17. Theorietag
Automaten und Formale Sprachen

27. – 29. September 2007, Leipzig

Manfred Droste und Markus Lohrey (Hrsg.)

Institut für Informatik
Universität Leipzig

17. Theorietag
Automaten und Formale Sprachen

Leipzig, 27. – 29. September 2007

Vorwort

Seit 1991 wird von der GI-Fachgruppe 0.1.5 *Automaten und Formale Sprachen* jährlich der Theorietag veranstaltet; im Laufe des Theorietags findet auch die jährliche Fachgruppensitzung statt. Die bisherigen Theorietage fanden in Magdeburg (1991), Kiel (1992), Dagstuhl (1993), Herrsching (1994), Schloss Rauischholzhausen (1995), Cunnersdorf (1996), Barnstorf (1997), Riveris (1998), Schauenburg-Elmshagen (1999), Wien (2000), Wendgräben (2001), Wittenberg (2002), Herrsching (2003), Potsdam (2004), Lauterbad (2005) und Wien (2006) statt.

Nach Cunnersdorf 1996 findet der Theorietag "Automaten und formale Sprachen" also bereits elf Jahre später wieder in Sachsen statt. Die Vorträge werden im neuen Senatssaal des Rektorats der Universität Leipzig gehalten.

Am ersten Tag des 17. Theorietags, dem 27. September 2007, werden von

- Christel Baier (Dresden)
- Didier Caucal (Paris)
- Volker Diekert (Stuttgart)
- Daniel Kirsten (Leipzig)
- Jacques Sakarovitch (Paris)
- Géraud Sénizergues (Bordeaux)

eingeladene Vorträge gehalten. Die nächsten eineinhalb Tage werden durch Beiträge der Teilnehmerinnen und Teilnehmer des Theorietags gestaltet.

Den Sponsoren sei für ihre Unterstützung herzlich gedankt.

Wir wünschen allen Teilnehmerinnen und Teilnehmern des 17. Theorietags eine interessante und anregende Tagung und einen schönen und angenehmen Aufenthalt in Leipzig.

Leipzig, im September 2007

Manfred Droste
Markus Lohrey

Inhalt

Eingeladene Vorträge

Beiträge zum Theorietag

Eingeladene Vorträge

Probabilistic Omega-Automata

Christel Baier
Institut für Theoretische Informatik
Technische Universität Dresden, Germany

This talk considers probabilistic automata as acceptors for languages over infinite words and discusses some fundamental aspects, such as expressiveness, efficiency, composition operators and decision problems. The first observation is that probabilistic Buechi automata (PBA) with the acceptance condition that the probability measure of the accepting runs is positive are more expressive than non-deterministic omega-automata. However, a certain structural property of PBA (called uniformity) can be identified that yields exactly the power of omega-regular languages. The same holds for probabilistic Streett or Rabon automata. Concerning the efficiency, probabilistic omega-automata are not comparable with their nondeterministic counterparts, as there are omega-regular languages that have uniform PBA of linear size, while any nondeterministic Streett automaton is of exponential size, and vice versa. Moreover, there are instances of verification problems for Markov chains against linear-time properties that have a simple and efficient solution when assuming a PBA-representation of the property, while corresponding algorithms with nondeterministic automata rely on certain powerset constructions and have exponential time complexity. On the other hand, the emptiness problem for PBA is undecidable, which yields the undecidability of, e.g., the model checking problem for Markov decision processes and PBA-specifications or stochastic games with Buechi winning objectives and observation-based strategies.

From Finite Automata to Deterministic Graph Grammars

Didier Caucal
IGM-CNRS, University of Paris-East
5 bd Descartes, 77454 Marne-la-Vallé, France
caucal@univ-mlv.fr

Context-free grammars are a very prominent tool in the field of language theory. A similar notion can be adapted to the more general setting of grammars generating graphs instead of words. In this case, grammar rules no longer express the replacement of a non-terminal letter by a string of terminal and non-terminal letters, but that of a non-terminal arc by a finite graph possibly containing new non-terminals, thus generating larger and larger graphs. Here we are concerned with the specific setting where the considered sets of grammar rules are deterministic, meaning that there is only one production rule for every non-terminal hyperarc. Consequently, from a given axiom, a grammar does not generate a set of graphs, but a unique graph up to isomorphism called a regular graph. These regular graphs were first considered by Muller and Schupp: they showed that the connected components of the transition graphs of pushdown automata are the connected regular graphs of finite degree. In this respect, graph grammars provide a natural and powerful tool to reason about context-free languages. They are not only finite representations of infinite graphs whose structure is regular but they are also to context-free languages what finite automata are to regular languages.

We give a survey of deterministic graph grammars and the class of graphs they generate, and we present two recent applications. A first application is the synchronization of grammars. For each grammar, the sets of path labels associated to the graphs generated by the synchronized grammars form an effective boolean algebra of deterministic context-free languages and containing the regular languages. This is just a simple extension on graph grammars of the usual constructions on finite automata. A second application is to use grammars to efficiently extend standard algorithms from (finite) graph theory to the class of regular graphs, like for instance shortest path algorithms.

Equations: From Words to Graph Products

Volker Diekert
FMI, Universität Stuttgart,
Universitätsstr. 38, D-70569 Stuttgart.
diekert@fmi.uni-stuttgart.de

A word equation is a simple object: It is just a pair of words in constants and unknowns (also called variables). The pair is written as an equation and the question is whether or we can replace the unknowns with words over the constants such that the equation becomes an identity. In the positive case we say that the equation is solvable.

