Technical Reports Mathematics/Computer Science
FB IV - Department of Computer Science
University of Trier
54296 Trier

Theorietag 2017

Automaten und Formale Sprachen

(Bonn, 18. September – 22. September 2017)

Herausgeber: Henning Fernau

Technical Report No. 17-1 September 2017

Vorwort

Theorietage haben eine lange Tradition bei verschiedenen Fachgruppen der Gesellschaft für Informatik (GI).

Die Fachgruppe *Automaten und Formale Sprachen* (AFS) aus dem Fachbereich *Grundlagen der Informatik* (GInf) innerhalb der GI trifft sich einmal jährlich zu ihrem zweitägigen Theorietag. In diesem Jahr feiert die Reihe ihre 27. Auflage in Bonn. Seit über zwanzig Jahren gibt es auch die Tradition, einen Workshop mit eingeladenen Vorträgen dem eigentlichen AFS-Theorietag voran- oder nachzustellen.

Theorietage haben zum Ziel, insbesondere dem wissenschaftlichen Nachwuchs ein Forum zu bieten, auf dem sie ihre Ergebnisse der nationalen Fachöffentlichkeit vorstellen können. Näheres zur Geschichte dieser Serien findet man im Internetauftritt unserer Fachgruppe.

Als Besonderheit haben wir in diesem Jahr einen zweitägigen Workshop angehängt, auf dem wir uns Gedanken zur Situation in der Lehre der Thematik AFS an den deutschen Hochschulen machen und austauschen werden. Sind die betreffenden (GI-)Vorgaben noch zeitgemäß? Diese und ähnliche Fragen sollen uns bewegen. Nähere Einzelheiten entnehme man bitte dem Tagungsprogramm.

In dem hier vorgelegten Tagungsband finden Sie ein- bis vierseitige Kurzfassungen der „nichteingeladenen“ Vorträge dieses Theorietags. Dieses folgt der Tradition der AFS-Theorietage. Der Tagungsband wird elektronisch zur Verfügung gestellt, soll aber bei Bedarf ein Nachschlagen der vorgestellten Resultate ermöglichen.

Wir danken der Gesellschaft für Informatik für die freundliche Unterstützung. Dadurch wurde es möglich, einen Teil der Reisekosten der eingeladenen Vortragenden abzudecken.

Wir wünschen allen Teilnehmern eine interessante und anregende Tagung sowie einen angenehmen Aufenthalt in Bonn. Hoffentlich finden Sie auch Zeit, diesen geschichtsträchtigen Ort zu erkunden.

Trier, im September 2017 — Henning Fernau
im Namen der gesamten FG-Leitung

P.S.: Da wir die angedeutete curriculare Diskussion nicht nur im nationalen Rahmen führen wollen, werden wir auf diesem Theorietag etliche internationale Gäste erwarten. Deshalb sind die meisten der folgenden Beiträge auch in englischer Sprache abgefasst.

Inhaltsverzeichnis

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 1–4.

(Polarized) Tissue P Systems with Vesicles of Multisets

Artiom Alhazov[(A)] Rudolf Freund[(B)] Sergiu Ivanov[(C)]
Sergey Verlan[(C)]

[(A)] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău, MD-2028, Moldova
artiom@math.md

[(B)] Faculty of Informatics, TU Wien
Favoritenstraße 9–11, 1040 Vienna, Austria
rudi@emcc.at

[(C)] Laboratoire d'Algorithmique, Complexité et Logique,
Université Paris Est Créteil,
61 av. du général de Gaulle, 94010 Créteil, France
sergiu.ivanov@u-pec.fr, verlan@u-pec.fr

Abstract

We consider tissue P systems working on vesicles of multisets with the very simple operations of insertion, deletion, and substitution of single objects. With the whole multiset being enclosed in a vesicle, sending it to a target cell can be indicated in those simple rules working on the multiset. As derivation modes we consider the sequential mode, where exactly one rule is applied in a derivation step, and the set maximal mode, where in each derivation step a non-extendable set of rules is applied. With the set maximal mode, computational completeness can already be obtained with tissue P systems having a tree structure, whereas tissue P systems even with an arbitrary communication structure are not computationally complete when working in the sequential mode. Adding polarizations – -1, 0, 1 are sufficient – allows for obtaining computational completeness even for tissue P systems working in the sequential mode.

1. Introduction and Preliminaries

For a comprehensive overview of different variants of (tissue) P systems and their expressive power we refer the reader to the handbook [4], and for a state of the art snapshot of the domain to the P systems website [6] as well as to the Bulletin series of the International Membrane Computing Society [5].

Very simple biologically motivated operations on strings are the so-called *point mutations*, i.e., *insertion*, *deletion*, and *substitution*. For example, these point mutations are used in *networks of evolutionary processors (NEPs)*. A computation step in a NEP consists of two substeps – performing an evolution step in each cell using point mutations and then redistributing the resulting strings via filters. In *hybrid networks of evolutionary processors* (HNEPs), each language processor performs only one of these operations at a certain position of the strings. For an overview on HNEPs and the best results known so far, we refer the reader to [2].

In *networks of evolutionary processors with polarizations*, each symbol has assigned a fixed integer value; the polarization of a string is computed according to a given evaluation function, and in the communication step the obtained string is moved to any of the connected cells having the same polarization, e,g. see [3].

In this extended abstract, we recall the results from [1] for (polarized) tissue P systems where a multiset is enclosed in a vesicle, point mutations are working on such a multiset in the evolution step, and the vesicle with the resulting multiset is moved from one cell to another one as a whole in the communication step.

Recursively Enumerable Sets. The family of recursively enumerable sets of Parikh sets (vectors of natural numbers) is denoted by $PsRE$.

Point Mutations. For an alphabet V, let $a \to b$ be a rewriting rule with $a, b \in V \cup \{\lambda\}$, and $ab \neq \lambda$; we call such a rule a *substitution rule* if both a and b are different from λ; such a rule is called a *deletion rule* if $a \neq \lambda$ and $b = \lambda$, and it is called an *insertion rule* if $a = \lambda$ and $b \neq \lambda$. The set of all insertion rules, deletion rules, and substitution rules over an alphabet V is denoted by Ins_V, Del_V, and Sub_V, respectively.

Register Machines. Register machines are well-known computationally complete (universal) devices for generating or accepting (computing on) sets of vectors of natural numbers.

A *register machine* is a construct $M = (m, B, l_0, l_h, P)$ where m is the number of registers, B is a set of labels bijectively labeling the instructions in the set P, $l_0 \in B$ is the initial label, and $l_h \in B$ is the final label. A labeled instruction of M in P can be a (non-deterministic) increment of a register r, a conditional decrement of a register r, or the $HALT$ instruction.

In the case when a register machine cannot check whether a register is empty we say that it is partially blind: the registers are increased and decreased by one as usual, but if the machine tries to subtract from an empty register, then the computation aborts without producing any result. $PsPBRM$ denotes the Parikh sets obtained by partially blind register machines.

2. (Polarized) Tissue P Systems with Vesicles of Multisets

A *tissue P systems working on vesicles of multisets* (a *tPV* system for short) is a tuple $\Pi = (L, V, T, R, (i_0, w_0), h)$ where

- L is a set of labels identifying in a one-to-one manner the $|L|$ cells of Π;
- V is the alphabet of the system,
- T is the terminal alphabet of the system,
- R is a set of rules of the form (i, p, j) where $i, j \in L$ and $p \in Ins_V \cup Del_V \cup Sub_V$;
- (i_0, w_0) describes the initial vesicle containing the multiset w_0 in cell i_0;
- h is the (label of the) *output cell*.

The tPV system can work with different derivation modes for applying the rules in R: in the sequential mode (abbreviated *sequ*), in each derivation step, with the vesicle enclosing the multiset w being in cell i, exactly one rule (i,p,j) from R_i is applied, i.e., p is applied to w and the resulting multiset in its vesicle is moved to cell j. In the set maximally parallel derivation mode (abbreviated *smax*), we apply a non-extendable multiset of rules from R_i such that all the evolution rules (i,p,j) in this multiset of rules specify the same target cell j.

In any case, the computation of Π starts with a vesicle containing the multiset w_0 in cell i_0, and the computation proceeds in the underlying derivation mode until one of the following output condition is fulfilled (in any case, we only consider the vesicle having arrived in the output cell h):.

- $halt$: the only condition is that the system halts;
- $term$: the resulting multiset contained in the vesicle to be found in cell h consists of terminal symbols only (yet the system need not have reached a halting configuration).
- $(halt, term)$: both conditions must be fulfilled.

The families of sets of vectors of natural numbers generated by tPV systems working in the derivation mode α and using the output strategy β are denoted by $Ps(tPV,\alpha,\beta)$. If the underlying communication structure is a tree, we omit the t and write $Ps(PV,\alpha,\beta)$.

Theorem 2.1 $PsRE = Ps([t]PV, smax, \beta)$ *for any* $\beta \in \{(halt,term), halt, term\}$.

Theorem 2.2 $PsPBRM \subseteq Ps(PV, sequ, \beta)$ *for any* $\beta \in \{(halt,term), halt, term\}$.

Theorem 2.3 $Ps([t]PV, sequ, term) = PsPBRM$.

In a polarized tissue P system Π working on vesicles of multisets, each cell gets assigned an elementary polarization from $\{-1,0,1\}$; each symbol from the alphabet V also has an integer polarization (again the elementary polarizations $\{-1,0,1\}$ are sufficient), but every terminal symbol from the terminal alphabet has polarization 0.

A *polarized tissue P systems working on vesicles of multisets* (a *ptPV* system for short) is a tuple $\Pi = (L,V,T,R,(i_0,w_0),h,\pi_L,\pi_V,\varphi)$ where

- L is a set of labels identifying in a one-to-one manner the $|L|$ cells of Π;
- V is the polarized alphabet of the system,
- T is the terminal alphabet of the system (the terminal symbols have no polarization, i.e., polarization 0),
- R is a set of rules of the form (i,p,j) where $i,j \in L$ and $p \in Ins_V \cup Del_V \cup Sub_V$;
- (i_0,w_0) describes the initial vesicle containing the multiset w_0 in cell i_0;
- h is the (label of the) *output cell*;
- π_L is the function assigning an integer polarization (from $\{-1,0,1\}$) to each cell;
- π_V is the function assigning an (elementary) integer polarization (from $\{-1,0,1\}$) to each symbol in V;
- φ is the evaluation function yielding an integer value for each multiset.

Given a multiset, we need an evaluation function computing the polarization of the whole multiset from the polarizations of the symbols it contains; here we only use the evaluation function φ which computes the value of a multiset as the sum of the values of the symbols contained in it. Given the result m of this evaluation of the multiset in the vesicle, we apply the

sign function $sign(m)$, which returns one of the values $+1/0/-1$, provided that m is a positive integer / is 0 / is a negative integer, respectively.

A derivation step in a ptPV system (as in (H)NEPs) consists of two substeps – the *evolution step*, with applying the rule(s) from R in the way required by the derivation mode (*sequ* or *smax*), and the *communication step* with sending the vesicle to a cell with the same polarization as the multiset in it. Although in the rules themselves still a target is specified, the vesicle can only move to a cell having the same polarization as the multiset contained in it. As a special additional feature we require that the vesicle must not stay in the current cell even if its polarization would fit (if there is no other cell with a fitting polarization, the vesicle is eliminated from the system). As the terminal symbols have polarization 0, the output cell itself also has to have polarization 0.

In any case, the computation of Π starts with a vesicle containing the multiset w_0 in cell i_0 (obviously, the initial multiset w_0 has to have the same polarization as the initial cell i_0), and the computation proceeds using the underlying derivation mode for the evolution steps until an output condition is fulfilled, which in all possible cases means that the vesicle has arrived in the output cell h. Again we use one of the output strategies $halt$, $term$ and $(halt, term)$.

The families of sets of vectors of natural numbers generated by ptPV systems working in the derivation mode α and using the output strategy β are denoted by $Ps(ptPV_n, \alpha, \beta)$.

Theorem 2.4 $PsRE = Ps(ptPV, sequ, term)$.

References

[1] A. ALHAZOV, R. FREUND, S. IVANOV, S. VERLAN, (Tissue) P Systems with Vesicles of Multisets. In: *Proceedings AFL 2017,* to appear. 2017.

[2] A. ALHAZOV, R. FREUND, V. ROGOZHIN, YU. ROGOZHIN, Computational completeness of complete, star-like, and linear hybrid networks of evolutionary processors with a small number of processors. *Natural Computing* **15** (2016) 1, 51–68. http://dx.doi.org/10.1007/s11047-015-9534-1

[3] R. FREUND, V. ROGOJIN, S. VERLAN, Computational Completeness of Networks of Evolutionary Processors with Elementary Polarizations and a Small Number of Processors. In: G. PIGHIZZINI, C. CÂMPEANU (eds.), *Descriptional Complexity of Formal Systems: 19th IFIP WG 1.02 International Conference, DCFS 2017, Milano, Italy, July 3-5, 2017, Proceedings*. Springer, 2017, 140–151. https://doi.org/10.1007/978-3-319-60252-3_11

[4] GH. PĂUN, G. ROZENBERG, A. SALOMAA (eds.), *The Oxford Handbook of Membrane Computing*. Oxford University Press, Oxford, England, 2010.

[5] Bulletin of the International Membrane Computing Society (IMCS). http://membranecomputing.net/IMCSBulletin/index.php.

[6] The P Systems Website. http://ppage.psystems.eu/.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 5–9.

Operational State Complexity and Decidability of Jumping Finite Automata

Simon Beier Markus Holzer Martin Kutrib

Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
{simon.beier,holzer,kutrib}@informatik.uni-giessen.de

Abstract

We consider jumping finite automata and their operational state complexity and decidability status. Roughly speaking, a jumping automaton is a finite automaton with a non-continuous input. We prove upper bounds on the intersection and complementation. The latter result on the complementation upper bound answers an open problem from [G. J. LAVADO, G. PIGHIZZINI, S. SEKI: Operational State Complexity of Parikh Equivalence, 2014]. Moreover, we correct an erroneous result on the inverse homomorphism closure. Finally, we also consider the decidability status of some problems for jumping finite automata.

1. Introduction

Jumping finite automata were recently introduced in [7] as a machine model for non-local information processing, which is motivated from modern information processing, see, for example, [1]. This non-locality is modeled by a non-continuous input. Roughly speaking, a jumping finite automaton is an ordinary finite automaton, which is allowed to read letters from anywhere in the input string, not necessarily only from the left of the remaining input. Already in [7] quite a large number of questions regarding jumping automata were studied and answered: inclusion relations to well-known formal language families, closure and non-closure results under standard formal language operations, decision problems on jumping finite automata languages, etc. Since then, a series of papers [3, 4, 9, 10] pushed the investigation on jumping finite automata further and obtained results on normalforms for jumping finite automata languages by shuffle expressions, computational complexity of jumping finite automata problems, etc. Nevertheless, still several problems for this new device remain open, such as, for example, questions on the descriptional complexity of operations on jumping finite automata languages.

2. Preliminaries

We define jumping finite automata where the notion from [7] is slightly adapted. A *nondeterministic general finite automaton* is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is the finite set of *states*, Σ is the finite set of *input symbols*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *accepting states*, and $\delta\colon Q \times (\Sigma \cup \{\lambda\}) \to 2^Q$ is the *transition function*. A nondeterministic general

finite automaton is *deterministic* if $\delta(q,a)$ is a singleton set, for every state $q \in Q$ and letter $a \in \Sigma$, and $\delta(q,\lambda) = \emptyset$ for every state $q \in Q$. In this case we simply write $\delta(q,a) = p$ instead of $\delta(q,a) = \{p\}$, assuming that δ is a function from $Q \times \Sigma$ to Q.

One can interpret the general finite automaton A on *configurations* of the form $Q\Sigma^*$ in two ways:

1. As ordinary *finite automaton*: the move relation $\vdash_A$ of A is defined as

$$qw \vdash_A pv \quad \text{iff} \quad w = av, \text{ for } v \in \Sigma^*, a \in \Sigma \cup \{\lambda\}, \text{ and } p \in \delta(q,a).$$

   As usual $\vdash_A^*$ refers to the reflexive transitive closure of $\vdash_A$.

2. As *jumping finite automaton*: the jumping relation $\curvearrowright_A$ of A is defined as

$$qw \curvearrowright_A puv \quad \text{iff} \quad w = uav, \text{ for } u, v \in \Sigma^*, a \in \Sigma \cup \{\lambda\}, \text{ and } p \in \delta(q,a).$$

   As usual $\curvearrowright_A^*$ refers to the reflexive transitive closure of $\curvearrowright_A$.

We obtain the following languages from a general finite automaton:

1. The language accepted by an ordinary finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ is defined as $L_F(A) = \{\, w \in \Sigma^* \mid q_0 w \vdash_A^* q_f, \text{ for some } q_f \in F \,\}$.

2. The language accepted by a jumping finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ is defined as $L_J(A) = \{\, w \in \Sigma^* \mid q_0 w \curvearrowright_A^* q_f, \text{ for some } q_f \in F \,\}$.

If there is no danger of confusion we simply write $L(A)$ instead of $L_F(A)$, for a finite automaton A, and $L_J(B)$, for a jumping finite automaton B.

As usual we write NFA (DFA) for nondeterministic (deterministic) finite automata. Moreover, NJFA (DJFA) is an abbreviation for nondeterministic (deterministic) finite jumping automata. The family of languages accepted by a device of type X is denoted by $\mathscr{L}(\mathsf{X})$. The closure and non-closure results of the family of all languages accepted by jumping finite automata have been obtained in [4, 7] and [9].

3. Operational State Complexity of Jumping Automata

We investigate the operational state complexity of jumping finite automata. In particular, we prove upper bounds on the intersection, complementation, and inverse homomorphism. To do this we use the following strategy for NJFAs: first we interpret the NJFAs that we start with as ordinary NFAs and construct a semilinear presentation of the languages under consideration. Here we use [8, Theorem 4.1]. Then we use a result on the descriptional complexity of our operation applied to semilinear sets from [2]. Finally we convert the resulting semilinear set back to an NFA which we interpret as an NJFA. In most cases we also get nice results for DJFAs with a little more effort. A summary of our results on the operational state complexity of jumping automata can be found in Table 1.

The bound for the complementation answers an open problem from [6]. By proving the bound for inverse homomorphism we correct an erroneous result on the inverse homomorphism closure from [7]. There it is claimed that the family $\mathscr{L}(\text{NJFA})$ is closed under inverse homomorphism, where the proof relies on an analogous construction as for ordinary finite automata,

Operation	Number of states of the resulting automaton	
	$X = D$	$X = N$
XJFA $\cap$ XJFA $\to$ XJFA	$(k \cdot n)^{O(k^5)}$	$(k \cdot n)^{O(k^2)}$
$\Sigma^* \setminus$ XJFA $\to$ DJFA	$2^{k^{O(k \cdot \log(k))} n^{O(k^2 \cdot \log(k))}}$	
h^{-1}(NJFA) $\to$ XJFA	$2^{(k_1 k_2 mn)^{5k_1k_2+k_2^2+O(k_1+k_2)}}$	$(k_1 k_2 mn)^{5k_1k_2+k_2^2+O(k_1+k_2)}$
XJFA $\cap$ XFA $\to$ XJFA	$(k \cdot n)^{O(k^5)}$	$(k \cdot n)^{O(k^2)}$

Table 1: Operational state complexity results for deterministic jumping finite automata (DJFAs) and nondeterministic jumping finite automata (NJFAs) over an input alphabet Σ of size k (k_2 for the operation of inverse homomorphism). For every operation the parameter n is the maximum of the numbers of states of the operand automata. For the operation of inverse homomorphism we have another alphabet Γ of size k_1, a homomorphism $h : \Gamma^* \to \Sigma^*$, and the parameter $m = \max(\{\, |h(a)|_b \mid a \in \Gamma, b \in \Sigma \,\} \cup \{1\})$, where $|h(a)|_b$ stands for the number of appearances of the symbol b in the word $h(a)$. For the last operation (intersection with regular languages) the given bounds are valid whenever the resulting language is accepted by a jumping automaton.

which reads as follows: Let $A = (Q, \Gamma, \delta, q_0, F)$ be an NFA, Σ be an alphabet, and $h : \Sigma^* \to \Gamma^*$ be a homomorphism. Then the automaton $A' = (Q, \Sigma, \delta', q_0, F)$ accepts the language $h^{-1}(L(A))$, where

$$q \in \delta'(p, a) \quad \text{if and only if} \quad pw_a \vdash_A^* q \quad \text{with } w_a = h(a).$$

The same construction is used in [7, Theorem 42] to show the closure of $\mathscr{L}$(NJFA) under inverse homomorphism. However, this construction does not work in general as shown by the following counterexample.

Example 3.1 *Consider the NJFA A with the input alphabet $\Sigma = \{a, b\}$ depicted on the left of Figure 1. It is easy to see that $L(A) = \{\, w \in \Sigma^* \mid |w|_a = |w|_b \,\}$. Set $\Gamma = \{a\}$ and define the*

Figure 1: (Left): NJFA A accepting the set $\{\, w \in \Sigma^* \mid |w|_a = |w|_b \,\}$. (Right): NJFA A' induced from A and the homomorphism $h : \{a\}^* \to \{a, b\}^*$ defined by $h(a) = ba$ by the standard construction on NFAs for the inverse homomorphism closure, accepting the set $\{\lambda\}$.

homomorphism $h : \Gamma^* \to \Sigma^*$ via $h(a) = ba$. Constructing the automaton A' as described above results in the jumping automaton drawn on the right of Figure 1. But then

$$L(A') = \{\lambda\} \neq a^* = h^{-1}(L(A)),$$

which shows the argument to be invalid. Even a more sophisticated construction taking permutation of words $h(a)$, for $a \in \Sigma$, into account is not properly working.

With our construction the fact that semilinear sets are closed under inverse homomorphism implies that the family $\mathscr{L}$(NJFA) is closed under inverse homomorphism.

4. Decidability and Complexity of Problems Involving Jumping Automata

The language family $\mathscr{L}(\text{NJFA})$ is not closed under intersection with regular languages, but it is decidable if such an intersection belongs to $\mathscr{L}(\text{NJFA})$:

Theorem 4.1 *Let A be an NJFA and B be an NFA. Then it is decidable whether the language $L(A) \cap L(B)$ is accepted by an NJFA.*

As an immediate consequence we obtain the following result:

Corollary 4.2 *Let A be an NFA. Then it is decidable whether the language $L(A)$ is closed under permutation.*

An NP lower bound on the permutation problem was obtained earlier in [4], if an NFA is given. Next an NP upper bound is given for DFAs.

Theorem 4.3 *Let A be a DFA. Then the problem to decide whether $L(A)$ is closed under permutation is in* coNP. *If the size of the input alphabet is fixed, then the problem to decide whether $L(A)$ is closed under permutation is in* P.

The regularity problem for NJFAs is decidable, which can be seen by a result of [5] on semilinear sets. The lower bound on this problem is NP-hardness as recently shown in [4]. Now the question on the descriptional complexity of NJFAs or DJFAs accepting a regular language compared to ordinary finite automata may arise. We can give an exponential lower bound for the conversion:

Theorem 4.4 *For any integer $n \geq 1$, there exists an $(n(n+1)/2)$-state DJFA A with input alphabet $\Sigma_n = \{a_1, a_2, \ldots, a_n\}$ accepting a regular language such that any DFA accepting $L(A)$ needs at least $n!$ states.*

Proof. We use the languages $L_n = \{\, w \in \Sigma_n^* \mid |w|_{a_i} = 0 \bmod i, \text{ for } 1 \leq i \leq n \,\}$ as witnesses. A DJFA A that accepts L_n is depicted in Figure 2. The number of states of A is $1+2+\cdots+n$, which is equal to $n(n+1)/2$.

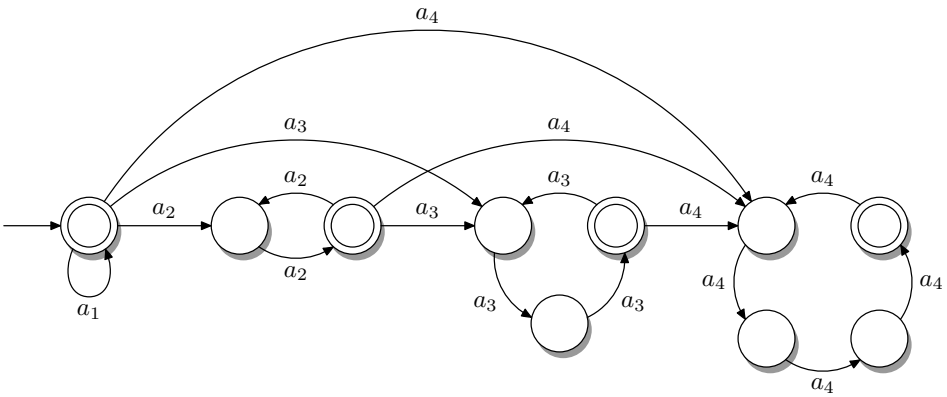

Figure 2: Let $n = 4$. The $(n(n+1)/2)$-state DJFA A with input alphabet $\Sigma_n = \{a_1, a_2, \ldots, a_n\}$ that accepts the regular language $L_n = \{\, w \in \Sigma_n^* \mid |w|_{a_i} = 0 \bmod i, \text{ for } 1 \leq i \leq n \,\}$.

Next, we describe an ordinary DFA $B = (Q, \Sigma_n, \delta, q_0, F)$ accepting the language L_n. Let $Q = \{0\} \times \{0,1\} \times \cdots \times \{0,1,\ldots,n-1\}$, $q_0 = (0,0,\ldots,0)$, $F = \{(0,0,\ldots,0)\}$, and the transition function be

$$\delta((i_1,i_2,\ldots,i_n),a_j) = (i_1,i_2,\ldots,i_{j-1},i_j+1 \bmod j,i_{j+1},\ldots,i_n),$$

for $0 \leq i_k \leq k-1$, for $1 \leq k \leq n$, and $1 \leq j \leq n$. The automaton B accepts L_n and all $n!$ states of B are necessary, as one can see with standard arguments. □

References

[1] Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval: The Concepts and Technology behind Search. Addison-Wesley (2011)

[2] Beier, S., Holzer, M., Kutrib, M.: On the descriptional complexity of operations on semilinear sets. In: Csuhaj-Varjú, E., Dömösi, P., Vaszil, G. (eds.) Proceedings of the 15th International Conference Automata and Formal Languages, EPTCS (2017)

[3] Fernau, H., Paramasivan, M., Schmid, M.L.: Jumping finite automata: Characterizations and complexity. In: Drewes, F. (ed.) Proceedings of the 20th Conference on Implementation and Application of Automata. pp. 89–101. No. 9223 in LNCS, Springer (2015)

[4] Fernau, H., Paramasivan, M., Schmid, M.L., Vorel, V.: Characterization and complexity results on jumping finite automata. Theoret. Comput. Sci. 679, 31–52 (2017)

[5] Ginsburg, S., Spanier, E.H.: Bounded ALGOL-like languages. Trans. AMS 113, 333–368 (1964)

[6] Lavado, G.J., Pighizzini, G., Seki, S.: Operational state complexity under Parikh equivalence. In: Jürgensen, H., Karhumäki, J., Okhotin, A. (eds.) Proceedings of the 16th International Workshop on Descriptional Complexity of Formal Systems. pp. 294–305. No. 8614 in LNCS, Springer (2014)

[7] Meduna, A., Zemek, P.: Jumping finite automata. Internat. J. Found. Comput. Sci. 23, 1555–1578 (2012)

[8] To, A.W.: Parikh images of regular languages: Complexity and applications. http://arxiv.org/abs/1002.1464v2 (2010)

[9] Vorel, V.: On basic properties of jumping finite automata. http://arxiv.org/abs/1511.08396v2 (2015)

[10] Vorel, V.: Two results on discontinuous input processing. In: Câmpeanu, C., Manea, F., Shallit, J.O. (eds.) Proceedings of the 17th International Workshop on Descriptional Complexity of Formal Systems. pp. 205–216. No. 9777 in LNCS, Springer (2016)

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 10–13.

