INSTITUT FÜR INFORMATIK

26. Theorietag *Automaten und Formale Sprachen*
23. Jahrestagung *Logik in der Informatik*

Tagungsband

Florin Manea, Dirk Nowotka, Thomas Wilke (Hrsg.)

Bericht Nr. 1606
Oktober 2016
ISSN 2192-6247

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

26. Theorietag *Automaten und Formale Sprachen*
23. Jahrestagung *Logik in der Informatik*

Tagungsband

Florin Manea, Dirk Nowotka, Thomas Wilke (Hrsg.)

Bericht Nr. 1606
Oktober 2016
ISSN 2192-6247

e-mail: {flm,dn}@zs.uni-kiel.de,
thomas.wilke@email.uni-kiel.de

26. Theorietag *Automaten und Formale Sprachen*
23. Jahrestagung *Logik in der Informatik*

Vorwort

Der Theorietag ist die Jahrestagung der Fachgruppe *Automaten und Formale Sprachen* der Gesellschaft für Informatik und fand erstmals 1991 in Magdeburg statt. Seit dem Jahr 1996 wird der Theorietag von einem eintägigen Workshop mit eingeladenen Vorträgen begleitet. Die Jahrestagung der Fachgruppe *Logik in der Informatik* der Gesellschaft für Informatik fand erstmals 1993 in Leipzig statt. Im Laufe beider Jahrestagungen finden auch die jährliche Fachgruppensitzungen statt.

In diesem Jahr wird der Theorietag der Fachgruppe *Automaten und Formale Sprachen* erstmalig zusammen mit der Jahrestagung der Fachgruppe *Logik in der Informatik* abgehalten. Organisiert wurde die gemeinsame Veranstaltung von der Arbeitsgruppe Zuverlässige Systeme des Instituts für Informatik an der Christian-Albrechts-Universität Kiel vom 4. bis 7. Oktober im Tagungshotel Tannenfelde bei Neumünster. Während des Treffens wird ein Workshop für alle Interessierten statt finden. In Tannenfelde werden

- Christoph Löding (Aachen)
- Tomáš Masopust (Dresden)
- Henning Schnoor (Kiel)
- Nicole Schweikardt (Berlin)
- Georg Zetzsche (Paris)

eingeladene Vorträge zu ihrer aktuellen Arbeit halten.

Darüber hinaus werden 26 Vorträge von Teilnehmern und Teilnehmerinnen gehalten, 17 auf dem Theorietag *Automaten und formale Sprachen* und neun auf der Jahrestagung *Logik in der Informatik*. Der vorliegende Band enthält Kurzfassungen aller Beiträge.

Wir danken der Gesellschaft für Informatik, der Christian-Albrechts-Universität zu Kiel und dem Tagungshotel Tannenfelde für die Unterstützung dieses Theorietags. Ein besonderer Dank geht an das Organisationsteam: Maike Bradler, Philipp Sieweck, Joel Day.

Kiel, Oktober 2016 Florin Manea, Dirk Nowotka und Thomas Wilke

INHALT

WORKSHOP

26. THEORIETAG *AUTOMATEN UND FORMALE SPRACHEN*

23. JAHRESTAGUNG *LOGIK IN DER INFORMATIK*

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 1–1.

Uniformization of Rational Relations by Sequential Transducers

Christof Löding

RWTH Aachen
loeding@informatik.rwth-aachen.de

Given a binary relation that relates inputs to admissible outputs, the uniformization problem asks for function that produces for each input an admissible output. Such a function is referred to as uniformization of the relation. We consider rational relations over finite words, which are defined by nondeterministic finite state transducers, and ask whether they can be uniformized by sequential transducers, which are transducers that are deterministic on the input. In its full generality, this problem is undecidable. We exhibit some decidable cases, in particular for finite valued rational relations and the class of deterministic rational relations.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 3–3.

Partially Ordered Nondeterministic Finite Automata

Tomáš Masopust

Technische Universität Dresden
tomas.masopust@tu-dresden.de

Partially ordered nondeterminsitic finite automata are automata whose transition relation induces a partial order on states, i.e., for which cycles occur only in the form of self-loops on a single state. Partially ordered automata define a natural class of languages that has been shown to coincide with level 3/2 of the Straubing-Therien hierarchy. In this talk, we summarize the latest results concerning the expressive power of two natural restrictions of partially ordered automata. The first restriction characterizes R-trivial languages and the other J-trivial languages (also called piecewise testable languages or level 1 of the Straubing-Therien hierarchy). We also discuss some complexity results for these automata, including inclusion, equivalence and universality.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 5–5.

Using Epistemic Logic to Reason About Security Protocols

Henning Schnoor

Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany
henning.schnoor@email.uni-kiel.de

We introduce QAPI (quantified ATL with probabilism and incomplete information), which extends epistemic and probabilistic ATL with a flexible mechanism to reason about strategies in the object language.

QAPI can express complex strategic properties such as equilibria. Based on QAPI, we propose a new, widely applicable model for analyzing knowledge-based (epistemic) and strategic properties of cryptographic protocols. We prove that the corresponding model checking problem is decidable. As corollaries, we obtain decidability of complex security properties including coercion-resistance of voting protocols, accountability of protocols using a trusted third party, and abuse-freeness of contract signing protocols.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 7–7.

$+\omega$-Picture Languages Recognizable by Büchi-Tiling Systems

Nicole Schweikardt

Humboldt Universität zu Berlin
schweikn@informatik.hu-berlin.de

We consider $+\omega$-pictures, i.e., 2-dimensional pictures with a finite number of rows and a countably infinite number of columns. We extend conventional tiling systems with a Büchi acceptance condition and define the class of Büchi-tiling recognizable $+\omega$-picture languages. This class has the same closure properties as the class of tiling-recognizable languages of finite pictures. We obtain characterizations of the class of Büchi-tiling recognizable $+\omega$-picture languages by generalized Büchi-tiling systems as well as by an extension of existential monadic second-order logic with quantification of infinite sets. The Büchi characterization theorem (stating that the omega-regular languages are finite unions of languages of the form $L_1 L_2^\omega$, for regular languages L_1 and L_2) however, does not carry over from regular omega-languages to Büchi-tiling recognizable languages of $+\omega$-pictures.

[The talk is based on joint work with Parvaneh Babari, published in the Proceedings of LATA 2016.]

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 9–12.

Subword Based Abstractions of Formal Languages

Georg Zetzsche

LSV, CNRS & ENS Cachan, Université Paris-Saclay, France
zetzsche@lsv.fr

1. Upward Closures and Downward Closures

A successful idea in the area of verification is to consider finite-state abstractions of infinite-state systems. A prominent example is the fact that many language classes satisfy a *Parikh's theorem*, i.e. for each language, there exists a finite automaton that accepts the same language *up to the order of letters*. Hence, provided that the abstraction preserves pertinent properties, this allows us to work with finite-state systems, which are much easier to handle.

While Parikh-style abstractions have been studied very intensely over the last decades, recent years have seen an increasing interest in abstractions based on the subword ordering. A word $u \in \Sigma^*$ is called a *subword* of $v \in \Sigma^*$, in symbols $u \preceq v$, if $u = u_1 \cdots u_n$ and $v \in \Sigma^* u_1 \Sigma^* \cdots u_n \Sigma^*$. Given a language $L \subseteq \Sigma^*$, we can now consider its *upward closure* and its *downward closure*, defined as

$$L{\uparrow} = \{u \in \Sigma^* \mid v \preceq u \text{ for some } v \in L\} \quad \text{and} \quad L{\downarrow} = \{u \in \Sigma^* \mid u \preceq v \text{ for some } v \in L\},$$

respectively. What makes these abstractions interesting is the following theorem.

Theorem 1.1 (Higman/Haines [13, 14]) *The order $\preceq$ is a well-quasi-ordering, i.e. for every language $L \subseteq \Sigma^*$, there is a finite $F \subseteq L$ with $L{\uparrow} = F{\uparrow}$. Hence, both $L{\uparrow}$ and $L{\downarrow}$ are regular.*

This means, for every language L, there exist automata for $L{\uparrow}$ and for $L{\downarrow}$. However, in order to use these abstractions in algorithms, we have to compute them. We say that upward (downward) closures are computable for a language class $\mathcal{C}$ if there is an algorithm that, given a language L from $\mathcal{C}$, outputs a finite automaton for $L{\uparrow}$ ($L{\downarrow}$). Unfortunately, it is not obvious how to do this.

Computing upward and downward closures In the case of upward closures, the computability has been settled early on by van Leeuwen.

Theorem 1.2 (van Leeuwen [21]) *Let $\mathcal{C}$ be closed effectively under intersections with regular sets. Then, upward closures are computable for $\mathcal{C}$ if and only if emptiness is decidable for $\mathcal{C}$.*

The author is supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD).

This means, upward closures are usually computable. However, they seem to be somewhat weak abstractions: If $L \subseteq \Sigma^*$ contains the empty word, we have $L\uparrow = \Sigma^*$. Thus, the upward closure cannot distinguish languages that contain the empty word.

The downward closure appears to encode more useful information. This is reflected by the fact that under mild assumptions, computability of downward closures is a stronger condition than computability of upward closures. We call a language class a *full trio* if it is closed effectively under intersection with reguar sets, homomorphisms, and inverse homomorphisms. The *simultaneous unboundedness problem (SUP) for* $\mathcal{C}$ is the following decision problem: Given a language $L \subseteq a_1^* \cdots a_n^*$ from $\mathcal{C}$, we are asked whether for each $k \geq 0$, there are $x_1, \ldots, x_n \geq k$ with $a_1^{x_1} \cdots a_n^{x_n} \in L$.

Theorem 1.3 ([22]) *Let $\mathcal{C}$ be a full trio. Then, downward closures are computable for $\mathcal{C}$ if and only if the SUP is decidable for $\mathcal{C}$.*

In fact, there are full trios where emptiness is decidable, but downward closures are not computable, namely the languages of lossy channel systems [17]. For Church-Rosser languages, downward closures are not computable either [9].

For which language classes can we compute downward closures? It is important to distinguish two types of algorithms. We call the algorithm provided by Theorem 1.3 *brute force*, because it merely enumerates regular expressions and compares them to $L\downarrow$ using a reduction to the SUP. This type of algorithm is available for most known language classes; notably for all full trios with *effectively semilinear Parikh images* [22] and such powerful models as *matrix grammars* [22], *higher-order pushdown automata* [12] and *higher-order recursion schemes* [4].

Open problem 1.4 *Are downward closures computable for forbidding grammars [8]?*

Since the brute force algorithms yield no useful complexity upper bounds, *direct constructions* are more desirable. Those have been known longer, but for significantly fewer language classes. They are available for *context-free languages* [21, 5, 3], *0L-languages* [1], *Petri net languages* [11], and *stacked counter automata* [24].

Descriptional complexity of closures In order to use upward and downward closures in applications, we want the construction to be optimal in size. The question of descriptional complexity of closures was first raised explicitly by Gruber, Holzer, Kutrib [9, 10].

For contex-free grammars, a tight $2^{\Theta(n)}$ bound on the number of states in an NFA for upward and downward closure was proved by Bachmeier, Luttenberger, and Schlund [3]. Here, n is the sum of all lengths of right-hand sides. For one-counter automata, a polynomial upper bound is known both for upward and downward closure [2]. For n-state blind k-counter automata, an upper bound of $(3n)^{5nk+7k^3}$ for the downward closure has been obtained [23].

Open problem 1.5 *Fix k. Is there a polynomial p such that every n-state blind k-counter automaton has a downward closure NFA with at most $p(n)$ states?*

However, in order to answer questions about downward closures, it is sometimes not optimal to compute an NFA. For example, given an NFA A and a context-free grammar G, it is coNP-complete to decide whether $L(A)\downarrow = L(G)\downarrow$ [23], whereas an NFA for $L(G)\downarrow$ requires

exponential space. For more results about the complexity of inclusion and equality of downward closures, see [23].

There has also been some interest in the number of states of a DFA for $L\uparrow$ or $L\downarrow$ when L is given by a DFA with n states [10, 18, 16]. For the upward closure, Okhotin obtained the tight bound of $2^{n-2}+1$ and for the downward closure, the tight bound of 2^{n-1} was shown by Karandikar, Niewerth, and Schnoebelen [16].

2. Piecewise Testable Separators

Instead of abstracting individual languages, it is sometimes useful to abstract a pair of languages. Suppose there are disjoint languages $K, L \subseteq \Sigma^*$ and given a word $w \in K \cup L$, an action should be taken depending on whether $w \in K$ or $w \in L$. If we have a language $S \subseteq \Sigma^*$ such that $K \subseteq S$ and $S \cap L = \emptyset$, then we can use S instead of K and L to determine whether $w \in K$ or $w \in L$. In this case, S is called a *separator* of K and L.

Now our goal is, given languages K and L, to find a "simple" separator S for K and L. Unfortunately, already for context-free K, L, it is undecidable whether K and L have a regular separator [20]. However, if we turn to subword based separators, separability is decidable for many language classes. A language is called *piecewise testable* if it is a (finite) boolean combination of sets $\{w\}\uparrow$. If there exists a piecewise testable separator for K and L, then we call K and L *separable by a piecewise testable language*. Perhaps surprisingly, there is a close relationship between the computability of downward closures and the decidability of separability by piecewise testable languages.

Theorem 2.1 (Czerwiński et. al.[7]) *Let $\mathcal{C}$ be a full trio. Then, separability of languages in $\mathcal{C}$ by piecewise testable languages is decidable if and only if the SUP is decidable for $\mathcal{C}$.*

In particular, we have decidability for all the powerful models mentioned after Theorem 1.3. However, the complexity of this problem is known only for regular languages; it is P-complete [6, 19, 15], both for NFAs and for DFAs. The paper [15] studies the size of the resulting separators.

References

[1] P. A. ABDULLA, L. BOASSON, A. BOUAJJANI, Effective Lossy Queue Languages. In: *ICALP 2001*.

[2] M. F. ATIG, D. CHISTIKOV, P. HOFMAN, K. N. KUMAR, P. SAIVASAN, G. ZETZSCHE, The complexity of regular abstractions of one-counter languages. To appear in LICS 2016. Available at `http://arxiv.org/abs/1602.03419`.

[3] G. BACHMEIER, M. LUTTENBERGER, M. SCHLUND, Finite Automata for the Sub- and Superword Closure of CFLs: Descriptional and Computational Complexity. In: *LATA 2015*. 2015.

[4] L. CLEMENTE, P. PARYS, S. SALVATI, I. WALUKIEWICZ, The Diagonal Problem for Higher-Order Recursion Schemes is Decidable. In: *LICS 2016*.

[5] B. COURCELLE, On constructing obstruction sets of words. *Bulletin of the EATCS* **44** (1991), 178–186.

[6] W. CZERWIŃSKI, W. MARTENS, T. MASOPUST, Efficient Separability of Regular Languages by Subsequences and Suffixes. In: *ICALP 2013*.

[7] W. CZERWIŃSKI, W. MARTENS, L. VAN ROOIJEN, M. ZEITOUN, G. ZETZSCHE, A Characterization for Decidable Separability by Piecewise Testable Languages. Available at `http://arxiv.org/abs/1410.1042`., 2015.

[8] J. DASSOW, GH. PĂUN, *Regulated Rewriting in Formal Language Theory*. EATCS Monographs in Theoretical Computer Science 18, Springer-Verlag Berlin, 1989.

[9] H. GRUBER, M. HOLZER, M. KUTRIB, The size of Higman-Haines sets. *Theoretical Computer Science* **387** (2007) 2, 167–176.

[10] H. GRUBER, M. HOLZER, M. KUTRIB, More on the size of Higman-Haines sets: effective constructions. *Fundamenta Informaticae* **91** (2009) 1, 105–121.

[11] P. HABERMEHL, R. MEYER, H. WIMMEL, The Downward-Closure of Petri Net Languages. In: *ICALP 2010*.

[12] M. HAGUE, J. KOCHEMS, C.-H. L. ONG, Unboundedness and Downward Closures of Higher-order Pushdown Automata. In: *POPL 2016*.

[13] L. H. HAINES, On free monoids partially ordered by embedding. *Journal of Combinatorial Theory* **6** (1969) 1, 94–98.

[14] G. HIGMAN, Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society. Third Series* **2** (1952), 326–336.

[15] S. HOLUB, T. MASOPUST, M. THOMAZO, Alternating Towers and Piecewise Testable Separators. Available at `http://arxiv.org/abs/1409.3943`, 2015.

[16] P. KARANDIKAR, M. NIEWERTH, P. SCHNOEBELEN, On the state complexity of closures and interiors of regular languages with subwords and superwords. *Theoretical Computer Science* **610, Part A** (2016), 91–107.

[17] R. MAYR, Undecidable problems in unreliable computations. *Theoretical Computer Science* **297** (2003) 1-3, 337–354.

[18] A. OKHOTIN, On the state complexity of scattered substrings and superstrings. *Fundamenta Informaticae* **99** (2010) 3, 325–338.

[19] T. PLACE, L. VAN ROOIJEN, M. ZEITOUN, Separating Regular Languages by Piecewise Testable and Unambiguous Languages. In: *MFCS 2013*.

[20] T. G. SZYMANSKI, J. H. WILLIAMS, Noncanonical Extensions of Bottom-Up Parsing Techniques. *SIAM Journal on Computing* **5** (1976) 2, 231–250.

[21] J. VAN LEEUWEN, Effective constructions in well-partially-ordered free monoids. *Discrete Mathematics* **21** (1978) 3, 237–252.

[22] G. ZETZSCHE, An Approach to Computing Downward Closures. In: *ICALP 2015*.

[23] G. ZETZSCHE, The Complexity of Downward Closure Comparisons. In: *ICALP 2016*.

[24] G. ZETZSCHE, Computing Downward Closures for Stacked Counter Automata. In: *STACS 2015*.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 13–16.

Synchronizing Data Words for Register Automata

Parvaneh Babari[(A)] Karin Quaas[(A)] Mahsa Shirmohammadi[(B)]

[(A)]Universität Leipzig

[(B)]University of Oxford

Zusammenfassung

Register automata (RA) are finite automata extended with a finite set of registers to store and compare data. We study the concept of synchronizing data words in RA: Does there exist a data word that sends all states of the RA to a single state?

For deterministic RA with k registers (k-DRA), we prove that inputting data words with $2k+1$ distinct data, from the infinite data domain, is sufficient to synchronize. We show that the synchronizing problem for DRA is PSPACE-complete. For nondeterministic RAs (NRA), the synchronizing problem for NRA is in general undecidable, however, we establish Ackermann-completeness of the problem for NRA that use at most one register. Our most substantial achievement is proving NEXTPTIME-completeness of the length-bounded synchronizing problem in NRA (length encoded in binary). A variant of this last construction allows to prove that the bounded universality problem in NRA is co-NEXPTIME-complete.

1. Data Words and Register Automata

Let Σ be a finite alphabet, and let D be an infinite set that we call *data domain*. Examples for data domains are the non-negative integers or the set of strings over an alphabet. A *data word* is a finite sequence

$$(a_1, d_1)(a_2, d_2)\dots(a_n, d_n)$$

over $\Sigma \times \mathsf{D}$. A *data language* $L \subseteq (\Sigma \times \mathsf{D})^*$ is a set of data words. For reasoning about data words, we use *register automata*.

Register automata (RA) are a natural generalization of finite automata over data words, and are equipped with a finite set of registers. When processing a data word, the automaton may store the data value of the current position in one or more registers. It may also test the data value of the current position for equality with the values stored in the registers, where the result of this test determines how the RA evolves. Figure 1 shows an RA over the singleton alphabet $\Sigma = \{a\}$ with state set $\{0, 1, 1', 2, 2', 3\}$ and two registers r_1 and r_2. For $i = 1, 2$, we use $\downarrow r_i$ to denote that the current datum is stored in r_i, and we use constraint formulae $= r_i$ and $\neq r_i$, respectively, as well as Boolean combinations of such constraint formulae, to denote that the current datum is tested for equality, respectively inequality, with the value that is currently stored in r_i. In a

[(A)]This work was partially supported by Deutsche Forschungsgemeinschaft (DFG), GRK 1763 (QuantLA) and project QU 316/1-2.

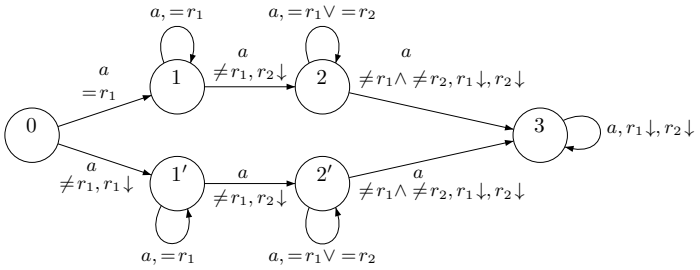


Figure 1: A deterministic register automaton with two registers r_1 and r_2

configuration of an RA, we store the current state and the current values of all registers. Note that the set of all configurations is infinite, due to the infinite data domain. Next, we show a computation of the RA in Figure 1 for the input data word $w = (a, d_1), (a, d_2)(a, d_1)(a, d_3)$, where $d_1, d_2, d_3 \in \mathsf{D}$ are pairwise distinct, starting in the configuration $(0, d_1, d_1)$:

$$(0, d_1, d_1) \xrightarrow{(a,d_1)} (1, d_1, d_1) \xrightarrow{(a,d_2)} (2, d_1, d_2) \xrightarrow{(a,d_1)} (2, d_1, d_2) \xrightarrow{(a,d_3)} (3, d_3, d_3).$$

If we are interested in the data language that an RA accepts, we fix sets of *initial* and *accepting* states and define the data language of RA to be the set of all data words for which there is a computation starting in some initial state and ending in some accepting state. If, e.g., we set state 0 to be the only initial state, and state 3 to be the only accepting state of the RA in Figure 1, then this RA accepts the set of all data words which contain at least 3 distinct data values.

The RA in Figure 1 is *deterministic*, i.e., for every configuration there is at most one successor configuration for every input letter in $\Sigma \times \mathsf{D}$.

The non-emptiness problem for RA, i.e., the question whether a given RA accepts any data word, is PSPACE-complete [2]. The non-universality problem for RA, i.e., the question whether there exists a data word that a given RA does not accept, is

- PSPACE-complete for the class of deterministic RA (DRA, for short) [2],
- undecidable for the class of non-deterministic RA (NRA, for short) [4],
- Ackermann-complete for the class of NRA with at most one register (1-NRA, for short) [2, 3].

2. Synchronizing Data Words

Let $\mathcal{A}$ be a RA with k registers. A data word w is *synchronizing for* $\mathcal{A}$ if there exists a configuration $(s, v_1, \ldots, v_k)$ such that $\mathcal{A}$ is in $(s, v_1, \ldots, v_k)$ after processing w, no matter from which (of the infinitely many) configuration $\mathcal{A}$ starts from.

As an example, consider the DRA $\mathcal{A}$ over the singleton alphabet $\{a\}$ with one register r shown on the left of Figure 2. We claim that the data word $w = (a, d_1)(a, d_2)(a, d_3)(a, d_4)$, where $d_1, d_2, d_3, d_4 \in \mathsf{D}$ are pairwise distinct, is synchronizing for $\mathcal{A}$. This can be proved by computing the *computation tree of* $\mathcal{A}$ *on* w, see right of Figure 2: Starting from the set $\{0, 1, 2, 1', 2', 3\} \times D$ of all configurations, we compute the set of configurations that can be reached by inputting w. If the computation tree ends in a set containing only a single configuration, here $(3, d_4)$, then w

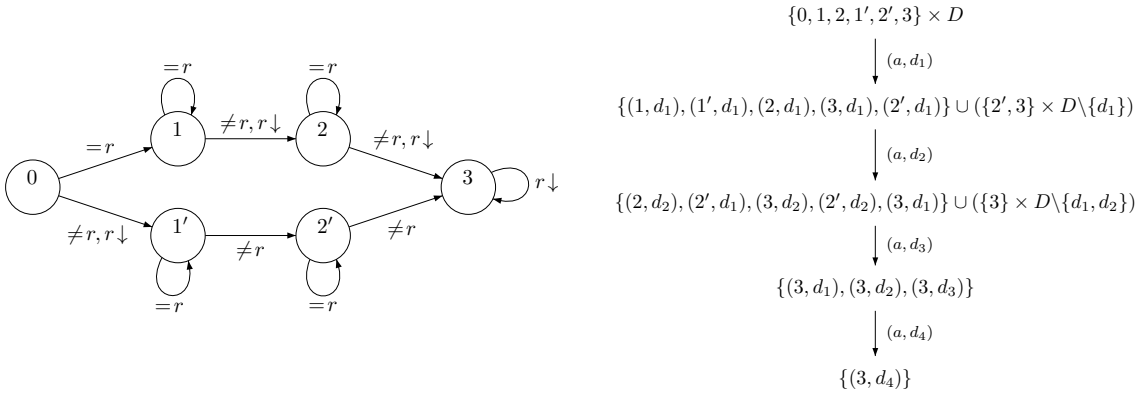


Figure 2: A DRA with a single register (left) and its computation tree on the data word $(a, d_1)(a, d_2)(a, d_3)(a, d_4)$ (right)

is indeed a synchronizing data word for $\mathcal{A}$. The *synchronizing problem* asks whether for a given RA there exists some synchronizing data word.

Synchronizing words for *finite automata* have been studied since the 70's, see [1, 5, 7]. The famous Černý conjecture on synchronizing words is a long-standing open problem in automata theory. The conjecture claims that the length of a shortest synchronizing data word for a deterministic finite automaton (DFA) with n states is at most $(n-1)^2$. In this paper, we investigate the decidability status of the synchronizing problem for RA. The main challenge is to deal with the source of infinity caused by the infinity of D.

3. Results

A natural idea to solve the synchronizing problem for RA is to compute a computation tree (like on the right of Figure 2), starting from the set of all configurations. If at some point we reach a singleton set, we have found a synchronizing data word and we return `true`. However, if even after a while we do not reach such a singleton set, when do we stop computing the computation tree and return `false`? Is there any bound N so that, if all data words with length less than N are not synchronizing for the RA, then no data word with length greater than or equal to N will be synchronizing for RA? A crucial point in answering this question is to consider the number of distinct data values that may occur in a synchronizing data word. For the DRA in Figure 2, we proved that the data word $(a, d_1)(a, d_2)(a, d_3)(a, d_4)$, containing four distinct data values, is synchronizing. But do we need four distinct data values to synchronize this DRA? The answer is no, as the data word $(a, d_1)(a, d_1)(a, d_1)(a, d_2)(a, d_1)(a, d_2)(a, d_1)$ is also synchronizing for the DRA in Figure 2. Indeed we can prove the following lemma.

Lemma 3.1 *If for a k-DRA $\mathcal{A}$ there exists some synchronizing data word, then there also exists one with at most $2k+1$ distinct data values.*

This Lemma is very helpful to prove the upper bound of the following complexity result.

Theorem 3.2 *The synchronizing problem for DRA is PSPACE-complete.*

For NRA, the situation is completely different. There is no way to bound the number of distinct data values needed for synchronizing an NRA. Indeed, we prove that

Theorem 3.3 *The synchronizing problem for NRA is undecidable.*

The proof is by a reduction from the non-universality problem for NRA. The reduction preserves the number of registers. Hence, for 1-NRA the proof does not imply undecidability, but only Ackermann-hardness.

Theorem 3.4 *For 1-NRA, the synchronizing problem is Ackermann-complete.*

For Ackermann-membership, we give a reduction to the non-universality problem for 1-NRA. Hence, for 1-NRA the synchronizing problem and the non-universality problem are interreducible.

The undecidability for the synchronizing problem for NRA shows that the length of synchronizing data words for NRA cannot be bounded. We thus consider the *length-bounded synchronizing problem*: Given an NRA $\mathcal{A}$ and some natural number N (in binary representation), does there exist a synchronizing data word for $\mathcal{A}$ with length less than N? As the main substantial achievement we prove the following theorem:

Theorem 3.5 *The length-bounded synchronizing problem for NRA is NEXPTIME-complete.*

The lower bound is proved by a reduction from the bounded universality problem for *regular-like expressions*, i.e., expressions built from letters from Σ and application of concatenation, sum, and squaring [6].

Literatur

[1] J. ČERNÝ, Poznámka k homogénnym experimentom s konečnými automatmi. *Matematicko-fyzikálny časopis* **14** (1964) 3, 208–216.

[2] S. DEMRI, R. LAZIC, LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.* **10** (2009) 3.

[3] D. FIGUEIRA, S. FIGUEIRA, S. SCHMITZ, P. SCHNOEBELEN, Ackermannian and Primitive-Recursive Bounds with Dickson's Lemma. In: *LICS 2011*. 2011, 269–278.

[4] F. NEVEN, T. SCHWENTICK, V. VIANU, Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.* **5** (2004) 3, 403–435.

[5] J. PIN, Sur les mots synthronisants dans un automate fini. *Elektronische Informationsverarbeitung und Kybernetik* **14** (1978) 6, 297–303.

[6] L. J. STOCKMEYER, A. R. MEYER, Word Problems Requiring Exponential Time: Preliminary Report. In: *STOC 1973*. 1973, 1–9.