Example: Let x, y, z, u be unknowns. Consider

$$abxauzaubababax = yyzbxaabyzzaabb.$$

This is a solvable equation. A possible solution is given by:

$$x = abb, \quad y = ab, \quad z = ba, \quad u = bab.$$

Instead of a single equation we can also consider systems of equations. But this does not change very much the picture, because there are techniques which transform any Boolean system of equations (including negations) into a single equation. Thus, deciding an existential sentence in the theory of free monoids can be reduced to the problem of solving a single word equation. Hence, the challenge is to find a method for solving a single equation.

In the late 1960's Russian mathematicians became interested in this challenge due to the observation that there is a reduction from the existential theory in free monoids to Hilbert's Tenth Problem. The tempting idea was to show undecidability of Hilbert 10 via undecidability of *Word Equations*. This program failed, but it led to major results. Hilbert's Tenth Problem was shown to be undecidable in 1970 by Matiyasevich using completely different methods, see [13] for his book on this subject. Seven years later, in 1977 Makanin presented an algorithm which solves word equations [10]. The reduction of *Word Equations* to polynomial equations over the integers is nevertheless still interesting. For example, as observed by Guba [5], this reduction and Hilbert's Basis Theorem yield a (positive) solution of the so-called Ehrenfeucht conjecture in formal language theory. The reduction is also the key for a simple randomized parallel algorithm for pattern matching (via fingerprints).

The original algorithm of Makanin is extremely complex, early time estimations mentioned some tower of exponentials; and it took another 22 years until Plandowski found 1999 a less complex method [14]. In fact, Plandowski showed that *Word Equations* can be decided in PSPACE, thus in single exponential time. This is the best known result although it is strongly conjectured that *Word Equations* is NP-complete. This is a dramatic change view: First the problem was believed to be undecidable; and when Makanin showed decidability, it was so complex and there was few hope to lower the complexity. Today, we believe that solving word equations is, in principle, as easy or difficult as finding a truth assignment to Boolean formulae.

There is a simple strategy for solving word equations where every variable occurs at most twice, so called *quadratic equations*: The strategy is non-deterministic and can be described as follows. We guess which variables can be replaced by the empty word, these variables are canceled. After that we may assume that the equation is either of the form (where x, y are variables and a is a constant):

$$\begin{array}{lrcll} & x\cdots & = & a & \\ \text{or} & x\cdots & = & y\cdots & \text{with } x \neq y. \end{array}$$

We may either write $x = az$ or, by symmetry, $x = yz$, where z is a new variable. Replacing the occurrences of x by az or yz respectively, we obtain a new equation where x does not occur any more and z occurs at most twice. On the left we may cancel either a or y, and then y also occurs at most twice. Hence we end up with something new, but where the number of variables is the same as before and every variable occurs at most twice. The length of the equation did not increase, but either the number of variables or the length of a minimal solution did decrease. Therefore the non-deterministic procedure will find a solution, if there is any. An upper non-deterministic time bound is the length of a minimal solution. Note that the algorithm itself does give a doubly exponential function bounding the length of a minimal solution, only.

Consider

$$abxcy = ycxba.$$

The algorithm above has a graphical interpretation as in Figure 1. The arcs are labeled such that we can reconstruct a solution by going backwards on a path from the initial equation to the trivial equation $ba = ba$. One of the paths has the following labels:

$$y \leftarrow ay,\ y \leftarrow by,\ x \leftarrow yx,\ x \leftarrow 1,\ y \leftarrow ay,\ y \leftarrow 1.$$

It corresponds to the minimal solution, where $x = a$ and $y = aba$. Nodes or arcs which cannot lead to any solution have been omitted in the picture.

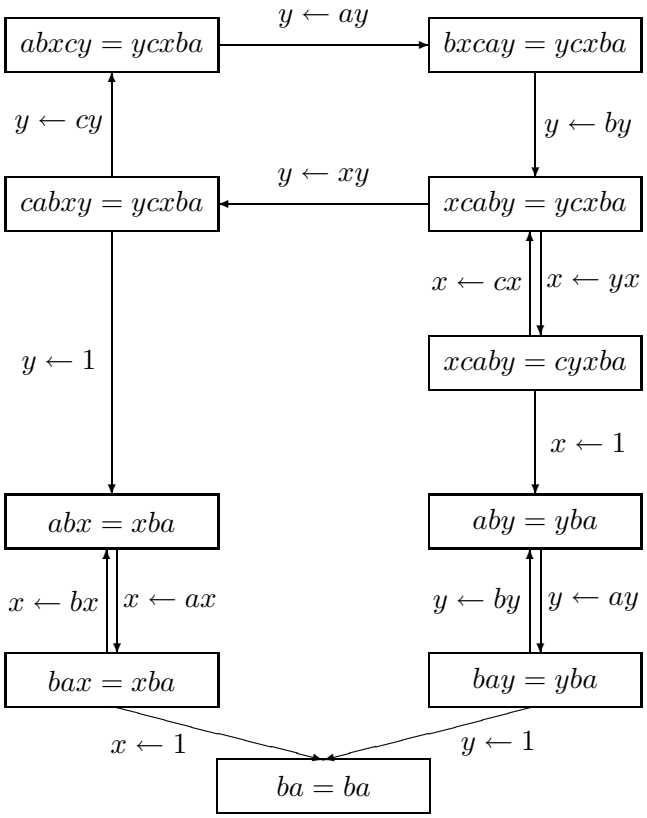


Figure 1: Solving the equation $abxcy = ycxba$.

The space requirement for this algorithm is linear, and we can translate it to deterministic exponential time. The exponential time bound might be unavoidable in the worst-case, because the satisfiability problem for quadratic word equations is know to be NP-hard, [17].

Some open problems

The central open problem in the field is related to the minimal length for a possible solution. In some sense this is what Makanin did, he gave an effective bound for this length. Decidability follows because we can view all words respecting the length bound as candidates for solutions and test them all.

