24. Theorietag
„Automaten und Formale Sprachen“

Caputh, 23. – 25. September 2014

Henning Bordihn, Bianca Truthe (Hrsg.)

Titel:
24. Theorietag „Automaten und Formale Sprachen“,
Caputh, 23. – 25.09.2014, Tagungsband

Reihe:
Preprint, TR-2014-2
ISSN 0946-7580
Veröffentlicht durch die
Universität Potsdam
Institut für Informatik und Computational Science
A.-Bebel-Str. 89
14482 Potsdam

Herausgegeben von
Henning Bordihn und Bianca Truthe, 2014

Vorwort

Der Theorietag ist die Jahrestagung der Fachgruppe *Automaten und Formale Sprachen* der Gesellschaft für Informatik und wird seit 1991 von Mitgliedern der Fachgruppe abwechselnd und an wechselnden Orten in Deutschland, Österreich und Tschechien veranstaltet. Seit dem Jahr 1996 wird der Theorietag von einem eintägigen Workshop mit eingeladenen Vorträgen begleitet. Im Laufe des Theorietags findet auch die jährliche Fachgruppensitzung statt.

Die bisherigen Theorietage wurden in Magdeburg (1991), in Kiel (1992), auf Schloß Dagstuhl (1993), in Herrsching bei München (1994 und 2003), auf Schloß Rauischholzhausen (1995), in Cunnersdorf in der Sächsischen Schweiz (1996), in Barnstorf bei Bremen (1997), in Riveris bei Trier (1998), in Schauenburg-Elmshagen bei Kassel (1999), in Wien (2000 und 2006), in Wendgräben bei Magdeburg (2001), in der Lutherstadt Wittenberg (2002 und 2009), in Caputh bei Potsdam (2004), in Lauterbad bei Freudenstadt (2005), in Leipzig (2007), in Wettenberg-Launsbach bei Gießen (2008), in Baunatal bei Kassel (2010), in Allrode im Harz (2011), in Prag (2012) und in Ilmenau (2013) ausgerichtet.

Der diesjährige Theorietag wird nach 2004 wieder von Mitgleidern der Fachgruppe organisiert, die der Universität Potsdam angehören. Er findet mit dem vorangehenden Workshop vom 23. bis 25. September 2014 im Hotel Müllerhof in Caputh am Templiner See bei Potsdam statt. Auf dem Workshop tragen

- Suna Bensch aus Umeå (Schweden),
- Falk Howar mit einem Beitrag von Bernhard Steffen aus Dortmund,
- Alexander Koller aus Potsdam und
- György Vaszil aus Debrecen (Ungarn)

vor. Auf dem Programm des Theorietags stehen 20 weitere Vorträge. Der vorliegende Tagungsband enthält Kurzfassungen aller 24 Beiträge. Die Teilnehmer kommen aus Österreich, Rumänien, Schweden, Ungarn und Deutschland.

Wir danken der Gesellschaft für Informatik und der Universität Potsdam für die Unterstützung dieses Theorietags. Ein besonderer Dank gilt Alexandra Roy für die organisatorische Unterstützung bei der Vorbereitung des Theorietags und des Workshops. Wir wünschen allen Teilnehmern eine interessante und anregende Tagung sowie einen angenehmen Aufenthalt in Caputh.

Potsdam, im September 2014 Henning Bordihn und Bianca Truthe

Inhalt

BEGLEITENDER WORKSHOP

SUNA BENSCH, FRANK DREWES, HELMUT JÜRGENSEN, BRINK VAN DER MERWE:
Graph Transformation for Incremental Natural Language Analysis 7

ALEXANDER KOLLER:
Parsing Expressive Grammar Formalisms with Tree Automata 9

BERNHARD STEFFEN:
Active Automata Learning: From DFA to Interface Programs and Beyond 13

GYÖRGY VASZIL:
On the Descriptional Complexity of Grammars with Certain Types of Regulation 15

THEORIETAG „AUTOMATEN UND FORMALE SPRACHEN“

ARTIOM ALHAZOV, BOGDAN AMAN, RUDOLF FREUND, GHEORGHE PĂUN:
P Systems with Anti-Matter 19

STEPHAN BARTH:
Minimization Beyond Myhill-Nerode 23

HENNING BORDIHN, PAOLO BOTTONI, ANNA LABELLA, VICTOR MITRANA:
Solving 2D-Pattern Matching with Networks of Picture Processors 27

JÜRGEN DASSOW:
Conditional Lindenmayer Systems with Conditions Defined by Bounded Resources 29

JENS-D. DOLL:
Anforderungen an ein formales Esperanto II 33

HENNING FERNAU:
Gesteuerte Baumautomaten 35

HENNING FERNAU, KLAUS-JÖRN LANGE, KLAUS REINHARDT:
Principal Trios, AFLs and Logarithmic Space 39

HENNING FERNAU, MEENAKSHI PARAMASIVAN, MARKUS L. SCHMID:
Remarks on Jumping Finite Automata 45

DOMINIK D. FREYDENBERGER, MARIO HOLLDACK:
Abschnittsanfragen und formale Sprachen 49

STEFAN HOFFMANN:
Topologische Verfeinerungen des Cantor-Raumes 2^ω 53

MARKUS HOLZER, SEBASTIAN JAKOBI, MATTHIAS WENDLANDT:
On the Computational Complexity of Partial Word Automata Problems 57

MARTIN HUSCHENBETT, DIETRICH KUSKE, GEORG ZETZSCHE:
The Monoid of Queue Actions - A New (?) Monoid Worth to be Studied (?) - 61

NATASHA JANOSKA, FLORIN MANEA, SHINNOSUKE SEKI:
News on the Square Conjecture 63

CHRIS KÖCHER:
Analyse der Entscheidbarkeit diverser Probleme in automatischen Graphen 65

MARTIN KUTRIB, ANDREAS MALCHER, MATTHIAS WENDLANDT:
Deterministic Set Automata 69

CHRISTOPH MATHEJA:
Reconciling Decidability of Separation Logic Entailment and Graph Grammar Language Inclusion 73

FRIEDRICH OTTO:
On the Descriptional Complexity of Deterministic Ordered Restarting Automata 77

PHILIPP SCHLAG:
Mittels Antimirovs Ableitungen zu gewichteten Baumautomaten 81

MARKUS L. SCHMID:
Characterising REGEX Languages by Regular Languages with Factor-Referencing 85

BIANCA TRUTHE:
A Relation Between Definite and Ordered Finite Automata 89

THEORIE-TAG 2014

H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, p. 7.

Graph Transformation for Incremental Natural Language Analysis

Suna Bensch[(A)] Frank Drewes[(A)] Helmut Jürgensen[(B)]
Brink van der Merwe[(C)]

[(A)]Department of Computing Science, Umeå University, Sweden
{suna,drewes}@cs.umu.se

[(B)]Department of Computer Science, Western University, London, Canada
hjj@csd.uwo.ca

[(C)]Department of Computer Science, Stellenbosch University, South Africa
abvdm@cs.sun.ac.za

Abstract

Millstream systems have been proposed as a non-hierarchical method for modelling natural language. Millstream configurations represent and connect multiple structural aspects of sentences. We present a method by which the Millstream configurations corresponding to a sentence are constructed. The *construction* is incremental, that is, it proceeds as the sentence is being read and is complete when the end of the sentence is reached. It is based on graph transformations and a lexicon which associates words with graph transformation rules that implement the incremental construction process. Our main result states that, for an effectively nonterminal-bounded reader $\mathcal{R}$ and a Millstream system M based on monadic second-order logic, the correctness of $\mathcal{R}$ with respect to M can be checked: it is decidable whether all graphs generated by $\mathcal{R}$ belong to the language of configurations specified by M.

THEORIE-TAG 2014
H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 9–11.

Parsing Expressive Grammar Formalisms with Tree Automata

Alexander Koller

Department of Linguistics
University of Potsdam
koller@ling.uni-potsdam.de

1. Background

One of the great success stories in computational linguistics is the availability of accurate statistical parsers based on probabilistic context-free grammars (PCFGs). State-of-the-art PCFG parsers can parse about ten English sentences per second and get over 90% of the syntactic phrases right [2, 10].

Nonetheless, researchers have recently become interested in grammar formalisms that are more expressive than PCFGs. On the one hand, many natural languages (e.g., German) have freer word order than English, which makes using PCFGs inconvenient and motivates the use of mildly context-sensitive grammar formalism such as tree-adjoining grammars (TAG) [6] and linear context-free rewrite systems (LCFRS) [7]. On the other hand, recent syntax-based approaches to statistical machine translation have explored the use of synchronous context-free grammars (SCFGs) [3] and of tree-to-string transducers [5], which relate strings in one language with strings or syntax trees in another. Finally, *semantic parsing* of sentences into graph-based representations of sentence meanings has lately received much attention (see e.g. [4]).

However, parsing algorithms for these more expressive grammar formalisms tend to be much slower and more brittle than those for PCFGs. This is only partially due to the fact that they have an asymptotically higher parsing complexity than context-free grammars. The more subtle problem is that PCFG has been the mainstream algorithm for decades, whereas each individual expressive formalism is only used by a few scattered research groups, which cannot invest the resources necessary to develop efficient implementations that scale to real-world inputs.

2. Interpreted Regular Tree Grammars

Over the past few years, my collaborators and I have developed *interpreted regular tree grammars (IRTGs)* [8, 9]. An IRTG $\mathcal{G} = (\mathcal{B}, \mathcal{A}_1, \ldots, \mathcal{A}_n)$ consists of an n-place *bimorphism* $\mathcal{B}$ along with n algebras $\mathcal{A}_1, \ldots \mathcal{A}_n$ over the signatures $\Delta_1, \ldots, \Delta_n$ (see the top box in Fig. 1). An n-place bimorphism (in the sense of this definition) consists of a regular tree grammar $\mathbb{G}$ over some

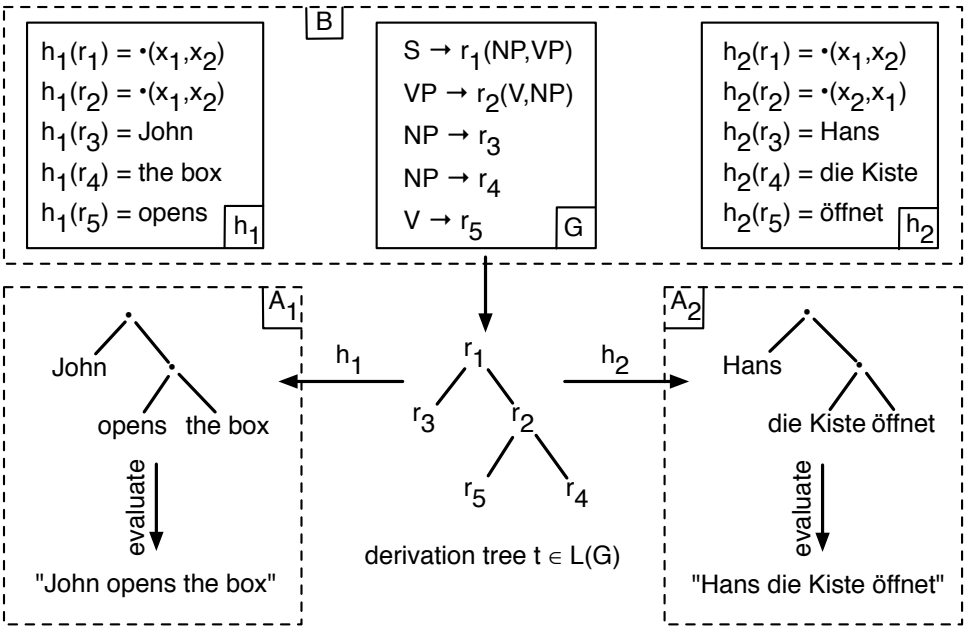


Figure 1: An IRTG with an example derivation.

signature Σ together with n tree homomorphisms $h_i : T_\Sigma \to T_{\Delta_i}$; the ordinary bimorphisms of [1] are the special case of $n = 2$.

We take the *language* $L(\mathcal{G})$ of an IRTG $\mathcal{G}$ to be the set

$$L(\mathcal{G}) = \{([\![h_1(t)]\!]_{\mathcal{A}_1}, \ldots, [\![h_n(t)]\!]_{\mathcal{A}_n}) \mid t \in L(\mathbb{G})\} \subseteq \mathcal{A}_1 \times \ldots \times \mathcal{A}_n,$$

where $[\![\cdot]\!]_{\mathcal{A}}$ is the evaluation of terms in the algebra $\mathcal{A}$. The intuition is illustrated in Fig. 1; the bimorphism $\mathcal{B}$ (with $n = 2$) is shown at the top, and the terms are interpreted over the string algebra with binary concatenation $\bullet$. The pair ("John opens the box", "Hans die Kiste öffnet") is generated by first generating a tree t from the language $L(\mathbb{G})$ of the RTG. This derivation tree t is then mapped by the homomorphism h_1 to the tree $h_1(t)$ shown at the lower left; it evaluates to the string "John opens the box" in the string algebra. At the same time, the homomorphism h_2 maps t to the tree $h_2(t)$ shown at the lower right. This term evaluates to the string "Hans die Kiste öffnet".

IRTGs over the string algebra with binary concatenation describe precisely the context-free languages; thus PCFGs are the special case where we have a single string algebra and a weighted RTG. We can extend this by using more powerful string algebras; for instance, we showed in [9] how tree-adjoining grammars [6] can be described using IRTGs with a more powerful string algebra. Synchronous grammars can be modeled by using multiple interpretations (i.e., $n > 1$): the example in Fig. 1 represents an SCFG. The multiple interpretations in a synchronous IRTG need not be over the same algebra; for instance, using one string algebra and one tree algebra gives us (bottom-up) tree-to-string transducers [5].

3. Outline of the talk

In my talk, I will first present the formal foundations of IRTGs. I will then devote some time on generalized parsing algorithms for IRTGs, which exploit the closure of tree automata under inverse homomorphism and intersection. Finally I will demonstrate the practical use of IRTGs on some examples from natural language processing.

Crucially, only a small part of the parsing algorithm is specific to the algebras whose objects we want to parse; the core of the parsing algorithm is algebra-independent. At the same time, we find that many of the most important methods for fast and accurate PCFG parsing do not rely on the fact that we are using context-free grammars, but only only the fact that the language of parse trees is regular. Thus in our current research we are working on generalizing these methods to IRTG parsing. In this way, we hope to help overcome the fragmentation of the algorithmic landscape of expressive grammar formalisms by providing a robust generalized implementation in the IRTG framework, which then applies directly to all the formalisms that can be captured as IRTGs.

References

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

[2] E. CHARNIAK, M. JOHNSON, Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In: *Proceedings of the 43rd ACL*. 2005.

[3] D. CHIANG, Hierarchical Phrase-Based Translation. *Computational Linguistics* **33** (2007) 2, 201–228.

[4] D. CHIANG, J. ANDREAS, D. BAUER, K. M. HERMANN, B. JONES, K. KNIGHT, Parsing Graphs with Hyperedge Replacement Grammars. In: *Proceedings of the 51st ACL*. 2013.

[5] J. GRAEHL, K. KNIGHT, J. MAY, Training Tree Transducers. *Computational Linguistics* **34** (2008) 3, 391–427.

[6] A. K. JOSHI, Y. SCHABES, Tree-Adjoining Grammars. In: *Handbook of Formal Languages*. Springer, 1997.

[7] L. KALLMEYER, *Parsing Beyond Context-Free Grammars*. Springer, 2010.

[8] A. KOLLER, M. KUHLMANN, A Generalized View on Parsing and Translation. In: *Proceedings of the 12th IWPT*. 2011, 2–13.

[9] A. KOLLER, M. KUHLMANN, Decomposing TAG Algorithms Using Simple Algebraizations. In: *Proceedings of the 11th TAG+ Workshop*. 2012, 135–143.

[10] J. K. KUMMERFELD, D. HALL, J. R. CURRAN, D. KLEIN, Parser Showdown at the Wall Street Corral: An Empirical Investigation of Error Types in Parser Output. In: *Proceedings of EMNLP*. Jeju Island, South Korea, 2012.

THEORIE-AG 2014

H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 13–14.

Active Automata Learning: From DFA to Interface Programs and Beyond

Bernhard Steffen

TU Dortmund, Germany
steffen@cs.tu-dortmund.de

Abstract

In this talk, we will review the development of active learning in the last decade under the perspective of treating data. Formal foundation for this development is the repeated adaption or extension of the Nerode relation for legular languages to new classes of languages.

1. Overview

Web services or other third party or legacy software components which come without code and/or appropriate documentation, are intrinsically tied to the increasingly popular orchestration-based development style of service-oriented solutions. (Active) automata learning has shown to be a powerful means to overcome the major drawback of these components, their inherent black box character. The success story began a decade ago, when its application led to major improvements in the context of regression testing [4, 3]. Since then, the technology has undergone an impressive development, in particular concerning the aspect of practical application.

Conceptually, many automata learning algorithms are based on the construction of canonical acceptors for regular languages using the Nerode relation [7]. The challenge for learning algorithms is that the Nerode relation can only be approximated from a finite number of sample words. Characteristic for *active* automata learning algorithms is a scenario in which the learning algorithm can pick sample words (in contrast to working on a given sample). These algorithms typically alternate iteratively between a sampling phase for completing the transition relation of the model aggregated from the observed behavior, and an equivalence checking phase, which either signals success or provides a counterexample, i.e., a word from the symmetric difference of the language accepted by the current conjectured model and the language to be learned. This technique, which originally has been introduced for dealing with formal languages, works very well also for reactive systems, whenever the chosen interpretation of the stimuli and reactions leads to a deterministic language. For such systems, active automata learning can be regarded as *regular extrapolation*, i.e., as a technique to construct the "best" regular model being consistent with the observations made.

In this talk, we will review the development of active learning in the last decade under the perspective of treating data, a major source of undecidability, and therefore the problem with the

highest potential for tailored, application-specific solutions. In the first practical applications of active learning, data was typically simply ignored. We will sketch the way form the first primitive treatment of data to the state of the art, where data are treated as first class citizens in models like the so-called Register Automata [1, 5, 2]. These models are able to faithfully represent *interface programs*, i.e., programs describing the typical protocol of interaction with components and services. In particular, we will discuss

- Mealy Machines as a model for systems explicitly distinguishing input and output (data),
- *automated alphabet abstraction refinement* as a two-dimensional extension of the partition-refinement based approach of active learning for inferring not only states but also optimal alphabet abstractions, and
- Register Mealy Machines, which can be regarded as programs restricted to data-independent data processing as it is typical for protocols or interface programs.

The formal foundation for each of these three steps is an adaption or extension of the Nerode relation to a new class of languages. We provide a detailed account of the development (w.r.t. data) of active automata learning algorithms in [6].

References

[1] S. CASSEL, F. HOWAR, B. JONSSON, M. MERTEN, B. STEFFEN, A Succinct Canonical Register Automaton Model. In: T. BULTAN, P.-A. HSIUNG (eds.), *Proc. ATVA'11*. LNCS 6996, Springer Berlin Heidelberg, 2011, 366–380.
`http://dx.doi.org/10.1007/978-3-642-24372-1_26`

[2] S. CASSEL, F. HOWAR, B. JONSSON, B. STEFFEN, Learning Extended Finite State Machines. In: *SEFM*. 2014, 250–264.

[3] A. HAGERER, H. HUNGAR, O. NIESE, B. STEFFEN, Model generation by moderated regular extrapolation. In: *FASE '02*. LNCS 2306, Springer, 2002, 80–95.

[4] A. HAGERER, T. MARGARIA, O. NIESE, B. STEFFEN, G. BRUNE, H.-D. IDE, Efficient regression testing of CTI-systems: Testing a complex call-center solution. *Annual review of communication, Int.Engineering Consortium (IEC)* **55** (2001), 1033–1040.

[5] F. HOWAR, B. STEFFEN, B. JONSSON, S. CASSEL, Inferring Canonical Register Automata. In: V. KUNCAK, A. RYBALCHENKO (eds.), *Verification, Model Checking, and Abstract Interpretation*. Lecture Notes in Computer Science 7148, Springer Berlin Heidelberg, 2012, 251–266.
`http://dx.doi.org/10.1007/978-3-642-27940-9_17`

[6] M. ISBERNER, F. HOWAR, B. STEFFEN, Learning register automata: from languages to program structures. *Machine Learning* **96** (2014) 1-2, 65–98.

[7] A. NERODE, Linear Automaton Transformations. *Proceedings of the American Mathematical Society* **9** (1958) 4, 541–544.

THEORIE-TAG 2014

H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 15–18.

On the Descriptional Complexity of Grammars with Certain Types of Regulation

György Vaszil

Department of Computer Science, Faculty of Informatics, University of Debrecen
Kassai út 26, 4028 Debrecen, Hungary
vaszil.gyorgy@inf.unideb.hu

The term *descriptional complexity* denotes an area of theoretical computer science studying various measures of complexity of grammars, automata, or related system (measuring the succintness of their descriptions) and the relationships, trade-offs, between the different variants of systems for a given measure, or the different variants of measures for a given system.

Descriptional complexity aspects of systems (automata, grammars, rewriting systems, etc.) have been a subject of intensive research since the beginning of computer science, but the field has also been actively studied in recent years. Examples of some early results of the type we are interested in, appeared in [10] about the size of context-free grammars, and the size measures of the number of nonterminal symbols and the number of productions were also introduced, see also [13] for a survey of "grammatical complexity" of context-free grammars. The succintness of representations of languages by different variants of automata were also considered by several authors, see [7] and [11] for more details, and a survey of results in these areas.

It is clear that the fact that a system is able to simulate some universal device implies that its size parameters can be bounded. This holds, since by simulating the universal device, all computations are carried out by a fixed (universal) system (having, therefore, fixed size parameters). On the other hand, it is still interesting to look for the best possible values of the bounds, or to study the relationship of certain size parameters with each other or with other properties of the given system.

In this talk we consider some variants of regulated grammars, that is, context-free grammars with some additional control mechanism to regulate the use of the rules during the derivations.

The need for rewriting devices which use rules of a simple form but still have a considerable generative power is justified by the study of phenomena occurring in different areas of mathematics, linguistics, or even developmental biology. To study problems in these areas which cannot be described by the capabilities of context-free languages, it is often desirable to construct generative mechanisms which have as many context-free-like properties as possible, but are also able to describe the non-context-free features of the specific problem in question. See [3] for a discussion about non-context-free phenomena in different areas, and also [4] for regulated rewriting in general.

This work was supported in part by the Hungarian Scientific Research Fund, "OTKA", grant no. K75952, and by the European Union through the TÁMOP-4.2.2.C-11/1/KONV-2012-0001 project, co-financed by the European Social Fund.

Tree controlled grammars were introduced in [2] as a pair (G, R) where G is a context-free grammar and R is a regular set, called the control language. The control language contains words composed of the terminal and nonterminal alphabets of G, and it is used to control the work of G by restricting the set of derivations which G is allowed to make. Only those words belong to the generated language of the tree controlled grammar which can be generated by the context-free grammar G, and moreover, which have a derivation tree where all the strings obtained by reading from left to right the symbols labeling nodes which belong to the same level of the tree (with the exception of the last level) are elements of the regular set R.

As it was already shown in [2], tree controlled grammars are able to generate any recursively enumerable language. Variants of the notion with control sets which are not regular but belong to different classes from the Chomsky hierarchy were studied in [17], the power of subregular control sets were examined in more detail in [5].

We also consider a different way of using the sentential form to control the rule application. It is based on the presence/absence of certain symbols or substrings in the sentential forms. One variant of this general idea is realized by conditional grammars, sometimes also called grammars with regular restriction, see [6], [19]. The derivations in these mechanisms are controlled by regular languages associated to the context-free rules: A rule can only be applied to the sentential form if it belongs to the language associated to the given rule.

A weaker restriction is used in the generative mechanisms called semi-conditional grammars, see [12], [18]. These are context-free grammars with a permitting and a forbidding condition associated to each production. The conditions are given in the form of two words: a permitting and a forbidding word. A production can only be used on a given sentential form if the permitting word is a subword of the sentential form and the forbidding word is not a subword of the sentential form. Note that semi-conditional grammars are special cases of conditional grammars, since the sets of those sentential forms which satisfy the conditions associated to the rules are regular sets. Semi-conditional grammars are also known to generate the class of recursively enumerable languages.

The study of semi-conditional grammars continued with the introduction of *simple semi-conditional grammars* in [16]. We speak of a simple semi-conditional grammar if each rule has at most one nonempty condition, that is, no controlling context condition at all, or either a permitting, or a forbidding one. Simple semi-conditional grammars were introduced in [16] where they were also were shown to be able to generate all recursively enumerable languages.

Finally, we will study *scattered context grammars*. Scattered context grammars, introduced in [9], also represent a class of generative devices which use the presence/absence of certain symbols or substrings in their current sentential forms to achieve additional control over the application of their rewriting rules.

The productions of these grammars are ordered sequences of context-free rewriting rules which have to be applied in parallel on nonterminals appearing in the sentential form in the same order as the nonterminals on the left-hand sides of the rules appear in the production sequence. Scattered context grammars are known to characterize recursively enumerable languages, see [8], [15], [14], and [22].

Natural measures of descriptional complexity for a formal grammar (regulated or not) are the number of nonterminal symbols and the number of productions (rewriting rules). In addition to measuring the complexity of the grammar, studying measures which also take into account the complexity of the control mechanism is also of interest.

Concerning tree controlled grammars we will study the number of nonterminals of the underlying grammar plus the minmal number of nonterminals necessary to generate the control language which is used to regulate the derivations. In simple semi-conditional grammars, we investigate the number of productions and the size of the context conditions used for controlling the rule application. Finally, for scattered-context grammars, we consider the number of nonterminal symbols and the number of context sensing priductions.

The presented results are from [21], [20], [1].

References

[1] E. CSUHAJ-VARJÚ, G. VASZIL, Scattered context grammars generate any recursively enumerable language with two nonterminals. *Information Processing Letters* **110** (2010) 20, 902–907.

[2] K. ČULIK II, H. MAURER, Tree controlled grammars. *Computing* **19** (1977), 129–139.

[3] J. DASSOW, G. PĂUN, *Regulated Rewriting in Formal Language Theory*. Springer, Berlin, 1989.

[4] J. DASSOW, G. PĂUN, A. SALOMAA, Grammars with controlled derivations. In: A. SALOMAA, G. ROZENBERG (eds.), *Handbook of Formal Languages*. II, Springer, Berlin Heidelberg, 1997, 101–154.

[5] J. DASSOW, R. STIEBE, B. TRUTHE, Generative capacity of subregularly tree controlled grammars. *International Journal of Foundations of Computer Science* **21** (2010).

[6] I. FRĬS, Grammars with partial ordering of rules. *Information and Control* **12** (1968), 415–425.

[7] J. GOLDSTINE, M. KAPPES, C. M. R. KINTALA, H. LEUNG, A. MALCHER, D. WOTSCHKE, Descriptional complexity of machines with limited resources,. *Journal of Universal Computer Science* **8** (2002) 2, 193–234.

[8] J. GONCZAROWSKI, M. K. WARMUTH, Scattered and context-sensitive rewriting. *Acta Informatica* **20** (1989), 391–411.

[9] S. A. GREIBACH, J. E. HOPCROFT, Scattered context grammars. *Journal of Computer and System Sciences* **3** (1969), 232–247.

[10] J. GRUSKA, Some classifications of context-free languages,. *Information and Control* **14** (1969), 152–179.

[11] M. HOLZER, M. KUTRIB, Descriptional complexity - An introductory survey. In: C. MARTÍN-VIDE (ed.), *Scientific Applications of Language Methods*. Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory 2, Imperial College Press, London, 2011, 1–58.

[12] J. KELEMEN, Conditional grammars: Motivations, definitions, and some properties. In: *Proc. Conf. Automata, Languages and Math. Sciences, Salgótarján*. 1984, 110–123.

[13] A. KELEMENOVÁ, Grammatical complexity of context-free languages and normal forms of context-free grammars. In: P. MIKULECKÝ (ed.), *Second Czechoslovak-Soviet Conference of Young Computer Scientists, Smolenice Castle, November 8-12, 1982*. Bratislava, 1982, 239–258.

[14] A. MEDUNA, Syntactic complexity of scattered context grammars. *Acta Informatica* **32** (1995), 285–298.

[15] A. MEDUNA, A trivial method of characterizing the family of recursively enumerable languages by scattered context grammars. *Bulletin of the EATCS* **56** (1995), 104–106.

[16] A. MEDUNA, A. GOPALARATNAM, On semi-conditional grammars with productions having either forbidding or permitting conditions. *Acta Cybernetica* **11** (1994), 307–323.

[17] G. PĂUN, On the generative capacity of tree controlled grammars. *Computing* **21** (1979), 213–220.

[18] G. PĂUN, A variant of random context grammars: Semi-conditional grammars. *Theoretical Computer Science* **41** (1985), 1–17.