[7] M. V. VOLKOV, Synchronizing Automata and the Cerny Conjecture. In: *LATA 2008,*. 2008, 11–27.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 17–20.

Diving into the Queue

Simon Beier Martin Kutrib Andreas Malcher
Matthias Wendlandt

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

Abstract

We introduce and study the model of diving queue automata which are basically finite automata enriched with a storage medium that is organized as a queue. Additionally, two queue heads are provided at both ends of the queue that can move in a read-only mode inside the queue. In particular, we consider suitable time constraints and the case where only a finite number of turns on the queue is allowed. As one main result we obtain a proper queue head hierarchy, that is, two heads are better than one head, and one head is better than no head. Moreover, it is shown that the model with one queue head, as well as the model with two queue heads, is P-complete. We show that a subclass of the model with two queue heads is captured by logarithmic space. Finally, we consider decidability questions and it turns out that almost nothing is decidable for the model with two queue heads whereas we obtain that at least emptiness and finiteness are decidable for subclasses of the model with one queue head.

1. Introduction

The model of a *queue* automaton where in comparison with pushdown automata the data structure of a pushdown store is replaced by a queue obeying a *first-in-first-out* principle and its investigation from a formal language point of view has not gained much attention in the literature yet. Some examples are [1, 2, 8] and some recent results on the variants of reversible queue automata and input-driven queue automata may be found in [4, 5, 6]. In general, queue automata are very powerful since they can simulate Turing machines [8] and, hence, accept every recursively enumerable language. Thus, suitable restrictions to obtain more manageable models have to be considered. One such restriction constrains the available time to (quasi) real-time [2]. However, questions such as emptiness, finiteness, or equivalence are still undecidable for real-time queue automata. Another restriction bounds the number of alternating enqueuing and dequeuing phases to a finite number and yields so-called finite-turn queue automata. It is shown in [4] that emptiness and finiteness are decidable for such automata.

In this paper, we translate the extension from pushdown automata to stack automata to queue automata with the additional constraints of quasi real-time and finite-turn boundedness. In analogy to stack automata, we provide two additional read-only heads, one at the front of the queue and one at the tail of the queue, which may move inside the queue. We call such automata

diving queue automata for which it is then possible to compare queue contents with the input several times without deleting them and, since there are two heads, to compare some part of the queue with some other part of the queue. We will also consider the restricted model where only one head is available. The paper is organized as follows. In Section 2 we define the different variants of diving queue automata and present some example languages that can be accepted by finite-turn diving queue automata with one or two queue heads. A hierarchy on the number of heads is obtained in Section 3 using the unary languages $\{ a^{n^2} \mid n \geq 0 \}$ and $\{ a^{n^3} \mid n \geq 0 \}$. It turns out that the first language shows that one head is better than no head, whereas the latter language gives that two heads are more powerful than one head. In Section 4 we investigate the computational capacity of finite-turn queue automata in more detail. We obtain that the family of languages accepted by weakly realtime D1DQA$_0$, as well as the family of languages accepted by weakly quasi-realtime DDQA, is P-complete. We also show that a subclass of the family of languages accepted by weakly quasi-realtime DDQA$_k$ is included in the complexity class L. On the other hand, every real-time deterministic pushdown automaton can be simulated by a queue automata having two heads and performing no turn. Finally, we study in Section 5 decidability questions for finite-turn queue automata. The main results are that in case of two queue heads and no turns even the question of emptiness is undecidable. On the other hand, if only one head is available it is possible to identify some subclasses for which at least the questions of emptiness and finiteness are decidable.

2. Preliminaries

We study *deterministic diving queue automata* (DDQA) that are classical queue automata with the additional ability to move the queue heads from the front and/or end inside the queue without altering the contents. Additionally, the automaton can neither append symbols to the end of the queue nor remove symbols from the front of the queue while at least one head is inside the queue. In this way, it is possible to read but not to change the stored information. To enable this behavior, the symbol at the front of the queue is marked by index $\rhd$, the symbol at the end of the queue by index $\lhd$, and a single symbol in the queue by index $\bowtie$.

A word is accepted if the DDQA halts in a configuration in which the whole input is read and an accepting state is entered.

In the sequel, we distinguish two further modes of moving the queue heads into the queue. If only the head from the front of the queue may move inside, the devices are called *deterministic front diving queue automata* (DfDQA), and if only the head from the tail of the queue may move inside, we called the devices *deterministic tail diving queue automata* (DtDQA). It is straightforward that both types of devices have the same computational capacity. So, we may use the abbreviation D1DQA for DfDQA as well as for DtDQA.

It is well known that general deterministic queue automata are computationally universal, that is, they can simulate Turing machines [8]. So, in the sequel we also consider restricted variants. More precisely, we limit the maximal number of λ-steps on each square of the input tape that can be performed *when the queue heads are not inside the queue*. A DDQA is said to be *weakly quasi realtime* if there is a constant that bounds this number for all computations. The DDQA is said to be *weakly realtime* if this constant is 0, that is, if there are no such λ-steps at all.

For a computation of a diving queue automaton, a turn is a phase in which the length of the queue first increases and then decreases. This may happen in one or at least two steps.

For any given $k \geq 0$, a *k-turn* computation is any computation containing exactly k turns.

A DDQA performing *at most* k turns in *any* computation is called k-turn DDQA and will be denoted by DDQA$_k$.

Example 2.1 *The non-semilinear language $L_3 = \{ a^{n^2} \mid n \geq 0 \}$ can be accepted by a deterministic tail diving queue automaton M_3 with no turns.*

3. Two-Party Diving is Better Than Lonesome Diving

Theorem 3.1 *For every weakly quasi-realtime diving queue automaton an equivalent weakly realtime diving queue automaton can effectively be constructed.*

It is shown in [4] that finite-turn DQA can only accept semilinear languages. Example 2.1 implicates that the family of languages accepted by finite-turn DQA is properly included in the family of languages accepted by finite-turn D1DQA.

Theorem 3.2 *Let $k \geq 0$ be a constant and M be a D1DQA$_k$ accepting a unary language. Then, an equivalent deterministic non-erasing stack automaton can be constructed.*

By results in [7] it follows that the language $L = \{ a^{n^3} \mid n \geq 0 \}$ is not accepted by any deterministic stack automaton. By Theorem 3.2 we know that L is not accepted by any finite-turn D1DQA either. In the next theorem, we are showing that L is accepted by some DDQA$_0$ which separates the family of languages accepted by finite-turn D1DQA and finite-turn DDQA.

Theorem 3.3 *The family of languages accepted by finite-turn D1DQA is a proper subset of the family of languages accepted by finite-turn DDQA.*

4. Computational Capacity

Theorem 4.1 *The family of languages accepted by weakly quasi-realtime DDQA is included in the complexity class P.*

Theorem 4.2 *The family of languages accepted by weakly realtime D1DQA$_0$ is P-hard.*

Let $\ell \geq 0$ be a constant. Then, we call a DDQA$_k$ (∞, ℓ)-DDQA$_k$ ((ℓ, ∞)-DDQA$_k$), if the number of queue head reversals in every dequeuing (enqueuing) phase is bounded by ℓ. Analogous definitions can be made for D1DQA$_k$.

Theorem 4.3 *Let $k, \ell \geq 0$ be constants. The family of languages accepted by weakly quasi-realtime (ℓ, ∞)-DDQA$_k$ is properly included in the complexity class L.*

Theorem 4.4 *For every realtime deterministic pushdown automaton an equivalent weakly realtime DDQA$_0$ can effectively be constructed.*

5. Diving and Decidability

Theorem 5.1 *Let $k \geq 0$ be a constant, M be a DDQA$_k$, and M' be a DDQA. Then the questions of emptiness, finiteness, universality, inclusion, and equivalence are not semidecidable for M. Moreover, it is not semidecidable whether or not M' is a DDQA$_k$.*

It has been shown in Theorem 3.2 that D1DQA$_k$ accepting unary languages can be simulated by non-erasing stack automata. Thus, we immediately obtain the decidability of emptiness and finiteness for such D1DQA$_k$, since it is shown in [3, 7] that both questions are decidable for one-way nondeterministic stack automata.

Theorem 5.2 *Let $k, \ell \geq 0$ be constants and M be a (∞, ℓ)-D1DQA$_k$. Then the emptiness and finiteness of $L(M)$ is decidable.*

Theorem 5.3 *Let $k, \ell \geq 0$ be constants and M be a (ℓ, ∞)-D1DQA$_k$. Then the emptiness and finiteness of $L(M)$ is decidable.*

References

[1] F.-J. BRANDENBURG, On the Intersection of Stacks and Queues. *Theoretical Computer Science* **58** (1988), 69–80.

[2] A. CHERUBINI, C. CITRINI, S. CRESPI-REGHIZZI, D. MANDRIOLI, QRT FIFO Automata, Breadth-First Grammars and Their Relations. *Theoret. Comput. Sci.* **85** (1991), 171–203.

[3] S. GINSBURG, S. A. GREIBACH, M. A. HARRISON, One-Way Stack Automata. *J. ACM* **14** (1967), 389–418.

[4] M. KUTRIB, A. MALCHER, C. MEREGHETTI, B. PALANO, M. WENDLANDT, Input-Driven Queue Automata: Finite Turns, Decidability, and Closure Properties. *Theoret. Comput. Sci.* **578** (2015), 58–71.

[5] M. KUTRIB, A. MALCHER, M. WENDLANDT, Reversible Queue Automata. In: *Non-Classical Models of Automata and Applications (NCMA 2014)*. books@ocg.at 304, Austrian Computer Society, Vienna, 2014, 163–178.

[6] M. KUTRIB, A. MALCHER, M. WENDLANDT, Input-Driven Queue Automata with Internal Transductions. In: *Language and Automata Theory and Applications (LATA 2016)*. Lecture Notes in Computer Science 9618, Springer, 2016, 156–167.

[7] W. F. OGDEN, Intercalation Theorems for Stack Languages. In: *Proceedings of the First Annual ACM Symposium on Theory of Computing (STOC 1969)*. ACM Press, New York, 1969, 31–42.

[8] R. VOLLMAR, Über einen Automaten mit Pufferspeicherung. *Computing* **5** (1970), 57–70.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 21–22.

Soliton-Automaten—Modellierung des Multi-Wellen-Falls

Henning Bordihn[(A)] Helmut Jürgensen[(B)] Heiko Ritter[(C)]

[(A)]Institut für Informatik, Universität Potsdam,
August-Bebel-Straße 89, 14482 Potsdam
henning@cs.uni-potsdam.de

[(B)]Department of Computer Science, The University of Western Ontario,
London, Ontario, Kanada, N6A 5B7
hjj@csd.uwo.ca

[(C)]Institut für Informatik, Technische Universität Dresden,
Nöthnitzer Straße 46, 01187 Dresden
heiko.ritter@mailbox.tu-dresden.de

Solitone können als Wellen oder Partikel angesehen werden, die sich ungehindert durch gewisse Substanzen bewegen können. Wenn sich eine Solitonwelle durch bestimmte Kohlenwasserstoff-Moleküle, zum Beispiel Polyacetylene, bewegt, dann ändern sich dabei deren Bindungsstrukturen. Diese Veränderungen können als Zustandsübergänge von Automaten aufgefasst werden. In den späten 1970-er Jahren schlug F. L. Carter vor [1, 2], dass Computerarchitekturen auf der Basis des Schaltverhaltens von Bindungen, verursacht durch Solitone, basieren könnten. Mehrere Experimente, die die prinzipielle Umsetzbarkeit dieser Idee nachweisen sollten, wurden von Carter und anderen in jener Zeit erfolgreich durchgeführt, [4, 5].

Im Jahr 1986 entwickelten J. Dassow und H. Jürgensen ein abstraktes mathematisches Modell für das Schaltverhalten von Polyacetylen-Molekülen. Ein Graph mit gewichteten Kanten repräsentiert das Molekül, und das Soliton ist eine Markierung, die sich entlang der Kanten von Knoten zu Knoten des Graphen bewegt, wobei sich das Gewicht der Knate ändert, die das Soliton gerade traversiert. Mit Hilfe dieses Modells wurde das logische Potenzial des Soliton-basierten Schaltungsverhaltens analysiert. Die wesentliche vereinfachende Annahme dabei war bisher, dass sich zu jedem Zeitpunkt höchstens ein Soliton im Molekül befindet [3].

Wir erweitern dieses Modell, so dass die Möglichkeit einer gleichzeitigen Präsenz von mehr als einem Soliton einbezogen ist. Dabei muss insbesondere die Interaktion von mehreren Solitonen im Molekül in einer physikalisch sinnvollen Weise modelliert werden. Verschiedene Situationen erweisen sich als spezifisch für den Fall mehrerer Solitone und können im bisherigen Modell von 1986 nicht erfasst werden. In disem Vortrag werden die Grundlagen für ein Modell von Multi-Soliton-Automaten gelegt, Modellierungsentscheidungen erläutert und neue Verhaltensweisen der Automaten diskutiert. Das Modell ist konsistent zu dem aus dem Jahr 1986, wenn nur ein Soliton präsent ist.

Literatur

[1] F. L. CARTER, Problems and prospects of future elctroactive polymers and „molecular“ electronic devices, in [5]. 1979, 121–175.

[2] F. L. CARTER (ed.), *Molecular Electronic Devices*. Marcel Dekker, New York, 1982.

[3] J. DASSOW, H. JÜRGENSEN, Soliton automata. *Journal of Computer and System Sciences* **40** (1990), 158–181.

[4] R. B. FOX, *The NRL Program on Electroactive Polymers, Second Annual Report*. Technical Report 4335, NRL Memorandum Report, Washington DC, 1980.

[5] L. B. LOCKHART, *The NRL Program on Electroactive Polymers, First Annual Report*. Technical Report 3960, NRL Memorandum Report, Washington DC, 1979.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 23–26.

On the Number of Accepting States of Finite Automata

Jürgen Dassow[(A)]

[(A)]Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik
PSF 4120, D-39016 Magdeburg, Germany
dassow@iws.cs.uni-magdeburg.de

Abstract

In this paper, we start a systematic study of the number of accepting states. For a regular language L, we define the complexity $\mathtt{asc}(L)$ as the minimal number of accepting states necessary to accept L by deterministic finite automata. With respect to nondeterministic automata, the corresponding measure is $\mathtt{nasc}(L)$. We prove that, for any nonnegative integer n, there is a regular language L such that $\mathtt{asc}(L) = n$, whereas we have $\mathtt{nasc}(R) \leq 2$ for any regular language R. Moreover, for a k-ary regularity preserving operation $\circ$ on languages, we define $g_{\circ}^{\mathtt{asc}}(n_1, n_2, \ldots, n_k)$ as the set of all integers r such that there are k regular languages L_i, $1 \leq i \leq k$, such that $\mathtt{asc}(L_i) = n_i$ for $1 \leq i \leq k$ and $\mathtt{asc}(\circ(L_1, L_2, \ldots, L_k)) = r$. We determine this set for the operations complement, union, product, Kleene closure, and set difference.

1. Introduction

State complexity is a fundamental part of automata theory. In the last 25 years, its importance draws from many applications, e. g. in natural language and speech processing, software engineering etc. where systems/automata with a large number of states are used for the description of natural languages or the behaviour of certain software systems. Thus, one is interested in questions as minimization, construction of relatively small finite automata from other devices, etc.

However, there is another measure concerning finite automata: the number of accepting states, i. e., we are not interested in all states of the automaton and count the accepting states only. This measure already occurred in some situations.

For instance (see e. g. [6]), if we consider the number of states which are necessary to accept the concatenation $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$ for two deterministic finite automata $\mathcal{A}_1$ and $\mathcal{A}_2$, then it is known that $\mathtt{sc}(L(\mathcal{A}_1) \cdot L(\mathcal{A}_2))$ depends on the number of accepting states of $\mathcal{A}_1$. An analogous situation holds if we consider the cut-operation instead of concatenation (see [2]).

The state complexity was also considered for Büchi automata which accept languages of infinite words. In the paper [1], the authors discuss the minimization of Büchi automata. The time complexity of the method they propose also depends on the number of accepting states.

Büchi automata are often used in model checking where the model is based on linear temporal logic. In the papers [3] and [4], the authors study algorithms for such model checking;

they show that the complexities of their algorithms depend on the number of accepting states of the Büchi automaton, too.

In this paper, we start a systematic investigation of the number of accepting states of finite automata. We shall see that, with respect to minimization, this measure has properties analogous to those known from "classical" state complexity, however, with respect to the behaviour under operations, the measure number of accepting states essentially differs from that of state complexity.

2. Definitions

We assume that the reader is familiar with the basic concepts of the theory of automata and formal languages (see e. g. [5]). Here we give some notation.

By card(M) we denote the cardinality of the set M. The set of all positive integers is denoted by $\mathbb{N}$.

In this paper, we consider only regular languages; therefore we omit the adjective "regular".

A nondeterministic finite automaton (shortly written as NFA) is a quintuple $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ where Σ is a finite non-empty set of input symbols, Q is a finite non-empty set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states, and $\delta : Q \times \Sigma \to 2^Q$ is the transition function of $\mathcal{A}$. The language $L(\mathcal{A})$ accepted by $\mathcal{A}$ is defined as

$$L(\mathcal{A}) = \{w \mid w \in \Sigma^*, \delta(q_0, w) \cap F \neq \emptyset\},$$

where the transition function δ is recursively extended to a mapping $\delta : Q \times \Sigma^* \to 2^Q$ as usual.

If δ is a total mapping and $\delta(q, x)$ is a singleton for any $q \in Q$ and any $x \in \Sigma$, then $\mathcal{A}$ is called a deterministic finite automaton (DFA, for short). In case of determinism, we write simply $\delta(q, x) = q'$ instead of $\delta(q, x) = \{q'\}$.

Let $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ be a DFA (or NFA, respectively). Then we define

$$\mathtt{sc}(\mathcal{A}) = \text{card}(Q) \text{ and } \mathtt{asc}(\mathcal{A}) = \text{card}(F)$$

(and

$$\mathtt{nsc}(\mathcal{A}) = \text{card}(Q) \text{ and } \mathtt{nasc}(\mathcal{A}) = \text{card}(F),$$

respectively). For a regular language L, we define

$$\begin{aligned}
\mathtt{sc}(L) &= \min\{\mathtt{sc}(\mathcal{A}) \mid \mathcal{A} \text{ is a DFA and } L(\mathcal{A}) = L\},\\
\mathtt{asc}(L) &= \min\{\mathtt{asc}(\mathcal{A}) \mid \mathcal{A} \text{ is a DFA and } L(\mathcal{A}) = L\},\\
\mathtt{nasc}(L) &= \min\{\mathtt{nasc}(\mathcal{A}) \mid \mathcal{A} \text{ is a NFA and } L(\mathcal{A}) = L\}.
\end{aligned}$$

We call the functions `sc` (`nsc`) and `asc` (`nasc`) the *state complexity* and *accepting state complexity*, respectively.

A deterministic finite automaton $\mathcal{A}$ is called *minimal* if $\mathtt{sc}(\mathcal{A}) = \mathtt{sc}(L(\mathcal{A}))$, i. e., a minimal DFA has the minimal number of states which is necessary to accept the language $L(\mathcal{A})$. There exists a well-developed theory of minimization of finite automata.

A deterministic finite automaton $\mathcal{A}$ is called *a-minimal* if $\mathtt{asc}(\mathcal{A}) = \mathtt{asc}(L(\mathcal{A}))$.

In order to characterize the behaviour of the number of accepting states under operations, we define the following set of integers.

Definition 2.1 *For a k-ary regularity preserving operation $\circ$ on languages and natural numbers $n_1, n_2, \ldots, n_k$, we define $g_\circ^{\mathtt{asc}}(n_1, n_2, \ldots, n_k)$ as the set of all integers r such that there are k languages L_i, $1 \leq i \leq k$, such that $\mathtt{asc}(L_i) = n_i$ for $1 \leq i \leq k$ and $\mathtt{asc}(\circ(L_1, L_2, \ldots, L_k)) = r$.*

3. Results

We first show that the minimization of automata with respect to the number of states is also useful for the minimization with respect to the number of accepting states.

Theorem 3.1 *i) If $\mathcal{A}$ is a minimal automaton, then we have $\mathtt{asc}(T(\mathcal{A}) = \mathtt{asc}(\mathcal{A})$, i. e., any minimal DFA is also a-minimal.*

ii) There is an algorithm which, for a given regular language L, determines $\mathtt{asc}(L)$.

We note that the converse of Theorem 3.1, i) does not hold. Obviously, the DFA $\mathcal{A} = (\{a\}, \{z_0, z_1, z_2, z_3\}, z_0, \{z_1\}, \delta)$ with $\delta(z_i, a) = z_{i+1}$ for $i \in \{0, 1, 2\}$ and $\delta(z_3, a) = z_2$ is a-minimal, since its number of accepting states is 1. On the other hand, $\mathcal{A}$ is not minimal, because the states z_2 and z_3 are equivalent.

Theorem 3.2 *i) For any alphabet X and any natural number $n \geq 0$, there is a finite language L_n over X such that $\mathtt{asc}(L_n) = n$.*

ii) For any alphabet X and any natural number $n \geq 1$, there is an infinite language L_n over X such that $\mathtt{asc}(L_n) = n$.

Obviously, $\mathtt{asc}(L) = 0$ if and only if $L = \emptyset$. Therefore, we have the restriction $n \geq 1$ in Theorem 3.2, ii).

However, the number of accepting states is not an interesting measure for NFAs, since we have the following result.

Theorem 3.3 *i) For any regular language L, $\mathtt{nasc}(L) \leq 2$.*

ii) For any non-empty regular language L with $\lambda \notin L$, $\mathtt{nasc}(L) = 1$.

We note that

- for every natural number $n \geq 1$, there is a regular language L_n such that $\mathtt{nasc}(L_n) = 1$ and $\mathtt{asc}(L_n) = n$, and
- for every natural number $n \geq 2$, there is a regular language L_n such that $\mathtt{nasc}(L_n) = 2$ and $\mathtt{asc}(L_n) = n$.

We now discuss the behaviour of the number of accepting states under the operations union ($\cup$), product ($\cdot$), set-theoretic difference ($\setminus$), complement (C), Kleene-closure (*), and positive Kleene-closure ($^+$). We have the following results.

Theorem 3.4 *The following relations hold:*

$$g_{\cup}^{\mathtt{asc}}(m,n) = \begin{cases} \{m\} & \text{for } n=0, m \geq 0 \\ \{n\} & \text{for } m=0, n \geq 0 \\ \mathbb{N} & \text{for } m \geq 1,\ n \geq 1 \end{cases}, \; g_{\cdot}^{\mathtt{asc}}(m,n) = \begin{cases} \{0\} & \text{for } n=0, m \geq 0 \\ \{0\} & \text{for } m=0, n \geq 0 \\ \mathbb{N} & \text{for } m \geq 1,\ n \geq 1 \end{cases},$$

$$g_{\setminus}^{\mathtt{asc}}(m,n) = \begin{cases} \{0\} & \text{for } m=0, n \geq 0 \\ \{m\} & \text{for } n=0, m \geq 0 \\ (N) \cup \{0\} & \text{for } m \geq 1,\ n \geq 1 \end{cases}, \; g_{C}^{\mathtt{asc}}(m) = \begin{cases} \{1\} & \text{for } m=0 \\ \{0\} \cup \mathbb{N} & \text{for } m=1 \\ \mathbb{N} & \text{for } m \geq 2 \end{cases},$$

$$g_{*}^{\mathtt{asc}}(m) = \begin{cases} \{1\} & \text{for } m=0 \\ \mathbb{N} & \text{for } m \geq 1 \end{cases}, \; g_{+}^{\mathtt{asc}}(m) = \begin{cases} \{0\} & \text{for } m=0 \\ \mathbb{N} & \text{for } m \geq 1 \end{cases}.$$

We mention that these results of Theorem 3.4 strongly differ from the corresponding results for the "usual" state complexity where in all cases only finite sets occur.

References

[1] J.-M. CHAMPARNAUD, F. COULON, *Büchi automata reduction by means of left and right trace inclusion preorder*. Manuscript, 2004.

[2] F. DREWES, M. HOLZER, S. JAKOBI, B. VAN DER MERWE, Tight bounds for cut-operations on deterministic finite automata. In: J. DURAND-LOSE, B. NAGY (eds.), *Machines, Computations, Universality*. LNCS 9288, Springer-Verlag, Berlin, 2015, 45–60.

[3] S. EDELKAMP, S. JABBAR, Large scale directed model checking LTL. In: A. VALMER (ed.), *Model Checking Software - SPIN*. LNCS 3925, Springer-Verlag, Berlin, 2006, 1–18.

[4] S. EVANGELISTA, L. KRISTENSEN, Hybrid on-the-fly LTL model checking with the sweep-line method. In: S. HADDAD, L. POMELLO (eds.), *Applications and Theory of Petri Nets*. LNCS 7347, Springer-Verlag, Berlin, 2012, 248–267.

[5] G. ROZENBERG, A. SALOMAA (eds.), *Handbook of Formal Languages. Vol. I –III*. Springer-Verlag Berlin, 1997.

[6] S. YU, State complexity of regular languages. *Journal of Automata, Languages and Combinatorics* **6** (2001), 221–234.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 27–30.

Ambiguity of Morphisms in Free Groups

Joel D. Day[A] Daniel Reidenbach[B]

[A]Department of Computer Science, Kiel University, D-24098 Kiel, Germany,
jda@informatik.uni-kiel.de

[B]Department of Computer Science, Loughborough University, Loughborough,
LE11 3TU, United Kingdom, D.Reidenbach@lboro.ac.uk

For alphabets $\mathcal{A}$ and $\mathcal{B}$, a morphism $h : \mathcal{A}^* \to \mathcal{B}^*$ is *ambiguous* with respect to a word $u \in \mathcal{A}^*$ if there exists a morphism $g : \mathcal{A}^* \to \mathcal{B}^*$ such that $g(u) = h(u)$ and $g(a) \neq h(a)$ for some $a \in \mathcal{A}$. Otherwise, h is *unambiguous* with respect to u. For example, the morphism $h : \{1,2\}^* \to \{\mathtt{a},\mathtt{b}\}^*$ given by $h(1) := \mathtt{a}$ and $h(2) := \mathtt{baa}$ is ambiguous with respect to the word $u := 1 \cdot 1 \cdot 2 \cdot 2$, since $h(u) = g(u)$ where $g : \{1,2\}^* \to \{\mathtt{a},\mathtt{b}\}^*$ is the morphism given by $g(1) := \mathtt{aab}$ and $g(2) := \mathtt{a}$:

$$h(u) = \overbrace{\underbrace{\mathtt{a}}_{h(1)}\ \underbrace{\mathtt{a}}_{h(1)}\ \underbrace{\mathtt{b}}_{h(2)}}^{g(1)}\ \overbrace{\underbrace{\mathtt{a}}_{h(1)}\ \underbrace{\mathtt{a}}_{h(2)}\ \underbrace{\mathtt{b}}_{h(2)}}^{g(1)}\ \overbrace{\underbrace{\mathtt{a}}_{h(2)}}^{g(2)}\ \overbrace{\underbrace{\mathtt{a}}_{h(2)}}^{g(2)} = g(u).$$

On the other hand, it can be shown with little effort that the morphism $h' : \{1,2,3\}^* \to \{\mathtt{a},\mathtt{b}\}^*$ given by $h'(1) := \mathtt{a}$, $h'(2) := \mathtt{b}$, and $h'(3) := \mathtt{ab}$ is unambiguous with respect to the word $u' := 1 \cdot 2 \cdot 3 \cdot 1 \cdot 3 \cdot 2$:

$$h'(u') = \overbrace{\mathtt{a}}^{h'(1)}\ \overbrace{\mathtt{b}}^{h'(2)}\ \overbrace{\mathtt{a}\ \mathtt{b}}^{h'(3)}\ \overbrace{\mathtt{a}}^{h'(1)}\ \overbrace{\mathtt{a}\ \mathtt{b}}^{h'(3)}\ \overbrace{\mathtt{b}}^{h'(2)}.$$

(Un)ambiguity is essentially a measure of (non-)determinism in the process of mapping u to $h(u)$, and determines to an extent the structural information preserved when the morphism is applied. In this respect, unambiguity is a similar notion to injectivity, although it is worth pointing out that unambiguity and injectivity are in fact incomparable: notice that the ambiguous morphism h above is injective, while the unambiguous morphism h' is not.

Due to their structure-preserving properties, unambiguous morphisms (and in particular unambiguous injective morphisms) were initially studied in the context of learning theory and inductive inference – as well as other aspects – of pattern languages. However, it is clear that the notion of unambiguity has implications in many areas concerning the combinatorial properties of morphisms such as equality sets (Salomaa [11], Engelfriet, Rozenberg [3]), the Post Correspondence Problem (Post [9]), and word equations (Makanin [7]).