A Characterization of Completely Reachable Automata

E. A. Bondar and M. V. Volkov

Institute of Natural Sciences and Mathematics
Ural Federal University, Lenina 51, 620000 Ekaterinburg, Russia
bondareug@gmail.com, mikhail.volkov@usu.ru

Abstract

A complete deterministic finite automaton in which every non-empty subset of the state set occurs as the image of the whole state set under the action of a suitable input word is called completely reachable. We characterize completely reachable automata in terms of a certain directed graph.

Recall that a *complete deterministic finite automaton* (DFA) is a triple $\mathscr{A} = \langle Q, \Sigma, \delta\rangle$, where Q and Σ are finite sets called the *state set* and the *input alphabet* respectively, and $\delta\colon Q \times \Sigma \to Q$ is a totally defined map called the *transition function*. Let Σ^* stand for the collection of all finite words over the alphabet Σ, including the empty word. The function δ extends to a function $Q \times \Sigma^* \to Q$ (still denoted by δ) in the following natural way: for every $q \in Q$ and $w \in \Sigma^*$, we set $\delta(q, w) := q$ if w is empty and $\delta(q, w) := \delta(\delta(q, v), a)$ if $w = va$ for some word $v \in \Sigma^*$ and some letter $a \in \Sigma$. Thus, via δ, every word $w \in \Sigma^*$ induces a transformation of the set Q.

Whenever we deal with a fixed DFA, we simplify our notation by suppressing the sign of the transition function; this means that we may introduce the DFA as the pair $\langle Q, \Sigma\rangle$ rather than the triple $\langle Q, \Sigma, \delta\rangle$ and may write $q\,.\,w$ for $\delta(q, w)$ and $P\,.\,w$ for $\{\delta(q, w) \mid q \in P\}$ where P is any non-empty subset of Q.

Given a DFA $\mathscr{A} = \langle Q, \Sigma\rangle$, we say that a non-empty subset $P \subseteq Q$ is *reachable* in $\mathscr{A}$ if $P = Q\,.\,w$ for some word $w \in \Sigma^*$. A DFA is called *completely reachable* if every non-empty subset of its state set is reachable.

Completely reachable automata were introduced and studied in [1]. In particular, [1] contains a sufficient condition for complete reachability which we are going to generalize in the present note. We start with recalling the condition from [1].

If Q is a finite set, we denote by $T(Q)$ the *full transformation monoid* on Q, i.e., the monoid consisting of all transformations $\varphi\colon Q \to Q$. For $\varphi \in T(Q)$, its *defect* $\mathrm{df}(\varphi)$ is defined as the size of the set $Q \setminus Q\varphi$. If $\mathscr{A} = \langle Q, \Sigma\rangle$ is a DFA, the defect $\mathrm{df}(w)$ of a word $w \in \Sigma^*$ with respect to $\mathscr{A}$ is the defect of the transformation of the set Q induced by w.

Supported by the Russian Foundation for Basic Research, grant no. 16-01-00795, the Ministry of Education and Science of the Russian Federation, project no. 1.3253.2017, and the Competitiveness Enhancement Program of Ural Federal University.

Consider a word w of defect 1. For such a word, the set $Q \setminus Q\,.\,w$ consists of a unique state, which is called the *excluded state* for w and is denoted by $\mathrm{excl}(w)$. Further, the set $Q\,.\,w$ contains a unique state p such that $p = q_1\,.\,w = q_2\,.\,w$ for some $q_1 \neq q_2$; this state p is called the *duplicate state* for w and is denoted by $\mathrm{dupl}(w)$. Let $D_1(\mathscr{A})$ stand for the set of all words of defect 1 with respect to $\mathscr{A}$, and let $\Gamma_1(\mathscr{A})$ denote the directed graph having Q as the vertex set and the set

$$E_1 := \{(\mathrm{excl}(w), \mathrm{dupl}(w)) \mid w \in D_1(\mathscr{A})\} \tag{1}$$

as the edge set. Since we consider only directed graphs in this paper, we call them just graphs in the sequel.

Given a graph Γ, a vertex p is said to be *reachable* from a vertex q if there exists a directed path starting at q and terminating at p. The *mutual reachability* relation on Γ consists of all pairs (p,q) of vertices such that either $p = q$ or each of the vertices p and q is reachable from the other. Clearly, the mutual reachability relation is an equivalence on the vertex set of Γ, and it partitions this set into classes of mutually reachable vertices. The subgraphs of Γ induced on these classes are called the *strongly connected components* of Γ and a graph whose mutual reachability relation is universal is said to be *strongly connected*.

Theorem 0.1 ([1]) *If a DFA $\mathscr{A} = \langle Q, \Sigma\rangle$ is such that the graph $\Gamma_1(\mathscr{A})$ is strongly connected, then $\mathscr{A}$ is completely reachable; more precisely, for every non-empty subset $P \subseteq Q$, there is a product w of words of defect 1 such that $P = Q\,.\,w$.*

In Theorem 0.1 and in similar statements below we do not exclude the case when $P = Q$ since we may consider the empty word as the product of the empty set of factors with any prescribed property.

The following example, also taken from [1], demonstrates that the condition of Theorem 0.1 is not necessary.

Example 0.2 *Consider the DFA $\mathscr{E}_3$ with the state set $\{1,2,3\}$ and the input letters $a_{[1]}, a_{[2]}, a_{[3]}, a_{[1,2]}$ that act as follows:*

$$i\,.\,a_{[1]} := \begin{cases} 2 & \text{if } i = 1,2, \\ 3 & \text{if } i = 3; \end{cases} \qquad i\,.\,a_{[2]} := \begin{cases} 1 & \text{if } i = 1,2, \\ 3 & \text{if } i = 3; \end{cases}$$

$$i\,.\,a_{[3]} := \begin{cases} 1 & \text{if } i = 1,2, \\ 2 & \text{if } i = 3; \end{cases} \qquad i\,.\,a_{[1,2]} := 3 \ \text{ for all } \ i = 1,2,3.$$

The graph $\Gamma_1(\mathscr{E}_3)$ is not strongly connected. However, it can be checked by a straightforward computation that the automaton $\mathscr{E}_3$ is completely reachable. (This will also follow from Theorem 0.3 below.)

In order to generalize Theorem 0.1, we first extend the operators $\mathrm{excl}(\_)$ and $\mathrm{dupl}(\_)$ to words with defect > 1. Namely, if $\mathscr{A} = \langle Q, \Sigma\rangle$ is a DFA and $w \in \Sigma^*$ is a word with $\mathrm{df}(w) \geq 1$, we define $\mathrm{excl}(w)$ as the set $Q \setminus Q\,.\,w$ and $\mathrm{dupl}(w)$ as the set $\{p \in Q \mid p = q_1\,.\,w = q_2\,.\,w \ \text{ for some } \ q_1 \neq q_2\}$. If we take the usual liberty of ignoring the distinction between singleton subsets and their elements, then for words of defect 1, the new meanings of $\mathrm{excl}(w)$ and $\mathrm{dupl}(w)$ agree with the definition from [1].

Now we describe an iterative process that assigns to each given DFA $\mathscr{A} = \langle Q, \Sigma \rangle$ a certain "layered" graph $\Gamma(\mathscr{A})$. The process starts with the graph $\Gamma_1(\mathscr{A})$ defined above. If the graph $\Gamma_1(\mathscr{A})$ is strongly connected, then we define $\Gamma(\mathscr{A})$ as $\Gamma_1(\mathscr{A})$ and stop the process. If all strongly connected components of $\Gamma_1(\mathscr{A})$ are singletons, we also set $\Gamma(\mathscr{A}) := \Gamma_1(\mathscr{A})$ and stop. Except for these two extreme cases, we extend the graph $\Gamma_1(\mathscr{A})$ as follows. Let Q_2 be the collection of the vertex sets of all at least 2-element strongly connected components of the graph $\Gamma_1(\mathscr{A})$ and let $D_2(\mathscr{A})$ stand for the set of all words of defect 2 with respect to $\mathscr{A}$. We define $\Gamma_2(\mathscr{A})$ as the graph whose vertex set is $Q \cup Q_2$ and whose edge set is the union of E_1 with the set

$$E_2 := \{(C,p) \in Q_2 \times Q \mid C \supseteq \mathrm{excl}(w),\, p \in \mathrm{dupl}(w) \text{for some } \ w \in D_2(\mathscr{A})\} \tag{2}$$

and the set $I_2 := \{(q,C) \in Q \times Q_2 \mid q \in C\}$ of *inclusion* edges representing the containment relation between the elements of Q and the strongly connected components in Q_2.

Observe that the definition of E_1 in (1) can be easily restated in the form similar to the one of the definition of E_2 in (2):

$$E_1 := \{(q,p) \in Q \times Q \mid \{q\} \supseteq \mathrm{excl}(w),\, p \in \mathrm{dupl}(w) \text{for some } \ w \in D_1(\mathscr{A})\}.$$

Before we proceed with the description of the generic step of our process, we present an intermediate result. Even though it will be superseded by our main theorem, we believe that its proof may help the reader to better understand the intuition behind our construction.

Theorem 0.3 *If a DFA $\mathscr{A} = \langle Q, \Sigma \rangle$ is such that the graph $\Gamma_2(\mathscr{A})$ is strongly connected, then $\mathscr{A}$ is completely reachable; more precisely, for every non-empty subset $P \subseteq Q$, there is a product w of words of defect at most 2 such that $P = Q\,.\,w$.*

Proof. Take an arbitrary non-empty subset $P \subseteq Q$. We prove that P is reachable in $\mathscr{A}$ via a product of words of defect at most 2 by induction on $m := |Q \setminus P|$. If $m = 0$, then $P = Q$ and nothing is to prove as Q is reachable via the empty word. Now let $m > 0$ so that P is a proper subset of Q. We aim to find a subset $R \subseteq Q$ such that $P = R\,.\,w$ for some word w of defect at most 2 and $|R| > |P|$. Then $|Q \setminus R| < m$, and the induction assumption applies to the subset R whence $R = Q\,.\,v$ for some product v of words of defect at most 2. Then $P = Q\,.\,vw$ so that P is reachable as required.

Consider two cases.

Case 1: *There exists an edge $(q,p) \in E_1$ such that $q \in Q \setminus P$ and $p \in P$.*

Since $(q,p) \in E_1$, there is a word w of defect 1 with respect to $\mathscr{A}$ for which q is the excluded state and p is the duplicate state. By the definition of the duplicate state, $p = q_1\,.\,w = q_2\,.\,w$ for some $q_1 \neq q_2$, and since the excluded state q for w does not belong to P, for each state $r \in P \setminus \{p\}$, there exists a unique state $r' \in Q$ such that $r'\,.\,w = r$. Now letting $R := \{q_1, q_2\} \cup \{r' \mid r \in P \setminus \{p\}\}$, we conclude that $P = R\,.\,w$ and $|R| = |P| + 1$.

Case 2: *For every edge $(q,p) \in E_1$, if $q \in Q \setminus P$, then also $p \in Q \setminus P$.*

In this case, it is easy to see that every strongly connected component of the graph $\Gamma_1(\mathscr{A})$ is either contained in P or disjoint with P. Consider the set $\widehat{P} := P \cup \{C \in Q_2 \mid C \subseteq P\}$. It is a proper subset of $Q \cup Q_2$ as P is a proper subset of Q. Since the graph $\Gamma_2(\mathscr{A})$ is strongly connected, there must exist an edge e that connects $(Q \cup Q_2) \setminus \widehat{P}$ with $\widehat{P}$ in the sense that the

head of the edge e belongs to $(Q \cup Q_2) \setminus \widehat{P}$ while the tail of e lies in $\widehat{P}$. Under the condition of Case 2, the edge e cannot belong to E_1. Furthermore, the definition of $\widehat{P}$ eliminates the possibility for e to be an inclusion edge: if $(q,C) \in I_2$ is such that $C \in \widehat{P}$, then $C \subseteq P$ whence $q \in P$. Thus, we conclude that $e \in E_2$, i.e., $e = (C,p)$ where $C \notin \widehat{P}$ and $p \in P$. By the definition of E_2, there exists a word w of defect 2 with respect to $\mathscr{A}$ such that $C \supseteq \mathrm{excl}(w)$ and $p \in \mathrm{dupl}(w)$. By the definition of $\mathrm{dupl}(w)$, there exist some $q_1, q_2 \in Q$ such that $q_1 \neq q_2$ and $q_1 \,.\, w = q_2 \,.\, w = p$. Since $C \notin \widehat{P}$, we have $C \cap P = \varnothing$, whence $\mathrm{excl}(w) \cap P = \varnothing$. Thus, for every state $r \in P \setminus \{p\}$, there is a state $r' \in Q$ such that $r' \,.\, w = r$. Now we can proceed as in Case 1: we set $R := \{q \mid q \,.\, w = p\} \cup \{r' \mid r \in P \setminus \{p\}\}$ and conclude that $P = R \,.\, w$ and $|R| > |P|$ since $q_1, q_2 \in R$. □

Now we return to our iterative definition of the graph $\Gamma(\mathscr{A})$. Suppose that $k > 2$ and the graph $\Gamma_{k-1}(\mathscr{A})$ with the vertex set $Q \cup Q_2 \cup \cdots \cup Q_{k-1}$ and the edge set

$$E_1 \cup E_2 \cup \cdots \cup E_{k-1} \cup I_2 \cup \cdots \cup I_{k-1} \tag{3}$$

has already been defined. If the graph $\Gamma_{k-1}(\mathscr{A})$ is strongly connected, then we define $\Gamma(\mathscr{A})$ as $\Gamma_{k-1}(\mathscr{A})$ and terminate the process. Now suppose that $\Gamma_{k-1}(\mathscr{A})$ is not strongly connected. Given a strongly connected component of $\Gamma_{k-1}(\mathscr{A})$, we define its *rank* as the number of vertices from Q that belong to the component. If all strongly connected components of $\Gamma_{k-1}(\mathscr{A})$ have rank less than k, we also set $\Gamma(\mathscr{A}) := \Gamma_{k-1}(\mathscr{A})$ and stop. Otherwise we define the set Q_k as the collection of the vertex sets of all strongly connected components of rank at least k in the graph $\Gamma_{k-1}(\mathscr{A})$. Let $D_k(\mathscr{A})$ stand for the set of all words of defect k with respect to $\mathscr{A}$. We define $\Gamma_k(\mathscr{A})$ as the graph whose vertex set is $Q \cup Q_2 \cup \cdots \cup Q_{k-1} \cup Q_k$ and whose edge set is the union of the set (3) with the set

$$E_k := \{(C,p) \in Q_k \times Q \mid C \supseteq \mathrm{excl}(w),\, p \in \mathrm{dupl}(w) \text{for some } \; w \in D_k(\mathscr{A})\} \tag{4}$$

and the set

$$I_k := \{(q,C) \in Q \times Q_k \mid q \in C\} \cup \bigcup_{i=2}^{k-1} \{(D,C) \in Q_i \times Q_k \mid D \subseteq C\}$$

of inclusion edges representing the inclusions between the elements of $Q \cup Q_2 \cup \cdots \cup Q_{k-1}$ and the strongly connected components in Q_k.

Our main result is the following theorem.

Theorem 0.4 *If a DFA $\mathscr{A} = \langle Q, \Sigma \rangle$ is such that the graph $\Gamma(\mathscr{A})$ is strongly connected and $\Gamma(\mathscr{A}) = \Gamma_k(\mathscr{A})$, then $\mathscr{A}$ is completely reachable; more precisely, for every non-empty subset $P \subseteq Q$, there is a product w of words of defect at most k such that $P = Q \,.\, w$.*

Its proof will be published elsewhere.

References

[1] E. A. BONDAR, M. V. VOLKOV, Completely Reachable Automata. In: C. CÂMPEANU, F. MANEA, J. SHALLIT (eds.), *Descriptional Complexity of Formal Systems, 18th Int. Conf., DCFS 2016*. LNCS 9777, Springer, 2016, 1–17.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 14–17.

Formal Language Techniques for Space Lower Bounds

Li-Hsuan Chen[(A)] Philipp Kuinke[(B)] Felix Reidl[(C)]
Peter Rossmanith[(B)] Fernando Sánchez Villaamil[(B)]

[(A)]National Chung Cheng University, Taiwan
clh100p@cs.ccu.edu.tw

[(B)]RWTH Aachen University
{kuinke,rossmani,fernando.sanchez}@cs.rwth-aachen.de

[(C)]Royal Holloway, University of London
felix.reidl@gmail.com

Abstract

For a fomally defined model of dynamic programming algorithms, we use a definition of Myhill–Nerode families to prove that for every $\varepsilon > 0$, no dynamic programming algorithm solves 3-COLORING/VERTEX COVER/DOMINATING SET on a tree, path or treedepth decomposition of width/depth k with space bounded by $O((3-\varepsilon)^k \cdot \log^{O(1)} n)$.

1. Introduction

The sentiment that the table size is the crucial factor in the complexity of dynamic programming algorithms is certainly not new (see e.g. [4]), so it seems natural to provide lower bounds to formalize this intuition. Our tool of choice will be a family of boundaried graphs that are distinct under Myhill–Nerode equivalence. The perspective of viewing graph decompositions as an "algebraic" expression of boundaried graphs that allow such equivalences is well- established [1, 2].

To formalize the notion of a dynamic programming algorithm on tree, path and treedepth decompositions, we consider algorithms that take as input a tree-, path- or treedepth decomposition of width/depth s and size n and satisfy the following constraints:

1. They pass a single time over the decomposition in a bottom-up fashion;
2. they use $O(f(s) \cdot \log^{O(1)} n)$ space; and
3. they do not modify the decomposition, including re-arranging it.

While these three constraints might look stringent, they include pretty much all dynamic programming algorithm for hard optimization problems on tree or path decompositions. For that reason, we will refer to this type of algorithms simply as *DP algorithms* in the following.

In order to show the aforementioned space lower bounds, we introduce a simple machine model that models DP algorithms on treedepth decompositions and construct superexponentially large *Myhill– Nerode families* that imply lower bounds for DOMINATING SET, VERTEX COVER/INDEPENDENT SET and 3-COLORABILITY in this algorithmic model. These lower bounds hold as well for tree and path decompositions and align nicely with the space complexity of known DP algorithms. While probably not surprising, we consider a formal proof for what previously were just widely held assumptions valuable. The provided framework should easily extend to other problems. Consequently, any algorithmic benefit of treedepth over pathwidth and treewidth must be obtained by non-DP means. A full version of the claimed results is also available [5].

2. Myhill-Nerode Families

In this section we introduce the basic machinery to formalize the notion of dynamic programming algorithms and how we prove lower bounds based on this notion. To make things easier, we assume that the input graphs are connected, which allows us to presume that the treedepth decomposition is always a tree instead of a forest.

First of all, we need to establish what we mean by *dynamic programming (DP)*. DP algorithms on graph decompositions work by visiting the bags/nodes of the decomposition in a bottom-up fashion (a post-order depth-first traversal), maintaining tables to compute a solution. For decision problems, these algorithms only need to keep at most $\log n$ tables in memory at any given moment (achieved in the case of treewidth by always descending first into the part of the tree decomposition with the greatest number of leaves). We propose a machine model with a read-only tape for the input that can only be traversed once, which only accepts as input decompositions presented in a valid order. This model suffices to capture known dynamic programming algorithms on path, tree and treedepth decompositions. More specifically, given a decision problem on graphs Π and some well-formed instance (G,ξ) of Π (where ξ encodes the non-graph part of the input), let T be a tree, path or treedepth decomposition of G of width/depth k. We fix an encoding $\hat{T}$ of T that lists the separators provided by the decomposition in the order they are normally visited in a dynamic programming algorithm (post-order depth-first traversal of the bag/nodes of a tree/path/treedepth decomposition) and additionally encodes the edges of G contained in a separator using $O(k \log k)$ bits per bag or path. Then $(k,\hat{T},\xi)$ is a well- formed instance of the *DP decision problem* Π_{DP}. Pairing DP decision problems with the following machine model provides us with a way to model DP computation over graph decompositions.

Definition 2.1 (Dynamic programming TM) *A* DPTM M *is a Turing machine with an input read-only tape, whose head moves only in one direction and a separate working tape. It accepts as inputs only well-formed instances of some DP decision problem.*

Any single-pass dynamic programming algorithm that solves a DP decision problem on tree, path or treedepth decompositions of width/depth k using tables of size $f(k)$ that does not rearrange the decomposition can be translated into a DPTM with a working tape of size $O(f(k) \cdot \log n)$. This model does not suffice to rule out algebraic techniques, since this technique, like

branching, requires to visit every part of the decomposition many times [3]; or algorithms that preprocess the decomposition first to find a suitable traversal strategy.

An *s-boundaried graph* $\mathcal{G}$ is a graph G with a set $bd(\mathcal{G}) \subseteq V(G)$ of s distinguished vertices labeled 1 through s, called the *boundary* of $\mathcal{G}$. We will call vertices that are not in $bd(\mathcal{G})$ *internal*. By $\mathcal{G}_s$ we denote the class of all s-boundaried graphs. For s-boundaried graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, we let the *gluing* operation $\mathcal{G}_1 \oplus \mathcal{G}_2$ denote the s-boundaried graph obtained by first taking the disjoint union of G_1 and G_2 and then unifying the boundary vertices that share the same label.

The following notion of a *Myhill–Nerode family* will provide us with the machinery to prove space lower-bounds for DPTMs where the input instance is an unlabeled graph and hence for common dynamic programming algorithms on such instances. Recall that $\mathcal{G}_s$ denotes the class of all s-boundaried graphs.

Definition 2.2 (Myhill–Nerode family) *A set $\mathcal{H} \subseteq \mathcal{G}_s \times \mathbf{N}$ is an* s-Myhill–Nerode family *for a DP-decision problem Π_{DP} if the following holds:*

(a) *For every $(\circ H, q) \in \mathcal{H}$ it holds that $|\circ H| = |\mathcal{H}| \cdot \log^{O(1)}|\mathcal{H}|$ and $q = 2^{|\mathcal{H}| \cdot \log^{O(1)}|\mathcal{H}|}$.*

(b) *For every subset $\mathcal{I} \subseteq \mathcal{H}$ there exists an s-boundaried graph $\mathcal{G}_\mathcal{I} \in \mathcal{G}_s$ with $|\mathcal{G}_\mathcal{I}| = |\mathcal{H}| \cdot \log^{O(1)}|\mathcal{H}|$ and an integer $p_\mathcal{I}$ such that for every $(\circ H, q) \in \mathcal{H}$ it holds that*

$$(\mathcal{G}_\mathcal{I} \oplus \circ H, p_\mathcal{I} + q) \notin \Pi_{DP} \iff (\circ H, q) \in \mathcal{I}.$$

Let $\circ\mathbf{td}(\mathcal{G})$ be the minimal depth over all treedepth decompositions of $\mathcal{G} \in \mathcal{G}_s$ where the boundary appears as a path starting at the root. We define the *size* of a Myhill–Nerode family $\mathcal{H}$ as $|\mathcal{H}|$, its *treedepth* as

$$\mathbf{td}(\mathcal{H}) = \max_{(\circ H, \cdot) \in \mathcal{H}, \mathcal{I} \subseteq \mathcal{H}} \circ\mathbf{td}(\mathcal{G}_\mathcal{I} \oplus \circ H)$$

and its *treewidth* and *pathwidth* as the maximum tree/path decomposition of lowest width of any $(\circ H, \cdot) \in \mathcal{H}$ where the boundary is contained in every bag.

The space lower bounds all follow the same basic construction. We define a problem-specific "state" for the vertices of a boundary set X and construct two boundaried graphs for it: one graph that enforces this state in any (optimal) solution of the respective problem and one graph that "tests" for this state by either rendering the instance unsolvable or increasing the costs of an optimal solution. This yields lower bounds for 3-COLORING, VERTEX COVER and DOMINATING SET.

Theorem 2.3 *For every $\varepsilon > 0$, no DPTM solves* 3-COLORING/VERTEX COVER/DOMINATING SET *on a tree, path or treedepth decomposition of width/depth k with space bounded by $O((3-\varepsilon)^k \cdot \log^{O(1)} n)$.*

3. Conclusion

We have shown that single-pass dynamic programming algorithms on treedepth, tree or path decompositions without preprocessing of the input must use space exponential in the width/depth, confirming a common suspicion and proving it rigorously for the first time. This complements

previous SETH-based arguments about the running time of arbitrary algorithms on low treewidth graphs. Our lower bounds appear as if they could be special cases of a general theory to be developed in future work and we further ask whether our result can be extended to less stringent definitions of "dynamic programming algorithms."

References

[1] H. L. BODLAENDER, Fixed-parameter tractability of treewidth and pathwidth. In: *The Multivariate Algorithmic Revolution and Beyond*. 2012, 196–227.

[2] R. BORIE, R. PARKER, C. TOVEY, Solving problems on recursively constructed graphs. *ACM Computing Surveys* **41** (2008) 1, 4.

[3] M. FÜRER, H. YU, Space Saving by Dynamic Algebraization Based on Tree-Depth. *Theory of Computing Systems* (2017), 1–22.

[4] J. V. ROOIJ, H. L. BODLAENDER, P. ROSSMANITH, Dynamic programming on tree decompositions using generalised fast subset convolution. In: *Algorithms – ESA 2009: 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*. Number 5193 in LNCS, Springer, 2009, 566–577.

[5] F. SÁNCHEZ VILLAAMIL, *About Treedepth and Related Notions*. Ph.D. thesis, RWTH Aachen University, 2017.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 18–22.