Surprisingly this idea might lead to an optimal algorithm. Instead of guessing the word written in plain it is enough to guess a Lempel-Ziv encoding of the solution and test whether the guess is correct in polynomial time [15]. If we bounded the length of the Lempel-Ziv encoding of a minimal solution by some polynomial, we would have that *Word Equations* is NP-complete. For this it would be enough to bound the length of minimal solutions by some exponential function. We have to confess that an answer to the following question is not in sight.

Question: Is it possible to bound the length of minimal solutions for (solvable) equations by some exponential function?

Question: What is the grows function for the length of minimal solutions in quadratic equations?

Actually, the bound for quadratic equations might be better than the general case. It should be at most polynomial.

Another way to attack the problem is to bound the number of variables. *Word Equations* is NP-hard, but if we restrict the number of variables by some constant k, then it is conjectured to be a problem in P.

Question: Let $k \in \mathbb{N}$. Is there a deterministic polynomial time algorithm for solving word equations with at most k variables?

In graph theory there is a notion of *string graph* and for many years it was open whether the string graph property is decidable. Finally it turned out to be an NP-complete problem as shown by Schaefer, Sedgwick, and Štefankovič in [18]. It is amazing that the key for the solution to the string graph problem are quadratic word equations. The result can be extended to string graphs on compact surfaces of any genus and then the recognition problem becomes a problem on quadratic word equations modulo partial commutation. This problem can be decided in PSPACE, [2].

This leads to another field of active research, namely to extend Makanin's result to other algebraic structures. The most important example is a free group. Makanin showed in [11, 12] that the existential and positive theories in free groups are decidable. The scheme of Makanin is not primitive recursive [8] although today we know that the problem itself is in PSPACE [6, 1, 4]. Actually, much more is true in free groups. It became clear over the past few years that Tarski's conjecture holds: The theory in free groups is decidable! This achievement is due to Kharlampovich and Miasnikov [7] involving a series of papers and (among others) ideas of Sela.

The extension of Makanin-Tarski-type results beyond free groups is an exciting program, but actually little is known.

Question: Is the theory decidable in any of the following classes:

i.) Virtually free groups.

ii.) Graph groups.

iii.) Torsion free hyperbolic groups.

The existential theories are decidable [9, 3, 16], but it is open whether the existential theory is decidable for hyperbolic groups, in general.

Many of the results hold in the presence of constrains, where solutions must respect some regular restrictions. However, this is not known for lengths constraints. The work of Büchi and Senger shows undecidabilty, if we may specify that in a solution of a word equation variables must respect linear conditions for each letter. But if we add to the existential theory only predicates of the form $|x| = |y|$ then the situation is not clear.

Question: Consider the existential theory with additional predicates of the form $|x| = |y|$. Is this theory decidable?

30 years ago Makanin taught us how to solve word equations. Since then there has been much progress, so we are optimistic that during the next 30 years some of our questions will have found an answer. But who knows the future?

Si même vous êtes promise,
je suis sûre que votre promis aurait désiré
que vous alliez dans le monde en son absence
plutôt que de dépérir d'ennui.
Lev N. Tolstoj
Voijna i Mir. Tome II, Part 5, Section 12

References

[1] V. Diekert, C. Gutierrez, and C. Hagenah. The existential theory of equations with rational constraints in free groups is PSPACE-complete. *Information and Computation*, 202:105–140, 2005.

[2] V. Diekert and M. Kufleitner. A remark about quadratic trace equations. In M. Ito and M. Toyama, editors, *Develpoments in Language Theory: DLT 2002, Kyoto, Japan*, number 2450 in Lecture Notes in Computer Science, pages 59–66. Springer-Verlag, 2003.

[3] V. Diekert and M. Lohrey. Word equations over graph products. In P. K. Pandya and J. Radhakrishnan, editors, *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2003), Mumbai (India)*, number 2914 in Lecture Notes in Computer Science, pages 156–167. Springer-Verlag, 2003.

[4] V. Diekert and M. Lohrey. Existential and positive theories of equations in graph products. *Theory of Computing Systems*, 37:133–156, 2004.

[5] V. S. Guba. Equivalence of infinite systems of equations in free groups and semigroups to finite subsystems. *Mat. Zametki*, 40:321–324, 428, 1986. English translation: Math. Notes 40 (1986), 688–690.

[6] C. Gutiérrez. Satisfiability of word equations with constants is in exponential space. In *Proc. 39th Ann. Symp. on Foundations of Computer Science (FOCS'98), Los Alamitos (California)*, pages 112–119. IEEE Computer Society Press, 1998.

[7] O. Kharlampovich and A. Miasnikov. Elementary theory of free nonabelian groups. *Journal of Algebra*, 302:451–552, 2006. Paper 5 on the theory of a free group, first version 1999.

[8] A. Kościelski and L. Pacholski. Complexity of unification in free groups and free semi-groups. In *Proc. 31st Annual Symposium on Foundations of Computer Science*, volume II, pages 824–829, Los Alamitos, 1990. IEEE Computer Society Press.

[9] M. Lohrey and G. Sénizergues. Theories of HNN-extensions and amalgamated products. In *Proc. 34th International Colloquium on Automata, Languages and Programming (ICALP'06), Venice*, number 4052 in Lecture Notes in Computer Science, pages 504–515, Berlin Heidelberg, 2006. Springer-Verlag.

[10] G. S. Makanin. The problem of solvability of equations in a free semigroup. *Math. Sbornik*, 103:147–236, 1977. English transl. in Math. USSR Sbornik 32 (1977).

[11] G. S. Makanin. Equations in a free group. *Izv. Akad. Nauk SSR*, Ser. Math. 46:1199–1273, 1983. English transl. in Math. USSR Izv. 21 (1983).