While the ambiguity of morphisms has generally been studied so far in the context of free monoids and semigroups (e.g., Freydenberger et al. [5], Freydenberger, Reidenbach [4] and Reidenbach, Schneider [10]), many of these ideas overlap with areas of study related to free

[A]This work was conducted under a scholarship at the Department of Computer Science at Loughborough University.

groups – in particular combinatorial group theory, in which morphisms play a central role. Recently, the concept of pattern languages has been extended to free groups (Jain et al. [6]), and a group-equivalent of the Post Correspondence Problem – the Identity PCP – has also been studied (Bell, Potapov [1]), while equations in free groups (Makanin [8]), fixed points of automorphisms (Ventura [13]) and equalizers are all connected to the ambiguity of morphisms.

We therefore discuss the topic of ambiguity of morphisms to the free group setting (cf. Day, Reidenbach [2]), and, in particular, try to replicate some strong results which exist so far for the free monoid. Our first observation is that the most straightforward generalisation leads to a trivial situation: that all morphisms are ambiguous with respect to all words. However, while this appears to be bad news for the concept of ambiguity in a free group, it becomes apparent on closer inspection that the general notion of unambiguity – and the question of how well the structure of a morphism is preserved in the image – is still not only relevant but leads to an intriguing and rich theory.

In particular, provided a specific construction based on composition with so-called inner automorphisms is disregarded, then once again there exist unambiguous morphisms. Since inner automorphisms are particularly close to the identity morphism, two morphisms which are related by composition with an inner automorphism are also closely related, both combinatorially and algebraically. In this context, the "unambiguous" morphic images preserve most (in fact, as much as is possible) of the structure of the morphism, and such morphisms are said to be *unambiguous up to inner automorphism*.

More formally, we say that for free groups $\mathcal{F}, \mathcal{F}'$, a morphism $\sigma : \mathcal{F} \to \mathcal{F}'$ is *ambiguous up to inner automorphism* with respect to a word $\alpha \in \mathcal{F}$ if there exists a morphism $\tau : \mathcal{F} \to \mathcal{F}'$ such that $\tau(\alpha) = \sigma(\alpha)$, and $\tau \neq \sigma \circ \varphi$ for every inner automorphism $\varphi : \mathcal{F} \to \mathcal{F}$. Otherwise σ is *unambiguous up to inner automorphism*. Similarly, since automorphisms are a super-set of the inner automorphisms and are also closely related to the identity morphism, we can define a slightly weaker notion of unambiguity up to automorphism in the same way. For example, if $\mathcal{F}$ is the free group generated by $\{1,2\}$, and $\mathcal{F}'$ the free group generated by $\{\mathtt{a},\mathtt{b}\}$, then the morphism $\sigma : \mathcal{F} \to \mathcal{F}'$ given by $\sigma(1) := \mathtt{aa}$ and $\sigma(2) := \mathtt{a}^{-4}\mathtt{ba}^{4}$ is ambiguous up to inner automorphism with respect to $\alpha := 1 \cdot 1 \cdot 2 \cdot 2 \cdot 2 \cdot 2$ since, for the morphism $\tau : \mathcal{F} \to \mathcal{F}'$ given by $\tau(1) := \mathtt{bb}$ and $\tau(2) := \mathtt{a}$,

$$\sigma(\alpha) = \overbrace{\underbrace{\mathtt{a}\ \ \mathtt{a}\ \ \mathtt{a}\ \ \mathtt{a}\ \ \mathtt{a}^{-4}}_{=\varepsilon}}^{\sigma(1)\ \ \sigma(1)\ \ \sigma(2)}\ \mathtt{b}\ \overbrace{\underbrace{\mathtt{a}^{4}\ \mathtt{a}^{-4}}_{=\varepsilon}}^{\sigma(2)}\ \mathtt{b}\ \overbrace{\underbrace{\mathtt{a}^{4}\ \mathtt{a}^{-4}}_{=\varepsilon}}^{\sigma(2)}\ \mathtt{b}\ \overbrace{\underbrace{\mathtt{a}^{4}\ \mathtt{a}^{-4}}_{=\varepsilon}}^{\sigma(2)}\ \mathtt{b}\ \mathtt{a}^{4} = \tau(\alpha)$$

and it is easily verified that for any inner automorphism $\varphi : \mathcal{F} \to \mathcal{F}$, $\sigma \circ \varphi \neq \tau$. On the other hand, it can be inferred from research on test words (Turner [12]) that the identity morphism mapping each letter onto itself its unambiguous up to automorphism with respect to the same word $\alpha = 1 \cdot 1 \cdot 2 \cdot 2 \cdot 2 \cdot 2$.

We discuss the question of whether, for a given word α, there exists a morphism σ which is unambiguous up to (inner) automorphism with respect to α, and consider possible analogies to some existing characterisations of such words in a free monoid in the case that σ must also be injective. Specifically, for ambiguity up to inner automorphism, we can generalise a characterisation which exists so far in the free monoid: that an injective unambiguous morphism exists if and only if α is not a fixed point of a non-trivial morphism. Furthermore, we present

a conjecture, which if correct, implies that an equivalent characterisation holds for ambiguity up to automorphism, and that a second characterisation existing in the free monoid – given in terms of so-called morphically primitive words – also extends naturally to the free group in the case of ambiguity up to automorphism. Interestingly, this second characterisation does not hold in the case of ambiguity up to inner automorphism, suggesting that, provided our conjecture is correct, ambiguity up to automorphism is a "closer fit" to ambiguity in a free monoid than ambiguity up to inner automorphism, despite the fact that the latter is the stronger form.

We also consider the existence of non-injective unambiguous morphisms (mostly in the case of ambiguity up to automorphism) and discuss the rather counterintuitive result that there exist words for which the only unambiguous morphisms can be non-injective, and even periodic. Similarly, and also surprisingly, this leads to the observation that there exist words such as $1 \cdot 2 \cdot 3 \cdot 4 \cdot 3 \cdot 4 \cdot 1 \cdot 2$ for which all morphisms (except for the morphism erasing all letters, which is always unambiguous in a free monoid) are ambiguous in the free monoid, but for which there exist morphisms which are unambiguous (up to automorphism) in the free group.

References

[1] P. C. BELL, I. POTAPOV, On the Undecidability of the Identity Correspondence Problem and its Applications for Word and Matrix Semigroups. *International Journal of Foundations of Computer Science* **21** (2010), 963–978.

[2] J. D. DAY, D. REIDENBACH, Ambiguity of Morphisms in a Free Group. In: *Proceedings of the 10th International Conference on Words, WORDS 2015*. Lecture Notes in Computer Science 9304, 2015, 97–108.

[3] J. ENGELFRIET, G. ROZENBERG, Equality languages and Fixed Point Languages. *Information and Control* **43** (1979), 20–49.

[4] D. D. FREYDENBERGER, D. REIDENBACH, The Unambiguity of Segmented Morphisms. *Discrete Applied Mathematics* **157** (2009), 3055–3068.

[5] D. D. FREYDENBERGER, D. REIDENBACH, J. C. SCHNEIDER, Unambiguous Morphic Images of Strings. *International Journal of Foundations of Computer Science* **17** (2006), 601–628.

[6] S. JAIN, A. MIASNIKOV, F. STEPHAN, The Complexity of Verbal Languages over Groups. In: *27th Annual IEEE Symposium on Logic in Computer Science (LICS)*. 2012, 405–414.

[7] G. S. MAKANIN, The problem of solvability of equations in a free semi-group. *Soviet Mathematics Doklady* **18** (1977), 330–334.

[8] G. S. MAKANIN, Equations in a Free Group. *Mathematics of the USSR-Izvestiya* **21** (1983).

[9] E. L. POST, A variant of a recursively unsolvable problem. *Bulletin of the of the American Mathematical Society* **52** (1946), 264–268.

[10] D. REIDENBACH, J. C. SCHNEIDER, Restricted Ambiguity of Erasing Morphisms. *Theoretical Computer Science* **412** (2011), 3510–3523.

[11] A. SALOMAA, Equality sets for homomorphisms of free monoids. *Acta Cybernetica* **4** (1978), 127–239.

[12] E. C. TURNER, Test Words for Automorphisms of Free Groups. *Bulletin of the London Mathematical Society.* **28** (1996), 255–263.

[13] E. VENTURA, Fixed subgroups in free groups: a survey. *Contemporary Mathematics* **296** (1997), 231–255.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 31–32.

A Kleene Theorem for Weighted Tree Automata over Tree Valuation Monoids

Manfred Droste[(A)] Zoltán Fülöp[(B)] Doreen Götze[(A)]

[(A)]Institute of Informatics, University of Leipzig, D-04109 Leipzig, Germany,
{droste,goetze}@informatik.uni-leipzig.de

[(B)]Institute of Informatics, University of Szeged, H-6701 Szeged, Hungary,
fulop@inf.u-szeged.hu

Weighted tree automata gained a lot of attention during the last decades. The semantics of a weighted tree automaton is a mapping, called tree series, from the set of trees to an appropriate algebraic structure D. The weight of a tree is the sum of the weights of all possible runs on that tree. When D is a semiring, the weight of a run is the product of the weights of the transitions in the run. In the weighted word automaton on real numbers introduced by Chatterjee, Doyen, and Henzinger (2008), the weight of a run is determined in a global way, for instance by calculating the average of all weights of the transitions of the run. This concept was generalized to weighted automata over a more general weight structure called valuation monoid by Droste and Meinecke (2010), and to weighted tree automata over tree valuation monoids by Droste, Götze, Märcker, and Meinecke (2011).

We define rational operations and rational expressions for tree series over tree valuation monoids. Then we prove a Kleene-type result by showing that rational expressions and weighted tree automata over tree valuation monoids define the same class of tree series. In general, Kleene-type results are based on the fact that the semantics of the involved automaton models, in particular the weight of a run, is defined inductively (for instance for semirings, the weight of a run on a tree is the product of the weights of the corresponding sub-runs on the direct subtrees and the weight of the transition applied at the root of the tree). However, the weight of a run of a weighted tree automaton over a tree valuation monoid is delivered by a global valuation function, hence is not defined inductively. Therefore, we enrich the tree valuation monoid by a family of decomposition operations. By these decomposition operations we

can define the rational operations concatenation and Kleene-star on tree series appropriately. We call this enriched structure a Cauchy tree valuation monoid because our approach is based on the ideas of Droste and Meinecke (2011) where Cauchy valuation monoids were introduced and a Kleene-type result was proved for weighted word automata over this kind of monoids. In order to ensure that the concatenation and the Kleene-star of tree series defined by using the decomposition operations preserve recognizability, we follow Fülöp, Maletti, and Vogler (2009) and use variables as additional labels for leaves of trees. Moreover, we assume that the tree valuation monoid has a unit element and hereby our weight structures will be Cauchy unital tree valuation monoids. Under these conditions, we are able to prove constructively that the expressive power of our rational tree series expressions is the same as that of weighted tree automata.

Reference:

M. Droste, Z. Fülöp, D. Götze: A Kleene theorem for weighted tree automata over tree valuation monoids, in: Language and Automata Theory and Applications (LATA 2016), LNCS, vol. 9618, Springer, 2016, pp. 452-463.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 33–36.

Weighted Automata and Logics on Graphs

Stefan Dück[A]

[A]Institute of Computer Science, Leipzig University, D-04109 Leipzig, Germany
dueck@informatik.uni-leipzig.de

1. Extended Abstract

In automata theory, the fundamental Büchi-Elgot-Trakhtenbrot theorem established the coincidence of regular languages with languages definable in monadic second order logic. This led both to practical applications, e.g. in verification of finite-state programs, and to important extensions covering, for example, trees, traces, pictures, nested words, and texts; a general result for graphs was given in [7].

At the same time as Büchi, Schützenberger introduced weighted finite automata and characterized their quantitative behaviors as rational formal power series. This model also quickly developed a flourishing theory. Recently, Droste and Gastin [2] introduced a weighted MSO logic and showed its expressive equivalence to weighted automata. Soon, different authors extended this result to weighted automata and weighted logics on trees, traces, pictures, nested words, and texts.

However, a general result for weighted automata and weighted logics covering graphs and linking the previous results remained, up to now, open. Our main contributions, based on the main results of jointed work with Manfred Droste [1], are the following:

- We establish a model of weighted automata on graphs, which extends both Thomas' graph acceptors [8] and the previous weighted automata models for words, trees, pictures, and others. We show that this enables us to model new quantitative properties of graphs which could not be expressed by the previous models.

- We derive a Büchi-type equivalence result for the expressive power of weighted automata and a suitable weighted logic on graphs. We obtain corresponding equivalence results for structures like words, trees, pictures, and nested words as a consequence.

Weighted Graph Automata. In the following, we are speaking about finite, directed, vertex- and edge-labeled graphs which are degree-bounded by a natural number t, i.e., every vertex has t or less than t direct neighbors. Let A denote the vertex alphabet.

Our automata model uses a tiling-approach as follows. Here, a *tile* is a pointed graph of a fixed radius r, i.e., every vertex of the tile has a maximal distance of r to the center-point. Every finite *weighted graph automaton wGA* $\mathcal{A} = (Q, \Delta, \mathrm{wt}, \mathrm{Occ}, r)$ consists of a set of states Q, a set of tiles Δ of radius r over the vertex alphabet $A \times Q$ (which can be seen as the transitions of the

[A]supported by Deutsche Forschungsgemeinschaft (DFG) Graduiertenkolleg 1763 (QuantLA)

automaton) together with a weight-function wt assigning a semiring-weight to every tile, and an occurrence-constraint Occ counting how often certain tiles appear up to a threshold.

Then, a *run* of $\mathcal{A}$ labels every vertex with a state such that the resulting labeled graph can be completely 'tiled' by the transitions, i.e., the local neighborhood around every vertex equals a tile of Δ. An run is accepting if it fulfills the occurrence-constraint. The weight of a run is the product of all transition weights used in it. Then, the *behavior* of a wGA is a function assigning to every graph a value of our semiring by summing up the weights of all accepting runs. We say a function is *recognizable* if it is the behavior of a wGA.

We give a short motivating example showing how a wGA can compute the chromatic number of a graph, i.e., the smallest number to color a graph such that no two neighboring vertices have the same color.

Example 1.1 *Let A be an alphabet. We consider the max-min semiring $\mathbb{N}_{\max,\min} = (\mathbb{N}_+ \cup \{\infty\}, \max, \min, 1, \infty)$. We define the wGA $\mathcal{A}$ as follows. The states of $\mathcal{A}$ are q_1,...,q_{t+1} and refer to the colors. The set Δ consists of all tiles of radius 1 which have no two neighbors sharing the same state. We let the weight of a tile equal the highest index of all states appearing in the tile and let the occurrence constraint always be true. Then, the weight of every run (computed by the max-function of the semiring) is exactly the highest index of all states appearing in the run. Since every run is a valid coloring and the behavior of $\mathcal{A}$ now takes the minimum over all accepting runs, $\mathcal{A}$ computes the chromatic number of a given graph.*

Weighted Logic for Graphs. We introduce a weighted MSO logic for graphs, following a modified approach of Droste and Gastin [2] for words. It consist of the following unweighted formulas β and weighted formulas φ.

$$\beta ::= P_a(x) \mid E_b(x,y) \mid x = y \mid x \in X \mid \neg\beta \mid \beta \vee \beta \mid \exists x.\beta \mid \exists X.\beta$$
$$\varphi ::= \beta \mid k \mid \varphi \oplus \varphi \mid \varphi \otimes \varphi \mid \bigoplus_x \varphi \mid \bigoplus_X \varphi \mid \bigotimes_x \varphi$$

where $a \in A$ and $b \in B$ are vertex- and edge-labels, respectively, and k is an element of a semiring $\mathbb{K}$.

We follow classical approaches in defining the natural semantics for the unweighted fragment. The weighted formulas are evaluated using the operations of our semiring. Then, the semantics of a weighted formula φ is a function assigning a weight to every graph. We call a weighted formula *restricted* if it does not contain the quantifier $\exists X$ and after every quantification $\bigotimes_x \varphi$, there occur no more weighted quantifiers. We call a semiring *regular* if its constant series are recognizable, i.e., for every weight k, there exists a wGA $\mathcal{A}_k$ which assigns the weight k to every graph.

Finally, our first main theorem is the following.

Theorem 1.2 *Let $\mathbb{K}$ be a commutative and regular semiring, and let S be a function assigning a weight of $\mathbb{K}$ to every graph. Then the following are equivalent:*

1. *S is recognizable by a weighted graph automaton.*
2. *S is definable by a restricted sentence of our weighted logic.*

Note that nearly all classical examples of semirings (e.g., the natural numbers, the tropical semirings, and the max-min semiring) are regular.

In our proofs, in comparison to the situation for words, trees, or pictures, several difficulties arise. The crucial difference to previous approaches is a global acceptance condition in form of a check of occurrence numbers of finite tiles in runs of our automata on graphs; we need this condition to connect logic and automata. Furthermore, since we are dealing with graphs, the underlying unweighted automata model cannot be determinized. Accordingly, the unweighted logic sub-fragment covers only existential MSO. Also, the closure under weighted universal quantifications requires new methods; here we employ a theorem of Thomas [8] whose proof in turn relied on Hanf's theorem. Finally, in contrast to words, trees, pictures, or nested words, our general graphs do not have a distinguished vertex, which yields technical complications.

Words and other Special Cases. Our second main result is to show that existing quantitative automata models over words, trees, pictures, and nested words can be seen as special incidences of weighted graph automata with the same expressive power. As already stated by [8], two significant differences to the previous models are the occurrence constraint and the possibly bigger tile-size. We show that in these specials cases, we can drop the occurrence constraint and reduce the tile-size of our weighted graph automata.

Hence, we get previous equivalence results connecting weighted logic and weighted automata over these structures as a consequence of our first main theorem (in a slightly modified version).

Theorem 1.3 *Let S be a function from the set of words into a commutative semiring $\mathbb{K}$. Then, the following are equivalent:*

1. *S is recognizable by a weighted graph automaton.*
2. *S is recognizable by a weighted finite (word) automaton.*
3. *S is definable by a restricted sentence of weighted logic for words.*

If we replace "word" by "tree", "nested word", or "picture" respectively, we can formulate a similar result for weighted tree automata, weighted nested word automata, and weighted picture automata, and their respective logics.

Conclusion. We introduced a weighted generalization of Thomas' graph acceptors [7, 8] and a suitable weighted logic in the sense of Droste and Gastin [2]. We proved a Büchi-like characterization of the expressive power of weighted graph automata and showed that weighted word, tree, picture, or nested word automata are special instances of these general weighted graph automata, which gives us results of [2], [3], [5], and [6] as corollaries (under the appropriate restrictions to the underlying logic). Although not considered explicitly, we conjecture that similar equivalence results also hold for other finite structures like traces, texts, or distributed systems and their respective automata models.

For technical simplicity, here we assumed that the weights stem from a commutative semiring. Several constructions also work in a more general setting. Indeed, a similar result for *valuation monoids*, a very general weight structure including non-commutative semirings and average or discounted computations, was derived for finite and infinite graphs in [4].

References

[1] M. DROSTE, S. DÜCK, Weighted Automata and Logics on Graphs. In: G. F. ITALIANO, G. PIGHIZZINI, D. SANNELLA (eds.), *Mathematical Foundations of Computer Science, Part I*. LNCS 9234, Springer, 2015, 192–204.

[2] M. DROSTE, P. GASTIN, Weighted automata and weighted logics. *Theor. Comput. Sci.* **380** (2007) 1-2, 69–86.

[3] M. DROSTE, H. VOGLER, Weighted tree automata and weighted logics. *Theor. Comput. Sci.* **366** (2006) 3, 228–247.

[4] S. DÜCK, Weighted Automata and Logics on Infinite Graphs. In: S. BRLEK, C. REUTENAUER (eds.), *Developments in Language Theory*. LNCS 9840, Springer, 2016, 151–163.

[5] I. FICHTNER, Weighted Picture Automata and Weighted Logics. *Theory Comput. Syst.* **48** (2011) 1, 48–78.

[6] C. MATHISSEN, Weighted Logics for Nested Words and Algebraic Formal Power Series. *Logical Methods in Computer Science* **6** (2010) 1. Selected papers of ICALP 2008.

[7] W. THOMAS, On Logics, Tilings, and Automata. In: *International Colloquium on Automata, Languages, and Programming, ICALP 1991*. LNCS 510, Springer, 1991, 441–454.

[8] W. THOMAS, Elements of an Automata Theory over Partial Orders. In: *Proc. DIMACS Workshop POMIV '96*. AMS Press, Inc., New York, NY, USA, 1996, 25–40.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 37–40.

Allgemeine Grammatiken: Generierung und Akzeptierung von Sprachen

Henning Fernau[(A)]

[(A)]FB 4 – Abteilung Informatikwissenschaften, Universität Trier
D-54296 Trier, Germany
fernau@uni-trier.de (H. Fernau)

Zusammenfassung

Freund *u. a.* haben allgemeine Grammatiken eingeführt, um in einheitlicher Weise über Mechanismen argumentieren zu können, die nicht "nur" Wortsprachen beschreiben, sondern beispielsweise auch Baumsprachen oder Tupelsprachen. Wir wollen hier beschreiben, wie sich Konzepte wie Generierung und Akzeptierung hier einordnen und allgemeine Strategien zur Umformung dieser Konzepte ineinander besprechen.

1. Begriffe und Aufgaben

Definition 1.1 *Eine* allgemeine Grammatik *[5] G ist beschrieben durch $(O, O_T, \omega, P, \Longrightarrow_G)$; hierbei ist*

- *O eine Menge von* Objekten*,*
- *$O_T \subseteq O$ eine Menge (potentieller)* Terminalobjekte*,*
- *$\omega \in O$ das* Axiom*;*
- *P eine endliche Regelmenge;*
- *$\Longrightarrow_G \subseteq O \times O$ die* Ableitungsrelation *von G. Genauer induziere jede Regel $p \in P$ eine Relation $\Longrightarrow_p \subseteq O \times O$. Setze*

$$\Longrightarrow_G = \bigcup_{p \in P} \Longrightarrow_p .$$

- *$p \in P$ heißt* anwendbar *auf $x \in O$, falls es ein $y \in O$ gibt mit $(x, y) \in \Longrightarrow_p$; dann schreibt man auch $x \Longrightarrow_p y$.*
- *$\overset{*}{\Longrightarrow}_G$ bezeichnet die reflexive und transitive Hülle von $\Longrightarrow_G$.*
- *$L_{gen}(G) = \{v \in O_T \mid \omega \overset{*}{\Longrightarrow}_G v\}$ ist die von G* erzeugte Sprache.
- *$L_{acc}(G) = \{v \in O_T \mid v \overset{*}{\Longrightarrow}_G \omega\}$ ist die von G* akzeptierte Sprache.

Bedingungen an O, O_T, P und den Zusammenhang von $p \in P$ und $\Longrightarrow_p$ legen den Typ X von G fest. $\mathcal{L}_{gen}(X)$ bzw. $\mathcal{L}_{acc}(X)$ bezeichnen die so beschreibbaren Sprachfamilien. Üblicherweise werden so auch automatisch gewisse Berechenbarkeitsanforderungen erfüllt, z.B., ob man $u \Longrightarrow_p v$ in Abhängigkeit von u, v, p entscheiden kann. Der Begriff "Grammatik" ist vielleicht irreführend in obiger Definition, weil sich auch Automaten in dieser Weise mühelos beschreiben lassen.

Generierung versus Akzeptierung

Wir wollen im vorgelegten allgemeinen Rahmen folgenden Fragen nachgehen:
- Wie kann man bei festem Typ X einen Typ X' konstruieren, sodass sich zu jeder Grammatik G vom Typ X ein G' vom Typ X' angeben lässt mit $L_{gen}(G) = L_{acc}(G')$?
- Oder anders herum?

Wir werden nun zwei unterschiedliche Ansätze vorstellen, die sich hierfür (in konkreten Fällen) in der Literatur finden.

2. Der externe oder aufzählende Ansatz

Dieser Ansatz wird gerne verfolgt, wenn man akzeptierende Grammatiken (wie Turingmaschinen) übersetzt in Maschinen, die Sprachen generieren (oder aufzählen). Hierbei werden typischerweise alle möglichen Eingaben aufgezählt und geschaut, ob diese akzeptiert werden; damit sie gegebenenfalls ausgegeben (generiert) werden können, müssen sie irgendwie zwischengespeichert werden. Was bedeutet diese Idee formal?

Sei $G = (O, O_T, \omega, P, \Longrightarrow_G)$ eine Grammatik (vom Typ X) für $L_{acc}(G)$. Wir geben jetzt (implizit, durch die folgende Konstruktion) einen Typ X' und eine Grammatik G' (vom Typ X') an, mit $G' = (O', O_T, \omega', P', \Longrightarrow_{G'})$, sodass $L_{gen}(G') = L_{acc}(G)$.
Erzeugungsannahme für O_T: Es gibt endliche viele Abbildungen $\pi = \{\pi_1, \ldots, \pi_I\}$ und ein Samenobjekt $\sigma \in O_T$, sodass

$$O_T = \bigcup_{n \geq 0} \pi^n(\{\sigma\}), \quad \text{wobei} \quad \pi(Y) = \{\pi_i(y) \mid y \in Y, 1 \leq i \leq I\},$$

Wir interpretieren $P_{init} = \pi \cup \{\mathit{switch}\}$ als Teil der Regelmenge $P' = P_{init} \cup P_{simul} \cup \{\mathit{term}\}$. Setze ferner $O' = O_T \cup (O_T \times O \times \{0,1\})$ und $\omega' = (\sigma, \sigma, 0)$. Zur Deutung der Regeln:
- $(\rho, \rho, 0) \Longrightarrow_{\pi_i} (\pi_i(\rho), \pi_i(\rho), 0)$ für beliebige $\rho \in O_T$;
- $(\rho, \rho, 0) \Longrightarrow_{\mathit{switch}} (\rho, \rho, 1)$ für beliebige $\rho \in O_T$.

Sei (der Deutlichkeit halber) $P_{simul} = \{p' \mid p \in P\}$, wobei

$$((\rho, w, 1) \Longrightarrow_{p'} (\rho, v, 1)) \iff (w \Longrightarrow_p v).$$

Schließlich setzen wir: $(v, \omega, 1) \Longrightarrow_{\mathit{term}} v$ für alle $v \in O_T$. Halten wir also fest:

Satz 2.1 *Jeder Grammatikmechanismus vom Typ X, der die Erzeugungsannahme für O_T erfüllt, kann in einen Mechanismus vom Typ X' so umgeformt werden, dass $\mathcal{L}_{acc}(X) = \mathcal{L}_{gen}(X')$.*

Dieser Ansatz gestattet auch den Nachweis einer Umkehrung, wofür sich kaum Belege in der Literatur finden.

Satz 2.2 *Jeder Grammatikmechanismus vom Typ X kann in einen Mechanismus vom Typ X' derart umgeformt werden, dass $\mathcal{L}_{gen}(X) = \mathcal{L}_{acc}(X')$.*

Sei also nun $G = (O, O_T, \omega, P, \Longrightarrow_G)$ eine Grammatik (vom Typ X) für $L_{gen}(G)$. Wir geben jetzt (implizit, durch die folgende Konstruktion) einen Typ X' und eine Grammatik G' (vom Typ X') an, mit $G' = (O', O_T, \omega', P', \Longrightarrow_{G'})$, sodass $L_{acc}(G') = L_{gen}(G)$.

Setze $O' = O_T \cup (O_T \times O) \cup \{\omega'\}$, wobei ω' ein neues Objekt sei. Sei $P' = P_{simul} \cup \{\mathit{init}, \mathit{term}\}$. G' enthält als Eingabe ein Objekt $v \in O_T$; diese wird durch $v \Longrightarrow_{\mathit{init}} (v, \omega)$ verarbeitet. Setze wiederum $P_{simul} = \{p' \mid p \in P\}$. Für $p \in P$ definieren wir

$$((\rho, w) \Longrightarrow_{p'} (\rho, v)) \iff (w \Longrightarrow_p v)\,.$$

Schließlich bewirkt *term* Folgendes:

$$(\rho, \rho) \Longrightarrow_{\mathit{term}} \omega' \quad \text{für alle} \quad \rho \in O_T\,.$$

G' akzeptiert also v genau dann, wenn G das Objekt v irgendwie erzeugen kann.

3. Der interne (umdeutende) Ansatz

Dieser Ansatz findet sich bereits in einem kurzen Absatz in [6] für Chomsky-Grammatiken. Er wurde auf viele weitere (gesteuerte) Grammatikarten später erweitert, siehe [1, 2, 4]. Allgemein handelt es sich um folgende Technik.

Sei wiederum $G = (O, O_T, \omega, P, \Longrightarrow_G)$ eine Grammatik vom Typ X für $L_{gen}(G)$. Grammatik G' (mit $L_{gen}(G) = L_{acc}(G')$) ist formal identisch mit G. X' unterscheidet sich von X aber dahingehend, dass die Regeln von G' anders interpretiert werden als die von G.

$$x \Longrightarrow_p y \text{ (in } G') \iff y \Longrightarrow_p x \text{ (in } G) \quad \text{für bel. } p \in P\,.$$

Genauso lassen sich akzeptierende Grammatiken als generierende deuten.

Dies liefert offenkundig einen anderen Weg, um die im vorigen Abschnitt formulierten Sätze zu beweisen. Man beachte, dass sich die abgeleiteten Typen voneinander unterscheiden.