A Normal Form, a Representation Theorem, and a Regular Approximation for Context-Free Languages

Liliana Cojocaru Erkki Mäkinen

University of Tampere, Faculty of Natural Sciences
{Liliana.Cojocaru, Erkki.Makinen}@uta.fi

Zusammenfassung

Wir stellen eine neue Normalform, genannt *Dyck-Normalform* (DNF), für *kontextfreie Grammatiken* vor. Mit Hilfe dieser Normalform beweisen wir, dass für jede kontextfreie Sprache L, eine ganze Zahl K und ein Homomorphismus φ existieren, so dass $L = \varphi(D'_K)$ wobei D'_K eine bestimmte Teilmenge der Dyck-Sprache über K Buchstaben ist. Die Sprache D'_K ist dabei abhängig von einer kontextfreien Grammatik G in DNF für die Sprache L. Weiterhin nutzen wir ein *übergangsartiges Diagramm* für G, um eine reguläre Sprache R zu konstruieren, so dass $D'_K{=}R \cap D_K$, womit wir den *Chomsky-Schützenberger-Satz* erhalten. Die verwendete Methode zur Konstruktion von R ist grafisch konstruktiv und liefert eine reguläre Grammatik, die eine Obermengen-Annäherung der anfänglichen Sprache erzeugt.

1. Introduction

A *normal form* for context-free grammars (CFGs) consists of restrictions imposed to the structure of context-free (CF) rules, especially on the number and order of terminals and nonterminals allowed on the right-hand side of a CF rule. A normal form is correct if it preserves the language generated by the original grammar. This condition is called *the weak equivalence*, i.e., a normal form preserves the language but may lose important syntactical or semantical properties of the original grammar. The more properties a normal form preserves, the *stronger* it is. Normal forms turned out to be useful tools in parsing theory, inference and learning theory, structural and descriptional complexity.

This paper is devoted to a new normal form for CFGs, called *Dyck normal form* (DNF). The DNF is a syntactical restriction of the Chomsky normal form (CNF), in which the two nonterminals occurring on the right-hand side of a rule are paired nonterminals. This pairwise property induces a nested structure on the derivation tree of a word generated by a grammar in DNF, i.e., each derivation tree of a word generated by a grammar in DNF, read in depth-first search order, is a Dyck word. Moreover, there exists a homomorphism between the derivation tree of a word generated by a grammar in CNF and its equivalent in DNF, i.e., the DNF and CNF are *strongly equivalent*. For a CFG G in DNF we define the *trace language* associated with derivations in G, which is the set of all derivation trees of G read in the depth-first search order. By exploiting the DNF we prove, in Section 2, that for each CF language L, generated by a grammar G in DNF, there exist an integer K and a homomorphism φ such that $L = \varphi(D'_K)$,

where D'_K (a subset of the Dyck language over K letters) equals the trace language associated with G. This representation theorem emerges through a *transition-like diagram* to the *Chomsky-Schützenberger* (C-S) theorem, i.e, we build a regular language R such that $D'_K = R \cap D_K$. As an application, in Section 3 we build a transition diagram for a regular grammar that generates a *regular superset approximation* of the initial CF language. We illustrate, through several examples, the manner in which the regular languages provided by C-S theorem and regular approximation can be built. Even if we reach the same famous C-S theorem, the method used to approach it is different from the other methods known in the literature. In brief, the method in [2] is based on pushdown approaches, while that in [1] uses equations on languages and algebraic techniques to derive several types of Dyck language generators for CF languages. In these works, the Dyck language is somehow hidden behind the derivative structure of the CF language. The Dyck language provided in this paper is merely found through a *pairwise-renaming procedure* of the nonterminals in the original CFG. Hence, it lies inside the CFG it describes.

2. Dyck Normal Form, Dyck Languages, and the Chomsky-Schützenberger Theorem

Definition 2.1 A context-free grammar $G = (N, T, P, S)$ is said to be in *Dyck normal form* (DNF) if it satisfies the following conditions:

1. G is in Chomsky normal form,
2. if $A \to a \in P$, $A \in N$, $A \neq S$, $a \in T$, then no other rule in P rewrites A,
3. for any $A \in N$, $X \to AB \in P$ ($X \to BA$) there is no other rule in P of the form $X' \to B'A$ ($X' \to AB'$),
4. for any two rules $X \to AB$, $X' \to A'B$ ($X \to AB$, $X' \to AB'$) we have $A' = A$ ($B' = B$).

Note that, condition 2 (Definition 2.1) allows to make a partition between nonterminals rewritten by nonterminals, and nonterminals rewritten by terminals. This is useful when defining a homomorphism from Dyck words to words generated by a grammar in DNF. Conditions 3 and 4 allow to split the set of nonterminals into paired nonterminals, and thus to introduce bracketed pairs.

Theorem 2.2 *For each CFG $G = (N, T, P, S)$ there exists a grammar $G' = (N', T, P', S)$ such that $L(G) = L(G')$ and G' is in DNF.*

Definition 2.3 A Dyck language D_k over k letters is a CF language defined by $G_k = (\{S\}, T_k, P, S)$, where $T_k = \{[_1, [_2, ..., [_k,]_1,]_2, ...,]_k\}$ and $P = \{S \to [_i S]_i, S \to SS, S \to [_i]_i | 1 \leq i \leq k\}$.

Definition 2.4 Let $G_k = (N_k, T, P_k, S)$ be a CFG in DNF with $|N_k - \{S\}| = 2k$. Let $D : S \Rightarrow u_1 \Rightarrow u_2 \Rightarrow ... \Rightarrow u_{2n-1} = w$, $n \geq 2$, be a leftmost derivation of $w \in L(G)$. The *trace-word* of a word w of length $n \geq 2$, associated with the derivation D, denoted by $t_{w,D}$, is defined as the concatenation of nonterminals consecutively rewritten in D, excluding the axiom. The *trace-language* Ł(G_k) associated with G_k is Ł$(G_k) = \{t_{w,D} | D$ is a leftmost derivation of $w, w \in L(G_k)\}$.

Note that $t_{w,D}$, $w \in L(G)$, can also be read from the derivation tree of w in the depth-first search order starting with the root (but omitting the root and the leaves). To be observed that Ł(G_k) is a subset of D_k.

Theorem 2.5 *Given a CFG G there exist an integer K, a homomorphism φ, and a subset D'_K of the Dyck language D_K, such that $L(G) = \varphi(D'_K)$. Furthermore, if G has no rules of the form $S \to t$, where t is a terminal, then D'_K equals the trace-language associated with G in DNF. Otherwise, D'_K is a very small extension of the trace-language of grammar G in DNF.*

Proof. Let G be a CFG and $G_k = (N_k, T, P_k, S)$ be the DNF of G such that $N_k = \{S, [_1, [_2, ..., [_k,]_1,]_2, ...,]_k\}$. Let $Ł(G_k)$ be the trace-language associated with G_k. Let $\{t_{k+1}, ..., t_{k+p}\}$ be the ordered subset of T, such that $S \to t_{k+i} \in P$, $1 \le i \le p$. Define $N_{k+p} = N_k \cup \{[_{t_{k+1}}, ..., [_{t_{k+p}},]_{t_{k+1}}, ...]_{t_{k+p}}\}$, and $P_{k+p} = (P_k - \{S \to t_{k+i} \in P | 1 \le i \le p\}) \cup \{S \to [_{t_{k+i}}]_{t_{k+i}}, [_{t_{k+i}} \to t_{k+i},]_{t_{k+i}} \to \lambda | S \to t_{k+i} \in P, 1 \le i \le p\}$. The new grammar $G_{k+p} = (N_{k+p}, T, P_{k+p}, S)$ has the property that $L(G_{k+p}) = L(G_k)$. Let $\varphi : (N_{k+p} - \{S\})^* \to T^*$ be the homomorphism defined by $\varphi(N) = \lambda$, for each rule $N \to XY$, $N, X, Y \in N_k - \{S\}$, $\varphi(N) = t$, for each rule $N \to t$, $N \in N_k - \{S\}$, and $t \in T \cup \{\lambda\}$, $\varphi([_{k+i}) = t_{k+i}$, and $\varphi(]_{k+i}) = \lambda$, for each $1 \le i \le p$. Obviously, $L = \varphi(D'_K)$, where $K = k + p$, $D'_K = Ł(G_k) \cup L_p(G_k)$, and $L_p(G_k) = \{[_{t_{k+1}}]_{t_{k+1}}, ..., [_{t_{k+p}}]_{t_{k+p}}\}$. If G has no rule of the form $S \to t$, $t \in T \cup \{\lambda\}$, then $L_p(G_k) = \emptyset$ and $D'_K = Ł(G_k)$. □

Next grammar G_{k+p} is called the *extended grammar* of G_k. G_k has an extended grammar if and only if G_k (or G) has rules of the form $S \to t$, $t \in T \cup \{\lambda\}$. Let $G_k = (N_k, T, P_k, S)$ be a CFG in DNF. We divide $N_k = \{S, [_1, ..., [_k,]_1, ...,]_k\}$ into three sets $N^{(1)}$, $N^{(2)}$, $N^{(3)}$ as follows:

1. $[_i,]_i \in N^{(1)}$ if and only if $\varphi([_i) = t$ and $\varphi(]_i) = t'$, $t, t' \in T$,
2. $[_i,]_i \in N^{(2)}$ if and only if $\varphi([_i) = t$ and $\varphi(]_i) = \lambda$, or $\varphi([_i) = \lambda$ and $\varphi(]_i) = t$, $t \in T$,
3. $[_i,]_i \in N^{(3)}$ if and only if $\varphi([_i) = \lambda$ and $\varphi(]_i) = \lambda$.

Certainly, $N_k - \{S\} = N^{(1)} \cup N^{(2)} \cup N^{(3)}$ and $N^{(1)} \cap N^{(2)} \cap N^{(3)} = \emptyset$. $N^{(2)}$ is further divided into $N_l^{(2)}$ and $N_r^{(2)}$, such that $N_l^{(2)}$ contains those bracketed pairs $[_i,]_i \in N^{(2)}$ for which $\varphi([_i) \neq \lambda$, and $N_r^{(2)}$ contains those bracketed pairs $[_i,]_i \in N^{(2)}$ for which $\varphi(]_i) \neq \lambda$. We have $N^{(2)} = N_l^{(2)} \cup N_r^{(2)}$ and $N_l^{(2)} \cap N_r^{(2)} = \emptyset$. To find a connection between C-S theorem and Theorem 2.5 we build some transition diagrams as follows.

Construction 2.6 A *dependency graph* of G_k is a directed graph $\mathcal{G}^X = (V_X, E_X)$, $X \in \{]_j |]_j \in N^{(3)}\} \cup \{S\}$, in which nodes are labeled with variables in $\tilde{N}_k \cup \{X\}$, $\tilde{N}_k = \{[_i | [_i \in N^{(1)} \cup N_r^{(2)} \cup N^{(3)}\} \cup \{]_j |]_j \in N_l^{(2)}\}$ and the set of edges is built as follows. For each $X \to [_i]_i \in P_k$, $]_i \in N_l^{(2)}$, $\mathcal{G}^X$ contains a directed edge from X to $]_i$, for each $X \to [_i]_i \in P_k$, $[_i \in N^{(1)} \cup N_r^{(2)} \cup N^{(3)}$, $\mathcal{G}^X$ contains an edge from X to $[_i$. There is an edge in $\mathcal{G}^X$ from a node $[_i$, $[_i \in N_r^{(2)} \cup N^{(3)}$ to a node $]_j$ or $[_k,]_j \in N_l^{(2)}$, $[_k \in N^{(1)} \cup N_r^{(2)} \cup N^{(3)}$, if there is a rule in P_k of the form $[_i \to [_j]_j$ or of the form $[_i \to [_k]_k$, respectively. There is an edge in $\mathcal{G}^X$ from a node $]_i$, $]_i \in N_l^{(2)}$, to a node $]_j$ or $[_k,]_j \in N_l^{(2)}$, $[_k \in N^{(1)} \cup N_r^{(2)} \cup N^{(3)}$, if there is a rule in P_k of the form $]_i \to [_j]_j$ or of the form $]_i \to [_k]_k$, respectively. X is the *initial node* of $\mathcal{G}^X$. Any node $[_i \in N^{(1)}$ is a *final node* in $\mathcal{G}^X$.

Let $\mathcal{G}^X$ be a dependency graph of G_k. Consider the set of all *terminal path* in $\mathcal{G}^X$ (starting from the initial node to a final node). The set of terminal paths in $\mathcal{G}_X$ can be characterized by a finite number of regular expressions (reg.exps). Denote by $\mathcal{R}_{[_i}^X$ the set of all reg.exps over $\tilde{N}_k \cup \{X\}$ that can be read in $\mathcal{G}^X$, starting from X and ending in the final node $[_i \in N^{(1)}$. Define the homomorphism $h_{\mathcal{G}} : \tilde{N}_k \cup \{X\} \to \{]_i |]_i \in N_r^{(2)} \cup N^{(3)}\} \cup \{\lambda\}$, $h_{\mathcal{G}}([_i) =]_i$ for any $[_i \in$

$N_r^{(2)} \cup N^{(3)}$, $h_{\mathcal{G}}(X) = h_{\mathcal{G}}([_i) = h_{\mathcal{G}}(]_i) = \lambda$, for any $[_i \in N_l^{(2)} \cup N^{(1)}$. For any $r.e_{[_i}^{(l,X)} \in \mathcal{R}_{[_i}^X$ we build a new reg.exp $r.e_{[_i}^{(r,X)} = h_{\mathcal{G}}^r(r.e_{[_i}^{(l,X)})$, where $h_{\mathcal{G}}^r$ is the mirror image of $h_{\mathcal{G}}$. Define $r.e_{[_i}^X = r.e_{[_i}^{(l,X)} r.e_{[_i}^{(r,X)}$, called the *extended reg.exp* of $r.e_{[_i}^{(l,X)}$. For a certain X and $[_i$ denote by $\mathcal{R}.e_{[_i}^X$ the set of all extended reg.exps $r.e_{[_i}^X$ obtained as above. Furthermore, $\mathcal{R}.e^X = \bigcup_{[_i \in N^{(1)}} \mathcal{R}.e_{[_i}^X$ and $\mathcal{R}.e = \mathcal{R}.e^S \cup (\bigcup_{]_i \in N^{(3)}} \mathcal{R}.e^{]_i})$. Connecting all reg.exps in $\mathcal{R}.e$, through brackets $]_i$, where $]_i \in N_r^{(2)} \cup N^{(3)}$ by preserving some rules in G_k we build the *extended dependency graph* of G_k, denoted by $\mathcal{G}_e = (\mathcal{V}_e, \mathcal{E}_e)$, which is a directed graph in which $\mathcal{V}_e = \tilde{N}_k \cup \{S\} \cup \{]_i |\ ]_i \in N_r^{(2)} \cup N^{(3)}\}$, S is the initial node, and final nodes are brackets $]_i$, where $]_i \in N_r^{(2)} \cup N^{(1)}$.

Theorem 2.7 *(Chomsky-Schützenberger)* For each CF language L there exist an integer K, a regular set R, and a homomorphism h, such that $L = h(D_K \cap R)$. Furthermore, if G is a CFG generating L, G_k the DNF of G, and G_k has no extended grammar, then $K = k$ and $D_K \cap R = Ł(G_k)$. Otherwise, there exists $p > 0$ such that $K = k + p$, and $D_K \cap R = D'_K$, where D'_K is the subset of D_K computed as in Theorem 2.5.

Note that, the proof is a direct consequence of the manner in which the extended dependency graph is built. The homomorphism h is equal to φ in the proof of Theorem 2.5. The interpretation that emerges from the graphical method we propose is that *the regular language in the Chomsky-Schützenberger theorem intersected with a (certain) Dyck language lists all derivation trees (read in the depth-first search order) associated with words in a CFG in DNF or in CNF (since these derivation trees are equal, up to a homomorphism).*

Example 2.8 Let $G = (\{S, [_1 ..., [_7,]_1 ...,]_7\}, \{a, b, c\}, S, P)$ in DNF, $P = \{S \to [_1]_1, [_1 \to [_5]_5 / [_1]_1,]_1 \to [_6]_6, [_2 \to [_6]_6 / [_7]_7, [_3 \to [_7]_7, [_5 \to [_4]_4, [_6 \to [_3]_3,]_6 \to [_2]_2,]_7 \to [_3]_3 / [_4]_4,]_2 \to b,]_3 \to a, [_4 \to c,]_4 \to c,]_5 \to b, [_7 \to a\}$.

The sets of reg.exps and extended reg.exps obtained by reading $\mathcal{G}^S$ (Figure 1.a) are $\mathcal{R}_{[_4}^S = \{S[_1^+[_5[_4]_5]_1^+\}$ and $\mathcal{R}.e^S = \mathcal{R}.e_{[_4}^S = \{S[_1^+[_5[_4]_5]_1^+\}$, respectively. The reg.exps and extended reg.exps readable from $\mathcal{G}^{]_1}$ (Figure 1.b) are $\mathcal{R}_{[_4}^{]_1} = \{]_1[_6([_3]_7)^+[_4\}$ and $\mathcal{R}.e^{]_1} = \{]_1[_6([_3]_7)^+[_4(]_3)^+]_6\}$, respectively. The reg.exps and extended reg.exps obtained by reading $\mathcal{G}^{]_6}$ (Figure 1.c) are $\mathcal{R}_{[_4}^{]_6} = \{]_6[_2[_6([_3]_7)^+[_4,]_6[_2(]_7[_3)^*]_7[_4\}$ and $\mathcal{R}.e^{]_6} = \mathcal{R}.e_{[_4}^{]_6} = \{]_6[_2[_6([_3]_7)^+[_4(]_3)^+]_6]_2,]_6[_2(]_7[_3)^*]_7[_4(]_3)^*]_2\}$, respectively. The extended dependency graph $\mathcal{G}_e$ of G is sketched in Figure 1.d. Edges in black, are built from reg.exps in $\mathcal{R}_{[_4}^X$, $X \in \{S,]_1,]_6\}$. Orange edges emphasize symmetrical structures, built with respect to the structure of trace-words. Some of them (e.g. $]_2]_1$ and $]_2]_2$) connect reg.exps in $\mathcal{R}.e$ with respect to the structure of trace-words in $Ł(G)$. Edge $]_2]_1$ is added because there is at least one reg.exps in $\mathcal{R}.e$ that contains $]_1]_1$ (e.g. $S[_1^+[_5[_4]_5]_1^+$), a reg.exps in $\mathcal{R}.e_{[_4}^{]_1}$ that ends in $]_6$ (e.g. $]_1[_6([_3]_7)^+[_4(]_3)^+]_6$) and a reg.exp in $\mathcal{R}.e_{[_4}^{]_6}$ that ends in $]_2$. Edge $]_2]_2$ is due to the existence of a reg.exp that contains $]_6]_2$ (e.g. $]_6[_2[_6([_3]_7)^+[_4(]_3)^+]_6]_2$) and a reg.exp in $\mathcal{R}.e_{[_4}^{]_6}$ that ends in $]_2$ (e.g. $]_6[_2[_6([_3]_7)^+[_4(]_3)^+]_6]_2$ or $]_6[_2(]_7[_3)^*]_7[_4(]_3)^*]_2$). The regular language provided by the C-S theorem is the homomorphic image, through h_k of all reg.exps associated with all terminal paths in $\mathcal{G}_e$ (Figure 1.d) reachable from S to the final node $]_2$, where $h_k : \tilde{N}_k \cup \{]_i |]_i \in N_r^{(2)} \cup N^{(3)}\} \cup \{S\} \to N_k$ is defined by $h_k(S) = \lambda$, $h_k([_i) = [_i$, $h_k(]_i) =]_i$, for $[_i \in N_r^{(2)} \cup N^{(3)}$, $h_k(]_i) = [_i]_i$, $]_i \in N_l^{(2)}$, $h_k([_i) = [_i]_i$, $[_i \in N^{(1)}$.

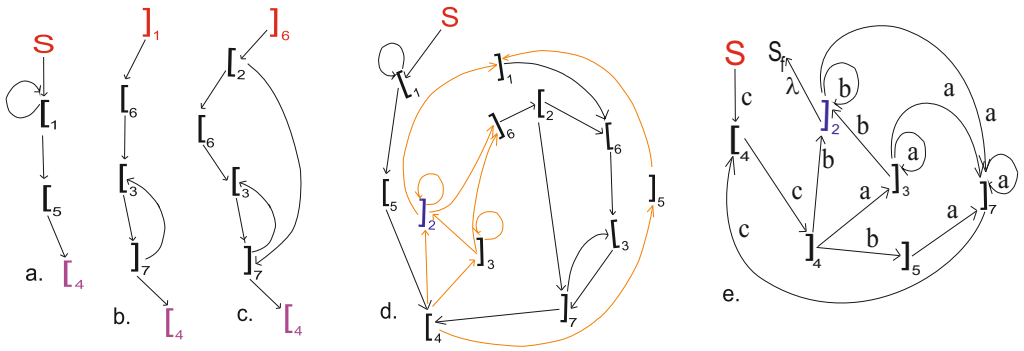

Figure 1: a. - d. The dependency graphs and extended dependency graph of the grammar in Example 2.8. e. The transition diagram $\mathcal{A}_e$ in Example 3.1. The nodes colored in blue or purple are final nodes.

3. Conclusions and Further Applications of the DNF

In this paper we introduced a normal form for CFGs, called *Dyck normal form* (DNF). Based on this normal form and on graphical approaches we give an alternative proof of the *Chomsky-Schützenberger (C-S) theorem*. The regular language in this theorem is obtained from a transition-like diagram called the *extended dependency graph* associated with derivations in a CFG in DNF. Let $G_k = (N_k, T, P_k, S)$ be a CFG in DNF and $\mathcal{G}_e = (\mathcal{V}_e, \mathcal{E}_e)$ the extended dependency graph of G_k. From $\mathcal{G}_e$ we can depict a *state diagram* $\mathcal{A}_e$ for a finite automaton and a regular grammar $G_r = (N_r, T, P_r, S)$ that generates a regular superset approximation for G_k. In brief, this can be obtained by dropping in $\mathcal{G}_e$ all left brackets in $N_r^{(2)}$ and all brackets in $N^{(3)}$, and labeling the edges with the symbols produced in G_k by left or right bracket in $N^{(2)} \cup N^{(1)}$. Based on graphical approaches, $\mathcal{G}_e$ can be further refined and therefore the regular superset approximation can be further adjusted. The method provides a tight approximation especially for linear CF languages. Besides, it preserves much of the structure of the original CFG, since the regular language in the C-S theorem is an approximation of the trace-language, which encodes derivation tress in the original CFG. A further goal is to compare our method with the approximation technique described in [3]. The graphical method used to approach the C-S theorem and the regular approximation we proposed may have further applications in the study of descriptional and complexity properties of (several subclasses of) CF languages.

Example 3.1 The regular grammar generating the regular superset approximation of the CF language in Example 2.8 is $G_r = (\{S,]_2,]_3, [_4,]_4,]_5,]_7\}, \{a, b, c\}, S, P_r)$, $P_r = \{S \to c[_4, [_4 \to c]_4,]_4 \to b]_2/a]_3/b]_5,]_2 \to b]_2/a]_7,]_3 \to a]_3/b]_2/a]_7,]_5 \to a]_7,]_7 \to a]_7/c[_4,]_2 \to \lambda\}$. $L(G_r) = ((c^2ba^+)^*(c^2a^*b^+a^+)^*)^*c^2a^*b^+$. The transition diagram $\mathcal{A}_e$ is sketched in Figure 1.e.

References

[1] J. BERSTEL, *Transductions and Context-Free Languages*. Teuber, 1979.

[2] M. HARRISON, *Introduction to Formal Language Theory*. Addison-Wesley Longman, 1978.

[3] M. N. M. MOHRI, Regular Approximation of Context-free Grammars through Transformation. *Robustness in Language and Speech Technolog* **9** (2000), 251–261.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 23–26.

The Hardness of Solving Simple Word Equations

Joel Day, Florin Manea, Dirk Nowotka

Kiel University,
{jda,flm,dn}@informatik.uni-kiel.de

A *word equation* is an equality $\alpha = \beta$, where α and β are words over an alphabet $\Sigma \cup X$ (called the left, respectively, right side of the equation); $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots\}$ is the alphabet of *constants* and $X = \{x_1, x_2, x_3, \ldots\}$ is the alphabet set of *variables*. A *solution* to the equation $\alpha = \beta$ is a morphism $h : (\Sigma \cup X)^* \to \Sigma^*$ that acts as the identity on Σ and satisfies $h(\alpha) = h(\beta)$. For instance, $\alpha = x_1\mathtt{ab}x_2$ and $\beta = \mathtt{a}x_1x_2\mathtt{b}$ define the equation $x_1\mathtt{ab}x_2 = \mathtt{a}x_1x_2\mathtt{b}$, whose solutions are the morphisms h with $h(x_1) = \mathtt{a}^k$, for $k \geq 0$, and $h(x_2) = \mathtt{b}^\ell$, for $\ell \geq 0$.

The study of word equations (or the existential theory of equations over free monoids) is an important topic found at the intersection of algebra and computer science. The problem of deciding whether a given word equation $\alpha = \beta$ has a solution or not, known as the satisfiability problem, was shown to be decidable by Makanin [12] (see Chapter 12 of [11] for a survey). Later it was shown that the satisfiability problem is in PSPACE by Plandowski [14]; a new proof of this result was obtained in [10], based on a new simple technique called recompression. However, it is conjectured that the satisfiability problem is in NP; this would match the known lower bounds: the satisfiability of word equations is NP-hard, as it follows immediately from, e.g., [4]. This hardness result holds in fact for much simpler classes of word equations, like quadratic equations (where the number of occurrences of each variable in $\alpha\beta$ is at most two), as shown in [3]. There are also cases when the satisfiability problem is tractable. For instance, word equations with only one variable can be solved in linear time in the size of the equation, see [9]; equations with two variables can be solved in time $\mathcal{O}(|\alpha\beta|^5)$, see [2].

In most cases, the NP-hardness of the satisfiability problem for classes of word equations was shown as following from the NP-completeness of the *matching problem* for corresponding classes of patterns with variables. In the matching problem we essentially have to decide whether an equation $\alpha = \beta$, with $\alpha \in (\Sigma \cup X)^*$ and $\beta \in \Sigma^*$, has a solution; that is, only one side of the equation, called pattern, contains variables. The aforementioned results [4, 3] show, in fact, that the matching problem is NP-complete for general α, respectively when α is quadratic. Many more tractability and intractability results concerning the matching problem are known (see [16, 6, 7]). In [5], efficient algorithms were defined for, among others, patterns which are *regular* (each variable has at most one occurrence), *non-cross* (between any two occurrences of a variable, no other distinct variable occurs), or patterns with only a constant number of variables occurring more than once.

Naturally, for a class of patterns that can be matched efficiently, the hardness of the satisfiability problem for word equations with sides in the respective class is no longer immediate. A study of such word equations was initiated in [13], where the following results were obtained. Firstly, the satisfiability problem for word equations with non-cross sides (for short non-cross

equations) remains NP-hard. In particular, solving non-cross equations $\alpha = \beta$ where each variable occurs at most three times, at most twice in α and exactly once in β, is NP-hard. Secondly, the satisfiability of one-repeated variable equations (where at most one variable occurs more than once in $\alpha\beta$, but arbitrarily many other variables occur only once) having at least one non-repeated variable on each side, was shown to be trivially in P.