[19] A. SALOMAA, *Formal Languages*. Academic Press, New York, 1973.

[20] G. VASZIL, On the descriptional complexity of some rewriting mechanisms regulated by context conditions. *Theoretical Computer Science* **330** (2005) 2, 361–373.

[21] G. VASZIL, On the Nonterminal Complexity of Tree Controlled Grammars. In: H. BORDIHN, M. KUTRIB, B. TRUTHE (eds.), *Languages Alive*. Lecture Notes in Computer Science 7300, Springer, Berlin Heidelberg, 2012, 265–272.

[22] V. VIRKKUNEN, On scattered context grammars. *Acta Universitatis Ouluensis Scientiae Rerum Naturalium Series A, Mathematica* **6** (1973), 75–82.

THEORIE-AG 2014

H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 19–22.

P Systems with Anti-Matter

Artiom Alhazov[(A)] Bogdan Aman[(B)] Rudolf Freund[(C)]
Gheorghe Păun[(D)]

[(A)] Academy of Sciences of Moldova, Chişinău, Moldova, artiom@math.md

[(B)] Romanian Academy, Iaşi, Romania, bogdan.aman@iit.academiaromana-is.ro

[(C)] Vienna University of Technology, Vienna, Austria, rudi@emcc.at

[(D)] Romanian Academy, Bucharest, Romania, gpaun@us.es

Abstract

The concept of a matter object being annihilated when meeting its corresponding anti-matter object is investigated in the context of P systems. Computational completeness can be obtained with using only non-cooperative rules besides these matter/anti-matter annihilation rules if these annihilation rules have priority over the other rules; without this priority condition, in addition catalytic rules with one single catalyst are needed. In the accepting case, even deterministic systems are possible. Allowing anti-matter objects as input and/or output, we even get a computationally complete computing model for computations on integer numbers. Interpreting sequences of symbols taken in from and/or sent out to the environment as strings, we get a model for computations on strings.

1. Catalytic P Systems

For details of formal language theory the reader is referred to handbooks in this area as [9]. For an overview on *membrane systems* (*P systems*) we refer to [8], for actual news and results to the P systems webpage [10].

A *catalytic P system* is a construct $\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, l_{in}, l_{out})$ where O is the alphabet of objects, C is the set of catalysts, μ is the membrane structure (with m membranes), $w_1, \ldots, w_m$ 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 evolution rules, associated with the regions of μ, l_{in} is the label of the membrane region where the inputs are put at the beginning of a computation, and l_{out} indicates the region from which the outputs are taken; l_{out}/l_{in} being 0 indicates that the output/input is taken from the environment.

The *evolution rules* are multiset rewriting rules of the form $u \to v$, where $u \in O^*$ and $v = (b_1, tar_1) \ldots (b_k, tar_k)$ with $b_i \in O^*$ and the targets $tar_i \in \{here, out, in\}$. If a rule $u \to v$ has $|u| > 1$, then it is called *cooperative* (abbreviated coo); otherwise, it is called *non-cooperative* (abbreviated $ncoo$); a *catalytic rule* is of the form $ca \to cv$, where c is a *catalyst* – a special object that never evolves and never passes through a membrane. In a *purely catalytic P system*

only catalytic rules are allowed. The evolution rules are used in the non-deterministic maximally parallel way, i.e., in any computation step of Π a multiset of rules is chosen from the sets $R_1, \ldots, R_m$ in such a way that no further rule can be added to it so that the obtained multiset would still be applicable to the existing objects in the membrane regions $1, \ldots, m$. A *configuration* of a system is given by the objects present in the membrane regions of the system. Starting from a given *initial configuration*, by applying evolution rules as described above, we get *transitions* among configurations; a sequence of transitions forms a computation; it is *halting* if it reaches a configuration where no rule can be applied any more.

In the *generative case*, the result of a halting computation is the number of objects present in membrane l_{out}, in the *accepting case*, for $l_{in} \neq 0$, we accept all (vectors of) non-negative integers whose input, given in membrane l_{in}, leads to a halting computation. For the input being taken from the environment (i.e., for *P automata*, see [2]), for $l_{in} = 0$, we use the target indication $come$; $(a, come)$ means that the object a is taken into the skin from the environment (all objects there are available in an unbounded number). The set of non-negative integers and the set of (Parikh) vectors of non-negative integers generated/accepted/accepted in the automaton way as results of halting computations in Π are denoted by $N_\delta(\Pi)$ and $Ps_\delta(\Pi)$, respectively, with $\delta \in \{gen, acc, aut\}$. A P system Π can also be considered as a system computing a partial recursive function (in the deterministic case) or even a partial recursive relation (in the non-deterministic case), with the input being given in a membrane region $l_{in} \neq 0$ as in the accepting case or being taken from the environment as in the automaton case. The corresponding functions/relations computed by halting computations in Π are denoted by $ZY_\alpha(\Pi)$, $Z \in \{Fun, Rel\}$, $Y \in \{N, Ps\}$, $\alpha \in \{acc, aut\}$. The family of sets $Y_\delta(\Pi)$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$, computed by (purely) catalytic P systems with at most m membranes and at most k catalysts is denoted by $Y_\delta OP_m(cat_k)$ ($Y_\delta OP_m(pcat_k)$).

Theorem 1.1 *(see [4]) For any $m \geq 1$, $d \geq 1$, $\delta \in \{gen, aut\}$,*

$$Ps_{acc}OP_m(cat_{d+2}) = Ps_{acc}OP_m(pcat_{d+3}) = N^dRE.$$
$$Ps_\delta OP_m(cat_2) = Ps_\delta OP_m(pcat_3) = PsRE.$$

It is a long-time open problem how to characterize the families of sets of (vectors of) natural numbers generated by (purely) catalytic P systems with only one (two) catalysts, except with using additional control mechanisms, e.g., see Chapter 4 in [8], [3], [5], and [6].

In this paper we are going to investigate the power of using matter/antimatter annihilation rules $aa^- \to \lambda$, i.e., if a matter object a meets its anti-object *(anti-matter)* a^-, they annihilate each other. The idea of anti-matter has already been considered in spiking neural P systems with *anti-spikes* as anti-matter, see [7]. Full proofs for the results described in this extended abstract on P systems with anti-matter can be found in [1].

2. Using Matter and Anti-Matter

For each object a we consider the anti-matter object a^-; related matter a and anti-matter a^- annihilate each other according to the application of the (non-context-free!) rule $aa^- \to \lambda$. We now consider catalytic P systems extended by also allowing for annihilation rules $aa^- \to$

λ, first with these rules having weak priority over all other rules, then without this priority. The family of sets $Y_\delta(\Pi)$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$, and the family of functions/relations $ZY_\alpha(\Pi)$, $Z \in \{Fun, Rel\}$, $\alpha \in \{acc, aut\}$, computed by such extended P systems with at most m membranes and k catalysts is denoted by $Y_\delta OP_m(cat(k), antim/pri)$ and $ZY_\alpha OP_m(cat(k), antim/pri)$; we omit $/pri$ for the families without priorities. In the accepting case, *detacc* indicates deterministic systems.

Theorem 2.1 *For any $n \geq 1$, $k \geq 0$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$, $\alpha \in \{acc, aut\}$, and $Z \in \{Fun, Rel\}$,*

$$\begin{aligned} Y_\delta OP_n(cat(k), antim/pri) &= YRE \quad \textit{and} \\ ZY_\alpha OP_n(cat(k), antim/pri) &= ZYRE; \\ Y_\delta OP_n(cat(k+1), antim) &= YRE \quad \textit{and} \\ ZY_\alpha OP_n(cat(k+1), antim) &= ZYRE; \\ Y_{detacc} OP_n(cat(k), antim/pri) &= YRE \quad \textit{and} \\ FunY_{detacc} OP_n(cat(k), antim/pri) &= FunYRE. \end{aligned}$$

Computing with Integers

For an alphabet $V = \{a_1, \cdots, a_d\}$, a (finite) multiset over V, as a mapping $f : V \longrightarrow \mathbb{N}$, can be extended to a mapping $f : V \longrightarrow \mathbb{Z}$ now also allowing for negative values; such an extended multiset can be described by a string over the (ordered) alphabet $\langle a_1, {a_1}^-, \cdots, a_d, {a_d}^- \rangle$ as ${a_1}^{f(a_1)} \cdots {a_d}^{f(a_d)}$ such that $(a_i)^{-m}$, $m > 0$, is represented by $({a_i}^-)^m$, $1 \leq i \leq d$. Matter and related anti-matter cannot be present in the same string or multiset over the alphabet $\{a_1, {a_1}^-, \cdots, a_d, {a_d}^-\}$; there is a one-to-one correspondence between vectors from $\mathbb{Z}^d$ and the corresponding Parikh vectors over $\langle a_1, {a_1}^-, \cdots, a_d, {a_d}^- \rangle$, which can also be viewed as vectors over $\mathbb{N}^{2d}$: for any of these vectors $v = (v_1, v_2, \cdots, v_{2d-1}, v_{2d})$, we have either $v_{2i-1} = 0$ or $v_{2i} = 0$ (or both), for all $1 \leq i \leq d$.

In order to specify that now we are dealing with d-dimensional vectors of integer numbers, we use the notation $Ps^{\mathbb{Z}^d}$: the family of sets of integer numbers $Ps_\delta^{\mathbb{Z}^d}(\Pi)$, $\delta \in \{gen, acc, aut\}$, and the set of functions/relations $ZPs_\alpha^{\mathbb{Z}^d}(\Pi)$, $Z \in \{Fun, Rel\}$, $\alpha \in \{acc, aut\}$, computed by such P systems with at most m membranes and k catalysts is denoted by $Ps_\delta^{\mathbb{Z}^d} OP_m(cat(k), antim/pri)$ and $ZPs_\alpha^{\mathbb{Z}^d} OP_m(cat(k), antim/pri)$; we omit $/pri$ for the families without priorities. Moreover, the family of recursively enumerable sets of integer numbers is denoted by $Ps^{\mathbb{Z}^d} RE$, the corresponding functions/relations by $ZPs^{\mathbb{Z}^d} RE$.

Theorem 2.2 *For any $d \geq 1$, $n \geq 1$, $k \geq 0$, $\delta \in \{gen, acc, aut\}$, $\alpha \in \{acc, aut\}$, and $Z \in \{Fun, Rel\}$,*

$$\begin{aligned} Ps_\delta^{\mathbb{Z}^d} OP_n(cat(k), antim/pri) &= Ps^{\mathbb{Z}^d} RE \quad \textit{and} \\ ZPs_\alpha^{\mathbb{Z}^d} OP_n(cat(k), antim/pri) &= ZPs^{\mathbb{Z}^d} RE; \\ Ps_{detacc}^{\mathbb{Z}^d} OP_n(cat(k), antim/pri) &= Ps^{\mathbb{Z}^d} RE \quad \textit{and} \\ FunPs_{detacc}^{\mathbb{Z}^d} OP_n(cat(k), antim/pri) &= FunPs^{\mathbb{Z}^d} RE. \end{aligned}$$

Computing with Languages

P systems with anti-matter can also be considered as devices computing with strings – the objects sent out can be concatenated to strings over a given alphabet, and the objects taken in

during a halting computation can be assumed to form a string. The set of strings generated or accepted (in the sense of automata) by a P system with anti-matter Π is denoted by $L_\delta(\Pi)$, $\delta \in \{gen, aut\}$, the function/relation computed by Π is denoted by $ZL_{aut}(\Pi)$, $Z \in \{Fun, Rel\}$. The family of sets $L_\delta(\Pi)$, $\delta \in \{gen, aut\}$, and the family of functions/relations $ZL_{aut}(\Pi)$, $Z \in \{Fun, Rel\}$, computed by such P systems with at most m membranes and k catalysts is denoted by $L_\delta OP_m(cat(k), antim/pri)$ and $ZL_{aut}OP_m(cat(k), antim/pri)$, respectively; we omit $/pri$ for the families without priorities; $cat(0)$ is used as a synonym for $ncoo$.

Theorem 2.3 *For any $n \geq 1$, $k \geq 0$, $\delta \in \{gen, aut\}$, and $Z \in \{Fun, Rel\}$,*

$$L_\delta OP_n(cat(k), antim/pri) = RE, \qquad L_\delta OP_n(cat(k+1), antim) = RE$$
$$ZL_{aut}OP_n(cat(k), antim/pri) = ZRE, \quad ZL_{aut}OP_n(cat(k+1), antim) = ZRE.$$

Acknowledgements. Artiom Alhazov acknowledges project STCU-5384 *Models of high performance computations based on biological and quantum approaches* awarded by the Science and Technology Center in the Ukraine.

References

[1] A. ALHAZOV, B. AMAN, R. FREUND, GH. PĂUN, Matter and Anti-Matter in Membrane Systems. In: *Proceedings of the Twelfth Brainstorming Week on Membrane Computing*. 2014, 1–26.

[2] E. CSUHAJ-VARJÚ, GY. VASZIL, P Automata or Purely Communicating Accepting P Systems. In: GH. PĂUN, G. ROZENBERG, A. SALOMAA, C. ZANDRON (eds.), *Membrane Computing, International Workshop, WMC-CdeA 2002, Curtea de Argeş, Romania, August 19-23, 2002, Revised Papers*. Lecture Notes in Computer Science 2597, Springer, 2003, 219–233.

[3] R. FREUND, Purely Catalytic P Systems: Two Catalysts Can Be Sufficient for Computational Completeness. In: *CMC14 Proceedings – The 14th International Conference on Membrane Computing, Chişinău, August 20–23, 2013*. Institute of Mathematics and Computer Science, Academy of Sciences of Moldova, 2013, 153–166.

[4] R. FREUND, L. KARI, M. OSWALD, P. SOSÍK, Computationally Universal P Systems without Priorities: Two Catalysts Are Sufficient. *Theoretical Computer Science* **330** (2005), 251–266.

[5] R. FREUND, M. OSWALD, Catalytic and Purely Catalytic P Automata: Control Mechanisms for Obtaining Computational Completeness. In: *Fifth Workshop on Non-Classical Models of Automata and Applications (NCMA 2013)*. books@ocg.at 294, 2013, 133–150.

[6] R. FREUND, GH. PĂUN, How to Obtain Computational Completeness in P Systems with One Catalyst. In: *Proceedings Machines, Computations and Universality 2013, MCU 2013, Zürich, Switzerland, September 9–11, 2013*. EPTCS 128, 2013, 47–61.

[7] L. PAN, GH. PĂUN, Spiking Neural P Systems with Anti-Matter. *International Journal of Computers, Communications & Control* **4** (2009) 3, 273–282.

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

[9] G. ROZENBERG, A. SALOMAA, *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.

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

THEORIE-TAG 2014

H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 23–26.

Minimization Beyond Myhill-Nerode

Stephan Barth

Ludwig-Maximilians-Universität München, Germany
stephan.barth@ifi.lmu.de

Abstract

One of the great advantages of deterministic finite automata (DFA) compared to many other automata models is the ease of minimization by using the Myhill-Nerode theorem. On the other hand the necessary state space of DFA might be quite large compared to other models.

We propose a new variant of automata for regular languages: coarser merging finite automata (CMFA). CMFA can be up to exponential more succinct than DFA, but still have polynomial algorithms for many computational problems.

This is achieved by retaining minimization according to Myhill-Nerode, but additional allowing state merging based on an equivalence relation that is coarser than the Nerode relation.

1. Introduction

We introduce a new automata model, coarser merging finite automata (CMFA). They share similarities with pushdown automata (PDA), but describe only regular languages. CMFA need at most the same number of states as deterministic finite automata (DFA), but are up to exponential more concise.

Alike PDA, CMFA contain a stack. But only transitions that traverse from one strongly connected component (SCC) of the automaton to another can push symbols onto the stack of CMFA. Furthermore, the symbols on the stack may only be read when the last symbol of the word was read.

The specific construction of CMFA allows still for merging all states that are equivalent according to the Nerode relation[2], thus CMFA need at most as many states as the corresponding DFA.

On the other hand, CMFA can also merge states that lie within different strongly connected components, but whose languages are mutual right derivations of each other. This additional merging of states can lead up to exponential more concise automata.

CMFA gains most advantage over DFA when a lot of SCC exist in the automaton, that are identical except for which states are final. Languages of this form motivated the development of CMFA.

This research was funded by the DFG (German Research Foundation), within the Research Training Group 1480: Programm- und Modell-Analyse (PUMA)

2. Notion

Denote the right derivation of a language by $\text{rder}(u, L) := \{w \mid wu \in L\}$

3. SCC in automata identical except for final states

DFA accepting regular languages, that represent ω-regular languages in the style described in [1], $L_\$:= \{u\$v \mid uv^\omega \in L\}$ occur to have lots of SCC identical except for final states, as long as the ω-part of the original language is non-trivial.

This advised to develop a more concise automata model for languages whose DFA contain this kind of structural identical SCC.

The obvious approach to store the structure of the SCC separate and to include a bit array for every occuring SCC, just referencing the, possibly shared, structure of the SCC has two big disadvantages:

- This will never lead to exponential less states; thus this is no true counteraction to the exponential growth from the representation in [1]
- Different final state settings of the same SCC may result in different SCC when minimizing, thus this simple approach would need a quite complex minimization procedure

In contrast it proved profitable to study the mutual right derivation equivalence relation $p \overset{\text{MR}}{\sim} q := \exists u, v. L(p) = \text{rder}(u, L(q)) \wedge L(q) = \text{rder}(v, L(p))$; (u, v) is called witness of $p \overset{\text{MR}}{\sim} q$. Note that $\overset{\text{MR}}{\sim}$ is coarser than the Nerode relation.

This relation is utilized by the fact, that if we know a witness of $p \overset{\text{MR}}{\sim} q$ as well as an automaton for $L(p)$ (which is given by the states reachable from p), we can calculate an automaton for $L(q)$. If q is not reachable from p we can therefor completely represent the automaton below q by the knowledge of $p \overset{\text{MR}}{\sim} q$, and the witness (u, v).

As p is reachable from q in this case $\iff$ q is reachable from p, this condition is equivalent to p and q to be in different SCC.

If two SCC have states in $\overset{\text{MR}}{\sim}$-relation, than the structures of the SCC are identical, the reverse does not necessary hold.

4. Coarser merging finite automata (CMFA)

CMFA are specific designed to allow for minimization according Myhill-Nerode, as well as merging states in different SCC that are in mutual right derivation relation. A CMFA $\mathfrak{A} = (Q, \Sigma, \Gamma, \Delta, F, q_0)$ consists of

- A set of states Q
- A word alphabet Σ
- A stack alphabet Γ; $\Gamma \supseteq \Sigma$

- A transition and push relation $\Delta \subseteq Q \times \Gamma \times Q \times \Gamma^* \times \Gamma^*$

  Δ must hold the property that for every tuple $(p, \_, q, w, r) \in \Delta$ if p and q are in the same SCC of the graph $(Q, \{(p,q) \mid \exists a \in \Gamma \exists w, r \in \Gamma^*.(p, a, q, w, r) \in \Delta\})$ (the graph of the automaton without symbols) then $w = r = \varepsilon$.

  Furthermore for every tuple $(\_, \_, q, w, r) \in \Delta$ it has to hold that $L(q) = \text{rder}(wr, L(q))$.

  (w, r) is precisely the witness for $\text{rder}(w, q) \overset{\text{MR}}{\sim} q$

- A set of accepting states $F \subseteq Q$

- An initial state q_0

A run of a CMFA $\mathfrak{A}$ on a word w has to store all the left parts of all witnesses of the transitions used into a stack. At the end of the word, the word that is then on the stack has to be used to further traverse the automaton. It can be proven that no SCC border can be crossed in this part of the run. The finality of the state reached when the stack is empty decides whether $w \in L(\mathfrak{A})$.

Two states in $\overset{\text{MR}}{\sim}$-relation in different SCC can be merged by changing all the incoming transitions to one state into transitions into the other state with additional adding the witness to the transition.

A CMFA is said to be in stack free path normal form, if every SCC has at least one path into it that has no stack operation. Every CMFA can be brought into stack free path normal form in linear time with no additional states necessary.

One remaining problem in minimization of CMFA is that computing $\overset{\text{MR}}{\sim}$ can be up to exponential complex with the known algorithms; computing a finer relation than $\overset{\text{MR}}{\sim}$, but still coarser than the Nerode relation is possible by limiting to witnesses of constant bounded length. This can be done in linear time. While some opportunities for merging states might be missed, it still grants more minimization possibilities than DFA alone.

A deterministic CMFA with $\Sigma = \Gamma$ minimized according to the Nerode relation as well as according to the $\overset{\text{MR}}{\sim}$-relation cannot have more states than the minimal DFA. If the CMFA is in stack free path normal form, the minimization according to the $\overset{\text{MR}}{\sim}$-relation does not even have to be complete, thus indeed a polynomial time minimization exists, that guarantees for no more states than a DFA but allows for fewer states.

A DFA can be converted to a CMFA by adding $(p, a, q, \varepsilon, \varepsilon)$ for every (p, a, q) from the DFA.

Inversely the corresponding DFA can be computed by copying the states behind transitions with nonempty witnesses and applying the right derivation to the parts of the automaton behind the transition.

Emptiness and complementation of deterministic CMFA are computed identical to DFA: Check whether there are final states in the automaton and flip the finality of all states respectively.

Boolean combinations of two automata $\mathfrak{A}, \mathfrak{B}$ can be computed by using a new stack alphabet $\Gamma = \Gamma_{\mathfrak{A}} \dot\cup \Gamma_{\mathfrak{B}} \dot\cup \Sigma$. The letters of the alphabet of the other automaton always perform a transition to the same state again. All witnesses are modified to only use the symbols of the automaton they are in. The transitions are extended such that they obey symbols from Σ as well as from the alphabet of their own automaton. Afterwards the product construction can be performed.

Possibly existing witnesses are combined in any order, as the letters of one automaton does not have any effect onto the other automaton. If required the stack alphabet can be shrunk afterwards by searching for witnesses with the same effect but symbols from Σ only.

5. Example

The language $(ab)^*|(ba)^*$ needs five states as DFA, but only three as CMFA, as shown in figure 1. Furthermore the two SCC of the DFA, that are identical up to finality of the states, and can be merged in the CMFA, are surrounded by a dashed line.

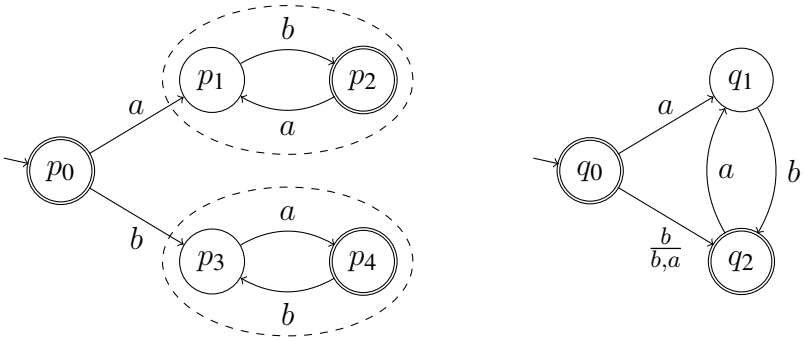


Figure 1: DFA and CMFA for $(ab)^*|(ba)^*$

References

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

[2] A. NERODE, Linear Automaton Transformations. *Proceedings of the American Mathematical Society* **9** (1958) 4, 541.
`http://dx.doi.org/10.2307/2033204`

THEORIE-AG 2014
H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 27–28.

Solving 2D-Pattern Matching with Networks of Picture Processors

Henning Bordihn[(A)] Paolo Bottoni[(B)] Anna Labella[(B)]
Victor Mitrana[(C)]

[(A)]Department of Computing Science, University of Potsdam, Germany
henning@cs.uni-potsdam.de

[(B)]Department of Computer Science, "Sapienza" University of Rome, Italy
{bottoni,labella}@di.uniroma1.it

[(C)]Faculty of Mathematics and Computer Science, University of Bucharest, Romania
mitrana@fmi.unibuc.ro

Picture languages defined by different mechanisms have been studied extensively in the literature. This work is a continuation of [1], where networks of evolutionary picture processors acting on rectangular pictures as *acceptors* are considered.

A *picture* is represented by a two-dimensional array of symbols from an alphabet V. A picture processor modifies a given picture according to a set of rules and a unique action mode that are assigned to the picture processor. A picture processor can either be deleting or substituting. The rules of a *deleting picture processor* are of the form $a \to \epsilon$, where $a \in V$ and ϵ denotes the empty word. A rule $a \to \epsilon$ indicates that a column or a row can be deleted from the picture if it contains the symbol a. The action mode determines whether only the leftmost, only the rightmost or an arbitrary column, or only the first, only the last or an arbitrary row shall be deleted. If an arbitrary column (row) can be selected and there is more than one column (row, respectively) containing the symbol a, then for each of these columns (rows) a copy of the given picture is produced having this column (row) deleted. If the column(s)/row(s) designated by the action mode do not contain the symbol a, then the given picture is left unchanged. Similarly, the rules of a *substituting picture processor* are of the form $a \to b$, $a, b \in V$, requiring that one occurrence of a is to be substituted by b in the column (row) designated by the action mode. Again, if there are several possibilities of applying the rule in accordance with the action mode, then a copy of the picture for each possibility is produced, and if the symbol a is not contained in the respective column(s)/row(s), then the picture is left unchanged. A picture processor is capable to comprise a finite set of pictures, operating on them simultaneously.

An *accepting network of evolutionary picture processors* (ANEPP) is a set of picture processors as described above that are connected with each other by communication channels. The network forms an undirected graph without loops, the picture processors being the nodes and the communication channels being the edges. Local data is then transmitted over the network following a given protocol. Only data which can pass a filtering process can be communicated. This filtering process is regulated by input and output filters (defined by some very simple context conditions) associated with each node. All the nodes simultaneously send their data to,

and receive data from, the nodes they are connected to. In a computation, those communication steps alternate with processing steps in which the picture processors modify the pictures that they contain. The computation starts with a processing step on the initial configuration, in which all nodes apart from a designated initial node are empty (contain no picture) and the initial node contains a single picture, the input to be analyzed. The computation halts, when a designated halting node contains at least one picture. The input is accepted, if another (not necessarily different) designated node, the accepting node, is not empty when the computation halts. In [1] it is shown that these networks can accept the complement of any local language, as well as languages that are not recognizable.

We consider here the pattern matching problem, which is largely motivated by different aspects in low-level image processing [2], and try to solve it in a parallel and distributed way with networks of picture processors. The network solving the problem can be informally described as follows: it consists of two subnetworks, one of them extracts simultaneously all subpictures of the same size from the input picture and sends them to the second subnetwork. In its turn, the second subnetwork consists of two subnetworks; one of them checks whether any of the received pictures is identical to the pattern, while the other one halts the computation if none of the received pictures is identical to the pattern. If the pattern is of size (k,l), with $1 \leq k \leq 3$, and $l \geq 1$, we present an efficient solution running in $\mathcal{O}(n+m+l)$ computational (processing and communication) steps, provided that the input picture is of size (n,m). Moreover, this solution can be extended at no further cost with respect to the number of computational steps to any finite set of patterns all of them of the same size.

In order to generalize our solution to the pattern matching problem, we introduce a new operation and its inverse that can convert a visible row/column into an invisible one and vice versa. The two operations which seem to be relevant with respect to picture processing (see, e.g. "zoom-in", "zoom-out") are called $mask$ and $unmask$, respectively. We show how this variant of networks of picture processors is able to solve efficiently (in $\mathcal{O}(n+m+kl+k)$ computational steps) the problem of pattern matching of an arbitrary pattern of size (k,l) in a given rectangular picture of size (n,m). Again, the solution can be extended at no further cost with respect to the number of computational steps to any finite set of patterns all of them of the same size.

References

[1] P. BOTTONI, A. LABELLA, V. MITRANA, Networks of evolutionary picture processors. *Fundamenta Informaticae* **131** (2014), 337–349.

[2] A. ROSENFELD, A. C. KAK, *Digital Picture Processing*. Academic Press, New York, 1982.

THEORIE-AG 2014

H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 29–32.

Conditional Lindenmayer Systems with Conditions Defined by Bounded Resources

Jürgen Dassow

Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik,
PSF 4120, D-39016 Magdeburg
dassow@iws.cs.uni-magdeburg.de

1. Introduction

In the theory of formal languages one imposes very often conditions to perform a step in the generation of words. By practical reasons – but also by theoretical considerations – it is very useful that one can check the condition by an efficient procedure. Thus one relates the condition to regular languages, for which the membership problem can be decided in linear time. We mention here as examples regularly controlled context-free grammars, conditional context-free grammars, tree controlled context-free grammars, and contextual grammars with selection languages (for details see [2]and [6]).

