated substitution coincides with the class $\mathfrak{L}_{\mathrm{ro},\infty}$ of the previous section:

Theorem 4.3 $\mathfrak{L}_{\mathrm{ro},\infty} = \mathcal{L}_{\mathrm{it}}(\mathrm{PE})$.

In the following, we define an alternative way of how pattern expressions can describe languages, i. e., instead of substituting the variables by words in an iterative way, we substitute them uniformly.

Definition 4.4 *Let* $p := (x_1 \to r_1, x_2 \to r_2, \ldots, x_n \to r_n) \in \mathrm{PE}$*. A word* $w \in \Sigma^*$ *is in the* language generated by p with respect to uniform substitution *(*$\mathcal{L}_{\mathrm{uni}}(p)$*, for short) if and only if there exists a substitution* h *such that* $h(x_n) = w$ *and, for every* i*,* $1 \leq i \leq n$*, there exists an* $\alpha_i \in \mathcal{L}(r_i)$ *with* $h(x_i) = h(\alpha_i)$*.*

For an arbitrary pattern expression $p := (x_1 \to r_1, x_2 \to r_2, \dots, x_n \to r_n)$, the language $\mathcal{L}_{\text{uni}}(p)$ can also be defined in a more constructive way. We first choose a word $u \in \mathcal{L}(r_1)$ and, for all i, $1 \leq i \leq n$, if variable x_1 occurs in r_i, then we substitute all occurrences of x_1 in r_i by u. Then we delete the element $x_1 \to r_1$ from the pattern expression. If we repeat this step with respect to variables $x_2, x_3, \dots, x_{n-1}$, then we obtain a pattern expression of form $(x_n \to r'_n)$, where r'_n is a regular expression over Σ. The language $\mathcal{L}_{\text{uni}}(p)$ is the union of the languages given by all these regular expression.

The language $\mathcal{L}_{\text{it}}(p)$ can be defined similarly. We first choose a word $u_1 \in \mathcal{L}(r_1)$ and then we substitute all occurrences of x_1 in r_2 by u_1. After that, we choose a *new* word $u_2 \in \mathcal{L}(r_1)$ and substitute all occurrences of x_1 in r_3 by u_2 and so on until there are no more occurrences of variable x_1 in q and then we delete the element $x_1 \to r_1$. Then this step is repeated with respect to $x_2, x_3, \dots, x_{n-1}$.

The above considerations yield the following proposition:

Proposition 4.5 *Let $p := (x_1 \to r_1, x_2 \to r_2, \dots, x_m \to r_m)$ be a pattern expression. Then $\mathcal{L}_{\text{uni}}(p) \subseteq \mathcal{L}_{\text{it}}(p)$ and if, for every i, j, $1 \leq i < j \leq m$, $\text{var}(r_i) \cap \text{var}(r_j) = \emptyset$, then also $\mathcal{L}_{\text{it}}(p) \subseteq \mathcal{L}_{\text{uni}}(p)$.*

While it can be easily shown that every language given by a pattern expression with respect to iterated substitution can also be defined by a pattern expression with respect to uniform substitution, the question of whether or not, for every pattern expression p, we can find a pattern expression p' with $\mathcal{L}_{\text{uni}}(p) = \mathcal{L}_{\text{it}}(p')$, is not that easy to answer. It can be shown that there are in fact languages that can be expressed by some pattern expression with respect to uniform substitution, but not by any pattern expression with respect to iterated substitution, which yields the main result of this section:

Theorem 4.6 $\mathcal{L}_{\text{it}}(\text{PE}) \subset \mathcal{L}_{\text{uni}}(\text{PE})$.

We mention that in Bordihn et al. [3], it has been shown that $\mathcal{H}^*(\text{REG}, \text{REG})$, a class of languages given by an iterated version of H-systems (see Albert and Wegner [1] and Bordihn et al. [3]), also coincides with $\mathcal{L}_{\text{it}}(\text{PE})$, which implies $\mathfrak{L}_{\text{ro},\infty} = \mathcal{L}_{\text{it}}(\text{PE}) = \mathcal{H}^*(\text{REG}, \text{REG}) \subset \mathcal{L}_{\text{uni}}(\text{PE})$. Next, we compare the class $\mathcal{L}_{\text{uni}}(\text{PE})$ to the class of REGEX languages.

A REGEX r is *star-free initialised* if and only if every referenced subexpression does not occur under a star. Let $\text{REGEX}_{\text{sfi}}$ be the set of REGEX that are star-free initialised. It can be shown that the class of languages described by pattern expressions with respect to uniform substitution coincides with the class of languages given by regular expressions that are star-free initialised:

Theorem 4.7 $\mathcal{L}(\text{REGEX}_{\text{sfi}}) = \mathcal{L}_{\text{uni}}(\text{PE})$.