In this talk, we mainly address the class of regular-ordered equations, whose sides are regular patterns and, moreover, the order of the variables occurring in both sides is the same. This seems to be one of the structurally simplest classes of equations whose number of variables is not bounded by a constant. One central motivation for studying these equations with a simple structure is that understanding their complexity and combinatorial properties may help us to identify a boundary between classes of word equations whose satisfiability is tractable and intractable. Moreover, we wish to gain a better understanding of the core reasons why solving word equations is hard. In the following, we overview our results, methods, and their connection to existing works from the literature.

Lower bounds: Our first result closes the main problem left open in [13]. Namely, we show that it is (still) NP-hard to solve regular (ordered) word equations. Note that in these word equations each variable occurs at most twice: at most once in every side. They are particular cases of both quadratic equations and non-cross equations, so the reductions showing the hardness of solving these more general equations do not carry over. To begin with, matching quadratic patterns is NP-hard, while matching regular patterns can be done in linear time. Showing the hardness of the matching problem for quadratic patterns in [3] relied on a simple reduction from 3-SAT, where the two occurrences of each variable were used to simulate an assignment of a corresponding variable in the SAT formula, respectively to ensure that this assignment satisfies the formula. To facilitate this final part, the second occurrences of the variables were grouped together, so the equation constructed in this reduction was not non-cross. Indeed, matching non-cross patterns can be done in polynomial time. So showing that solving non-cross equations is hard, in [13], required slightly different techniques. This time, the reduction was from an assignment problem in graphs. The (single) occurrences of the variables in one side of the equation were used to simulate an assignment in the graph, while the (two) occurrences of the variables from the other side were used for two reasons: to ensure that the previously mentioned assignment is correctly constructed and to ensure that it also satisfies the requirements of the problem. For the second part it was also useful to allow the variables to occur in one side in a different order as in the other side.

As stated in [13], showing that the satisfiability problem for regular equations seems to require a totally different approach. Our hardness reduction relies on some novel ideas, and, unlike the aforementioned proofs, has a deep word-combinatorics core. As a first step, we define a reachability problem for a certain type of (regulated) string rewriting systems, and show it is NP-complete. This is achieved via a reduction from the strongly NP-complete problem 3-PARTITION [8]. Then we show that this reachability problem can be reduced to the satisfiability of regular-ordered word equations; in this reduction, we essentially try to encode the applications of the rewriting rules of the system into the periods of the words assigned to the variables in a solution to the equation. In doing this, we are able to only use one occurrence of each variable per side, and moreover to even have the variables in the same order in both sides.

Upper bounds: A consequence of the results in [15] is that the satisfiability problem for a certain class of word equations is in NP if the lengths of the minimal solutions of such equations

(where the length of the solution defined by a morphism h is the image of the equation's sides under h) are at most exponential. With this in mind, which obtain an insight in the combinatorial structure of the minimal solutions of quadratic equations: if we follow around the minimal solutions the positions that are fixed inside the images of the variables by each terminal of the original equation (in order, starting with that terminal), we obtain sequences that should not contain repetitions. Consequently, we give a simple and concise proof of the fact that the image of any variable in a minimal solution to a regular-ordered equation is at most linear in the size of the equation. It immediately follows that the satisfiability problem for regular-ordered equations is in NP. While this result was expected, the approach we use to obtain it seems rather interesting to us, and also a promising approach to showing that other, more complicated, classes of restricted word equations can be solved in NP-time. For instance, it is an open problem to show this for arbitrary regular or quadratic equations. It is worth noting that our polynomial upper bound on length of minimal solutions of regular-ordered equations does not hold even for slightly relaxed versions of such equations. More precisely, non-cross equations $\alpha = \beta$ where the order of the variables is the same in both sides and each variable occurs exactly three times in $\alpha\beta$, but never only on one side, may already have exponentially long minimal solutions. To this end, it seems even more surprising that it is NP-hard to solve equations with such a simple structure (regular-ordered), which, moreover, have quadratically short solutions. As such, regular-ordered equations seem to be among the structurally simplest word-equations, whose satisfiability problem is intractable.

Extending our ideas, we settle the complexity of solving regular-ordered equations with regular constraints (as defined in [3], where each variable is associated with an NFA), which is in NP for regular-ordered equations whose sides contain exactly the same variables, or when the languages defining the scope of the variables are all accepted by NFAs with at most c states, where c is a constant. For regular-ordered equations with regular constraints without these restrictions, the problem remains PSPACE-complete.

These results appeared in [1].

References

[1] J. D. DAY, F. MANEA, D. NOWOTKA, The Hardness of Solving Simple Word Equations. *CoRR* **abs/1702.07922** (2017). To appear in MFCS 2017.

[2] R. DĄBROWSKI, W. PLANDOWSKI, Solving two-variable word equations. In: *Proc. 31th International Colloquium on Automata, Languages and Programming, ICALP 2004*. Lecture Notes in Computer Science 3142, 2004, 408–419.

[3] V. DIEKERT, J. M. ROBSON, On Quadratic Word Equations. In: *Proc. 16th Annual Symposium on Theoretical Aspects of Computer Science, STACS 1999*. Lecture Notes in Computer Science 1563, 1999, 217–226.

[4] A. EHRENFEUCHT, G. ROZENBERG, Finding a Homomorphism Between Two Words is NP-Complete. *Information Processing Letters* **9** (1979), 86–88.

[5] H. FERNAU, F. MANEA, R. MERCAŞ, M. SCHMID, Pattern Matching with Variables: Fast Algorithms and New Hardness Results. In: *Proc. 32nd Symposium on Theoretical Aspects of Computer*

Science, STACS 2015. Leibniz International Proceedings in Informatics (LIPIcs) 30, 2015, 302–315.

[6] H. FERNAU, M. L. SCHMID, Pattern matching with variables: A multivariate complexity analysis. *Information and Computation* **242** (2015), 287–305.

[7] H. FERNAU, M. L. SCHMID, Y. VILLANGER, On the Parameterised Complexity of String Morphism Problems. *Theory of Computing Systems* (2015). Http://dx.doi.org/10.1007/s00224-015-9635-3.

[8] M. R. GAREY, D. S. JOHNSON, *Computers And Intractability*. W. H. Freeman and Company, 1979.

[9] A. JEŻ, One-Variable Word Equations in Linear Time. *Algorithmica* **74** (2016), 1–48.

[10] A. JEŻ, Recompression: A Simple and Powerful Technique for Word Equations. *Journal of the ACM* **63** (2016).

[11] M. LOTHAIRE, *Algebraic Combinatorics on Words*. Cambridge University Press, Cambridge, New York, 2002.

[12] G. S. MAKANIN, The problem of solvability of equations in a free semigroup. *Matematicheskii Sbornik* **103** (1977), 147–236.

[13] F. MANEA, D. NOWOTKA, M. L. SCHMID, On the Solvability Problem for Restricted Classes of Word Equations. In: *Proc. 20th International Conference on Developments in Language Theory, DLT 2016*. Lecture Notes in Computer Science 9840, Springer, 2016, 306–318.

[14] W. PLANDOWSKI, An efficient algorithm for solving word equations. In: *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, STOC 2006*. 2006, 467–476.

[15] W. PLANDOWSKI, W. RYTTER, Application of Lempel-Ziv Encodings to the Solution of Words Equations. In: *Proc. 25th International Colloquium on Automata, Languages and Programming, ICALP'98*. Lecture Notes in Computer Science 1443, Springer, 1998, 731–742.

[16] D. REIDENBACH, M. L. SCHMID, Patterns with bounded Treewidth. *Information and Computation* **239** (2014), 87–99.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 27–30.

Regular Grammars for Array Languages

Henning Fernau[(A)] Meenakshi Paramasivan[(A)] D. G. Thomas[(B)]

[(A)]Fachbereich 4 – Abteilung Informatik, Universität Trier, D-54286 Trier, Germany
fernau@uni-trier.de; meena_maths@yahoo.com

[(B)]Department of Mathematics, Madras Christian College, Chennai - 600059, India
dgthomasmcc@yahoo.com

Abstract

We study two different types of regular grammars of array languages introduced in the literature and we refine the presentation of (regular : regular) array grammars in order to clarify the non-closure under union.

1. Introduction and Definitions

Let Σ_m^n denote the set of two-dimensional arrays over the alphabet Σ with n columns and m rows and consequently, $\Sigma_*^* = \bigcup_{n,m\geq 0} \Sigma_m^n$ is the set of all (rectangular) arrays over Σ. Several regular-like mechanisms have been proposed in the literature to generalize, say, right-linear grammars from the 1-dimensional (string) case to the 2-dimensional world.

The original definition of a two-dimensional right-linear grammars (2RLG) [6] under the name regular matrix grammars (RMG) can be found in [7, 8]. RMG and boustrophedon finite automata (BFA) [1], are among the simpler devices. RMG have been somehow extended towards so-called regular-regular array grammars ((R:R)AG) in [8]. By splitting the definition of (R:R)AG into two parts, according to the types. We are giving only an intuitive description of BFA here. BFA can be seen as a restriction of the so-called isometric regular array grammars (IRAG, originally introduced to describe non-rectangular-shaped pictures) that can only move east, west and south. We denote the corresponding language families as $\mathcal{L}(\text{IRAG})$ and $\mathcal{L}(\text{BFA})$, respectively, adding subscripts indicating alphabets when necessary. As described in [3], IRAGs can be viewed as natural extensions of BFAs and then seen to describe a superclass of BFA- and RMG-languages. The class of all languages generated by RMGs (defined below) is $\mathcal{L}(\text{RMG})$.

It is customary to study operations on array and array languages. For reasons of space, we only list the operations that we use in this paper in the following, together with a short description of their meaning, otherwise referring to [1, 4].

A *2-dimensional word* (also called *picture*, *matrix* or an *array*) *over* Σ is a tuple

$$W := ((a_{1,1}, a_{1,2}, \ldots, a_{1,n}), (a_{2,1}, a_{2,2}, \ldots, a_{2,n}), \ldots, (a_{m,1}, a_{m,2}, \ldots, a_{m,n})),$$

where $m, n \in \mathbb{N}$ and, for every i, $1 \leq i \leq m$, and j, $1 \leq j \leq n$, $a_{i,j} \in \Sigma$. Every subset $L \subseteq \Sigma_*^*$ is a (rectangular) *array language*. Let $W := [a_{i,j}]_{m,n}$ and $W' := [b_{i,j}]_{m',n'}$ be two non-empty

pictures over Σ. The *column concatenation* of W and W', denoted by $W \oplus W'$, is undefined if $m \neq m'$. The *row concatenation* of W and W', denoted by $W \ominus W'$, is undefined if $n \neq n'$. For a picture W and $k, k' \in \mathbb{N}$, by W^k we denote the k-fold column-concatenation of W, by W_k we denote the k-fold row-concatenation of W, and we write $W^k_{k'} := (W^k)_{k'}$. If L_1 and L_2 are two array languages the *column product* and the *row product* is defined as $L_1 \oplus L_2 = \{W \oplus W' : W \in L_1, W' \in L_2\}$ and $L_1 \ominus L_2 = \{W \ominus W' : W \in L_1, W' \in L_2\}$ respectively.

Let L be a array language and $L^{1,\oplus} = L$, $L^{i+1,\oplus} = L^{i,\oplus} \oplus L$ and $L_{i+1,\ominus} = L_{i,\ominus} \ominus L$ for $i \geq 1$; then $L^+ = \bigcup_{i=1}^{\infty} L^{i,\oplus}$ and $L_+ = \bigcup_{i=1}^{\infty} L_{i,\ominus}$. If W is an array, then the *transpose* and *half-turn* of W are given by $T(W) := ((a_{1,1}, a_{1,2}, \ldots, a_{m,1}), (a_{1,2}, a_{2,2}, \ldots, a_{m,2}), \ldots, (a_{1,n}, a_{2,n}, \ldots, a_{m,n}))$ and $H(W) := ((a_{m,n}, a_{m,2}, \ldots, a_{m,1}), (a_{2,n}, a_{2,2}, \ldots, a_{2,1}), \ldots, (a_{1,n}, a_{1,2}, \ldots, a_{1,1}))$, respectively. In a natural way, these operations are lifted to array languages and even to families of array languages.

Definition 1.1 *[6] A* two-dimensional right-linear grammar (RMG for short) *is defined by a 7-tuple $G = (V_h, V_v, \Sigma_I, \Sigma, S, R^h, R^v)$, where: V_h is a finite set of* horizontal non-terminals*; V_v is a finite set of* vertical non-terminals*, with $V_h \cap V_v = \emptyset$; $\Sigma_I \subseteq V_v$ is a finite set of* intermediates*; Σ is a finite set of* terminals*; $S \in V_h$ is a* starting symbol*; R^h is a finite set of* horizontal rules *of the form $V \to AV'$ or $V \to A$, where V, $V' \in V_h$ and $A \in \Sigma_I$; and R^v is a finite set of* vertical rules *of the form $W \to aW'$ or $W \to a$, where $W, W' \in V_v$ and $a \in \Sigma$.*

Example 1.2 *$L = \{0\}^+_+ \oplus \{1\}_+ \oplus \{0\}^+_+$ is generated by the RMG $G = (V_h, V_v, \Sigma_I, \Sigma, S, R^h, R^v)$, where $V_h = \{S, X, Y\}$, $V_v = \{A, B\}$, $\Sigma_I = V_v$, $\Sigma = \{0, 1\}$, $R^h = \{S \to AX, X \to AX, X \to BY, Y \to AY, Y \to A\}$ and $R^v = \{A \to 0A, A \to 0, B \to 1B, B \to 1\}$.*

Theorem 1.3 *[2]* $\mathcal{L}_\Sigma(\text{BFA}) = T(\mathcal{L}_\Sigma(\text{RMG}))$.

Siromoney et al. introduced another interesting class of array grammars called (R:R)AG [8] to generate picture languages which cannot be generated by RMG. As we are not so much interested in other language families (as described in [8]), we are now giving a different yet equivalent formalization of this picture language description as follows:

Definition 1.4 *An (R:R)AG can be specified as $G = (S, V_N, V_I, \Sigma, P_N, P_I, \pi, \tau)$, where the components are as follows: A nonterminal alphabet V_N with a distinctive start symbol $S \in V_N$; an intermediate alphabet V_I, disjoint from V_N; a terminal alphabet Σ, disjoint from $V_N \cup V_I$; a set P_N of non-terminal rules that are either of the form $A \to XB$ (right-linear), or of the form $A \to BX$ (left-linear), where $A, B \in V_N$ and $X \in V_I$; a set P_I of rules of the form $A \to X$, with $A \in V_N$ and $X \in V_I$; a picture association mapping $\pi : V_I \to \mathcal{L}_\Sigma(\text{RMG}) \cup \mathcal{L}_\Sigma(\text{BFA})$; a type interpretation mapping $\tau : P_N \to \{\ominus, \oplus\}$ such that $\tau(p_1) = \tau(p_2)$ implies that p_1 is right-linear if and only if p_2 is right-linear.*

Observe that the last condition implies that p_1 is left-linear if and only if p_2 is left-linear. The derivation proceeds as follows: first, a derivation tree T is generated by the linear rules given by P_N, P_I, starting with S.

According to the type, the inner nodes are henceforth interpreted as row or as column catenation. Finally, π is applied to all leaves of the tree.

So, we obtain a tree whose leaves correspond to array languages and whose inner nodes show catenation operators; hence, we can inductively, bottom-up, associate a language to all inner nodes and hence to the root of T. The language we associate with G is then the union of all languages associated to roots of derivation trees of G in this manner. This describes the language family $\mathcal{L}((\mathrm{R:R})\mathrm{AG})$.

Example 1.5 *A (R:R)AG that generates the staircase of* x*'s of a fixed proportion is defined as* $G = (S, V_N, V_I, \Sigma, P_N, P_I, \pi, \tau)$, *where* $V_N = \{S, A\}$, $V_I = \{X_\uparrow, X_\rightarrow, X\}$, $\Sigma = \{\mathtt{x}, \bullet\}$, $P_N = \{S \to AX_\rightarrow, A \to X_\uparrow S\}$, $P_I = \{S \to X\}$, *a picture association mapping* $\pi : V_I \to \mathcal{L}_\Sigma(\mathrm{RMG}) \cup \mathcal{L}_\Sigma(\mathrm{BFA})$ *is given by* $\pi(X_\uparrow) = \{(\begin{smallmatrix}\bullet\\\bullet\\\bullet\end{smallmatrix})^n \oplus (\begin{smallmatrix}\bullet\end{smallmatrix}) \mid n \geq 1\}$, $\pi(X_\rightarrow) = \{(\begin{smallmatrix}\bullet&\bullet&\mathtt{x}\\\bullet&\bullet&\mathtt{x}\\\mathtt{x}&\mathtt{x}&\mathtt{x}\end{smallmatrix}) \ominus (\begin{smallmatrix}\bullet&\bullet&\bullet\end{smallmatrix})_n \mid n \geq 1\}$ *and* $\pi(X) = \begin{smallmatrix}\bullet&\bullet&\bullet&\mathtt{x}\\\bullet&\bullet&\bullet&\mathtt{x}\\\mathtt{x}&\mathtt{x}&\mathtt{x}&\mathtt{x}\end{smallmatrix}$, *Here* $\pi(X_\uparrow) \in \mathcal{L}_\Sigma(\mathrm{RMG})$, $\pi(X_\rightarrow) \in \mathcal{L}_\Sigma(\mathrm{BFA})$ *and interestingly* $\pi(X) \in \mathcal{L}_\Sigma(\mathrm{RMG}) \cap \mathcal{L}_\Sigma(\mathrm{BFA})$, *and a type interpretation mapping* $\tau : P_N \to \{\ominus, \oplus\}$ *is given by* $\tau(S \to AX_\rightarrow) = \oplus$ *and* $\tau(A \to X_\uparrow S) = \ominus$. *Here* $\tau(S \to AX_\rightarrow) \neq \tau(A \to X_\uparrow S)$.

Definition 1.6 *An (R:R)AG* G *with* $|\tau(P_N)| = 2$ *is called* $\ominus$-left, $\oplus$-right

- *if* $A \to BX \in P_N$ *then* $\tau(A \to BX) = \ominus$,
- *if* $A \to XB \in P_N$ *then* $\tau(A \to XB) = \oplus$.

$\ominus$-left, $\oplus$-right (R:R)AG describes the language family $\mathcal{L}_{\ominus-\ell,\oplus-r}((\mathrm{R:R})\mathrm{AG})$.

Definition 1.7 *An (R:R)AG* G *with* $|\tau(P_N)| = 2$ *is called* $\oplus$-left, $\ominus$-right

- *if* $A \to BX \in P_N$ *then* $\tau(A \to BX) = \oplus$,
- *if* $A \to XB \in P_N$ *then* $\tau(A \to XB) = \ominus$.

$\oplus$-left, $\ominus$-right (R:R)AG describes the language family $\mathcal{L}_{\oplus-\ell,\ominus-r}((\mathrm{R:R})\mathrm{AG})$.

2. Results

Lemma 2.1 $\mathcal{L}((\mathrm{R:R})\mathrm{AG}) = \mathcal{L}_{\ominus-\ell,\oplus-r}((\mathrm{R:R})\mathrm{AG}) \cup \mathcal{L}_{\oplus-\ell,\ominus-r}((\mathrm{R:R})\mathrm{AG})$.

Lemma 2.2 *(a) A regular string language corresponds to a language of single-row arrays generated by RMG. (b) A regular string language corresponds to a language of single-column arrays generated by RMG.*

By combining Lemma 2.2 with Theorem 1.3 we obtain Corollary 2.3 and Remark 2.4, as by definition BFA and RMG languages are in $\mathcal{L}_{\ominus-\ell,\oplus-r}((\mathrm{R:R})\mathrm{AG}) \cap \mathcal{L}_{\oplus-\ell,\ominus-r}((\mathrm{R:R})\mathrm{AG})$.

Corollary 2.3 *(a) A regular string language corresponds to a language of single-row arrays accepted by BFA. (b) A regular string language corresponds to a language of single-column arrays accepted by BFA.*

Remark 2.4 *A language of single-row (column) arrays is in both families* $\mathcal{L}_{\ominus-\ell,\oplus-r}((\mathrm{R:R})\mathrm{AG})$ *and* $\mathcal{L}_{\oplus-\ell,\ominus-r}((\mathrm{R:R})\mathrm{AG})$.

In [8, Theorem 3.1], it is claimed that $\mathcal{L}((\mathrm{R}:\mathrm{R})\mathrm{AG})$ is closed under union, without giving any proof. Unfortunately, this claim is wrong.

Theorem 2.5 *$\mathcal{L}((\mathrm{R}:\mathrm{R})\mathrm{AG})$ is not closed under union.*

Proof. (Sketch) Let us reconsider the array language $L = \{0\}_+^+ \oplus \{1\}_+ \oplus \{0\}_+^+$ that can be described by some RMG (see Example 1.2). Let $\hat{L} = (T(L) \ominus L \ominus T(L)) \oplus L \oplus T(L)$. This language can be described by some (R:R)AG that starts puzzling together the pieces from the left lower corner. Notice that $H(\hat{L}) = T(L) \oplus L \oplus (T(L) \ominus L \ominus T(L))$. Assume that there is some (R:R)AG G generating $\hat{L} \cup H(\hat{L})$. Consider a picture P from $H(\hat{L})$ that is sufficiently big and that is generated by G starting in the left lower corner. In the process of generating P, we have to distinguish two cases which will lead to a contradiction as explained in [5]. □

References

[1] H. FERNAU, M. PARAMASIVAN, M. L. SCHMID, D. G. THOMAS, Scanning Pictures the Boustrophedon Way. In: R. P. BARNEVA, B. B. BHATTACHARYA, V. E. BRIMKOV (eds.), *International Workshop on Combinatorial Image Analysis IWCIA*. LNCS 9448, Springer, 2015, 202–216.

[2] H. FERNAU, M. PARAMASIVAN, M. L. SCHMID, D. G. THOMAS, *Simple Picture Processing Based on Finite Automata and Regular Grammars*, 2017. Submitted to Journal of Computer and System Sciences.

[3] H. FERNAU, M. PARAMASIVAN, D. G. THOMAS, Regular Array Grammars and Boustrophedon Finite Automata. In: H. BORDIHN, R. FREUND, B. NAGY, G. VASZIL (eds.), *Eighth Workshop on Non-Classical Models of Automata and Applications (NCMA 2016); Short Papers*. 2016, 55–63.

[4] H. FERNAU, M. PARAMASIVAN, D. G. THOMAS, Picture Scanning Automata. In: R. P. BARNEVA, V. E. BRIMKOV, J. M. R. S. TAVARES (eds.), *Computational Modeling of Objects Presented in Images. Fundamentals, Methods, and Applications - 5th International Symposium, CompIMAGE 2016*. LNCS 10149, Springer, 2017, 132–147.

[5] H. FERNAU, M. PARAMASIVAN, D. G. THOMAS, Regular Grammars for Array Languages. In: R. FREUND, F. MRÁZ, D. PRŮŠA (eds.), *Ninth Workshop on Non-Classical Models of Automata and Applications (NCMA)*. 2017, 119–134.

[6] D. GIAMMARRESI, A. RESTIVO, Two-dimensional languages. In: G. ROZENBERG, A. SALOMAA (eds.), *Handbook of Formal Languages, Volume III*. Springer, 1997, 215–267.

[7] G. SIROMONEY, R. SIROMONEY, K. KRITHIVASAN, Abstract Families of Matrices and Picture Languages. *Computer Graphics and Image Processing* **1** (1972), 284–307.

[8] G. SIROMONEY, R. SIROMONEY, K. KRITHIVASAN, Picture Languages with Array Rewriting Rules. *Information and Control (now Information and Computation)* **22** (1973) 5, 447–470.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 31–31.

Deterministic Regular Expressions with Back-References

Dominik D. Freydenberger[A] Markus L. Schmid[B]

[A]Loughborough University, Loughborough, United Kingdom
ddfy@ddfy.de

[B]University of Trier, Germany
MSchmid@uni-trier.de

Abstract

Most modern libraries for regular expression matching allow back-references (i.e., repetition operators) that substantially increase expressive power, but also lead to intractability. In order to find a better balance between expressiveness and tractability, we combine these with the notion of determinism for regular expressions used in XML DTDs and XML Schema. This includes the definition of a suitable automaton model, and a generalization of the Glushkov construction.

The talk is based on the paper [1]. See http://ddfy.de/ for the full version.

References

[1] D. D. FREYDENBERGER, M. L. SCHMID, Deterministic Regular Expressions with Back-References. In: *STACS*. 2017, 33:1–33:14.

[A]Supported by DFG grant FR 3551/1-1.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 32–36.

Concise Description of Finite Languages, Revisited

Hermann Gruber[(A)] Markus Holzer[(B)] Simon Wolfsteiner[(C)]

[(A)]Knowledgepark GmbH, Leonrodstr. 68, 80636 München, Germany
hermann.gruber@kpark.de

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

[(C)]Institut für Diskrete Mathematik und Geometrie, TU Wien,
Wiedner Hauptstr. 8–10, 1040 Wien, Austria
simon.wolfsteiner@tuwien.ac.at

Abstract

We investigate the grammatical complexity of finite languages w.r.t. context-free grammars and variants thereof. It is shown that the minimal number of productions necessary for a finite language encoded by a context-free grammar cannot be approximated within a ratio of $o(n^d)$, for all $d \geq 1$, unless $\mathsf{P} = \mathsf{NP}$. Here, n is the length of longest word in the finite language. Similar inapproximability results hold for linear context-free and right-linear (or regular) grammars.

1. Introduction

Questions regarding the economy of descriptions of formal languages by different formalisms such as automata, grammars, and formal systems have been studied quite extensively in the past, see, e.g., [13, 14]. The results in [4] mark the starting point of a theory of the grammatical complexity of finite languages where the chosen complexity measure is the number of productions. In particular, [4] gives a relative succinctness classification for various kinds of context-free grammars. Further results along these lines can be found in [1, 2, 3, 15] as well as some newer ones in, e.g., [6, 7, 9, 10]. It is worth mentioning that in [10] a method for proving lower bounds on the number of productions for context-free grammars was developed. For instance, it was shown that the set of all squares of a given length requires an exponential number of productions to be generated by a context-free grammar.