The checking of the condition given by a regular language is now very simple and efficient, however, the increase of generative power is considerable (for instance, for the first three devices, one has an increase from context-free languages to recursively enumerable languages). Since on the one hand practical requirements do not ask for arbitrary regular languages and on the other hand theoretical studies – for instance proofs – show that only special regular languages are used, it is very natural to study the devices with subregular languages for the control. Investigations on the change of the generative power, if subregular restrictions defined by combinatorial and algebraic properties are done for all the above mentioned devices. A summary of the results is given in [1].

Furthermore, one can also restrict the regular languages by requiring that they have bounded complexities. The most well-known complexity for regular languages is the state complexity which is given as the number of states of a minimal deterministic automaton that accepts the given language. It is well-known that, for any natural number n, there is a regular language with state complexity n.

Other measures were defined for context-free languages but are also of interest for regular languages. An example is the nonterminal complexity introduced by J. GRUSKA in 1967. Moreover, J. GRUSKA also proved that, for any natural number n, there is a regular language with nonterminal complexity n.

Therefore, the question arises which hierarchies are obtained if one restricts to regular languages with limited state, nonterminal, and production complexities. Results in this direction are given for all the above mentioned grammars.

The topic of this paper are conditional tabled Lindenmayer systems which were introduced in [7]. Here a table can only be applied to a sentential form if the form belongs to a regular set, which is associated with the table. In the extended case, such systems are very powerful; they can generate all recursively enumerable languages. Investigations, where one restricts to special subregular families defined by combinatorial or algebraic properties, can be found in [3] and [4].

In this paper we study the power of conditional ET0L systems if the conditions are given by regular languages with a limited state or nonterminal complexity. We show that conditions obtained by regular grammars with two nonterminals and finite automata with three states are sufficient to generate all recursively enumerable languages. Similar results are given for the generation of all context-sensitive languages. Moreover, in the non-extended case, one gets infinite hierarchies for both complexity measures.

2. Definitions and Notation

We assume that the reader is familiar with the basic concepts of the theory of formal languages and automata. In this section we only recall some notations and some definitions such that a reader can understand the results. For details, we refer to [6].

By $\mathcal{L}(\mathrm{CS})$ and $\mathcal{L}(\mathrm{RE})$ we denote the families of all context-sensitive and recursively enumerable languages, respectively.

Let $\mathcal{A} = (X, Z, z_0, F, \delta)$ be a deterministic finite automaton (with the set X of input symbols, the set Z of states, the initial state z_0, the set F of accepting states, and the transition function δ). By $T(\mathcal{A})$ we denote the language accepted by $\mathcal{A}$. We define the state complexity $s(\mathcal{A})$ of $\mathcal{A}$ as the number of states of $\mathcal{A}$, i. e., $s(\mathcal{A}) = \#(Z)$.

Let $G = (N, T, P, S)$ be a grammar (with the set N of nonterminals, the set T of terminals, the set P of rules or productions, and the start symbol S). We say that G is regular, if all rules of P have the form $A \to wB$ or $A \to w$ with $A, B \in N$ and $w \in T^*$.[1] The nonterminal complexity $n(G)$ of G is defined as the number of its nonterminals[2], i. e., $n(G) = \#(N)$.

For a regular language R, we set

$$s(L) = \min\{s(\mathcal{A}) \mid \mathcal{A} \text{ is a finite automaton }, L = T(\mathcal{A})\},$$
$$n(L) = \min\{n(G) \mid G \text{ is a regular grammar }, L = L(G)\}.$$

We now define the type of Lindenmayer systems studied in this paper.

Definition 2.1 *An extended conditional tabled Lindenmayer system without interaction (ECT0L system, for short) is a 4-tuple $H = (V, T, \mathcal{P}, w)$, where*

- *V is an alphabet, T is a subset of V,*
- *$\mathcal{P}$ is a finite set of pairs (P, R), where*
  - *P is a finite set of rules $a \to v$ with $a \in V$ and $v \in V^*$ such that, for any $b \in V$, there is a word v_b with $b \to v_b \in P_i$, and*
  - *R is a regular language over V, and*

[1] Sometimes such grammars are called right-linear, and then, for regularity, it is required that $w \in T \cup \{\lambda\}$.

[2] If regularity is defined by $w \in T \cup \{\lambda\}$, then state complexity and nonterminal complexity only differ by 1.

– $w \in V^+$.

For $x \in V^+$ and $y \in V^*$, we say that x derives y in H, written as $x \Longrightarrow_H y$, if and only if there is a pair $(P, R) \in \mathcal{P}$ such that

– $x = a_1 a_2 \ldots a_t$ *with* $a_i \in V$ *for* $1 \leq i \leq t$,

– $y = y_1 y_2 \ldots y_t$,

– $a_i \to y_i \in P$ *for* $1 \leq i \leq t$, *and*

– $x \in R$.

The language $L(H)$ generated by H is defined as $L(H) = \{z \mid z \in T^*,\ w \Longrightarrow_H^* z\}$, where $\Longrightarrow_H^*$ is the reflexive and transitive closure of $\Longrightarrow_H$.

The sets P are called the tables of the system. By definition, in a ECT0L system, a regular set R is associated with any table P, and P is only applicable to a sentential form x, if x belongs to the associated language R. Therefore, R is called the condition of P.

Let $H = (V, T, \{(P_1, R_1), (P_2, R_2), \ldots, (P_n, R_n)\}, w)$ be an ECT0L system. It is called a propagating (ECPT0L system, for short) if, for $1 \leq i \leq n$ and any $a \in V$, P_i does not contain the rule $a \to \lambda$. It is called a CT0L system if $V = T$. Furthermore, it is called an ET0L system if, for $1 \leq i \leq n$, $R_i = V^*$ (i. e., any table P_i can be apply at any moment). We can combine the restrictions; for instance, we can have propagating ET0L systems (EPT0L systems, for short).

Let $\mathcal{G} = \{$ ECT0L, ECPT0L, CT0L, CPT0L $\}$. For $X \in \mathcal{G}$, $\mathcal{L}(X)$ denotes the family of languages generated by X systems.

By $\mathcal{L}$(ET0L) and $\mathcal{L}$(EPT0L), we denote the family of all languages generated by ET0L and EPT0L systems, respectively.

It is known that

$$\mathcal{L}(\text{EPT0L}) = \mathcal{L}(\text{ET0L}) \subset \mathcal{L}(\text{ECPT0L}) = \mathcal{L}(\text{CS}) \subset \mathcal{L}(\text{ECT0L}) = \mathcal{L}(\text{RE}).$$

For $X \in \mathcal{G}$, by $\mathcal{L}(X, s, n)$ we denote the family of all languages which can be generated by X systems $H = (V, T, \{(P_1, R_1), (P_2, R_2), \ldots, (P_n, R_n)\}, w)$ with $s(R_i) \leq n$ for $1 \leq i \leq n$.

For $X \in \mathcal{G}$, by $\mathcal{L}(X, s, n)$ we denote the family of all languages which can be generated by X systems $H = (V, T, \{(P_1, R_1), (P_1, R_1), \ldots, (P_n, R_n)\}, w)$, where $n(R_i) \leq n$ for $1 \leq i \leq n$.

3. Results

The following theorem states that, for extended systems and both measures, we get finite hierarchies with

Theorem 3.1 *For any $n \geq 2$, $m \geq 3$ and $k \geq 4$,*

$$\begin{aligned}
&\mathcal{L}(\text{ET0L}) \subset \mathcal{L}(\text{ECT0L}, v, 1) \subseteq \mathcal{L}(\text{ECT0L}, v, n) = \mathcal{L}(\text{RE}),\\
&\mathcal{L}(\text{ET0L}) \subseteq \mathcal{L}(\text{ECPT0L}, v, 1) \subseteq \mathcal{L}(\text{ECT0L}, v, 2) = \mathcal{L}(\text{CS}),\\
&\mathcal{L}(\text{ET0L}) = \mathcal{L}(\text{ECT0L}, s, 1) \subseteq \mathcal{L}(\text{ECT0L}, s, 2) \subseteq \mathcal{L}(\text{ECT0L}, s, m) = \mathcal{L}(\text{RE}),\\
&\mathcal{L}(\text{ET0L}) = \mathcal{L}(\text{ECPT0L}, s, 1) \subseteq \mathcal{L}(\text{ECPT0L}, s, 2) \subseteq \mathcal{L}(\text{ECPT0L}, s, 3)\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \subseteq \mathcal{L}(\text{ECPT0L}, s, k) = \mathcal{L}(\text{CS}).
\end{aligned}$$

It is open whether our bounds are optimal, i. e., we do not know whether some of the inclusion given in Theorem 3.1 are strict.

If we consider non-extended systems (i. e., $V = T$), then the situation changes completely. We obtain infinite hierarchies.

Theorem 3.2 *Let $p_1, p_2, \ldots, p_n, \ldots$ be an infinite sequence of prime numbers with $p_{i+1} - p_i \geq 4$ for $i \geq 1$. Then*

$$\mathcal{L}(\text{CT0L}, s, p_i + 4) \subset \mathcal{L}(\text{CT0L}, s, p_{i+1} + 4) \text{ and } \mathcal{L}(\text{CPT0L}, s, p_i + 4) \subset \mathcal{L}(\text{CPT0L}, s, p_{i+1} + 4).$$

By Theorem 3.2 we obtain an infinite hierarchy with respect to the state complexity, but it remains open whether $\mathcal{L}(\text{CPT0L}, s, n) \subset \mathcal{L}(\text{CPT0L}, s, n+1)$ for $n \geq 1$.

Theorem 3.3 *Let $n_1, n_2, \ldots, n_m, \ldots$ be an infinite sequence of numbers such that $n_{i+1} - n_i \geq 3$ for $i \geq 1$. Then*

$$\mathcal{L}(\text{CT0L}, v, n_i) \subset \mathcal{L}(\text{CT0L}, v, n_{i+1}) \text{ and } \mathcal{L}(\text{CPT0L}, v, n_i) \subset \mathcal{L}(\text{CPT0L}, v, n_{i+1}).$$

We mention that it is open whether $\mathcal{L}(\text{CPT0L}), v, n) \subset \mathcal{L}(\text{CPT0L}), v, n+1)$ holds for $n \geq 1$.

References

[1] J. DASSOW, Subregular restrictions for some language generating devices. In: R. FREUND, M. HOLZER, B. TRUTHE, U. ULTES-NITSCHE (Eds.), *Fourth Workshop on Non-Classical Models of Automata and Applications* 2010, Proceedings 290 der Österreichischen Computer Gesellschaft, Austria, 2012, 11–26.

[2] J. DASSOW, GH. PĂUN, *Regulated Rrewriting in Formal Language Theory*. Springer-Verlag, Berlin, 1989.

[3] J. DASSOW, ST. RUDOLF, Conditional Lindenmayer systems with subregular conditions: the non-extended case. *RAIRO - Theor. Inf. Appl.* **48** (2014), 127–147.

[4] J. DASSOW, ST. RUDOLF, Conditional Lindenmayer systems with subregular conditions: the extended case. Submitted.

[5] G. PĂUN, *Marcus Contextual Grammars*. Kluwer Publ. House, Doordrecht, 1998.

[6] G. ROZENBERG, A. SALOMAA (Eds.), *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.

[7] G. ROZENBERG, S. H. VON SOLMS, Priorities on context conditions in rewriting systems. *Inform. Sci.* **14** (1978), 15–51.

THEORIE-TAG 2014
H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 33–34.

Anforderungen an ein formales Esperanto II

Jens-D. Doll

Universität Hamburg

Zusammenfassung

Der Folienvortrag baut auf dem Teil I aus 2013 auf und stellt die Grundlagen einer solchen Sprachklasse dar. Softwareartefakte bilden dabei die Brücke zwischen Informatik, Philosophie und Welt. Zunächst werden die maSSgeblichen Bestandteile eines wissensverarbeitenden Systems informal dargestellt und möglichst umgangssprachlich erläutert. Diese sind eine Wissenshierarchie, Methoden der Entscheidungsfindung und wissenschaftliche Begriffsbildung. Sodann werden die Bestandteile intuitiv miteinander verbunden und Betrachtungen über das entstandene Netzwerk angestellt. Als Ergebnisse werden ein Schnittstellenschema und Präformalismen vorgestellt. Aus der Betrachtung dieser Elemente ergibt sich, dass Sätze einer universellen formalen Sprache nicht nur aus sequentiellem Text bestehen können, sondern eine Verbindung zwischen mehreren Sätzen verschiedener formaler Sprachen sein müssen. Das drängt die Vorstellung von einem mehrdimensionalen Würfel auf, der schichtweise verständlich, aber im Ganzen nicht mit dem Verstand erfassbar ist. In ihren Fachsprachen benutzen Wissenschaftler schon seit Jahrhunderten Präformalismen, die sich aus Vorsilbe, Nachsilbe, Wortstamm und Wortkombination zusammensetzen. Es ist für diese Sprachklasse also eine Verbindung zwischen der Formalstruktur von Maschinen, der Bedeutung natürlicher Sprache und der Wirklichkeit zu schaffen.

1. Softwareartefakte

Die zu definierenden Softwareartefakte sollen einerseits Begriffe der praktischen und der theoretischen Informatik sein und andererseits solche der technischen Welt. Als Beispiele seien {Programm, Datenbank}, {Medium, Plattform} oder {Semantik, Ontologie} genannt. Alle Begriffe haben für die sie nutzende Gemeinde eine innere Struktur und äuSSere Verbindungen mit anderen Begriffen oder mit der realen Welt. Der Einfachheit halber werden nur Daten- und Programmartefakte betrachtet, die stellvertretend für alle anderen stehen.

2. Wissenshierarchie

In der Wissenschaftsphilosophie bzw. Wissenschaftstheorie hat sich die Vorstellung vom unterschiedlichen Wert des Wissens herausgebildet und diese ist auf den zugehörigen Gebieten von ökonomischer und struktureller Bedeutung. Als Beispiel dient Blooms Taxonomie und ihr Zusammenhang mit der Wissensverarbeitung.

3. Entscheidungsfindung

Nicht nur in der Chomsky-Hierarchie, wo die Entscheidbarkeit sich als Determinismus niederschlägt, sondern auch auf anderen Gebieten haben sich Regeln entwickelt, die gewährleisten sollen, dass gute Entscheidungen getroffen werden. Es wird verlangt, dass analog zur Wissenshierarchie eine Entscheidungsgüte gemessen werden kann, um universellen Anforderungen zu genügen.

4. Begriffsbildung

Begriffe durchlaufen wie Worte und Namen eine Entwicklung, die mit anfänglich intuitiver Unklarheit beginnt und sich durch Experimente und Zusammenarbeit mit der wissenschaftlichen Gemeinde weiterentwickelt. Begriffe sind daher niemals statisch sondern an Zeit und Gemeinschaft gekoppelt. Diese Erkenntnis verlangt eine Versionierung von Begriffen bis in die Detailebene, um zu konsistenten Aussagen zu gelangen.

5. Schnittstellenschema

Ein bestehendes allgemeines Schnittstellenschema wird von seinen technischen Bestandteilen befreit und auf breitere Grundlagen gestellt. Sein Zweck soll es einmal sein, Schnittstellen genau klassifizieren zu können, um ihnen passende Algorithmen zuzuordnen.

6. Präformalismen

Wenn man einen Blick in die Grundlagenliteratur verschiedener wissenschaftlicher Disziplinen wirft, erkennt man, dass lateinische und griechische Wörter weiterhin die Wurzeln für das Denken der Wissenschaftler sind. Der Entscheidungsprozess geht eng mit räumlichen und zeitlichen Vorstellungen einher, die sich in der Fachsprache als Vor- und Nachsilben wiederfinden. Man kombiniert genauso wie es die Römer zur Zeit Ciceros und Griechen zur Zeit Aristoteles getan haben. Im Gegensatz zu den von einer Phasendreschmaschine erzeugten sinnlosen Satzteilen haben Kombinationen aus Vorsilbe, Wortstamm und Nachsilbe an den Forschungsstätten aber fundierte Bedeutungen.

7. Würfelstruktur

Nun ist es nur noch nötig, die vorgestellten Elemente so zu verwenden, dass sie mit der geforderten universellen Sprache, dem formalen Esperanto, FE, harmonieren. Aus formalsprachlichen Erkenntnissen und praktischer Erfahrung heraus kann gesagt werden, dass ein FE nicht auf sequentiellen Text reduzierbar sein wird. Es stellen sich ohnehin Fragen nach Existenz, Beweisbarkeit und Vollständigkeit, auf welche ein Ausblick gegeben wird.

THEORIE-TAG 2014
H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 35–37.

Gesteuerte Baumautomaten

Henning Fernau

Informatikwissenschaften, FB 4, Universität Trier
fernau@uni-trier.de

Zusammenfassung

Wir führen gesteuerte reguläre Baumautomaten ein und zeigen, in welcher Weise sich manche aus dem Bereich regulärer Baumsprachen bekannte Ergebnisse übertragen. Zum Beispiel sind die Yields der Sprachen, die von durch endliche Automaten gesteuerten regulären Baumautomaten beschrieben werden, genau die Wortsprachen, die von durch endliche Automaten gesteuerten kontextfreien Sprachen beschrieben werden.

1. Einleitung

Die Theorie der Baumautomaten ist ein klassisches Gebiet der Formalen Sprachen, das zahlreiche Anwendungen im Bereich der Linguistik und der Verarbeitung semistrukturierter Daten besitzt, um nur zwei Bereiche zu benennen. Für weitere Beispiele und Einzelheiten sei auf das im Internet verfügbare Lehrbuch [2] verwiesen. Ebenso gehört die Theorie der gesteuerten Ersetzungsverfahren zu den traditionellen Gebieten der Formalen Sprachen, siehe [3]. Leider gibt es praktisch keine Arbeit, die beide Teilgebiete gemeinsam betrachtet. Das ist insbesondere deshalb bedauerlich, weil den gesteuerten Ersetzungsverfahren auf diese Weise mögliche Anwendungsgebiete verschlossen bleiben.

Wir möchten im Folgenden darstellen, dass es sehr wohl möglich ist, Querbezüge herzustellen, so wie wir es auch schon für Grammatiksysteme in [5] entwickelt haben.

2. Definitionen und Hinweise

Aus Platzgründen verzichten wir auf eine detaillierte Definition von (geordneten) Bäumen und endlichen Baumautomaten, möchten aber auf wesentliche Begriffe und Unterschiede zur wohlvertrauten Theorie der endlichen Wortautomaten hinweisen.

- Das Eingabealphabet Σ besitzt eine Stelligkeitsfunktion $\alpha : \Sigma \to \mathbb{N}$. Setze $\Sigma_n := \alpha^{-1}(n)$.
- Bäume können als Terme im Sinne der Allgemeinen Algebra verstanden werden, wobei Σ die zulässigen Operatorensymbole beinhaltet. So versammelt Σ_0 die Konstanten.
- Endliche Baumautomaten arbeiten in der üblicheren Weise Bäume von den Blättern zur Wurzel ab. Hierzu ist es hilfreich, als Blattbeschriftungen sowohl Zeichen aus Σ_0 als auch Zustände $q \in Q$ zuzulassen. Die Regeln eines Baumautomaten haben die Form

$$f(q_1, \ldots, q_n) \to q \quad \text{oder} \quad a \to q,$$

wobei $q_1, \ldots, q_n, q \in Q$ und $a \in \Sigma_0$. Dies bedeutet (für den ersten Fall), dass der Teilbaum (der Höhe 1) mit Wurzelbeschriftung f und Blattbeschriftungen $q_1, \ldots, q_n$ durch den (kleineren) Baum q ersetzt wird. Ein Baum wird akzeptiert, wenn auf diese Weise schließlich ein Baum mit einem Knoten erreicht wird, der mit einem Endzustand beschriftet ist.

Man könnte nun die Arbeitsweise eines endlichen Baumautomaten (für einen konkreten Eingabebaum) dadurch beschreiben, dass man die Folge der durch Regelanwendungen neu als Blattbeschriftungen auftauchenden Zustände notiert. Man kann also zu jeder Abarbeitung eines Baumes t ein *Szilardwort* $s_t \in Q^+$ notieren; dieses muss sich natürlich nicht eindeutig aus t ergeben. Umgekehrt kann man durch Vorgabe einer Steuersprache $R \subseteq Q^*$ die Menge der zulässigen Berechnungen eines Baumautomaten dadurch einschränken, dass höchstens solche Bäume t akzeptiert werden, die ein Szilardwort s_t besitzen mit $s_t \in R$. In dieser Weise sei die Klasse der *regulär gesteuerten regulären Baumsprachen* definiert.

3. Einige Resultate zu regulär gesteuerten Baumsprachen

Wir sammeln hier eine Reihe von Resultaten ohne Beweis zusammen. Diese zeigen, dass sich viele Ergebnisse aus dem Bereich klassischer Baumsprachen recht leicht übertragen.

Satz 3.1 *Regulär gesteuerte deterministische endliche Baumautomaten können jede regulär gesteuerte reguläre Baumsprache beschreiben.*

Liest man die an den Blättern eines geordneten Baumes t notierten Konstanten von links nach rechts, so erhält man ein Wort $w = yield(t)$ (über dem Alphabet Σ_0). Entsprechend kann man zu einer Baumsprache B eine Wortsprache $yield(B)$ zuordnen. Das folgende Ergebnis verallgemeinert ein bekanntes Resultat über reguläre Baumsprachen.

Satz 3.2 *Eine Wortsprache L kann genau dann von einer regulär gesteuerten Grammatik mit kontextfreien, nicht-löschenden Kernregeln erzeugt werden, wenn es eine regulär gesteuerte reguläre Baumsprache B gibt mit $L = yield(B)$.*

Im Beweis hierzu (in Analogie zu den bekannten Überlegungen von Doner und Thatcher; wir verweisen der Kürze halber auf [2]) benötigt man zum einen eine Normalform für regulär gesteuerten Grammatiken, sodass die Nichtterminalsymbole die Definition einer Stelligkeitsfunktion gestatten, und zum anderen die Überlegung, dass die von regulär gesteuerten Grammatiken erzeugten Sprachen mit den Sprachen übereinstimmen, die von regulär gesteuerten Grammatiken akzeptiert werden, so wie dies in [1] erklärt wurde. Für dieses Ergebnis gibt es noch einen alternativen Beweisgang. Hierzu kann man sich (wie im Fall der üblichen endlichen Baumautomaten) eine Variante regulär gesteuerter Baumautomaten definieren, welche Bäume von oben nach unten verarbeiten. Da die regulären Sprachen unter Spiegelbild abgeschlossen sind, überträgt sich ein klassisches Resultat über endliche Baumautomaten wie folgt:

Satz 3.3 *Die Klasse der Baumsprachen, welche von regulär gesteuerten endlichen Baumautomaten akzeptiert werden, die Bäume von den Blättern zur Wurzel abarbeiten, stimmt mit der Klasse der Baumsprachen überein, welche von regulär gesteuerten endlichen Baumautomaten akzeptiert werden, die Bäume von der Wurzel zu den Blättern abarbeiten.*

Nun kann man wiederum Satz 3.2 in klassischer Manier beweisen, woraus sich als Folgerung die Gleichwertigkeit von generierendem und akzeptierendem Verhalten bei regulär gesteuerten kontextfreien Grammatiken ergäbe. Aus den bekannten Ergebnissen über den Wortfall kann man nun auch schlussfolgern, dass regulär gesteuerte endliche Baumautomaten nichtreguläre Baumsprachen beschreiben können.

4. Ausblick

Wir haben hier nur einen kleinen Einblick in ein laufendes Forschungsprojekt gegeben. Im Folgenden möchten wir daher kurz auf (mögliche) weitere Ergebnisse hinweisen.

– Wir untersuchen derzeit noch, wie sich genau diejenigen Baumsprachen einordnen, die von regulär gesteuerten deterministischen endlichen Baumautomaten akzeptiert werden, welche von der Wurzel zu den Blättern arbeiten. Das ist insofern interessant, als dass (bekanntermaßen) im ungesteuerten Fall für derartige Automaten Determinismus eine echte Einschränkung ihrer Mächtigkeit bedeutet.
– Neben der Steuerung durch reguläre Sprachen kann man viele weitere Steuerungsmechanismen untersuchen. Ausgehend von der Steuerung durch Graphen lassen sich fast alle bekannten Mechanismen einheitlich beschreiben und davon auch entsprechende Ergebnisse ableiten. Hier verweisen wir auf [3, 4, 6]. Diese Überlegungen zeigen, dass die hier definierten Klassen von Baumsprachen robuste Modelle mit verschiedenen Kennzeichnungen sind. Selbstverständlich kann man hierbei auch noch den Vorkommenstests betrachten.

Literatur

[1] H. BORDIHN, H. FERNAU, Accepting Grammars with Regulation. *International Journal of Computer Mathematics* **53** (1994), 1–18.

[2] H. COMON, M. DAUCHET, R. GILLERON, F. JACQUEMARD, D. LUGIEZ, C. LÖDING, S. TISON, M. TOMASSI, *Tree Automata, Techniques and Applications*. 2007. `http://tata.gforge.inria.fr/`

[3] J. DASSOW, G. PĂUN, *Regulated Rewriting in Formal Language Theory*. EATCS Monographs in Theoretical Computer Science 18, Springer, 1989.

[4] H. FERNAU, Graph-controlled grammars as language acceptors. *Journal of Automata, Languages and Combinatorics* **2** (1997) 2, 79–91.

[5] H. FERNAU, Cooperating Distributed Tree Automata. In: H. BORDIHN, M. KUTRIB, B. TRUTHE (eds.), *Languages Alive; Dassow Festschrift*. LNCS 7300, Springer, 2012, 75–85.

[6] 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.

THEORIE-TAG 2014

H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 39–43.

Principal Trios, AFLs and Logarithmic Space

Henning Fernau[(B)] Klaus-Jörn Lange[(A)] Klaus Reinhardt[(A)]

[(B)]Theoretische Informatik, Universität Trier, Universitätsring, D-54286 Trier, Germany
fernau@uni-trier.de

[(A)]Wilhelm-Schickard-Institut für Informatik, University of Tübingen, Germany
{lange,reinhardt}@informatik.uni-tuebingen.de

Abstract

We study computational (space) complexity aspects of the membership problem for several principal trios and AFLs.

1. Introduction and Definitions

A *(full) trio* is a family of languages that is closed under (erasing) homomorphisms, inverse homomorphisms and intersection with regular sets. By the famous theorem of Nivat, this is equivalent to saying that a trio is a family of languages closed under rational transductions. A *(full) semi-AFL* is a trio closed under union, and a *(full) AFL* (i.e., an *abstract family of languages*) is a semi-AFL closed under Kleene star. Notice that in the literature, mostly also closure under concatenation is required for an AFL, but this closure property follows from our requirements. The trio / (semi-)AFL *generated* by some formal language L is the smallest trio / (semi-)AFL that contains L. We write TRIO(L) and AFL(L) to denote these language families. For instance, TRIO$(\emptyset)$ = AFL$(\emptyset)$ is the family of regular languages. Conversely, a trio or AFL $\mathcal{L}$ is called *principal* if there is a language L such that $\mathcal{L} = \text{TRIO}(L)$ or $\mathcal{L} = \text{AFL}(L)$, respectively.

We will consider in this paper the following six well-known non-regular languages and the language families generated thereof.

$$S_1 = \{a^n b^n : n \in \mathbb{N}\}, \qquad D_1 = \{w \in \{a,b\}^* : \#_a w = \#_b w\},$$

$$S_1^{\neq} = \{a^n b^m : n,m \in \mathbb{N}, n \neq m\}, \qquad D_1^{\neq} = \{w \in \{a,b\}^* : \#_a w \neq \#_b w\},$$

$$S_1^{<} = \{a^n b^m : n,m \in \mathbb{N}, n < m\}, \qquad D_1^{<} = \{w \in \{a,b\}^* : \#_a w < \#_b w\}.$$

We will also study the well-known complexity classes **L** and **NL** collecting languages that can be accepted in logarithmic space by deterministic or nondeterministic machines, respectively. It is at a level of a student exercise to show that the trios generated by each of the

[(A)]The third autor is currently at Humboldt University of Berlin, Rudower Chaussee 25, 12489 Berlin

mentioned languages is contained in **NL**. We call a language family $\mathcal{L}$ **NL**-*complete* if $\mathcal{L} \subseteq$ **NL** and there exists a language $L \in \mathcal{L}$ that is **NL**-hard (under logspace reductions). We should also study the circuit class $\mathbf{NC}^1$. To be able to state $\mathbf{NC}^1$-completeness of some language class, we need weaker reductions. In this sense, it is well-known that the regular languages, i.e., TRIO($\emptyset$), is $\mathbf{NC}^1$-complete. Moreover, Sudborough has given a linear context-free language that is complete for **NL**; see [8].

The question if TRIO$(S_1) \subseteq$ **L** is related to unary knapsack and similar pseudo-polynomial problems. These were conjectured to lie somewhere inbetween **L** and **NL**, see [2], but were recently proven to lie within **L**, see [3, 5]. We employ similar techniques to show a more general claim.