[12] G. S. Makanin. Decidability of the universal and positive theories of a free group. *Izv. Akad. Nauk SSSR*, Ser. Mat. 48:735–749, 1984. In Russian; English translation in: *Math. USSR Izvestija, 25*, 75–88, 1985.

[13] Yu. V. Matiyasevich. *Hilbert's Tenth Problem*. MIT Press, Cambridge, Massachusetts, 1993.

[14] W. Plandowski. Satisfiability of word equations with constants is in PSPACE. *Journal of the Association for Computing Machinery*, 51:483–496, 2004.

[15] W. Plandowski and W. Rytter. Application of Lempel-Ziv encodings to the solution of word equations. In K. G. Larsen et al., editors, *Proc. 25th International Colloquium on Automata, Languages and Programming (ICALP'98), Aalborg (Denmark), 1998*, number 1443 in Lecture Notes in Computer Science, pages 731–742, Berlin Heidelberg, 1998. Springer-Verlag.

[16] E. Rips and Z. Sela. Canonical representatives and equations in hyperbolic groups. *Inventiones Mathematicae*, 120:489–512, 1995.

[17] J. M. Robson and V. Diekert. On quadratic word equations. In C. Meinel and S. Tison, editors, *Proc. 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS'99), Trier (Germany), 1999*, number 1563 in Lecture Notes in Computer Science, pages 217–226. Springer-Verlag, 1999.

[18] M. Schaefer, E. Sedgwick, and D. Štefankovič. Recognizing string graphs in NP. *Journal of Computer and System Sciences*, 2003:365–380, 2003.

Bottom-Up Rewriting for Words and Terms

Irène Durand and Géraud Sénizergues
LABRI, Université Bordeaux I, France

For the whole class of linear term rewriting systems, we define *bottom-up rewriting* which is a restriction of the usual notion of rewriting. We show that bottom-up rewriting effectively inverse-preserves recognizability and analyze the complexity of the underlying construction. The *Bottom-Up* class (BU) is, by definition, the set of linear systems for which every derivation can be replaced by a bottom-up derivation. Membership to BU turns out to be undecidable, we are thus lead to define more restricted classes: the classes $\mathrm{SBU(k)}, \mathrm{k} \in \mathbb{N}$ of *Strongly Bottom-Up*(k) systems for which we show that membership is decidable. We define the class of *Strongly Bottom-Up* systems by $\mathrm{SBU} = \bigcup_{\mathrm{k}\in\mathbb{N}} \mathrm{SBU(k)}$. We give a polynomial sufficient condition for a system to be in SBU. The class SBU contains (strictly) several classes of systems which were already known to inverse preserve recognizability: the inverse left-basic semi-Thue systems (viewed as unary term rewriting systems), the linear growing term rewriting systems, the inverse Linear-Finite-Path-Ordering systems.

References

[1] M. Benois. Descendants of regular language in a class of rewriting systems: algorithm and complexity of an automata construction. In *Rewriting techniques and applications (Bordeaux, 1987)*, volume 256 of *Lecture Notes in Comput. Sci.*, pages 121–132. Springer, Berlin, 1987.

[2] M. Benois and J. Sakarovitch. On the complexity of some extended word problems defined by cancellation rules. *Inform. Process. Lett.*, 23(6):281–287, 1986.

[3] L. Boasson and M. Nivat. Centers of context-free languages. *LITP technical report no84-44*, 1984.

[4] R.V. Book, M. Jantzen, and C. Wrathall. Monadic Thue systems. *TCS 19*, pages 231–251, 1982.

[5] W.S. Brainerd. Tree generating regular systems. *Information and Control*, 14:217–231, 1969.

[6] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2002. Draft, available from `www.grappa.univ-lille3.fr/tata`.

[7] R. Cremanns and F. Otto. Finite derivation type implies the homological finiteness condition FP_3. *J. Symbolic Comput.*, 18(2):91–112, 1994.

[8] M. Dauchet, T. Heuillard, P. Lescanne, and S. Tison. Decidability of the confluence of finite ground term rewrite systems and of other related term rewrite systems. *Inf. Comput.*, 88(2):187–201, 1990.

[9] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Fifth Annual IEEE Symposium on Logic in Computer Science (Philadelphia, PA, 1990)*, pages 242–248. IEEE Comput. Soc. Press, Los Alamitos, CA, 1990.

[10] A. Deruyver and R. Gilleron. The reachability problem for ground TRS and some extensions. In *TAPSOFT '89 (Barcelona, 1989)*, volume 351 of *Lecture Notes in Comput. Sci.*, pages 227–243. Springer, Berlin, 1989.

[11] I. Durand and A. Middeldorp. Decidable call-by-need computations in term rewriting. *Information and Computation*, 196:95–126, 2005.

[12] Z. Fülöp, E. Jurvanen, M. Steinby, and S. Vágvölgyi. On one-pass term rewriting. In *MFCS*, pages 248–256, 1998.

[13] A. Geser, D. Hofbauer, and J. Waldmann. Match-bounded string rewriting systems. *Journal Applicable Algebra in Engineering, Communication and Computing*, 15(3-4):149–171, November, 2004.

[14] A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. On tree automata that certify termination of left-linear term rewriting systems. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications*, volume 3467 of *Lecture Notes in Computer Science*. Springer Verlag, 2005.

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

[16] F. Jacquemard. Decidable approximations of term rewriting systems. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *Lecture Notes in Computer Science*, pages 362–376, 1996.

[17] J.W. Klop. Term rewriting systems. In *Handbook of Logic in Computer Science, Vol. 2*, pages 1–116. Oxford University Press, 1992.

[18] T. Knapik and H. Calbrix. Thue specifications and their monadic second-order properties. *Fund. Inform.*, 39(3):305–325, 1999.

[19] Y. Lafont and A. Prouté. Church-Rosser property and homology of monoids. *Math. Structures Comput. Sci.*, 1(3):297–326, 1991.