In Sections 3 and 4, we have investigated several proper subclasses of the class of REGEX languages and their mutual relations. We conclude this section, by summarising these results:

$$\mathcal{L}_{\{\Sigma^*\}}(\text{PAT}) \subset \mathcal{L}_{\text{REG}}(\text{PAT}) \subset \mathfrak{L}_{\text{ro},1} \subset \mathfrak{L}_{\text{ro},2} \subseteq \mathfrak{L}_{\text{ro},3} \subseteq \ldots \subseteq \mathfrak{L}_{\text{ro},\infty} = \\ \mathcal{H}^*(\text{REG}, \text{REG}) = \mathcal{L}_{\text{it}}(\text{PE}) \subset \mathcal{L}_{\text{uni}}(\text{PE}) = \mathcal{L}(\text{REGEX}_{\text{sfi}}) \subseteq \mathcal{L}(\text{REGEX}).$$

References

[1] J. Albert, L. Wegner, Languages with homomorphic replacements. *Theoretical Computer Science* **16** (1981), 291–305.

[2] D. Angluin, Finding patterns common to a set of strings. In: *Proc. 11th Annual ACM Symposium on Theory of Computing*. 1979, 130–141.

[3] H. Bordihn, J. Dassow, M. Holzer, Extending regular expressions with homomorphic replacement. *RAIRO Theoretical Informatics and Applications* **44** (2010), 229–255.

[4] C. Câmpeanu, S. Yu, Pattern expressions and pattern automata. *Information Processing Letters* **92** (2004), 267–274.

[5] J. E. F. Friedl, *Mastering Regular Expressions*. Third edition, O'Reilly, Sebastopol, CA, 2006.

[6] L. Kari, G. Rozenberg, A. Salomaa, L systems. In: G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages*. 1. chapter 5, Springer, 1997, 253–328.

[7] S. Kleene, Representation of events in nerve nets and finite automata. In: C. Shannon, J. McCarthy (eds.), *Automata Studies*. Annals of Mathematics Studies 34, Princeton University Press, 1956, 3–41.

[8] G. D. Penna, B. Intrigila, E. Tronci, M. V. Zilli, Synchronized regular expressions. *Acta Informatica* **39** (2003), 31–70.

[9] K. Thompson, Programming Techniques: Regular expression search algorithm. *Communications of the ACM* **11** (1968).

[10] S. Yu, Regular Languages. In: G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages*. 1. chapter 2, Springer, 1997, 41–110.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 133 – 138

Closure Properties of Parallel Communicating Restarting Automata Systems

Marcel Vollweiler$^{(A)}$

$^{(A)}$Fachbereich Elektrotechnik/Informatik
Universität Kassel, 34109 Kassel, Germany
`vollweiler@theory.informatik.uni-kassel.de`

Abstract

Systems of parallel communicating restarting automata were announced in [4] and introduced in [13, 14]. Within these systems restarting automata work together in a parallel way with a high degree of independence. For this, a communication protocol is used for transmitting messages between the automata. Depending on the type of restarting automata that are used as the components of the system, various language classes were characterized. Here, closure properties are presented for these language classes, and it is shown that three of them, namely $\mathcal{L}(\mathsf{PC\text{-}RLWW})$, $\mathcal{L}(\mathsf{PC\text{-}RRWW})$, and $\mathcal{L}(\mathsf{PC\text{-}RWW})$ are actually so-called AFLs (abstract families of languages).

1. Introduction

On the one hand, systems of parallel communicating components were considered for several formal language devices, e.g. parallel communicating grammar systems [10, 1], parallel communicating finite automata systems [8], parallel communicating pushdown automata systems [2], and parallel communicating Watson-Crick automata systems [3]. The basic motivation for this kind of systems is the *classroom model* that is a problem solving strategy where several agents work together in a parallel manner and, although they work independently of each other, they are allowed to transmit messages by communication.

On the other hand, restarting automata were introduced in [6] as a device to model the linguistic technique of *analysis by reduction* that is used to check the syntactical correctness of sentences of a natural language with a free word order [11]. For a comprehensive overview and further details of restarting automata the reader is referred to, e.g., [9].

In systems of parallel communicating restarting automata both concepts are combined, i.e. several restarting automata work together in a parallel way to accept a language. In [13] and [14] first results on these systems and the corresponding language classes were shown, e.g. on the computational power and centralization. Here, results on closure properties of the language classes that are characterized by parallel communicating restarting automata systems are presented.