2. Results

We are going to show the following result in this paper, where the first inclusion easily follows from a rational transducer which deletes at least one b (by not copying to the output) and the second inclusion easily follows from a rational transducer which simply checks (while copying to the output) that the a's come before the b's:

Theorem 2.1 TRIO$(S_1^<) \subseteq$ TRIO$(S_1) \subseteq$ TRIO$(D_1) \subseteq$ **L**.

Similarly TRIO$(S_1^{\neq}) \subseteq$ TRIO$(D_1^{\neq}) \subseteq$ TRIO$(D_1^<) \subseteq$ TRIO$(D_1) \subseteq$ **L** where the first inclusion is by a rational transducer which checks that the a's come before the b's, the second by one which nondeterministcally decides to either always swap a's and b's or not and the third by one which again deletes ≥ 1 b's.

Lemma 2.2 *Every principal trio is a semi-AFL.*

Hence, the Theorem 2.1 can be easily strengthened in its formulation towards semi-AFLs generated by the according languages.

In view of the listed results, the power of the Kleene star operation appears to be interesting. Here, the situation becomes different for the languages under study.

Theorem 2.3 TRIO(S_1^*) *is* **NL**-*complete.*

Corollary 2.4 AFL(S_1) *is* **NL**-*complete.*

Theorem 2.5 TRIO$((S_1^{\neq})^*) \subseteq$ **L**.

We claim that TRIO$((S_1^<)^*) \subseteq$ TRIO$((D_1^<\#)^*) \subseteq$ **L** and TRIO$((D_1^{\neq}\#)^*) \subseteq$ TRIO$((D_1^<\#)^*) \subseteq$ **L** again with the first inclusions by rational trasductions as above but bockwise. For the last inclusion, we use a variant of marked Kleene closure.

3. Proof of Theorem 2.1

Let $L \in \text{TRIO}(D_1)$. Hence, there is a rational transducer A whose transduction τ_A satisfies $\tau_A(L) = D_1$. We may assume that A reads (and writes) only single letters or ε. Furthermore we may regard A as a counter automaton where a stands for increment and b stands for decrement.

Case 1. First we consider the case that A has no loop reading ε which means there is an upper bound l_A for the maximum length of a word being written for a single input letter. (This might be two times the maximum length of an ε-path plus 1.)

The naive idea of an algorithm would be that, as we proceed on the input w, we keep track on the set of all possible counter values (differences of the number of the so far written a's and b's) for each state of A. Obviously this would require polynomial space. Instead, we calculate the number of paths which output a word in D_1 with a Chinese remainder representation like trick.

To do this, we use a generating function $f(x) = f_{q_e,w}(x) = \sum_{PathP} x^{\Delta_P}$ where P is a path from the start-state q_0 to the end-state q_e in A reading w and Δ_P is the difference of the number of a's and b's written on P. We will show in Lemma 3.1 which is analogous to Lemma 1 in [5] that $-\sum_{x=1}^{p-1} f(x)$ is $\pmod p$ equivalent to the number of paths which output a word in D_1. If this number of paths is not zero then there must exist a p between $n := |w| \cdot l_A$ (which bounds the length of the longest word written on a path in A reading w) and $poly(n)$ with $\sum_{x=1}^{p-1} f(x) \not\equiv 0 \pmod p$ by Chinese remainder theorem since there are at least n such primes. For that reason the following algorithm (like that in [5]) decides if there is a path which outputs a word in D_1 (that means $w \in L$):

$c := 0$; $p := n$;
repeat
  $c := c + \lfloor \log_2 p \rfloor$; $p :=$ NextPrime(p);
until $c > n$ or $\sum_{x=1}^{p-1} f(x) \not\equiv 0 \pmod p$
return $\sum_{x=1}^{p-1} f(x) \not\equiv 0 \pmod p$

The algorithm only needs logarithmic space since p and values $\bmod\ p$ are only polynomial in size. Furthermore $f(x) = f_{q_e,w}(x)$ can be calculated recursively by

$$f_{q,wc}(x) = \sum_{(q',c,u,q)\in\delta_A} x^{|u|_a - |u|_b} \cdot f_{q',w}(x)$$

starting with $f_{q_0,\varepsilon}(x) = 1$. Only a constant $|A|$ number of values have to be remembered from one position in w to the next.

Lemma 3.1 *For $p > |w| \cdot |A| + 1$ and p prime, $\sum_{x=1}^{p-1} f(x) \equiv -N \pmod p$ where N is the number of paths which output a word in D_1.*

Proof. It holds that

$$\sum_{x=1}^{p-1} f(x) \equiv \sum_{PathP} \sum_{x=1}^{p-1} x^{\Delta_P} \equiv \sum_{PathP, \Delta_P \equiv 0 \pmod{p-1}} -1 \pmod{p} \equiv -N.$$

The first equivalence follows from changing the order of the summation, the second equivalence follows from

$$\sum_{x=1}^{p-1} x^k \equiv \begin{cases} -1 \pmod{p} & \text{if } k \equiv 0 \pmod{p-1}, \\ 0 \pmod{p} & \text{else} \end{cases}$$

(see proof of Lemma 1 in [5]) and the third equivalence follows from $\Delta_P = 0 \Leftrightarrow \Delta_P \equiv 0 \pmod{p-1}$ since $p-1 > |w| \cdot |A|$. □

Case 2. Let us now consider the case that A has a loop reading ε which means there is a subset $Q' \subseteq Q$ of states of A which is strongly connected by transitions reading ε. Paths through A not using any state in Q' can be handled separately considering a transducer having only the states $Q \setminus Q'$. For all other paths we consider a transducer A' which has additional copies for each state in $Q \setminus Q'$ (including the new start-state of A' but excluding the end-state) which can only be assumed before assuming any state in Q'. This means each path in A' has to go through a state in Q'. Furthermore, we recursively make this distinction for each component strongly connected by transitions reading ε.This means we may assume in each distinguished case that each path in A' touches each component strongly connected by transitions reading ε.

Let g be the greatest common divisor of all values $|u|_a - |u|_b$ for words u written on a loop reading ε.

If there are positive and negative such values then it is possible to pump up and down by any multiple of g, thus deciding $w \in L$ can easily be done by recursively calculating for each position in w and each state the subset of possible counter-values modulo g, which requires only a constant amount of information from one position in w to the next.

In the case that all values $|u|_a - |u|_b$ for words u written on a loop reading ε are zero, we may make the following change for each connected component Q': For each pair of a non-ε transition from q to $q' \in Q'$ and each state in $q'' \in Q'$ we introduce a chain of new states and transitions from q to q'' which has the same effect on the counter as before (which must be unique because different possible values would mean that one of the loops would be not zero). Then we can remove all ε transitions inside Q'. This leads to Case 1.

In the remaining casse, we may, w.l.o.g., assume that there are positive but no negative values $|u|_a - |u|_b$ for words u written on a loop reading ε. Then there exists an $m \in \mathbb{N}$ such that for each $k \in \mathbb{N}$ each path can be pumped up increasing the counter by $(k+m)g$. Now deciding $w \in L$ can be done by recursively calculating for each position in w and each state and each $h < g$ the minimal value $\min_{q,h}$ for the counter and for each $k < m$ the information whether $\min_{q,h} + kg$ is a possible counter value. The calculation is done in the obvious way looking at the corresponding information for the previous states which will not change anymore after a constant number of repetitions in a connected component. Finally we accept $w \in L$ if $\min_{q_e,0} < -mg$ or 0 is explicitly a possible counter value in the last position in w. Since m and g are constant and $\min_{q,h}$ is linear in $|w|$, this requires only a logarithmic amount of information from one position in w to the next one.

4. Further questions

1. TRIO$(S_1^{\neq}) \subseteq \mathbf{NC}^1$?

2. TRIO$(S_1) \subseteq \mathbf{NC}^1$? Here, [4] might be helpful. Elberfeld et al. continue working on metatheorems (now for circuit complexity classes) as they already did in [3] for logspace. For instance, we think that [4, Theorem 13] implies that Unary Knapsack is in $\mathbf{TC}^0$, which is even a subclass of $\mathbf{NC}^1$. But we should also check out the language from [8]. Possibly, there is even a more general argument on the level of circuits: If L is in $\mathbf{NC}^1$, then TRIO(L) is in $\mathbf{NC}^1$, as well, as transducers are kind of $\mathbf{NC}^1$-devices.

3. AFL$((S_1^{\neq})^*) \subseteq \mathbf{L}$?

   According to Theorem 9.5.9 in [6], (also confer [1]) the smallest AFL containing a family $\mathcal{L}$ of languages coincides with the rational closure (i.e., the closure under union, Kleene star and concatenation) of the smallest trio containing $\mathcal{L}$. This might help solving this question.

4. How do our results relate to the groupoid theory apparently developed for languages between $\mathbf{NC}^1$ and $\mathbf{NL}$ by Lemieux [7]?

Aknowledement We thank Andreas Krebs and Henning Bordihn for helpful comments.

References

[1] L. BOASSON, M. NIVAT, Sur diverses familles de langages fermées par tranduction rationelle. *Acta Informatica* **2** (1973), 180–188.

[2] S. CHO, T. HUYNH, On a complexity hierarchy between L and NL. *Information Processing Letters* **29** (1988), 177–182.

[3] M. ELBERFELD, A. JAKOBY, T. TANTAU, Logspace Versions of the Theorems of Bodlaender and Courcelle. In: *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS*. IEEE Computer Society, 2010, 143–152.

[4] M. ELBERFELD, A. JAKOBY, T. TANTAU, Algorithmic Meta Theorems for Circuit Classes of Constant and Logarithmic Depth. In: C. DÜRR, T. WILKE (eds.), *29th International Symposium on Theoretical Aspects of Computer Science, STACS*. LIPIcs 14, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012, 66–77.

[5] D. M. KANE, Unary Subset-Sum is in Logspace. *CoRR* **abs/1012.1336** (2010).

[6] G. LALLEMENT, *Semigroups and combinatorial applications*. Wiley-Interscience, 1979.

[7] F. LEMIEUX, *Finite Groupoids and their Applications to Computational Complexity*. Ph.D. thesis, School of Computer Science, McGill University, Montréal, Canada, 1996.

[8] I. H. SUDBOROUGH, A Note on Tape-Bounded Complexity Classes and Linear Context-Free languages. *Journal of the ACM* **22** (1975) 4, 499–500.

THEORIE-TAG 2014
H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 45–47.

Remarks on Jumping Finite Automata

Henning Fernau Meenakshi Paramasivan Markus L. Schmid

Abteilung Informatikwissenschaften, Fachbereich IV
Universität Trier, Germany
{paramasivan,fernau,mschmid}@uni-trier.de

Abstract

We consider *(general) jumping finite automata* [5] and prove characterizations of the corresponding language class(es) which was an open problem in [5]. To this end, we define special forms of shuffle expressions.

1. Introduction

Meduna and Zemek [5] introduced (general) jumping finite automata that can be viewed as classical finite automata that do not read the input in a continuous fashion but are rather allowed to jump at any position of the input during their work. We provide characterizations of the power of these automata in terms of expressions which enables us to put them into the context of classical formal language results from around 1980.

2. Definitions and Preliminaries

The shuffle operation, shuffle closure (iterated shuffle) and shuffle expressions have been thoroughly investigated by Höpner / Jantzen & Opp and later by Jedrzejowicz & Szepietowski; here we only refer to [2, 3, 4].

Definition 2.1 *Let $u, v \in \Sigma^*$, the shuffle operation, denoted by $\sqcup\!\sqcup$, is a binary operation on words, defined by $u \sqcup\!\sqcup v = \{x_1y_1x_2y_2 \ldots x_ny_n : u = x_1x_2 \ldots x_n, v = y_1y_2 \ldots y_n, x_i, y_i \in \Sigma^*, 1 \leq i \leq n, n \geq 1\}$. The shuffle operation on languages is extended in the following way: for $L_1, L_2 \subseteq \Sigma^*$, $L_1 \sqcup\!\sqcup L_2 = \{z : z \in x \sqcup\!\sqcup y, x \in L_1, y \in L_2\}$.*

Definition 2.2 *For $L \subseteq \Sigma^*$, the shuffle closure operator is defined by:*

$$L^{\sqcup\!\sqcup,*} = \bigcup_{n=0}^{\infty} L^{\sqcup\!\sqcup,n} \text{ where } L^{\sqcup\!\sqcup,0} = \{\epsilon\} \text{ and } L^{\sqcup\!\sqcup,i} = L^{\sqcup\!\sqcup,i-1} \sqcup\!\sqcup L.$$

These operations are basic to understand the meaning of the classes of α-SHUF and of SHUF expressions.

Definition 2.3 *$\emptyset$, ϵ and each $a \in \Sigma$ are α-SHUF expressions. If S_1, S_2 are α-SHUF expressions, then $(S_1 + S_2)$, $(S_1 \sqcup\!\sqcup S_2)$ and $S_1{}^{\sqcup\!\sqcup,*}$ are α-SHUF expressions.*

The semantics of α-SHUF expressions is defined in the expected way. For instance, $L((a+b)^{\sqcup\!\sqcup,*}) = \{a,b\}^{\sqcup\!\sqcup,*}$. The corresponding class of languages was termed $\mathcal{L}_3$ in [2]. If all words over Σ form a basis for such expressions, we arrive at SHUF expressions. More generally, in shuffle expressions also concatenation and Kleene star are admitted as operators.

Permutations of words and languages are defined as usual; we write $perm(x)$ to collect all permutations of a word x.

Theorem 2.4 *a)* $perm(L_1) \sqcup\!\sqcup perm(L_2) = perm(L_1 \sqcup\!\sqcup L_2)$.
b) $(perm(L))^{\sqcup\!\sqcup,*} = perm(L^{\sqcup\!\sqcup,*}) = perm(L^*)$.

Following Meduna and Zemek, we denote a *general finite machine* like $M = (Q, \Sigma, R, s, F)$, where Q is a finite set of *states*, Σ is the *input alphabet*, $\Sigma \cap Q = \emptyset$, $R \subseteq (Q \times \Sigma^*) \times Q$ is finite, $s \in Q$ is the *start state*, and F is a set of *final states*. If $R \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$, we call M a finite machine.

We interpret M in two ways:

- As a (general) finite automaton, a *configuration* of M is any string in $Q\Sigma^*$. The binary *move relation*, symbolically denoted by $\Rightarrow$, over $Q\Sigma^*$, is defined as follows:

$$qw \Rightarrow pz \iff \exists (q,y,p) \in R \exists z \in \Sigma^* : w = yz$$

- As a (general) jumping finite automaton, a configuration of M is any string in $\Sigma^* Q \Sigma^*$. The binary *jumping* relation, symbolically denoted by $\curvearrowright$, over $\Sigma^* Q \Sigma^*$, is defined as follows:

$$vqw \curvearrowright v'pz' \iff \exists (q,y,p) \in R \exists z \in \Sigma^* : w = yz \wedge vz = v'z'$$

If M is a (generalized) finite machine, we can hence obtain the languages $L_{FA}(M) = \{w \in \Sigma^* : \exists f \in F : sw \Rightarrow^* f\}$ and $L_{JFA}(M) = \{w \in \Sigma^* : \exists u, v \in \Sigma^* \exists f \in F : w = uv \wedge usv \curvearrowright^* f\}$.

This defines us in particular the classes of JFA languages (accepted by jumping finite automata) and of GJFA languages (accepted by generalized jumping finite automata).

3. Major Results

Lemma 3.1 *Let $M = (Q, \Sigma, R, s, F)$ be a finite machine. Then, we have the following connection between both rewrite relations defined via M:*
$\forall p, q \in Q\ \forall w \in \Sigma^* (\exists\, u, v \in \Sigma^* : w = uv \text{ and } upv \curvearrowright^* q) \iff (\exists\, x \in perm(w) : px \Rightarrow^* q)$.

Lemma 3.2 *Let M be a finite machine. Then, $L_{JFA}(M) = perm(L_{FA}(M))$.*

This enables us to prove the following characterization theorem, as requested in [5].

Theorem 3.3 *Let $L \subseteq \Sigma^*$. L is a* JFA *language if and only if there exists a regular language $L' : L = perm(L')$.*

By using results from [2], we can now conclude another characterization:

Corollary 3.4 *Let $L \subseteq \Sigma^*$. L is a* JFA *language if and only if it can be described by some α-SHUF expression.*

As a consequence, we obtain the following corollary, adding to the list of closure properties given in [5].

Corollary 3.5 JFA *languages are closed under shuffle closure.*

In a similar manner, we can relate general jumping finite automata with SHUF expressions.

Theorem 3.6 *Let $L \subseteq \Sigma^*$. L is a* GJFA *language if and only if it can be described by some SHUF expression.*

4. Conclusions

We related several language classes introduced in the literature and obtained new characterizations thereof in this way. Several variants of jumping automata have been suggested in [5], and more variants can be easily thought of, like:

- Automata that process the input by jumping at the begin or at the end of it; so, the input is accepted from both ends of the word. This resembles a specific form of revolving automata, see [1].
- Further restricting the previously sketched working mode, notice that automata that strictly alternate between processing the input word at the begin or at the end would characterize the linear languages.

We are going to look for characterizations for such variants in the near future. Also, we hope to solve some (more) of the open problems mentioned by Meduna and Zemek.

References

[1] S. BENSCH, H. BORDIHN, M. HOLZER, M. KUTRIB, On input-revolving deterministic and nondeterministic finite automata. *Information and Computation* **207** (2009) 11, 1140–1155.

[2] M. HÖPNER, M. OPP, About Three Equations Classes of Languages Built Up By Shuffle Operations. In: A. W. MAZURKIEWICZ (ed.), *Mathematical Foundations of Computer Science 1976, 5th Symposium, MFCS*. LNCS 45, Springer, 1976, 337–344.

[3] M. JANTZEN, The Power of Synchronizing Operations on Strings. *Theoretical Computer Science* **14** (1981), 127–154.

[4] J. JEDRZEJOWICZ, A. SZEPIETOWSKI, Shuffle languages are in P. *Theoretical Computer Science* **250** (2001) 1-2, 31–53.

[5] A. MEDUNA, P. ZEMEK, Jumping Finite Automata. *International Journal of Foundations of Computer Science* **23** (2012) 7, 1555–1578.

THEORIE-TAG 2014

H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 49–52.

Abschnittsanfragen und formale Sprachen

Dominik D. Freydenberger[(A)] Mario Holldack[(B)]

[(A)]Goethe-Universität, Frankfurt am Main
freydenberger@em.uni-frankfurt.de

[(B)]Goethe-Universität, Frankfurt am Main
holldack@informatik.uni-frankfurt.de

Zusammenfassung

Ausgehend von einem von IBM entwickelten Informationsextraktionssystem haben Fagin et al. 2013 die sogenannten *Abschnittsanfragen* formalisiert und aus Sicht der theoretischen Informatik untersucht. Dabei können boolesche Abschnittsanfragen als ein Modell zur Definition formaler Sprachen interpretiert werden, das eine echte Oberklasse der regulären Sprachen sowie der Patternsprachen beschreibt. Diese Sprachklasse ist eng verwandt mit der Klasse der Sprachen, die durch reguläre Ausdrücke mit Rückreferenzen definiert werden. Boolesche Abschnittsanfragen können zwar nicht alle dieser Sprachen ausdrücken; allerdings können die Negativresultate aus Freydenberger 2013 und Angluin 1980 auf sie übertragen werden.

1. Einleitung

In der Informationsextraktion werden aus unstrukturierten Texten Informationen extrahiert. Ein solches Informationsextraktionssystem ist das von IBM entwickelte SystemT, welches wiederum ein Teil des Produkts IBM InfoSphere BigInsights ist[1]. Um die diesem System zugrunde liegende Anfragesprache AQL formal zu untersuchen, wurden von Fagin et al. [2] *Abschnittsanfragen* eingeführt.

Allgemein ist eine Abschnittsanfrage eine Funktion, die ein Wort $w \in \Sigma^*$ auf eine Relation über Abschnitten von w abbildet. Wir werden hier nur eine eingeschränkte Klasse dieser Anfragen betrachten, die in [2] als *core spanners* bezeichnet wird. Diese bauen auf sogenannten *Regex-Formeln* auf.

Diese sind reguläre Ausdrücke, die zusätzlich um Variablen zum Markieren von Match-Bereichen erweitert werden. Zusätzlich dazu benötigen wir noch weitere Definitionen: Sei $w \in \Sigma^*$. Wir bezeichnen die Menge $[w] := \{i \in \mathbb{N} \mid 1 \leq i \leq |w|+1\}$ als die *Positionen* von w. Durch jedes Paar natürlicher Zahlen i, j mit $1 \leq i \leq j \leq |w|+1$ wird ein *Abschnitt von* w definiert als

$$[i, j\rangle := \{k \in \mathbb{N} \mid i \leq k < j\}.$$

[1]Siehe http://www.ibm.com/software/data/infosphere/biginsights/

Dabei seien $[i,j\rangle$ und $[i',j'\rangle$ genau dann identisch, wenn $i = i'$ und $j = j'$ Ein Abschnitt von w beschreibt also nicht nur ein Teilwort von w, sondern die exakte Stelle, an der dieses Teilwort in w vorkommt. Der erste Buchstabe erhält hierbei die Position 1, und ist $w = w_1 \cdots w_n$ (mit $w_k \in \Sigma$), so entspricht der Abschnitt $[i, j+1\rangle$ dem Teilstück $w_i \cdots w_j$. Wir betrachten dazu ein Beispiel:

Beispiel 1.1 *Die Regex-Formel α über dem Terminalalphabet $\Sigma := \{\mathtt{a},\mathtt{b}\}$ und dem Variablenalphabet $V := \{x,y,z\}$ sei definiert als $\alpha := \Sigma^* \cdot z\big\{x\{\mathtt{a}\} \cdot \Sigma^* \cdot y\{\mathtt{b}\}\big\} \cdot \Sigma^*$. Außerdem sei $w := \mathtt{ababa}$. Es gilt $[w] = \{1,\ldots,6\}$. Die von α beschriebene Abschnittsanfrage $[\![\alpha]\!]$ bildet nun w auf die Relation ab, die der folgenden Tabelle zu entnehmen ist:*

x	y	z
$[1,2\rangle$	$[2,3\rangle$	$[1,3\rangle$
$[1,2\rangle$	$[4,5\rangle$	$[1,5\rangle$
$[3,4\rangle$	$[4,5\rangle$	$[3,5\rangle$

Die Regex-Formel α bildet also w auf die Relation aller Tupel (x,y,z) ab, in denen x und y ein a *bzw.* b *beschreiben, die aufeinander folgen. Dabei enthält z den kompletten Abschnitt, der zwischen diesen beiden Buchstaben aufgespannt wird. Hierbei entsprechen (zum Beispiel) die Abschnitte $[1,2\rangle$ und $[2,3\rangle$ unterschiedlichen Vorkommen von* a*. Da sie sich auf unterschiedliche Stellen von w beziehen, sind sie nicht identisch.*

Aus technischen Gründen beschränken wir uns auf funktionale Regex-Formeln: In diesen dürfen Variablen nicht unterhalb eines Kleene-Sterns vorkommen; und Variablen dürfen nur dann mehrfach vorkommen, wenn sichergestellt ist, dass sie in jedem Match genau einmal verwendet werden (insbesondere müssen sie also stets definiert sein). Somit ist $(x\{\mathtt{a}\} \mid x\{\mathtt{b}\})$ eine funktionale Regex-Formel, während $(x\{\mathtt{a}\} \cdot x\{\mathtt{a}\})$, $(x\{\mathtt{a}\} \mid \varepsilon)$ und $(x\{\mathtt{a}\})^*$ keine funktionalen Regex-Formeln sind.

Die in [2] definierten *core spanner* verwenden zusätzlich zu Regex-Formeln die folgenden Operatoren:

(a) **Vereinigung**: Sind P_1 und P_2 Abschnittsanfragen mit den gleichen Variablenmengen, so ist die Abschnittsanfrage $P_1 \cup P_2$ definiert durch $(P_1 \cup P_2)(w) = P_1(w) \cup P_2(w)$ für alle Wörter $w \in \Sigma^*$.

(b) **Projektion**: Ist P eine Abschnittsanfrage mit Variablenmenge V, so ist für jedes $X \subseteq V$ auch $\pi_X(P)$ eine Abschnittsanfrage. Dabei enthält $\pi_X(P(w))$ für jedes $w \in \Sigma^*$ die Tupel, die entstehen, indem man alle Tupel aus P auf die Variablen aus X einschränkt.

(c) **Natürlicher Join**: Sind P_1 und P_2 Abschnittsanfragen, so ist auch $P_1 \bowtie P_2$ eine Abschnittsanfrage. Diese erzeugt für jedes $w \in \Sigma^*$ die Menge aller Kombinationen von Tupeln aus $P_1(w)$ und $P_2(w)$, bei denen Tupel zu gleichen Variablen die gleichen Werte enthalten.

(d) **Selektion**: Ist P eine Abschnittsanfrage, und sind $x_1,\ldots,x_n$ ($n \geq 2$) Variablen von P, so ist $\varsigma^{=}_{x_1,\ldots,x_n} P$ ebenfalls eine Abschnittsanfrage. Diese liefert auf jedem Wort $w \in \Sigma^*$ die Menge aller Tupel aus $P(w)$ zurück, bei denen jeweils den Variablen $x_1,\ldots,x_n$ der gleiche Wert zugewiesen wird.

Anhand dieser können nun komplexere Anfragen beschrieben werden. Die Menge aller Anfragen, die anhand von Regex-Formeln und den oben angegebenen Operatoren beschrieben werden können, bezeichnen wir als *Kern-Abschnittsanfragen*, und bezeichnen diese Menge als

$\left[\!\left[\mathrm{RGX}^{\{\cup,\pi,\bowtie,\varsigma^{=}\}}\right]\!\right]$. Anfragen, durch Regex-Funktionen ohne diese Operatoren beschrieben werden, bezeichnen wir als *reguläre Abschnittsanfragen*, und schreiben diese Menge als $[\![\mathrm{RGX}]\!]$. Durch Einschränken der erlaubten Operatoren können wir auch weitere Klassen zwischen diesen beiden definieren, so enthält z. B. $\left[\!\left[\mathrm{RGX}^{\{\cup,\pi,\varsigma^{=}\}}\right]\!\right]$ die Menge aller Kern-Abschnittsanfragen, die $\bowtie$ nicht verwenden.

Wir interessieren uns dabei besonders für *boolesche Anfragen*, also Anfragen, bei denen auf die leere Variablenmenge projiziert wird. Dadurch erhalten wir für jedes Wort w entweder die leere Menge als Resultat (welche „nein“ entspricht), oder die Menge, die genau das leere Tupel enthält (welche „ja“ entspricht). Boolesche Abschnittsanfragen können als Entscheidungsfunktionen auf Wörtern interpretiert werden; somit definiert jede dieser Anfragen eine Sprache. Aus der Definition folgt unmittelbar, dass die booleschen Anfragen aus $[\![\mathrm{RGX}]\!]$ genau die regulären Sprachen definieren. Außerdem ist leicht zu sehen, dass boolesche Kern-Abschnittsanfragen nicht-reguläre Sprachen ausdrücken können:

Beispiel 1.2 *Sei Σ ein Terminalalphabet. Wir betrachten die Regex-Formel $\alpha := x\{\Sigma^*\}y\{\Sigma^*\}$ und die boolesche Abschnittsanfrage $P := \pi_\emptyset(\varsigma^{=}_{x,y}([\![\alpha]\!]))$. Für alle Wörter $w \in \Sigma^*$ gilt $P(w) \neq \emptyset$ genau dann, wenn $w = xx$ für ein $x \in \Sigma^*$.*

Auf den ersten Blick lässt sich vermuten, dass boolesche Anfragesprachen die gleiche Ausdrucksstärke haben wie durch Rückreferenzen erweiterte reguläre Ausdrücke (siehe z. B. [3]).

Allerdings wurde in [2] gezeigt, dass die *„Uniform-0-chunk“-Sprache*

$$L := \left\{ s_1 \cdot t \cdot s_2 \cdot t \cdots s_{n-1} \cdot t \cdot s_n \;\middle|\; n > 0,\ s_1, \ldots, s_n \in \{1\}^+,\ t \in \{0\}^* \right\}$$

zwar durch den erweiterten regulären Ausdruck $1^+ \cdot (0^*)\%x \cdot (1^+ \cdot x)^* \cdot 1^+$ beschrieben werden kann, jedoch durch keinen booleschen Abschnittsanfrage.