More recently, it was shown that there is a close relationship between a certain class of formal proofs in first-order logic and a certain class of (tree) grammars. In particular, the number of productions in such a grammar corresponds to the number of certain inference rules in

[(C)]This research was completed while the author was on leave at the Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany, in Summer 2017 and is partially supported by FWF project W1255-N23.

the proof [12, 8]. This correspondence sparked our interest in further investigating questions regarding the grammatical complexity of finite languages. The main result of this paper is that the minimal number of productions necessary for a finite language encoded by a context-free grammar cannot be approximated within a ratio of $o(n^d)$, for all $d \geq 1$, unless $\mathsf{P} = \mathsf{NP}$. Here, n is the length of a longest word in the finite language. This result nicely generalizes the inapproximability of the smallest grammar problem with approximation ratio less than $\frac{8569}{8568}$ unless $\mathsf{P} = \mathsf{NP}$ from [5]. Here, the smallest grammar problem asks for the smallest (in terms of the number of productions) context-free grammar that generates exactly *one* given word. As a byproduct of our inapproximability result, we show that the set of all cubes of a given length requires an exponential number of productions using elementary methods already developed in [4]. To be more precise, the language $T_n = \{\, w\$w\#w \mid w \in \{0,1\}^n \,\}$ requires $\Theta(2^n)$ context-free productions, for $n \geq 1$. This constitutes a drastic improvement of previous results obtained in [4] and, moreover, is more precise than using the lower bound method from [10] that results only in a lower bound of $\Omega(2^{n/8}/\sqrt{3n})$ many context-free productions.

2. Preliminaries

We assume the reader to be familiar with the basic notions on grammars and languages as contained in [11]. In particular, a *context-free grammar* (CFG) is a 4-tuple $G = (N, T, P, S)$, where N and T are disjoint alphabets of *nonterminals* and *terminals*, respectively, $S \in N$ is the *axiom*, and P is a finite set of *productions* of the form $A \to \alpha$, where $A \in N$ and $\alpha \in (N \cup T)^*$. As usual, the derivation relation of G is denoted by $\Rightarrow_G$, and the reflexive and transitive closure of $\Rightarrow_G$ is written as $\Rightarrow_G^*$. The *language generated* by G is defined as

$$L(G) = \{\, w \in T^* \mid S \Rightarrow_G^* w \,\}.$$

We also consider the following restrictions of context-free grammars: (i) a context-free grammar is said to be *linear context-free* (LIN) if the productions are of the form $A \to \alpha$, where $A \in N$ and $\alpha \in T^*(N \cup \{\varepsilon\})T^*$—here ε refers to the *empty word*, and (ii) a context-free grammar is said to be *right-linear* or *regular* (REG) if the productions are of the form $A \to \alpha$, where $A \in N$ and $\alpha \in T^*(N \cup \{\varepsilon\})$. Moreover, a grammar is said to have *weight* at most two, if every right-hand side α of each production $A \to \alpha$ in P is of length at most two, that is, $|\alpha| \leq 2$. Linear context-free and regular grammars of weight at most two are abbreviated by SLIN and SREG, respectively—the prefix S stands for *strict*—this naming was coined in [4]. Furthermore, Γ will denote the set of those abbreviations in the sequel, that is, $\Gamma = \{\mathsf{SREG}, \mathsf{REG}, \mathsf{SLIN}, \mathsf{LIN}, \mathsf{CFG}\}$.

We are interested in the complexity of finite languages w.r.t. different types of grammars. To be more precise: what is the smallest number of productions of a grammar required to generate the language L? Let $G = (N, T, P, S)$ be a context-free grammar. We define $|G|$ to be the number of productions if not stated otherwise, i.e., the number of elements in P. Then the *complexity* of a finite language L w.r.t. an X-grammar, for $X \in \Gamma$, also called the *X-complexity of L*, is defined as

$$X\mathsf{c}(L) = \min\{\, |G| \mid G \text{ is an } X\text{-grammar and } L = L(G) \,\}.$$

By definition, the following relations hold: $\mathsf{CFG} \leq \mathsf{LIN} \leq \mathsf{REG} \leq \mathsf{SREG}$ and moreover we have $\mathsf{CFG} \leq \mathsf{LIN} \leq \mathsf{SLIN} \leq \mathsf{SREG}$, where $X \leq Y$, for $X, Y \in \Gamma$, if and only if $X\mathsf{c}(L) \leq Y\mathsf{c}(L)$, for every finite language L. In the case that $X \leq Y$, we say that X is *more succinct than* Y.

3. Results

In the seminal paper [4] on concise description of finite languages by different types of grammars, certain languages were identified that can only be generated minimally by listing all words that belong to the language under consideration. For instance, the language

$$U_n = \{\, a^k b^k c a^\ell b^\ell d a^m b^m \mid 0 \leq k+\ell+m \leq n \,\}$$

contains a quadratic number of words and satisfies $\mathsf{CFGc}(U_n) = \Omega(n^2)$. The proof of this fact is based on [4, Lemma 2.1] which states some easy facts about *minimal* context-free grammars: let $G = (N, T, P, S)$ be a *minimal* context-free grammar for the finite language L. Then for every nonterminal $A \in N \setminus \{S\}$, there are words α_1 and α_2 with $\alpha_1 \neq \alpha_2$ such that $A \to \alpha_1$ and $A \to \alpha_2$ are in P. Moreover, for every $A \in N \setminus \{S\}$, the set $L_A(G) = \{\, w \in T^* \mid A \Rightarrow_G^* w \,\}$ contains at least two words, and there is no derivation of the form $A \Rightarrow_G^+ \alpha A \beta$ with $\alpha, \beta \in (N \cup T)^*$. Finally, for every $A \in N \setminus \{S\}$, there are $u_1, u_2, v_1, v_2 \in T^*$ such that $u_1 A u_2 \neq v_1 A v_2$ as well as $S \Rightarrow_G^* u_1 A u_2$ and $S \Rightarrow_G^* v_1 A v_2$. Using these facts we show that the set

$$T_n = \{\, w\$w\#w \mid w \in \{0,1\}^n \,\}$$

of all tripels of length n can be generated minimally by a context-free grammar only by listing all words. Thus, we have the following result—observe that this result is more precise than using the lower bound technique from [10] for the language under consideration:

Theorem 3.1 *Let $X \in \Gamma$ and $n \geq 1$. Then $X\mathsf{c}(T_n) = \Theta(2^n)$.*

The language T_n will be a basic building block for our main result, which states that the minimal number of context-free productions for a finite language cannot be approximated within a certain factor unless $\mathsf{P} = \mathsf{NP}$. The main result reads as follows:

Theorem 3.2 *Let $X \in \Gamma$. Given an X-grammar generating a finite language, it is impossible to approximate $X\mathsf{c}(L)$ within a factor of $o(n^d)$, for $n = \max\{\, |w| \mid w \in L \}$ and all $d \geq 1$, unless $\mathsf{P} = \mathsf{NP}$.*

The proof strategy is by a reduction from the coNP-complete unsatisfiability problem for 3SAT-formulae: given a formula F with m clauses and n variables, where each clause is the disjunction of at most 3 literals, it is coNP-complete to determine whether F is unsatisfiable—in other words whether the negation of F is a *tautology*. Then the core idea is to give a suitable presentation of non-satisfying assignments of F in $\{0,1\}^n$ for the n variables in form of a grammar G, such that F is unsatisfiable if and only if $L(G) = \{0,1\}^n$; by construction there is a one-to-one correspondence between assignments and words from the set $\{0,1\}^n$. In order to finish our reduction we embed G into a grammar that generates the language

$$L_F = L(G) \cdot \{0,1,\$,\#\}^{3c \cdot \log n + 2} \cup \{0,1\}^n \cdot T_{c \cdot \log n},$$

for some carefully chosen constant c. It is not hard to see that this reduction is polynomial, even if we force the grammar for L_F to be (strict) regular. Then we distinguish two cases: (i) clearly, if F is unsatisfiable then $L_F = \{0,1\}^n \cdot \{0,1,\$,\#\}^{3c \cdot \log n + 2}$ and there is a CFG-grammar with a constant number of productions that generates L_F. For the other types of X-grammars,

for $X \in \{\mathsf{REG}, \mathsf{SREG}, \mathsf{LIN}, \mathsf{SLIN}\}$, a linear number of productions suffices, i.e., the number is $O(n)$. (ii) On the other hand, if F is satisfiable, there is an assignment that evaluates F to *true*. Hence, there is a word $w \in \{0,1\}^n$ that corresponds to that assignment and is *not* a member of $L(G)$. But then the left-quotient of L_F w.r.t. the word w, that is, the language $w^{-1}L_F = \{\, v \in \{0,1,\$,\#\}^* \mid wv \in L_F \,\}$, is equal to the language of cubes $T_{c\cdot\log n}$. In order to estimate the number of productions for the set $w^{-1}L$, for some word $w \in T^*$ and a language $L \subseteq T^*$, we apply the following lemma.

Lemma 3.3 *Let $X \in \Gamma$ and $G = (N, T, P, S)$ be an X-grammar generating a finite language with $n = \max\{\, |w| \mid w \in L(G) \,\}$. Then one can effectively construct a grammar G' of the same type with $|G'| \leq |G|$, if $X \in \{\mathsf{REG}, \mathsf{SREG}\}$, and $|G'| = O(|G| \cdot n^4)$, if $X \in \{\mathsf{LIN}, \mathsf{SLIN}, \mathsf{CFG}\}$, satisfying $L(G') = w^{-1}L(G)$, for every $w \in T^*$.*

Before we continue with the outline of the proof strategy of the main theorem, we briefly explain the construction of the proof of the lemma. First, we transform the grammar into an equivalent grammar of the same type and *weight at most two*. This increases the number of productions at most by a factor of $O(n)$. Then we apply the triple construction of this grammar with the partial deterministic finite automaton that accepts wT^* in order to accept the intersection of both languages. Simultaneously during this construction, we take care of the triples that directly terminate to letters from the word w and replace them by the empty word ε. These triples can be easily identified, because the partial deterministic finite automaton for wT^* is actually a chain, where the word w is read, followed by the sole accepting state that has a trivial loop on all letters from T. The tedious details are left to the reader. The triple construction with the simultaneous modification increases the grammar at most by a factor of $O(n^3)$. Overall, this gives an increase by a factor of $O(n^4)$ from the original grammar. This proves the stated result for SLIN-, LIN-, and CFG-grammars. An alternative proof shows the linear bound for REG- and SREG-grammars for the quotient w.r.t a single word w.

Now let us come back to the proof outline for the main theorem. Assume that there is a polynomial time approximation algorithm for the minimal number of productions problem within $o(n^d)$, where n is the length of the longest word in the language under consideration and $d \geq 1$. Then this algorithm could be applied to decide whether $\mathsf{CFGc}(L_F) = O(1)$. To this end, set $c = d + 5$. If F is unsatisfiable, then $\mathsf{CFGc}(L_F) = O(1)$, as mentioned above. But if F is satisfiable, let w present a satisfying assignment for F. Then $w^{-1}L_F = T_{c\cdot\log n}$, and by Theorem 3.1, we have $\mathsf{CFGc}(T_{c\cdot n}) = \Theta(n^{d+5})$. By Lemma 3.3 we deduce that $\mathsf{CFGc}(L_F) = \Omega(n^{d+1})$ in this case. Thus, the putative approximation algorithm returns a grammar size of $o(n^{d+1})$ if and only if F is unsatisfiable. This solves a coNP-hard problem in deterministic polynomial time, which implies $\mathsf{P} = \mathsf{NP}$. A similar reasoning can be done with the other types of grammars from Γ. The details are left to the reader. This proves Theorem 3.2 and shows that the X-complexity, for $X \in \Gamma$, of a given finite language cannot be approximated within a factor of $o(n^d)$, for all $d \geq 1$, unless $\mathsf{P} = \mathsf{NP}$.

References

[1] B. ALSPACH, P. EADES, G. ROSE, A Lower-Bound For the Number of Productions Required For A Certain Class of Languages. *Discrete Appl. Math.* **6** (1983), 109–115.

[2] W. BUCHER, A Note on a Problem in the Theory of Grammatical Complexity. *Theoret. Comput. Sci.* **14** (1981) 3, 337–344.

[3] W. BUCHER, H. A. MAURER, K. CULIK II, Context-Free Complexity of Finite Languages. *Theoret. Comput. Sci.* **28** (1983) 3, 277–285.

[4] W. BUCHER, H. A. MAURER, K. CULIK II, D. WOTSCHKE, Concise Description of Finite Languages. *Theoret. Comput. Sci.* **14** (1981) 3, 227–246.

[5] M. CHARIKAR, E. LEHMAN, D. LIU, R. PANIGRAHY, M. PRABHAKARAN, A. SAHAI, S. SHELAT, The smallest grammar problem. *IEEE Trans. Inf. Theory.* **51** (2005) 7, 2554–2576.

[6] J. DASSOW, Descriptional Complexity and Operations—Two Non-classical Cases. In: G. PIGHIZZINI, C. CÂMPEANU (eds.), *Proceedings of the* 19*th Workshop on Descriptional Complexity of Formal Systems*. Number 10316 in LNCS, Springer, Milano, Italy, 2017, 33–44.

[7] J. DASSOW, R. HARBICH, Production Complexity of Some Operations on Context-Free Languages. In: M. KUTRIB, N. MOREIRA, R. REIS (eds.), *Proceedings of the* 14*th Workshop on Descriptional Complexity of Formal Systems*. Number 7386 in LNCS, Springer, Braga, Portugal, 2012, 141–154.

[8] S. EBERHARD, S. HETZL, Compressibility of Finite Languages by Grammars. In: J. SHALLIT, A. OKHOTIN (eds.), *Proceedings of the* 17*th Workshop on Deescriptional Complexity of Formal Systems*. Number 9118 in LNCS, Springer, Waterloo, Ontario, Canada, 2015, 93–104.

[9] K. ELLUL, B. KRAWETZ, J. SHALLIT, M.-W. WANG, Regular Expressions: New Results and Open Problems. *J. Autom., Lang. Comb.* **9** (2004) 2/3, 233–256.

[10] Y. FILMUS, Lower Bounds for Context-Free Grammars. *Inform. Process. Lett.* **111** (2011) 18, 895–898.

[11] M. A. HARRISON, *Introduction to Formal Language Theory*. Addison-Wesley, 1978.

[12] S. HETZL, Applying Tree Languages in Proof Theory. In: A. H. DEDIU, C. MARTÍN-VIDE (eds.), *Proceedings of the* 6*th International Conference Language and Automata Theory and Applications*. Number 7183 in LNCS, Springer, A Coruña, Spain, 2012, 301–312.

[13] M. HOLZER, M. KUTRIB, Descriptional Complexity—An Introductory Survey. In: C. MARTÍN-VIDE (ed.), *Scientific Applications of Language Methods*. World Scientific, 2010, 1–58.

[14] A. R. MEYER, M. J. FISCHER, Economy of description by automata, grammars, and formal systems. In: *Proceedings of the* 12*th Annual Symposium on Switching and Automata Theory*. IEEE Computer Society Press, 1971, 188–191.

[15] Z. TUZA, On the Context-Free Production Complexity of Finite Languages. *Discrete Appl. Math.* **18** (1987) 3, 293–304.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 37–40.

Deciding regular intersection emptiness of complete problems for PSPACE and the polynomial hierarchy

Demen Güler[(A)] Andreas Krebs[(A)] Klaus-Jörn Lange[(A)]
Petra Wolf[(A)]

[(A)]WSI - University of Tübingen
Sand 13, 72076 Tübingen, Germany
{gueler,krebs,lange,wolfp}@informatik.uni-tuebingen.de

Abstract

For a regular set R of quantified Boolean formulae we decide whether R contains a true formula. We conclude that there is a PSPACE-complete problem for which emptiness of regular intersection is decidable. Furthermore, by restriction of quantification depth and order we obtain complete problems for each level of the polynomial hierarchy with this decidability as well. [1]

1. Introduction

We consider for a language L of quantified Boolean formulae of several types the emptiness problem of intersection with a regular set, i.e. whether $L \cap R = \emptyset$ for a regular set R given by a finite automaton. Van Leeuwen [12] showed that the satisfiability problem SAT in an appropriate *variable-free* coding is an ET0L (and hence indexed) language, which implies the decidability of the regular intersection emptiness problem [1]. This might be contrasted with the fact that the NP-complete *machine language* $\{\langle M, x, a^n\rangle \mid M \text{ is NTM accepting } x \text{ in } n \text{ steps}\}$ has an undecidable regular intersection emptiness problem.

The original motivation of this work was to distinguish *formal languages* like the contextfree or regular ones with iteration or pumping lemmata and various decidabilities from *non-formal languages* like the machine language.

Since all known families of *formal* languages are contained in NP we were interested in problems (most probably) outside of NP which have a decidable regular intersection emptiness problem. In particular, the existence of families of formal languages being densely complete in the classes NP, SAC[1] and NSPACE($\log n$) [7, 8] motivated this line of research.

In this article we consider the complexity classes PSPACE and the levels of the polynomial hierarchy. *Machine versions* of languages complete for these classes would be defined by using polynomially time bounded Turing machines of unbounded and respectively bounded alternation depth. In both cases the emptiness of intersection with a regular set would be undecidable.

[1]The presented results are extracted from our main article [6].

In analogy to the satisfiability problem languages of (true) quantified formulae are the canonical complete languages for these classes. Arbitrary true quantified Boolean formulae generate a PSPACE-complete language, where constraining quantification depth and order yields languages for the classes of the polynomial hierarchy. We show for Σ_k^P-, Π_k^P- and respectively PSPACE-complete languages L_{Σ_k}, L_{Π_k} and L_{TQBF} that intersection emptiness problem with regular languages is decidable.

A converse viewpoint of this problem is to consider a regular set of encoded quantified Boolean formulae and to decide whether at least one does evaluate to true. For finite sets this problem is, from a decidability perspective, trivial, since each quantified Boolean formula can be evaluated by a Turing machine with a polynomial space bound. On the contrary, the question whether an arbitrary (not necessarily regular) infinite set contains a true quantified Boolean formula is not decidable. In our proof we make use of the finiteness of the state set of a finite automaton which ensures us the repetitions of certain subwords in words accepted by the automaton. This way we only have to evaluate a finite number of candidates when we search for true quantified Boolean formulae in infinite regular sets.

2. Definitions

We use the common notation for Boolean formulae with 0 and 1 as truth values. For readability reasons we extend regular expressions by operations $E^{\leq n} := \bigcup_{i=1}^{n} E^i$ and $E^{\geq n} := E^n E^*$ for a fixed $n \in \mathbb{N}$ and E a regular expression. In particular we write $E^{\geq 1}$ instead of E^+ because signs are part of the alphabet we use.

Definition 2.1 *Let $\Gamma = \{a, b, \langle, \rangle, \wedge, \vee, +, -\}$ and $\pm$ be the regular expression of $\{+, -\}$. We define $L_{k\text{-}QBF}$ as the regular set of encoded quantified Boolean formulae in 3-CNF:*

$$L_{k\text{-}QBF} := \left\{ \langle \pm b^{\leq k} a^{\geq 1} \vee \pm b^{\leq k} a^{\geq 1} \vee \pm b^{\leq k} a^{\geq 1} \rangle \left(\wedge \langle \pm b^{\leq k} a^{\geq 1} \vee \pm b^{\leq k} a^{\geq 1} \vee \pm b^{\leq k} a^{\geq 1} \rangle \right)^* \right\}$$

The literals $\pm b^{\leq k} a^{\geq 1}$ are interpreted in the following way. A $+$ deontes a positive literal, while $-$ denotes negated literal. The number of bs determines the quantifier type (odd numbers are existentially, even are universally quantified) and depth of the throught the number of as indicated variable.
Furthermore, let $L_{QBF} := \bigcup_{k \geq 1} L_{k\text{-}QBF}$ be the set of encoded quantified Boolean formulae without bound of quantifier alternation depth.

Definition 2.2 *Let L_{Σ_k} $(L_{\Pi_k}) \subseteq L_{k\text{-}QBF}$ be the set of all true quantified Boolean formulae in sequential encoding and 3-CNF, where the first quantifier is existential (universal).*

Let $L_{k\text{-}TQBF} := L_{\Sigma_k} \cup L_{\Pi_k}$ and $L_{TQBF} := \bigcup_{k \geq 1} L_{k\text{-}TQBF}$.

Remark 2.3 *For odd (even) k, the language L_{Σ_k} (L_{Π_k}) is Σ_k^P $\left(\Pi_k^P\right)$-complete [13]. The set $L_{TQBF} \subseteq L_{QBF}$ of encoded true quantified Boolean formulae in 3-CNF is PSPACE-complete [11].*

3. Results

In this section we present our two main theorems for the Σ_k^P-complete language L_{Σ_k}, the Π_k^P-complete language L_Π and the PSPACE-complete language L_{TQBF}. We will only sketch the proofs, for details we refer to the main article [6].

Theorem 3.1 *Let R be a regular language. For each $k \in \mathbb{N}$ it is decidable whether $L_{\Sigma_k} \cap R = \emptyset$ and $L_{\Pi_k} \cap R = \emptyset$.*

Proof Idea:
Let A be a DFA recognizing R. For every pair of states in A we compute the (possibly empty) regular set of end-to-end literals that can be read in-between them. Each such literal set is then assigned a finite set of *representing literals*. We define an automaton condense(A) based on the finitely many representatives and show that condense(A) recognizes a true quantified Boolean formula if and only if A accepts one. Finally we show that for each $k \in \mathbb{N}$ the emptiness of condense$(A) \cap L_{\Sigma_k}$ and condense$(A) \cap L_{\Pi_k}$ is decidable, which in total proves Theorem 3.1.

Theorem 3.2 *Let R be a regular language. It is decidable whether $L_{TQBF} \cap R = \emptyset$.*

Proof Idea:
Let R be given as a DFA A. Formulae recognized by A can be unbounded in their quantification depth. We construct a new automaton *restrict*(A) which only accepts formulae of quantification depth up to d, where d is only dependent on the size of A. The idea is to pump unbounded quantifiers in two large groups of quantifiers of the same type. We show that A accepts a *true* quantified Boolean formula if and only if *restrict*(A) accepts one. Following Theorem 3.1 it is decidable whether $L($*restrict*$(A))$ contains a true quantified Boolean formula with at most d alternating quantifiers and thus $L_{\text{TQBF}} \cap R = \emptyset$ is decidable, too.

References

[1] A. V. AHO, Indexed grammars – an extension of context-free grammars. *Journal of the ACM (JACM)* **15** (1968) 4, 647–671.

[2] S. ARORA, B. BARAK, *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[3] H. CALBRIX, T. KNAPIK, A string-rewriting characterization of Muller and Schupp's context-free graphs. In: *FSTTCS*. 98, Springer, 1998, 331–342.

[4] S. A. COOK, The complexity of theorem-proving procedures. In: *Proceedings of the third annual ACM symposium on Theory of computing*. ACM, 1971, 151–158.

[5] J. ENGELFRIET, Iterated stack automata and complexity classes. *Information and computation* **95** (1991) 1, 21–75.

[6] D. GÜLER, A. KREBS, K.-J. LANGE, P. WOLF, Deciding regular intersection emptiness of complete problems for PSPACE and the polynomial hierarchy. *(submitted)* (2017).

[7] A. KREBS, K.-J. LANGE, Dense Completeness. In: *Developments in Language Theory*. Springer, 2012, 178–189.

[8] A. KREBS, K.-J. LANGE, M. LUDWIG, On Distinguishing NC^1 and NL. In: *International Conference on Developments in Language Theory*. Springer, 2015, 340–351.

[9] J.-É. PIN, Mathematical foundations of automata theory. *Lecture notes LIAFA, Université Paris* **7** (2010).

[10] L. J. STOCKMEYER, The polynomial-time hierarchy. *Theoretical Computer Science* **3** (1976) 1, 1–22.

[11] L. J. STOCKMEYER, A. R. MEYER, Word problems requiring exponential time (Preliminary Report). In: *Proceedings of the fifth annual ACM symposium on Theory of computing*. ACM, 1973, 1–9.

[12] J. VAN LEEUWEN, The membership question for ETOL-languages is polynomially complete. *Information Processing Letters* **3** (1975) 5, 138–143.

[13] C. WRATHALL, Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science* **3** (1976) 1, 23–33.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 41–44.

Parallel Contextual Array Insertion Deletion Grammar

S. James Immanuel[(A)] D. G. Thomas[(A)]

[(A)]Department of Mathematics, Madras Christian College, Chennai - 600059, India
james_imch@yahoo.co.in , dgthomasmcc@yahoo.com

Abstract

We introduce a new grammar, called parallel contextual array insertion deletion grammar and show that it has higher generative power as the family of languages generated by these grammars includes families of recognizable languages and Siromoney matrix languages.

1. Introduction and Definitions

Contextual grammars offer novel insight to a number of issues central to formal language theory and hence have been intensively investigated by formal language theorists. A contextual grammar produces a language by starting from a given finite set of strings and adding, iteratively, pairs of strings(called as contexts), associated to sets of words(called selectors) to the string already obtained. Extension of these grammars to 2-dimensional array structures has been attempted in [1, 2, 3, 5]. Based on the modified contextual style of internal parallel contextual array grammars considered in [1] and by using array insertion and deletion operations we introduce a new grammar called parallel contextual array insertion and deletion grammar. We give the comparison of the family of picture languages generated by this new grammar with the families of local languages, recognizable languages [4] and Siromoney matrix languages [6], thus bringing out their generative powers.

Let us briefly recall some standard definitions. Let V be a finite alphabet, V^* is the set of words over Σ including the empty word λ. $V^+ = V^* - \{\lambda\}$. For $w \in V^*$ and $a \in V$, $|w|_a$ denotes the number of occurrences of a in w. An array consists of finitely many symbols from V that are arranged as rows and columns in some particular order and is written in the form, $A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$ or in short $A = [a_{ij}]_{m\times n}$, for all $a_{ij} \in \Sigma$, $i = 1,2,\ldots,m$ and $j = 1,2,\ldots,n$. The set of all arrays over V is denoted by V^{**} which also includes the empty array Λ (zero rows and zero columns). $V^{++} = V^{**} - \{\Lambda\}$. For $a \in V$, $|A|_a$ denotes the number of occurrences of a in A. The column concatenation of $A = \begin{bmatrix} a_{11} & \cdots & a_{1p} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mp} \end{bmatrix}$, and $B = \begin{bmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nq} \end{bmatrix}$, defined only

Acknowledgement: The authors are thankful to Prof. Henning Fernau *(Fachbereich 4 – Abteilung Informatik, Universität Trier, D-54286 Trier, Germany)* for his valuable input to this paper.

when $m = n$, is given by $A \bigoplus B = \begin{bmatrix} a_{11} & \cdots & a_{1p} & b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mp} & b_{n1} & \cdots & b_{nq} \end{bmatrix}$. Similarly, the row concatenation of A and B, defined only when $p = q$, is given by $A \bigoplus B = \begin{bmatrix} a_{11} & \cdots & a_{1p} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mp} \\ b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nq} \end{bmatrix}$.

As pictures are geometrical objects, several further unary operations can be introduced: *quarter-turn* (rotate clockwise by 90°), *half-turn* (rotate by 180°), *anti-quarter-turn* (rotate anti-clockwise by 90° or rotate clockwise by 270°), *transpose* (reflection along the main diagonal), *anti-transpose* (reflection along the anti-diagonal), reflection along the horizontal base line, reflection along the rightmost vertical line.