2. Parallel communicating restarting automata systems

A *restarting automaton* consists of a finite control and a flexible tape that initially contains the input enclosed by the left and the right sentinels ¢ and \$, respectively. To read the tape content a window of a specified size is used. Each computation of an automaton begins in the initial state and the window is positioned on the left end of the tape. Now, the automaton behaves as follows: First, it can move the window several steps to the right and to the left. Then, a rewrite operation is executed, whereby the currently read tape content is replaced by a shorter word. The sentinels are not allowed to be removed. Thereafter, the window can be moved again and finally a restart operation follows. That means, the finite control is reset into the initial state and the window is set to its initial position on the left end of the tape. This sequence of operations is called a *cycle* and can be repeated until either the automaton reaches an *accepting configuration* (denoted by 'Accept') by applying an accept step or it gets stuck. The language that is accepted by a restarting automaton consists of all words over a specified input alphabet for which there exists a computation such that the automaton reaches the accepting configuration.

The most general type of a restarting automaton (as described above) is called RLWW-automaton. If the window is not allowed to be moved to the left, then the automaton is of type RRWW. In case of an RRWW-automaton that has to perform a restart operation immediately after a rewrite operation, it is an RWW-automaton. Moreover, if no auxiliary symbols are allowed to be used, then the automaton is of type RLW, RRW, or RW. And finally, if the currently read tape content is only allowed to be rewritten by a scattered subword, i.e. only symbols are allowed to be removed, then it is of type RL, RR, or R. For the purpose of simplicity the set of all mentioned types of restarting automata is denoted by $\mathcal{T}$. The set of all one-way restarting automata is defined by $\mathcal{T}_R = \{\mathsf{RRWW}, \mathsf{RRW}, \mathsf{RR}, \mathsf{RWW}, \mathsf{RW}, \mathsf{R}\}$. In general, a restarting automaton is nondeterministic. For deterministic automata the prefix 'det-' is used. In some situations the prefix '(det-)' (in brackets) is used to emphasize that it does not matter whether the automaton is deterministic or not.

A *parallel communicating restarting automata system*, PCRA system for short, is a finite collection of restarting automata: $\mathcal{M} = (M_1, M_2, \ldots, M_n)$, whereby n is the degree of the system and $M_i = (Q_i, \Sigma, \Gamma_i, ¢, \$, q_i, k_i, \delta_i)$ for each $i \in \{1, \ldots, n\}$. Here, for all i, $1 \leq i \leq n$, Q_i is the set of states, Σ is the input alphabet, Γ_i is the tape alphabet, ¢ and \$ are the left and right sentinels of the tape, q_i is the initial state, k_i is the window size, and δ_i is the transition mapping. The automata $M_1, M_2, \ldots, M_n$ are also called the components of the system.

A PCRA system works as follows. Initially, each component contains the input word on its tape, starts the computation in its initial state, and behaves like a usual restarting automaton. At one point of the computation a component M_i may need some information from another component M_j. This is signalized by reaching a request state req^j_d, whereby d is a local information that is not transmitted by the component but can be kept during the communication. On the other hand, a component may have some information that it wants to transmit to another particular component. For this purpose, it can switch into a response state $\mathsf{res}^i_{d',c}$, where c denotes the message to be transmitted. In the communication protocol that is used here, all messages are strings of constant length. A communication does only take place, if both communication partners reach the corresponding request and response state, respectively. However, it is not important *when* they reach these states: as long as only one of the two communication partners reaches a request or response state, it just waits for the answer of the other one. In particular, there is no explicit synchronization like a *global clock* that is used in other definitions of PC systems (a global clock is a mechanism that forces all components to execute exactly one computation step in each time unit; such mechanism is avoided for PCRA systems, since it can be seen as a kind of implicit communication). If the communication partner does not reach the corresponding communication state, then the component is blocked for the rest of the system's computation (the system just does not notice this and continues its computation). Otherwise, if both communication partners reach corresponding communication states, then the communication step can be executed. For this, the requesting component M_i is set from state req_d^j into the receive state $\mathsf{rec}_{d,c}^j$ that contains the transmitted message (because it has *received* the requested information from M_j) and the responding component M_j is set from the state $\mathsf{res}_{d',c}^i$ into the acknowledge state $\mathsf{ack}_{d',c}^i$ (the receipt of the message is *acknowledged*). Afterwards, both components continue their local computations.