2. Resultate

Der technische Hauptbeitrag der hier beschriebenen Arbeit ist der Beweis, dass Kern-Abschnittsanfragen eine mächtige und natürliche Teilmenge der in [3] verwendeten erweiterten regulären Ausdrücke simulieren können, nämlich die erweiterten regulären Ausdrücke, bei denen unterhalb eines Kleene-Sterns Variablen weder gebunden, noch referenziert werden dürfen. Da die beiden Modelle mit Variablen unterschiedlich umgehen, ist dieser Nachweis technisch aufwändiger, als die Beschreibung hier vermuten lässt. Die folgenden Resultate lassen sich dann unmittelbar aus den Beweisen in [3] ableiten:

(a) Es ist unentscheidbar, ob eine boolesche Kern-Abschnittsanfrage die Allsprache Σ^* definiert.
(b) Regularität von booleschen Kern-Abschnittsanfragen ist unentscheidbar.
(c) Boolesche Kern-Abschnittsanfragen sind nicht effektiv minimierbar.
(d) Der Tradeoff zwischen booleschen Kern-Abschnittsanfragen und boolschen regulären Abschnittsanfragen ist durch keine rekursive Funktion beschränkt.

Diese unteren Schranken lassen sich unmittelbar von booleschen auf allgemeine Kern-Abschnittsanfragen übertragen. Da außerdem in den Konstruktionen $\bowtie$ nicht verwendet wird, gelten diese Resultate selbst für die Teilklasse $\left[\!\left[\mathrm{RGX}^{\{\cup,\pi,\varsigma^{=}\}}\right]\!\right]$.

Zusätzlich dazu lässt sich eine weitere Sprachklasse anhand von Kern-Abschnittsanfragen ausdrücken, nämlich die der *Patternsprachen*. Ein Pattern ist ein Wort $\alpha \in (\Sigma \cup X)^+$ über einem endlichen Terminalalphabet Σ und einem unendlichen Variablenalphabet X und definiert die Sprache aller Wörter, die durch homomorphes Ersetzen aller Variablen in α durch Wörter aus Σ^+ entstehen können. Es ist leicht zu sehen, dass jede Patternsprache durch eine boolesche Anfrage aus $\left[\!\left[\mathrm{RGX}^{\{\pi,\varsigma^=\}}\right]\!\right]$ erzeugt werden kann:

Beispiel 2.1 *Sei $\Sigma := \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Wir betrachten das Pattern $\alpha := x\mathtt{a}y\mathtt{b}yz$. Dieses erzeugt die Sprache $\mathcal{L}(\alpha) = \{x\mathtt{a}y\mathtt{b}yz \mid x, y, z \in \Sigma^+\}$. Um eine äquivalente boolesche Kern-Abschnitts-anfrage P zu konstruieren, definieren wir zuerst die Regex-Formel*

$$\alpha' := x\{\Sigma^+\} \cdot \mathtt{a} \cdot y_1\{\Sigma^+\} \cdot \mathtt{b} \cdot y_2\{\Sigma^+\} \cdot z\{\Sigma^+\}$$

und wählen dann $P := \pi_\emptyset(\varsigma^=_{y_1,y_2}([\![\alpha']\!]))$. Für alle $w \in \Sigma^*$ gilt nun $P(w) \neq \emptyset$ genau dann, wenn $w \in \mathcal{L}(\alpha)$.

Da das Wortproblem für Patternsprachen NP-vollständig ist (siehe Angluin [1]), ist das Auswertungsproblem für Anfragen aus $\left[\!\left[\mathrm{RGX}^{\{\cup,\pi,\varsigma^=\}}\right]\!\right]$ NP-hart (und somit für alle Kern-Abschnitts-anfragen).

Literatur

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

[2] R. FAGIN, B. KIMELFELD, F. REISS, S. VANSUMMEREN, Spanners: a formal framework for information extraction. In: *Proceedings of the 32nd symposium on Principles of database systems*. ACM, 2013, 37–48.

[3] D. D. FREYDENBERGER, Extended regular expressions: Succinctness and decidability. *Theory of Computing Systems* **53** (2013) 2, 159–193.

THEORIE-TAG 2014
H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 53–56.

Topologische Verfeinerungen des Cantor-Raumes 2^ω

Stefan Hoffmann

Martin-Luther-Universität
Institut für Informatik
Von-Seckendorff-Platz 1
D-06120 Halle
st.hoffmann@student.uni-halle.de

Zusammenfassung

Die übliche Topologie auf der Menge der einseitig unendlichen Wörter ist durch eine Metrik gegegeben, welche gemeinsame Präfixe misst. Diese stimmt mit der Produkttopologie überein, wenn man das Alphabet X mit der diskreten Topologie versieht. Die Menge der einseitig unendlichen Wörter X^ω zusammen mit dieser Topologie wird auch Cantor-Raum genannt. In dieser Arbeit untersuchen wir verschiedene Verfeinerungen dieser Topologie auf X^ω und stellen sie in Beziehung zueinander. Anwendung finden solche Verfeinerung zur Charakterisierung formaler Sprachen [2], oder um zufällige Folgen unendlicher Wörter zu beschreiben [7].

1. Einleitung

Im folgenden sei stets $X = \{0, 1\}$ und X^ω bezeichne die Menge aller einseitig unendlichen Folgen über X, Elemente diese Menge heißen ω-Wörter. Für zwei $\xi, \eta \in X^\omega$ sei

$$d_C(\xi, \eta) := \sup \left\{ \frac{1}{2^r} : \xi_r \neq \eta_r \right\}$$

mit der Festlegung $\sup \emptyset := 0$. Diese Metrik induziert auf X^ω eine Topologie, zusammen mit dieser Topologie heißt X^ω Cantor-Raum. Die offenen Kugeln, und damit die Basismengen, sind Mengen der Form $u \cdot X^\omega$ für $u \in X^*$. Der so konstruierte topologische Raum ist homöomorph zur der aus der Analysis bekannten Cantor-Menge (middle-third-construction) zusammen mit der Teilraumtopologie. Für weitere Eigenschaften sei auf [1, 3] verwiesen.

Diese Topologie wurde in [4, 5] zur Charakterisierung regulärer ω-Sprachen benutzt, Automatentypen und Akzeptanzmodi entsprechen Borel-Mengen über X^ω.

Durch [1] wurden erstmal systematisch verschiedene Verfeinerungen untersucht, angeregt durch deren Untersuchung habe ich in [6] diese Untersuchung fortgeführt.

Die Arbeit entstand aus der Diplomarbeit des Autors und eines Besuches in Trier und Stuttgart bei Prof. Dr. Henning Fernau und Dr. Manfred Kufleitner.

Mit X^* sei die Menge aller endlichen Wörter über X bezeichnet, ist $\xi = uv\tau$ für $u, v \in X^*, \tau \in X^\omega$ so nennen wir v einen *Faktor* von ξ, ein Faktor, welcher am Anfang steht (hier u) heißt *Präfix*. Wir schreiben $u \sqsubset \xi$, falls u ein Präfix von ξ ist, weiter bezeichnen wir mit $\xi[1 \dots n]$ das eindeutig bestimmte Präfix von ξ der Länge n. Mit $F_n(\xi)$ sei die Menge aller Faktoren der Länge n in ξ bezeichnet, mit $F_n^{(\infty)}(\xi)$ die Menge aller Faktoren der Länge n, welche unendlich oft vorkommen. Zu $\xi \in X^\omega$ bezeichnen wir mit $\xi_{\geq i} \in X^\omega$ den hinteren Teil von ξ ab der i-ten (und einschließlich dieser) Position. Teilmengen $L \subseteq X^\omega$ heißen ω-Sprachen. Zu $A \subseteq X$ sei $A^{im} := \{\xi \in A^\omega : F_1^{(\infty)} = A\}$ die Menge aller Wörter, welche jedes Zeichen aus A unendlich oft enthalten.

2. Die Topologien

Neben der üblichen Cantor-Metrik $d_C : X^\omega \times X^\omega \to \mathbb{R}$ betrachten wir folgende beiden Metriken auf X^ω, die *Infix-Metrik*

$$d_I(\xi, \eta) = \max(d_C(\xi, \eta), \sup\{2^{-r} : F_r(\xi) \neq F_r(\eta)\})$$

und die *Infix-Unendlich-Metrik*

$$d_\infty(\xi, \eta) = \max(d_C(\xi, \eta), \sup\{2^{-r} : F_r^{(\infty)}(\xi) \neq F_r^{(\infty)}(\eta)\})$$

d.h. neben dem Präfix vergleichen wir noch die längsten gemeinsamen Faktoren, bzw. jene Faktoren die unendlich oft auftreten. Die durch diese Metriken induzierten topologischen Räume bezeichnen wir mit $(X^\omega, \mathcal{T}_{d_I})$ und $(X^\omega, \mathcal{T}_{d_\infty})$. Da $d_C(\xi, \eta) \leq d_I(\xi, \eta)$ und $d_C(\xi, \eta) \leq d_\infty(\xi, \eta)$ sind beide topologischen Räume Verfeinerungen des Cantor-Raumes. Weiterhin betrachten wir die durch folgende Basismengen gegebenen Topologien:

1. Die automatische Topologie $\mathcal{T}_A$

$$\mathbb{B}_A = \{F : F \subseteq X^\omega, F \text{ ist reguläre und abgeschlossene } \omega\text{-Sprache im Cantorraum}\}.$$

2. Die alphabetische Topologie $\mathcal{T}_\alpha$

$$\mathbb{B}_\alpha = \{w \cdot A^\omega, w \in X^*, A \subseteq X\}.$$

3. Die stark alphabetische Topologie $\mathcal{T}_s$

$$\mathbb{B}_s = \{w \cdot (A^\omega \cap A^{im}) : w \in X^*, A \subseteq X\}.$$

4. Die Büchi-Topologie $\mathcal{T}_B$

$$\mathbb{B}_B = \{F : F \subseteq X^\omega, F \text{ ist reguläre } \omega\text{-Sprache }\}.$$

Zu zwei Topolgien $\mathcal{T}, \mathcal{T}'$ auf einer Menge sagen wir $\mathcal{T}'$ *verfeinert* $\mathcal{T}$, falls $\mathcal{T} \subseteq \mathcal{T}'$ gilt, in diesem Fall sagt man auch $\mathcal{T}$ ist *gröber* als $\mathcal{T}'$. Im folgenden Diagramm zeichnen wir einen Pfeil von einer Topologie zu einer anderen, wenn jene Topologie, bei welcher die Pfeil endet, erstere, von welcher der Pfeil ausgeht, verfeinert. In [1] und [6] wurden folgende Beziehungen gezeigt:

$$\begin{tikzcd}
 & \mathcal{T}_B & \\
T_A \arrow[ur] & & \\
T_{d_I} \arrow[u] & T_s \arrow[u] & T_{d_\infty} \arrow[ul] \\
 & T_\alpha \arrow[ul] & \\
 & T_C \arrow[u] \arrow[ur] &
\end{tikzcd}$$

3. Zur Metrisierbarkeit

Alle auch nicht durch eine konkrete Metrik gegebenen Topologien sind metrisierbar, in [1] wurde dies unter Rückgriff auf einen abstrakten topologischen Satz gezeigt, konkrete Metriken wurden nicht genannt. Im folgenden möchte ich für alle Topologien noch konkrete Metriken nennen[1]

Es sei $\mathcal{A} = (X, Q, \delta, q_0)$ ein deterministischer, partieller, endlicher Automat, dann definieren wir

$$T(\mathcal{A}, q_0) := \{\xi \in X^\omega : \delta(q_0, v) \text{ ist definiert für alle } v \sqsubset \xi\}.$$

Wir definieren

$$\begin{aligned} d_P(\xi, \eta) := \sup\{2^{-|Q|} : \mathcal{A} = (X, Q, \delta, q_0),\ &((\xi \in T(\mathcal{A}) \text{ und } \eta \notin T(\mathcal{A})) \\ &\text{oder } (\xi \notin T(\mathcal{A}) \text{ und } \eta \in T(\mathcal{A})))\} \end{aligned}$$

Es sei $\mathcal{A} = (X, Q, \delta, q_0, \mathcal{F})$ ein beliebiger (möglicherweise nichtdeterministischer) endlicher Automat mit ausgezeichneter Finalzustandsmenge, dann bezeichne mit $L_B(\mathcal{A})$ die nach der Büchi-Bedingung akzeptierte Sprache, d.h. ein Wort wird akzeptiert falls es von q_0 startend einen Zustand aus $\mathcal{F}$ unendlich oft besucht. Dann definiere

$$\begin{aligned} d_B(\xi, \eta) := \sup\{2^{-|Q|} : \mathcal{A} = (X, Q, \delta, q_0, \mathcal{F}),\ &((\xi \in L_B(\mathcal{A}) \text{ und } \eta \notin L_B(\mathcal{A})) \\ &\text{oder } (\xi \notin L_B(\mathcal{A}) \text{ und } \eta \in L_B(\mathcal{A})))\} \end{aligned}$$

Es gilt:

Satz 3.1 *Die Metrik $d_B : X^\omega \times X^\omega \to \mathbb{R}$ induziert die Büchi-Topologie $\mathcal{T}_B$ auf X^ω, die Metrik $d_P : X^\omega \times X^\omega \to \mathbb{R}$ induziert die automatische Topologie auf X^ω.*

[1] Mein Dank gilt Herrn Prof. Henning Fernau und Dr. Manfred Kufleitner für angeregte Diskussionen zu diesem Thema bei meinem Besuch in Trier und Stuttgart, Herrn Dr. Manfred Kufleitner sind die beiden Metriken für die alphabetische und stark alphabetische Topologie zu verdanken.

Definiere $\sup\emptyset := 0$ und

$$\delta_a(\xi,\eta) = \begin{cases} 0 & \text{falls } F_1(\xi) = F_1(\eta) \\ 1 & \text{sonst.} \end{cases}$$

weiter

$$d_\alpha(\xi,\eta) := \sup\left\{2^{-i} + \delta_a(\xi_{\geq i},\eta_{\geq i}) : \xi_i \neq \eta_i\right\}$$

und

$$\delta_s(\xi,\eta) = \begin{cases} 0 & \text{falls } F_1(\xi) = F_1(\eta) \text{ und } F_1^{(\infty)}(\xi) = F_1^{(\infty)}(\eta) \\ 1 & \text{sonst.} \end{cases}$$

sowie

$$d_s(\xi,\eta) = \sup\left\{2^{-i} + \delta_s(\xi_{\geq i},\eta_{\geq i}) : \xi_i \neq \eta_i\right\}$$

dann gilt:

Satz 3.2 *Die Metrik $d_\alpha : X^\omega \times X^\omega \to \mathbb{R}$ induziert die automatische Topologie $\mathcal{T}_\alpha$ auf X^ω, die Metrik $d_s : X^\omega \times X^\omega \to \mathbb{R}$ induziert die strikt automatische Topologie $\mathcal{T}_s$ auf X^ω.*

Damit sind die genannten Topologien und deren Beziehungen erschöpfend dargestellt. Für weitere Eigenschaften der topologischen Räume, Beweise, Motivationen und Literatur verweise ich auf [1, 6], weiter liegt in [6] eine hinreichende Bedingung vor, damit Cantor-abgeschlossene Mengen offen in $\mathcal{T}_{d_I}$ sind, die Frage ob diese Bedingung auch notwendig ist bleibt offen.

Literatur

[1] S. SCHWARZ, L. STAIGER, Topologies refining the Cantor topology on X^ω. In: C.S. CALUDE et al. (eds.), *Theoretical Computer Science, IFIP Advances in Information and Communication Technology* **323**, Springer-Verlag, Berlin 2010, 271 - 285.

[2] V. DIEKERT, M. KUFLEITNER, Fragments of first-order logic over infinite words. In: S. ALBERS, J.-Y. MARION (eds.), *STACS 2009, Freiburg, Germany, February 26-28, 2009, Proceedings*, Online proceedings at DROPS and HAL, 2009, 325-336.

[3] D. PERRIN, J.-E. PIN, *Infinite Words*, Elsevier, Amsterdam, 2004.

[4] L. STAIGER, K. WAGNER, Automatentheoretische und automatenfreie Charakterisierungen topologischer Klassen regulärer Folgenmengen. *Elektronische Informationsverarbeitung und Kybernetik* **10** (1974) 7, 379-392.

[5] L.H. LANDWEBER, Decision Problems for ω-Automata. *Mathematical Systems Theory* **3** (1969) (4), 376-384.

[6] S. HOFFMANN, Metriken zur Verfeinerung des Cantor-Raumes auf X^ω, Diplomarbeit, Insitut für Informatik, Lehrstuhl für Theoretische Informatik, Martin-Luther-Universität Halle-Wittenberg, 2014.

[7] C.S. CALUDE, S. MARCUS, L. STAIGER, A topological characterization of random sequences. *Information Processing Letters* **88** (2003), 245-250.

THEORIE-AG 2014
H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 57–60.

On the Computational Complexity of Partial Word Automata Problems

Markus Holzer Sebastian Jakobi Matthias Wendlandt

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

1. Introduction

A *partial* word over alphabet Σ is a word in which certain positions are unspecified. Usually, these undefined positions are represented by a "hole" symbol $\diamond \notin \Sigma$. During the last two decades a vast amount on literature appeared on the combinatorics and algorithms on partial words—we refer to [2] for further reading. In a recent attempt [4], partial words were linked to regular languages. The motivation for this approach was to use these partial words to compress the representation of ordinary word languages. To this end, one has to specify the possible contents of the holes. For instance, the language $L = \{ab, bb, ac, bc\}$ can be represented by the language $L' = \{\diamond b, \diamond c\}$ on partial words, when replacing a hole $\diamond$ by either the letter a or b, resulting in the words ab, ac in the former case and bb, bc in the latter. Formally, the replacement of the holes is modeled by a $\diamond$-*substitution* σ over Σ, which is a mapping from $\Sigma_\diamond = \Sigma \cup \{\diamond\}$ to 2^{Σ^*} satisfying (i) $\sigma(a) = \{a\}$, for every $a \in \Sigma$, (ii) $\sigma(\diamond) \subseteq \Sigma$, and (iii) $\sigma(vw) = \sigma(v) \cdot \sigma(w)$, for every $v, w \in \Sigma_\diamond^*$.

For the representation of the (partial) word languages it is convenient to use deterministic finite automata (DFAs). In fact, besides [4], also [1] and [3] stick to this representation. In the former two papers, descriptional and computational complexity issues related to the compression based on partial words are studied, while in the latter paper [3], an approximation algorithm for the compression of finite languages is given. Moreover, there it is also mentioned that the minimization problem, that is, asking whether a language given by a DFA that is associated to a $\diamond$-substitution σ can be compressed down to state size k by using σ on a partial word DFA, for short $\diamond$-DFA, is computationally intractable, namely NP-hard. In [4], it was also shown that the equivalence problem on partial word automata, that is, given a DFA A, a $\diamond$-DFA B, and $\diamond$-substitution σ, decide whether $L(A) = \sigma(L(B))$ holds, is coNP-hard. From the computation complexity point of view these are the only results known for partial word automata, up to our knowledge. The aim of the present paper is to just about complete the picture on decision problems related to partial word automata.

This is an extended abstract of: M. Holzer, S. Jakobi, M. Wendlandt. *On the Computational Complexity of Partial Word Automata Problems.* In S. Bensch, R. Freund, F. Otto (eds.): Proceedings of the 6th Workshop on Non-Classical Models of Automata and Applications, volume 304 of books@ocg.at, Austrian Computer Society, pages 131-146, Kassel, Germany, July 2014.

2. Language Problems

Basic decision problems concerning the language accepted by automata are the emptiness, universality, and equivalence problems. We study the complexity of corresponding problems for partial word automata. We use the following notation. Let $\mathcal{X}$ be the class of $\diamond$-DFAs or the class of $\diamond$-NFAs. The *emptiness problem* $\mathcal{X}$-eq-$\emptyset$ is to decide for a given partial word automaton $A \in \mathcal{X}$ and a given $\diamond$-substitution σ, whether $\sigma(L(A)) = \emptyset$. Here the $\diamond$-substitution σ is given as input. However, one could also consider the following "existential"-variant of this problem: the problem $\mathcal{X}$-eq-$\emptyset_{(\exists\sigma)}$ is to decide for a given partial word automaton $A \in \mathcal{X}$, whether *there exists* some $\diamond$-substitution σ such that $\sigma(L(A)) = \emptyset$. Similarly, the problem $\mathcal{X}$-eq-$\emptyset_{(\forall\sigma)}$ asks whether $L(A) = \emptyset$ holds *for all* "appropriate" $\diamond$-substitutions σ. Here it has to be specified, what an appropriate $\diamond$-substitution should be. If A is a partial word automaton with input alphabet $\Sigma_\diamond$, then the $\diamond$-substitution σ satisfies $\sigma(\diamond) \subseteq \Sigma$. But still the question is, whether the *empty* $\diamond$-substitution σ with $\sigma(\diamond) = \emptyset$ is appropriate or not. If not stated otherwise, we only consider *non-empty* $\diamond$-substitutions in this case. Although at a first glance this seems to be a negligibility, it turns out that this issue may induce a significant difference in the complexity of the corresponding decision problems. Concerning the complexity of the emptiness problem we have the following result.

Theorem 2.1 (Emptiness) *For $\mathcal{X} \in \{\diamond\text{-DFA}, \diamond\text{-NFA}\}$ the problems $\mathcal{X}$-eq-$\emptyset$, $\mathcal{X}$-eq-$\emptyset_{(\exists\sigma)}$, and $\mathcal{X}$-eq-$\emptyset_{(\forall\sigma)}$ are* NL*-complete.* □

Similar to emptiness problems, we define the *universality problems* $\mathcal{X}$-eq-Σ^*, $\mathcal{X}$-eq-$\Sigma^*{}_{(\exists\sigma)}$, and $\mathcal{X}$-eq-$\Sigma^*{}_{(\forall\sigma)}$. Moreover, for automata classes $\mathcal{X}$ and $\mathcal{Y}$, which can be the classes of DFAs, NFAs, $\diamond$-DFAs, or $\diamond$-NFAs, we define the *equivalence problems* $\mathcal{X}$-eq-$\mathcal{Y}$, $\mathcal{X}$-eq-$\mathcal{Y}_{(\exists\sigma)}$, and $\mathcal{X}$-eq-$\mathcal{Y}_{(\forall\sigma)}$. Here a $\diamond$-substitution σ of course only applies to partial word automata. For example the problem $\diamond$-DFA-eq-DFA$_{(\exists\sigma)}$ is to decide for a given $\diamond$-DFA A and a given DFA B, whether there exists a $\diamond$-substitution σ such hat $\sigma(L(A)) = L(B)$. The following theorem shows that all these problems are PSPACE-complete.

Theorem 2.2 (Universality, Equivalence) *The universality problems $\mathcal{X}$-eq-Σ^*, $\mathcal{X}$-eq-$\Sigma^*{}_{(\exists\sigma)}$, and $\mathcal{X}$-eq-$\Sigma^*{}_{(\forall\sigma)}$, and the equivalence problems $\mathcal{X}$-eq-$\mathcal{Y}$, $\mathcal{X}$-eq-$\mathcal{Y}_{(\exists\sigma)}$, and $\mathcal{X}$-eq-$\mathcal{Y}_{(\forall\sigma)}$ are* PSPACE*-complete for $\mathcal{X} \in \{\diamond\text{-DFA}, \diamond\text{-NFA}\}$ and $\mathcal{Y} \in \{\diamond\text{-DFA}, \diamond\text{-NFA}, \text{DFA}, \text{NFA}\}$.* □

Finally we show that the "for all"-variant of the equivalence problem for $\diamond$-DFAs becomes much easier if also the empty $\diamond$-substitution $\sigma_\emptyset$ with $\sigma_\emptyset(\diamond) = \emptyset$ is considered. The proof uses the fact that it suffices to check whether $\sigma_\emptyset(L(A)) = L(B)$ and $\sigma_\Sigma(L(A)) \subseteq L(B)$ holds, where $\sigma_\Sigma(\diamond) = \Sigma$. Both conditions can be checked in NL.

Theorem 2.3 *The problem of deciding for a given $\diamond$-DFA A and a DFA B with input alphabet Σ, whether for all $\diamond$-substitutions σ, with $\sigma(\diamond) \subseteq \Sigma$, we have $\sigma(L(A)) = L(B)$, is* NL*-complete.* □

In [4], the authors introduce a property of languages called essentially σ-definable. Let σ be a $\diamond$-substitution over Σ. A language $L \subseteq \Sigma^*$ is *essentially σ-definable* if $L = \sigma(L')$ for some language $L' \subseteq (\Sigma \cup \{\diamond\})^* \cdot \{\diamond\} \cdot (\Sigma \cup \{\diamond\})^*$, where every word contains at least one $\diamond$ symbol. Here we study the following decision problems. The problem DFA-σ-def is to decide for a given DFA A and a $\diamond$-substitution σ, whether language $L(A)$ is essentially σ-definable. The

problems DFA-σ-def$_{(\exists\sigma)}$ and DFA-σ-def$_{(\forall\sigma)}$ ask whether $L(A)$ is essentially σ-definable for *some* $\diamond$-substitution σ, or, respectively, for *all* $\diamond$-substitutions σ, with $\sigma(\diamond) \neq \emptyset$.

In [4], it is shown that, given a regular language L and a $\diamond$-substitution σ, it is decidable whether L is essentially σ-definable. In particular, it is shown that the language $L(A)$ accepted by a DFA $A = (Q, \Sigma, q_0, F, \delta)$ is essentially σ-definable if and only if $L(A) = \bigcup_{q \in Q} R_q$, where

$$R_q = \{\, x \mid x \in \Sigma^*, \delta(q_0, x) = q \,\} \cdot \Sigma' \cdot \{\, y \mid y \in \Sigma^*, \delta(q', y) \in F, \text{ for all } q' \in \delta(q, \Sigma) \,\}$$

with $\Sigma' = \sigma(\diamond)$. Using this result, one can deduce a PSPACE-algorithm for deciding the different definability problems for automata over a constant size alphabet. In fact, all these problems are also PSPACE-complete, as the following result shows.

Theorem 2.4 (Essential σ-Definability) *Let Σ be a fixed alphabet with $|\Sigma| \geq 4$. The problems DFA-σ-def, DFA-σ-def$_{(\exists\sigma)}$, and DFA-σ-def$_{(\forall\sigma)}$, when restricted to DFAs with input alphabet Σ, are* PSPACE*-complete.* □

3. Minimization Problems

We now study minimization problems for partial word automata, where we use a similar notation for our problems as in Section 2. For automata classes $\mathcal{X}$ and $\mathcal{Y}$ we consider the $\mathcal{X}$-to-$\mathcal{Y}$ and $\mathcal{X}$-to-$\mathcal{Y}_{(\exists\sigma)}$ *minimization problems*. For example, the NFA-to-$\diamond$-DFA$_{(\exists\sigma)}$ minimization problem is to decide for a given NFA A and an integer n, which is given in unary notation, whether there exist an n-state $\diamond$-DFA B and a $\diamond$-substitution σ, such that $L(A) = \sigma(L(B))$. The following result shows PSPACE-completeness of these problems.

Theorem 3.1 (Minimization) *The problems $\mathcal{X}$-to-$\mathcal{Y}$ and $\mathcal{X}$-to-$\mathcal{Y}_{(\exists\sigma)}$ are* PSPACE*-complete, if $\mathcal{X}, \mathcal{Y} \in \{\diamond\text{-DFA}, \diamond\text{-NFA}, \text{DFA}, \text{NFA}\}$, with $\{\mathcal{X}, \mathcal{Y}\} \cap \{\diamond\text{-DFA}, \diamond\text{-NFA}\} \neq \emptyset$.* □

Concerning "for all"-variants of the minimization problem, there are two possible problem definitions, depending on the order of the quantifiers:

Theorem 3.2 *Given a $\diamond$-DFA A with input alphabet $\Sigma_\diamond$ and an integer k, the following problems are* PSPACE*-complete:* (i) *Does for all $\diamond$-substitutions σ over Σ, with $\sigma(\diamond) \neq \emptyset$, exist a k-state $\diamond$-DFA B, such that $\sigma(L(A)) = \sigma(L(B))$?* (ii) *Does there exist a k-state $\diamond$-DFA B, such that for all $\diamond$-substitutions σ over Σ with $\sigma(\diamond) \neq \emptyset$, we have $\sigma(L(A)) = \sigma(L(B))$?* □

4. Problems on Finite Languages

In this section we shortly consider decision problems for partial word automata that accept only finite languages. Here the complexity drops significantly from PSPACE-completeness down to coNP-completeness or -hardness and containment in Σ_2^{P}, depending on the problem under consideration. For proving coNP-hardness of these problems we use the following result:

Theorem 4.1 (Union Bounded-Universality) *Deciding for given DFAs $A_1, A_2, \ldots A_n$ with input alphabet Σ and an integer ℓ coded in unary, whether $\bigcup_{i=1}^{n} L(A_i) = \Sigma^{\leq \ell}$, is* coNP*-complete.* □

We denote by DFA_{fin} and NFA_{fin} the deterministic and, respectively, nondeterministic finite automata that accept finite languages, and similarly use $\diamond$-DFA_{fin} and $\diamond$-NFA_{fin} to denote the respective partial word automata. Our findings on the complexity of the partial word automata problems for finite languages are summarized in the following theorem.