[20] M. Lohrey and G. Sénizergues. Rational subsets of HNN-extensions. To appear in IJAC. Manuscript at `http://dept-info.labri.u-bordeaux.fr/~ges`, 2005.

[21] P.V. Silva M. Kambites and B. Steinberg. On the rational subset problem for groups. *J. of Algebra*, To appear.

[22] T. Nagaya and Y. Toyama. Decidability for left-linear growing term rewriting systems. *Information and Computation*, 178(2):499–514, 2002.

[23] D. Park. Concurrency and automata on infinite sequences. *LNCS 104*, pages 167–183, 1981.

[24] P. Réty and J. Vuotto. Tree automata for rewrite strategies. *J. Symb. Comput.*, 40(1):749–794, 2005.

[25] J. Sakarovitch. Syntaxe des langages de Chomsky, essai sur le déterminisme. *Thèse de doctorat d'état de l'université Paris VII*, pages 1–175, 1979.

[26] H. Seki, T. Takai, Y. Fujinaka, and Y. Kaji. Layered transducing term rewriting system and its recognizability preserving property. In *Proceedings of the 13th International Conference on Rewriting Techniques and Applications*, volume 2378 of *Lecture Notes in Computer Science*. Springer Verlag, 2002.

[27] G. Sénizergues. Formal languages & word-rewriting. In *Term rewriting (Font Romeu, 1993)*, volume 909 of *Lecture Notes in Comput. Sci.*, pages 75–94. Springer, Berlin, 1995.

[28] F. Seynhaeve, S. Tison, and M. Tommasi. Homomorphisms and concurrent term rewriting. In *FCT*, pages 475–487, 1999.

[29] T. Takai, Y. Kaji, and H. Seki. Right-linear finite-path overlapping term rewriting systems effectively preserve recognizability. *Scienticae Mathematicae Japonicae*, 2006. (to appear, preliminary version: IEICE Technical Report COMP98-45).

Distance Desert Automata and the Star Height Problem

Daniel Kirsten*
Institut für Informatik
Universität Leipzig
Postfach 10 09 20
04009 Leipzig, Germany

http://www.informatik.uni-leipzig.de/~kirsten/

In the talk, we introduce desert automata. Desert automata are nondeterministic finite automata over words with a set of marked transitions which are called water transitions. The weight of a path is defined as the length of a longest subpath which does not contain a water transition. The weight of a word is the minimum of the weights of all successful paths of the word. In this way, desert automata compute mappings from a free monoid to the integers.

We generalize desert automata to distance desert automata which include Hashiguchi's distance automata as a particular case.

We consider the limitedness problems of these automata, i.e., we show that it is decidable and PSPACE-complete whether the range of the mapping of a given desert automaton is finite.

As an application of this result, we show the decidability and the first upper complexity bound to the star height problem.

References

[1] D. Kirsten. Distance desert automata and the star height problem. *R.A.I.R.O. - Informatique Théorique et Applications, special issue of long versions of selected best papers from FoSSaCS 2004*, 29(3):455–509, 2005.

[2] D. Kirsten. Distance desert automata and star height substitutions. Habilitationsschrift, Universität Leipzig, Fakultät für Mathematik und Informatik, 2006.
(Available on the author's homepage.)

*Supported by the research grant KI 822/1–1 of the German Research Community (Deutsche Forschungsgemeinschaft).

Powers of Rationals Modulo 1 and Rational Base Number Systems

Jacques Sakarovitch
LTCI, ENST/CNRS, Paris

In a first part, I'll present the framework of this work, the relationships between the writing of numbers and finite automata theory. They begin with toy examples of automata (due to Blaise Pascal indeed) and more seriously with the beautiful Cobham's theorem (1969).

The study of non standard numeration systems has made these links even tighter. Representation in integer base with signed digits was popularized in computer arithmetic by Avizienis and clearly uses finite automata. When the base is a real number $\beta > 1$, a number can be given several representation (even on the canonical alphabet) and the normal one, that is the one obtained by the greedy algorithm can be computed (from the other representations) by a finite automaton if and only if β is a Pisot number, that is an algebraic integer such that all its Galois conjugates have a modulus smaller than 1.

In a second part, I would like to present some recent results on rational base systems obtained in cooperation with my colleagues S. Akiyama (Niigata Univ.)and Ch. Frougny (Paris 8 Univ.).

A new method for representing positive integers and real numbers in a rational base is considered. It amounts to computing the digits from right to left, least significant first. Every integer has a unique such expansion. In contrast with the Pisot case, the set of expansions of the integers is not a regular language but nevertheless addition can be performed by a letter-to-letter finite right transducer. Every real number has at least one such expansion and a countable infinite set of them have more than one. We explain how these expansions can be approximated and characterize the expansions of reals that have two expansions.

These results are developed not only for their own sake but also as they relate to other problems in combinatorics and number theory. A first example is a new interpretation and expansion of the constant $K(p)$ from the so-called "Josephus problem". More important, these expansions in the base p/q allow us to make some progress in the problem of the distribution of the fractional part of the powers of rational numbers.

Beiträge zum Theorietag

Betrachtungen zu schwach kontextabhängigen parallelen Grammatikformalismen

Suna Bensch
Institut für Informatik
Universität Potsdam
August-Bebel-Str. 89
14482 Potsdam

Ein wichtiges Forschungsthema in der Linguistik befasst sich mit der Lokalisierung der natürlichen Sprachen (als Menge von Zeichenketten) innerhalb der Chomskyhierarchie. Die kontextfreien Grammatiken vermögen die meisten strukturellen Aspekte der natürlichen Sprachen zu beschreiben, doch es existieren syntaktische Konstruktionen in den natürlichen Sprachen, die nicht von kontextfreien Grammatiken beschrieben werden können (siehe u.a. [2]). Die Familie der kontextabhängigen Sprachen eignet sich auch nicht als Modell für die natürlichen Sprachen, da sie u.a. nicht-semilineare Sprachen enthält und somit der linguistischen Intuition widerspricht, dass sich natürlichsprachliche Sätze in einer Art zusammensetzen, die einer linearen Längenprogression unterliegen. Ausserdem gibt es für die kontextabhängigen Sprachen keine bekannten deterministisch polynomiellen Parsingalgorithmen.