Formally, for two *configurations* $K = (\kappa_1, \dots, \kappa_n)$ and $K' = (\kappa'_1, \dots, \kappa'_n)$, whereby κ_i and κ'_i are the current configurations of the components, a *computation step* of the system is denoted by $\vdash$ and is defined as follows: $K \vdash K'$ holds if and only if one of the following conditions holds ($u_i v_i$ and $u_j v_j$ are the current tape contents of M_i and M_j, respectively)

1. $\kappa_i \vdash_{M_i} \kappa'_i$ (a *local computation step*)
2. $\exists j \in \{1,2,\dots,n\} \setminus \{i\} : \kappa_i = u_i \mathsf{req}_{d_i}^j v_i, \quad \kappa_j = u_j \mathsf{res}_{d_j,c}^i v_j,$
   $\kappa'_i = u_i \mathsf{rec}_{d_i,c}^j v_i, \ \kappa'_j = u_j \mathsf{ack}_{d_j,c}^i v_j$ (a *communication*)
3. $\exists j \in \{1,2,\dots,n\} \setminus \{i\} : \kappa_i = u_i \mathsf{res}_{d_i,c}^j v_i, \quad \kappa_j = u_j \mathsf{req}_{d_j}^i v_j,$
   $\kappa'_i = u_i \mathsf{ack}_{d_i,c}^j v_i, \ \kappa'_j = u_j \mathsf{rec}_{d_j,c}^i v_j$ (a *communication*)
4. $\kappa_i = \kappa'_i$ and no local operation (MVR, MVL, rewrite, restart) or communication of M_i is possible (M_i *waits*).

Observe that especially the fourth condition provides the idea of a non-synchronized behaviour, since a component can wait arbitrarily long for the execution of a communication step. The reflexive and transitive closure of $\vdash$ is denoted by $\vdash^*$ and describes a *computation*. Then, the language that is accepted by a system $\mathcal{M}$ is defined as

$$L(\mathcal{M}) = \{w \in \Sigma^* \mid (q_1 ¢ w \$, q_2 ¢ w \$, \dots, q_n ¢ w \$) \vdash^* (\kappa_1, \kappa_2, \dots, \kappa_n), \\ \{\kappa_1, \kappa_2, \dots, \kappa_n\} \cap \{\mathsf{Accept}\} \neq \emptyset\}.$$

That means, a system accepts all words for which at least one of the components reaches the accepting configuration; it does not matter which one.

If the components are of type X, $\mathsf{X} \in \mathcal{T}$, then the system is called of type $\mathsf{PC\text{-}X}$. In general, a PCRA system is *nondeterministic*. If all components are deterministic and of type X, then the system is also called *deterministic* and of type $\mathsf{det\text{-}PC\text{-}X}$. The classes of all languages that are accepted by any PCRA system of type $\mathsf{PC\text{-}X}$ or $\mathsf{det\text{-}PC\text{-}X}$ is denoted by $\mathcal{L}(\mathsf{PC\text{-}X})$ or $\mathcal{L}(\mathsf{det\text{-}PC\text{-}X})$, respectively.

A PCRA system is called *centralized* if each component is only allowed to communicate with the first (master) component. In [13, 14] it is shown that centralized PCRA systems have the same computational power as non-centralized systems. Moreover, the centralized system that can be effectively constructed from a non-centralized system accepts if and only if its first component accepts.

3. Closure properties

The following table summarizes the closure properties that we have established so far. The first column contains the language classes that are mainly divided into deterministic and non-deterministic classes and classes with auxiliary symbols and those without. The meaning of the operations are from left to right: union, intersection, intersection with a regular language, complementation, marked product, product, Kleene closure, positive closure, application of arbitrary homomorphisms, application of ε-free (non-erasing) homomorphisms, and inverse homomorphisms. Thereby, '+' means that the language classes are closed under the particular operation, '-' means they are not closed, and in the cases of '?' it is still open.

	$\cup$	$\cap$	$\cap_{REG}$	^{c}	$\cdot_{\#}$	$\cdot$	$*$	$+$	h	h_ε	h^{-1}
$\mathcal{L}$(det-PC-R(R)(W))	$+$	$+$	$+$	$+$	$+$	?	?	?	$-$	?	?
$\mathcal{L}$(det-PC-RL(W))	$+$	$+$	$+$	$+$	$+$	?	?	?	$-$	?	?
$\mathcal{L}$(det-PC-R(R)WW)	$+$	$+$	$+$	$+$	$+$	?	?	?	$-$	?	$+$
$\mathcal{L}$(det-PC-RLWW)	$+$	$+$	$+$	$+$	$+$	?	?	?	$-$	?	$+$
$\mathcal{L}$(PC-R(R)(W))	$+$	$+$	$+$	?	$+$	?	?	?	$-$	?	?
$\mathcal{L}$(PC-RL(W))	$+$	$+$	$+$	?	$+$	?	?	?	$-$	?	?
$\mathcal{L}$(PC-R(R)WW)	$+$	$+$	$+$	?	$+$	$+$	$+$	$+$	$-$	$+$	$+$
$\mathcal{L}$(PC-RLWW)	$+$	$+$	$+$	?	$+$	$+$	$+$	$+$	$-$	$+$	$+$