4. Beispiele

Registermaschinen können wir folgt als allgemeine Grammatiken begriffen werden. Sei $G = (O, O_T, \omega, P, \Longrightarrow_G)$ mit: $O = \mathbb{N}^{r+1} \cup \mathbb{N}^d$, $O_T = \mathbb{N}^d$, $\omega = (0, \ldots, 0, h)$ für eine Konstante h, die *Haltemarke*. Die Regelmenge P enthält Elemente der folgenden Art:

- $p = (i : \text{ADD}(\rho), j)$, $0 \le i < h$, $1 \le \rho \le r$, $0 \le j \le h$;
- $p = (i : \text{SUB}(\rho), j, k)$, $0 \le i < h$, $1 \le \rho \le r$, $0 \le j, k \le h$.
- *init*

Die Semantik von $\Longrightarrow_p$ ist:

- Für $p = (i : \text{ADD}(\rho), j)$ gilt:
  $(x_1, \ldots, x_r, i) \Longrightarrow_p (y_1, \ldots, y_r, j)$, falls $y_q = x_q$ für beliebige $q \in \{1, \ldots, r\} \setminus \{\rho\}$ und falls $y_\rho = x_\rho + 1$.
- Für $p = (i : \text{SUB}(\rho), j, k)$ gilt:
  $(x_1, \ldots, x_r, i) \Longrightarrow_p (y_1, \ldots, y_r, j)$, falls $y_q = x_q$ für alle $q \in \{1, \ldots, r\} \setminus \{\rho\}$, $x_\rho > 0$ und $y_\rho = x_\rho - 1$, aber $(x_1, \ldots, x_r, i) \Longrightarrow_p (x_1, \ldots, x_r, k)$, falls $x_\rho = 0$.
- $(s_1, \ldots, s_d) \Longrightarrow_{\mathit{init}} (s_1, \ldots, s_r, s_{r+1})$, falls $\forall i = d+1, \ldots, r+1 : s_i = 0$.

Der externe Ansatz liefert G' with $O' = \mathbb{N}^{d+r+2}$, Regelmenge $P' = P \cup P_{init}$ und Axiom $\omega' = (0, \ldots, 0)$.

P_{init} kann man konkret wie folgt darstellen: $\{k - \mathit{gen} \mid k \in \{1, \ldots, d\}\} \cup \{\mathit{switch}\}$, wobei:

Eingabeerzeugung $(i_1,\ldots,i_d,i_1,\ldots,i_d,0,\ldots,0) \Longrightarrow_{k-gen} (j_1,\ldots,j_d,j_1,\ldots,j_d,0,\ldots,0)$
mit $i_q = j_q$ für alle q bis auf eine Position k mit $j_k = i_k + 1$;

switch $(i_1,\ldots,i_d,i_1,\ldots,i_d,0,\ldots,0,0) \Longrightarrow_{switch} (i_1,\ldots,i_d,i_1,\ldots,i_d,0,\ldots,0,0,1)$.

Die Regeln von P erhalten als neue Semantik:

- $p = (i : \mathrm{ADD}(\rho), j)$ bedeutet nun
  $(s_1,\ldots,s_d,x_1,\ldots,x_r,i,1) \Longrightarrow_p (s_1,\ldots,s_d,y_1,\ldots,y_r,j,1)$, mit $y_q = x_q$ für beliebige $q \in \{1,\ldots,r\}$, $q \neq \rho$, und $y_\rho = x_\rho + 1$.
- $p = (i : \mathrm{SUB}(\rho), j, k)$ bedeutet nun
  $(s_1,\ldots,s_d,x_1,\ldots,x_r,i,1) \Longrightarrow_p (s_1,\ldots,s_d,y_1,\ldots,y_r,j,1)$, angenommen $x_\rho > 0$, wobei $y_q = x_q$ für alle $q \in \{1,\ldots,r\}$, $q \neq \rho$, und $y_\rho = x_\rho - 1$, aber
  $(s_1,\ldots,s_d,x_1,\ldots,x_r,i,1) \Longrightarrow_p (s_1,\ldots,s_d,x_1,\ldots,x_r,k,1)$, falls $x_\rho = 0$.

Wir ersetzen *init* durch *term*, wobei $(s_1,\ldots,s_d,0,\ldots,0,h,1) \Longrightarrow_{term} (s_1,\ldots,s_d)$.

Führt man den internen Ansatz für *Einfüge- / Löschsysteme* (kurz engl. *insdel systems*) durch, so bemerkt man, dass der Typ für generierende und akzeptierende Systeme identisch ist.

5. Schlussbemerkungen

Weitergehende Darlegungen findet man in [3]. Alle Ergebnisse lassen sich auch für Grammatiken formulieren und beweisen, die (meist endliche) Axiomenmengen anstelle einzelner Axiome zulassen. Ebenso lassen sich allgemeine Steuerungsoperatoren definieren und untersuchen.

Literatur

[1] H. BORDIHN, H. FERNAU, Accepting Grammars with Regulation **53** (1994), 1–18.

[2] H. FERNAU, Graph-controlled grammars as language acceptors **2** (1997) 2, 79–91.

[3] H. FERNAU, An Essay on General Grammars **21** (2016), 69–92.

[4] H. FERNAU, H. BORDIHN, Remarks on Accepting Parallel Systems **56** (1995), 51–67.

[5] R. FREUND, M. KOGLER, M. OSWALD, A General Framework for Regulated Rewriting Based on the Applicability of Rules. In: J. KELEMEN, A. KELEMENOVÁ (eds.), *Computation, Cooperation, and Life*. LNCS 6610, Springer, 2011, 35–53.

[6] A. SALOMAA, *Formal Languages*. Academic Press, 1973.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 41–44.

Picture Scanning Automata

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

[(A)]Fachbereich 4 – Abteilung Informatik, Universität Trier, D-54286 Trier, Germany
{fernau,paramasivan}@uni-trier.de

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

Abstract

We are systematically discussing finite automata working on rectangular-shaped arrays (i. e., pictures), reading them with 16 different scanning strategies. We show that all of 32 different variants only describe two different classes of array languages.

1. Introduction and Definitions

Syntactic considerations of digital images have a tradition of about five decades that reflect methods applied to picture processing, one of the basic methods of scanning pictures in practice have not been thoroughly investigated from a more theoretical point of view: that of using space-filling curves [4, 6]. We have investigated in [2, 3] two main different scanning methods. Let us briefly recall some standard definitions. Let $\mathbb{N} := \{1, 2, 3, \ldots\}$ be the set of natural numbers.

The notation $|w|$ stands for the length of a string w. A *two-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, $1 \leq j \leq n$, $a_{i,j} \in \Sigma$. We define the *number of columns* (or *width*) and *number of rows* (or *height*) of W by $|W|_c := n$ and $|W|_r := m$, respectively. For the sake of convenience, we also denote W by $[a_{i,j}]_{m,n}$ or by a matrix in a more pictorial form. If we want to refer to the j^{th} symbol in row i of the picture W, then we use $W[i, j] = a_{i,j}$. By Σ^{++}, we denote the set of all (non-empty) pictures over Σ. Every subset $L \subseteq \Sigma^{++}$ is a *picture language*. $\bar{L} = \Sigma^{++} - L$ is the *complement* of the picture language L. 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 $n \neq n'$.

As pictures are (also) geometrical objects, several further unary operations can be introduced [5]: *quarter-turn* (rotate clockwise by 90°) Q, *half-turn* (rotate by 180°) H, *anti-quarter-turn* (rotate anti-clockwise by 90° (or rotate clockwise by 270°)) Q^{-1}, *transpose* T (reflection along the main diagonal), *anti-transpose* T' (reflection along the anti-diagonal), R_b (reflection along a horizontal (base) line), R_v (reflection along a vertical line). Together with the *identity* I, these (now eight) operators form a non-commutative group (with respect to composition), the well-known dihedral group D_4 [1]; see Table [1a] in [3].

Here, $\circ$ is the function composition. So, if $f : X \to Y$ and $g : Y \to Z$ are two functions, then $g \circ f : X \to Z$ is defined by $(g \circ f)(x) = g(f(x))$ for all $x \in X$. How $T \circ R_v)(W)$ works is shown below for $W := [a_{i,j}]_{m,n}$ in the following.

$$(T \circ R_v)(W) = T(R_v(W)) = T\begin{pmatrix} a_{1,n} & \dots & a_{1,2} & a_{1,1} \\ a_{2,n} & \dots & a_{2,2} & a_{2,1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,n} & \dots & a_{m,2} & a_{m,1} \end{pmatrix} = \begin{matrix} a_{1,n} & a_{2,n} & \dots & a_{m,n} \\ \vdots & \ddots & \vdots & \vdots \\ a_{1,2} & a_{2,2} & \dots & a_{m,2} \\ a_{1,1} & a_{2,1} & \dots & a_{m,1} \end{matrix} = Q^{-1}(W).$$

From Table [1a] in [3] a Normal Form: $I = (Q \circ Q) \circ (Q \circ Q)$, $Q^{-1} = Q \circ (Q \circ Q)$, $H = Q \circ Q$, $R_v = Q \circ T$, $R_b = T \circ Q$ and $T' = Q \circ (Q \circ T)$ has been deduced. Let $\mathcal{O} = \{I, Q^{-1}, H, Q, R_v, R_b, T, T'\}$ be the set of these 8 unary operators comprising D_4. The operators in D_4 are usually partitioned into the four rotations (including the identity) $\{I, Q^{-1}, H, Q\}$, which form the subgroup D_2 of D_4, and four reflections $\{R_v, R_b, T, T'\}$. These operations can be also applied (picture-wise) to picture languages and (language-wise) to families of picture languages. It is interesting to add the fact from the normal form above that one single rotation Q generates all rotations (as a subgroup of D_4) and that all of D_4 are generated by one rotation Q and one reflection T.

Definition 1.1 *A* general boustrophedon finite automaton*, or GBFA for short, can be specified as an 8-tuple $M = (Q, \Sigma, R, s, F, \#, \square, D)$, where Q is a finite set of states, where Q is partitioned into Q_f and Q_b, Σ is an input alphabet, $R \subseteq Q \times (\Sigma \cup \{\#\}) \times Q$ is a finite set of rules. A rule $(q, a, p) \in R$ is usually written as $qa \to p$. We impose some additional restrictions. If $q \in Q_f$ and $a \in \Sigma$, then $qa \to p \in R$ is only possible if $p \in Q_f$. Such transitions are also called* forward transitions *and collected within R_f. Similarly, if $q \in Q_b$ and $a \in \Sigma$, $qa \to p \in R$ is only possible if $p \in Q_b$ (*backward transitions*, collected in R_b). Finally,* border transitions *(collected in $R_\#$) are of the form $q\# \to p$ with $q \in Q_f$ iff $p \in Q_b$. Namely, the special symbol $\# \notin \Sigma$ indicates the border of the rectangular picture that is processed, $s \in Q_f$ is the initial state, F is the set of final states, and $D \in \mathcal{D}$ indicates the move directions. Here,*

$$\mathcal{D} = \left\{ \begin{pmatrix} s\to & \downarrow \\ \downarrow & \leftarrow \end{pmatrix}, \begin{pmatrix} s\downarrow & \to \\ \to & \uparrow \end{pmatrix}, \begin{pmatrix} \downarrow & \leftarrow s \\ \to & \downarrow \end{pmatrix}, \begin{pmatrix} \leftarrow & \downarrow s \\ \uparrow & \leftarrow \end{pmatrix}, \begin{pmatrix} \to & \downarrow \\ s\uparrow & \to \end{pmatrix}, \begin{pmatrix} \uparrow & \leftarrow \\ s\to & \uparrow \end{pmatrix}, \begin{pmatrix} \downarrow & \leftarrow \\ \leftarrow & \uparrow s \end{pmatrix}, \begin{pmatrix} \to & \uparrow \\ \uparrow & \leftarrow s \end{pmatrix} \right\}$$

Now, we give the notions of configurations, valid configurations and an according configuration transition to formalize the work of GBFAs, based on snapshots of their work. Let $\square$ be a new symbol indicating an erased *position and let $\Sigma_{\#,\square} := \Sigma \cup \{\#, \square\}$. Then $C_M := Q \times (\Sigma^{++}_{\#,\square} \cap (\{\#\}^+ \ominus (\{\#\}_+ \oplus (\Sigma \cup \{\square\})^{++} \oplus \{\#\}_+) \ominus \{\#\}^+)) \times \{f, d\}$ is the set of configurations of M. Hence, the first and last columns and the first and last rows are completely filled with #, and these are the only positions that contain #. If the input array $A \in \Sigma^{++}$ has m rows and n columns, then* initial configuration *$(c_{init}(A))$: $\big(s, \#^{n+2} \ominus (\#_m \oplus A \oplus \#_m) \ominus \#^{n+2}, f\big)$*
final configuration: *$\big(q_f, \#^{n+2} \ominus (\#_m \oplus \square^n_m \oplus \#_m) \ominus \#^{n+2}, d\big)$ for some $q_f \in F$ and $d \in \{b, f\}$.*

The processing of the automaton is then crucially depending on $D \in \mathcal{D}$.

When the automaton encounters a border symbol, it processes the next line in the reversed way (backward processing). This is also indicated in the little pictures that describe $D \in \mathcal{D}$.

Let us also now formalize this description. Notice that an odd-numbered row of the input array corresponds to an even-numbered row if we consider the input array bordered by a #-layer.

- *If (p, A, f) and (q, A', f) are two configurations such that A and A' are identical but for one position (i, j), $1 \le i \le m+2$, $1 \le j \le n+2$, where $A'[i, j] = \square$ while $A[i, j] \in \Sigma$, then $(p, A, f) \vdash_M (q, A', f)$ if $pA[i, j] \to q \in R_f$. Moreover, i is even.*

– *Conversely, if* (p, A, f) *and* (q, A', f) *are two configurations such that* A *and* A' *are identical but for one position* (i, j)*,* $1 \leq i \leq m+2$*,* $1 \leq j \leq n+2$*, where* $A'[i, j] = \square$ *while* $A[i, j] \in \Sigma$*, then* $(p, A, b) \vdash_M (q, A', b)$ *if* $pA[i, j] \to q \in R_b$*. Moreover,* i *is odd.*
– *If* (p, A, f) *and* (q, A, b) *are two configurations, then* $(p, A, f) \vdash_M (q, A, b)$ *or* $(p, A, b) \vdash_M (q, A, f)$ *if* $p\# \to q \in R_\#$*.*

The reflexive transitive closure of the relation $\vdash_M$ *is denoted by* $\vdash_M^*$*.* $A \in \Sigma^{++}$ *is accepted by a GBFA* M *with direction* $D_{BFA} := \left(\begin{smallmatrix} s\to & \downarrow \\ \downarrow & \leftarrow \end{smallmatrix}\right)$ *if* $c_{init}(A) \vdash_M^* c$ *such that* c *is a final configuration.*

The GBFA M is deterministic, or a GBDFA for short, if for each $p \in Q$ and $a \in \Sigma \cup \{\#\}$, there is at most one $q \in Q$ with $pa \to q \in R$. This way, we define language classes like $\mathcal{L}_D(\text{GBFA})$ of those array languages accepted by GBFAs working with direction D, as well as

$$\mathcal{L}(\text{GBFA}) := \bigcup_{D \in \mathcal{D}} \mathcal{L}_D(\text{GBFA}).$$

An useful auxiliary model introduced in [2] was so-called *returning finite automata* (RFA). We can also generalize their work, similar to GBFA to get $\mathcal{L}_D(\text{GRFA})$. There are (again) eight processing modes D: $(s\to\downarrow)$, $(s\downarrow\to)$, $(\downarrow\leftarrow s)$, $(\leftarrow\downarrow s)$, $(s\to\uparrow)$, $(s\uparrow\to)$, $(\uparrow\leftarrow s)$, $(\leftarrow\uparrow s)$. These can be naturally partioned into the row-first modes $\mathcal{D}_{row-f} = \{(s\to\downarrow), (\downarrow\leftarrow s), (s\to\uparrow), (\uparrow\leftarrow s)\}$ and the four other column-first modes in $\mathcal{D}_{col-f}$. Again, we have deterministic variants, and we can consider the union of all languages pertaining to these GRFA-variants.

Example 1.2 *The set of all arrays over* $\{\mathtt{a}, \mathtt{b}\}$ *such that each array in the set has exactly one row completely filled with* `b`*'s and* `a`*'s everywhere else is accepted by a GRFA* M *as shown in Figure 1.*

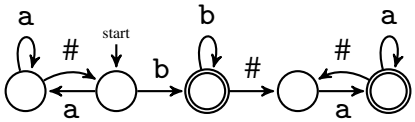


Figure 1:: GRFA M that accepts the language in Example 1.2

2. Results

Theorem 2.1 *The class* $\mathcal{L}_{D_{BFA}}(\text{GBFA})$ *coincides with the following classes:*
$T\left(\mathcal{L}_{\left(\begin{smallmatrix} s\downarrow & \to \\ \to & \uparrow \end{smallmatrix}\right)}(\text{GBFA})\right)$, $R_v\left(\mathcal{L}_{\left(\begin{smallmatrix} \downarrow & \leftarrow s \\ \to & \downarrow \end{smallmatrix}\right)}(\text{GBFA})\right)$, $Q^{-1}\left(\mathcal{L}_{\left(\begin{smallmatrix} \leftarrow & \downarrow s \\ \uparrow & \leftarrow \end{smallmatrix}\right)}(\text{GBFA})\right)$,
$Q\left(\mathcal{L}_{\left(\begin{smallmatrix} \to & \downarrow \\ s\uparrow & \to \end{smallmatrix}\right)}(\text{GBFA})\right)$, $R_b\left(\mathcal{L}_{\left(\begin{smallmatrix} \uparrow & \leftarrow \\ s\to & \uparrow \end{smallmatrix}\right)}(\text{GBFA})\right)$, $T'\left(\mathcal{L}_{\left(\begin{smallmatrix} \downarrow & \leftarrow \\ \leftarrow & \uparrow s \end{smallmatrix}\right)}(\text{GBFA})\right)$, $H\left(\mathcal{L}_{\left(\begin{smallmatrix} \to & \uparrow \\ \uparrow & \leftarrow s \end{smallmatrix}\right)}(\text{GBFA})\right)$.

Theorem 2.2 *We obtain the following list of characterizations.*
$\mathcal{L}_{\left(\begin{smallmatrix} s\downarrow & \to \\ \to & \uparrow \end{smallmatrix}\right)}(\text{GBFA}) = T\left(\mathcal{L}_{D_{BFA}}(\text{GBFA})\right)$, $\mathcal{L}_{\left(\begin{smallmatrix} \downarrow & \leftarrow s \\ \to & \downarrow \end{smallmatrix}\right)}(\text{GBFA}) = (Q \circ T)\left(\mathcal{L}_{D_{BFA}}(\text{GBFA})\right)$,
$\mathcal{L}_{\left(\begin{smallmatrix} \to & \downarrow \\ s\uparrow & \to \end{smallmatrix}\right)}(\text{GBFA}) = (Q \circ (Q \circ Q))\left(\mathcal{L}_{D_{BFA}}(\text{GBFA})\right)$, $\mathcal{L}_{\left(\begin{smallmatrix} \leftarrow & \downarrow s \\ \uparrow & \leftarrow \end{smallmatrix}\right)}(\text{GBFA}) = Q\left(\mathcal{L}_{D_{BFA}}(\text{GBFA})\right)$,

$\mathcal{L}_{\binom{\uparrow\ \leftarrow}{s\rightarrow\ \uparrow}}(\text{GBFA}) = (T \circ Q)\left(\mathcal{L}_{D_{BFA}}(\text{GBFA})\right)$, $\mathcal{L}_{\binom{\downarrow\ \leftarrow}{\leftarrow\ \uparrow s}}(\text{GBFA}) = (Q \circ (Q \circ T))$
$\left(\mathcal{L}_{D_{BFA}}(\text{GBFA})\right)$, $\mathcal{L}_{\binom{\rightarrow\ \uparrow}{\uparrow\ \leftarrow s}}(\text{GBFA}) = (Q \circ Q)\left(\mathcal{L}_{D_{BFA}}(\text{GBFA})\right)$.

Theorem 2.3 *For each direction mode D, $\mathcal{L}_D(\text{GBFA}) = \mathcal{L}_D(\text{GBDFA})$.*

Theorem 2.4 *The class $\mathcal{L}_{D_{RFA}}(\text{GRFA})$ coincides with the following classes:*
$T\left(\mathcal{L}_{(s\downarrow\ \rightarrow)}(\text{GRFA})\right), R_v\left(\mathcal{L}_{(\downarrow\ \leftarrow s)}(\text{GRFA})\right), Q^{-1}\left(\mathcal{L}_{(\leftarrow\ \downarrow s)}(\text{GRFA})\right), Q\left(\mathcal{L}_{(s\uparrow\ \rightarrow)}(\text{GRFA})\right)$,
$R_b\left(\mathcal{L}_{(s\rightarrow\ \uparrow)}(\text{GRFA})\right), T'\left(\mathcal{L}_{(\leftarrow\ \uparrow s)}(\text{GRFA})\right)$, *and* $H\left(\mathcal{L}_{(\uparrow\ \leftarrow s)}(\text{GRFA})\right)$.

Theorem 2.5 *We obtain the following list of characterizations.*
$\mathcal{L}_{(s\downarrow\ \rightarrow)}(\text{GRFA}) = T\left(\mathcal{L}_{D_{RFA}}(\text{GRFA})\right)$, $\mathcal{L}_{(\downarrow\ \leftarrow s)}(\text{GRFA}) = (Q \circ T)\left(\mathcal{L}_{D_{RFA}}(\text{GRFA})\right)$,
$\mathcal{L}_{(s\uparrow\ \rightarrow)}(\text{GRFA}) = (Q \circ (Q \circ Q))\left(\mathcal{L}_{D_{RFA}}(\text{GRFA})\right)$, $\mathcal{L}_{(\leftarrow\ \downarrow s)}(\text{GRFA}) = Q\left(\mathcal{L}_{D_{RFA}}(\text{GRFA})\right)$,
$\mathcal{L}_{(s\rightarrow\ \uparrow)}(\text{GRFA}) = (T \circ Q)\left(\mathcal{L}_{D_{RFA}}(\text{GRFA})\right)$, $\mathcal{L}_{(\leftarrow\ \uparrow s)}(\text{GRFA}) = (Q \circ (Q \circ T))$
$\left(\mathcal{L}_{D_{RFA}}(\text{GRFA})\right)$, $\mathcal{L}_{(\uparrow\ \leftarrow s)}(\text{GRFA}) = (Q \circ Q)\left(\mathcal{L}_{D_{RFA}}(\text{GRFA})\right)$.

Theorem 2.6 *The picture language family $\mathcal{L}_{D_{BFA}}(\text{GBFA})$ equals*
$\mathcal{L}_D(\text{GBFA})$ *for* $D \in \left\{D_{BFA}, \binom{\downarrow\ \leftarrow s}{\rightarrow\ \downarrow}, \binom{\uparrow\ \leftarrow}{s\rightarrow\ \uparrow}, \binom{\rightarrow\ \uparrow}{\uparrow\ \leftarrow s}\right\}$; $\mathcal{L}_D(\text{GRFA})$ *for* $D \in \mathcal{D}_{row-f}$.
The picture language family $T(\mathcal{L}_{D_{BFA}}(\text{GBFA}))$ equals
$\mathcal{L}_D(\text{GBFA})$ *for* $D \in \left\{\binom{s\downarrow\ \rightarrow}{\rightarrow\ \uparrow}, \binom{\leftarrow\ \downarrow s}{\uparrow\ \leftarrow}, \binom{\rightarrow\ \downarrow}{s\uparrow\ \rightarrow}, \binom{\downarrow\ \leftarrow}{\leftarrow\ \uparrow s}\right\}$; $\mathcal{L}_D(\text{GRFA})$ *for* $D \in \mathcal{D}_{col-f}$.

Theorem 2.7 $\forall L_1, L_2 \in \mathcal{L}_{D_{RFA}}(\text{GRFA}), L_1 \ominus L_2 \in \mathcal{L}_{D_{RFA}}(\text{GRFA})$.

Theorem 2.8 $\exists L_1, L_2 \in \mathcal{L}_{D_{RFA}}(\text{GRFA}) : L_1 \oslash L_2 \notin \mathcal{L}_{D_{RFA}}(\text{GRFA})$.

Theorem 2.9 $\mathcal{L}(\text{GRFA})$ *is closed neither under column catenation nor under row catenation.*

References

[1] M. A. ARMSTRONG, *Groups and Symmetry*. Springer-Verlag, 1988.

[2] 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.

[3] H. FERNAU, M. PARAMASIVAN, D. G. THOMAS, Picture Scanning Automata. In: R. P. BARNEVA, V. E. BRIMKOV, J. M. R. S. TAVARES (eds.), *International Symposium on Computational Modeling of Objects Presented in Images: Fundamentals, Methods, and Applications (CompIMAGE)*. LNCS, Springer, 2016, To appear.

[4] H. SAGAN, *Space-Filling Curves*. Springer, 1994.

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

[6] R. SIROMONEY, K. G. SUBRAMANIAN, Space-filling curves and infinite graphs. In: H. EHRIG, M. NAGL, G. ROZENBERG (eds.), *Graph grammars and their application to computer science*. LNCS 153, 1983, 380–391.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 45–48.

On Bonded Sequential and Parallel Insertion Systems

Wan Heng Fong[A] Markus Holzer[B] Bianca Truthe[B]
Sherzod Turaev[C] Ahmad Firdaus Yosman[A]

[A]Department of Mathematical Sciences, Faculty of Science, Universiti Teknologi Malaysia
81310 UTM Johor Bahru, Johor, Malaysia
fwh@utm.my afirdaus49@live.utm.my

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

[C]Department of Computer Science, Kulliyyah of Information & Communication Technology
Intern. Islamic University Malaysia, 53100 Jalan Gombak, Selangor D. E., Malaysia
sherzod@iium.edu.my

Abstract

We introduce a new variant of insertion systems, namely bonded insertion systems. In such systems, words are not only formed by usual letters but also by bonds between letters. Words which can be inserted, have "free" bonds at their ends which control at which positions in a word they can be inserted (namely only there, where the bonds "fit"). Two kinds of bonded insertion systems are defined in this paper: so-called bonded sequential insertion systems and bonded parallel insertion systems. In a sequential system, there is only one word inserted at a time. In a parallel system, there is a word inserted at every possible position in parallel in one time step. We investigate the generative capacity of those two kinds and relate the families of generated languages to some families of the Chomsky hierarchy and to families of languages generated by Lindenmayer systems.

1. Introduction

The operations of (*sequential*) *insertion* and *deletion*, defined and studied in [3] are generalizations of the operations of concatenation and left/right quotients. Intuitively, for given two strings u and v, instead of concatenating v at the right end of u, the insertion operation interpolates v in an arbitrary place in u. Conversely, the deletion operation, instead of removing v from the left or right ends of u, extracts v from an arbitrary position in u.

This is a short version of a paper presented at NCMA 2016 [1].

[A]The first author would like to acknowledge the Ministry of Higher Education Malaysia (MOHE) and Research Management Centre of Universiti Teknologi Malaysia for the financial funding through the Fundamental Research Grant Scheme (FRGS) Vote No. 4F590 and the last author was supported by MOHE with a MyBrainSC scholarship. The results presented in this paper were obtained during the research stay of the last author at the Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany.

[C]The fourth author would like to thank the International Islamic University Malaysia through MOHE for his financial funding of Vote No. FRGS13-066-0307.

In this paper, we show that insertion-deletion operations and systems can describe "atomic" level behaviours of bio-molecular compounds: we introduce a new variant of insertion operations on strings and languages, called bonded sequential and parallel insertions, as formal models of (bio-)chemical reactions that make and break chemical bonds. We consider strings over a bonding alphabet, i. e., an alphabet of symbols with left non-negative and right non-positive bonds. For two strings u and v over a bonding alphabet, we impose the condition that the insertion of v into u can be performed only at positions where the bonds fit. A bonded insertion system defined over an alphabet is described by a finite set of axioms and a finite set of insertion words. In a sequential system, words are generated by repeated insertions of insertion words at feasable bonds (one insertion in each step). In a parallel system, insertions take place at every possible position in one step in parallel.

A formal introduction with definitions is presented in the paper [1]. There, also first results on the generative power of those systems are proven.

2. On the Generative Power

We give here an intuitive introduction to bonding alphabets and bonded insertion systems. A bonding alphabet is a finite non-empty set of symbols $[_i\ a\ _{-j}]$ where the letter a is taken from a basic (underlying) alphabet and i and j are non-negative integers called the bonds of the symbol. A sequence

$$[_{i_0}\ a_1\ _{i_1}][_{i_2}\ a_2\ _{i_3}][_{i_4}\ a_3\ _{i_5}]\cdots[_{i_{2n-2}}\ a_n\ _{i_{2n-1}}]$$

of symbols from a bonding alphabet is said to be well-formed if all bonds fit: $i_{2j-1}+i_{2j}=0$, for $1\leq j\leq n-1$. If additionally, $i_0+i_{2n-1}=0$ holds, then the sequence is said to be a balanced word or, for short, a word. Moreover, a word is called neutral if $i_0=i_{2n-1}=0$.