Definition 1.1 *The parallel column contextual insertion operation is defined as follows: Let V be an alphabet, C be a finite subset of V^{**} whose elements are the column array contexts and $\varphi_c^i : V^{**} \times V^{**} \to 2^C$ be a choice mapping. For arrays, $A = \begin{smallmatrix} a_{1j} & \cdots & a_{1(k-1)} \\ \vdots & \ddots & \vdots \\ a_{mj} & \cdots & a_{m(k-1)} \end{smallmatrix}, B = \begin{smallmatrix} a_{1k} & \cdots & a_{1(l-1)} \\ \vdots & \ddots & \vdots \\ a_{mk} & \cdots & a_{m(l-1)} \end{smallmatrix}, j < k < l, a_{ij} \in V$, we define $\hat{\varphi}_c^i : V^{**} \times V^{**} \to V^{**}$ such that, $I_c \in \hat{\varphi}_c^i(A, B)$, $I_c = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}$ if $c_i = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \in \varphi_c^i \left(\begin{smallmatrix} a_{ij} & \cdots & a_{i(k-1)} \\ a_{(i+1)j} & \cdots & a_{(i+1)(k-1)} \end{smallmatrix}, \begin{smallmatrix} a_{ik} & \cdots & a_{i(l-1)} \\ a_{(i+1)k} & \cdots & a_{(i+1)(l-1)} \end{smallmatrix} \right)$, $c_i \in C, 1 \leq i \leq m-1$, not all need to be distinct.*

Given an array $X = [a_{ij}]_{m \times n} \in V^{**} : X = X_1 \bigoplus A \bigoplus B \bigoplus X_2$, $X_1 = \begin{smallmatrix} a_{11} & \cdots & a_{1(j-1)} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{m(j-1)} \end{smallmatrix}$, $A = \begin{smallmatrix} a_{1j} & \cdots & a_{1(k-1)} \\ \vdots & \ddots & \vdots \\ a_{mj} & \cdots & a_{m(k-1)} \end{smallmatrix}$, $B = \begin{smallmatrix} a_{1k} & \cdots & a_{1(l-1)} \\ \vdots & \ddots & \vdots \\ a_{mk} & \cdots & a_{m(l-1)} \end{smallmatrix}$, $X_2 = \begin{smallmatrix} a_{1l} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{ml} & \cdots & a_{mn} \end{smallmatrix}$, $1 \leq j \leq k < l \leq n+1$ (or) $1 \leq j < k \leq l \leq n+1$, we write $X \Rightarrow_i Y$ if $Y = X_1 \bigoplus A \bigoplus I_c \bigoplus B \bigoplus X_2$, such that $I_c \in \hat{\varphi}_c^i(A, B)$. I_c is called as the inserted column context. We say that Y is obtained from X by parallel column contextual insertion operation. The following 4 special cases for $X = X_1 \bigoplus A \bigoplus B \bigoplus X_2$ are also considered,(i) For $j = 1$, we have $X_1 = \Lambda$. (ii) For $j = k = 1$, we have $X_1 = \Lambda$ and $A = \Lambda$. (iii) For $l = n+1$, we have $X_2 = \Lambda$. (iv) For $k = l = n+1$, we have $B = \Lambda$ and $X_2 = \Lambda$.

The case $j = k = l$ is not possible for performing parallel column contextual insertion operation.

Similarly we can define parallel row contextual insertion operation also.

Definition 1.2 *The parallel column contextual deletion operation is defined as follows: Let V be an alphabet, C be a finite subset of V^{**} whose elements are the column array contexts and $\varphi_c^d : V^{**} \times V^{**} \to 2^C$ be a choice mapping. For arrays, $A = \begin{smallmatrix} a_{1j} & \cdots & a_{1(k-1)} \\ \vdots & \ddots & \vdots \\ a_{mj} & \cdots & a_{m(k-1)} \end{smallmatrix}, B = \begin{smallmatrix} a_{1(k-p)} & \cdots & a_{1(l-1)} \\ \vdots & \ddots & \vdots \\ a_{m(k-p)} & \cdots & a_{m(l-1)} \end{smallmatrix}, j < k < l, a_{ij} \in V$, we define $\hat{\varphi}_c^d : V^{**} \times V^{**} \to V^{**}$ such that, $D_c \in \hat{\varphi}_c^d(A)$, $D_c = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}$ if $c_i = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \in \varphi_c^d \left(\begin{smallmatrix} a_{ij} & \cdots & a_{i(k-1)} \\ a_{(i+1)j} & \cdots & a_{(i+1)(k-1)} \end{smallmatrix}, \begin{smallmatrix} a_{i(k+p)} & \cdots & a_{i(l-1)} \\ a_{(i+1)(k+p)} & \cdots & a_{(i+1)(l-1)} \end{smallmatrix} \right)$, $c_i \in C, 1 \leq i \leq m-1$, not all need to be distinct.*

Given an array $X = [a_{ij}]_{m \times n} \in V^{**} : X = X_1 \bigoplus A \bigoplus D_c \bigoplus B \bigoplus X_2$, $X_1 = \begin{smallmatrix} a_{11} & \cdots & a_{1(j-1)} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{m(j-1)} \end{smallmatrix}$, $A = \begin{smallmatrix} a_{1j} & \cdots & a_{1(k-1)} \\ \vdots & \ddots & \vdots \\ a_{mj} & \cdots & a_{m(k-1)} \end{smallmatrix}, B = \begin{smallmatrix} a_{1(k+p)} & \cdots & a_{1(l-1)} \\ \vdots & \ddots & \vdots \\ a_{m(k+p)} & \cdots & a_{m(l-1)} \end{smallmatrix}, X_2 = \begin{smallmatrix} a_{1l} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{ml} & \cdots & a_{mn} \end{smallmatrix}$, $1 \leq j \leq k < l \leq n+1$, we write

$X \Rightarrow_d Y$ if $Y = X_1 \bigoplus A \bigoplus B \bigoplus X_2$, such that $D_c \in \hat{\varphi}_c^d(A,B)$. D_c is called as the deleted column context. We say that Y is obtained from X by parallel column contextual deletion operation.The following 4 special cases for $X = X_1 \bigoplus A \bigoplus D_c \bigoplus B \bigoplus X_2$ are also considered, (i) For $j = 1$ we have $X_1 = \Lambda$. (ii) For $j = k = 1$, we have $X_1 = \Lambda$ and $A = \Lambda$. (iii) For $l = n+1$, we have $X_2 = \Lambda$. (iv) For $k+p = l = n+1$, we have $B = \Lambda$ and $X_2 = \Lambda$.

Similarly we can define parallel row contextual deletion operation also.

Definition 1.3 *A parallel contextual array insertion deletion grammar is defined as $G = (V,T,M,C,R,\varphi_c^i,\varphi_r^i,\varphi_c^d,\varphi_r^d)$ where, V is an alphabet, $T \subseteq V$ is a terminal alphabet, M is a finite subset of V^{**} called the base of G, C is a finite subset of V^{**} called column array contexts, R is a finite subset of $\$_r V^{**} \$_r$ called row array contexts, $\varphi_c^i : V^{**} \times V^{**} \to 2^C$, $\varphi_r^i : V^{**} \times V^{**} \to 2^R$, $\varphi_c^d : V^{**} \times V^{**} \to 2^C$, $\varphi_r^d : V^{**} \times V^{**} \to 2^R$, are the choice mappings which perform the parallel column contextual insertion, row contextual insertion, column contextual deletion and row contextual deletion operations, respectively.*

The insertion derivation with respect to G is a binary relation $\Rightarrow_i$ on V^{**} and is defined as $X \Rightarrow_i Y$, where $X,Y \in V^{**}$ if and only if $X = X_1 \bigoplus A \bigoplus B \bigoplus X_2$, $Y = X_1 \bigoplus A \bigoplus I_c \bigoplus B \bigoplus X_2$ or $X = X_3 \bigoplus A \bigoplus B \bigoplus X_4$, $Y = X_3 \bigoplus A \bigoplus I_r \bigoplus B \bigoplus X_4$ for some $X_1, X_2, X_3, X_4 \in V^{**}$ and I_c, I_r are inserted column and row contexts obtained by the parallel column or row contextual insertion operations according to the choice mappings.

The deletion derivation with respect to G is a binary relation $\Rightarrow_d$ on V^{**} and is defined as $X \Rightarrow_d Y$, where $X,Y \in V^{**}$ if and only if $X = X_1 \bigoplus A \bigoplus D_c \bigoplus B \bigoplus X_2$, $Y = X_1 \bigoplus A \bigoplus B \bigoplus X_2$ or $X = X_3 \bigoplus A \bigoplus D_r \bigoplus B \bigoplus X_4$, $Y = X_3 \bigoplus A \bigoplus B \bigoplus X_4$ for some $X_1, X_2, X_3, X_4 \in V^{**}$ and D_c, D_r are deleted column and row contexts with respect to the parallel column or row contextual deletion operations according to the choice mappings.

The direct derivation with respect to G is a binary relation $\Rightarrow_{i,d}$ on V^{**} which is either $\Rightarrow_i$ or $\Rightarrow_d$.

Definition 1.4 *Let $G = (V,T,M,C,R,\varphi_c^i,\varphi_r^d,\varphi_c^d,\varphi_r^d)$ be a parallel contextual array insertion deletion grammar. The language generated by G, denoted by L(G) is defined as,*

$$L(G) = \{Y \in T^{**} | \exists X \in M \text{ with } X \Rightarrow_{i,d}^* Y\}.$$

The family of all array languages generated by parallel contextual array insertion deletion grammar is denoted by PCAIDG.

Example 1.5 *Let $G = (V,T,M,C,R,\varphi_c^i,\varphi_r^i,\varphi_c^d,\varphi_r^d)$ where,*

$V = \{X,Y,Z,\bullet\}$, $T = \{X,\bullet\}$, $M = \left\{ \begin{matrix} Z & \bullet & \bullet & X \\ Z & \bullet & \bullet & X \\ Z & X & X & X \\ Z & Z & Z & Z \end{matrix} \right\}$, $C = \left\{ \begin{matrix} Y \\ Y \end{matrix} \begin{matrix} \bullet \\ \bullet \end{matrix} \begin{matrix} \bullet \\ \bullet \end{matrix}, \begin{matrix} Y \\ Y \end{matrix} \begin{matrix} \bullet \\ Z \end{matrix} \begin{matrix} \bullet \\ Z \end{matrix} \begin{matrix} \bullet \\ Z \end{matrix}, \begin{matrix} Y \\ Y \end{matrix}, \begin{matrix} Z \\ Z \end{matrix} \right\}$,

$R = \left\{ \begin{matrix} Z & Y \\ Z & Y \\ Y & Y \end{matrix}, \begin{matrix} Y & \bullet \\ Y & X \\ Y & Y \end{matrix}, \begin{matrix} \bullet & \bullet \\ X & X \\ Y & Y \end{matrix}, \begin{matrix} \bullet & X \\ X & X \\ Y & Y \end{matrix}, \begin{matrix} X & \bullet \\ X & \bullet \\ Y & Y \end{matrix}, \begin{matrix} \bullet & \bullet \\ \bullet & \bullet \\ Y & Y \end{matrix}, \begin{matrix} Y & Y \end{matrix}, \begin{matrix} Z & Z \end{matrix} \right\}$,

$\varphi_c^i \left[\begin{matrix} Z \\ Z \end{matrix}, \begin{matrix} \bullet & \bullet \\ \bullet & \bullet \end{matrix} \right] = \left\{ \begin{matrix} Y \\ Y \end{matrix} \begin{matrix} \bullet \\ \bullet \end{matrix} \begin{matrix} \bullet \\ \bullet \end{matrix} \right\}, \varphi_c^i \left[\begin{matrix} Z \\ Z \end{matrix}, \begin{matrix} \bullet & \bullet \\ X & X \end{matrix} \right] = \left\{ \begin{matrix} Y \\ Y \end{matrix} \begin{matrix} \bullet \\ \bullet \end{matrix} \begin{matrix} \bullet \\ \bullet \end{matrix} \right\}, \varphi_c^i \left[\begin{matrix} Z \\ Z \end{matrix}, \begin{matrix} X & X \\ Z & Z \end{matrix} \right] = \left\{ \begin{matrix} Y \\ Y \end{matrix} \begin{matrix} \bullet \\ Z \end{matrix} \begin{matrix} \bullet \\ Z \end{matrix} \begin{matrix} \bullet \\ Z \end{matrix} \right\}$

$\varphi_r^i \left[\begin{matrix} Z & Y \\ Z & Y \end{matrix}, \begin{matrix} Z & Y \end{matrix} \right] = \left\{ \begin{matrix} Z & Y \\ Z & Y \\ Y & Y \end{matrix} \right\}, \varphi_r^i \left[\begin{matrix} Y \\ Y \end{matrix} \begin{matrix} \bullet \\ \bullet \end{matrix}, \begin{matrix} Y & Z \end{matrix} \right] = \left\{ \begin{matrix} Y & \bullet \\ Y & X \\ Y & Y \end{matrix} \right\}, \varphi_r^i \left[\begin{matrix} \bullet & \bullet \\ \bullet & \bullet \end{matrix}, \begin{matrix} Z & Z \end{matrix} \right] = \left\{ \begin{matrix} \bullet & \bullet \\ X & X \\ Y & Y \end{matrix}, \begin{matrix} \bullet & \bullet \\ \bullet & \bullet \\ Y & Y \end{matrix} \right\}$,

$\varphi_r^i \left[\begin{matrix} \bullet & \bullet \\ \bullet & X \end{matrix}, \begin{matrix} Z & Z \end{matrix} \right] = \left\{ \begin{matrix} \bullet & X \\ X & X \\ Y & Y \end{matrix} \right\}, \varphi_r^i \left[\begin{matrix} \bullet & \bullet \\ X & X \end{matrix}, \begin{matrix} Z & Z \end{matrix} \right] = \left\{ \begin{matrix} X & \bullet \\ X & \bullet \\ Y & Y \end{matrix}, \begin{matrix} \bullet & \bullet \\ \bullet & \bullet \\ Y & Y \end{matrix} \right\}, \varphi_r^i \left[\begin{matrix} \bullet & X \\ X & X \end{matrix}, \begin{matrix} Z & Z \end{matrix} \right] = \left\{ \begin{matrix} \bullet & \bullet \\ \bullet & \bullet \\ Y & Y \end{matrix} \right\}$

$\varphi_c^d\left[\begin{smallmatrix} Z \\ Z \end{smallmatrix}, \begin{smallmatrix} \bullet & \bullet \\ \bullet & \bullet \end{smallmatrix}\right] = \left\{\begin{smallmatrix} Y \\ Y \end{smallmatrix}\right\}, \varphi_c^d\left[\begin{smallmatrix} Z \\ Z \end{smallmatrix}, \begin{smallmatrix} \bullet & \bullet \\ X & X \end{smallmatrix}\right] = \left\{\begin{smallmatrix} Y \\ Y \end{smallmatrix}\right\}, \varphi_c^d\left[\begin{smallmatrix} Z \\ Y \end{smallmatrix}, \begin{smallmatrix} X & X \\ Y & Y \end{smallmatrix}\right] = \left\{\begin{smallmatrix} Y \\ Y \end{smallmatrix}\right\}, \varphi_c^d\left[\begin{smallmatrix} Y \\ Z \end{smallmatrix}, \begin{smallmatrix} Y & Y \\ Z & Z \end{smallmatrix}\right] = \left\{\begin{smallmatrix} Y \\ Y \end{smallmatrix}\right\},$
$\varphi_c^d[\Lambda, \begin{smallmatrix} \bullet & \bullet \\ \bullet & \bullet \end{smallmatrix}] = \left\{\begin{smallmatrix} Z \\ Z \end{smallmatrix}\right\}, \varphi_c^d[\Lambda, \begin{smallmatrix} \bullet & \bullet \\ X & X \end{smallmatrix}] = \left\{\begin{smallmatrix} Z \\ Z \end{smallmatrix}\right\}, \varphi_c^d\left[\Lambda, \begin{smallmatrix} X & X \\ Z & Z \end{smallmatrix}\right] = \left\{\begin{smallmatrix} Z \\ Z \end{smallmatrix}\right\}$

$\varphi_r^d\left[\begin{smallmatrix} Z & \bullet \\ Z & X \end{smallmatrix}, \begin{smallmatrix} Z & Z \end{smallmatrix}\right] = \left\{\begin{smallmatrix} Y & Y \end{smallmatrix}\right\}, \varphi_r^d[\begin{smallmatrix} \bullet & \bullet \\ \bullet & \bullet \end{smallmatrix}, \begin{smallmatrix} Z & Z \end{smallmatrix}] = \left\{\begin{smallmatrix} Y & Y \end{smallmatrix}\right\}, \varphi_r^d\left[\begin{smallmatrix} X & \bullet \\ X & \bullet \end{smallmatrix}, \begin{smallmatrix} Z & Z \end{smallmatrix}\right] = \left\{\begin{smallmatrix} Y & Y \end{smallmatrix}\right\},$
$\varphi_r^d[\begin{smallmatrix} \bullet & \bullet \\ X & X \end{smallmatrix}, \begin{smallmatrix} Z & Z \end{smallmatrix}] = \left\{\begin{smallmatrix} Y & Y \end{smallmatrix}\right\}, \varphi_r^d\left[\begin{smallmatrix} \bullet & X \\ X & X \end{smallmatrix}, \begin{smallmatrix} Z & Z \end{smallmatrix}\right] = \left\{\begin{smallmatrix} Y & Y \end{smallmatrix}\right\}, \varphi_r^d[\begin{smallmatrix} \bullet & \bullet \\ X & X \end{smallmatrix}, \Lambda] = \left\{\begin{smallmatrix} Z & Z \end{smallmatrix}\right\},$
$\varphi_r^d\left[\begin{smallmatrix} \bullet & X \\ X & X \end{smallmatrix}, \Lambda\right] = \left\{\begin{smallmatrix} Z & Z \end{smallmatrix}\right\}, \varphi_r^d[\begin{smallmatrix} \bullet & \bullet \\ \bullet & \bullet \end{smallmatrix}, \Lambda] = \left\{\begin{smallmatrix} Z & Z \end{smallmatrix}\right\}, \varphi_r^d\left[\begin{smallmatrix} X & \bullet \\ X & \bullet \end{smallmatrix}, \Lambda\right] = \left\{\begin{smallmatrix} Z & Z \end{smallmatrix}\right\}$

The picture language generated by this grammar consists of arrays describing staircases of X's of the form, $\begin{smallmatrix} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & X \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & X \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & X & X & X \\ \bullet & \bullet & \bullet & \bullet & \bullet & X & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & X & X & X & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & X & \bullet & \bullet & \bullet & \bullet & \bullet \\ X & X & X & X & \bullet & \bullet & \bullet & \bullet & \bullet \end{smallmatrix}$. *We note that* $L(G) \notin REC \cup \mathfrak{L}(CSML)$ *[4, 6].*

2. Results

Theorem 2.1 *PCAIDG is closed under union, row catenation and column catenation.*

Theorem 2.2 *PCAIDG is closed under transpose, anti-transpose, quarter-turn, half-turn, anti-quarter-turn, reflection along the horizontal base line and reflection along the rightmost vertical line.*

Theorem 2.3 *PCAIDG is closed under projection*

Theorem 2.4 $REC \subsetneq PCAIDG$.

Theorem 2.5 $\mathfrak{L}(CSML) \subsetneq PCAIDG$.

References

[1] P. H. CHANDRA, K. G. SUBRAMANIAN, D. G. THOMAS, Parallel Contextual array grammars and languages. *Electronic Notes in Discrete Math.* **12** (2003), 106–117.

[2] H. FERNAU, R. FREUND, M. L. SCHMID, K. G. SUBRAMANIAN, P. WIELDERHOLD, Contextual array grammars and array P systems. *Annals of Mathematics and Artificial Intelligence* **75** (2015), 5–26.

[3] R. FREUND, GH. PĂUN, G. ROZENBERG, Contextual array grammars. In: K. G. SUBRAMANIAN, K. RANGARAJAN, M. MUKUND (eds.), *Formal Models, Languages and Application Series in Machine Perception and Artificial Intelligence*. 66, World Scientific, 2007, 112–136.

[4] D. GIAMMARRESI, A. RESTIVO, Two- Dimensional Languages. In: *Handbook of Formal Languages*. 3, Springer, 1997, 215–267.

[5] K. KRITHIVASAN, M. S. BALAN, R. RAMA, Array Contextual Grammars. In: C. MARTÍN-VIDE, GH. PĂUN (eds.), *Recent Topics in Mathematical and Computational Linguistics*. The Publishing House of the Romanian Academy, 2000, 154–168.

[6] G. SIROMONEY, R. SIROMONEY, K. KRITHIVASAN, Abstract Families of Matrices and Picture Languages. *Computer Graphics and Image Processing* **1** (1972), 284–307.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 45–47.

Rational, Recognizable, and Aperiodic Sets in the Partially Lossy Queue Monoid

Chris Köcher

Technische Universität Ilmenau, Institut für Theoretische Informatik
chris.koecher@tu-ilmenau.de

The study of different models of automata along with their expressiveness and algorithmic properties is one of the most important areas in automata theory. Many of these models differ in the mechanism to store their data. One very important mechanism is the so called fifo queue (or channel), where data could be written to one end and read from the other end of its contents. If we equip these queues with a finite state automaton we obtain a Turing complete computation model [3], which results in the undecidability of all non-trivial decision problems on these devices.

A surprising result was the decidability of some decision problems like reachability, fair termination or control-state-maintainability if we replace the (reliable) fifo queue by a lossy queue [6, 4, 1, 15] (although of prohibitive complexity, cf. [5]). These lossy queues are a variation of the classical queues which are allowed to forget any part of their contents at any time.

To obtain some algebraic results on the behavior of these storage mechanisms we can model them as monoid of transformations. Some important results on the transformation monoid of reliable queues can be found in [9, 13]. Furthermore in [11] we considered the transformation monoid of lossy queues. When studying the similarities and differences between those two monoids in [12] we found it convenient to join both, the reliable and lossy queues, respectively, into one model, the so called *partially lossy queues* (or *plq*s). Those are given by their underlying alphabet A as well as a subset $X \subseteq A$ of letters that are unforgettable while the letters contained in $A \setminus X$ can be forgotten at any time. We denote their transformation monoid by $\mathcal{Q}(A, X)$ and call it the *partially lossy queue monoid* or *plq monoid*.

Another main topic in the theory of automata and formal languages is the study of regular languages. This revealed strong relations to logic, combinatorics, and algebra. For example, we can generalize the notion of regularity from free monoids to arbitrary monoids. This generalization results in two notions: the rational subsets, which are a generalization of languages that are described by regular expressions, and recognizable subsets, which are a generalization of sets accepted by finite automata (see, e.g., [2, 19]). Kleene's Theorem [10] states that both notions are equivalent in the free monoid.

At first we consider some algorithmic properties of rational subsets of the plq monoid. Such properties encountered increased attention in recent years, e.g., [14] provides a survey on the membership problem for rational sets. We prove in this paper that the membership problem of

Supported by the DFG-Project "Speichermechanismen als Monoide", KU 1107/9-1.

the plq monoid is NL-complete. Furthermore we give some negative results: e.g., we can prove the undecidability of universality, inclusion, and recognizability of rational subsets in $\mathcal{Q}(A, X)$ by reduction from their counterparts in the direct product of $(\mathbb{N}, +)$ and $\{a, b\}^*$ (cf. [17, 7]).

If the given subsets are recognizable, all of these decision problems are decidable by known constructions from automata theory. Hence these results also imply that the classes of rational and recognizable subsets in the plq monoid do not coincide. Due to McKnight's Theorem [16], each recognizable subset is rational in the plq monoid (note that $\mathcal{Q}(A, X)$ is finitely generated). Therefore, it is a natural question to ask in which cases a rational subset is recognizable.

Since we cannot generalize Kleene's Theorem to plq monoids we have to use another approach. A similar situation is known from trace monoids. Here, Ochmański could prove in [18] that it suffices to restrict the Kleene star in an appropriate way to characterize the recognizable subsets in the trace monoid. Here, we will use an approach similar to Ochmański's. In other words, we can prove by restricting Kleene's star-operator and the monoid's product, we can characterize the recognizable subsets in $\mathcal{Q}(A, X)$.

The final result regards the connection between the aperiodic and star-free subsets. Recall that a set is aperiodic if it is accepted by a counter-free automaton and a set is star-free if it can be generated from finite sets by application of Boolean operations and concatenation, only. Schützenberger's Theorem [20] states that both classes coincide in the free monoid. This result gives a decision procedure for regular languages to be star-free. Additionally, in [8] was proven that these classes also coincide in trace monoids. In contrast to these two cases this equality does hold in the plq monoid. But we can characterize the aperiodic subsets in $\mathcal{Q}(A, X)$ as well as the recognizable ones with the help of the same restrictions to star-freeness of subsets as in our result regarding the rational subsets.

References

[1] P. A. ABDULLA, B. JONSSON, Verifying programs with unreliable channels. *Information and Computation* **127** (1996) 2, 91–101.

[2] J. BERSTEL, *Transductions and Context-Free Languages*. Teubner Studienbücher, 1979.

[3] D. BRAND, P. ZAFIROPULO, On Communicating Finite-State Machines. *Journal of the ACM* **30** (1983) 2.

[4] G. CÉCÉ, A. FINKEL, S. PURUSHOTAMAN IYER, Unreliable channels are easier to verify than perfectchannels. *Information and Computation* **124** (1996) 1, 20–31.

[5] P. CHAMBART, P. SCHNOEBELEN, The Ordinal Recursive Complexity of Lossy Channel Systems. In: *LICS'08*. IEEE Computer Society Press, 2008, 205–216.

[6] A. FINKEL, Decidability of the termination problem for completely specified protocols. *Distributed Computing* **7** (1994) 3, 129–135.

[7] A. GIBBONS, W. RYTTER, On the decidability of some problems about rational subsets of free partially commutative monoids. *Theoretical Computer Science* **48** (1986), 329–337.

[8] G. GUAIANA, A. RESTIVO, S. SALEMI, On aperiodic trace languages. In: *STACS'91*. Lecture Notes in Computer Science 480, Springer, 1991, 76–88.

[9] M. HUSCHENBETT, D. KUSKE, G. ZETZSCHE, The monoid of queue actions. *Semigroup forum* (2017). To appear.

[10] S. C. KLEENE, *Representation of events in nerve nets and finite automata*. Technical report, DTIC Document, 1951.

[11] C. KÖCHER, *Einbettungen in das Transformationsmonoid einer vergesslichen Warteschlange*. Master's thesis, TU Ilmenau, 2016.

[12] C. KÖCHER, D. KUSKE, O. PRIANYCHNYKOVA, The Transformation Monoid of a Partially Lossy Queue (2017). Submitted.

[13] D. KUSKE, O. PRIANYCHNYKOVA, The trace monoids in the queue monoid and in the direct product of two free monoids. In: *DLT'16*. Lecture Notes in Computer Science 9840, Springer, 2016, 256–267.

[14] M. LOHREY, The rational subset membership problem for groups: a survey. In: *Groups St Andrews*. 422, Cambridge University Press, 2013, 368–389.