Theorem 4.2 (Bounded-Universality, Equivalence) *The two bounded-universality problems $\mathcal{X}$-eq-$\Sigma^{\leq \ell}$ and $\mathcal{X}$-eq-$\Sigma^{\leq \ell}{}_{(\forall\sigma)}$, and the equivalence problems $\mathcal{X}$-eq-$\mathcal{Y}$ and $\mathcal{X}$-eq-$\mathcal{Y}_{(\forall\sigma)}$ are* coNP*-complete, and the problems $\mathcal{X}$-eq-$\Sigma^{\leq \ell}{}_{(\exists\sigma)}$ and $\mathcal{X}$-eq-$\mathcal{Y}_{(\exists\sigma)}$ are* coNP*-hard and contained in Σ_2^{P}, for $\mathcal{X} \in \{\diamond\text{-}DFA_{fin}, \diamond\text{-}NFA_{fin}\}$ and $\mathcal{Y} \in \{\diamond\text{-}DFA_{fin}, \diamond\text{-}NFA_{fin}, DFA_{fin}, NFA_{fin}\}$.* □

For the σ-definability problems for DFAs for finite languages we have the following result.

Theorem 4.3 (Essential σ-Definability) *Let Σ be fixed alphabet with $|\Sigma| \geq 4$. The problems DFA_{fin}-σ-def and DFA_{fin}-σ-$def_{(\forall\sigma)}$, when restricted to DFAs with input alphabet Σ, are* coNP*-complete, and the problem DFA_{fin}-σ-$def_{(\exists\sigma)}$, when restricted to DFAs with input alphabet Σ, is* coNP*-hard and contained in Σ_2^{P}.* □

Our results concerning minimization of partial word automata for finite languages are summarized in the next theorem.

Theorem 4.4 (Minimization) *The two minimization problems $\mathcal{X}$-to-$\mathcal{Y}$ and $\mathcal{X}$-to-$\mathcal{Y}_{(\exists\sigma)}$ are hard for* coNP *and are contained in Σ_2^{P}, for all $\mathcal{X}, \mathcal{Y} \in \{\diamond\text{-}DFA_{fin}, \diamond\text{-}NFA_{fin}, DFA_{fin}, NFA_{fin}\}$, with $\{\mathcal{X}, \mathcal{Y}\} \cap \{\diamond\text{-}DFA_{fin}, \diamond\text{-}NFA_{fin}\} \neq \emptyset$.* □

In the remainder of this section we briefly recall what is known for the "for all"-variants of the minimization problem for automata accepting finite languages. By the definition of these problem variants it is easily seen that the first one from Theorem 3.2 belongs to Σ_3^{P}, while the second variant from Theorem 3.2 is contained in Σ_2^{P}. We have to leave open the lower bounds for these problems.

References

[1] E. Balkanski, F. Blanchet-Sadri, M. Kilgore, B. J. Wyatt, Partial Word DFAs. In: S. Konstantinidis (ed.), *Proceedings of the 18th International Conference on Implementation and Application of Automata*. Number 7982 in LNCS, Springer, Halifax, Nova Scotia, Canada, 2013, 36–47.

[2] F. Blanchet-Sadri, *Algorithmic Combinatorics on Partial Words*. Discrete Mathematics and Its Applications, Chapman and Hall/CRC, 2007.

[3] F. Blanchet-Sadri, K. Goldner, A. Shackleton, Minimal Partial Languages and Automata. In: M. Holzer, M. Kutrib (eds.), *Proceedings of the 19th International Conference on Implementation and Application of Automata*. Number 8587 in LNCS, Springer, Giessen, Germany, 2014, 110–123.

[4] J. Dassow, F. Manea, R. Mercas, Regular languages of partial words. *Information Sciences* **268** (2014), 290–304.

THEORIE-TAG 2014

H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 61–62.

The Monoid of Queue Actions – A New (?) Monoid Worth to be Studied (?) –

Martin Huschenbett[(A)] Dietrich Kuske[(A)] Georg Zetzsche[(B)]

[(A)]TU Ilmenau, Institut für Theoretische Informatik

[(B)]TU Kaiserslautern, Fachbereich Informatik

Abstract

We model the behaviour of a fifo queue as a monoid of transformations that are induced by sequences of writing and reading. We describe this monoid by means of a confluent and terminating semi-Thue system and study some of its basic algebraic properties, e.g., conjugacy. Moreover, we show that while several properties concerning its rational subsets are undecidable, their uniform membership problem is NL-complete. Furthermore, we present an algebraic characterization of this monoid's recognizable subsets. Finally, we prove that it is not Thurston-automatic.

These results have been accepted for publication in the proceedings of MFCS.

Basic computing models differ in their storage mechanisms: there are finite memory mechanisms, counters, blind counters, partially blind counters, pushdowns, Turing tapes, queues and combinations of these mechanisms. Every storage mechanism naturally comes with a set of basic actions like reading a symbol from or writing a symbol to the pushdown. As a result, sequences of basic actions transform the storage. The set of transformations induced by sequences of basic actions then forms a monoid. As a consequence, fundamental properties of a storage mechanism are mirrored by algebraic properties of the induced monoid. For example, the monoid induced by a deterministic finite automaton is finite, a single blind counter induces the integers with addition, and stacks induce polycyclic monoids. In this paper, we are interested in a queue as a storage mechanism. In particular, we investigate the monoid $\mathcal{Q}$ induced by a single queue.

The basic actions on a queue are writing the symbol a into the queue and reading the symbol a from the queue (for each symbol a from the alphabet of the queue). Since a can only be read from a queue if it is the first entry in the queue, these actions are partial. Hence, for every sequence of basic actions, there is a queue of shortest length that can be transformed by the sequence without error (i.e., without attempting to read a from a queue that does not start with a). Our first main result provides us with a normal form for transformations induced by sequences of basic actions: the transformation induced by a sequence of basic actions is uniquely given by the subsequence of write actions, the subsequence of read actions, and the length of the shortest queue that can be transformed by the sequence without error. The proof is based on a convergent finite semi-Thue system for the monoid $\mathcal{Q}$. This semi-Thue system also

allows to describe the normal form of the product of two sequences of basic actions in normal form, i.e., to describe the monoid operation in terms of normal forms.

The fundamental notion of conjugacy in groups has been extended to monoids in two different ways: call x and y *conjugate* if the equation $xz = zy$ has a solution, and call them *transposed* if there are u and v such that $x = uv$ and $y = vu$. In the free monoid as well as in any group, these two notions coincide which is not the case for the queue monoid $\mathcal{Q}$. We prove that conjugacy is the transitive closure of transposition and that two elements of $\mathcal{Q}$ are conjugate if and only if their subsequences of write and of read actions, respectively, are conjugate in the free monoid. This characterization allows in particular to decide conjugacy in polynomial time. Furthermore, we show that the set of solutions $z \in \mathcal{Q}$ of $xz = zy$ is effectively rational but not necessarily recognizable.

We proceed by investigating algorithmic properties of rational subsets of $\mathcal{Q}$. Employing the fact that every element of $\mathcal{Q}$ has only polynomially many left factors, we can nondeterministically solve the rational subset membership problem in logarithmic space. Since the direct product of two free monoids embeds into $\mathcal{Q}$, all the negative results on rational transductions as, e.g., the undecidability of universality of a rational subset translate into our setting.

Then we charaterize the recognizable subsets of $\mathcal{Q}$. Recall that an element of $\mathcal{Q}$ is completely given by its subsequences of write and read actions, respectively, and the length of the shortest queue that can be transformed without an error. Regular conditions on the subsequences of write and read actions, respectively, lead to recognizable sets in $\mathcal{Q}$. Regarding the shortest queue that can be transformed without error, the situation is more complicated: the set of elements of $\mathcal{Q}$ that operate error-free on the empty queue is not recognizable. Based on an approximation of the length of the shortest queue, we define recognizable subsets $\Omega_k \subseteq \mathcal{Q}$. The announced characterization then states that a subset of $\mathcal{Q}$ is recognizable if and only if it is a Boolean combination of regular conditions on the subsequences of write and read actions, respectively, and sets Ω_k.

THEORIE-TAG 2014
H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 63–64.

News on the Square Conjecture

Natasha Jonoska[(A)] Florin Manea[(B)] Shinnosuke Seki[(C)]

[(A)]Department of Mathematics and Statistics, University of South Florida,
4202 East Fowler Avenue, Tampa, FL 33620, USA,
jonoska@math.usf.edu

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

[(C)]Helsinki Institute for Information Technology (HIIT),
Department of Information and Computer Science,
Aalto University, P. O. Box 15400, FI-00076, Aalto, Finland,
shinnosuke.seki@aalto.fi

Abstract

The squares conjecture claims that the number of distinct squares that occur in a word of length n is at most n. We present a series of new results connected to this conjecture.

1. Introduction

Let Σ be an alphabet and Σ^* be the set of all words over Σ. Let $w \in \Sigma^*$. By $|w|$, we denote its length. For a letter $a \in \Sigma$, we denote the number of occurrences of a's in w by $|w|_a$. In this paper, n exclusively denotes the length of a word in which squares are to be counted.

Let $\mathrm{Sq}(w) = \{uu \mid w = xuuy \text{ for some } x, y \in \Sigma^* \text{ with } w \neq xy\}$ be the set of all squares occurring in w. Its size, denoted by $\#\mathrm{Sq}(w)$, has been conjectured to be bounded from above by the length of w [2].

That is to say, $\#\mathrm{Sq}(w) \leq n$ for any word w of length n; a slightly stronger conjecture is $\#\mathrm{Sq}(w) \leq n - |\Sigma|$, given in [1]. Notable upper bounds shown so far are $\#\mathrm{Sq}(w) \leq 2n$ [2], further improved by Ilie to $\#\mathrm{Sq}(w) \leq 2n - \lg n$ [3], this being the best bound known so far.

An infinite word, over the binary alphabet $\Sigma_2 = \{a, b\}$, whose finite factors have a relatively large number of distinct squares compared to their length was given by Fraenkel and Simpson [2]:

$$w_{\mathrm{fs}} = a^1ba^2ba^3ba^2ba^3ba^4ba^3ba^4ba^5ba^4ba^5ba^6b\cdots. \tag{1}$$

None of its factors of length n with k letters b contain more than $\frac{2k-1}{2k+2}n$ distinct squares. In fact, we proposed (Conjecture 1, [4]) that this upper bound holds not only for the factors of w_{fs} but for all binary words. A computer program verified the conjecture for all binary words of length less than 30, as well as for randomly generated binary words of length up to 500 without any counterexample found.

Conjecture 1 *Let $k \geq 2$. For any binary word $w \in \Sigma_2^+$ of length n with k b's where $k \leq \lfloor \frac{n}{2} \rfloor$,*

$$\#\mathrm{Sq}(w) \leq \frac{2k-1}{2k+2} n.$$

The bound is defined here as a function of the number of b's. However, Conjecture 1 gives an upper bound on number of squares more generally by redefining k as $\min\{|w|_a, |w|_b\}$, as the number of distinct squares in a binary word is invariant under the isomorphism swapping the letters a and b.

Note that our conjecture not only strengthens the conjecture that $\#\mathrm{Sq}(w) \leq n$, but its dependency on the number of b's suggests that a possible proof might be obtained by induction on this number. We show here that it holds when at most nine b's are present in the word.

Parenthesizing the sequence of positive integers representing the powers of a's in the word w_{fs}, in a convenient manner gives the sequence $(1,2,3)$, $(2,3,4)$, $(3,4,5), \ldots$. This reveals the structure of w_{fs} as catenation of simpler words $a^i b a^{i+1} b a^{i+2} b$, $i = 1,2,\ldots$. As another contribution of [4], we propose a structurally simpler infinite word, whose coefficients just increment:

$$w_{\mathrm{jms}} = a^1 b a^2 b a^3 b a^4 b a^5 b a^6 b \cdots, \tag{2}$$

and prove that it is quite rich with respect to the number of squares its factors contain. Indeed, we show that its factors achieve the upper bound in Conjecture 1 asymptotically.

The word w_{jms} points out that a word does not necessarily need a complicated structure in order to have many squares. Thus, we further proved in [4] that for any word w of length n with k letters b, whose coefficient sequence is sorted (incrementing or decrementing), Conjecture 1 holds. This result follows by induction on the number of b's on the word.

As an important technical tool, our analysis was not based on combinatorial properties that the word itself has, but rather on the combinatorial properties of the sequence of powers of the letters a (called here "coefficient sequence"). This allows us to define more general classes of words for which the conjecture holds (see [4]).

A series of other recent results connected to the squares conjecture (more precisely, about ways to construct words with a high squares density) will be overviewed in the talk.

References

[1] A. Deza, F. Franek, M. Jiang, A d-step Approach for Distinct Squares in Strings. In: *CPM 2011: Proceedings of the Combinatorial Pattern Matching - 22nd Annual Symposium*. Lecture Notes in Computer Science 6661, Springer, 2011, 77–89.

[2] A. S. Fraenkel, J. Simpson, How Many Squares Can a String Contain? *Journal of Combinatorial Theory, Series A* **82** (1998), 112–120.

[3] L. Ilie, A Note on the Number of Squares in a Word. *Theoretical Computer Science* **380** (2007), 373–376.

[4] N. Jonoska, F. Manea, S. Seki, A Stronger Square Conjecture on Binary Words. In: *SOFSEM 2014: Theory and Practice of Computer Science - 40th International Conference on Current Trends in Theory and Practice of Computer Science*. Lecture Notes in Computer Science 8327, Springer, 2014, 339–350.

THEORIE-TAG 2014

H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 65–68.

Analyse der Entscheidbarkeit diverser Probleme in automatischen Graphen

Chris Köcher

Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
chris.koecher@tu-ilmenau.de

Zusammenfassung

In dieser Arbeit werden automatische und rekursive Versionen verschiedener NP-vollständiger Probleme untersucht und in die arithmetische bzw. analytische Hierarchie eingeordnet. Hierbei stellt sich heraus, dass die Frage nach einem Perfekten Matching in automatischen wie in rekursiven Graphen Σ_1^1-vollständig ist. Ebenso wird gezeigt, dass die Frage nach der Erfüllbarkeit einer (unendlichen) aussagenlogischen Formel, welche hier als Graphproblem modelliert wird, sich ebenfalls in Σ_1^1 einordnet. Gleichzeitig wird aber auch gezeigt, dass sich für eine natürliche Einschränkung dieses Problems in der automatischen Version die Π_1^0-Vollständigkeit ergibt. Zuletzt wird noch die Frage nach einer Graphfärbung in automatischen Graphen in Π_1^0 eingeordnet, wobei sogar die Entscheidbarkeit im Falle von unendlichen Farbmengen gezeigt werden kann.

1. Einleitung

In den Anwendungsgebieten der Informatik und Mathematik stellt sich häufig heraus, dass verschiedenste Probleme sich auf eines der klassischen Graphprobleme reduzieren lassen. Beispielsweise kann das Graphfärbungsproblem für verschiedene Schedulingprobleme oder im Compilerbau verwendet werden [7].

Für endliche Graphen ist dieses Problem entscheidbar, aber für mehr als zwei Farben NP-vollständig. Es stellt sich hierbei die vollkommen natürliche Frage, wie sich das Graphfärbungsproblem und auch andere Graphprobleme in unendlichen Graphen verhalten. Hirst und Harel haben in [2] eine Reihe solcher NP-vollständiger Graphprobleme auf rekursive Graphen (d.h. Knoten- und Kantenmenge können von Turingmaschinen akzeptiert werden) ausgeweitet und hierfür die Σ_1^1-Vollständigkeit festgestellt.

Da in rekursiven Strukturen bekanntlich die Menge der erfüllbaren prädikatenlogischen Formeln erster Stufe im Allgemeinen unentscheidbar ist, wurde infolgedessen die Klasse der automatischen Strukturen (dies sind Strukturen, deren Universum und Relationen von endlichen

Die zugrundeliegende Arbeit wurde vom Autor als Abschlussarbeit seines Bachelor-Studiums in der Informatik eingereicht [4].

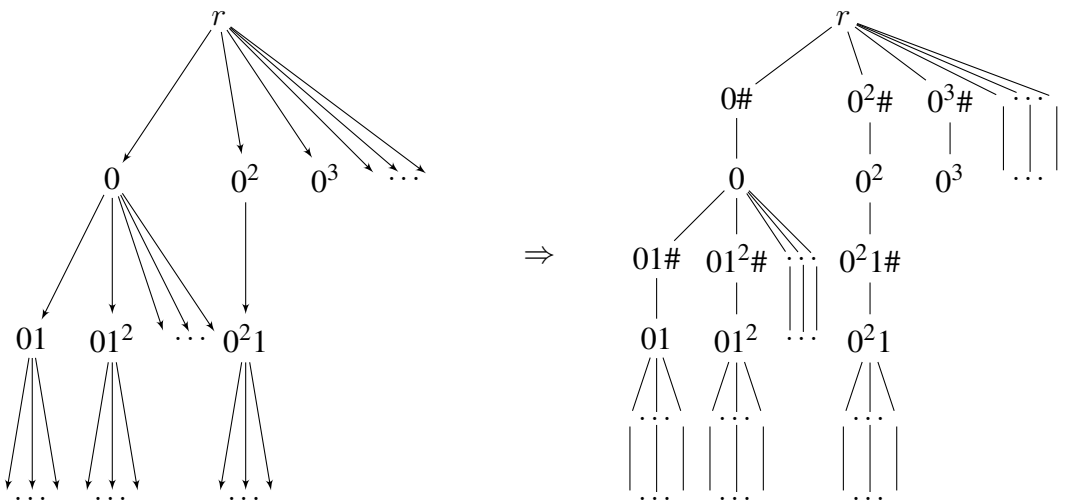
Automaten akzeptiert werden können) untersucht. Khoussainov und Nerode haben in [3] für diese Klasse die Entscheidbarkeit der erfüllbaren Formeln der Prädikatenlogik erster Stufe festgestellt. Es wurden des Weiteren mit $\mathsf{FO}[\exists^{\infty}]$, $\mathsf{FO}[\exists^{\text{mod}}]$ und $\mathsf{FO}[\exists^{\text{ram}}]$ verschiedene Erweiterungen der Prädikatenlogik erster Stufe untersucht und deren Entscheidbarkeiten in automatischen Graphen nachgewiesen [1, 8]. In [5] sowie [6] wurde sogar die Entscheidbarkeit der sogenannten FSO-Logik, welche eine Einschränkung der Prädikatenlogik zweiter Stufe darstellt (Variablen zweiter Stufe dürfen relativ zum Quantoren nicht positiv in einer Formel vorkommen), bewiesen.

Für einige der Graphprobleme aus [2] wurde bereits die Entscheidbarkeit mithilfe der FSO-Logik oder aber auch die Unentscheidbarkeit für automatische Graphen nachgewiesen. In dieser Arbeit sollen nun eine Menge weiterer dieser Probleme auf ihre Entscheidbarkeit hin untersucht werden und in der arithmetischen bzw. analytischen Hierarchie eingeordnet werden.

2. Ergebnisse

Satz 2.1 (Perfektes Matching / EXACT COVER^{aut}) *Das folgende Problem ist Σ^1_1-vollständig:*
Eingabe: *Ein automatischer Graph $\mathcal{G} = (V, E)$.*
Frage: *Gibt es eine Menge $C \subseteq E$, sodass für jedes $v \in V$* genau *ein $u \in V \setminus \{v\}$ existiert mit $(v, u) \in C$ oder $(u, v) \in C$?*

Beweis. Betrachte das folgende nach [6, Theorem 2.6] Σ^1_1-vollständige Problem:
Eingabe: Ein automatischer Nachfolgerbaum $\mathcal{T} = (V, E, r)$.
Frage: Gibt es einen unendlichen Pfad in $\mathcal{T}$?
Dieses Problem soll nun auf EXACT COVER^{aut} reduziert werden. Ersetze hierzu im Nachfolgerbaum $\mathcal{T} = (V, E, r)$ jeden Knoten $v \in V \setminus \{r\}$ durch zwei Kopien $v, v\#$. Die Kanten aus $\mathcal{T}$ $(u, v) \in E$ werden zu $(u, \#v)$ und $(\#v, u)$. Zudem werden noch Kanten $(\#v, v), (v, \#v)$ für alle $v \in V \setminus \{r\}$ hinzugefügt.



Der konstruierte Graph besitzt genau dann eine exakte Überdeckung, wenn $\mathcal{T}$ einen unendlichen Pfad besitzt. □

Satz 2.2 (Erfüllbarkeit KNF-Formeln / SAT^{aut}) *Betrachte das folgende Problem:*
Eingabe: *Ein automatischer, bipartiter Graph $\mathcal{G} = (A, C, P, N)$ mit $P, N \subseteq A \times C$ (A sind die Atome, C die Klauseln, P und N positive bzw. negative Literalzuordnungen).*
Frage: *Gibt es eine Menge $S \subseteq A$, sodass für alle $c \in C$ ein $a \in S$ existiert mit $(a, c) \in P$ oder ein $a \in A \setminus S$ existiert mit $(a, c) \in N$?*

(i) *Dieses Problem ist Σ_1^1-vollständig.*
(ii) *Dieses Problem mit ausschließlich endlichen Klauseln ist Π_1^0-vollständig.*

Beweis.

(i) Reduziere EXACT COVER^{aut} auf SAT^{aut} wie folgt: Sei $\mathcal{G} = (V, E)$ automatischer Graph. Dann konstruiere folgende unendliche Formel und kodiere diese als automatischen, bipartiten Graphen:

$$\bigwedge_{v \in V} \bigvee_{\substack{u \in V: \\ (u,v) \in E}} (u, v) \wedge \bigwedge_{\substack{u,v \in V: \\ (u,v) \in E}} (u, v) \leftrightarrow (v, u)$$
$$\wedge \bigwedge_{\substack{u,v,w \in V, v \neq w: \\ (u,v),(u,w) \in E}} ((u, v) \rightarrow \neg(u, w) \wedge (u, w) \rightarrow \neg(u, v))$$

Diese Formel ist genau dann erfüllbar, wenn $\mathcal{G}$ ein Perfektes Matching besitzt.

(ii) Das Enthaltensein in Π_1^0 folgt aus dem Endlichkeitssatz aussagenlogischer Formeln. Die Π_1^0-Härte ergibt sich wie folgt: Reduziere PCP auf SAT^{aut} wie folgt: Sei dazu $I = ((x_1, y_1), \ldots, (x_n, y_n))$ eine PCP-Instanz über dem Alphabet Σ. Dann konstruiere die folgende unendliche Formel und kodiere diese als automatischen, bipartiten Graphen:

$$(\varepsilon, \varepsilon) \wedge \bigwedge_{u \in \Sigma^+} \neg(u, u) \wedge \bigwedge_{i=1}^{n} \bigwedge_{u,v \in \Sigma^*} (u, v) \rightarrow (ux_i, vy_i).$$

Diese Formel ist genau dann erfüllbar, wenn I keine PCP-Lösung besitzt.

□

Satz 2.3 (Graphfärbung / COLOR^{aut}) *Betrachte das folgende Problem:*
Eingabe: *Ein automatischer Graph $\mathcal{G} = (V, E)$, eine reguläre Menge von Farben C und eine Farbe $c_0 \in C$.*
Frage: *Ist $\mathcal{G}$ mit den Farben aus C färbbar (d.h. es existiert eine Abbildung $c\colon V \rightarrow C$, sodass für alle $u, v \in V$ mit $(u, v) \in E$ gilt: $c(u) \neq c(v)$), wobei die Farbe $c_0 \in C$ unendlich oft verwendet wird?*

(i) *Dieses Problem ist für unendliche Farbmengen entscheidbar.*
(ii) *Dieses Problem ist für Farbmengen mit $2 \leq |C| < \infty$ Π_1^0-vollständig.*

Beweis.

(i) Es genügt, zu überprüfen, ob eine unendliche Menge unabhängiger Knoten existiert. Dies kann mit der folgenden FSO-Formel realisiert werden:

$$\exists X \text{ infinite}\colon \forall x \forall y\colon ((X(x) \wedge X(y)) \to \neg E(x,y)).$$

Nach [5, 6] ist die FSO-Logik für automatische Strukturen entscheidbar. Für die restlichen Knoten $V \setminus X$ genügt eine injektive Abbildung $c'\colon V \setminus X \to C \setminus \{c_0\}$.

(ii) Es wird nur die Zweifärbbarkeit betrachtet. Verwende zur Reduktion erneut PCP. Sei dazu $I = ((x_1, y_1), \ldots, (x_n, y_n))$ eine PCP-Instanz. Konstruiere wie folgt einen Graphen $\mathcal{G} = (V, E)$: Zu jeder Indexfolge $(i_1, \ldots, i_m)$ mit $1 \leq i_1, \ldots, i_m \leq n$ und $m > 0$ konstruiere je einen Pfad

$$(x, \varepsilon) \to (x, x_{i_1}) \to (x, x_{i_1} x_{i_2}) \to \cdots \to (x, x_{i_1} \ldots x_{i_m})$$

sowie analog für $y, y_1, \ldots, y_n$. Die Wurzeln dieser beiden Pfade werden mit einem zusätzlichen Knoten verbunden. Zudem werden die Enden miteinander verbunden, wenn $x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$ gilt. Der Graph besitzt genau dann einen Kreis ungerader Länge (und ist deshalb nicht zweifärbbar), wenn I eine PCP-Lösung besitzt.

□

Literatur

[1] A. BLUMENSATH, E. GRÄDEL, Automatic structures. In: *Logic in Computer Science, 2000. Proceedings. 15th Annual IEEE Symposium on*. IEEE, 2000, 51–62.

[2] T. HIRST, D. HAREL, Taking it to the limit: on infinite variants of NP-complete problems. *Journal of Computer and System Sciences* **53** (1996) 2, 180–193.

[3] B. KHOUSSAINOV, A. NERODE, Automatic presentations of structures. In: *Logic and computational complexity*. Springer, 1995, 367–392.

[4] C. KÖCHER, Analyse der Entscheidbarkeit diverser Probleme in automatischen Graphen. Bachelor thesis, Technische Universität Ilmenau, 2014.

[5] D. KUSKE, Theories of automatic structures and their complexity. In: *Algebraic Informatics*. Springer, 2009, 81–98.

[6] D. KUSKE, M. LOHREY, Some natural decision problems in automatic graphs. *Journal of Symbolic Logic* **75** (2010) 2, 678–710.

[7] D. MARX, Graph colouring problems and their applications in scheduling. *Electrical Engineering* **48** (2004) 1-2, 11–16.

[8] S. RUBIN, Automata presenting structures: A survey of the finite string case. *Bulletin of Symbolic Logic* **14** (2008) 02, 169–209.

THEORIE-TAG 2014
H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 69–72.

Deterministic Set Automata

Martin Kutrib Andreas Malcher Matthias Wendlandt

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

1. Introduction

The regular languages and their corresponding automata model, deterministic and nondeterministic finite automata, are well investigated [4]. It is well known that this family of languages has many desirable properties. For example, all commonly studied decidability questions are decidable and the regular languages are closed under almost all commonly studied operations such as, for example, the Boolean operation, concatenation, (inverse) homomorphism, and substitution. From a practical point of view, finite automata are in particular interesting, since many of the decidability questions are decidable in polynomial time and, in addition, an effective minimization algorithm is known for deterministic finite automata.

But with respect to the computational power, this model is quite weak since it builds the lowermost level of the Chomsky hierarchy. Hence, much efforts have been made to find models that extend the computational power of regular languages by adding storage media, but keep as many 'good' properties as possible. Consider, for example, the extension by a stack [3] or by a pushdown store [1], which leads to the context-free languages. For both models nondeterministic variants are more powerful than deterministic variants, which is in contrast to finite automata. Moreover, some positive closure properties are lost. On the other hand, the decidability of emptiness and finiteness is preserved [3, 4]. In addition, equivalence is decidable for deterministic pushdown automata [6], but not for the nondeterministic variant [4].

The paper [2] introduces bag automata which are basically finite automata equipped with a finite number of bags in which the automaton can put symbols and also multiple versions of symbols. The symbols are stored as multisets and, therefore, the order in which they are added to the bags is not remembered. This model is quite powerful, because it is possible to simulate certain counter machines.

In this paper, we consider *set automata* (DSA) that extend deterministic finite automata by adding the storage medium of a set which, in contrast to bag automata, allows words to be stored and is not a multiset. As operations on the set it is possible to add elements, to remove elements, and to test whether some element is in the set. To prepare a set operation the DSA can write on a one-way write-only tape. For the set operation the contents of that tape are taken and added to the set, removed from the set, or tested. After the set operation, the writing tape is reset to the empty tape and a new set operation may be prepared. A similar model has been introduced

by Lange and Reinhardt in [5]. In contrast to DSA, their model may work nondeterministically, allows no remove operations, and a test operation implicitly adds the word tested to the set. The main results in [5] are the decidability of emptiness for the model considered and the closure of the corresponding language class under the operations homomorphism, inverse homomorphism, and intersection with regular languages.