A bonded insertion system is a triple

$$\gamma=(\Sigma,A,I)$$

where Σ is a finite alphabet, A is a finite set of axioms (which are neutral words over the basic alphabet Σ), and I is a finite set of insertion strings (which are balanced words over the basic alphabet Σ).

A balanced word v is derived sequentially from a balanced word u by a bonded insertion system in one step if at exactly one position before or after a letter in the word u an insertion string can be inserted such that the resulting word is v. It is derived parallely in one step if at every position (before the first letter, after the last letter, and between any two letters) either exactly one insertion string is inserted or no insertion is possible (without violating the balance).

The language generated is the set of all words which are derived in arbitrarily many derivation steps from an axiom and from which the bonds of its letters have been erased. A system where the derivation is sequential is called a bonded sequential insertion system; a system where the derivation is parallel is called a bonded parallel insertion system. The family of all languages generated by a bonded sequential or parallel insertion system is denoted by $\mathcal{L}(\mathrm{bSINS})$ and $\mathcal{L}(\mathrm{bPINS})$, respectively.

We now illustrate at an example how a bonded insertion system works in the sequential or parallel mode and which languages are generated.

Example 2.1 *Let $\gamma = (\{a\}, A, I)$ be a bonded insertion system with the two words $[_0\ a\ _0]$ and $[_0\ a\ _{-1}][_1\ a\ _0]$ as the axioms of A and $[_1\ a\ _{-2}][_2\ a\ _{-1}]$ as the insertion string of I. The words of the set A yield the words a and aa. The word $[_0\ a\ _{-1}][_1\ a\ _{-0}]$ has one position where an insertion can be applied, namely between the two letters. At the other positions, the resulting word would not be balanced. After inserting, a word of length four is obtained which has two insertion positions. Inside the insertion word, no insertion is possible. In the sequential derivation mode, the insertion string may be inserted at exactly one of these positions in order to obtain a word in the next step; in the parallel derivation mode, the insertion string is inserted simultaneously at both positions in the next step. Hence, sequentially, the system generates the language $\{\, a^n \mid n \geq 1 \,\}$. In the parallel mode, in each derivation step, from a word of length 2^n for some natural number $n \geq 1$ (which always has 2^{n-1} possible insertion positions), the word with length 2^{n+1} is obtained which has 2^n positions where an insertion can be applied. Other words are not generated. Thus, parallely, the system γ generates the language $\{\, a^{2^n} \mid n \geq 0 \,\}$.*

For bonded sequential insertion systems, we proved in [1] that every regular language can be generated by such a system and that every generated language is context-free. It was further shown that every bonded sequential insertion system can be simulated by a parallel one and that every bonded parallel insertion system can be simulated by an extended interactionless Lindenmayer system (an E0L system). We have seen that the language $\{\, a^{2^n} \mid n \geq 0 \,\}$ is generated by a bonded parallel insertion system (Example 2.1). This language is not context-free. Hence, $\mathcal{L}(\text{bPINS}) \not\subseteq \mathcal{L}(\text{CF})$. A candidate for a witness language for the relation $\mathcal{L}(\text{CF}) \not\subseteq \mathcal{L}(\text{bPINS})$ is the context-free language $\{\, a^n b^n \mid n \geq 0 \,\}$. We leave it as a task for future research to prove that the language cannot be generated by a bonded parallel insertion system.

As a summery, the hierarchy of language families relating the families $\mathcal{L}(\text{bSINS})$ and $\mathcal{L}(\text{bPINS})$ to each of the families $\mathcal{L}(\text{REG})$ of the regular languages, $\mathcal{L}(\text{CF})$ of the context-free languages, and $\mathcal{L}(\text{E0L})$ of the languages generated by E0L systems is illustrated in Figure 1. An arrow from an entry X to an entry Y represents the proper inclusion $X \subset Y$.

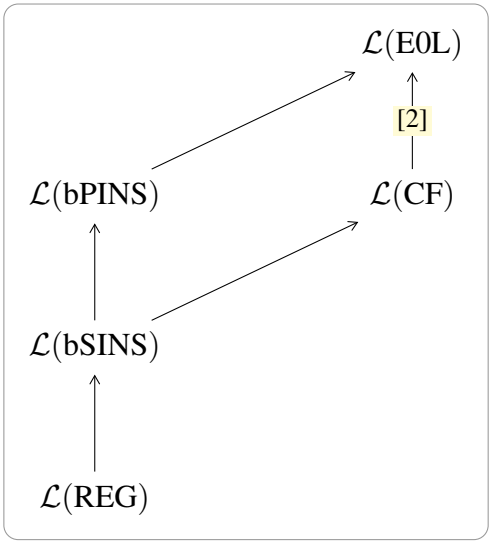


Figure 1: Hierarchy of language families.

The incomparability between $\mathcal{L}(\text{bPINS})$ and $\mathcal{L}(\text{CF})$ is still open.

With respect to some variants of Lindenmayer systems (see [2]), we have also obtained the relations depicted in Figure 2.

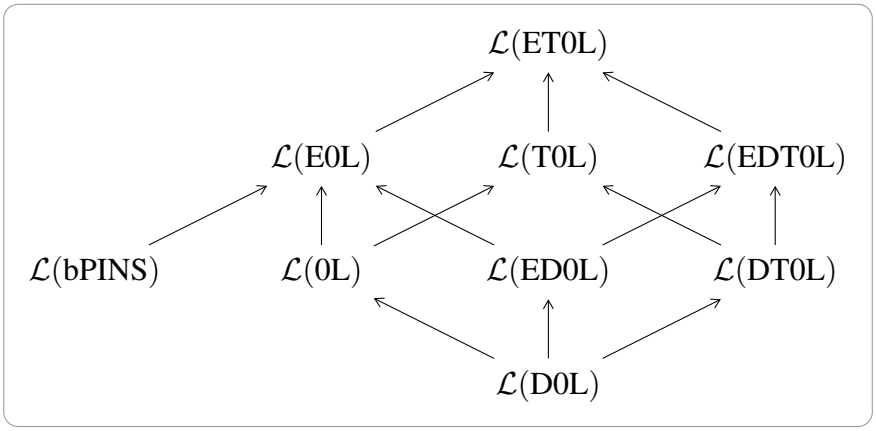


Figure 2: Relation to the hierarchy of L-systems. An arrow from an entry X to an entry Y denotes the proper inclusion $X \subset Y$. If two entries X and Y are not connected by a directed path, then the classes X and Y are incomparable. [1]

3. Future Research

In our paper, the systems may have an arbitrary number of different bonds. A task for future research is to investigate the hierarchy which is obtained when only systems with a fixed number of different bonds are considered. Additionally, one can consider bonded insertion systems which do not work purely sequentially but also not maximally parallel. In [4], we introduced so-called bonded Indian-parallel insertion systems and bonded uniformly parallel insertion systems. Research on the descriptional complexity of the family of languages generated by bonded insertion systems is another idea. Further, it remains to investigate bonded deletion systems.

References

[1] W. H. FONG, M. HOLZER, B. TRUTHE, S. TURAEV, A. F. YOSMAN, On Bonded Sequential and Parallel Insertion Systems. In: H. BORDIHN, R. FREUND, B. NAGY, G. VASZIL (eds.), *8th Workshop on Non-Classical Models of Automata and Applications (NCMA), Debrecen, Hungary, August 29–30, 2016, Proceedings*. books@ocg.at 321, Österreichische Computer Gesellschaft, Austria, 2016, 151–166.

[2] G. T. HERMANN, G. ROZENBERG, *Developmental Systems and Languages*. North-Holland/American Elsevier, 1975.

[3] L. KARI, *On insertion and deletion in formal languages*. Ph.D. thesis, University of Turku, 1991.

[4] A. F. YOSMAN, M. HOLZER, B. TRUTHE, W. H. FONG, S. TURAEV, Two Variants of Bonded Parallel Insertion Systems and Their Generative Power. In: *6th International Graduate Conference on Engineering, Science, and Humanities (IGCESH), Johor Bahru, Malaysia, August 15–17, 2016, Proceedings*. Universiti Teknologi Malaysia, Johor Bahru, Malaysia, 2016.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 49–52.

Red-Green P Automata

Rudolf Freund

TU Wien, Institute of Computer Languages
Favoritenstraße 9–11, 1040 Wien, Austria
rudi@emcc.at

Abstract

P automata have been introduced as purely communicating accepting P systems working in an automaton-like fashion for accepting strings by defining the string to be accepted by the interaction with the environment. In this paper, we focus on several variants of P automata allowing for accepting finite strings by infinite runs, thus allowing these systems to "go beyond Turing".

1. Introduction

Membrane systems have been introduced by Gheorghe Păun as a theoretical model of computing devices working in a distributed and parallel manner inspired by the functioning of the living cell, see [8]. Many variants have been considered since then. For a detailed overview the reader is referred to the Handbook of Membrane Computing [9]; actual news can be looked up in the P Webpage [10].

The first variant of *P automata* was introduced in [4], and at the same time, *analyzing P system* were defined in [6] based on a similar concept using antiport rules in an automaton-like way. Both models are computationally complete computing devices. For the latest overview article on P automata see [5]. We here focus on red-green P automata as first described in detail in [3], which allow to "go beyond Turing".

2. Definitions

A *register machine* is a tuple $M = (m, B, l_0, l_h, R)$, where m is the number of registers, B is a set of labels, $l_0 \in B$ is the initial label, $l_h \in B$ is the final label, and R is the set of instructions bijectively labeled by elements of B. The instructions of M can be an *increment* $l_1 : (ADD(r), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq r \leq m$, which increments the value of register r, followed by a non-deterministic jump to instruction l_2 or l_3, or a *conditional decrement* $l_1 : (SUB(r), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq r \leq m$, which decrements the value of register r if possible and jumps to instruction l_2, or else jumps to instruction l_3. The halting instruction $l_h : HALT$ stops the execution of the register machine.

The *Arithmetical Hierarchy* is usually developed with the universal ($\forall$) and existential ($\exists$) quantifiers restricted to the integers. Levels in the Arithmetical Hierarchy are labeled as Σ_n if they can be defined by expressions beginning with a sequence of at most n alternating quantifiers starting with $\exists$; levels are labeled as Π_n if they can be defined by such expressions of at most n alternating quantifiers that start with $\forall$; Δ_n represents $\Pi_n \cap \Sigma_n$. In particular, the languages in Δ_1 and Σ_1 are the computable languages and the recursively enumerable languages, respectively.

3. Variants of P Systems

In their basic variants, P systems (with symbol objects), usually apply multisets of rules in parallel to the objects in the underlying configuration, i.e., in the maximally parallel derivation mode (abbreviated max), a non-extendable multiset of rules is applied to the current configuration. Yet we will also consider the derivation modes, where each rule is only used in at most one copy, i.e., we consider sets of rules to be applied in parallel, for example, in the *set-maximally parallel derivation mode* (abbreviated $smax$) we apply non-extendable *sets* of rules. As in [2] we also consider the derivation modes max_{rules}, $max_{objects}$ and $smax_{rules}$, $smax_{objects}$, where multisets or sets of rules with the maximal number of rules and multisets or sets of rules affecting the maximal number of objects, respectively, are taken into account.

A (*cell-like* or *hierarchical*) *P system* is a construct

$$\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, f_{out}, f_{in})$$

where O is the alphabet of objects, $C \subset O$ is the set of catalysts, μ is the hierarchical membrane structure, with the membranes uniquely labeled by the numbers from 1 to m, $w_1, \ldots, w_m$ are multisets of objects present in the m regions of μ at the beginning of a computation, $R_1, \ldots, R_m$ are finite sets of rules, associated with the regions of μ, f_{out} is the label of the membrane region from which the outputs are taken, and f_{in} is the label of the membrane region where the inputs are put at the beginning of a computation; $f_{out} = 0/f_{in} = 0$ indicates that the output/input is taken from the environment.

If a rule $u \to v$ has at least two objects in u, then it is called *cooperative*, otherwise it is called *non-cooperative*. *Catalytic rules* are of the form $ca \to cv$, where $c \in C$ is a special object which never evolves and never passes through a membrane, it just assists object a to evolve to the multiset v. In *catalytic P systems* we use non-cooperative as well as catalytic rules. In a *purely catalytic P system* we only allow catalytic rules. The right-hand sides of the rules $u \to v$ are of the form $v \in (O \times \{here, in, out\})^*$. We may also directly indicate one of the inner membranes by the target in_k.

In *P systems with promoters and inhibitors*, with each rule, we may also assign sets of *promoters* and *inhibitors*, i.e., sets P and Q, both finite sets of multisets over O; in this case we write $(u \to v, P, Q)$. The rule $(u \to v, P, Q)$ in a membrane region r then is applicable to the contents w in region r if each multiset in P is contained in w and no multiset from Q is a submultiset of w. If at most one symbol a is used as a promoter/inhibitor in each rule, we speak of a P system with *atomic* promoters/inhibitors.

In *P systems with target selection*, all objects on the right-hand side of a rule must have the same target, and in each derivation step, for each region a (multi)set of rules – non-empty if possible – having the same target is chosen (see [7]).

The basic model of P automata as introduced in [4] and in a similar way in [6] is based on antiport rules, i.e., on rules of the form $(u, out; v, in)$, which means that the multiset u goes out through the membrane and v comes in instead. In a *P system with antiport rules* we also specify the alphabet of *environmental objects* available in an unbounded number in the environment.

In *P systems with anti-matter*, for each object a we may have its anti-matter object a^-. If an object a meets its anti-matter object a^-, then these two objects annihilate each other, which corresponds to the application of the cooperative erasing rule $aa^- \to \lambda$. We only consider the variant where these annihilation rules have weak priority over all other rules, which allows for a deterministic simulation of deterministic register machines, e.g., see [1].

For P automata, the most common way to define strings as inputs is to take the sequence of symbols taken in from the environment into the skin membrane to constitute the input string, e.g., by using antiport rules. For other variants of P automata, e.g., catalytic P automata, we need the additional target $come$ for requesting a symbol a from the environment by using a rule in the skin membrane with $(a, come)$ on its right-hand side.

Several variants of P automata even allow for the deterministic simulation of the computations of register machines, e.g., P automata with antiport rules, P automata with non-cooperative and matter/anti-matter annihilation rules, for any of the derivation modes $max, max_{rules}, max_{objects}, smax, smax_{rules}, smax_{objects}$; the same holds for (purely) catalytic P automata with *toxic* objects (if a toxic object does not evolve, the whole computation branch is abandoned). P systems with atomic inhibitors and P systems with atomic inhibitors only obtain computational completeness when working in any of the set derivation modes $smax, smax_{rules}, smax_{objects}$. For P systems with target selection, a deterministic simulation of deterministic register machines is only obtained with the derivation mode $smax_{rules}$.

4. Red-Green (P) Automata

Various possibilities how one may "go beyond Turing" are discussed in [11], for example, the definitions and results for red-green Turing machines can be found there. The basic idea of red-green automata is to distinguish between two different sets of states (red and green states) and to consider infinite runs of the automaton on finite input strings. An infinite run of a red-green Turing machine M on input w is called *recognizing* if and only if no red state is visited infinitely often. A set of strings L is said to be *accepted* by M if and only if L is recognized by M, and for every string $w \notin L$, the computation of M on input w eventually stabilizes in red (i.e., w is *rejected*). The phrase "mind change" is used in the sense of changing the color, i.e., changing from red to green or vice versa. In [3] the notion of red-green automata for register machines with input strings given on an input tape (often also called *counter automata*) is introduced and the concept of *red-green P automata* for several specific models of P automata is explained. Via red-green counter automata, the results for acceptance and recognizability of finite strings by red-green Turing machines can be carried over to red-green P automata.

Theorem 4.1 *(Computational power of red-green P automata) For any type X of a P automaton considered in this paper, allowing for the deterministic simulation of the computations of a register machine, the red-green variant of this P automaton of type X, yields the following characterizations:*

(i) *A set of strings L is recognized by a red-green P automaton of type X with one mind change if and only if L is recursively enumerable.*
(ii) *Red-green P automata of type X recognize exactly the Σ_2-sets.*
(iii) *Red-green P automata of type X accept exactly the Δ_2-sets.*

References

[1] A. ALHAZOV, B. AMAN, R. FREUND, P systems with anti-matter. In: M. GHEORGHE, G. ROZENBERG, A. SALOMAA, P. SOSÍK, C. ZANDRON (eds.), *Membrane Computing - 15th International Conference, CMC 2014, Prague, Czech Republic, August 20-22, 2014, Revised Selected Papers*. Lecture Notes in Computer Science 8961, Springer, 2014, 66–85.

[2] A. ALHAZOV, R. FREUND, S. VERLAN, Maximal variants of the set derivation mode. In: A. LEPORATI, C. ZANDRON (eds.), *Proceedings of the 17th International Conference on Membrane Computing CMC17, Milan, Italy, July 25-29, 2016*. 2016, 57–72.

[3] B. AMAN, E. CSUHAJ-VARJÚ, R. FREUND, Red-green P automata. In: M. GHEORGHE, G. ROZENBERG, A. SALOMAA, P. SOSÍK, C. ZANDRON (eds.), *Membrane Computing - 15th International Conference, CMC 2014, Prague, Czech Republic, August 20-22, 2014, Revised Selected Papers*. Lecture Notes in Computer Science 8961, Springer, 2014, 139–157.

[4] E. CSUHAJ-VARJÚ, GY. VASZIL, P automata. In: GH. PĂUN, C. ZANDRON (eds.), *Pre-Proceedings of the Workshop on Membrane Computing WMC-CdeA 2002, Curtea de Argeş, Romania, August 19-23, 2002*. Pub. No. 1 of MolCoNet-IST-2001-32008, 2002, 177–192.

[5] R. FREUND, P automata: New ideas and results. In: H. BORDIHN, R. FREUND, B. NAGY, GY. VASZIL (eds.), *Eighth Workshop on Non-Classical Models for Automata and Applications (NCMA 2016), Debrecen, Hungary, August 29 – 30, 2016, Proceedings*. books@ocg.at 321, Österreichische Computer Gesellschaft, 2016, 13–40.

[6] R. FREUND, M. OSWALD, A short note on analysing P systems. *Bulletin of the EATCS* **78** (2002), 231–236.

[7] R. FREUND, M. OSWALD, GH. PĂUN, Catalytic and purely catalytic P systems and P automata: control mechanisms for obtaining computational completeness. *Fundam. Inform.* **136** (2015) 1-2, 59–84.

[8] GH. PĂUN, Computing with membranes. *Journal of Computer and System Sciences* **61** (2000), 108–143.

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

[10] THE P SYSTEMS WEBSITE, http://ppage.psystems.eu.

[11] J. VAN LEEUWEN, J. WIEDERMANN, Computation as an unbounded process. *Theoretical Computer Science* **429** (2012), 202–212.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 53–55.

Descriptional Complexity of Spanner Representations

Dominik D. Freydenberger[(A)]

[(A)]University of Bayreuth
ddfy@ddfy.de

1. Introduction

Fagin et al. [1] introduced *core spanners* as a formal framework for information extraction that describes the properties of the annotation query language AQL, which is used in IBM's SystemT. The basic building block of core spanners are *primitive spanner representations*, which can be understood as automata or regular expressions with variables that map an input string w to a table with intervals positions of w (similar to the way search-and-replace functionality is usually implemented). Core spanner representations then combine these primitive spanners with the relational operators projection (i. e., discarding columns of the table), union (i. e., combining tables that have the same columns), natural join (on the positions in the table), and string equality selection (i. e., keeping only rows where two chosen intervals of positions describe the same substring of w).

For primitive spanners, [1] introduced *regex formulas* (regular expressions with capture groups) and *vset-* and *vst-automata* (ε-NFAs with different variable operations) and compared their expressive power, both with and without adding the algebraic operators. The cost of converting between these representations was left unexamined; and while Freydenberger and Holldack [3] showed that the trade-off from core spanner representations to any representation system that describes exactly the class of regular languages is not bounded by any regular function, the remaining comparisons of the relative succinctness was left open.

This talk summarizes [2], which discusses various issues of descriptional complexity of spanner representations.

2. Definitions

For space reasons, we only sketch the definitions (more details can be found in [2]). We mostly focus on the automata based spanner representations, where Fagin et al. [1] introduced *variable set automata (vset-automata)* and *variable stack automata (vstk-automata)*. We use an alternative definition that is based on the ref-word semantics for regex by Schmid [4]. Let Ξ be a finite set of variables, and Σ be a set of terminals. A vstk-automaton can be understood as an ε-NFA A over the alphabet $\Sigma \cup \Gamma$, $\Gamma := \{\vdash_x, \dashv_x \mid x \in \Xi\}$, where the symbols $\vdash_x$ and $\dashv_x$ stand for opening and closing the variable x, respectively. The ref-language of A, $\mathcal{R}(A)$, is then the

[(A)]supported by Deutsche Forschungsgemeinschaft (DFG) under grant FR 3551/1-1.

language of A that is obtained by interpreting these special symbols as ordinary terminals. A word $r \in \mathcal{R}(A)$ is *valid* if, for every $x \in \Xi$, each of the symbols $\vdash_x$ and $\dashv_x$ occurs exactly once in r, and $\vdash_x$ occurs to the left of $\dashv_x$. Let $\mathsf{Ref}(A) := \{r \in \mathcal{R}(A) \mid r \text{ is valid}\}$.

In order to compare ref-words (words with letters from Γ) to words from Σ^*, we define the morphism $\mathsf{clr}\colon (\Sigma \cup \Gamma)^*$ by $\mathsf{clr}(\chi) := \chi$ if $\chi \in \Sigma$, and $\mathsf{clr}(\chi) := \varepsilon$ if $\chi \in \Gamma$. For every vset-automaton A and every $w \in \Sigma^*$, let $\mathsf{Ref}(A, w) := \{r \in \mathsf{Ref}(A) \mid \mathsf{clr}(r) = w\}$. In other words, these ref-words describe how A can open and close variables in w such that every variable is opened and closed exactly once.

Every valid ref-word $r \in \mathsf{Ref}(A, w)$ has a unique factorization $r = r_1 \vdash_x r_2 \dashv_x r_3$ for every $x \in \Xi$. This can be used to define $\mu_r(x)$ (i. e., the interval that is assigned to x in a run of A on w, that is described in r), as $\mathsf{clr}(r_1)$ contains the part of w that precedes $\mu_r(x)$, and $\mathsf{clr}(r_2)$ contains $w_{\mu_r(x)}$. Formally, $\mu_r(x)$ contains the interval from $|\mathsf{clr}(r_1)| + 1$ to $|\mathsf{clr}(r_1 \cdot r_2)|$. We define $[\![A]\!]$, the *spanner* that is computed by a vset-automaton A, as the function that maps each $w \in \Sigma^*$ to the set $\{\mu_r \mid r \in \mathsf{Ref}(A, w)\}$. Intuitively, we can see each μ_r as a row in the table $[\![A]\!](w)$, which has columns that are labeled with the variables of Ξ.

Similar to this, *variable stack automata (vstk-automata)* can be understood as ε-NFAs over the alphabet $\Sigma \cup \Gamma'$, $\Gamma' := \{\vdash_x \mid x \in \Xi\} \cup \{\dashv\}$. Intuitively, vstk-automata have only a single closing symbol, which closes the variable that was opened most recently. The notion of a valid ref-word is adapted accordingly (every variable is opened exactly once, and there are exactly enough closing symbols to close all variables), which allows us to extend the definitions of $\mathsf{Ref}(A, w)$ and $[\![A]\!](w)$ to vstk-automata.

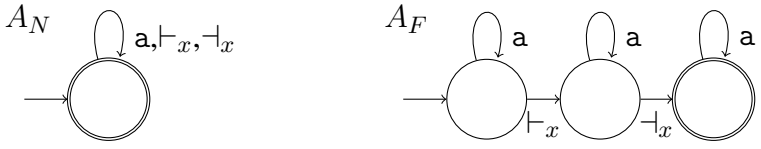
If we do not want to distinguish whether we are talking about vset- or vstk-automata, we use the more general term *v-automata*.

Finally, regex formulas are regular expressions over the alphabet $\Sigma \cup \Gamma$, which open and close variables in a balanced manner. This allows us to define $\mathcal{R}(\alpha)$, $\mathsf{Ref}(\alpha, w)$ and $[\![\alpha]\!]$ as for vset-automata.

It was shown by Fagin et al. [1] that vstk-automata have the same expressive power as regex formulas, and that vset-automata are strictly more expressive. This is for the simple reasons that vset-automata can open and close variables independently, while the stack of the vstk-automata and the balancing criteria of regex-formulas allow only spanners where the variables are hierarchical (their intervals can only overlap trivially). Furthermore, [1] shows that when adding join to the weaker models yields the same expressive power as for vstk-automata. Hence, core spanners can use any of the three models.

3. New results

One of the contributions of [2] is the notion of a *functional*ăvset- or vstk-automaton, for which $\mathsf{Ref}(A) = \mathcal{R}(A)$ holds (hence, every accepting run through a functional automaton yields a valid ref-word). Consider the following example:



The two vset-automata A_N and A_F both define the universal spanner for the single variable x over the alphabet $\{\mathtt{a}\}$ (cf. [1]). As $\mathcal{R}(A_N)$ contains ref-words like $\mathtt{a}\dashv_x\mathtt{a}\vdash_x$ or $\mathtt{a}\vdash_x\mathtt{a}\vdash_x$, A_N is not functional. In contrast to this, A_F is functional, as it uses its three states to ensure that its ref-words contain each of $\vdash_x$ and $\dashv_x$ exactly once, and in the right order.

While every vset- (or vstk-)automaton can be converted into an equivalent functional automaton, the trade-off is exponential in the number of variables. Furthermore, also vstk-automata are strictly less expressive than vset-automata, even converting a functional vstk-automaton into a functional vset-automaton can cause a blowup that is exponential in the number of variables.

But this situation changes significantly when we add the algebraic operators: A core spanner that is defined using any v-automata types can be converted in polynomial time into an equivalent core spanner that uses functional v-automata (and we can choose independently for input and output whether we want to use vset- or vstk-automata). Obviously, this also yields polynomial space bounds. (In fact, for curious technical reasons, the situation is a little bit more complicated: If we start with non-functional automata, the constructed core spanners may not be correct on the input word ϵ, unless one drops the condition that the construction has to run in polynomial time. But as a spanner $[\![A]\!](w)$ can be understood as a search in w, this is a small price to pay).

The results are obtained by proving that there are polynomial time conversions in both directions between core spanners and the logic SpLog (short for *spanner logic*), a fragment of the existential theory of concatenation.

4. Further Directions

A natural question that remains open is the trade-off from core spanners that are based on automata to core spanners that are based on regex formulas (when considering only the primitive spanners, we have an exponential trade-off, which follows immediately from the trade-off from NFAs to regular expressions). As discussed in [2], this question can be solved by examining the trade-off from NFAs to $\mathsf{SpLog}_{\mathsf{rx}}$ (the fragment of SpLog that defines constraints with regular expressions).

References

[1] R. FAGIN, B. KIMELFELD, F. REISS, S. VANSUMMEREN, Document Spanners: A Formal Approach to Information Extraction. *J. ACM* **62** (2015) 2, 12.

[2] D. D. FREYDENBERGER, A Logic for Document Spanners. In: *Proc. ICDT 2017*. 2017. Accepted. Available at `http://ddfy.de/publications/F-ALfDS.html`.

[3] D. D. FREYDENBERGER, M. HOLLDACK, Document Spanners: From Expressive Power to Decision Problems. In: *Proc. ICDT 2016*. 2016.

[4] M. L. SCHMID, Characterising REGEX Languages by Regular Languages Equipped with Factor-Referencing. *Inform. Comput.* **249** (2016), 1–17.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 57–57.

The Word Problem for Omega-Terms over the Trotter-Weil Hierarchy

Manfred Kufleitner Jan Philipp Wächter

University of Stuttgart, Formal Methods in Computer Science
{kufleitner,waechter}@fmi.uni-stuttgart.de

We show that the word problem for omega-terms over each level of the Trotter-Weil Hierarchy is decidable. More precisely, for every fixed variety in the Trotter-Weil Hierarchy, our approach yields an algorithm in nondeterministic logarithmic space (NL). In addition, we provide deterministic polynomial time algorithms, which are more efficient than straightforward translations of the NL-algorithms. As an application of our results, we show that separability by the so-called corners of the Trotter-Weil Hierarchy is witnessed by omega-terms (this property is also known as omega-reducibility). In particular, their separation problem is decidable.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 59–60.

The Trace Monoids in the Queue Monoid and in the Direct Product of Two Free Monoids

Dietrich Kuske Olena Prianychnykova[(A)]

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

Abstract

We prove that a trace monoid embeds into the queue monoid if and only if it embeds into the direct product of two free monoids. We also give a decidable characterization of these trace monoids.