[15] B. MASSON, P. SCHNOEBELEN, On verifying fair lossy channel systems. In: *MFCS'02*. Lecture Notes in Computer Science 2420, Springer, 2002, 543–555.

[16] J. MCKNIGHT, Kleene quotient theorems. *Pacific Journal of Mathematics* **14** (1964) 4, 1343–1352.

[17] A. MUSCHOLL, H. PETERSEN, A Note on the Commutative Closure of Star-Free Languages. *Information Processing Letters* **57** (1996) 2, 71–74.

[18] E. OCHMAŃSKI, Regular behaviour of concurrent systems. *Bulletin of the EATCS* **27** (1985), 56–67.

[19] J.-É. PIN, Mathematical foundations of automata theory. *Lecture notes LIAFA, Université Paris* **7** (2010).

[20] M. P. SCHÜTZENBERGER, On finite monoids having only trivial subgroups. *Information and control* **8** (1965) 2, 190–194.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 48–48.

Die Teilwort- und -spurordnung

Dietrich Kuske

Fachgebiet Automaten und Logik, Technische Universität Ilmenau Helmholtzplatz 5,
D-98684, Ilmenau, Germany
dietrich.kuske@tu-ilmenau.de

Zusammenfassung

Zusammenfassung: Von Schnoebelen und Karandikar bzw. unabhängig davon von Lindner wurde 2015 gezeigt, daß die elementare Theorie der Teilwortordnung für zwei Variable entscheidbar ist (drei Variablen führen zur Unentscheidbarkeit). Der Beweis beruht darauf, daß sowohl die Teilwortordnung als auch ihr Komplement rationale Relationen sind und daß die Klasse der regulären Sprachen unter rationalen Transduktionen abgeschlossen ist. Verwendet man zusätzlich die Theorie der formalen Potenzreihen, so erhält man die Entscheidbarkeit für die Erweiterung der Logik 1. Stufe um Quantoren der Form "es gibt wenigstens k viele Zeugen" (Zählquantor) bzw. "die Anzahl der Zeugen ist gerade" (Mod-Quantor).

Zur Zeit untersuchen wir die analogen Fragen für Spuren. Als erstes fällt auf, daß rationale Relationen nicht hilfreich sind, da die Klasse der regulären Spursprachen unter rationalen Transduktionen nicht abgeschlossen ist. Die Potenzreihen können aber weiter verwendet werden. Aus unseren (bisherigen) Ergebnissen folgt, daß die elementare 2-Variablen-Theorie der Teilspurordnung entscheidbar ist. Die Erweiterung um den Zählquantor erscheint ebenfalls möglich, der Mod-Quantor macht (noch?) an einer Stelle Probleme.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 49–52.

Transducing Reversibly with Finite State Machines

Martin Kutrib[(A)] Andreas Malcher[(A)] Matthias Wendlandt[(A)]

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

Abstract

Finite state machines are investigated towards their ability to reversibly compute transductions, that is, to transform inputs into outputs in a reversible way. This means that the transducers are backward deterministic and hence are able to uniquely step the computation back and forth. The families of transductions computed are classified with regard to three types of length-preserving transductions as well as to the property of working reversibly. It is possible to settle all inclusion relations between the families of transductions. Finally, the standard closure properties are investigated and the non-closure under almost all operations can be shown.

1. Introduction

One main motivation for the study of computational devices performing reversible computations is the physical observation that a loss of information results in heat dissipation [13]. It is therefore of great interest to avoid such situations and to privilege computations in which every configuration has a unique successor configuration as well as a unique predecessor configuration so that at every point of the computation no information gets lost. Reversibility has been studied for many computational devices starting with Lecerf's [15] and Bennett's [5] investigations for Turing machines, where it is shown that for every (possibly irreversible) Turing machine an equivalent reversible Turing machine can be constructed. This result has been achieved also for deterministic space-bounded Turing machines in [14]. For deterministic multi-head finite automata, the results depend on whether or not two-way motion of the heads is allowed. It is shown in [16] that the general model and the reversible model coincide for two-way multi-head finite automata, whereas the reversible model is weaker than the general model in case of one-way motion [12]. A similar result has been obtained in [10] for deterministic pushdown automata. In both cases, the loss of information in computations is inevitable. Reversible computations in deterministic finite automata (DFA) have been introduced in [3] and it is shown in [17] that there are regular languages which cannot be accepted by any (one-way) reversible deterministic finite automaton. On the other hand, it is known due to [9] that the general model and the reversible model coincide if the input head is two-way. Recent results on reversible

regular languages are given in [8], where it is shown that it is NL-complete to decide whether a given one-way DFA accepts a reversible language. Additionally, exponential upper and lower bounds for the conversion of one-way DFAs to equivalent reversible DFAs are given.

Computational models are not only interesting from the vantage point of accepting some input, but also from the viewpoint of transforming some input into some output. For example, a parser for a formal language should not only return the information whether or not the input word can be parsed, but also the parse tree in the positive case. The simplest model in this context is the finite state transducer which is a finite automaton with an output alphabet that assigns to each input accepted at least one output word. Transductions computed by different variants of such transducers are studied in detail in [7]. Deterministic and nondeterministic pushdown transducers are investigated in [2]. Furthermore, characterizations of pushdown transductions as well as applications to the parsing of context-free languages are given. A more general theory of transducing devices has been outlined already 1969 in [1]. More recently, transducing variants of stack automata have been considered in [6], whereas the parallel model of cellular automata has been investigated in [11] towards its transducing capabilities.

Here, we study *reversible* deterministic finite state transducers (DFST). Since reversible devices should be able to preserve information and DFSTs use and produce information concerning the input *and* the output, the transition function in DFSTs will be defined depending on the input and the output. Thus, reversible DFSTs may be considered as reversible Turing machines (see, for example, [4, 5]) with a one-way input tape and a one-way output tape. To start with a weak form of transductions and, again, from the viewpoint of information preserving computations, we are here considering essentially *length-preserving* transductions.

2. Preliminaries

First, we define reversible deterministic finite state transducers. We define this model as usual with two tapes, namely, an input and an output tape. The model can be seen as a restricted variant of a Turing machine having a one-way read-only input tape and a one-way output tape. In the forward computation the transducer decides its operation depending on the current state, the current input symbol, and the symbol at the current square of the output tape. It may perform a right move on the input tape and may rewrite the current tape square on the output tape and afterwards may perform a right move on the output tape as well. The output tape is initially filled with blank symbols.

We define a *deterministic finite state transducer* (DFST) as a system $M = \langle Q, \Sigma, \Delta, q_0, \delta, F \rangle$, where Q is the set of *internal states*, Σ is the set of *input symbols*, Δ is the set of *output symbols* containing the blank symbol $\sqcup$, q_0 is the initial state, $F \subseteq Q$ is the set of *accepting states*, and

$$\delta \colon Q \times \Sigma \times \Delta \to Q \times (\Delta \setminus \{\sqcup\}) \times \{0,1\} \times \{0,1\}$$

is the partial *transition function*.

A *configuration* of DFST M at some time $t \geq 0$ is a quadruple (v, p, w, z), where $v \in \Sigma^*$ is the already read part of the input to the left of the input head, $p \in Q$ is the current state, $w \in \Sigma^*$ is the still unread part of the input to the right of the input head, and $z \in \Delta^+$ is the already written part of the output, the rightmost symbol of z being the currently scanned symbol on the output tape. The *initial configuration* for input w is set to $(\lambda, q_0, w, \sqcup)$. During the course of its

computation, M runs through a sequence of configurations. One step from a configuration to its *successor configuration* is denoted by $\vdash$ and defined as follows. For $p \in Q$, $a \in \Sigma$, $v, w \in \Sigma^*$, $z \in \Delta^*$, and $b \in \Delta$, let (v, p, aw, zb) be a configuration. Then we define

$$\begin{aligned}
&(v,p,aw,zb) \vdash (va,q,w,zc), && \text{if } \delta(p,a,b) = (q,c,1,0),\\
&(v,p,aw,zb) \vdash (v,q,aw,zc), && \text{if } \delta(p,a,b) = (q,c,0,0),\\
&(v,p,aw,zb) \vdash (va,q,w,zc_{\sqcup}), && \text{if } \delta(p,a,b) = (q,c,1,1),\\
&(v,p,aw,zb) \vdash (v,q,aw,zc_{\sqcup}), && \text{if } \delta(p,a,b) = (q,c,0,1).
\end{aligned}$$

The reflexive transitive closure of $\vdash$ is denoted by $\vdash^*$.

A DFST *halts* if the transition function is undefined for the current configuration. The *output* written by a DFST M on input $w \in \Sigma^*$ is denoted by $M(w) \in (\Delta \setminus \{\sqcup\})^*$ and is defined as $M(w) = v$, if $(\lambda, q_0, w, \sqcup) \vdash^* (w, q, \lambda, v')$ with $q \in F$, v is the non-blank part of v', and M halts. Otherwise, $M(w)$ is defined to be the empty set. Now, the *transduction* defined by M is the set $T(M) = \{\, (w, M(w)) \mid w \in \Sigma^* \text{ and } M(w) \neq \emptyset \,\}$. We remark that M may also be considered as a partial function mapping some $w \in \Sigma^*$ to $v \in (\Delta \setminus \{\sqcup\})^*$. If we build the projection on the first components of $T(M)$, denoted by $L(M)$, then the transducer degenerates to a deterministic language acceptor.

In general, the family of all transductions performed by some device of type X is denoted by $\mathscr{T}(X)$.

Now, we turn to *reversible* DFST. Basically, reversibility is meant with respect to the possibility of stepping the computation back and forth. So, the machines have also to be backward deterministic. In particular, the machines reread the symbols which they have read in a preceding forward computation step. So, for reverse computation steps each head is either moved to the *left* or stays stationary. Figuratively, one can imagine that in a forward step, first the current symbols are read and then the heads are moved, whereas in a backward step first the heads are moved and then the symbols are read.

A DFST is said to be *reversible*, abbreviated as REV-DFST, if for any two *distinct* transitions

$$\begin{aligned}
\delta(p, x_0, x_1) &= (q, y_1, d_0, d_1) \quad \text{and}\\
\delta(p', x'_0, x'_1) &= (q', y'_1, d'_0, d'_1),
\end{aligned}$$

if $q = q'$, then $(d_0, d_1) = (d'_0, d'_1)$ and $(x_0, y_1) \neq (x'_0, y'_1)$. The first condition means that transitions reaching the same state have to move both heads in the same way. The second condition ensures that for any configuration the predecessor state can uniquely be determined from the state (which then implies the head movements), the input symbol read and the output symbol written.

In this paper, we consider in particular length-preserving DFST and differentiate between three notions. We call a DFST a *Mealy* transducer (M-DFST) if the transition function δ maps from $Q \times \Sigma \times \Delta$ to $Q \times (\Delta \setminus \{\sqcup\}) \times \{1\} \times \{1\}$. That is, in every time step an input symbol is read, an output symbol is written, and both heads proceed one position to the right. We call a DFST *strongly length-preserving* (s-DFST) if the transition function δ maps from $Q \times \Sigma \times \Delta$ to $Q \times (\Delta \setminus \{\sqcup\}) \times \{(1,1),(0,0)\}$. That is, both heads are moved synchronously. Finally, we call a DFST M *weakly length-preserving* (w-DFST), if $|w| = |M(w)|$, for all $(w, M(w)) \in T(M)$. That is, the length of the input word read and the length of the output word written is equal at the end of the transduction.

References

[1] Aho, A.V., Hopcroft, J.E., Ullman, J.D.: A general theory of translation. Math. Systems Theory 3, 193–221 (1969)

[2] Aho, A.V., Ullman, J.D.: The theory of parsing, translation, and compiling, vol. I: Parsing. Prentice-Hall (1972)

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

[4] Axelsen, H.B., Jakobi, S., Kutrib, M., Malcher, A.: A hierarchy of fast reversible Turing machines. In: Reversible Computation (RC 2015). LNCS, vol. 9138, pp. 29–44. Springer (2015)

[5] Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. 17, 525–532 (1973)

[6] Bensch, S., Björklund, J., Kutrib, M.: Deterministic stack transducers. In: Implementation and Application of Automata (CIAA 2016). LNCS, vol. 9705, pp. 27–38. Springer (2016)

[7] Berstel, J.: Transductions and Context-Free-Languages. Teubner (1979)

[8] Holzer, M., Jakobi, S., Kutrib, M.: Minimal reversible deterministic finite automata. In: Developments in Language Theory (DLT 2015). LNCS, vol. 9168, pp. 276–287. Springer (2015)

[9] Kondacs, A., Watrous, J.: On the power of quantum finite state automata. In: Foundations of Computer Science (FOCS 97). pp. 66–75. IEEE Computer Society (1997)

[10] Kutrib, M., Malcher, A.: Reversible pushdown automata. J. Comput. System Sci. 78, 1814–1827 (2012)

[11] Kutrib, M., Malcher, A.: One-dimensional cellular automaton transducers. Fundam. Inform. 126, 201–224 (2013)

[12] Kutrib, M., Malcher, A.: One-way reversible multi-head finite automata. Theor. Comput. Sci., to appear

[13] Landauer, R.: Irreversibility and heat generation in the computing process. IBM J. Res. Dev. 5, 183–191 (1961)

[14] Lange, K.J., McKenzie, P., Tapp, A.: Reversible space equals deterministic space. J. Comput. System Sci. 60, 354–367 (2000)

[15] Lecerf, Y.: Logique Mathématique: Machines de Turing réversible. C. R. Séances Acad. Sci. 257, 2597–2600 (1963)

[16] Morita, K.: Two-way reversible multi-head finite automata. Fund. Inform. 110, 241–254 (2011)

[17] Pin, J.E.: On reversible automata. In: Latin 1992: Theoretical Informatics. LNCS, vol. 583, pp. 401–416. Springer (1992)

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 53–53.

Decidability Questions for Insertion Systems

Andreas Malcher[(A)]

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

Zusammenfassung

Insertion systems or insertion grammars are a generative formalism in which words can only be generated by starting with some axioms and by iteratively inserting strings subject to certain contexts of a fixed maximal length. It is known that languages generated by such systems are always context sensitive and that the corresponding language classes are incomparable with the regular languages. On the other hand, it is possible to generate non-semilinear languages with systems having contexts of length two. Here, we study decidability questions for insertion systems. On the one hand, it can be seen that emptiness and universality is decidable. Moreover, the fixed membership problem is solvable in deterministic polynomial time. On the other hand, the usually studied decidability questions such as, for example, finiteness, inclusion, equivalence, regularity, inclusion in a regular language, and inclusion of a regular language turn out to be undecidable.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 54–54.

On Matching Restricted Patterns with Variables

Florin Manea

Kiel University,
flm@informatik.uni-kiel.de

Our talk covers a series of new results regarding restricted patterns with variables that can be matched efficiently.

In the first part of the talk we address unary patterns. More precisely, for a pattern $p = s_1x_1s_2x_2\cdots s_{r-1}x_{r-1}s_r$ such that $x_1, x_2, \ldots, x_{r-1} \in \{x, \overleftarrow{x}\}$, where x is a variable and $\overleftarrow{x}$ its reversal, and $s_1, s_2, \ldots, s_r$ are strings that contain no variables, we describe an algorithm that constructs in $O(rn)$ time a compact representation of all P instances of p in an input string of length n over a polynomially bounded integer alphabet, so that one can report those instances in $O(P)$ time.

If the pattern contains only a constant number of variables (e.g., generalized squares or cubes with terminals between the variables), our algorithm is asymptotically as efficient as the algorithms detecting fixed exponent (pseudo-)repetitions. For arbitrary patterns, our solution generalizes and improves the results of [3], where an $O(r^2n)$-time solution to the problem of finding one occurrence of a unary pattern with reversals (without terminals) was given. Here, compared to [3], we work with patterns that contain both variables and terminals and we detect, even faster, all their instances. Also, we improve the results of [2] in several directions: as said, we find all instances of a unary pattern (in [2] such a problem was solved as a subroutine in the algorithm detecting non-cross patterns, and only some instances of the patterns were found), our algorithm is faster by a $\log n$ factor, and our patterns also contain reversed variables. Our results were published in [4].

The efficient detection of unary patterns is then used to obtain efficient matching algorithms for a series of more general classes of patterns with variables (k-local patterns, nested patterns). These results are based on [1].

References

[1] J. D. DAY, P. FLEISCHMANN, F. MANEA, D. NOWOTKA, Local Patterns. *work in progress.* .

[2] H. FERNAU, F. MANEA, R. MERCAS, M. L. SCHMID, Pattern matching with variables: fast algorithms and new hardness results. In: *STACS 2015*. LIPIcs 30, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015, 302–315.

[3] P. GAWRYCHOWSKI, F. MANEA, D. NOWOTKA, Testing generalised freeness of words. In: *STACS 2014*. LIPIcs 25, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014, 337–349.

[4] D. KOSOLOBOV, F. MANEA, D. NOWOTKA, Detecting Unary Patterns. *CoRR* **abs/1604.00054** (2016). To appear in SPIRE 2017.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 55–57.

Deciding Equivalence of Tree Transducers

Sebastian Maneth

Universität Bremen, FB3 Mathematik/Informatik, Bibliothekstraße 5, 28359 Bremen
maneth@uni-bremen.de

Abstract

Three different techniques for deciding equivalence of tree transducers are discussed.

Tree transducers have been invented in the 1970s as formal model for compilers and mathematical linguistics. Since then, many different models have been considered, such as top-down tree transducers, bottom-up tree transducers, attributed tree transducers, and macro tree transducers. While equivalence is known to be decidable for the first two models (in the deterministic case), it remains open for the latter two. Here we consider, as in [8], equivalence for

(1) top-down tree transducers,

(2) finite-copying top-down tree-to-string transducers, and

(3) top-down tree-to-string transducers with monadic input.

The top-down tree transducer is a generalization of the finite-state string transducer (aka GSM) from strings to ranked ordered trees. Equivalence of nondeterministic GSMs is known to be undecidable, even for ϵ-free transducers [4]. Hence, also equivalence of nondeterministic top-down tree transducers is undecidable. For deterministic top-down tree transducers it was proved already in 1980 by Zoltan Ésik [3] that equivalence is decidable. The proof relies on the fact that during a computation, one transducer can only be boundedly "ahead" of the other transducer. The different aheadnesses can be recorded in the states of a tree automaton.

Ésik [3] did not study the complexity of his decision procedure (it requires at least exponential time). A completely different way of deciding equivalence of deterministic top-down tree transducers is given in [2]. The idea is to transform the given transducers into a certain canonical normal form, and then to test isomorphism (i.e., whether there exists a state renaming that transforms one transducer into the other). The normal form demands that output is produced as *early* as possible. More specifically, if there is a common prefix in the rules of a given state for all input symbols, then this transducer is *not earliest*, because the common prefix should have been produced earlier (as it does not depend on the label of the current input node). It can be shown that if a transducer is total (for each state and input symbol there exists a rule), then its canonical earliest form can be constructed in polynomial time; thus, equivalende for total deterministic top-down tree transducers can be decided in polynomial time. The canonical normal form has been extended to a Myhill-Nerod theorem [7] which can be used for MAT/Gold style learning.

The second type of equivalence problem relies on properties of semi-linear sets. A top-down tree transducer can translate a monadic tree into a full binary tree, i.e., can realize a translation of exponential size increase. This implies that its range is *not* semi-linear. If we restrict top-down tree transductions (with look-ahead) to *linear size increase*, then we obtain the class of finite-copying top-down tree transducers with regular look-ahead [1], the ranges of which are semi-linear. The latter even holds for top-down *tree-to-string* transducers. Thus, we can decide equivalence of finite-copying top-down tree-to-string transducers with regular look-ahead (which are equivalent to MSO-definable tree-to-string translations). The particular property of semi-linearity that is being used, is the problem of intersection emptiness with the set $\{(n,n) \mid n \geq 1\}$. We change and combine the given transducers M_1, M_2 so that on a common input tree the resulting transducer produces $a^n b^m$ if and only if M_1 produces the letter a in the output string at position n, and M_2 produces the letter b at position m.

In the third kind of problem we consider top-down tree-to-string transducers with monadic input (i.e., string-to-string transducers). It should be noted that deciding equivalence of transducers with string output is far more difficult than of transducers with tree output. E.g., while equivalence of deterministic top-down tree transducers was solved in 1980, the corresponding tree-to-string problem had been a big open problem, and was only recently solved in 2015 [10]. Our third problem here is the younger cousin of the latter problem, in the sense that the input is a monadic tree (i.e., a string) instead of a tree. Here, equivalence can be reduced to the so called "sequence equivalence problem of HDT0L systems" (these are a variant of L (Lindemeyer) systems), which is known to be decidable. The first proof was by Culik II and Karhumäki [6], using Ehrenfeucht's Conjecture and Makanin's algorithm. The later proof of Ruohonen [9] is based on the theory of metabelian groups. The final (very short) proof by Honkala [5] relies on Hilbert's Basis Theorem. The stronger result mentioned above about tree-to-string transducers also uses Hilbert's Basis Theorem but relies on polynomial ideals.

References

[1] J. ENGELFRIET, S. MANETH, Macro Tree Translations of Linear Size Increase are MSO Definable. *SIAM J. Comput.* **32** (2003) 4, 950–1006.
https://doi.org/10.1137/S0097539701394511

[2] J. ENGELFRIET, S. MANETH, H. SEIDL, Deciding equivalence of top-down XML transformations in polynomial time. *J. Comput. Syst. Sci.* **75** (2009) 5, 271–286.
https://doi.org/10.1016/j.jcss.2009.01.001

[3] Z. ÉSIK, Decidability results concerning tree transducers I. *Acta Cybern.* **5** (1980) 1, 1–20.
http://www.inf.u-szeged.hu/actacybernetica/edb/vol05n1/Esik_1980_ActaCybernetica.xml

[4] T. V. GRIFFITHS, The Unsolvability of the Equivalence Problem for Lambda-Free Nondeterministic Generalized Machines. *J. ACM* **15** (1968) 3, 409–413.

[5] J. HONKALA, A short solution for the HDT0L sequence equivalence problem. *Theor. Comput. Sci.* **244** (2000) 1-2, 267–270.
https://doi.org/10.1016/S0304-3975(00)00158-4

[6] K. C. II, J. KARHUMÄKI, The Equivalence of Finite Valued Transducers (On HDT0L Languages) is Decidable. *Theor. Comput. Sci.* **47** (1986) 3, 71–84.
https://doi.org/10.1016/0304-3975(86)90134-9

[7] A. LEMAY, S. MANETH, J. NIEHREN, A learning algorithm for top-down XML transformations. In: *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*. 2010, 285–296.
http://doi.acm.org/10.1145/1807085.1807122

[8] S. MANETH, A Survey on Decidable Equivalence Problems for Tree Transducers. *Int. J. Found. Comput. Sci.* **26** (2015) 8, 1069–1100.
https://doi.org/10.1142/S0129054115400134

[9] K. RUOHONEN, *Equivalence Problems for Regular sets of Word Morphisms*. Springer, Berlin, Heidelberg, 1986, 393–401.
https://doi.org/10.1007/978-3-642-95486-3_33

[10] H. SEIDL, S. MANETH, G. KEMPER, Equivalence of Deterministic Top-Down Tree-to-String Transducers is Decidable. In: *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. 2015, 943–962.
https://doi.org/10.1109/FOCS.2015.62

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 58–58.

An Automaton Learning Approach to Solving Safety Games over Infinite Graphs

Daniel Neider[(A)]

[(A)]Max-Planck-Institut für Software-Systeme, 67663 Kaiserslautern, Deutschland
neider@mpi-sws.org

Abstract

We propose a method to construct finite-state reactive controllers for systems whose interactions with their adversarial environment are modeled by infinite-duration two-player games over (possibly) infinite graphs. The method targets safety games with infinitely many states or with such a large number of states that it would be impractical—if not impossible—for conventional synthesis techniques that work on the entire state space. We resort to constructing finite-state controllers for such systems through an automata learning approach, utilizing a symbolic representation of the underlying game that is based on finite automata. Throughout the learning process, the learner maintains an approximation of the winning region (represented as a finite automaton) and refines it using different types of counterexamples provided by the teacher until a satisfactory controller can be derived (if one exists). We present a symbolic representation of safety games (inspired by regular model checking), propose implementations of the learner and teacher, and evaluate their performance on examples motivated by robotic motion planning.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 59–62.

On Deterministic Ordered Restart-Delete Automata

Friedrich Otto

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

Abstract

We consider ordered restart-delete automata, which are deterministic ORWW-automata that have an additional delete operation. We show that these automata don't need states, that they accept all deterministic context-free languages, and that they even accept some languages that are not deterministic context-free. On the other hand, these automata only accept languages that are at the same time context-free and Church-Rosser. In addition, closure properties for the class of languages accepted by deterministic ordered restart-delete automata are studied.

1. Introduction

An *ordered restarting automaton* consists of a finite-state control, a tape with endmarkers, a read-write window of size three, and a (partial) ordering on its tape alphabet. Based on the actual state and window contents, the automaton can move its window one position to the right and change its state, or it can replace the letter in the middle of the window by a smaller letter and restart, or it can accept. During a restart the window is placed onto the left end of the tape, and the finite-state control is reset to the initial state. In [5] it is shown that deterministic ordered restarting automata (det-ORWW-automata) don't need states and that they just characterize the class of regular languages. In fact, these automata yield very compact descriptions for (some) regular languages, as there exist det-ORWW-automata with n letters such that the smallest NFAs for the corresponding languages have $2^{O(n)}$ states [3]. On the other hand, the nondeterministic ordered restarting automata define an abstract family of languages that properly extends the regular languages, but that is incomparable to the (deterministic) linear languages, the (deterministic) context-free languages, the Church-Rosser languages, and the growing context-sensitive languages with respect to inclusion [4].

Here we extend the det-ORWW-automaton by introducing an additional delete/restart operation that allows to delete the symbol from the middle of the window and to restart, obtaining the *deterministic ordered restart-delete automaton* (or det-ORD-automaton, for short). This extension has surprisingly strong implications as we will see below.

2. Definition and Example

A *det-ORD-automaton* is described by an 8-tuple $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite tape alphabet such that $\Sigma \subseteq \Gamma$, the symbols $\rhd, \lhd \notin \Gamma$ serve as markers for the left and right border of the work space, respectively, $q_0 \in Q$ is the initial state,

$$\delta : Q \times (((\Gamma \cup \{\rhd\}) \cdot \Gamma \cdot (\Gamma \cup \{\lhd\})) \cup \{\rhd\lhd\}) \to (Q \times \{\mathsf{MVR}\}) \cup (\Gamma \cup \{\lambda\}) \cup \{\mathsf{Accept}\}$$

is the *transition function*, and $>$ is a *partial ordering* on Γ. The transition function describes four different types of transition steps:

(1) A *move-right step* has the form $\delta(q, a_1a_2a_3) = (q', \mathsf{MVR})$, where $q, q' \in Q$, $a_1 \in \Gamma \cup \{\rhd\}$ and $a_2, a_3 \in \Gamma$. It causes M to shift the window one position to the right and to enter state q'.