Most of the results are obtained by straightforward constructions. Whereas the closure under the first three operations result more or less from the definition of PCRA systems and the computational power of individual restarting automata (even the weakest types of restarting automata accept all regular languages), the closure under complementation is based on the bounded number of different configurations of a PCRA system and the fact that, whenever a deterministic system reaches a configuration twice, then it is in a computation loop (and cannot accept the input anymore).

PCRA systems that accept the product of two languages, the Kleene closure, or the positive closure can work in two main phases. First, a factorization of the input is guessed nondeterministically and stored with markers on the tape (for the marked product this step is omitted). Second, the systems for the given languages are simulated on the specified factors and the markers are interpreted as one of the sentinels. Within the first phase all components move their windows simultaneously and write the markers at the same positions. For this, communication and centralization is used. Moreover, setting a marker on the tape means to rewrite two tape symbols because of the length reducing property of restarting automata. The rewritten symbols can be stored within the finite control of the according components. Since it can be shown that the nonforgetting property (i.e. the automaton can change into any state in a restart operation instead of the initial state only) is no advantage for PCRA systems, this can be used to keep the symbols in the finite control even during a restart operation of a component.

To show the closure under non-erasing and inverse homomorphisms an additional property is used: a PCRA system can be defined that behaves exactly like an individual component, merges the tapes of its components to one common tape and thus increases the size of the tape by a constant factor. Thus, subsystems can be defined such that each subsystem represents one component with a bigger tape. That can be used to translate an input into its image or preimage according to a given homomorphism. None of the language classes above is closed under arbitrary homomorphisms, that means homomorphisms that map symbols to the empty word. This follows immediately from the well-known fact that each recursively enumerable language L can be represented by $L = h(L_1 \cap L_2)$, whereby L_1 and L_2 are deterministic context-free languages and h is an arbitrary homomorphism [5]. On the one hand, even individual restarting automata of the weakest types accept all deterministic context-free languages [7] and all language classes of PCRA systems are closed under intersection. Thus, if for any type of PCRA systems the corresponding language class would be closed under arbitrary homomorphisms, then this kind of PCRA systems can accept all recursively enumerable languages. On the other hand, the available work space of a PCRA system is linear bounded, i.e. each language of any PCRA system is included in the set of context-sensitive languages that is again a proper subset of the recursively enumerable languages. Hence, we have a contradiction.

A consequence of the obtained closure properties is that the three language classes $\mathcal{L}(\mathsf{PC\text{-}RLWW})$, $\mathcal{L}(\mathsf{PC\text{-}RRWW})$, and $\mathcal{L}(\mathsf{PC\text{-}RWW})$ are so-called AFLs (abstract families of languages) but they are not full AFLs [12].

References

[1] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, G. Păun (eds.), *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*. Gordon and Breach Science Publishers, Inc., Newark, NJ, USA, 1994.

[2] E. Csuhaj-Varjú, C. Martín-Vide, V. Mitrana, G. Vaszil, Parallel Communicating Pushdown Automata Systems. *International Journal of Foundations of Computer Science* **11** (2000) 4, 631–650.

[3] E. Czeizler, E. Czeizler, Parallel Communicating Watson-Crick Automata Systems. In: Z. Ésik, Z. Fülöp (eds.), *Automata and Formal Languages, 11th International Conference, AFL 2005, Dobogókő, Hungary, May 17-20*. Institute of Informatics, University of Szeged, 2005, 83–96.

[4] M. Goehring, PC-Systems of Restarting Automata. In: J. Mielke, L. Staiger, R. Winter (eds.), *Theorietag Automaten und Formale Sprachen 2009*. Universität Halle-Wittenberg, Institut für Informatik, 2009, 26–27. Technical Report 2009/03.

[5] M. A. Harrison, *Introduction to Formal Language Theory*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1978.

[6] P. Jančar, F. Mráz, M. Plátek, J. Vogel, Restarting Automata. In: H. Reichel (ed.), *Fundamentals of Computation Theory*. Lecture Notes in Computer Science 965, Springer Berlin / Heidelberg, 1995, 283–292.