Trace monoids model the behavior of concurrent systems whose concurrency is governed by the use of joint resources. They were introduced into computer science by Mazurkiewicz in his study of Petri nets [8]. Since then, much work has been invested on their structure, see [3] for comprehensive surveys. A basic fact about trace monoids is that they can be embedded into the direct product of free monoids [1]. Since the proof of this fact is constructive, an upper bound for the number of factors needed in such a free product is immediate (it is the number α of cliques needed to cover the dependence alphabet). If the dependence alphabet is a path on n vertices, than this upper bound equals the exact number, namely $n-1$. But there are cases where the exact number is considerably smaller (the examples are from [2]): if the independence alphabet is the disjoint union of two copies of C_4 (the cycle on four vertices), then $\alpha = 4$, but 3 factors suffice; if the independence alphabet is the disjoint union of n copies of K_k (the complete graph on k vertices), then $\alpha = k^n$, but k factors suffice.

The strongest result in this respect is due to Kunc [7]: Given a C_3- and C_4-free dependence alphabet and a natural number k, it is decidable whether the trace monoid embeds into the direct product of k free monoids. In this work, we extend this positive result to all dependence alphabets (also those containing C_3 or C_4), but only for the case $k = 2$. More precisely, we give a complete and decidable characterization of all independence alphabets whose generated trace monoid embeds into the direct product of two free monoids.

Queue monoids, another class of monoids, have been introduced recently [5, 6]. They model the behavior of a single fifo-queue. Intuitively, the basic actions (i.e., generators of the monoid) are the action of writing the letter a into the queue (denoted a) and reading the letter a from the queue (denoted $\bar{a}$). Sequences of actions are equivalent if they induce the same state change on any queue. For instance, writing a symbol into the queue and reading *another* symbol from the other end of the queue are two actions that can be permuted without changing the overall behavior, symbolically: $a\bar{b} \equiv \bar{b}a$. But there are also more complex equivalences that can be

[(A)] Supported by the DFG-Project "Speichermechanismen als Monoide", KU 1107/9-1.

understood as "conditional commutativity", e.g., $ab\bar{b} \equiv a\bar{b}b$. The unconditional commutations allow to embed the direct product of two free monoids into the queue monoid [6]. In [6], it is conjectured that the monoid $\mathbb{N}^3$ cannot be embedded into the queue monoid. Note that these two monoids are special trace monoids and that any trace monoid embedding into the direct product of two free monoids consequently embeds into the queue monoid. We prove the conjecture from [6] and characterize, more generally, the class of trace monoids that embed into the queue monoid.

In summary, we characterize two classes of trace monoids defined by their embedability into $\{a,b\}^* \times \{c,d\}^*$ and into the queue monoid, respectively. As it turns out, these two classes are the same, i.e., a trace monoid embeds into the direct product of two free monoids if and only if it embeds into the queue monoid, and this property is decidable. Since this class is not closed under free or direct products, it follows that the classes of submonoids of $\{a,b\}^* \times \{c,d\}^*$ and of the queue monoid Q are not closed under these operations.

An extended abstract of this work appeared as [4].

References

[1] P. CORI, D. PERRIN, Automates et commutations partielles. *R.A.I.R.O. - Informatique Théorique et Applications* **19** (1985), 21–32.

[2] V. DIEKERT, A. MUSCHOLL, K. REINHARDT, On codings of traces. In: *STACS'95*. Lecture Notes in Comp. Science vol. 900, Springer, 1995, 385–396.

[3] V. DIEKERT, G. ROZENBERG, *The Book of Traces*. World Scientific Publ. Co., 1995.

[4] 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 Comp. Science vol. 9840, Springer, 2016, 256–267.

[5] M. HUSCHENBETT, D. KUSKE, G. ZETZSCHE, The monoid of queue actions. In: *MFCS'14*. Lecture Notes in Comp. Science vol. 8634, Springer, 2014, 340–351.

[6] M. HUSCHENBETT, D. KUSKE, G. ZETZSCHE, The monoid of queue actions. *Semigroup forum* (2016). Accepted.

[7] M. KUNC, Undecidability of the trace coding problem and some decidable cases. *Theor. Comput. Sci.* **310** (2004) 1-3, 393–456.

[8] A. MAZURKIEWICZ, *Concurrent program schemes and their interpretation*. Technical report, DAIMI Report PB-78, Aarhus University, 1977.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 61–64.

Expressive Capacity of Subregular Expressions

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

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

Zusammenfassung

Three different types of subregular expressions are studied. Each type is obtained by replacing one of the regular operations by intersection. For uniformity and in order to allow non-trivial languages to be expressed, the set of literals is a finite set of words instead of letters. The power and limitations as well as relations with each other and subregular expressions where a regular operation is traded for complementation is considered mainly in terms of unary languages. Characterizations of some of the languages are obtained. A finite hierarchy is shown that reveals that the operation complementation is generally stronger than intersection.

1. Overview

The investigation of regular expressions originates in [6]. They allow a set-theoretic characterization of languages accepted by finite automata. Compared to automata, regular expressions may be better suited for human users and therefore are often used as interfaces to specify certain patterns or languages. For example, regular(-like) expressions can be found in many software tools, where the syntax used to represent them may vary, but the concepts are very much the same everywhere. The leading idea is to describe languages by using constants and operator symbols. It is well known that the operations union, concatenation, and star yield expressions that capture the regular languages. Due to the strong closure properties of regular languages several more operations could be added to the expressions without increasing their expressive capacity. On the other hand, removing some operation or replacing it by another one may have an impact on the expressive power. For example, replacing the star by complementation yields the well-known and important subregular family of star-free (or regular non-counting) languages [1] that obeys nice characterizations, for example, in terms of aperiodic syntactic monoids [13], permutation-free DFA [8], and loop-free alternating finite automata [12]. Recently, the concatenation-free languages have been studied, where the concatenation is replaced by complementation [7]. Playing around with different sets of opeartions allowed many important results on the complexity of decision problems have been obtained, for example, in [2, 5, 10, 11, 15, 14]. A recent survey on the complexity of regular(-like) expressions can be found in [3].

Here we study regular expressions where each of the classical operations is replaced by intersection. Our main interest is on the power and limitations as well as on the relations with

each other and subregular expressions where, in analogy with star-free expressions, a regular operation is traded for complementation. However, in order to allow non-trivial languages to be expressed, we allow any finite set of words as literals.

In view of the fact that the operation star plays often a crucial role for generating an infinite language out of a finite one, we get a preliminary result on the star of unary regular languages that is often used in the sequel. We explored the limits of the expressive capacity of union-free expressions, that is, regular expressions where the union is either traded for complementation or intersection (the term union free has been used differently before in [9] for ordinary regular expressions where the operation of union is removed). An immediate question is whether or not all regular languages can be described by such expressions, and if not, how they are related. We are going to answer the first question negatively for both expression classes and show that the unary languages described by the latter class are properly included in the languages described by the former class. To this end, it is shown that the unary languages are all of a certain form obtained by stretching a cofinite language and extending the lengths of the words by a finite set of numbers. Further we considered expressions where the concatenation is replaced by the intersection. The case where the concatenation is traded for complementation has recently been studied in [7]. A characterization of the unary languages in question by the union closure of certain languages shows a strict inclusion of the latter family in the former. Some further results for non-unary languages are presented as well. The remaining class of expressions have a quite simple characterization as they coincide with the finite languages. Further relations between the language classes are obtained as well. The hierarchical structure summarizing results are presented in Figure 1.

Example 1.1 *The unary language* $L = \{a\} \cup \{a^{5 \cdot n} \mid n \geq 0\}$ *is described by the union-free expression* $r = \{aa\} \cdot \overline{\overline{\{aaa\} \cdot \{aaaaa\}^*}}$.

The expression $\{aaaaa\}^*$ *describes all words whose lengths are congruent* 0 *modulo* 5*, that is,* $L_4 = \{\, a^n \mid n \equiv 0 \pmod 5 \,\}$*. The concatenation of the word* aaa *results in all words whose lengths are congruent* 3 *modulo* 5*, that is,* $L_3 = \{\, a^n \mid n \equiv 3 \pmod 5 \,\}$*. Then the complementation gives language* L_2 *which is* $\{\, a^n \mid n \not\equiv 3 \pmod 5 \,\}$*. The following concatenation with the word* aa *describes the language* L_1 *of all words whose lengths are at least* 2 *and are not congruent* 0 *modulo* 5*, that is,* $L_1 = \{\, a^n \mid n \geq 2 \text{ and } n \not\equiv 0 \pmod 5 \,\}$*. Finally, the complement of* L_1 *is language* L*.* ■

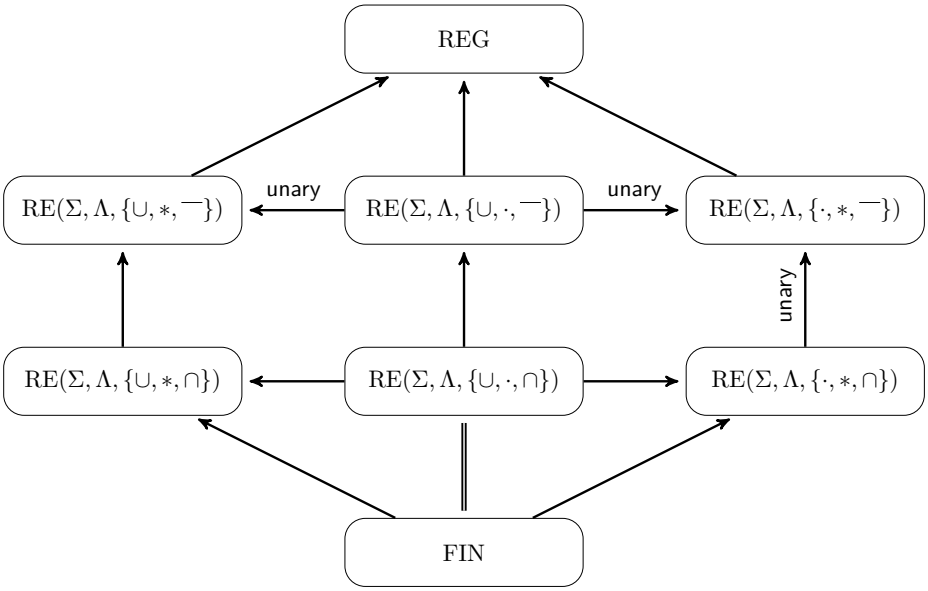


Figure 1: Inclusion structure of language families described by different types of subregular expressions. The solid arrows indicate strict inclusions. An arrow labeled unary indicates that the inclusion is for unary languages only. The relations for general languages are open in these cases, except for the shown incomparability of the families described by $\mathrm{RE}(\Sigma, \Lambda, \{\cup, \cdot, \overline{\ }\})$ and $\mathrm{RE}(\Sigma, \Lambda, \{\cdot, *, \overline{\ }\})$.

Literatur

[1] R. S. COHEN, J. A. BRZOZOWSKI, Dot-Depth of Star-Free Events. *J. Comput. System Sci.* 5 (1971), 1–16.

[2] M. FÜRER, The Complexity of the Inequivalence Problem for Regular Expressions with Intersection. In: *International Colloquium on Automata, Languages and Programming (ICALP 1980)*. LNCS 85, Springer, 1980, 234–245.

[3] M. HOLZER, M. KUTRIB, The Complexity of Regular(-like) Expressions. *Int. J. Found. Comput. Sci.* 22 (2011), 1533–1548.

[4] M. HOLZER, M. KUTRIB, K. MECKEL, Nondeterministic State Complexity of Star-Free Languages. *Theoret. Comput. Sci.* 450 (2012), 68–80.

[5] H. B. HUNT III, *The Equivalence Problem for Regular Expressions with Intersections is Not Polynomial in Tape*. Technical Report TR 73-161, Department of Computer Science, Cornell University, 1973.

[6] S. C. KLEENE, Representation of Events in Nerve Nets and Finite Automata. In: C. E. SHANNON, J. MCCARTHY (eds.), *Automata Studies*. Princeton University Press, 1956, 3–42.

[7] M. KUTRIB, M. WENDLANDT, Expressive Capacity of Concatenation Freeness. In: F. DREWES (ed.), *Implementation and Application of Automata (CIAA 2015)*. LNCS 9223, Springer, 2015, 199–210.

[8] R. MCNAUGHTON, S. PAPERT, *Counter-Free Automata*. Number 65 in Research Monographs, MIT Press, 1971.

[9] B. NAGY, Union-Free Regular Languages and 1-Cycle-Free-Path Automata. *Publicationes Mathematicae Debrecen* 68 (2006), 183–197.

[10] H. PETERSEN, Decision Problems for Generalized Regular Expressions. In: *Descriptional Complexity of Automata, Grammars and Related Structures (DCAGRS 2000)*. London, Ontario, 2000, 22–29.

[11] H. PETERSEN, The Membership Problem for Regular Expressions with Intersection is Complete in LOGCFL. In: *Theoretical Aspects of Computer Science (STACS 2002)*. LNCS 2285, Springer, 2002, 513–522.

[12] K. SALOMAA, S. YU, Alternating Finite Automata and Star-Free Languages. *Theoret. Comput. Sci.* 234 (2000), 167–176.

[13] M.-P. SCHÜTZENBERGER, On Finite Monoids Having Only Trivial Subgroups. *Inform. Control* 8 (1965), 190–194.

[14] L. J. STOCKMEYER, *The Complexity of Decision Problems in Automata Theory and Logic*. Ph.D. thesis, Massachusetts Institute of Technology, 1974.

[15] L. J. STOCKMEYER, A. R. MEYER, Word Problems Requiring Exponential Time. In: *Symposium on Theory of Computing (STOC 1973)*. ACM Press, 1973, 1–9.

[16] D. WERNER, *Erweiterte union-free Sprachen über unärem Alphabet*. Bachelor's thesis, Universität Giessen, Institut für Informatik, 2013 (in German).

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 65–68.

Ordered Restarting Automata: Recent Results and Open Problems

Kent Kwee[(A)] Friedrich Otto[(A)]

[(A)]Fachbereich Elektrotechnik/Informatik, Universität Kassel
34109 Kassel, Germany
{kwee,otto}@theory.informatik.uni-kassel.de

Abstract

We summarize recent results on ordered restarting automata and present a number of open problems that concern the expressive power of the various types of ordered restarting automata, the closure properties of the defined classes of languages, decision problems, and the descriptional complexity of these types of automata.

1. Introduction

The *ordered restarting automaton* (ORWW-automaton), which was introduced by Mráz and Otto in [6], can be seen as a very special type of the *shrinking* restarting automaton of [1]. It has a window of size three, and during a combined rewrite/restart step, it just replaces the symbol in the middle position of its window by a symbol that is strictly smaller with respect to a predefined partial ordering on the tape alphabet. Of course, instead of the partial ordering, one could use a weight function, but the use of a partial ordering seems to be more intuitive.

In [6] the deterministic variant of the ORWW-automaton was used as a basic device for several types of two-dimensional restarting automata that accept picture languages. In [7], the study of the descriptional complexity of deterministic ORWW-automata was initiated, which was then continued in [2] and [8]. The nondeterministic ORWW-automaton was studied in detail in [5], further results are to appear in [3]. Finally, by separating the rewrite from the restart operation, we obtain the ORRWW-automaton that is studied in [4].

In the following we summarize the main results on ORWW- and ORRWW-automata obtained in these papers and present a number of open problems for future work.

2. Definitions

An *ORWW-automaton* is a one-tape machine that is described by an 8-tuple $M = (Q, \Sigma, \Gamma, \rhd, \lhd, q_0, \delta, >)$, where Q is a finite set of states containing the initial state q_0, 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,

$$\delta : (Q \times ((\Gamma \cup \{\rhd\}) \cdot \Gamma \cdot (\Gamma \cup \{\lhd\}) \cup \{\rhd\lhd\})) \to 2^{(Q \times \{\mathsf{MVR}\}) \cup \Gamma \cup \{\mathsf{Accept}\}}$$

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

(1) A *move-right step* has the form $(q', \mathsf{MVR}) \in \delta(q, a_1a_2a_3)$, 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 change from state q to state q'. Observe that no move-right step is possible if the window contains the right sentinal $\lhd$.

(2) A *rewrite/restart step* has the form $b \in \delta(q, a_1a_2a_3)$, 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, that is, the window is moved back to the left end of the tape, and M reenters its initial state q_0.

(3) An *accept step* has the form $\mathsf{Accept} \in \delta(q, a_1a_2a_3)$, 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 $\delta(q, u) = \emptyset$ for some state q and a word u, then M necessarily halts, when it is in state q seeing u in its window, and we say that M *rejects* in this situation. Further, the letters in $\Gamma \setminus \Sigma$ are called *auxiliary symbols*. If $|\delta(q, u)| \leq 1$ for all q and u, then M is a *deterministic ORWW-automaton* (det-ORWW-automaton), and if $Q = \{q_0\}$, then we call M a *stateless ORWW-automaton* (stl-ORWW-automaton) or a *stateless deterministic ORWW-automaton* (stl-det-ORWW-automaton), as in this case the state is actually not needed.

An input $w \in \Sigma^*$ is accepted by M if there is a computation of M which starts with the initial configuration $q_0 \rhd w \lhd$ and ends with an accept step. The language consisting of all (input) words that are accepted by M is denoted by $L(M)$.

The *ORRWW-automaton* is obtained from the ORWW-automaton by splitting the rewrite/restart operation into separate rewrite and restart operations. However, it is still required that an ORRWW-automaton executes exactly one rewrite operation in each cycle.

Definition 2.1 *A* stl-ORRWW-automaton *is described by a 7-tuple* $M = (\Sigma, \Gamma, \rhd, \lhd, \delta_1, \delta_2, >)$, *where* Σ, Γ, $\rhd$, $\lhd$, *and* $>$ *are defined as for ORWW-automata, and*

$$\delta_1 : ((\Gamma \cup \{\rhd\}) \cdot \Gamma \cdot (\Gamma \cup \{\lhd\})) \cup \{\rhd\lhd\} \to 2^{\Gamma \cup \{\mathsf{MVR}, \mathsf{Accept}\}}$$

and

$$\delta_2 : (\Gamma^{\leq 2} \cdot (\Gamma \cup \{\lhd\})) \to 2^{\{\mathsf{MVR}, \mathsf{Restart}, \mathsf{Accept}\}}$$

are the transition relations. *Here it is required that* $b > b'$ *holds for each rewrite instruction* $b' \in \delta_1(abc)$.

Given a word $w \in \Sigma^+$ as input, at first the transition relation δ_1 is used until either an accept instruction is executed, a rewrite instruction $b' \in \Gamma$ is executed, or the window contains a word for which δ_1 is undefined. In the first case, M accepts, in the second case the letter in the middle of the window is replaced by the letter b', the window is moved one step to the right, and the computation is continued using the transition relation δ_2. Finally, in the third case M simply halts without accepting. The transition relation δ_2, which is used in the second phase of a cycle *after* the execution of a rewrite step, shifts the window to the right until either an accept

instruction is executed, and then M accepts, until a restart instruction is executed, which resets the window to the left end of the tape and starts the next cycle, or until a window content is reached for which δ_2 is undefined. In the latter case M halts without accepting. For $w = \lambda$, there either is no applicable operation for the initial configuration $\rhd\lhd$, or $\delta_1(\rhd\lhd) = \{\mathsf{Accept}\}$.

Finally, for each type X of automaton, we denote by $\mathcal{L}(\mathsf{X})$ the class of languages that are accepted by automata of type X.

3. Language Classes and Their Inclusion Relations

Concerning the expressive power of the various types of ordered restarting automata, the following results have been obtained.

Theorem 3.1 (Cut-and-Paste Lemma) [5]
For each ORWW-automaton M, there exists a constant $N(M) > 0$ such that each word $w \in L(M)$, $|w| \geq N(M)$, has a factorization $w = xyz$ satisfying all of the following conditions:

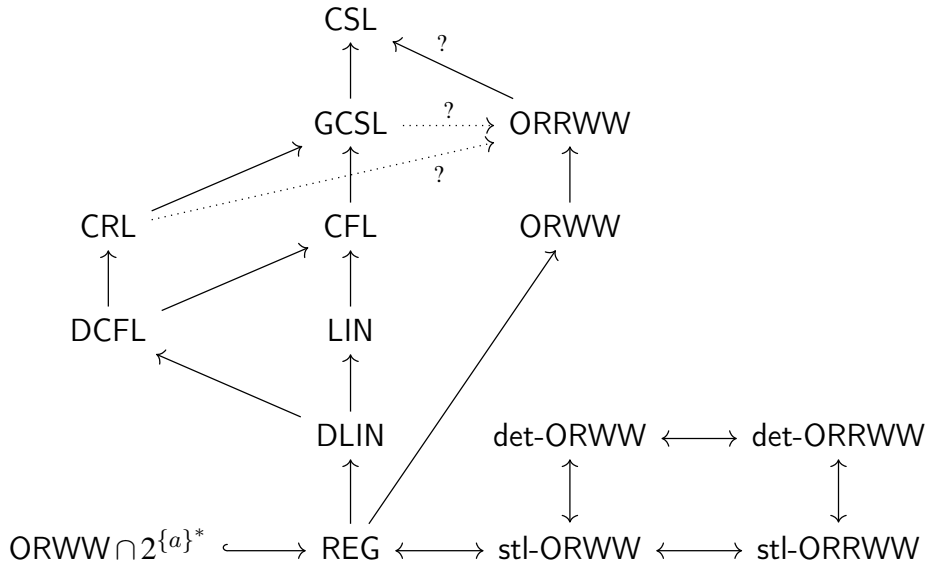
$$\text{(a) } |yz| \leq N(M), \text{ (b) } |y| > 0, \textit{ and } \text{(c) } xz \in L(M).$$

Actually, also a Pumping Lemma has been obtained for ORWW-automata that nicely complements the above lemma.

Theorem 3.2 (Pumping Lemma) [3]
For each ORWW automaton M, there exists a constant $N(M) > 0$ such that each word $w \in L(M)$, $|w| \geq N(M)$, has a factorization $w = xyz$ satisfying all of the following conditions:

$$\text{(a) } |xy| \leq N(M), \text{ (b) } |y| > 0, \textit{ and } \text{(c) } xy^m z \in L(M) \textit{ for all } m \geq 1.$$

We have the following hierarchy of language classes. It is still unknown whether GCSL or even CRL is contained in ORRWW. Additionally, we do not know if ORRWW is properly contained in CSL. The equality would imply NP = PSPACE. Additionally, we conjecture that $\mathcal{L}(\mathsf{ORWW})$ only contains semi-linear languages.



4. Closure Properties and Decision Problems

Concerning the closure properties for the nondeterministic language classes $\mathcal{L}(\mathsf{ORWW})$ and $\mathcal{L}(\mathsf{ORRWW})$, the following results are known.

Theorem 4.1 [5, 4] *The language class $\mathcal{L}(\mathsf{ORWW})$ is closed under union, intersection, product, Kleene star, inverse morphisms, and non-erasing morphisms, but it is not closed under complementation nor under reversal. The language class $\mathcal{L}(\mathsf{ORRWW})$ is closed under union, intersection, product, Kleene star, inverse morphisms, non-erasing morphisms, and reversal.*

We still do not know whether $\mathcal{L}(\mathsf{ORWW})$ is closed under arbitrary morphisms.

Theorem 4.2 [5, 3] *For ORWW-automata, the emptiness and finiteness problems are decidable.*

Theorem 4.3 [4] *For ORRWW-automata, emptiness, finiteness, universality, regularity, inclusion, and equivalence are all undecidable.*

Finally, there are still many open problems concerning the descriptional complexity of conversions and language operations.

References

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

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

[3] K. KWEE, F. OTTO, *A pumping lemma for ordered restarting automata*, 2016. Submitted.

[4] K. KWEE, F. OTTO, On ordered RRWW-automata. In: S. BRLEK, C. REUTENAUER (eds.), *DLT 2016, Proc.. Lect. Notes Comput. Sci. 9840*, Springer, Heidelberg, 2016, 268–279.

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

[6] F. MRÁZ, F. OTTO, Ordered restarting automata for picture languages. In: V. GEFFERT, B. PRENEEL, B. ROVAN, J. ŠTULLER, A. MIN TJOA (eds.), *SOFSEM 2014, Proc.. Lect. Notes Comput. Sci. 8327*, Springer, Heidelberg, 2014, 431–442.

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

[8] F. OTTO, M. WENDLANDT, K. KWEE, Reversible ordered restarting automata. In: J. KREVINE, J. STEFANI (eds.), *RC 2015, Proc.. Lect. Notes Comput. Sci. 9138*, Springer, Heidelberg, 2015, 60–75.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 69–72.

On Languages Accepted By Weighted Restarting Automata

Qichao Wang

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

1. Introduction

Restarting automata have been introduced as a formal model for the analysis by reduction [1], which a linguistic technique that is used to check the correctness of sentences of natural languages through sequences of local simplifications. After their introduction, many different types and variants of restarting automata have been introduced and studied. A recent overview on restarting automata is given in [3]. In order to study quantative 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 looking at different semirings S and different weight functions ω, various quantitative aspects of the behavior of M can be expressed through these functions. For example, for each input $w \in \Sigma^*$, the value $f(w)$ can be the number of accepting computations on the input w, or the minimal number of auxiliary symbols applied during an accepting computation on the input w. Therefore, 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 can be 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 $w \in L(M)$.

However, the acceptance conditions relative to different semirings are not equally powerful. For example, the *boolean semiring* $(\{0,1\},+,\cdot,0,1)$ can easily be simulated by other semirings. Here I study the *tropical semiring* and the semiring of some formal languages. In the latter case I restrict our attention to the *word-weighted* restarting automata introduced in [5], that is, the weight of each transition is taken to be a singleton. In [5] these automata are studied as transducers, while here I use them as language acceptors.

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, 2]. 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 I 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 R-automaton that is defined by taking $Q := \{q_0, q_r\}$, $\Gamma := \Sigma := \{a, b\}$, and $k := 4$, where δ is defined as follows:*

$$\begin{array}{llll}
t_1: & (q_0, ¢aaa) \to (q_0, \mathsf{MVR}), & t_7: & (q_0, abb\$) \to (q_r, b\$), \\
t_2: & (q_0, aaaa) \to (q_0, \mathsf{MVR}), & t_8: & (q_0, abb\$) \to (q_r, \$), \\
t_3: & (q_0, aaab) \to (q_0, \mathsf{MVR}), & t_9: & (q_0, ¢ab\$) \to \mathsf{Accept}, \\
t_4: & (q_0, aabb) \to (q_0, \mathsf{MVR}), & t_{10}: & (q_0, ¢\$) \to \mathsf{Accept}, \\
t_5: & (q_0, abbb) \to (q_r, bb), & t_{11}: & (q_0, ¢aab) \to (q_0, \mathsf{MVR}), \\
t_6: & (q_0, abbb) \to (q_r, b), & t_{12}: & (q_0, ¢abb) \to (q_0, \mathsf{MVR}), \\
t_{13,x}: & (q_r, x) \to \mathsf{Restart} \text{ for all admissible } x. & &
\end{array}$$

It is easily seen that $L(M_1) = \{ a^m b^n \mid 0 \leq m \leq n \leq 2m \}$. Further, for $a^n b^n$ and for $a^n b^{2n}$, M_1 has just a single accepting computation.

Let $(\mathsf{REG}(\Delta), \cup, \cdot, \emptyset, \{\lambda\})$ be the semiring of regular languages over $\Delta = \{c, d\}$, let ω_1 be the weight function that assigns the set $\{c\}$ to the transitions t_5, t_7 and t_9, that assigns the set $\{d\}$ to the transitions t_6 and t_8, and that assigns the set $\{\lambda\}$ to all other transitions. Finally, let $\mathcal{M}_1 = (M_1, \omega_1)$, and let

$$T_1 = \{ \{c^m\} \mid m \geq 0 \} \cup \{ \{d^n\} \mid n \geq 0 \}.$$

Then $f_{\omega_1}^{M_1}(w) \in T_1$ iff $w \in L(M_1)$, and $|w|_a = |w|_b$ or $2 \cdot |w|_a = |w|_b$, which yields

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

It is known that the language $L_{T_1}(\mathcal{M}_1)$ is not even accepted by any RW-automaton [2]. Hence, we see that the notion of relative acceptance increases the expressive power of R-automata. In this work, I will compare some different semirings and acceptance conditions.

Literatur

[1] P. JANCAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, Restarting Automata. In: H. REICHEL (ed.), *FCT'95*. Lecture Notes in Computer Science 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] F. OTTO, Restarting Automata. In: Z. ÉSIK, C. MARTÍN-VIDE, V. MITRANA (eds.), *Recent Advances in Formal Languages and Applications*. 25, Springer, Heidelberg, 2006, 269–303.

[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, N. HUNDESHAGEN, F. OTTO, Weighted Restarting Automata and Pushdown Relations. In: A. MALETTI (ed.), *Algebraic Informatics - 6th International Conference, CAI 2015, Proceedings*. Lecture Notes in Computer Science 9270, Springer, Switzerland, 2015, 196–207.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 73–76.