Die sogenannten *schwach kontextabhängigen* Sprachfamilien, liegen zwischen der kontextfreien Sprachfamilie und der kontextabhängigen Sprachfamilie in der Chomskyhierarchie.

Der Begriff der schwachen Kontextabhängigkeit wurde in [2] eingeführt, wo der Autor vorschlägt, dass eine Sprachfamilie als ein adäquates Modell für die natürlichen Sprachen bestimmte nicht-kontextfreie Sprachen enthalten sollte, um die nicht-kontextfreien Phänomene in den natürlichen Sprachen zu beschreiben. Ausserdem sollten die Sprachen jener Familie einer linearen Längenprogression unterliegen und die Sprachen sollten polynomial parsbar sein. In der Literatur trifft man auf unterschiedliche Definitionen für die schwache Kontextabhängigkeit einer Sprachfamilie. Wir folgen der Definiten von [1]:

Definition 1 Eine Grammatikfamilie $\mathcal{G}$ heisst *schwach kontextabhängig*, wenn folgende Bedingungen erfüllt sind:

1. Für jede kontextfreie Sprache L, gibt es eine Grammatik G in $\mathcal{G}$ mit

$L = L(G)$; desweiteren existieren Grammatiken G_1, G_2, und G_3 in $\mathcal{G}$, so dass

- $L(G_1) = \{ a^n b^n c^n \mid n \geq 1 \}$,
- $L(G_2) = \{ a^n b^m c^n d^m \mid n, m \geq 1 \}$,
- $L(G_3) = \{ ww \mid w \in \{a, b\}^+ \}$.

2. Für jede Grammatik G in $\mathcal{G}$ ist die Sprache $L(G)$ semilinear.

3. Für $\mathcal{G}$ ist das Wortproblem:

   `Gegeben:` Eine Grammatik G in $\mathcal{G}$ und ein Wort w.

   `Frage:` Ist w in $L(G)$?

   in deterministisch polynomieller Zeit entscheidbar.

Es existieren andere Definitionen für eine schwach kontextabhängige Sprachfamilie, in denen man nicht fordert, dass alle kontextfreien Sprachen enthalten sein müssen. In der Linguistik wurden viele schwach kontextabhängige sequentielle Grammatikformalismen betrachtet.

In unserem Vortrag untersuchen wir parallele Grammatikformalismen hinsichtlich ihrer schwachen Kontextabhängigkeit. Unser besonderes Augenmerk gilt der Familie der deterministischen ET0L Systeme mit endlichem Index ($\mathcal{L}(\text{EDT0L})_{\text{FIN}}$), welche in [3, 4] betrachtet wurden und der Familie der propagierenden k-uniform-limitierten Lindenmayersysteme ($k$ulEPT0L Systeme), die in [5] eingeführt wurden.

Literatur

[1] H. Bordihn. Mildly Context-Sensitive Grammars. In C. Martín-Vide, V. Mitrana, Gh. Păun, editors, *Formal Languages and Application, Studies in Fuzziness and Soft Computing 148*, 163–173. Springer, Berlin, 2004.

[2] A. K. Joshi. Tree Adjoining Grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In D. R. Dowty, L. Karttunen, A. M. Zwicky, editors, *Natural Language Parsing. Psychological, Computational, and Theoretical Perspectives*, 206–250. Cambridge University Press, New York, 1985.

[3] G. Rozenberg, D. Vermeir. On ET0L systems of finite index. *Information and Control*, 38(1): 103–133, 1978.

[4] G. Rozenberg, D. Vermeir. On the effect of the finite index restriction on several families of grammars. *Information and Control*, 39(3): 284–302, 1978.

[5] D. Wätjen, E. Unruh. On Extended k-uniformly-limited T0L Systems and Languages. *Journal of Information Processing and Cybernetics*, EIK 26(5/6): 283–299, 1990.

Modelling Logical Gates with ESNPA Systems

Aneta BINDER[1], Rudolf FREUND[1], Marion OSWALD[1], Lorenz VOCK[2]

[1] Vienna University of Technology
Faculty of Informatics
Favoritenstr. 9, 1040 Vienna, Austria
{ani,rudi,marion}@emcc.at

[2] Medical University of Vienna
Währinger Gürtel 18-20, 1090 Vienna, Austria
lorenz.vock@meduniwien.ac.at

1 Introduction

Incorporating the ideas of P systems and the functioning of the human brain, spiking neural P systems have been introduced in [4]. For a comprehensive overview of P systems see [5] and for the actual status of research on this area we refer the reader to [9]. More information on spiking neural models can be found in [3]. Based on the biological background of astrocytes, as intermediaries in the modulation of neuronal excitability (see [6] and [7]), we develop a model of *extended spiking neural P systems with excitatory and inhibitory astrocytes* (ESNPA) on the ideas of (extended) spiking neural P systems [1] and add the concept of astrocytes influencing the signals along the axons. For the astrocytes themselves, we assume their membrane potential to be changed according to external inputs which may come from neural cells or the firing intensity and frequency along the axon. In this paper we focus on applications of discrete functions and show the potentials of our model by formalizing networks of logical gates.

2 Extended Spiking Neural P Systems with Excitatory and Inhibitory Astrocytes

For the basic elements of formal language theory needed in the following, we refer to any monograph in this area, in particular, to [8]. The following definition of the ESPNA model is mainly taken from [2].