We investigated the model of deterministic set automata according to the computational power and considered closure properties. We showed also that regularity for this model is decidable and get out of this results upper and lower bounds for the conversion to deterministic finite automata. Last we examined also decidability questions for nondeterministic set automata and considered conversion costs to other models.

2. Preliminaries

A set automaton is a system consisting of a finite state control, a one-way writing tape where transductions of parts of the input can be temporarily stored, and a data structure *set* where words of arbitrary length can be stored. At each time step, it is possible to either write a transduction of the current input letter to the end of the writing tape, to insert or remove the word written on the tape to or from the set, or to test whether the word written on the tape belongs to the set. Each time a set operation $\{\texttt{in}, \texttt{out}, \text{ or } \texttt{test}\}$ is done, the content of the writing tape is erased and its head is reset to the left end.

Definition 2.1 *A* deterministic set automaton *(*DSA*) is a system* $M = \langle S, \Sigma, \Gamma, \lhd, \delta, s_0, F\rangle$*, where*

1. S *is the finite set of* internal states*,*
2. Σ *is the finite set of* input symbols*,*
3. Γ *is the finite set of* tape symbols*,*
4. $\lhd \notin \Sigma$ *is the* right endmarker*,*
5. $s_0 \in S$ *is the* initial state*,*
6. $F \subseteq S$ *is the set of* accepting states*, and*
7. $\delta : S \times (\Sigma \cup \{\lambda, \lhd\}) \to (S \times (\Gamma^* \cup \{\texttt{in}, \texttt{out}\})) \cup (S \times \{\texttt{test}\} \times S)$ *is the partial transition function, where* $\texttt{in}$ *is the instruction to add the content of the tape to the set,* $\texttt{out}$ *is the instruction to remove the content of the tape from the set, and* $\texttt{test}$ *is the instruction to test whether or not the content of the tape is in the set. If the transition function is defined for some pair* (s, λ) *with* $s \in S$*, then it is not defined for any pair* (s, a) *with* $a \in \Sigma \cup \{\lhd\}$*.*

A *configuration* of a DSA $M = \langle S, \Sigma, \Gamma, \lhd, \delta, s_0, F\rangle$ is a quadruple $(s, v, z, \mathbb{S})$, where $s \in S$ is the current state, $v \in \{\Sigma^*\lhd\} \cup \{\lambda\}$ is unread part of the input, $z \in \Gamma^*$ is the content of the tape, and $\mathbb{S} \subseteq \Gamma^*$ is the finite set of stored words. The *initial configuration* for an input string w is set to $(s_0, w\lhd, \lambda, \emptyset)$. During the course of its computation, M runs through a sequence of configurations. One step from a configuration to its successor configuration is denoted by $\vdash$. Let $s, s', s'' \in S$, $x \in \Sigma \cup \{\lambda, \lhd\}$, $v \in \{\Sigma^*\lhd\} \cup \{\lambda\}$, $z, z' \in \Gamma^*$, and $\mathbb{S} \subseteq \Gamma^*$. We set

1. $(s, xv, z, \mathbb{S}) \vdash (s', v, zz', \mathbb{S})$, if $\delta(s, x) = (s', z')$,
2. $(s, xv, z, \mathbb{S}) \vdash (s', v, \lambda, \mathbb{S} \cup \{z\})$, if $\delta(s, x) = (s', \texttt{in})$,
3. $(s, xv, z, \mathbb{S}) \vdash (s', v, \lambda, \mathbb{S} \setminus \{z\})$, if $\delta(s, x) = (s', \texttt{out})$,
4. $(s, xv, z, \mathbb{S}) \vdash (s', v, \lambda, \mathbb{S})$, if $\delta(s, x) = (s', \texttt{test}, s'')$ and $z \in \mathbb{S}$,
5. $(s, xv, z, \mathbb{S}) \vdash (s'', v, \lambda, \mathbb{S})$, if $\delta(s, x) = (s', \texttt{test}, s'')$ and $z \notin \mathbb{S}$.

We denote the reflexive and transitive closure of $\vdash$ by $\vdash^*$. It should be noted that an instruction to remove some z from $\mathbb{S}$ does not test whether $z \in \mathbb{S}$; it only ensures that $z \notin \mathbb{S}$ after the operation. defined for the current configuration. The language accepted by the DSA M is the set $L(M)$ of words for which there exists a computation beginning in the initial configuration and ending in a configuration in which the whole input is read and an accepting state is entered. Formally:

$$L(M) = \{\, w \in \Sigma^* \mid (s_0, w\triangleleft, \lambda, \emptyset) \vdash^* (s_f, \lambda, z, \mathbb{S}) \text{ with } s_f \in F, z \in \Gamma^*, \mathbb{S} \subseteq \Gamma^* \,\}.$$

The family of all languages accepted by DSA is denoted by $\mathscr{L}(\text{DSA})$. The family of languages accepted by *nondeterministic set automaton* (NSA) is denoted by $\mathscr{L}(\text{NSA})$.

3. Computational Power

Theorem 3.1 *Every unary language accepted by a* DSA *is semilinear.*

Theorem 3.2 *The family of languages accepted by* DSA *is incomparable with the family of languages accepted by quasi-real-time queue automata.*

Theorem 3.3 *The family of languages accepted by* DSA *is incomparable with the (deterministic) context-free languages.*

Theorem 3.4 *The family of languages accepted by* DSA *is incomparable with the family of languages accepted by queue automata with finite turns.*

4. Closure Properties

Language class	$\cup$	$\cap$	$\sim$	$\cup_{\text{REG}}$	$\cap_{\text{REG}}$
REG	$+$	$+$	$+$	$+$	$+$
DCFL	$-$	$-$	$+$	$+$	$+$
$\mathscr{L}(\text{DSA})$	$-$	$-$	$+$	$+$	$+$
CFL	$+$	$-$	$-$	$+$	$+$

5. Decidability

Theorem 5.1 *It is decidable whether or not a given deterministic set automaton accepts the empty language.*

Theorem 5.2 *It is decidable whether or not a given deterministic set automaton accepts a regular language.*

Theorem 5.3 *For* NSA *the questions of universality, equivalence with regular sets, equivalence, inclusion, and regularity are not semi-decidable. Furthermore, it is not semi-decidable whether the language accepted by some* NSA *belongs to* $\mathscr{L}$(DSA).

6. Descriptional Complexity

Theorem 6.1 *Let* M *by an* n*-state* DSA *with set of tape symbols* Γ *that accepts a regular language. Then an equivalent* DFA *with at most* $2^{|\Gamma|^{O(n^2)}}$ *states can effectively be constructed.*

Theorem 6.2 *For* $n \geq 1$*, language* L_n *is accepted by an* $(n+2)$*-state* DSA*, but any equivalent* DFA *needs at least* 2^{2^n} *states.*

Corollary 6.3 *For every* $n \geq 1$*, there are regular languages* L_n *which are accepted by* $(n+2)$*-state* DSA *with tape alphabet* Γ *such that any equivalent* DFA *needs at least* $2^{|\Gamma|^n}$ *states.*

Theorem 6.4 *The following trade-offs are non-recursive. (1) From* NSA *to* DSA*. (2) From* NSA *to* DPDA*. (3) From* PDA *to* DSA*.*

References

[1] N. CHOMSKY, On certain formal properties of grammars. *Information and Control* **2** (1959), 137–167.

[2] M. DALEY, M.G. ERAMIAN, I. MCQUILLAN, The bag automaton: A model of nondeterministic storage. *Journal of Automata, Languages and Combinatorics* **13** (2008), 185–206.

[3] S. GINSBURG, S.A. GREIBACH, M.A. HARRISON, One-way stack automata. *Journal of the ACM* **14** (1967), 389–418.

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

[5] K.-J. LANGE, K. REINHARDT, Automaten mit der Datenstruktur Menge. In: M. KUTRIB, T. WORSCH, (eds.) *5. Theorietag Automaten und Formale Sprachen*. Universität Giessen, Giessen, Germany, 1995, 159–167.

[6] G. SÉNIZERGUES, $L(A) = L(B)$? decidability results from complete formal systems. *Theoretical Computer Science* **251** (2001), 1–166.

THEORIE-TAG 2014

H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 73–76.

Reconciling Decidability of Separation Logic Entailment and Graph Grammar Language Inclusion

Christoph Matheja

christoph.matheja@rwth-aachen.de

Abstract

We identify a syntactic fragment of hyperedge replacement grammars (HRG) that can be translated into sentences in monadic second-order logic (MSO) over graphs. It follows, that the language inclusion problem of the corresponding class of *tree-like* HRGs is decidable. Based on the close relationship between HRGs and separation logic with recursive definitions established by Jansen et al. [9], we show that the notion of tree-like HRGs extends recently studied fragments of separation logic with a decidable entailment problem by Iosif et al. [7], [8], which can be used to model dynamic data structures, like linked lists and trees.

1. Introduction

Although dynamic data structures are common in modern programming languages, they still pose a complex challenge for formal verification, because their use often leads to infinite state spaces. Thus, finite representations of infinite languages of memory states, i.e. heaps, are needed. Ideally, these finite representations have a decidable language inclusion problem such that automated verification is possible, i.e. it can be checked whether a language of heaps representing a dynamic data structure has a certain property (given as the language of all heaps with this property). Two formalisms to model languages of heaps (or arbitrary graphs) are *separation logic* (SL) [10] and *hyperedge replacement grammars* (HRG) [6]. As shown by Dodds [4] and Jansen et al. [9], there exist restrictions of SL and HRGs such that both formalisms realize a common fragment. The HRGs realizing only languages in this fragment are called *data structure grammars* (DSG) and the corresponding fragment of SL is denoted by SL_{HR}.

Similar equivalence results are well-known for languages of strings or trees: It is a classical result that the class of regular string (or tree) languages can be characterized in terms of finite automata, monadic-second order logic (MSO) and regular (tree) grammars. Furthermore, the entailment problem for two MSO formulae (over word or tree models) can be solved by checking the language inclusion problem for two equivalent finite automata. There are prominent examples of methods in formal verification, like automata-based LTL model-checking, that rely on such an equivalence result between automata and logic.

Unfortunately, the language inclusion problem for DSGs is undecidable, because every context-free string grammar can be translated into a DSG. By the close relationship between

DSGs and SL_{HR}, the same holds for the entailment problem of SL_{HR}. Thus, verification tools working with SL often consider only a small class of dynamic data structures to guarantee that the entailment problem remains decidable. For instance, PREDATOR [5] allows only the definition of list-like data structures.

Recently, Iosif et al. presented more general fragments of SL_{HR} with a decidable entailment problem, which are denoted by SL_{MSO} and local SL_{MSO} [7] [8]. However, they did not consider the link between SL_{HR} and DSGs. Thus, we study the question which conditions are necessary to ensure decidability of the language inclusion problem for HRGs and how the link between SL_{HR} and DSGs might support the development of verification algorithms. Furthermore, since decidability of the entailment problem is shown in [7] by a translation to MSO over graphs with bounded tree width, this question is related to an older problem: (How) can we lift the concept of regular languages to hypergraphs (cf. [1], [2], [3])?

2. Hyperedge Replacement Grammars

HRGs are akin to context-free string grammars in that they define the replacement of a nonterminal (labeled edge) by some hypergraph $\mathcal{H}$ equipped with a sequence of external vertices. These are used to merge vertices attached to the nonterminal with the graph $\mathcal{H}$. Formally, an HRG is a tuple $\mathfrak{G} = (N, T, P)$ consisting of finite sets N and T of nonterminal and terminal symbols and a finite set P of production rules of the form $\alpha \to \mathcal{A}$, where $\mathcal{A}$ is a hypergraph. The language of all hypergraphs that can be derived by an HRG $\mathfrak{G}$ with initial nonterminal $\xi \in N$ is denoted by $L(\mathfrak{G}, \xi)$. As an example, Figure 1 illustrates how the left nonterminal α of a hypergraph $\mathcal{A}$ is replaced using a production rule $\alpha \to \mathcal{A}$. In this figure, boxes denote edges labeled with nonterminals and vertices with numbers inside denote external vertices.

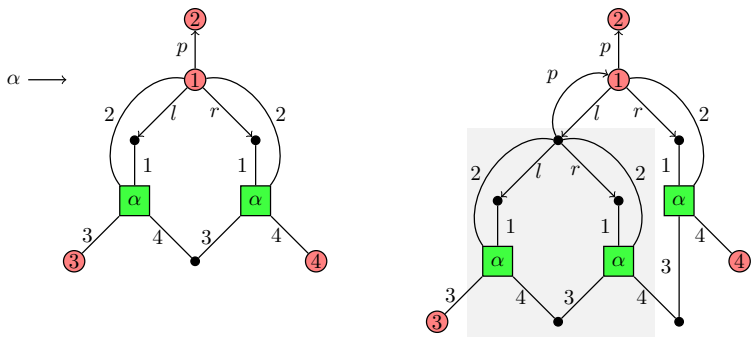


Figure 1: A production rule $\alpha \to \mathcal{A}$ and replacement of the left nonterminal α by $\mathcal{A}$

3. Tree-like Hyperedge Replacement Grammars

Our goal is to define a syntactic fragment of HRGs such that every HRG in this fragment can be translated into an equivalent MSO sentence. Since all graphs generated by an HRG have bounded tree width (cf. [6]) and the satisfiability problem for MSO over graphs of bounded tree width is decidable, it follows then that the language inclusion problem is decidable.

Informally, a hypergraph $\mathcal{H}$ is called *tree-like* if there exists a vertex $root_{\mathcal{H}}$ such that every vertex in $\mathcal{H}$ is either connected to nonterminals only or reachable from $root_{\mathcal{H}}$ by an undirected path containing no nonterminals. In addition to that, we require that every nonterminal must be connected to a vertex which is reachable from $root_{\mathcal{H}}$. These vertices are called the *context vertices*. Then, a *tree-like* HRG is an HRG where

1. every production rule maps to a tree-like hypergraph,

2. replacing every production rule $\alpha \to \mathcal{A}$ by a tree with $root_{\mathcal{A}}$ as root and the context vertices as leaves induces a regular tree grammar, and

3. for every derivation and every vertex v there exists at most one production rule that adds outgoing edges to v.

Intuitively, tree-like HRGs allow the reconstruction of a derivation tree for every hypergraph in a given language. This derivation tree can then be used to characterize the actual hypergraph in MSO. Analogously, the fragment of tree-like DSGs is obtained. For completeness, we state our main theorem.

Theorem 3.1 (MSO Definability of Tree-like HRGs) *For every tree-like HRG $\mathfrak{G} = (N, T, P)$ and every nonterminal $\xi \in N$, there exists an MSO sentence $\Psi^{\xi}_{\mathfrak{G}}$ such that for all finite hypergraphs $\mathcal{H}$ over T, it holds that $\mathcal{H} \in L(\mathfrak{G}, \xi)$ if and only if $\underline{\mathcal{H}} \models \Psi^{\xi}_{\mathfrak{G}}$.*

Tree-like DSGs are more expressive than the separation logic fragments studied by Iosif et al. [7], [8] and a fragment of separation logic equivalent to tree-like DSGs, denoted by $SL_{tree-like}$, can be defined. For example, the language of all binary trees in which the direction of all edges is reversed can be realized by a tree-like DSG, but is neither SL_{MSO} nor local SL_{MSO} definable. We can characterize the class of languages definable in SL_{MSO} by a restriction of tree-like DSGs to *directed* tree-like DSGs. In this fragment, for instance, the language of all binary trees that are additionally connected to their parents and collect their leaves in a singly-linked list from left to right is definable. This is not possible in local SL_{MSO}. Furthermore, the most general classes of graph languages we considered - MSO, SL, and the class $\mathcal{HRL}$ ($\mathcal{DSL}$) of languages realizable by HRGs (DSGs) - are incomparable. Figure 2 illustrates the relationships between these classes of graph languages. It is an open question, whether a language of connected MSO definable graphs that is also realizable by an HRG, but not by a tree-like HRG, exists.

There are some immediate consequences of the relationship between SL_{MSO} and directed tree-like DSGs. For example, since the emptiness problem for tree-like DSGs is decidable in polynomial time, the same follows for the satisfiability problem of SL_{MSO} formulae. However, since neither SL_{MSO} nor $SL_{tree-like}$ are closed under Boolean operations, this does not hold for the entailment / language inclusion problem. It can be shown that the entailment problem for $SL_{tree-like}$ and SL_{MSO} is EXPTIME-hard, but an upper bound is only known for local SL_{MSO} where Iosif et al. showed that the entailment problem is EXPTIME-complete [8].

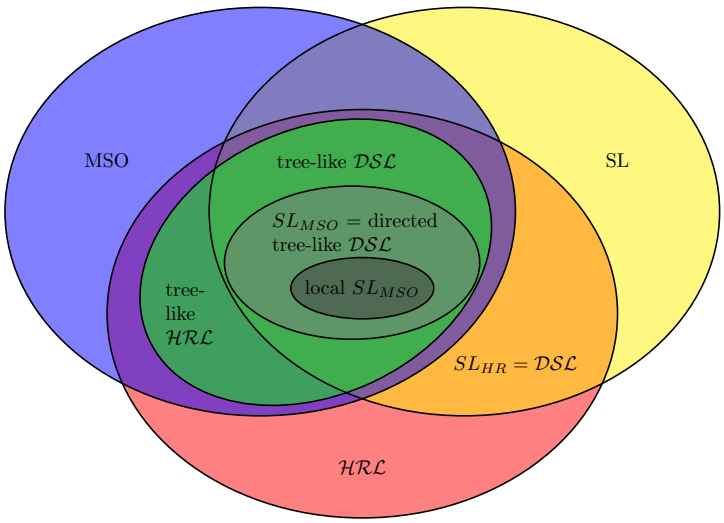


Figure 2: Relationships between classes of graph languages

References

[1] B. COURCELLE, The Monadic Second-Order Logic of Graphs I: Recognizable Sets of Finite Graphs. *Information and computation* **85** (1990) 1, 12–75.

[2] B. COURCELLE, The Monadic Second-Order Logic of Graphs V: On Closing the Gap between Definability and Recognizability. *Theoretical Computer Science* **80** (1991) 2, 153–202.

[3] B. COURCELLE, J. ENGELFRIET, *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. 138, Cambridge University Press, 2012.

[4] M. DODDS, From Separation Logic to Hyperedge Replacement and Back. In: *Graph Transformations*. Springer, 2008, 484–486.

[5] K. DUDKA, P. PERINGER, T. VOJNAR, Predator: A Practical Tool for Checking Manipulation of Dynamic Data Structures using Separation Logic. In: *Computer Aided Verification*. Springer, 2011, 372–378.

[6] A. HABEL, *Hyperedge Replacement: Grammars and Languages*. 643, Springer, 1992.

[7] R. IOSIF, A. ROGALEWICZ, J. SIMACEK, The Tree Width of Separation Logic with Recursive Definitions. In: *Automated Deduction–CADE-24*. Springer, 2013, 21–38.

[8] R. IOSIF, A. ROGALEWICZ, T. VOJNAR, Deciding Entailments in Inductive Separation Logic with Tree Automata. *arXiv preprint arXiv:1402.2127* (2014).

[9] C. JANSEN, F. GÖBE, T. NOLL, *Generating Inductive Predicates for Symbolic Execution of Pointer-Manipulating Programs*. Technical Report AIB-2014-08, RWTH Aachen University, 2014. `http://aib.informatik.rwth-aachen.de/2014/2014-08.pdf`

[10] J. C. REYNOLDS, Separation Logic: A Logic for Shared Mutable Data Structures. In: *Logic in Computer Science, 2002. Proceedings. 17th Annual IEEE Symposium on*. IEEE, 2002, 55–74.

THEORIE-TAG 2014

H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 77–79.

On the Descriptional Complexity of Deterministic Ordered Restarting Automata

Friedrich Otto

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

Abstract

We show that the stateless deterministic ordered restarting automaton is polynomially related in size to the weight-reducing Hennie machine. Accordingly, it allows very compact representations of (some) regular languages.

1. Summary

The restarting automaton was introduced in [1] as a formal device to model the linguistic technique of *analysis by reduction*. Since then many variants and extensions of the basic model have been introduced and studied (for an overview, see, e.g., [3]), and several classical families of formal languages have been characterized by certain types of restarting automata.

Here we report on the early stage of a project that studies the descriptional complexity of the *deterministic ordered restarting automaton* (or *det-ORWW-automaton*), which is a restricted type of restarting automaton that was introduced in [2] in the setting of picture languages. A det-ORWW-automaton has a finite-state control, a tape with end markers that initially contains the input, and a window of size three. Based on its state and the content of its window, the automaton can perform one of three types of operations: it may perform a *move-right step* that shifts the window one position to the right and changes the state, or it may perform a combined *rewrite/restart step* that replaces the symbol in the middle of the window by a symbol that is smaller with respect to a predefined ordering on the working alphabet, that moves the window back to the left end of the tape, and that resets the state to the initial state, or it may perform an *accept step* that causes the automaton to halt and accept. It has been shown in [2] that the nondeterministic variant of the ordered restarting automaton accepts some languages that are not even context-free, while the deterministic variant characterizes the regular languages.

First we show that each det-ORWW-automaton can be simulated by an automaton of the same type that has only a single state, which means that for these automata, states are not really

These results have been previously announced at DCFS 2014, Turku, Finland, August 2014. A corresponding contribution appeared in the proceedings of DCFS 2014 [4].

needed. Accordingly, we restrict our attention to the stateless det-ORWW-automaton (stl-det-ORWW-automaton). For such an automaton, we take the size of its working alphabet as the measure for its descriptional complexity, and we show that these automata are polynomially related in size to the weight-reducing Hennie machines studied by Průša in [5]. This implies that there is a double exponential trade-off when changing from a stl-det-ORWW-automaton to an equivalent deterministic finite-state acceptor (DFA).

If M_1 and M_2 are stl-det-ORWW-automata that accept languages L_1 and L_2, respectively, and if $\circ$ is a binary operation on languages, as e.g. union or intersection, then there exists a stl-det-ORWW-automaton M for the language $L_1 \circ L_2$. We provide upper bounds for the size of M in terms of the sizes of M_1 and M_2 for these operations and some others. For the operations of union and intersection, the bounds obtained are comparable to those for DFAs, while for the operation of reversal, we get a better bound as for DFAs. Finally, we provide upper bounds for the sizes of stl-det-ORWW-automata for languages of the form $L_1 \circ L_2$, where L_1 and L_2 are given through DFAs. Here we obtain better bounds for the operations of reversal, union, intersection, and product as for DFA s.

2. Open Problems

The stl-det-ORWW-automata provide more succinct representations for regular languages than DFAs, as the corresponding trade-off is double exponential. Also for the operations of reversal, union, and intersection, the representation by stl-det-ORWW-automata is much better suited than the representation by DFAs. However, many open problems remain:

- Are the given upper bounds sharp, that is, can they be obtained by example languages?
- How can two stl-det-ORWW-automata for L_1 and L_2 be combined into a 'small' stl-det-ORWW-automaton for the product $L_1 \cdot L_2$?
- How can a stl-det-ORWW-automaton for L be transformed into a 'small' stl-det-ORWW-automaton for L^*?
- What is the trade-off from nondeterministic ORWW-automata to stl-det-ORWW-automata?
- What is the complexity of deciding emptiness or finiteness for stl-det-ORWW-automata?

Finally, because of its window of size three, a stl-det-ORWW-automaton with an alphabet of size n can have $O(n^3)$ many transitions, while a DFA of size n over an alphabet of size k has at most $k \cdot n$ many transitions. Thus, when measuring the sizes of these types of automata in terms of their number of transitions, then the comparison is less favourable for the stl-det-ORWW-automaton.

References

[1] P. JANČAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, Restarting automata. In: H. REICHEL (ed.), *FCT'95, Proc.. LNCS 965*, Springer, Heidelberg, 1995, 283–292.

[2] 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.. LNCS 8327*, Springer, Heidelberg, 2014, 431–442.

[3] F. OTTO, Restarting automata. In: Z. ÉSIK, C. MARTÍN-VIDE, V. MITRANA (eds.), *Recent Advances in Formal Languages and Applications*. Studies in Computational Intelligence 25, Springer, Berlin, 2006, 269–303.

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

[5] D. PRŮŠA, Weight-reducing Hennie machines and their descriptional complexity. In: A.-H. DEDIU, C. MARTÍN-VIDE, J. SIERRA-RODRÍGUEZ, B. TRUTHE (eds.), *LATA 2014, Proc.. LNCS 8370*, Springer, Heidelberg, 2014, 553–564.

THEORIE-TAG 2014

H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 81–83.

Mittels Antimirovs Ableitungen zu gewichteten Baumautomaten

Philipp Schlag

Fachgebiet Automaten und Logik, Technische Universität Ilmenau
philipp.schlag@tu-ilmenau.de

Zusammenfassung

Eine Baumreihe r ist eine Abbildung von der Menge aller Bäume T_Σ über einem Rangalphabet Σ in einen kommutativen Semiring S. Wir betrachten das Problem, aus rationalen Baumreihenausdrücken E gewichtete Baumautomaten $\mathcal{A}_E$ zu konstruieren, welche die durch E beschriebene Baumreihe erkennen. Dazu verallgemeinern wir Antimirovs Begriff der partiellen Ableitungen von rationalen Ausdrücken auf rationale Baumreihenausdrücke.

Ausgehend von einem rationalen Baumreihenausdruck E geben wir konstruktive Definitionen für die Menge der Zustände und die Transitionsabbildungen des gewichteten Baumautomaten $\mathcal{A}_E$ an. Wir zeigen, dass die Anzahl der Zustände beschränkt ist durch die Anzahl von Vorkommen von Buchstaben sowie leeren Ausdrücken $\emptyset$ in E. Ebenso zeigen wir, dass die Anzahl der Transitionen mit Gewichten ungleich 0 beschränkt ist durch die Anzahl von Vorkommen von Buchstaben sowie leeren Ausdrücken $\emptyset$ in E multipliziert mit der Anzahl von Vorkommen von Buchstaben sowie a-Kleene-Sternen ^{*a} in E.

Themengebiet und Motivation

Baumautomaten als Beschreibungsmittel von Baumsprachen stellen nützliche Modelle zur qualitativen Analyse von Bäumen dar. So lässt sich beispielsweise feststellen, ob ein Baum ein bestimmtes Muster enthält. Mittels gewichteter Baumautomaten und Baumreihen können zusätzlich quantitative Merkmale untersucht werden, z.B. die Anzahl von Vorkommen dieses Musters im Baum. Dazu wird jeder Transition des gewichteten Baumautomaten ein Gewicht aus einem Semiring zugeordnet. Bei einem Lauf auf einem Baum werden alle Gewichte der genutzten Transitionen multipliziert. Gibt es mehrere Läufe auf demselben Baum, so werden die Gewichte der einzelnen Läufe schließlich addiert. Zahlreiche Anwendungsgebiete, z.B. die Verarbeitung natürlicher Sprache und Bildkompression, motivierten und motivieren weiterhin zur Beschäftigung mit zugehörigen Problemen der Theorie der formalen Sprachen und der Automatentheorie.

Ein prominentes Problem der Automatentheorie ist die Überführung eines rationalen Ausdrucks in einen endlichen Automaten. Solche Überführungen stellen konstruktive Beweise für eine Richtung von *Kleenes Theorem* [5] über die Äquivalenz rationaler Ausdrücke und endlicher Automaten als Beschreibungsmittel regulärer Sprachen dar. Schützenberger [10] zeigte als

analoges Ergebnis, dass die Menge der von gewichteten Automaten erkennbaren formalen Potenzreihen über einem beliebigen Semiring genau die Menge der rationalen formalen Potenzreihen über diesem Semiring ist. Thatcher und Wright [11] wiederum verallgemeinerten Kleenes Theorem auf rationale Baumausdrücke und Baumautomaten für Baumsprachen. Pech zeigte in [8] und [9], dass die Menge der von gewichteten Baumautomaten erkennbaren Baumreihen über einem beliebigen kommutativen Semiring genau die Menge der rationalen Baumreihen über diesem Semiring ist.

Für die Konstruktion eines endlichen Automaten aus einem rationalen Ausdruck existieren viele verschiedene Vorgehensweisen. Eine einfache Möglichkeit ist eine induktive Konstruktion. Allerdings kann es dabei bisweilen schwierig werden, die Größe des entstehenden Automaten - insbesondere die Anzahl seiner Transitionen - abzuschätzen. Eine weitere Möglichkeit ist die Konstruktions mittels *partieller Ableitungen*. Das Konzept der partiellen Ableitungen wurde von Antimirov [1] in Anlehnung an Brzozowskis Ableitungen [2] eingeführt, um aus einem rationalen Ausdruck einen nichtdeterministischen endlichen Automaten zu erzeugen. Lombardy und Sakarovitch [7] formulierten eine entsprechende Konstruktion für gewichtete Wortautomaten über starken Semiringen. Ebenso verallgemeinerten Kuske und Meinecke [6] Antimirovs Konstruktion auf rationale Baumausdrücke und Baumautomaten.