(2) A *rewrite/restart step* has the form $\delta(q, a_1a_2a_3) = b$, where $q \in Q$, $a_1 \in \Gamma \cup \{\rhd\}$, $a_2, b \in \Gamma$, and $a_3 \in \Gamma \cup \{\lhd\}$ such that $a_2 > b$ holds. It causes M to replace the symbol a_2 in the middle of its window by the symbol b and to restart.

(3) A *delete/restart step* has the form $\delta(q, a_1a_2a_3) = \lambda$ (the empty word), where $q \in Q$, $a_1 \in \Gamma \cup \{\rhd\}$, $a_2 \in \Gamma$, and $a_3 \in \Gamma \cup \{\lhd\}$. It causes M to delete the symbol a_2 in the middle of its window and to restart. Through this step, the tape field that contains a_2 is also removed, which means that the length of the tape is reduced by one.

(4) An *accept step* has the form $\delta(q, a_1a_2a_3) = \mathsf{Accept}$, where $q \in Q$, $a_1 \in \Gamma \cup \{\rhd\}$, $a_2 \in \Gamma$, and $a_3 \in \Gamma \cup \{\lhd\}$. It causes M to halt and accept. In addition, we allow an accept step of the form $\delta(q_0, \rhd\lhd) = \mathsf{Accept}$.

If $\delta(q, u)$ is undefined for some pair (q, u), then M necessarily halts, when it is in state q with the word u in its window, and we say that M *rejects* in this situation. Further, the letters in $\Gamma \setminus \Sigma$ are called *auxiliary symbols*. The det-ORD-automaton M is called *stateless* if $Q = \{q_0\}$. For a stateless det-ORD-automaton, we drop the components Q and q_0 from its description, and we abbreviate it as stl-det-ORD-automaton.

A *configuration* of a det-ORD-automaton M is a word $\alpha q \beta$, where $q \in Q$ and $|\beta| \geq 3$, and either $\alpha = \lambda$ and $\beta \in \{\rhd\} \cdot \Gamma^+ \cdot \{\lhd\}$ or $\alpha \in \{\rhd\} \cdot \Gamma^*$ and $\beta \in \Gamma \cdot \Gamma^+ \cdot \{\lhd\}$; here $q \in Q$ represents the current state, $\alpha\beta$ is the current content of the tape, and it is understood that the window contains the first three symbols of β. In addition, we admit the configuration $q_0 \rhd \lhd$. A *restarting configuration* has the form $q_0 \rhd w \lhd$; if $w \in \Sigma^*$, then $q_0 \rhd w \lhd$ is also called an *initial configuration*. A configuration that is reached by an accept step is an *accepting configuration*, denoted by Accept. By $\vdash_M$ we denote the *single-step computation relation* that M induces on its set of configurations, and $\vdash_M^*$, its reflexive and transitive closure, is the *computation relation* of M. An input $w \in \Sigma^*$ is accepted by M, if the computation of M which starts with the initial configuration $q_0 \rhd w \lhd$ ends with an accept step. The language consisting of all words that are accepted by M is denoted by $L(M)$.

After executing a restart step, a det-ORD-automaton performs MVR-steps until its window contains the newly written symbol or the first symbol to the right of the one just deleted. It follows that M can actually be simulated by a deterministic single-tape Turing machine in time $O(n)$.

Theorem 2.1 *For each det-ORD-automaton $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$, there exists a stl-det-ORD-automaton $M' = (\Sigma, \Delta, \rhd, \lhd, \delta', >')$ such that $L(M') = L(M)$ and $|\Delta| = |Q| \cdot |\Gamma|^2 + 2 \cdot |\Gamma|$.*

Accordingly we restrict our attention to stl-det-ORD-automata. The following example illustrates the way in which a stl-det-ORWW-automaton works.

Example 2.2 *Let $L = \{ a^n b^n c \mid n \geq 0 \} \cup \{ a^n b^{2n} d \mid n \geq 0 \}$. It is well-known that L is a context-free language that is not deterministic context-free.*

We present a stl-det-ORD-automaton $M = (\Sigma, \Gamma, \rhd, \lhd, \delta, >)$ for L that is defined by taking $\Sigma = \{a, b, c, d\}$ and $\Gamma = \Sigma \cup \{ a_1, e, e_1, f, f_1, f_2 \}$, by choosing the ordering $>$ such that $a > a_1$, $b > e > e_1$, and $b > f > f_1 > f_2$, and by defining the transition function δ as follows:

(1) $\delta(\rhd c \lhd) = \mathsf{Accept}$,	(17) $\delta(\rhd a_1 e) = \mathsf{MVR}$,	(33) $\delta(a_1 ee) = e_1$,
(2) $\delta(\rhd d \lhd) = \mathsf{Accept}$,	(18) $\delta(a_1 ec) = e_1$,	(34) $\delta(aa_1 e_1) = \lambda$,
(3) $\delta(\rhd ab) = \mathsf{MVR}$,	(19) $\delta(\rhd a_1 e_1) = \lambda$,	(35) $\delta(aae_1) = \mathsf{MVR}$,
(4) $\delta(abc) = e$,	(20) $\delta(\rhd e_1 c) = \lambda$,	(36) $\delta(ae_1 e) = \lambda$,
(5) $\delta(\rhd aa) = \mathsf{MVR}$,	(21) $\delta(\rhd af) = a_1$,	(37) $\delta(aa_1 f) = \mathsf{MVR}$,
(6) $\delta(aaa) = \mathsf{MVR}$,	(22) $\delta(\rhd a_1 f) = \mathsf{MVR}$,	(38) $\delta(aae) = a_1$,
(7) $\delta(aab) = \mathsf{MVR}$,	(23) $\delta(a_1 ff) = f_1$,	(39) $\delta(aaa_1) = \mathsf{MVR}$,
(8) $\delta(abb) = \mathsf{MVR}$,	(24) $\delta(\rhd a_1 f_1) = \mathsf{MVR}$,	(40) $\delta(aaf) = a_1$,
(9) $\delta(bbb) = \mathsf{MVR}$,	(25) $\delta(a_1 f_1 f) = \mathsf{MVR}$,	(41) $\delta(a_1 ff) = f_1$,
(10) $\delta(bbc) = e$,	(26) $\delta(f_1 fd) = f_2$,	(42) $\delta(aa_1 f_1) = \mathsf{MVR}$,
(11) $\delta(bbd) = f$,	(27) $\delta(\rhd a_1 f_2) = \lambda$,	(43) $\delta(f_1 ff) = f_2$,
(12) $\delta(bbe) = e$,	(28) $\delta(a_1 f_1 f_2) = \lambda$,	(44) $\delta(aa_1 f_2) = \lambda$,
(13) $\delta(bbf) = f$,	(29) $\delta(aa_1 f_2) = \lambda$,	(45) $\delta(aaf_2) = \mathsf{MVR}$,
(14) $\delta(abe) = e$,	(30) $\delta(\rhd f_2 d) = \lambda$,	(46) $\delta(af_2 f) = \lambda$,
(15) $\delta(abf) = f$,	(31) $\delta(\rhd aa_1) = \mathsf{MVR}$,	(47) $\delta(\rhd ae_1) = \mathsf{MVR}$,
(16) $\delta(\rhd ae) = a_1$,	(32) $\delta(aa_1 e) = \mathsf{MVR}$,	(48) $\delta(\rhd af_2) = \mathsf{MVR}$.

Given $c = a^0 b^0 c$ or $d = a^0 b^{2 \cdot 0} d$ as input, M accepts immediately using rules (1) or (2). On input abc, M proceeds as follows, where we underline the word currently in the window of M:

$$\underline{\rhd abc} \lhd \vdash_{(3)} \rhd \underline{abc} \lhd \vdash_{(4)} \underline{\rhd aec} \lhd \vdash_{(16)} \underline{\rhd a_1 ec} \lhd \vdash_{(17)} \rhd \underline{a_1 ec} \lhd$$
$$\vdash_{(18)} \underline{\rhd a_1 e_1 c} \lhd \vdash_{(19)} \underline{\rhd e_1 c} \lhd \vdash_{(20)} \underline{\rhd c \lhd} \vdash_{(1)} \mathsf{Accept},$$

and on input $abbd$, M proceeds as follows:

$$\underline{\rhd abbd} \lhd \vdash_{(3)} \rhd \underline{abbd} \lhd \vdash_{(8)} \rhd \underline{abbd} \lhd \vdash_{(11)} \underline{\rhd abfd} \lhd \vdash_{(3)} \rhd \underline{abfd} \lhd$$
$$\vdash_{(15)} \underline{\rhd affd} \lhd \vdash_{(21)} \underline{\rhd a_1 ffd} \lhd \vdash_{(22)} \rhd \underline{a_1 ffd} \lhd \vdash_{(23)} \underline{\rhd a_1 f_1 fd} \lhd$$
$$\vdash_{(24)} \rhd \underline{a_1 f_1 fd} \lhd \vdash_{(25)} \rhd a_1 \underline{f_1 fd} \lhd \vdash_{(26)} \underline{\rhd a_1 f_1 f_2 d} \lhd \vdash_{(24)} \rhd \underline{a_1 f_1 f_2 d} \lhd$$
$$\vdash_{(28)} \underline{\rhd a_1 f_2 d} \lhd \vdash_{(27)} \underline{\rhd f_2 d} \lhd \vdash_{(30)} \underline{\rhd d \lhd} \vdash_{(4)} \mathsf{Accept}.$$

Thus, if the input is of the form $a^m b^n c$, then the factor b^n is rewritten, from right to left, into e^n, and then by alternatingly rewriting the last letter a into a_1 and the first letter e into e_1, it is checked whether $m = n$. On the other hand, if the input is of the form $a^m b^n d$, then the factor b^n is rewritten, from right to left, into f^n, and then by alternatingly rewriting the last letter a into a_1 and the first factor ff into $f_1 f_2$, it is checked whether $n = 2m$. Hence, it follows that $L(M) = L$.

3. Results

The det-ORD-automaton is a special type of deterministic shrinking RWW-automaton, and as the class of languages accepted by the latter coincides with the class CRL of Church-Rosser languages [2], we see that $\mathcal{L}(\text{det-ORD}) \subseteq \text{CRL}$. On the other hand, each DPDA can be simulated by a det-ORD-automaton.

Theorem 3.1 $\text{DCFL} \subsetneq \mathcal{L}(\text{stl-det-ORD})$.

Theorem 3.2 *The language class* $\mathcal{L}(\text{det-ORD})$ *is closed under the operations of complement, reversal, and intersection with regular languages, but it is not closed under union or intersection.*

A language $L \subseteq \Sigma^*$ belongs to the class LRR of *left-to-right regular languages* of [1] if there exists a deterministic context-free language $L' \subseteq \Delta^*$ and a right-to-left sequential transducer S such that, for all $w \in \Sigma^*$, $w \in L$ iff $S(w) \in L'$. As such a transducer can easily be combined with a det-ORD-automaton for the language L', we see that $\text{LRR} \subseteq \mathcal{L}(\text{det-ORD})$ holds. Finally, the language $L_{3blocks} = \{\, a^m b^m a^n b^n a^p b^p, a^m b^{2m} a^{n+k} b^n a^p b^{2p} \mid m, n, p, k \geq 1 \,\}$ does not belong to the class LRR and neither does its reversal $L_{3blocks}^R$ [1]. However, this language is accepted by a det-ORD-automaton, which yields the inclusion $\text{LRR} \cup \text{RLR} \subsetneq \mathcal{L}(\text{det-ORD})$, where $\text{RLR} = \text{LRR}^R$. Finally, it can be shown that the accepting computations of a stl-det-ORD-automaton can be simulated by an unambiguous PDA.

Theorem 3.3 $\mathcal{L}(\text{det-ORD}) \subsetneq \text{UCFL}$.

Thus, in summary we have the following chain of inclusions, where Co-UCFL denotes the class of languages that are the complements of unambiguous context-free languages.

Corollary 3.4 $\text{DCFL} \cup \text{DCFL}^R \subsetneq \text{LRR} \cup \text{RLR} \subsetneq \mathcal{L}(\text{det-ORD}) \subseteq \text{CRL} \cap \text{UCFL} \cap \text{Co-UCFL}$.

References

[1] K. CULIK, II, R. COHEN, LR-regular grammars - an extension of LR(k) grammars. *Journal of Computer and System Sciences* **7** (1973), 66–96.

[2] T. JURDZIŃSKI, F. OTTO, Shrinking restarting automata. *International Journal of Foundations of Computer Science* **18** (2007), 361–385.

[3] K. KWEE, F. OTTO, On some decision problems for stateless deterministic ordered restarting automata. In: J. SHALLIT, A. OKHOTIN (eds.), *DCFS 2015, Proc.*. Lecture Notes in Computer Science 9118, Springer, Heidelberg, 2015, 165–176.

[4] K. KWEE, F. OTTO, On the effects of nondeterminism on ordered restarting automata. In: R. FREIVALDS, G. ENGELS, B. CATANIA (eds.), *SOFSEM 2016, Proc.*. Lecture Notes in Computer Science 9587, Springer, Heidelberg, 2016, 369–380.

[5] F. OTTO, On the descriptional complexity of deterministic ordered restarting automata. In: H. JÜRGENSEN, J. KARHUMÄKI, A. OKHOTIN (eds.), *DCFS 2014, Proc.*. Lecture Notes in Computer Science 8614, Springer, Heidelberg, 2014, 318–329.

THEORIE-TAG 2017

H. Fernau (Herausgeber): Theorietag 2017, Bonn, 18. – 22.9.2017
Universität Trier, Technischer Bericht, S. 63–66.

On the Expressive Power of Weighted Restarting Automata

Qichao Wang

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

1. Introduction

Analysis by reduction is a linguistic technique that is used to check the correctness of sentences of natural languages through sequences of local simplifications. Restarting automata have been introduced as a formal model for the analysis by reduction [1]. In order to study quantitative aspects of computations of restarting automata, *weighted restarting automata* were introduced in [4].

A weighted restarting automaton $\mathcal{M}$ is given by a pair (M,ω), where M is a restarting automaton on some finite input alphabet Σ, and ω is a weight function that assigns a weight from some semiring S to each transition of M. The product (in S) of the weights of all transitions that are used in a computation yields a weight for that computation, and by forming the sum over the weights of all accepting computations for a given input $w \in \Sigma^*$, a value from S is assigned to w. Thus, a partial function $f : \Sigma^* \to S$ is obtained. By placing a condition T on the value $f(w)$, some words from the language $L(M)$ that is accepted by the underlying (unweighted) restarting automaton M are filtered out. In this way, we can define a sublanguage $L_T(\mathcal{M})$ of the language $L(M)$ by combining the acceptance condition of M with a condition T on the weight $f(w)$ for $w \in L(M)$. We will see that by using a relative acceptance condition the expressive power of a restarting automaton can be increased, and a restarting automaton of a weak type can simulate a restarting automaton of a strong type.

2. Definitions and Examples

A *restarting automaton* (or RRWW-automaton) is a nondeterministic machine with a finite-state control, a flexible tape with endmarkers, and a read/write window [1]. Formally, it is described by an 8-tuple $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite tape alphabet containing Σ, the symbols $¢, \$ \notin \Gamma$ are used as markers for the left and right border of the work space, respectively, $q_0 \in Q$ is the initial state, $k \geq 1$ is the size of the *read/write window*, and δ is the (partial) *transition relation* that associates finite sets of transition steps to pairs of the form (q, u), where q is a state and u is a possible content

of the read/write window. There are four types of transition steps. A *move-right step* (MVR) causes M to shift its read/write window one position to the right and to change the state. A *rewrite step* causes M to replace the content u of the read/write window by a shorter string v, thereby reducing the length of the tape, and to change the state. Further, the read/write window is placed immediately to the right of the string v. However, occurrences of the delimiters ¢ and \$ can neither be deleted nor newly created by a rewrite step. A *restart step* causes M to place its read/write window over the left end of the tape, so that the first symbol it sees is the left sentinel ¢, and to reenter the initial state q_0, and, finally, an *accept step* causes M to halt and accept.

If $\delta(q,u)$ is undefined for some pair (q,u), then M necessarily halts in a corresponding situation, and we say that M *rejects*. Finally, if each rewrite step is combined with a restart step into a joint rewrite/restart operation, then M is called an RWW-automaton. An RRWW-automaton is called an RRW-automaton if its tape alphabet Γ coincides with its input alphabet Σ, that is, if no auxiliary symbols are available. It is an RR-automaton if it is an RRW-automaton for which the right-hand side v of each rewrite step $(q',v) \in \delta(q,u)$ is a scattered subword of the left-hand side u. Analogously, we obtain the RW-automaton and the R-automaton from the RWW-automaton.

A *configuration* of M is a string $\alpha q \beta$, where $q \in Q$, and either $\alpha = \lambda$ and $\beta \in \{¢\} \cdot \Gamma^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot \Gamma^*$ and $\beta \in \Gamma^* \cdot \{\$\}$; here q is the current state, and $\alpha\beta$ is the current content of the tape, where it is understood that the window contains the first k symbols of β or all of β when $|\beta| \leq k$. A *restarting configuration* is of the form $q_0¢w\$$. If $w \in \Sigma^*$, then $q_0¢w\$$ is an *initial configuration*.

Any computation of M consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration, the head moves along the tape performing move-right operations and a single rewrite operation until a restart operation is performed and thus a new restarting configuration is reached. If no further restart operation is performed, the computation necessarily finishes in a halting configuration – such a phase is called a *tail*. It is required that in each cycle M performs *exactly one* rewrite step. A word $w \in \Sigma^*$ is accepted by M, if there is an accepting computation which starts from the initial configuration $q_0¢w\$$. By $L(M)$ we denote the language consisting of all (input) words that are accepted by M.

For studying quantitative aspects of computations of restarting automata, the weighted restarting automaton has been introduced in [4]. A *weighted restarting automaton* of type X, a wX-automaton for short, is a pair (M,ω), where M is a restarting automaton of type X, and ω is a *weight function* from the transitions of M into a semiring S, that is, ω assigns an element $\omega(t) \in S$ as a weight to each transition t of M. The product (in S) of the weights of all transitions that are used in a computation then yields a weight for that computation, and the sum over all weights of all accepting computations of M for a given input word $w \in \Sigma^*$ yields a value from S. In this way, a partial function $f_\omega^M : \Sigma^* \to S$ is obtained. Here we use weighted restarting automata to define sublanguages of the language that is accepted by the underlying (unweighted) restarting automaton.

Definition 2.1 *Let $M = (Q,\Sigma,\Gamma,¢,\$,q_0,k,\delta)$ be a restarting automaton, let ω be a weight function from M into a semiring S, and let $\mathcal{M} = (M,\omega)$. For a subset T of S, $L_T(\mathcal{M}) = \{\, w \in L(M) \mid f_\omega^M(w) \in T \,\}$ is the language* accepted by M relative to T, *that is, a word $w \in \Sigma^*$ belongs to the language $L_T(\mathcal{M})$ iff $w \in L(M)$ and $f_\omega^M(w) \in T$.*

Definition 2.2 *Let* X *be a type of restarting automaton, let* S *be a semiring, and let* $\mathbb{H}$ *be a family of subsets of* S*. Then*

$$\mathcal{L}(\mathsf{X}, S, \mathbb{H}) = \{\, L_T(\mathcal{M}) \mid \mathcal{M} \text{ is a weighted restarting automaton of type } \mathsf{X}, \text{ and } T \in \mathbb{H} \,\}$$

is the class of languages that are accepted by weighted restarting automata of type X *relative to* $\mathbb{H}$*.*

We continue with an example that illustrates our definitions.

Example 2.3 *Let* $M_1 = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ *be the* RW*-automaton that is defined by taking* $Q = \{q_0, q_r\}$, $\Gamma = \Sigma = \{a, b, c, d\}$, *and* $k = 5$, *where* δ *is defined as follows:*

$$\begin{array}{llll}
t_1 : (q_0, ¢aaaa) \to (q_0, \mathsf{MVR}), & & t_{15} : (q_0, ¢aaac) \to (q_0, \mathsf{MVR}), \\
t_2 : (q_0, aaaaa) \to (q_0, \mathsf{MVR}), & & t_{16} : (q_0, ¢aaad) \to (q_0, \mathsf{MVR}), \\
t_3 : (q_0, aaaab) \to (q_0, \mathsf{MVR}), & & t_{17} : (q_0, ¢acbc) \to (q_r, ¢cc), \\
t_4 : (q_0, aaabb) \to (q_0, \mathsf{MVR}), & & t_{18} : (q_0, ¢adbb) \to (q_0, \mathsf{MVR}), \\
t_5 : (q_0, aabbb) \to (q_r, acbb), & & t_{19} : (q_0, adbbd) \to (q_r, dd), \\
t_6 : (q_0, aabbb) \to (q_r, adb), & & t_{20} : (q_0, ¢aabb) \to (q_0, \mathsf{MVR}), \\
t_7 : (q_0, aaaac) \to (q_0, \mathsf{MVR}), & & t_{21} : (q_0, aabbc) \to (q_r, acbc), \\
t_8 : (q_0, aaacb) \to (q_0, \mathsf{MVR}), & & t_{22} : (q_0, ¢abc\$) \to (q_r, ¢cc\$), \\
t_9 : (q_0, aacbb) \to (q_r, acb), & & t_{23} : (q_0, ¢abbd) \to (q_r, ¢dd), \\
t_{10} : (q_0, aaaad) \to (q_0, \mathsf{MVR}), & & t_{24} : (q_0, ¢cc\$) \to \mathsf{Accept}, \\
t_{11} : (q_0, aaadb) \to (q_0, \mathsf{MVR}), & & t_{25} : (q_0, ¢dd\$) \to \mathsf{Accept}, \\
t_{12} : (q_0, aadbb) \to (q_0, \mathsf{MVR}), & & t_{26} : (q_0, ¢c\$) \to \mathsf{Accept}, \\
t_{13} : (q_0, adbbb) \to (q_r, db), & & t_{27} : (q_0, ¢d\$) \to \mathsf{Accept}, \\
t_{14} : (q_0, ¢aaab) \to (q_0, \mathsf{MVR}), & & t_{28,x} : (q_r, x) \to \mathsf{Restart} \text{ for all admissible } x.
\end{array}$$

It is easily seen that $L(M_1) = \{\, a^n c^i b^n c, a^n d^i b^{2n} d \mid n \geq 0, i \in \{0, 1\} \,\}$, *and that for each input,* M_1 *has just a single accepting computation. Let* $\mathbb{N}_\infty = (\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$ *be the* tropical semiring, *and let* ω_1 *be the weight function from the transitions of* M_1 *into the semiring* $\mathbb{N}_\infty$ *that is defined as follows*

$$\omega_1(t_i) = \begin{cases} 1, & \text{if } i \in \{5, 6, 21, 22, 23, 24, 25, 26, 27\}, \\ 0, & \text{otherwise.} \end{cases}$$

Let $\mathcal{M}_1 = (M_1, \omega_1)$, *let* $L_1 = \{\, a^n c b^n c, a^n d b^{2n} d \mid n \geq 0 \,\}$, *and* $L_2 = \{\, a^n b^n c, a^n b^{2n} d \mid n \geq 0 \,\}$. *It is rather clear that* $L(M_1) = L_1 \cup L_2$. *Given an input word* $w \in \Sigma^*$, $f_{\omega_1}^{M_1}(w) = 1$ *if and only if* $w \in L_1$, *and* $f_{\omega_1}^{M_1}(w) = 2$ *if and only if* $w \in L_2$. *Further, we take* $T_1 = \{2\}$. *It follows that*

$$L_{T_1}(\mathcal{M}_1) = L_2 = \{\, a^n b^n c, a^n b^{2n} d \mid n \geq 0 \,\}.$$

It is known that $L_{T_1}(\mathcal{M}_1)$ *is not accepted by any* RW*-automaton [2]. Hence, we see that the notion of relative acceptance increases the expressive power of* RW*-automata.*

3. Results

First, we study the classes of languages that are accepted by weighted restarting automata relative to some semirings over integers. We will see that by using the tropical semiring $\mathbb{Z}_\infty = (\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$, we can avoid the use of auxiliary symbols in rewrite steps [7], and that a #P-complete function can be computed by a weighted R-automaton relative to subsets of the semiring $\overline{\mathbb{N}} \times \mathbb{N}$, i.e., the direct product of the semirings $\overline{\mathbb{N}} = (\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$ and $\mathbb{N} = (\mathbb{N}, +, \cdot, 0, 1)$ [5].

Further, we study the case that S is the semiring of formal languages $\mathbb{P}(\Delta^*) = (\mathbb{P}(\Delta^*), \cup, \cdot, \emptyset, \{\lambda\})$, and its subsets such as the semiring of regular languages $\mathsf{REG}(\Delta) = (\mathsf{REG}(\Delta), \cup, \cdot, \emptyset, \{\lambda\})$. As the weight of a transition of a restarting automaton can be any language over Δ, the general model of weighted restarting automata is quite powerful. Therefore, some more restricted types of weighted restarting automata such as *word-weighted* restarting automata and *regular-weighted* restarting automata were introduced in [5, 6]. We prove that the class of languages accepted by word-weighted restarting automata with the *weak* acceptance condition coincides with the class of languages accepted by *non-forgetting* restarting automata of the corresponding type [3, 7]. In addition, we show that RRWW-automata can be simulated by regular-weighted RW-automata with the weak acceptance condition [5].

Finally, word-weighted restarting automata with the *strong* acceptance condition turn to be more expressive than the ones with the weak acceptance condition [5]. In particular, the classes of languages accepted by word-weighted RWW- and RRWW-automata with the strong acceptance condition are closed under the operation of intersection [7]. This is the first result that shows that a class of languages defined in terms of a quite general class of restarting automata is closed under the operation of intersection.

References

[1] P. JANCAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, Restarting Automata. In: H. REICHEL (ed.), *FCT*. LNCS 965, Springer, Heidelberg, 1995, 283–292.

[2] P. JANCAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, On Monotonic Automata with a Restart Operation. *J. Auto. Lang. Comb.* **4** (1999) 4, 287–312.

[3] H. MESSERSCHMIDT, F. OTTO, On Nonforgetting Restarting Automata That Are Deterministic and/or Monotone. In: D. GRIGORIEV, J. HARRISON, E. A. HIRSCH (eds.), *CSR*. LNCS, Springer, Heidelberg, 2006, 247–258.

[4] F. OTTO, Q. WANG, Weighted Restarting Automata. *Soft Computing* (2016). DOI: 10.1007/s00500-016-2164-4. The results of this paper have been announced at WATA 2014 in Leipzig.

[5] Q. WANG, *On the expressive power of weighted restarting automata*. NCMA, 2017. Accepted.

[6] Q. WANG, F. OTTO, Weighted restarting automata and pushdown relations. *Theor. Comput. Sci.* **635** (2016), 1–15.

[7] Q. WANG, F. OTTO, Weighted Restarting Automata as Language Acceptors. In: Y. HAN, K. SALOMAA (eds.), *CIAA*. LNCS 9705, Springer, Switzerland, 2016, 298–309.