[7] P. JANČAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, On Restarting Automata with Rewriting. In: G. PĂUN, A. SALOMAA (eds.), *New Trends in Formal Languages*. Lecture Notes in Computer Science 1218, Springer Berlin / Heidelberg, 1997, 119–136.

[8] C. MARTÍN-VIDE, A. MATEESCU, V. MITRANA, Parallel Finite Automata Systems Communicating by States. *International Journal of Foundations of Computer Science* **13** (2002) 5, 733–749.

[9] F. OTTO, Restarting Automata. In: Z. ÉSIK, C. MARTÍN-VIDE, V. MITRANA (eds.), *Recent Advances in Formal Languages and Applications*. Studies in Computational Intelligence 25, Springer Berlin / Heidelberg, 2006, 269–303.

[10] G. PĂUN, L. SANTEAN, Parallel communicating grammar systems: the regular case. *Analele Universitatii din Bucuresti, Seria matematica-informatica* **2** (1989), 55–63.

[11] M. PLÁTEK, M. LOPATKOVÁ, K. OLIVA, Restarting Automata: Motivations and Applications. In: M. HOLZER (ed.), *Workshop 'Petrinetze' and 13. Theorietag 'Formale Sprachen und Automaten'*. Technische Universität München, 2003, 90–96.

[12] G. ROZENBERG, A. SALOMAA (eds.), *Handbook of Formal Languages, vol. 1: Word, Language, Grammar*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[13] M. VOLLWEILER, Centralized Versus Non-Centralized Parallel Communicating Systems of Restarting Automata. In: F. OTTO, N. HUNDESHAGEN, M. VOLLWEILER (eds.), *20. Theorietag Automaten und Formale Sprachen 2010*. Number 2010/3 in Kasseler Informatikschriften, Fachbereich Elektrotechnik / Informatik, Universität Kassel, 2010, 130–135. http://nbn-resolving.de/urn:nbn:de:hebis:34-2010110534894.

[14] M. VOLLWEILER, F. OTTO, Systems of Parallel Communicating Restarting Automata. In: R. FREUND, M. HOLZER, B. TRUTHE, U. ULTES-NITSCHE (eds.), *4th Workshop on Non-Classical Models for Automata and Applications*. Österreichische Computer Gesellschaft, 2012.

F. Mráz (Ed.): Theorietag 2012, Prague, 3. – 5.10.2012
Faculty of Mathematics and Physics, Charles University in Prague pp. 139 – 142

Regular Ideal Languages and Their Boolean Combinations

Franz Jahn Manfred Kufleitner Alexander Lauser

FMI, University of Stuttgart, Germany

jahnfz@studi.informatik.uni-stuttgart.de
{kufleitner, lauser}@fmi.uni-stuttgart.de

Abstract

We consider ideals and Boolean combinations of ideals. For the regular languages within these classes we give expressively complete automaton models. In addition, we consider general properties of regular ideals and their Boolean combinations. These properties include effective algebraic characterizations and lattice identities.

In the main part of this extended abstract, we consider the following deterministic one-way automaton models: unions of flip automata, weak automata, and Staiger-Wagner automata. We show that each of these models is expressively complete for regular Boolean combination of right ideals. Right ideals over finite words resemble the open sets in the Cantor topology over infinite words. An omega-regular language is a Boolean combination of open sets if and only if it is recognizable by a deterministic Staiger-Wagner automaton; our result can be seen as a finitary version of this classical theorem. In addition, we also consider the canonical automaton models for right ideals, prefix-closed languages, and factorial languages.

Finally, we consider a two-way automaton model which is known to be expressively complete for two-variable first-order logic. We show that the above concepts can be adapted to these two-way automata such that the resulting languages are the right ideals (resp. prefix-closed languages, resp. Boolean combinations of right ideals) definable in two-variable first-order logic.

The results of this extended abstract were already presented at CIAA 2012, *cf.* [3].

The Cantor topology over infinite words is an important concept for classifying languages over infinite words. For example, an ω-regular language is deterministic if and only if it is a countable intersection of open sets, *cf.* [11, Remark 5.1]. There are many other properties of ω-languages which can be described using the Cantor topology, see *e.g.* [7, 8]. Ideals are the finitary version of open sets in the Cantor topology. A subset P of a monoid M is a right (resp. left, two-sided) *ideal* if $PM \subseteq P$ (resp. $MP \subseteq P$, $MPM \subseteq P$). In particular, a language $L \subseteq A^*$ is a right ideal if $LA^* \subseteq L$. A *filter* is the complement of an ideal. Thus over finite words, a language $L \subseteq A^*$ is a right filter if and only if it is *prefix-closed*, *i.e.*, if $uv \in L$ implies $u \in L$. Prefix-closed languages correspond to closed sets in the Cantor topology. A language $L \subseteq A^*$ is a two-sided filter if and only if it is *factorial* (also known as *factor-closed* or *infix-closed*), *i.e.*, if $uvw \in L$ implies $v \in L$. Our first series of results gives effective algebraic