An *extended spiking neural P system with excitatory and inhibitory astrocytes* (of degree $m \geq 1$) (*ESNPA system*) is a construct

$$\Pi = (m, n, S, R, U)$$

where

- m is the number of *neurons*; (between 1 and m);
- n is the number of *astrocytes*; (between $m+1$ and $m+n$);
- S is the *initial configuration*; an initial value (of spikes) for each neuron and an initial value (membrane potential) for each astrocyte;
- R is a finite set of *rules* of the form $\left(i, E/a^k \rightarrow P\right)$ such that $i \in [1..m]$, (specifying that this rule is assigned to cell i) $E \subseteq REG(\mathbb{N})$ is the *checking set* (the current number of spikes in the neuron has to be from E if this rule shall be executed), $k \in \mathbb{N}$ is the "number of spikes" consumed by this rule, P is a set of *productions* of the form (l, w), where $l \in [1..m+n]$ (specifying the target neuron or astrocyte), $w \in \mathbb{N}$ is the *weight* of the energy sent along the axon from neuron i to neuron or astrocyte l;
- U is a finite set of *rules* describing the influence of an *excitatory astrocyte* or an *inhibitory astrocyte* on an axon between two neurons.

An ESNPA system can be used to generate sets of numbers from $RE(\mathbb{N})$ as follows: A computation is called *successful* if it halts, i.e., if for no neuron, a rule can be activated. We then consider the contents (the number of spikes), of a specific *output neuron* in halting computations.

Furthermore we assume external input signals arriving in some designated *input neurons* as well as several *output neurons* for sending out the computed function with a spike indicating the signal 1 and with no spike indicating the signal 0.

The rules $\left(i, E/a^k \rightarrow P\right)$ in the examples given in the succeeding section will be of a very special form, i.e., we always have $E = \left\{a^k\right\}$, hence, we can omit E. We can indicate such rules as in Figure 1, where the rule $a^k \rightarrow a^l$ in neuron p means, that k spikes are consumed in neuron p and l spikes are sent to every neuron q if there exists an axon from p to q. Moreover, a^m in neuron p indicates the initial value of m spikes in p.

The influence of an *excitatory astrocyte* r on an axon between two neurons p and q is depicted in Figure 2: $\geq k|f$ in astrocyte r means that if $x \geq k$ spikes are present in the astrocyte r and y spikes are sent out from neuron p then $f(y)$ spikes will reach neuron q, whereas for a number of spikes $x < k$ in astrocyte r, no spike will reach q. On the other hand (Figure 3) $\leq k|f$ in astrocyte r means that if $x \leq k$ spikes are present in the astrocyte r and y spikes are sent out from neuron p then $f(y)$ spikes will reach neuron q, whereas for a number of spikes $x > k$ no spike will reach q.

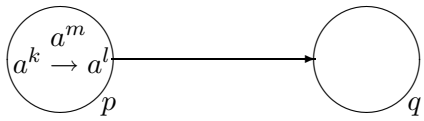


Figure 1: Representation of simple rules in neurons.

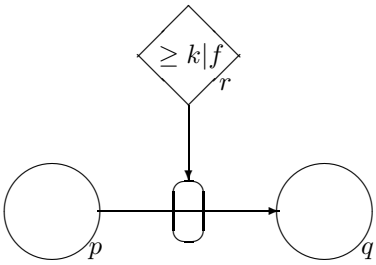


Figure 2: Excitatory astrocyte.

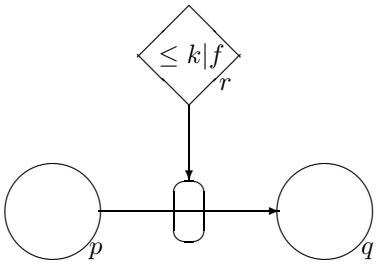


Figure 3: Inhibitory astrocyte.

3 Computing with ESNPA Systems

3.1 Computational Completeness

As already the original model of spiking neural P systems was shown to be computationally complete ([4]), we can immediately obtain computational completeness for ESNPA systems, too, which are, by omitting astrocytes, a sufficiently powerful submodel of spiking neural P systems.

3.2 Networks of Logical Gates

As is well known, any Boolean function can be obtained by networks only consisting of NAND-gates (and units representing the identity function).

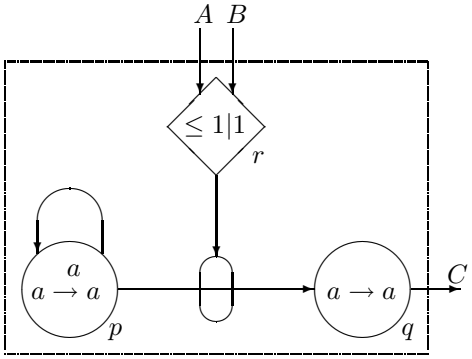


Figure 4: NAND-gate.

The NAND-gate is shown in Figure 4: A, B are inputs, C is the output; the neuron p is a source sending out one spike in each time step. The notion $\leq 1|1$ in astrocyte r means that only if the sum of input spikes (A and B) is ≤ 1, then one spike is sent to output neuron q, whereas if more than one input spikes arrive in r, then no spike will reach q. This is if both inputs represent different signals ($A \neq B$) or if both are set to 0.

3.3 A Discrete Amplifier

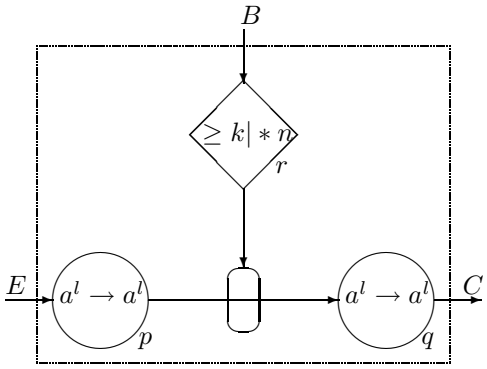


Figure 5: An ESNPA amplifier.