Uniformly Automatic Classes and Their Application in Finite Model Theory

Faried Abu Zaid[(A)] Erich Grädel[(B)] Frederic Reinhardt[(B)]

[(A)]TU Ilmenau
faried.abu-zaid@tu-ilmenau.de
[(B)]RWTH Aachen University
{graedel, reinhardt}@logic.rwth-aachen.de

Abstract

We consider uniformly automatic classes, which are classes that are generated by a fixed automatic presentation and a set of advices. Prototypic examples are graphs of bounded treewidth, the torsion free abelian groups of rank 1, and the class of all countable linear orders. Uniformly automatic classes are found in the core of several algorithmic meta-theorems. We investigate the efficiency of this approach by analysing the runtime of the generic automata based model checking algorithm in terms of the complexity of the presentation. We show that the runtime on a presentation of the direct product closure is only one exponential higher than the runtime on the primal class. We apply these findings to show that first-order model checking is fixed parameter tractable on the classes of all finite boolean algebras and the class of all finite abelian groups. In both cases the parameter dependence of the runtime is elementary. The runtime, which we achieve on these classes is either provably optimal or outperforms other known approaches. Further we show that the runtime of the generic automata based algorithm for MSO model checking on graphs of treedepth at most d has an $(d+1)$-fold exponential parameter dependence, which matches the runtime of the best known algorithm for model checking on these classes.

1. Introduction

A structure is automatic is it has a finite presentations by automata. Roughly speaking, an isomorphic copy of the structure consists of a regular domain, such that all relations of the structure are recognizable by synchronous multi-tape automata. Equivalently a structure is automatic if it is finite-set interpretable in $(\mathbb{N}, <)$. The history of automatic structures reaches back to the early days of automata theory, where logicians like Büchi and Rabin provided automata theoretic decision procedures for the theory of structures like Presburger arithmetic. A more systematic investigation was started by Khoussainov and Nerode [9] who also coined the term automatic structures. In [2] the concept was lifted from finite words to automata that read trees as well as their infinite counter parts ((ω-)[tree]automatic presentations). For an overview we refer the reader to [1].

A lot of research in the field of automatic structures is devoted to the characterisation of the automatic models inside certain classes of structures. In particular a long standing open problem

was whether addition on the rationals is automatic, until Tsankov [12] gave a negative answer to the question. It was noted, however, that $(\mathbb{Q}, +)$ is "almost" automatic in the sense that there is a presentation in which addition is automatic, but the domain is not a regular set [11]. Kruckman et al. remarked in [10] that also the domain becomes regular, if the automaton has access to a fixed infinite advice string. Further, this advice string has a decidable MSO-theory, which is sufficient to give an automaton based decision procedure for the theory of $(\mathbb{Q}, +)$.

This motivates our studies of advice automatic structures. A structure is advice automatic if it has an automatic presentation in the same way as $(\mathbb{Q}, +)$ does, i.e., it can be represented by automata that have access to a fixed advice. Such (ω-)[tree]automatic presentations with advice correspond to (finite) set interpretations in the respective structures expanded by unary relations. This setting appeared occasionally in the literature (e.g. [4]) but to the authors knowledge no systematic investigation has been initiated so far. Further we lift this notion from structures to classes of structures that can be represented by a single presentation and regular set of advices, leading to uniformly automatic classes.

2. Classes of Finite Structures

We consider the application of uniform tree-automaticity in finite model-theory. More precisely we investigate the use of automata in algorithmic meta-theorems.

The prototype of an automata based algorithmic meta-theorem is the aforementioned Theorem of Courcelle. Interestingly, the presentations, which build the core of this theorem, correspond to MSO interpretations (instead of the stronger set-interpretations) and therefore are located more on the lower end of what is presentable by uniform automatic presentations (while on the other hand allowing MSO-to-MSO translations instead of FO-to-MSO translations).

We investigate atomaticity as generic notion of simpleness which might bring up new and interesting classes of structures with a fixed parameter tractable model checking problem. As a first step to develop the theory, we are concerned with the efficiency of this approach. Note that the non-elementary worst-case runtime of the automaton construction leads in general to a non-elementary parameter dependence in the algorithmic meta-theorems. However, in recent years several application have been found where the generic automata based approach to MSO achieves an almost optimal runtime, even when the actual complexity of the solved problem is much lower than non-elementary. Frick and Grohe [7] showed, unless PTIME = NP, there is no algorithm that solves the model checking problem for MSO on words (and hence also on trees) in time

$$\mathcal{O}(f(|\varphi|) \cdot \text{poly}(|t|))$$

for any elementary function $f : \mathbb{N} \to \mathbb{N}$. A similar statement holds for first-order logic on words. This renders Courcelle's approach to model checking of graphs with bounded treewidth or cliquewith optimal. Eisinger [6] gave a triply-exponential upper bound on the size of the minimal automaton for formulae of integer and mixed real addition. In [5] Durand-Gasselin and Habermehl showed for word automatic structures that the runtime of the generic algorithm can be bounded by a function which estimates how well the presentation goes along with the Ehrenfeucht-Fraisse relations of the structure and gave runtime bounds for integer addition matching the bound of Eisinger. Additionally they gave a triply exponential bound for automatic graphs of bounded degree.

We adopt the technique of Durand-Gasselin and Habermehl and generalise their result to uniformly tree-automatic presentations. We apply this technique to the presentations of the direct product closures that we described earlier. We prove that the runtime of the model checking algorithm is at most exponential in the runtime of the primal classes. Further we apply these findings in the context of FPT model checking for first order logic. We demonstrate the efficiency of the automata theoretic approach by analysing the runtime in terms the parameter dependence on structurally rather simple classes. Our results are as follows.

- First-order model checking is FPT on the class of all finite Boolean algebras, succinctly encoded by the number of atoms in unary, and can be performed in

  $$\mathcal{O}(\exp_2(\text{poly}(|\varphi|)) \cdot \log|\mathfrak{B}|).$$

  Unless NEXP $=$ STA$(*, 2^{cn}, n)$, this is optimal.

- First-order model checking is FPT on the class of all finite abelian groups, succinctly encoded by the orders of the direct product factors, and can be performed in

  $$\mathcal{O}(\exp_4(\text{poly}(|\varphi|)) \cdot \log|\mathfrak{G}|).$$

  This answers a question by Grohe, who asked if (among other algebraic classes) FO model checking is FPT on the class of all finite abelian groups. We should mention that the mere FPT result was independently discovered by Bova and Martin [3]. However, our approach has two advantages. Their algorithm yields a non-elementary parameter dependence and it does not work for succinct encodings.

  We slightly generalise the result to finite groups of bounded non-Abelian decomposition width. We obtain the same asymptotic runtime on these classes.

- Monadic second-order model checking is FPT on every class of graphs with treedepth at most d and can be performed in

  $$\mathcal{O}\left(\exp_{d+1}(\text{poly}(|\varphi|)) \cdot \text{poly}(|\mathfrak{G}|)\right).$$

  This matches best known runtime for these classes due to Gajarsky and Hlinený [8], which was achieved by a kernelisation procedure. Our proof makes use of their analysis.

References

[1] V. BÁRÁNY, E. GRÄDEL, S. RUBIN, Automata-Based Presentations of Infinite Structures. In: J. ESPARZA, C. MICHAUX, C. STEINHORN (eds.), *Finite and Algorithmic Model Theory*. 2011, 1–76.
http://logic.rwth-aachen.de/pub/graedel/surveyBGR.pdf

[2] A. BLUMENSATH, *Automatic Structures*. 1999. Published: Diploma thesis, RWTH-Aachen.
http://www.logic.rwth-aachen.de/pub/blume/AutStr.ps.gz

[3] S. BOVA, B. MARTIN, First-Order Queries on Finite Abelian Groups. In: S. KREUTZER (ed.), *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*. Leibniz International Proceedings in Informatics (LIPIcs) 41, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2015, 41–59.
http://drops.dagstuhl.de/opus/volltexte/2015/5406

[4] T. COLCOMBET, C. LÖDING, Transforming structures by set interpretations. *CoRR* **abs/cs/0703039** (2007).
http://arxiv.org/abs/cs/0703039

[5] A. DURAND-GASSELIN, P. HABERMEHL, Ehrenfeucht-Fraïssé goes elementarily automatic for structures of bounded degree. In: C. DÜRR, T. WILKE (eds.), *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*. LIPIcs 14, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012, 242–253.

[6] J. EISINGER, Upper Bounds on the Automata Size for Integer and Mixed Real and Integer Linear Arithmetic (Extended Abstract). In: M. KAMINSKI, S. MARTINI (eds.), *Computer Science Logic*. Number 5213 in Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, 431–445. DOI: 10.1007/978-3-540-87531-4_31.
http://link.springer.com/chapter/10.1007/978-3-540-87531-4_31

[7] M. FRICK, M. GROHE, The complexity of first-order and monadic second-order logic revisited. In: *17th Annual IEEE Symposium on Logic in Computer Science, 2002. Proceedings*. 2002, 215–224.

[8] J. GAJARSKY, P. HLINENY, Faster Deciding MSO Properties of Trees of Fixed Height, and Some Consequences. In: D. D'SOUZA, T. KAVITHA, J. RADHAKRISHNAN (eds.), *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012)*. Leibniz International Proceedings in Informatics (LIPIcs) 18, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2012, 112–123.
http://drops.dagstuhl.de/opus/volltexte/2012/3855

[9] B. KHOUSSAINOV, A. NERODE, Automatic presentations of structures. In: D. LEIVANT (ed.), *Logic and Computational Complexity*. Number 960 in Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1995, 367–392.
http://link.springer.com/chapter/10.1007/3-540-60178-3_93

[10] A. KRUCKMAN, S. RUBIN, J. SHERIDAN, B. ZAX, A Myhill-Nerode theorem for automata with advice. *Electronic Proceedings in Theoretical Computer Science* **96** (2012), 238–246. ArXiv: 1210.2462.
http://arxiv.org/abs/1210.2462

[11] A. NIES, Describing groups. *Bulletin of Symbolic Logic* **13** (2007) 3, 305–339.
http://projecteuclid.org/euclid.bsl/1186666149

[12] T. TSANKOV, The additive group of the rationals does not have an automatic presentation. *Journal of Symbolic Logic* **76** (2011) 4, 1341–1351.
http://projecteuclid.org/euclid.jsl/1318338853

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 77–79.

Weighted Register Automata and Weighted Logic on Data Words

Parvaneh Babari[X] Manfred Droste[X] Vitaly Perevoshchikov[X]

[A]Universität Leipzig
{babari,droste,perev}@informatik.uni-leipzig.de

Zusammenfassung

In the areas of static analysis of databases and software verification, there is much interest in processing information taken from an infinite domain. The notion of data words is a well-known concept for the modelling of these situations. A data word can be considered as a sequence of pairs where the first element is taken from a finite alphabet (as in classical words) and the second element is taken from an infinite data domain. Register automata introduced by Kaminski and Francez [18] provide a widely studied model for reasoning on data words. These automata can be considered as classical nondeterministic finite automata equipped with a finite set of registers which are used to store data in order to compare it with some data in the future. This enables them to handle parameters like user names, passwords, identifiers of connections, etc., in a fashion similar to, and slightly more expressive than, the class of *data-independent* systems. This model served as a basis for the study of various automata models and logics on data words and trees [6, 9, 16, 17, 19]. Classical *timed automata* of Alur and Dill [2] for real-time systems are another example of automata on data (timed) words.

For many Computer Science applications, quantitative properties of systems such as costs, probabilities, vagueness and uncertainty of a statement, consumption of memory and energy are of significant importance. Weighted automata (cf. [13] for surveys) form a well-known model for quantitative aspects which has been extended to various different settings (cf., e.g., weighted timed automata [5, 20] for the quantitative analysis of real-time systems). In this paper, we introduce *weighted register automata* for quantitative reasoning on data words. Motivated by the seminal Büchi-Elgot-Trakhtenbrot theorem [11] about the expressive equivalence of finite automata and monadic second-order (MSO) logic and by the weighted MSO logic of Droste and Gastin [12], we introduce *weighted MSO logic on data words* and give a logical characterization of weighted register automata.

To the best of our knowledge, quantitative extensions of register automata have not been studied yet. However, there is a plenty of quantitative models which are quite natural in the context of register automata. We give several examples. For instance, since register automata process data taken from an infinite domain, there arises a natural question about the efficiency of the processing of big data which can invoke big costs for data storing and data comparison. As another example, one may ask about the maximal size of data which will be stored in registers along a computation of a register automaton. Thus, our goal is to develop a model which will reflect this kind of data-dependent costs. Now we give a summary of our results.

We introduce weighted register automata over *commutative data semirings* equipped with a collection of binary data functions in the spirit of the classical theory of weighted automata [13]. Whereas in the models of register automata known from the literature data are usually compared with respect to equality or a linear order, here we allow data comparison by means of an arbitrary collection of binary data relations. This approach permits easily to incorporate timed automata [2] and weighted timed automata [5, 20] into our framework. Moreover, this approach gives rise to the further investigations of the models for data processing, e.g., data comparison with an approximation error.

We introduce semiring-weighted existential MSO logic on data words equipped with binary data functions. Recall that weighted MSO logic on classical words is introduced and investigated in [12]. In order to model data comparison, we use the data predicates in the spirit of relative distance predicates of Wilke [21]. Our goal is to prove the expressive equivalence of this new logic with weighted register automata. To reach this goal, we faced the following difficulties.

- The unrestricted use of binary data functions and weighted universal quantifiers goes beyond recognizability even for very simple formulas.
- Register automata are neither determinizable nor closed under complement.

In order to overcome these problems, we obtain the suitable fragment of our weighted existential MSO logic by restricting the use of the weighted universal quantifier to formulas without weighted quantifiers and by restricting the use of data functions to an intuitively defined logical operator. In our main result, we state that this restricted weighted EMSO-logic is equivalent to weighted register automata. The existing proof techniques and ideas for weighted logic cannot be applied in our setting, since register automata are not closed under complement and we need a new construction to deal with binary data functions. For this purpose, we introduce a determinizable class of unweighted register automata, called visibly register automata. In visibly register automata, the transition label determines which registers will be updated when taking a transition. This determinizable class of register automata could be also of independent interest. Moreover, we introduce a new normalization technique for the binary data functions, in order to provide a translation of formulas with weighted universal quantifiers into weighted register automata. With this, we achieve our goal; moreover, our construction of weighted register automata equivalent to a given weighted MSO formula is effective.

Literatur

[1] Alur, R., D'Antoni, L., Deshmukh, J., Raghothaman, M., Yuan, Y.: Regular functions and cost register automata. In: LICS 2013, pp. 13–22. IEEE Computer Society (2013).

[2] Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994).

[3] Alur, R., Fix, L., Henzinger, T.: Event-clock automata: a determinizable class of timed automata. Theoretical Computer Science 211(1-2), 253–273 (1999).

[4] Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC 2004, pp. 202?211. ACM (2004).

[5] Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. In: HSCC 2001. LNCS, vol. 2034, pp. 49–62. Springer (2001).

[6] Bojańczyk, M., David, C., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data words. ACM Trans. Comput. Log 12(4), 27 pages (2011).

[7] Bollig, B.: An automaton over data words that captures EMSO logic. In: CONCUR 2011. LNCS, vol. 6901, pp. 171–186. Springer (2011).

[8] Bollig, B., Gastin, P.: Weighted versus probabilistic logics. In: DLT 2009. LNCS, vol. 5583, pp. 18–38. Springer (2009).

[9] Bouyer, P.: A logical characterization of data languages. Inf. Process. Lett. 84(2), 75–85 (2002).

[10] Bouyer, P., Petit, A., Thérien, D.: An algebraic characterization of data and timed languages. In: CONCUR 2001. LNCS, vol. 2154, pp. 248–261. Springer (2001).

[11] Büchi, J.R.: Weak second order arithmetic and finite automata. Zeitschrift für Mathematische Logik und Grundlagen der Informatik 6, 66–92 (1960).

[12] Droste, M., Gastin, P.: Weighted automata and weighted logics. Theoret. Comp. Sci. 380(1-2), 69-86 (2007).

[13] Droste, M., Kuich, W., Vogler, H. (eds.): Handbook of Weighted Automata. EATCS Monographs on Theoretical Computer Science. Springer (2009).

[14] Droste, M., Perevoshchikov, V.: A Nivat theorem for weighted timed automata and relative distance logic. In: ICALP 2014. LNCS, vol. 8573, pp. 171–182. Springer (2014).

[15] Demri, S., Lazić, R.: LTL with the freeze quantifier and register automata. ACM Trans. Compute. Logic, 10(3), 30 pages (2009).

[16] Figueira, D.: Alternating register automata on finite data words and trees. Logical Methods in Computer Science 8(1:22), 1–43 (2012).

[17] Figueira, D., Hofman, P., Lasota, S.: Relating timed and register automata. In: EXPRESS 2010. EPTCS 41, pp. 61–75 (2010).

[18] Kaminski, M., Francez, N.: Finite-memory automata. Theoretical Computer Science 134, 329–363 (1994).

[19] Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Transactions on Computational Logic 5(3), 403–435 (2004).

[20] Quaas, K.: MSO logics for weighted timed automata. Formal Methods in System Design 38(3), 193–222 (2011).

[21] Wilke, T.: Specifying timed state sequences in powerful decidable logics and timed automata. In: FTRTFT 1994. LNCS, vol. 863, pp. 694–715. Springer (1994).

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 81–84.

First-Order Logic with Reachability for Valence Automata over Graph Monoids

Emanuele D'Osualdo[(A)] Roland Meyer[(A)] Georg Zetzsche[(B)]

[(A)]Fachbereich Informatik, Technische Universit"at Kaiserslautern, Germany
{dosualdo,meyer}@cs.uni-kl.de

[(B)]LSV, CNRS & ENS Cachan, Université Paris-Saclay, France
zetzsche@lsv.fr

1. Introduction

First-order logic with reachability ($\mathsf{FO}[\mathsf{R}]$) can uniformly specify a wide range of correctness properties. Moreover, $\mathsf{FO}[\mathsf{R}]$ can provide a useful middle ground between first-order logic without reachability and monadic second-order logic: The latter significantly increases expressivity but at the cost of being decidable for fewer models.

However, to the best of our knowledge, there is no systematic account of decidability of $\mathsf{FO}[\mathsf{R}]$ for general classes of storage mechanisms. The question we address in this paper is:

Which features of the storage mechanism of the automaton model determine the decidability status of $\mathsf{FO}[\mathsf{R}]$*?*

In order to investigate this question, we need a unifying framework that encompasses and generalizes all the aforementioned infinite state models. We thus set out to study $\mathsf{FO}[\mathsf{R}]$ on the configuration graphs of *valence systems*: Finite state automata with auxiliary storage, where the storage mechanism is specified via a monoid. In this model, a configuration of a valence system over a monoid M consists of a pair (q, m) where q is a control state and m is the storage content, an element of M. A transition (q, r, q') has the effect of multiplying $r \in M$ (to the right) to the current storage content. In order to model actions that are not allowed at the current storage content (such as popping the wrong stack symbol), we restrict the configurations to right-invertible monoid elements. As an example, a stack can be modeled by the monoid whose elements can be represented by sequences of non-barred (a) and barred ($\bar{a}$) versions of each stack symbol (representing a push and a pop respectively) with the rule $a\bar{a} = \mathbf{1}$. Then, the right-invertible elements are the sequences over non-barred symbols. A counter over $\mathbb{N}$ with increment and decrement (which we call a *partially blind counter*) can be similarly modeled by having only one stack symbol. Correspondingly, a *blind counter* is a counter over $\mathbb{Z}$, represented by also having $a\bar{a} = \bar{a}a$.

[(B)]This author is supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD).

Most importantly, new storage mechanisms can be built by composing monoids using two kinds of products. The direct product $M \times N$ of two monoids M and N allows the automaton to use the two storage mechanisms independently. This permits, for example, building counter systems of any finite dimension. The free product $M * N$ amounts, roughly speaking, to forming stacks whose entries alternate between elements of M and elements of N. A valence system over $M * N$ can push elements of N or M into the stack, manipulate the top entry using the relevant storage mechanism (N or M) and pop the top of the stack when empty.

This is an extended abstract of the paper [1].

2. Concepts and Main Result

Valence Systems We represent storage mechanisms as monoids. Let M be a monoid; we denote multiplication in the monoid by juxtaposition, and the neutral element by $\mathbf{1}$. The elements of M are actions that can be applied to the storage content and the elements of

$$\mathcal{R}_1(M) \coloneqq \{x \in M \mid \exists y \in M \colon xy = \mathbf{1}\}$$

correspond to valid storage contents. The elements of $\mathcal{R}_1(M)$ are said to be *right-invertible*. A *valence system over* M is a tuple $S = (Q, M, E)$, in which 1. Q is a finite set of *states*, 2. E is a finite subset of $Q \times M \times Q$, called the set of *edges*. A *configuration of* S is a pair $(q, r) \in Q \times \mathcal{R}_1(M)$. For $p, q \in Q$ and $r, s \in \mathcal{R}_1(M)$, we write $(p, r) \to_S (q, s)$ if there is an edge $(p, m, q) \in E$ such that $s = rm$. The reflexive transitive closure of $\to_S$ is denoted $\to_S^*$.

Graphs A *graph* is a pair $\Gamma = (V, E)$ where V is a finite set and E is a subset of $\{S \subseteq V \mid 1 \leq |S| \leq 2\}$. The elements of V are called *vertices* and those of E are called *edges*. Vertices $v, w \in V$ are *adjacent* if $\{v, w\} \in E$. If $\{v\} \in E$ for some $v \in V$, then v is called a *looped* vertex, otherwise it is *unlooped*. A *subgraph* of Γ is a graph (V', E') with $V' \subseteq V$ and $E' \subseteq E$. Such a subgraph is called *induced (by V')* if $E' = \{S \in E \mid S \subseteq V'\}$, i.e. E' contains all edges from E between vertices from V'. By $\Gamma \setminus \{v\}$, for $v \in V$, we denote the subgraph of Γ induced by $V \setminus \{v\}$. Moreover, a *clique* is a graph in which any two distinct vertices are adjacent.

Graph monoids Let A be a (not necessarily finite) set of symbols and R be a subset of $A^* \times A^*$. The pair (A, R) is called a *(monoid) presentation*. The smallest congruence of A^* containing R is denoted by $\equiv_R$ and we will write $[w]_R$ for the congruence class of $w \in A^*$. The *monoid presented by* (A, R) is defined as $A^*/\equiv_R$. Note that since we did not impose a finiteness restriction on A, up to isomorphism, every monoid has a presentation. Furthermore, for monoids M_1, M_2 we can find presentations (A_1, R_1) and (A_2, R_2) such that $A_1 \cap A_2 = \emptyset$. We define the *free product* $M_1 * M_2$ to be presented by $(A_1 \cup A_2, R_1 \cup R_2)$. Note that $M_1 * M_2$ is well-defined up to isomorphism. In analogy to the n-fold direct product, we write $M^{(n)}$ for the n-fold free product of M.

With each graph $\Gamma = (V, E)$, we associate the alphabet of *generators* $X_\Gamma \coloneqq \{a_v, \bar{a}_v \mid v \in V\}$,

and the smallest congruence $\equiv_\Gamma$ satisfying

$$a_v\bar{a}_v \equiv_\Gamma \varepsilon \qquad \text{for each } v \in V, \text{ and} \tag{1}$$

$$xy \equiv_\Gamma yx \qquad \text{for each } x \in \{a_v, \bar{a}_v\}, \; y \in \{a_w, \bar{a}_w\} \text{ with } \{v,w\} \in E. \tag{2}$$

In particular, we have $a_v\bar{a}_v \equiv_\Gamma \bar{a}_v a_v$ whenever $\{v\} \in E$. With each graph Γ, we associate the monoid $\mathbb{M}\Gamma := X_\Gamma^*/\equiv_\Gamma$. The monoids of the form $\mathbb{M}\Gamma$ are called *graph monoids*.

Let us briefly discuss how to realize storage mechanisms by graph monoids. First, suppose Γ_0 and Γ_1 are disjoint graphs. If Γ is the union of Γ_0 and Γ_1, then $\mathbb{M}\Gamma \cong \mathbb{M}\Gamma_0 * \mathbb{M}\Gamma_1$ by definition. Moreover, if Γ is obtained from Γ_0 and Γ_1 by drawing an edge between each vertex of Γ_0 and each vertex of Γ_1, then $\mathbb{M}\Gamma \cong \mathbb{M}\Gamma_0 \times \mathbb{M}\Gamma_1$.

If Γ consists of one vertex v and has no edges, the only congruence rule is $a_v\bar{a}_v \equiv_\Gamma \varepsilon$. In this case, $\mathbb{M}\Gamma$ is also denoted as $\mathbb{B}$ and we will refer to it as the *bicyclic monoid*. The generators a_v and $\bar{a}_v$ are then also written a and $\bar{a}$, respectively. Observe that $\mathcal{R}_1(\mathbb{B}) = \{a\}^*$ and thus $\mathcal{R}_1(\mathbb{B}) \cong \mathbb{N}$. Multiplying a or $\bar{a}$ on the right represents increment or decrement, respectively. Since a and $\bar{a}$ do not commute, $\mathbb{B}$ corresponds to a *partially blind counter*, i.e. one that attains only non-negative values. For instance, starting from a, we cannot decrement twice and add one by multiplying $\bar{a}\bar{a}a$ because $a\bar{a}\bar{a}a \equiv \bar{a}a \notin \mathcal{R}_1(\mathbb{B})$.

Graph Γ	Monoid $\mathbb{M}\Gamma$	Storage mechanism
	$\mathbb{B}^{(3)}$	Stack (of 3 symbols)
	$\mathbb{B}^3$	Three partially blind counters
	$\mathbb{Z}^3$	Three blind counters
	$\mathbb{B}^{(2)} \times \mathbb{B}^3$	Stack (of two symbols) and three partially blind counters
	$\mathbb{B} * \mathbb{B}^3$	Stack of three partially blind counters
	$\mathbb{B}^{(2)} \times \mathbb{B}^{(2)}$	Two stacks (of two symbols each)

Figure 1: Examples of storage mechanisms

Moreover, if Γ consists of one looped vertex, then $\mathbb{M}\Gamma$ (and also $\mathcal{R}_1(\mathbb{M}\Gamma)$) is isomorphic to $\mathbb{Z}$ and thus realizes a *blind counter*, which can go below zero.

If one storage mechanism is realized by a monoid M, then the monoid $\mathbb{B} * M$ corresponds to the mechanism that *builds stacks*: A configuration of this new mechanism consists of a sequence $c_0ac_1 \cdots ac_n$, where $c_0, \ldots, c_n$ are configurations of the mechanism realized by M. We interpret this as a stack with the entries $c_0, \ldots, c_n$. One can open a new stack entry on top (by multiplying $a \in \mathbb{B}$), remove the topmost entry if empty (by multiplying $\bar{a} \in \mathbb{B}$) and operate on the topmost entry using the old mechanism (by multiplying elements from M). In particular, $\mathbb{B} * \mathbb{B}$ describes a pushdown stack with two stack symbols. Observe that $\mathcal{R}_1(\mathbb{B} * \mathbb{B})$ is isomorphic to the monoid of words over two symbols, i.e. stack words. Figure 1 shows more examples. See [5] for details.

Logic Let S be a valence system over the monoid M. Then the *first-order logic with reachability for* S, $\mathsf{FO}[\mathsf{R}](S)$, consists of the first-order formulas over the following signature:

- The *constant symbols* are the configurations of S.

- $\mathsf{state}_q(\cdot)$: A unary *state predicate* for each $q \in Q$.
- $\mathsf{step}(\cdot,\cdot)$: The binary *step predicate* for one-step reachability.
- $\mathsf{reach}(\cdot,\cdot)$: The binary *reachability predicate*.

Given a valence system S, its *reachability structure* is the relational structure with domain $Q \times \mathcal{R}_1(M)$ over the above signature defined in the obvious way. For a first-order formula φ, we write $S \models \varphi$ when φ holds on the reachability structure of S. The *first-order theory with reachability over* S is then the set of formulas $\{\varphi \in \mathsf{FO}[\mathsf{R}](S) \mid \varphi \text{ closed}, S \models \varphi\}$.

Main Result We call Γ a $\mathbb{B}^2$*-triangle* if it is one of the graphs of fig. 2. A graph Γ is said to be $\mathbb{B}^2$*-triangle-free* if it does not contain a $\mathbb{B}^2$-triangle as an induced subgraph.

Figure 2: $\mathbb{B}^2$-triangles

Theorem 2.1 *Let Γ be a graph. The first-order theory with reachability for valence systems over $\mathbb{M}\Gamma$ is decidable if and only if Γ is a disjoint union of $\mathbb{B}^2$-triangle-free cliques.*

Operationally, the storage mechanism implemented by such an $\mathbb{M}\Gamma$ is composed of a stack, each entry of which is either a) a pair of partially blind counters, or b) a partially blind counter with a number of blind counters.

To establish our decidability results, we show that the reachability structures are effectively automatic [2], employing a characterization of reachability relations of two-dimensional vector addition systems with states due to Leroux and Sutre [3]. For the undecidability results, we first show that the decomposition into cliques is needed. If Γ is not a disjoint union of cliques, we can reduce the universality problem of rational subsets of $\{a,b\}^* \times \{c\}^*$ [4]. On the other hand, if Γ decomposes into cliques but is not $\mathbb{B}^2$-triangle-free, then we show how to interpret the Σ_1 theory of arithmetic with addition and multiplication, $(\mathbb{N},+,\cdot)$, in the Σ_2 fragment of $\mathsf{FO}[\mathsf{R}]$ on valence systems. In both cases, undecidability already holds for a fixed valence system.