[†] The last two authors were supported by the German Research Foundation (DFG) under grant DI 435/5-1.

characterizations of right (resp. left, two-sided) ideal languages and of Boolean combinations of such languages. In addition, we give lattice identities for each of the resulting language classes. As a byproduct, we show that a language is both regular and a Boolean combination of right (resp. left, two-sided) ideals if and only if it is a Boolean combination of regular right (resp. left, two-sided) ideals, *i.e.*, if I is the class of right (resp. left, two-sided) ideals and REG is the class of regular languages, then $\text{REG} \cap \mathbb{B}I = \mathbb{B}(\text{REG} \cap I)$. Here, $\mathbb{B}$ denotes the Boolean closure.

The second contribution consists of expressively complete (one-way) automaton models for right ideals, prefix-closed languages, factorial languages, and Boolean combinations of right ideals. The results concerning ideals and closed languages are straightforward and stated here only to draw a more complete picture. Our main original contribution are automaton models for regular Boolean combinations of right ideals. We always assume that every state in an automaton is reachable from some initial state, *i.e.*, all automata in this paper are accessible.

- A *flip automaton* is an automaton with no transitions from final states to non-final states, *i.e.*, it "flips" at most once from a non-final to a final state. Consequently, every minimal complete flip automaton has at most one final state which has a self-loop for each letter of the alphabet. Paz and Peleg have shown that if a language L is recognized by a complete deterministic automaton $\mathscr{A}$, then L is a right ideal if and only if $\mathscr{A}$ is a flip automaton [6]. A language is a regular Boolean combination of right ideals if and only if it is recognized by a union of flip automata (which do not have to be complete).

- An automaton is *fully accepting* if all states are final. A word u is rejected in a fully accepting automaton $\mathscr{A}$ if and only if there is no u-labeled path in $\mathscr{A}$ which starts in an initial state. Nondeterministic fully accepting automata are expressively complete for prefix-closed languages. Moreover, if a language L is recognized by a deterministic trim automaton $\mathscr{A}$, then L is prefix-closed if and only if $\mathscr{A}$ is fully accepting.

- A *path automaton* is an automaton $\mathscr{A}$ such that all states are both initial and final, *i.e.*, a word u is accepted by $\mathscr{A}$ if there exists a u-labeled path in $\mathscr{A}$. Both deterministic and nondeterministic path automata recognize exactly the class of regular factorial languages. This characterization can be implicitly found in the work of Avgustinovich and Frid [1].

- An automaton is *weak* if in each strongly connected component either all states are final or all states are non-final. Any run of a weak automaton flips only a bounded number of times between final and non-final states. Nondeterministic weak automata can recognize all regular languages. On the other hand, if a language L is recognized by a deterministic automaton $\mathscr{A}$, then L is a Boolean combination of right ideals if and only if $\mathscr{A}$ is weak. Weak automata have been introduced by Muller, Saoudi, and Schupp [5].

- *Deterministic Staiger-Wagner automata* over infinite words have been used to characterize ω-languages $L \subseteq A^{\omega}$ such that both L and $A^{\omega} \setminus L$ are deterministic [9]. Acceptance of a run in a Staiger-Wagner automaton only depends on the set of states visited by the run (but not on their order or their number of occurrences). We show that, over finite words, deterministic Staiger-Wagner automata are expressively complete for Boolean combinations of right ideals. In particular, deterministic Staiger-Wagner automata and deterministic weak automata accept the same class of languages.

We note that flip automata, fully accepting automata, and weak automata yield effective characterizations of the respective language classes. For example, in order to check whether a deterministic automaton $\mathscr{A}$ recognizes a Boolean combination of right ideals, it suffices to test if $\mathscr{A}$ is weak. Moreover, the above automaton models can easily be applied to subclasses of automata such as counter-free automata [4]. This immediately yields results of the following kind: A regular language L is both star-free and a Boolean combination of right ideals if and only if its minimal automaton is weak and counter-free.