Wir betrachten rationale Baumreihenausdrücke wie von Fülöp und Vogler [4, Kap. 3.9] beschrieben. Eine induktive Variante der Konstruktion gewichteter Baumautomaten wurde von Droste, Pech und Vogler in [3, Kap. 6] angegeben. Wir verallgemeinern das Konzept der partiellen Ableitungen auf rationale Baumreihenausdrücke und gewinnen dadurch eine neue Möglichkeit, gewichtete Baumautomaten zu konstruieren.

Literatur

[1] V. ANTIMIROV, Partial derivatives of regular expressions and finite automaton constructions. *Theoretical Computer Science* **155** (1996) 2, 291–319.

[2] J. A. BRZOZOWSKI, Derivatives of Regular Expressions. *J. ACM* **11** (1964) 4, 481–494.

[3] M. DROSTE, C. PECH, H. VOGLER, A Kleene Theorem for Weighted Tree Automata. *Theory of Computing Systems* **38** (2005) 1, 1–38.

[4] Z. FÜLÖP, H. VOGLER, Weighted Tree Automata and Tree Transducers. In: M. DROSTE, W. KUICH, H. VOGLER (eds.), *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. An EATCS Series, Springer-Verlag, Berlin, Heidelberg, 2009, 313–403.

[5] S. C. KLEENE, Representation of events in nerve nets and finite automata. In: C. SHANNON, J. MCCARTHY (eds.), *Automata Studies*. Princeton University Press, Princeton, NJ, 1956, 3–41.

[6] D. KUSKE, I. MEINECKE, Construction of tree automata from regular expressions. *RAIRO - Theor. Inf. and Applic.* **45** (2011) 3, 347–370.

[7] S. LOMBARDY, J. SAKAROVITCH, Derivatives of rational expressions with multiplicity. *Theoretical Computer Science* **332** (2005) 1–3, 141–177.

[8] C. PECH, *Kleene type results for weighted tree automata*. Dissertation, Dresden University of Technology, 2003.

[9] C. PECH, Kleene's Theorem for Weighted Tree-Automata. In: A. LINGAS, B. J. NILSSON (eds.), *Fundamentals of Computation Theory*. Lecture Notes in Computer Science 2751, Springer Berlin Heidelberg, 2003, 387–399.

[10] M. P. SCHÜTZENBERGER, On the Definition of a Family of Automata. *Information and Control* **4** (1961) 2-3, 245–270.

[11] J. W. THATCHER, J. B. WRIGHT, Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic. *Mathematical Systems Theory* **2** (1968) 1, 57–81.

THEORIE-TAG 2014

H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 85–88.

Characterising REGEX Languages by Regular Languages with Factor-Referencing

Markus L. Schmid

Universität Trier, FB IV–Abteilung Informatikwissenschaften,
D-54286 Trier, Germany, MSchmid@uni-trier.de

Abstract

A (factor-)reference in a word is a special symbol that refers to another factor in the same word; a reference is dereferenced by substituting it with the referenced factor. We introduce and investigate the class ref-REG of all languages that can be obtained by taking a regular language R and then dereferencing all possible references in the words of R. We show that ref-REG coincides with the class of languages defined by regular expressions as they exist in modern programming languages like Perl, Python, Java, etc. (often called REGEX languages).

1. Introduction

Most natural languages contain at least some structure that cannot be described by context-free grammars and also with respect to artificial languages, e. g., programming languages, it is often necessary to deal with structural properties that are inherently non-context-free. The three most commonly encountered non-context-free features in formal languages are *reduplications*, leading to languages of the form $\{ww \mid w \in \Sigma^*\}$, *multiple agreements*, modelled by languages of the form $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 1\}$ and *crossed agreements*, as modeled by $\{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^n\mathtt{d}^m \mid n, m \geq 1\}$. In this work, we solely focus on the first such feature: reduplication. The concept of reduplication has been mainly investigated by designing language generators that are tailored to reduplications. A more recent approach is to extend regular expressions with some kind of copy operation. An interesting such variant are regular expressions with backreferences (REGEX for short), which play a central role in this work. REGEX are regular expressions that contain special symbols that refer to the word that has been matched to a specific subexpression. Unlike other language descriptors, REGEX seem to have been invented entirely on the level of software implementation, without prior theoretical formalisation. An attempt to formalise and investigate REGEX and the class of languages they describe from a theoretical point of view has been started recently (see [1, 3, 5, 4]). The widespread implementations of REGEX suggest that they are considered practically useful, but the theoretical investigation of the language class they describe proves to be complicated. Hence, we consider it worthwhile to develop a characterisation of this language class, which is independent from actual REGEX.

To this end, we introduce the concept of *unresolved* reduplications on the word level. In a fixed word, such a reduplication is represented by a reference to a factor of the word and dereferencing such a reference is done by replacing the pointer by the value it refers to, e. g.,

$$w = \mathtt{a}\underbrace{\mathtt{b}\mathtt{a}\overbrace{\mathtt{c}\mathtt{b}}^{y}}_{x}\mathtt{c}\overbrace{x\,\mathtt{c}\mathtt{b}}^{z}zy\mathtt{a},$$

where the symbols x, y and z are pointers to the factors marked by the brackets labelled with x, y and z, respectively. Resolving the references x and y yields $\mathtt{abacbcbaccb}z\mathtt{cba}$ and resolving reference z leads to $\mathtt{abacbcbaccbbaccbcba}$. Such words are called ref-words and sets of ref-words are ref-languages. For a ref-word w, $\mathcal{D}(w)$ denotes the word w with all references resolved and for a ref-language L, $\mathcal{D}(L) := \{\mathcal{D}(w) \mid w \in L\}$. We shall investigate the class of ref-regular languages, i. e., the class of languages $\mathcal{D}(L)$, where L is both regular and a ref-language, and, as our main result, we show that it coincides with the class of REGEX languages and the class of languages accepted by a natural extension of classical finite automata. This automaton model is used in order to introduce a subclass of REGEX languages, that, in contrast to other recently investigated such subclasses, has a polynomial time membership problem and we investigate the closure properties of this subclass.

2. Definitions

Let $\Gamma := \{[_{x_i},]_{x_i}, x_i \mid i \in \mathbb{N}\}$, where, for every $i \in \mathbb{N}$, the pairs of symbols $[_{x_i}$ and $]_{x_i}$ are *parentheses* and the x_i are *variables*. A *reference-word over* Σ (or *ref-word*, for short) is a word over the alphabet $(\Sigma \cup \Gamma)$. For each $i \in \mathbb{N}$, let $h_i : (\Sigma \cup \Gamma)^* \to (\Sigma \cup \Gamma)^*$ be the morphism with $h_i(z) := z$, $z \in \{[_{x_i},]_{x_i}, x_i\}$, $h_i(z) := \varepsilon$, $z \notin \{[_{x_i},]_{x_i}, x_i\}$. A reference word is *valid* if, for every $i \in \mathbb{N}$, $h_i(w) = x_i^{\ell_1}\,[_{x_i}]_{x_i}\,x_i^{\ell_2}\,[_{x_i}]_{x_i}\,x_i^{\ell_3}\ldots x_i^{\ell_{k_i-1}}\,[_{x_i}]_{x_i}\,x_i^{\ell_{k_i}}$, for some $k_i \in \mathbb{N}$, $\ell_j \in \mathbb{N}_0$, $1 \leq j \leq k_i$.

The set of valid reference-words is denoted by $\Sigma^{[*]}$. A factor $[_x\, u]_x$ of a $w \in \Sigma^{[*]}$ where the occurrences of $[_x$ and $]_x$ are matching parentheses is called a *reference for variable* x, and u is the *value* of this reference. A reference is a *first order reference*, if its value does not contain another reference and it is called *pure*, if it is a first order reference and its value does not contain variables. Two references of some ref-word w are *overlapping* if one reference contains exactly one of the delimiting parentheses of the other reference, e. g., in $w_1[_x w_2[_y w_3]_x w_4]_y w_5$ the references $[_x w_2[_y w_3]_x$ and $[_y w_3]_x w_4]_y$ are overlapping. Let $w \in \Sigma^{[*]}$ and let x be a variable that occurs in w. An occurrence of a variable x in w that is not preceded by a reference for x is called *undefined*. Every occurrence of a variable x in w that is not undefined *refers* to the reference for x, which precedes this occurrence. Next, we define how a ref-word over Σ can be dereferenced, i. e., how it can be transformed into a (normal) word over Σ. To this end, let $w \in \Sigma^{[*]}$. The *dereference* of w, denoted by $\mathcal{D}(w)$, is constructed by first deleting all undefined occurrences of variables in w and then substituting all pure references and their variables by its value (ignoring possible parentheses in the value), until there is no pure reference left.

For every $i \in \mathbb{N}$, let $\Gamma_i := \{[_{x_j},]_{x_j}, x_j \mid j \leq i\}$. A set of ref-words L is a *ref-language* if $L \subseteq (\Sigma \cup \Gamma_i)^*$, for some $i \in \mathbb{N}$. For the sake of convenience, we write $L \subseteq \Sigma^{[*]}$ to denote that L is a ref-language. For every ref-language L, we define the *dereference of* L by $\mathcal{D}(L) := \{\mathcal{D}(w) \mid w \in L\}$ and, for any class $\mathfrak{L}$ of ref-languages, $\mathcal{D}(\mathfrak{L}) := \{\mathcal{D}(L) \mid L \in \mathfrak{L}\}$. An $L \subseteq \Sigma^{[*]}$ is a *regular ref-language* if L is regular. A language L is called *ref-regular* if it is the dereference

of a regular ref-language, i. e., $L = \mathcal{D}(L')$ for some regular ref-language L'. For example, the copy language $L_c := \{ww \mid w \in \Sigma^*\}$ is ref-regular, since $L_c = \mathcal{D}(L'_c)$, where L'_c is the regular ref-language $\{[_x\, w\,]_x\, x \mid w \in \Sigma^*\}$. We denote the class of ref-regular languages by ref-REG.

If we use the concept of references directly in regular expressions, i. e., we use variables x in the expression and enclose subexpressions by parentheses $[_x$ and $]_x$, then we obtain *extended regular expressions with backreferences* (or REGEX for short). For more detailed definitions and further information on REGEX, we refer to [1, 5, 4, 2]. A convenient definition of the semantics of a REGEX can also be given in terms of classical regular expressions and ref-words. We can interpret a REGEX r as a classical regular expression r' over the alphabet $(\Sigma \cup \Gamma_k)$, where k is the number of backreferences in r. Now $L(r')$ is a ref-language and $\mathcal{D}(L(r'))$ is the REGEX language described by the REGEX r. This observation shows that $\mathcal{L}(\text{REGEX}) \subseteq$ ref-REG. On the other hand, a regular expression s with $L(s) \subseteq \Sigma^{[*]}$ does not translate into a REGEX in an obvious way, which is due to the fact that in s it is not necessarily the case that every occurrence of $[_x$ matches with an occurrence of $]_x$ and, furthermore, even matching pairs of parentheses do not necessarily enclose subexpressions. We say that a regular expression r over the alphabet $(\Sigma \cup \Gamma_k)$ has the REGEX *property* if it is also a valid REGEX.

3. Memory Automata

A memory automaton is an NFA equipped with a finite number of k memory cells, each capable of storing a word. Each memory is either *closed*, which means that it is not affected in a transition, or *open*, which means that it records the currently scanned input symbol. In a transition it is possible to *consult* a closed memory, which means that its content, if it is a prefix of the remaining input, is consumed in one step from the input and, furthermore, also stored in all the open memories. A closed memory can be opened again, but then it completely loses its previous content; thus, memories always store factors of the input. For any $k \in \mathbb{N}$, MFA(k) is the class of k-memory automata and MFA $:= \bigcup_{k \geq 0}$ MFA(k). We also use MFA(k) and MFA in order to denote an instance of a k-memory automaton or a memory automaton with some number of memories. The set $\mathcal{L}(\text{MFA}) := \{L(M) \mid M \in \text{MFA}\}$ is the *class of* MFA *languages*.

A k-memory automaton M is *pseudo deterministic* if it is never the case that several transitions that read the same symbol or consult the same memory are applicable and M is *deterministic* if it is ε-free and it is not possible that several different transitions are applicable at the same time. We denote the class of deterministic k-memory automata by DMFA(k) and DMFA $:= \bigcup_{k \in \mathbb{N}}$ MFA(k).

A memory automaton is in *normal form* if every transition can either read a part of the input without changing the status of any memory or it changes the status of exactly one memory, but then it does not touch the input. Furthermore, in every accepting configuration, the automaton does not try to open (close) memories that are already opened (closed, respectively). An MFA is *nested*, if there is no computation in which a memory i is opened before memory j and also closed before memory j.

Theorem 3.1 *Let $k \in \mathbb{N}$. For every $M \in$ MFA(k) there exists a $M' \in$ MFA(k^2) with $L(M) = L(M')$ that is pseudo-deterministic, nested and in normal form.*

4. Main Results

Let NFA_{ref} be the set of NFA that accept ref-languages and let MFA_{nf} be the set of MFA in normal form.

Lemma 4.1 *There exists an isomorphism* $\psi_{\mathcal{D}} : \text{NFA}_{ref} \to \text{MFA}_{nf}$ *such that, for every* $M \in \text{NFA}_{ref}$ *and* $N \in \text{MFA}_{nf}$, $\mathcal{D}(L(M)) = L(\psi_{\mathcal{D}}(M))$ *and* $L(N) = \mathcal{D}(L(\psi_{\mathcal{D}}^{-1}(N)))$.

We can show that ref-REG is included in $\mathcal{L}(\text{REGEX})$ as follows. Let M be a nested $\text{MFA}(k)$ in normal form and let $N := \psi_{\mathcal{D}}^{-1}(M)$. We can now transform N into a regular expression r that satisfies the REGEX property, i. e., in r every $[_{x_i}$ matches a $]_{x_i}$ and such matching parentheses enclose a subexpression. This is done in such a way that, for each pair of transitions $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$ and $\delta(r_{i,\ell},]_{x_i}) \ni s_{i,\ell}$, we transform the set $R_{i,j,\ell}$ of words that take N from $q_{i,j}$ to $r_{i,\ell}$ into a regular expression individually. For the correctness of this construction, it is crucial that M is nested and that we transform the $R_{i,j,\ell}$ into regular expressions in the right order.

These considerations directly imply the following result.

Theorem 4.2 ref-REG $= \mathcal{L}(\text{MFA}) = \mathcal{L}(\text{REGEX})$.

We now take a closer look at the class of languages accepted by DMFA.

Theorem 4.3 $\mathcal{L}(\text{DMFA}) \subset \mathcal{L}(\text{MFA})$. *For an* $M \in \text{DMFA}$ *and* $w \in \Sigma^*$, *we can decide in time* $\text{O}(|w|)$ *whether* $w \in L(M)$. $\mathcal{L}(\text{DMFA})$ *is closed under complementation and intersection with regular languages, but it is not closed under union or intersection.*

References

[1] C. CÂMPEANU, K. SALOMAA, S. YU, A formal study of practical regular expressions. *Int. Journal of Foundations of Computer Science* **14** (2003), 1007–1018.

[2] C. CÂMPEANU, N. SANTEAN, On the intersection of regex languages with regular languages. *Theoretical Computer Science* **410** (2009), 2336–2344.

[3] C. CÂMPEANU, S. YU, Pattern expressions and pattern automata. *Information Processing Letters* **92** (2004), 267–274.

[4] D. D. FREYDENBERGER, Extended Regular Expressions: Succinctness and Decidability. *Theory of Computing Systems* **53** (2013), 159–193.

[5] M. L. SCHMID, Inside the Class of REGEX Languages. *International Journal of Foundations of Computer Science* **24** (2013), 1117–1134.

THEORIE-TAG 2014

H. Bordihn, B. Truthe (Eds.): Theorietag 2014, Caputh, 23. – 25.09.2014
Universität Potsdam, Technical Report TR-2014-2, pp. 89–92.

A Relation Between Definite and Ordered Finite Automata

Bianca Truthe

Justus-Liebig-Universität Gießen, Institut für Informatik
Arndtstr. 2, D-35392 Gießen, Germany
bianca.truthe@informatik.uni-giessen.de

Abstract

We show that every definite language can be represented as a union of finitely many languages which are accepted by ordered deterministic finite automata. A consequence of this result is that every contextual grammar with definite selection languages generating a language in the external or internal derivation mode can be simulated by a contextual grammar working in the same derivation mode where every selection language is accepted by an ordered deterministic finite automaton.

1. Introduction

In the area of formal languages and automata theory, regular languages and finite automata are widely studied. Several classes of specific finite automata and their accepted languages have been investigated separately, for example, definite automata by M. Peres, M. O. Rabin, and E. Shamir in [4] and ordered automata by H. J. Shyr and G. Thierrin in [7].

In the present paper, we consider the subregular families of the definite languages and of the ordered languages. A definite automaton is a finite automaton where the acceptance of an input word depends only on its last k letters for a fixed natural number k depending on the automaton. The languages accepted by definite finite automata are called definite languages. Any definite language can be written in the form $A \cup V^*B$ for an alphabet V and two finite languages A and B over the alphabet V. An ordered automaton is a deterministic finite automaton where an order of the set of states exists such that the respective semi-order is preserved under the transitions. The languages accepted by ordered finite automata are called ordered languages.

We show that every definite language can be represented as a union of finitely many ordered languages and use this result to show that contextual grammars ([3]) with definite selection languages can be simulated by contextual grammars where all selection languages are ordered.

2. Definitions

We assume that the reader is familiar with the basic concepts of formal language theory (see e. g. [6]). We only recall here some notations used in the paper.

An alphabet is a non-empty and finite set of letters. For an alphabet V, we denote by V^* the set of all words over the alphabet V. The empty word is denoted by λ. The cardinality of a set A is denoted by $|A|$.

Let L be a language over an alphabet V. We say that the language L – with respect to the alphabet V – is definite if and only if it can be represented in the form $L = A \cup V^*B$ where A and B are finite subsets of V^* and ordered if and only if the language L is accepted by some deterministic finite automaton $\mathcal{A} = (V, Z, z_0, F, \delta)$ where $(Z, \preceq)$ is a totally ordered set and, for any $a \in V$, the relation $z \preceq z'$ implies $\delta(z,a) \preceq \delta(z',a)$.

The families of the definite languages and of the ordered languages are denoted by *DEF* and *ORD*, respectively.

Let $\mathcal{F}$ be a family of languages. A *contextual grammar with selection in* $\mathcal{F}$ is a triple $G = (V, \mathcal{P}, A)$ where

- V denotes an alphabet,
- $\mathcal{P}$ is a finite set of pairs (S,C) with a language S over the alphabet V which belongs to the family $\mathcal{F}$ with respect to the alphabet of the language S (which is a subset of the alphabet V) and a finite set $C \subset V^* \times V^*$,
- A denotes a finite subset of V^*.

The set V is called the basic alphabet; for a selection pair $(S,C) \in \mathcal{P}$, the language S is called a selection language and the set C is called a set of contexts of the grammar G; the elements of A are called axioms. In a contextual grammar with selection in *DEF* or *ORD*, every selection language is definite or ordered with respect to its (minimal) alphabet (but not necessarily with respect to the total alphabet V of the grammar).

A direct external derivation step is defined as follows: a word x derives a word y if and only if there is a pair $(S,C) \in \mathcal{P}$ such that $x \in S$ and $y = uxv$ for some pair $(u,v) \in C$. A direct internal derivation step is defined as follows: a word x derives a word y if and only if there are words $x_1, x_2, x_3 \in V^*$ such that $x = x_1x_2x_3$ and a pair $(S,C) \in \mathcal{P}$ such that $x_2 \in S$ and $y = x_1ux_2vx_3$ for some pair $(u,v) \in C$.

The language generated externally or internally by a grammar G is the set of all words derived from an axiom in finally many external or internal derivation steps, respectively.

By $\mathcal{EC}(\mathcal{F})$ and $\mathcal{IC}(\mathcal{F})$, we denote the family of all languages generated externally and internally, respectively, by contextual grammars with selection in $\mathcal{F}$.

3. Results

We first state the main theorem and then use it to prove that every contextual grammar where the selection languages are definite can be simulated by a contextual grammar where every selection language is accepted by an ordered deterministic finite automaton.

Theorem 3.1 *Every definite language is the union of finitely many ordered languages.*

Proof. Let V be an alphabet, $m \geq 0$ be a natural number, and $w_1, w_2, \ldots, w_m$ be words over the alphabet V. Further, let $B = \{w_1, w_2, \ldots, w_m\}$ and let A also be a finite language over the alphabet V. The definite language $L = A \cup V^*B$ is also the union

$$L = A \cup V^*\{w_1\} \cup V^*\{w_2\} \cup \cdots \cup V^*\{w_m\}.$$

Every finite language, hence also the language A, is ordered ([7]).

We now show that, for every word $w \in V^*$, the language $V^*\{\, w \,\}$ is accepted by an ordered deterministic finite automaton. If $w = \lambda$, then the language $V^*\{\, w \,\}$ is equal to the language V^* which is accepted by a deterministic finite automaton with a single state, hence by an ordered deterministic finite automaton.

Now let w be a word with at least one letter and let $x_1, x_2, \ldots, x_n$ for a natural number $n \geq 1$ be the letters of the alphabet V such that $w = x_1 x_2 \cdots x_n$. By $\overline{x_i}$ for $1 \leq i \leq n$, we denote the complementary set $V \setminus \{\, x_i \,\}$.

In the following figure, we show the transition graph of a non-deterministic finite automaton $\mathcal{A}$ which accepts the language $V^*\{\, w \,\}$. We order the states along the strong line beginning at the left end and denote them in this order by

$$z_{0,0}, z_{1,0}, z_{2,0}, z_{2,1}, z_{3,0}, z_{3,1}, z_{3,2}, \ldots, z_{i,0}, \ldots, z_{i,i-1}, \ldots, z_{n-1,0}, \ldots, z_{n-1,n-2}, z_{n,0}, z_{n,1}.$$

The set of these states is denoted by Z; the order mentioned above by $\prec$ and the corresponding semi-order by $\preceq$ (for any two states p and q, we have $p \preceq q$ if and only if $p \prec q$ or $p = q$).

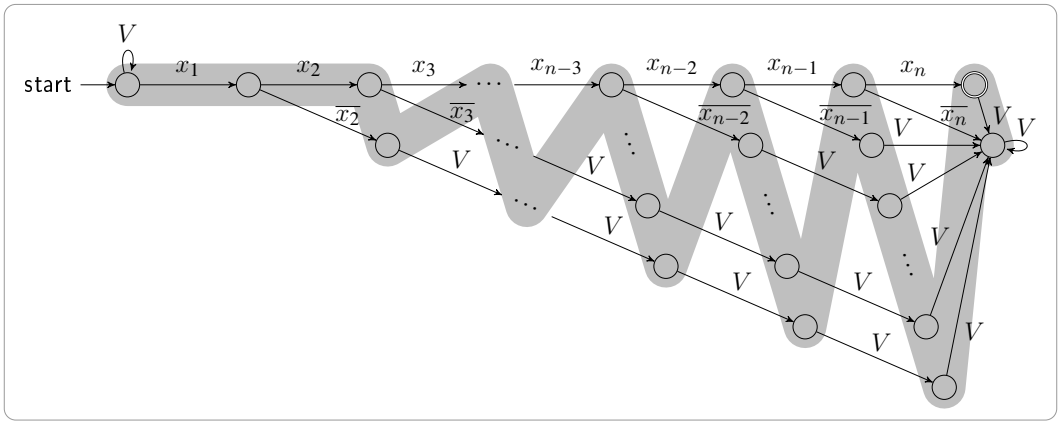


Figure 1: NFA $\mathcal{A}$ accepting the language $V^*\{\, w \,\}$ with an order of the states

Let $\mathcal{A}_{\mathrm{d}} = (V, Z_{\mathrm{d}}, \{\, z_{0,0} \,\}, F_{\mathrm{d}}, \delta_{\mathrm{d}})$ be the equivalent deterministic finite automaton obtained by the power-set construction ([5], see also [6]) from the automaton $\mathcal{A}$.

Note that the semi-order of the states of the set Z is preserved by the mapping δ.

We order the states of the set Z_{d} 'lexicographically' in the following way:

For two different sets $R \in Z_{\mathrm{d}}$ and $S \in Z_{\mathrm{d}}$, we say that $R \prec S$ if and only if there is a natural number $m \geq 0$ such that $m \leq |R|$ and $m < |S|$ and $z_{R,k} = z_{S,k}$ for $1 \leq k \leq m$ and $m < |R|$ implies $z_{R,m+1} \prec z_{S,m+1}$.

Hence, the deterministic finite automaton $\mathcal{A}_{\mathrm{d}}$ is ordered and accepts the language $V^*\{\, w \,\}$. This completes the proof that every definite language is the union of finitely many ordered languages. □

The Theorem 3.1 implies that contextual grammars where all selection languages are definite can be simulated by contextual grammars where all selection languages are ordered.

Lemma 3.2 *The relations $\mathcal{EC}(DEF) \subset \mathcal{EC}(ORD)$ and $\mathcal{IC}(DEF) \subset \mathcal{IC}(ORD)$ hold.*

Proof. Let $G = (V, \{ (S_i, C_i) \mid 1 \leq i \leq n \}, A)$ be a contextual grammar where all the selection languages S_i for $1 \leq i \leq n$ are definite languages. According to Theorem 3.1, for every language S_i for $1 \leq i \leq n$, there are a natural number $n_i \geq 1$ and ordered languages $S_{i,1}, \ldots, S_{i,n_i}$ such that

$$S_i = S_{i,1} \cup \cdots \cup S_{i,n_i}.$$

Then, the contextual grammar $G' = (V, \{ (S_{i,j}, C_i) \mid 1 \leq i \leq n, 1 \leq j \leq n_i \}, A)$ generates the same language as the grammar G and has only selection languages which are accepted by ordered automata.

This proves the relations $\mathcal{EC}(DEF) \subseteq \mathcal{EC}(ORD)$ and $\mathcal{IC}(DEF) \subseteq \mathcal{IC}(ORD)$. A witness language for the properness of the first inclusion is

$$L_1 = \{ a^n b^n \mid n \geq 1 \} \cup \{ b^n \mid n \geq 1 \},$$

of the second inclusion $L_2 = L(G_2)$ with

$$G_2 = (\{a,b,c,d,e\}, \{ (\{d\}^* \{b\} \cup \{\lambda\}, \{(a,b)\}), (\{a,\lambda\}, \{(c,d)\}) \}, \{ecadb\}).$$

The language L_1 can be generated by the contextual grammar

$$G_1 = \Big(\{a,b\}, \Big\{ (\{b\}^* \{a\} \{a,b\}^*, \{(a,b)\}), (\{b\}^+, \{(\lambda,b)\}) \Big\}, \{ab,b\} \Big)$$

but not by a contextual grammar with definite selection languages ([1]). In [2], it was shown that $L_2 \notin \mathcal{IC}(DEF)$. In [8], it was shown that $L_2 \in \mathcal{IC}(ORD)$. □

References

[1] J. DASSOW, Contextual grammars with subregular choice. *Fundamenta Informaticae* **64** (2005) 1–4, 109–118.

[2] J. DASSOW, F. MANEA, B. TRUTHE, On Subregular Selection Languages in Internal Contextual Grammars. *Journal of Automata, Languages, and Combinatorics* **17** (2012) 2–4, 145–164.

[3] S. MARCUS, Contextual grammars. *Revue Roum. Math. Pures Appl.* **14** (1969), 1525–1534.

[4] M. PERES, M. RABIN, E. SHAMIR, The Theory of Definite Automata. *IEEE Transactions on Electronic Computers* **12** (1963) 3, 233–243.

[5] M. RABIN, D. SCOTT, Finite Automata and Their Decision Problems. *IBM Journal of Research and Development* **3** (1959) 2, 114–125.

[6] G. ROZENBERG, A. SALOMAA (eds.), *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.

[7] H. SHYR, G. THIERRIN, Ordered Automata and Associated Languages. *Tankang Journal of Mathematics* **5** (1974) 1, 9–20.

[8] B. TRUTHE, A Relation Between Definite and Ordered Finite Automata. In: *Sixth Workshop on Non-Classical Models of Automata and Applications (NCMA), Kassel, Germany, July 28–29, 2014, Proceedings*. books@ocg.at 304, Österreichische Computer Gesellschaft, Austria, 2014, 235–247.