References

[1] E. D'Osualdo, R. Meyer, G. Zetzsche, First-order logic with reachability for infinite-state systems. 2016. To appear in Proc. of the Thirty-First Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2016).

[2] B. Khoussainov, A. Nerode, Automatic Presentations of Structures. In: *LCC: International Workshop on Logic and Computational Complexity*. LNCS 960, Springer, 1995, 367–392.

[3] J. Leroux, G. Sutre, On Flatness for 2-Dimensional Vector Addition Systems with States. In: *CONCUR*. LNCS 3170, Springer, 2004, 402–416.

[4] J. Sakarovitch, The "last" decision problem for rational trace languages. In: *LATIN*. LNCS 583, Springer, 1992, 460–473.

[5] G. Zetzsche, *Monoids as Storage Mechanisms*. Ph.D. thesis, Technische Universität Kaiserslautern, 2016. Available at `http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:hbz:386-kluedo-44003`.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 85–85.

Efficient Algorithms for Morphisms over Omega-Regular Languages

Lukas Fleischer Manfred Kufleitner

University of Stuttgart, Formal Methods in Computer Science
{fleischer,kufleitner}@fmi.uni-stuttgart.de

Morphisms to finite semigroups can be used for recognizing omega-regular languages. The so-called strongly recognizing morphisms can be seen as a deterministic computation model which provides minimal objects (known as the syntactic morphism) and a trivial complementation procedure. We give a quadratic-time algorithm for computing the syntactic morphism from any given strongly recognizing morphism, thereby showing that minimization is easy as well. In addition, we give algorithms for efficiently solving various decision problems for weakly recognizing morphisms. Weakly recognizing morphism are often smaller than their strongly recognizing counterparts. Finally, we describe the language operations needed for converting formulas in monadic second-order logic (MSO) into strongly recognizing morphisms.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 87–90.

A Logic for Document Spanners

Dominik D. Freydenberger[A]

[A]University of Bayreuth
ddfy@ddfy.de

1. Introduction

Fagin et al. [2] introduced *core spanners* as a formal framework for information extraction that describes the properties of the annotation query language AQL, which is used in IBM's SystemT. The basic building block of core spanners are *primitive spanners*, which can be understood as automata or regular expressions with variables that map an input string w to a table with intervals positions of w (similar to the way search-and-replace functionality is usually implemented). Core spanners then combine these primitive spanners with the relational operators projection (i. e., discarding columns of the table), union (i. e., combining tables that have the same columns), natural join (on the positions in the table), and string equality selection (i. e., keeping only rows where two chosen intervals of positions describe the same substring of w).

As shown by Freydenberger and Holldack [4], each core spanner representation can be converted in polynomial time into an equivalent formula of the *existential theory of concatenation with rational constraints* $\mathsf{EC}^{\mathsf{reg}}$. (As a consequence, satisfiability and another property of core spanners can be decided in PSPACE; using this result and the fact that satisfiability of $\mathsf{EC}^{\mathsf{reg}}$ can be decided in PSPACE, see Diekert [1]). Furthermore, every $\mathsf{EC}^{\mathsf{reg}}$-formula can be converted into a equisatisfiable core spanner (but the construction cannot be used directly to simulate $\mathsf{EC}^{\mathsf{reg}}$ without encodings on the input word of the core spanner).

This talk summarizes [3], which introduces SpLog (short for *spanner logic* and *substring logic*), a fragment of $\mathsf{EC}^{\mathsf{reg}}$ that has exactly the same expressive power as core spanners.

2. SpLog and the Existential Theory of Concatenation

2.1. The Existential Theory of Concatenation with Regular Constraints

Before we define SpLog, we first define $\mathsf{EC}^{\mathsf{reg}}$. A *pattern* is a word $\alpha \in (\Sigma \cup \Xi)^*$, and a *word equation* is a pair of patterns (η_L, η_R), which can also be written as $\eta_L = \eta_R$. A *pattern substitution* (or just *substitution)* is a morphism $\sigma\colon (\Xi \cup \Sigma)^* \to \Sigma^*$ with $\sigma(a) = a$ for all $a \in \Sigma$. Recall that a morphism from a free monoid A^* to a free monoid B^* is a function $h\colon A^* \to B^*$ such that $h(x \cdot y) = h(x) \cdot h(y)$ for all $x, y \in A^*$. Hence, in order to define h, it suffices to define $h(x)$ for all $x \in A$. Therefore, we can uniquely define a pattern substitution σ by defining $\sigma(x)$ for each $x \in \Xi$.

[A]supported by Deutsche Forschungsgemeinschaft (DFG) under grant FR 3551/1-1.

A substitution σ is a *solution* of a word equation (η_L, η_R) if $\sigma(\eta_L) = \sigma(\eta_R)$. The set of all variables in a pattern α is denoted by $\mathsf{var}(\alpha)$. We extend this to word equations $\eta = (\eta_L, \eta_R)$ by $\mathsf{var}(\eta) := \mathsf{var}(\eta_L) \cup \mathsf{var}(\eta_R)$.

The *existential theory of concatenation* EC is obtained by combining word equations with $\wedge$, $\vee$, and existential quantification over variables. Formally, every word equation η is an EC-formula, and $\sigma \models \eta$ if σ is a solution of η. If φ_1 and φ_2 are EC-formulas, so are $\varphi_\wedge := (\varphi_1 \wedge \varphi_2)$ and $\varphi_\vee := (\varphi_1 \vee \varphi_2)$, with $\sigma \models \varphi_\wedge$ if $\sigma \models \varphi_1$ and $\sigma \models \varphi_2$; and $\sigma \models \varphi_\vee$ if $\sigma \models \varphi_1$ or $\sigma \models \varphi_2$. Finally, for every EC-formula φ and every $x \in \Xi$, $\psi := (\exists x\colon \varphi)$ is an EC-formula, and $\sigma \models \psi$ if there exists a $w \in \Sigma^*$ such that $\sigma_{[x \to w]} \models \varphi$, where the substitution $\sigma_{[x \to w]}$ is defined by $\sigma_{[x \to w]}(y) := w$ if $y = x$, and $\sigma_{[x \to w]}(y) := \sigma(y)$ if $y \neq x$.

We also consider the *existential theory of concatenation with regular constraints*, $\mathsf{EC}^{\mathsf{reg}}$. In addition to word equations, $\mathsf{EC}^{\mathsf{reg}}$-formulas can use constraints $\mathsf{C}_A(x)$, where $x \in \Xi$ is a variable, A is an NFA, and $\sigma \models \mathsf{C}_A(x)$ if $\sigma(x) \in \mathcal{L}(A)$. As every regular expression can be directly converted into an equivalent NFA, we also allow constraints $\mathsf{C}_\alpha(x)$ that use regular expressions instead of NFAs.

The set $\mathsf{free}(\varphi)$ of *free variables* of an $\mathsf{EC}^{\mathsf{reg}}$-formula φ is defined by $\mathsf{free}(\eta) = \mathsf{var}(\eta)$, $\mathsf{free}(\varphi_1 \wedge \varphi_2) := \mathsf{free}(\varphi_1 \vee \varphi_2) := \mathsf{free}(\varphi_1) \cup \mathsf{free}(\varphi_2)$, and $\mathsf{free}(\exists x\colon \varphi) := \mathsf{free}(\varphi) - \{x\}$. Finally, we define $\mathsf{free}(C) = \emptyset$ for every constraint C. We also use $\varphi(x_1, \ldots, x_k)$ to denote $\mathsf{free}(\varphi) = \{x_1, \ldots, x_k\}$.

Example 2.1 *Consider the* EC*-formula* $\varphi_1(x,y,z) := \exists \hat{x}, \hat{y}\colon (x = z\hat{x} \wedge y = z\hat{y})$ *and the* $\mathsf{EC}^{\mathsf{reg}}$*-formula* $\varphi_1(x,y,z) := \exists \hat{x}, \hat{y}\colon (x = z\hat{x} \wedge y = z\hat{y} \wedge \mathsf{C}_{\Sigma^+}(z))$ *. Then* $\sigma \models \varphi_1$ *if and only if* $\sigma(x)$ *and* $\sigma(y)$ *have* $\sigma(z)$ *as common prefix. If, in addition to this,* $\sigma(z) \neq \varepsilon$*, then* $\sigma \models \varphi_2$*.*

Every EC-formula can be converted into a single word equation (cf. Karhumäki, Mignosi, and Plandowski [5]), and every $\mathsf{EC}^{\mathsf{reg}}$-formula into a single word equation with rational constraints (cf. Diekert [1]). For conjunctions, the construction is easily explained: Choose distinct letters $\mathtt{a}, \mathtt{b} \in \Sigma$. Hmelevskii's pattern pairing function is defined by $\langle \alpha, \beta \rangle := \alpha\mathtt{a}\beta\mathtt{a}\alpha\mathtt{b}\beta\mathtt{b}$. Then $(\alpha_L = \alpha_R) \wedge (\beta_L = \beta_R)$ holds if and only if $\langle \alpha_L, \beta_L \rangle = \langle \alpha_R, \beta_R \rangle$. The construction for disjunctions is similar, but more involved (and, in general, converting a formula with alternating disjunctions and conjunctions leads to an exponential size increase).

Satisfiability for $\mathsf{EC}^{\mathsf{reg}}$ is PSPACE-complete; but even for EC, showing the upper bound is by no means trivial (cf. [1]).

2.2. SpLog: A Logic for Spanners

As shown by Freydenberger and Holldack [4], every core spanner can be converted into an $\mathsf{EC}^{\mathsf{reg}}$-formula, and every word equation with regular constraints (and, hence, every $\mathsf{EC}^{\mathsf{reg}}$-formula) can be converted onto a core spanner. While the latter results in a spanner that is satisfiable if and only if the formula is satisfiable, the input word of the spanner needs to encode the whole word equation. Hence, the spanner can only simulate satisfiability, but not evaluation in a stricter sense. To overcome this problem, [3] introduces SpLog (short for *spanner logic*), a fragment of $\mathsf{EC}^{\mathsf{reg}}$ that directly corresponds to core spanners:

Definition 2.2 *A formula* $\varphi \in \mathsf{EC}$ *is called* safe *if the following two conditions are met:*

1. *If* $(\varphi_1 \vee \varphi_2)$ *is a subformula of* φ*, then* $\mathsf{free}(\varphi_1) = \mathsf{free}(\varphi_2)$.
2. *Every constraint* $\mathsf{C}_A(x)$ *occurs only as part of a subformula* $(\psi \wedge \mathsf{C}_A(x))$*, with* $x \in \mathsf{free}(\psi)$.

Let $W \in \Xi$*. The set of all SpLog-*formulas with main variable W, $\mathsf{SpLog}(\mathsf{W})$*, is the set of all safe* $\varphi \in \mathsf{EC}^{\mathsf{reg}}$ *such that*

1. *all word equations in* φ *are of the form* $\mathsf{W} = \eta_R$*, with* $\eta_R \in ((\Xi - \{\mathsf{W}\}) \cup \Sigma)^*$,
2. *for every subformula* ψ *of* φ*,* $\mathsf{W} \in \mathsf{free}(\psi)$.

We also define the set of all SpLog-formulas by $\mathsf{SpLog} := \bigcup_{W \in \Xi} \mathsf{SpLog}(\mathsf{W})$*, and we use* SpLog_{rx} *to denote the fragment of SpLog that exclusively defines constraints with regular expressions instead of NFAs.*

Less formally, for every $\varphi \in \mathsf{SpLog}(\mathsf{W})$, the main variable W appears on the left side of every equation (and is never bound with a quantifier). The requirement that φ is safe ensures that each variable corresponds to a substring of W. When declaring the free variables of a SpLog-formula, we slightly diverge from our convention for $\mathsf{EC}^{\mathsf{reg}}$-formulas, and write $\varphi(\mathsf{W}; x_1, \ldots, x_k)$ to denote a formula with main variable W, and $\mathsf{free}(\varphi) = \{\mathsf{W}, x_1, \ldots, x_k\}$.

Example 2.3 *Let* $\varphi_1(\mathsf{W}; x) := \exists y, z_1, z_2 \colon (\mathsf{W} = yy \wedge \mathsf{W} = z_1 x z_2 \wedge \mathsf{C}_{\Sigma+}(x))$*. Then* φ_1 *is a* $\mathsf{SpLog}(\mathsf{W})$*-formula, and* $\sigma \models \varphi_1$ *iff.* $\sigma(\mathsf{W})$ *is a square and contains* $\sigma(x)$ *as a nonempty substring. In contrast to this,* $\varphi_2(\mathsf{W}; x, y) := (\mathsf{W} = xx \vee \mathsf{W} = yyy)$ *is not a SpLog-formula, as it is not safe (intuitively, if e. g.* $\sigma(\mathsf{W}) = \sigma(x)^2$*, then* $\sigma \models \varphi_2$*, even if* $\sigma(y)$ *is not a substring of* $\sigma(\mathsf{W})$*).*

Further examples for SpLog-formulas can be found in [3], which also contains various results that make the syntax less restricted. While this is not part of the paper, SpLog can directly be generalized to multiple main variables. Then it can be understood as the fragment of $\mathsf{EC}^{\mathsf{reg}}$ that expresses all formulas without using additional space (i. e., everything has to be encoded in the content of the main variables).

It is currently open whether SpLog has the same expressive power as $\mathsf{EC}^{\mathsf{reg}}$ (the author conjectures that this is not the case, but constructing an appropriate example proved to be surprisingly hard), but we can observe a difference in the complexity of the two classes: While satisfiability is PSPACE-complete for SpLog and for $\mathsf{EC}^{\mathsf{reg}}$, the evaluation problem (given σ and φ, decide whether $\sigma \models \varphi$) is PSPACE-complete for $\mathsf{EC}^{\mathsf{reg}}$, but only NP-complete for SpLog.

3. Main Results

The main result of [3] can be summarized as follows:

Theorem 3.1 *There are polynomial time transformations in both directions between*

- *automata based core spanners and SpLog,*
- *regular expression based core spanners and* SpLog_{rx}*.*

For technical reasons, the conversion from automata based spanners to SpLog is only correct on non-empty inputs. But as spanners can be seen as a general form of search, losing the special case of searching in the empty word does not make a big difference.

This result has various consequences: It shows that SpLog can be used to reason about core spanners, even regarding matters of complexity. As a consequence, SpLog can be used to define selections for core spanners that use other relations than string equality. Furthermore, by using the fact that all bounded regular languages are expressible in EC, it is possible to adapt some of the EC-inexpressibility results by Karhumäki, Mignosi, and Plandowski [5] to SpLog (and, hence, to core spanners).

References

[1] V. DIEKERT, Makanin's Algorithm. In: M. LOTHAIRE (ed.), *Algebraic Combinatorics on Words*. chapter 12, Cambridge University Press, 2002.

[2] R. FAGIN, B. KIMELFELD, F. REISS, S. VANSUMMEREN, Document Spanners: A Formal Approach to Information Extraction. *J. ACM* **62** (2015) 2, 12.

[3] D. D. FREYDENBERGER, A Logic for Document Spanners. In: *Proc. ICDT 2017*. 2017. Accepted. Available at `http://ddfy.de/publications/F-ALfDS.html`.

[4] D. D. FREYDENBERGER, M. HOLLDACK, Document Spanners: From Expressive Power to Decision Problems. In: *Proc. ICDT 2016*. 2016.

[5] J. KARHUMÄKI, F. MIGNOSI, W. PLANDOWSKI, The expressibility of languages and relations by word equations. *J. ACM* **47** (2000) 3, 483–505.

THEORIE-TAGE 2016 F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 91–91.

Model-theoretic Characterization of Counting Classes

Anselm Haak[(A)]

[(A)]Institut für theoretische Informatik, Leibniz Universität Hannover
haak@thi.uni-hannover.de

The field of descripitve complexity was started with Fagin's theorem. It aims for model-theoretic characterizations of complexity classes in order to better understand their structural properties and possibly show new results by using the connection between the two in either direction. Such characterizations were already accomplished for many classes of decision problems. The situation is quite different concerning counting clasess: The class #P of all functions counting accepting computation paths of NP machines can be characterized as the class of functions counting satisfying assignments to free function variables within FO-forumlae, namely #FO, but no other characterizations are known. #P being a quite powerful class, we tried to change this situation by looking for a characterization of a class at the low end of the spectrum. It was shown by Immerman that $\mathrm{AC}^0 = \mathrm{FO}$. By adepting the proof of this result to the counting world, we showed that $\#\mathrm{AC}^0 = \#\text{Skolem-FO}$, where $\#\mathrm{AC}^0$ is the class of all functions computable by arithmetic circuits of constant depth and polynomial size, using $+$- and $\times$-gates and working with binary inputs (and their negations) and #Skolem-FO is the class of all functions counting Skolem functions of FO-formulae. As a corollary this result also yields a new characterization of the class TC^0.

When comparing the characterizations of #P and $\#\mathrm{AC}^0$ one can observe a direct connection: Both classes contain functions counting assignments to free function variables in FO-formulae. The difference is, that for #Skolem-FO only formulae arising from Skolemization can be used. In fact, this restriction is a simple syntactic restriction, giving a nice connection between the two characterizations. Also, as of yet only a variant of the class #FO was studied, which we call $\#\mathrm{FO}^{\mathrm{rel}}$. This class is defined as the class of functions counting satisfying assignments to free relational variables of FO-formulae and also captures #P. Saluja et al studied the alternation hierarchy below it. Motivated by the fact that #Skolem-FO fits so well in this framework, we did the same for the class #FO. We also studied the hierarchy's connections to #Skolem-FO, proving the following:

$$\#\Sigma_0 \subsetneq \#\text{Skolem-FO} = \#\Pi_1^{\mathrm{prefix}} \subsetneq \#\Pi_1 = \#\mathrm{FO} = \#\mathrm{P}.$$
$$\#\Sigma_0 \subsetneq \#\Sigma_1 \subsetneq$$

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 93–96.

Hanf Normal Form for First-Order Logic with Unary Counting Quantifiers

Lucas Heimberg[(A)] Dietrich Kuske[(B)] Nicole Schweikardt[(A)]

[(A)]Humboldt-Universität zu Berlin

[(B)]Technische Universität Ilmenau

Abstract

We study the existence of Hanf normal forms for extensions FO($\mathbf{Q}$) of first-order logic by some set of unary counting quantifiers $\mathsf{Q} \subseteq \mathbb{N}$. A formula is in Hanf normal form if it is a Boolean combination of formulas $\zeta(\overline{x})$ describing the isomorphism type of a local neighbourhood around its free variables $\overline{x}$ and statements of the form "the number of witnesses y of $\psi(y)$ belongs to $(\mathsf{Q}{+}k)$" where $\mathsf{Q} \in \mathbf{Q}$, $k \in \mathbb{N}$, and ψ describes the isomorphism type of a local neighbourhood around its unique free variable y.

We show that all formulas from FO($\mathbf{Q}$) permit a formula in Hanf normal form that is equivalent on all structures of degree $\leq d$ if, and only if, all counting quantifiers from $\mathbf{Q}$ are ultimately periodic. This transformation can be carried out in worst-case optimal 3-fold exponential time.

In particular, this yields an algorithmic version of Nurmonen's extension of Hanf's theorem for first-order logic with modulo-counting quantifiers. As an immediate consequence, we obtain that on finite structures of degree $\leq d$, model checking of first-order logic with modulo-counting quantifiers, and, more generally, first-order logic with ultimately periodic quantifiers, is fixed-parameter tractable.

1. Introduction

Two elements of a given graph are indistinguishable by first-order formulas whenever some automorphism maps the first to the second. More generally, if the two elements have isomorphic neighbourhoods of radius 4^q, then they cannot be distinguished by first-order formulas of quantifier depth q. This and similar phenomena are summarised under the slogan "first-order logic can only express local properties" and formalised by the theorems by Hanf, by Gaifman, and by Schwentick and Barthelmann [7, 4, 6, 15]. All these results give rise to normal forms for first-order formulas. Hanf's and Gaifman's theorem have found various applications in algorithms and complexity (cf., e.g., [10, 11]). In particular, there are very general algorithmic meta-theorems stating that first-order model checking is fixed-parameter tractable for various classes of structures [16, 5], and that the results of first-order queries against various classes of databases can be enumerated with constant delay after a linear-time preprocessing phase [3, 9, 17]. In the context of such algorithms, questions about the efficiency of the normal forms have recently attracted interest (cf. e.g., [2, 12, 1]).

Notions of locality have also been developed for extensions of first-order logic, and they have found application in proving inexpressibility results for these logics (cf., e.g., [8, 14, 11]). When restricting attention to classes of finite structures of bounded degree, these locality notions also give rise to normal forms for the respective logics. Let us focus on the particular case of Hanf-locality:

Hanf's locality theorem for first-order logic implies that for every first-order sentence ϕ over a relational signature σ, and for every degree bound $d \in \mathbb{N}$, there exists a first-order sentence ψ that is equivalent to ϕ on all σ-structures of degree $\leq d$, such that ψ is a Boolean combination of statements of the form "the number of elements whose r-neighbourhood has isomorphism type τ is $\geq k$". Such a sentence ψ is said to be in *Hanf normal form*. A worst-case optimal algorithm for constructing ψ when given ϕ and d has been developed in [1].

In [14], Nurmonen extended Hanf's locality theorem to the extension of first-order logic by modulo-counting quantifiers D_p (for positive integers p), where a formula of the form $\mathsf{D}_p\, y\, \psi(\overline{x}, y)$ states that the number of witnesses y for $\psi(\overline{x}, y)$ is divisible by p. As an easy consequence of Nurmonen's theorem, one obtains that for every sentence ϕ of first-order logic with modulo-counting quantifiers, and for every degree bound $d \in \mathbb{N}$ there exists a first-order sentence with modulo-counting quantifiers ψ that is equivalent to ϕ on all finite structures of degree $\leq d$, such that ψ is a Boolean combination of statements of the form "the number of elements whose r-neighbourhood has isomorphism type τ is congruent k modulo p" and statements of the form "the number of elements whose r-neighbourhood has isomorphism type τ is $\geq k$". Again, we say that ψ is in *Hanf normal form*.

For algorithmic applications, an effective procedure for computing ψ when given ϕ and d, would be highly desirable (cf., e.g., the use of Nurmonen's theorem in the full version of [13]). The proof of [14], however, does *not* lead to such an effective procedure. The two main questions which started the research whose results are presented in this paper are the following.

(1) Is there an algorithmic version of Nurmonen's result?

(2) For which classes of unary counting quantifiers does an analogue of Nurmonen's result hold?

Answering Question (2), our first main result provides a precise characterisation: A class $\mathbf{Q}$ of unary counting quantifiers permits "*Hanf normal forms*" (analogous to the ones obtained from Nurmonen's result) if, and only if, all counting quantifiers in $\mathbf{Q}$ are ultimately periodic.

Answering Question (1), our second main result provides an algorithm which, when given a degree bound d and a formula ϕ of the extension of first-order logic with ultimately periodic unary counting quantifiers, transforms ϕ into a corresponding "Hanf normal form" ψ which is equivalent to ϕ on all structures of degree $\leq d$. This algorithm uses 3-fold exponential time for $d \geq 3$ and 2-fold exponential time for $d = 2$, and is worst-case optimal in both cases.

Furthermore, ongoing work by Kuske and Schweikardt shows that "weak Hanf normal forms" exist for all classes of unary counting quantifiers $\mathbf{Q}$. Here, a formula is in weak Hanf normal form if it is a Boolean combination of sentences of the form "the number of elements x whose r-neighbourhood satisfies the first-order formula $\beta(x)$ belongs to Q" (where Q is a quantifier from $\mathbf{Q}$) and formulas of the form "the r-neighbourhood of the free variables $\overline{x}$ satiesfies the first-order formula $\gamma(\overline{x})$". Even more, these weak Hanf normal forms can be computed effectively, although the complexity increases by at least one exponent (from 3-fold to 4-fold exponential).

As an easy application of our algorithm, we obtain that Seese and Frick/Grohe's [16, 5] fixed-parameter tractability result for first-order model checking (where the degree serves as parameter) can be generalised to first-order logic with unary counting quantifiers.

References

[1] B. BOLLIG, D. KUSKE, An optimal construction of Hanf sentences. *J. Applied Logic* **10** (2012) 2, 179–186.
http://dx.doi.org/10.1016/j.jal.2012.01.002

[2] A. DAWAR, M. GROHE, S. KREUTZER, N. SCHWEIKARDT, Model Theory Makes Formulas Large. In: *Proc. ICALP'07*. 2007, 1076–1088. Full version available as preprint NI07003-LAA, Isaac Newton Institute of Mathematical Sciences (2007).

[3] A. DURAND, E. GRANDJEAN, First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.* **8** (2007) 4.

[4] R. FAGIN, L. STOCKMEYER, M. VARDI, On Monadic NP vs. Monadic co-NP. *Inf. and Comp.* **120** (1995) 1, 78–92.

[5] M. FRICK, M. GROHE, Deciding first-order properties of locally tree-decomposable structures. *J. ACM* **48** (2001) 6, 1184–1206.

[6] H. GAIFMAN, On local and non-local properties. In: J. STERN (ed.), *Proceedings of the Herbrand Symposium, Logic Colloquium '81*. North Holland, 1982, 105–135.

[7] W. HANF, Model-theoretic methods in the study of elementary logic. In: J. ADDISON, L. HENKIN, A. TARSKI (eds.), *The Theory of Models*. North Holland, 1965, 132–145.

[8] L. HELLA, L. LIBKIN, J. NURMONEN, Notions of Locality and Their Logical Characterizations over Finite Models. *J. Symb. Log.* **64** (1999) 4, 1751–1773.
http://dx.doi.org/10.2307/2586810

[9] W. KAZANA, L. SEGOUFIN, First-order query evaluation on structures of bounded degree. *Logical Methods in Computer Science* **7** (2011) 2.
http://dx.doi.org/10.2168/LMCS-7(2:20)2011

[10] S. KREUTZER, Algorithmic Meta-Theorems. In: *Finite and Algorithmic Model Theory*. Cambridge University Press, 2011. London Mathematical Society Lecture Notes, No. 379.

[11] L. LIBKIN, *Elements of Finite Model Theory*. Springer, 2004.

[12] S. LINDELL, A normal form for first-order logic over doubly-linked data structures. *Int. J. Found. Comp. Sci.* **19** (2008) 1, 205–217.

[13] F. NEVEN, N. SCHWEIKARDT, F. SERVAIS, T. TAN, Distributed Streaming with Finite Memory. In: *Proc. 18th International Conference on Database Theory (ICDT 2015)*. 2015, 324–341.

[14] J. NURMONEN, Counting Modulo Quantifiers on Finite Structures. *Inf. Comput.* **160** (2000) 1-2, 62–87.
http://dx.doi.org/10.1006/inco.1999.2842

[15] T. SCHWENTICK, K. BARTELMANN, Local normal forms for first-order logic with applications to games and automata. *Discrete Mathematics and Computer Science* **3** (1999), 109–124.

[16] D. SEESE, Linear Time Computable Problems and First-Order Descriptions. *Math. Struc. in Comp. Sci.* **6** (1996) 6, 505–526.

[17] L. SEGOUFIN, A glimpse on constant delay enumeration (Invited Talk). In: *Proc. 31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*. 2014, 13–27. http://dx.doi.org/10.4230/LIPIcs.STACS.2014.13

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 97–97.

The Relative Succinctness and Expressiveness of Modal Logics Can Be Arbitrarily Complex

Henning Schnoor

Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany
henning.schnoor@email.uni-kiel.de

We study the relative succinctness and expressiveness of modal logics, and prove that these relationships can be as complex as any countable partial order. For this, we use two uniform formalisms to define modal operators, and obtain results on succinctness and expressiveness in these two settings. Our proofs are based on formula size games introduced by Adler and Immerman and bisimulations.

THEORIE-TAGE 2016
F. Manea, D. Nowotka, Th. Wilke (Hrsg.): Theorietage, Tannenfelde, 4. – 7.10.2016
Christian-Albrechts-Universität zu Kiel, Technical Report, pp. 99–99.

Past, Present, and Infinite Future

Thomas Wilke

Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany
thomas.wilke@email.uni-kiel.de

I was supposed to deliver one of the speeches at Wolfgang Thomas's retirement ceremony. Wolfgang had called me on the phone earlier and posed some questions about temporal logic, but I hadn't had good answers at the time. What I decided to do at the ceremony was to take up the conversation again and show how it could have evolved if only I had put more effort into answering his questions. Here is the imaginary conversation with Wolfgang. The contributions are:

1. the first direct translation from counter-free ω-automata into future temporal formulas,
2. a definition of bimachines for ω-words,
3. a translation from arbitrary temporal formulas (including both, future and past operators) into counter-free ω-bimachines, and
4. an automata-based proof of separation: every arbitrary temporal formula is equivalent to a boolean combination of pure future, present, and pure past formulas when interpreted in ω-words.