For some classes of languages it is more adequate to use two-way automata. The relation between two-way automata and ideals (resp. closed languages, Boolean combinations of ideals) is more complex than for one-way automata. In the last section, we consider deterministic partially ordered two-way automata (po2dfa). In a *partially ordered automaton*, the transition relation induces a partial ordering of the states, *i.e.*, in each run of a partially ordered automaton, no state is re-entered once it is left. Partially ordered automata are also known as *very weak*, *1-weak*, or *linear* automata. We give restrictions of po2dfa's which define the right ideals (resp. prefix-closed languages, Boolean combinations of right ideals) inside the po2dfa-recognizable languages. The class of languages recognized by po2dfa has a huge number of equivalent characterizations; these include the variety **DA** of finite monoids, two-variable first-order logic, unary temporal logic, unambiguous polynomials, and rankers; see *e.g.* [10, 2]. Some of these characterizations admit natural restrictions which are expressively complete for their ideal (resp. prefix-closed, Boolean combination of ideals) counterparts. We introduce *one-pass flip* po2dfa (resp. *one-pass fully accepting* po2dfa, *one-pass* po2dfa) as expressively complete automaton models for right ideals (resp. prefix-closed languages, Boolean combinations of right ideals) inside the class of po2dfa-recognizable languages. Here, a two-way automaton is *one-pass* if acceptance of the input depends only on the state of the automaton which encounters the right end marker for the first time. The notions of *flip*, *fully accepting*, and *weak* two-way automata are defined similarly to the one-way case. The main challenge for each of the above automaton models is to show closure under union and intersection since standard techniques, such as sequentially executing one automaton after the other, cannot be applied. As a complementary result we have that *weak one-pass two-way* dfa's have the same expressive power as their one-way counterparts, *i.e.*, recognize regular Boolean combinations of right ideals.

References

[1] S. V. AVGUSTINOVICH, A. E. FRID, Canonical decomposition of a regular factorial language. In: D. GRIGORIEV (ed.), *CSR 2006*, vol. 3967 of *LNCS*, pp. 18–22. Springer, 2006.

[2] V. DIEKERT, P. GASTIN, M. KUFLEITNER, A survey on small fragments of first-order logic over finite words. *Int. J. Found. Comput. Sci.*, 19(3):513–548, 2008.

[3] F. JAHN, M. KUFLEITNER, A. LAUSER, Regular ideal languages and their Boolean combinations. In: *CIAA 2012*, vol. 7381 of *LNCS*, pp. 205–216. Springer, 2012. Full version available on-line under `http://arxiv.org/abs/1102.5013`.

[4] R. MCNAUGHTON, S. PAPERT, *Counter-Free Automata.* The MIT Press, 1971.

[5] D. E. MULLER, A. SAOUDI, P. E. SCHUPP, Alternating automata, the weak monadic theory of the tree, and its complexity. In: L. KOTT (ed.), *ICALP 1986*, vol. 226 of *LNCS*, pp. 275–283. Springer, 1986.

[6] A. PAZ, B. PELEG, Ultimate-definite and symmetric-definite events and automata. *J. Assoc. Comput. Mach.*, 12(3):399–410, 1965.

[7] D. PERRIN, J.-É. PIN. *Infinite words*, vol. 141 of *Pure and Applied Mathematics*. Elsevier, 2004.

[8] L. STAIGER, ω-languages. In: A. SALOMAA, G. ROZENBERG (eds.), *Handbook of Formal Languages*, vol. 3, pp. 339–387. Springer, 1997.

[9] L. STAIGER, K. W. WAGNER, Automatentheoretische und automatenfreie Charakterisierungen topologischer Klassen regulärer Folgenmengen. *Elektron. Inform.-verarb. Kybernetik*, 10(7):379–392, 1974.

[10] P. TESSON, D. THÉRIEN, Diamonds are forever: The variety DA. In: GRACINDA GOMES (ed.) et al., *Semigroups, Algorithms, Automata and Languages 2001*, pp. 475–500. World Scientific, 2002.

[11] W. THOMAS, Automata on infinite objects. In: J. VAN LEEUWEN (ed.), *Handbook of Theoretical Computer Science*, ch. 4, pp. 133–191. Elsevier, 1990.

Author Index

22ND THEORIETAG
AUTOMATA AND FORMAL LANGUAGES
OCTOBER 3–5, 2012, PRAGUE
PROCEEDINGS

František Mráz (Ed.)

Published by
MATFYZPRESS
publishing house
of the Faculty of Mathematics And Physics
Charles University in Prague
Sokolovská 83, CZ – 186 75 Praha 8
as the 415. publication

This book was prepared from input files supplied by the authors. No additional corrections of the included articles were made by the editor. This volume was typeset using LaTeX.

Printed by Reproduction center UK MFF
Sokolovská 83, CZ – 186 75 Praha 8

Prague 2012

ISBN 978-80-7378-221-4