The ESNPA system depicted in Figure 5 represents a discrete amplifier which, as soon as the input from B goes beyond the given threshold k, from the input x given at E computes the function $f(x) = nx$ at C. We have to remark that the rules $a^l \to a^l$ given in the neurons p and q represent the (theoretically infinite) set of rules $\left\{\{a\}^* / a^l \to a^l \mid l \in \mathbb{N}\right\}$.

References

[1] A. Alhazov, R. Freund, M. Oswald and M. Slavkovik, Extended Spiking Neural P Systems Generating Strings and Vectors of Non-Negative Integers, in: H. J. Hoogeboom, Gh. Paun, G. Rozenberg (eds), Preproceedings of WMC7, 2006, pp. 88–101.

[2] A. Binder, R. Freund and M. Oswald, Extended spiking neural P systems with excitatory and inhibitory astrocytes, in: Proceedings of 8th WSEAS Conference on Evolutionary Computing, *Lecture Notes in Computational Intelligence*, ISBN 9789608457812, Vancouver, Canada, 2007, pp. 318–323.

[3] W. Gerstner, W. Kistler, *Spiking Neuron Models. Single Neurons, Populations, Plasticity.* Cambridge Univ. Press, 2002.

[4] M. Ionescu, Gh. Păun and T. Yokomori, Spiking neural P systems, *Fundamenta Informaticae* 71, 2–3, 2006, pp. 279–308.

[5] Gh. Păun, *Computing with Membranes: An Introduction*. Springer, Berlin, 2002.

[6] G. Perea and A. Araque, Communication between astrocytes and neurons: a complex language. *Journal of Physiology Paris* 96, 2002, 199–207.

[7] X. Shen and P. de Wilde, Long-term neuronal behavior caused by two synaptic modification mechanisms. *Neurocomputing* 70, 7–9, 2007, 1482–1488.

[8] G. Rozenberg and A. Salomaa (eds), *Handbook of Formal Languages* (3 vol), Springer, Berlin, 1997.

[9] The P Systems Web Page, `http://psystems.disco.unimib.it`

On the Expressive Power of 2-Stack Visibly Pushdown Automata

Benedikt Bollig*

Visibly pushdown automata are input-driven pushdown automata that recognize some non-regular context-free languages while preserving the nice closure and decidability properties of finite automata [1]. Visibly pushdown automata with multiple stacks have been considered recently and independently by La Torre, Madhusudan, and Parlato [4], as well as Carotenuto, Murano, and Peron [3], who exploit the concept of visibility further to obtain a rich pushdown-automata class that can even express properties beyond the class of context-free languages. At the same time, their automata are closed under boolean operations, come up with a decidable emptiness and inclusion problem, and enjoy a logical characterization in terms of monadic second-order logic over nested words, which add a nesting structure to ordinary words. These results, however, require a restricted version of visibly pushdown automata. In [4], the domain is restricted to words whose behavior can be split up into a fixed number of phases, whereas in [3], a pop operation on the second stack is only possible if the first stack is empty.

In this talk, we consider 2-stack visibly pushdown automata (i.e., visibly pushdown automata with two stacks) in their unrestricted form. Our main results in this regard read as follows:

1. 2-stack visibly pushdown automata are expressively equivalent to the existential fragment of monadic second-order logic.

2. Over nested words, monadic second-order quantifier alternation forms an infinite hierarchy (unlike in the restricted domain of [4], where full monadic second-order logic is only as expressive as its existential fragment).

3. 2-stack visibly pushdown automata are not closed under complementation.

Finally, we discuss the expressive power of Büchi 2-stack visibly pushdown automata running over infinite words. Extending the logic by an infinity quantifier, we can likewise establish equivalence to existential monadic second-order logic.

*LSV, ENS Cachan, CNRS — 61, avenue du Président Wilson, 94235 Cachan Cedex, France — email address: bollig@lsv.ens-cachan.fr

A full version of this abstract is available [2].

References

[1] R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC 2004)*, pages 202–211. ACM Press, 2004.

[2] B. Bollig. On the expressive power of 2-stack visibly pushdown automata. Research Report LSV-07-27, Laboratoire Spécification et Vérification, ENS Cachan, France, September 2007. `www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2007-27.pdf`.

[3] D. Carotenuto, A. Murano, and A. Peron. 2-visibly pushdown automata. In *Proceedings of the 11th International Conference on Developments in Language Theory (DLT 2007)*, volume 4588 of *Lecture Notes in Computer Science*, pages 132–144. Springer, 2007.

[4] S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *Proceedings of the 22nd IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 161–170. IEEE Computer Society Press, 2007.

Top-down Deterministic Parsing of Languages Generated by CD Grammar Systems

Henning Bordihn*
Institut für Informatik, Universität Gießen,
Arndstraße 2, D-35392 Gießen, Germany
bordihn@informatik.uni-giessen.de

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

Most of the efficient parsers which have been developed are applicable to the class of context-free grammars or a subclass thereof. On the other hand, also non-context-free aspects are encountered in several applications of formal languages [7]. In many programming languages, for instance, an identifier has to be declared prior to its use, and this syntactic feature is being checked during the semantic analysis of compilers as it cannot be expressed by means of context-free grammars [9]. Also in natural languages there are non-context-free phenomena which led to the rich theory of mildly context-sensitive grammars [10, 2].

Since the late sixties, a series of grammars has been proposed which use—in the core—context-free productions but posses some additional control on the application of the productions, adding to the generative capacity of context-free grammars [7]. In 1990, the concept of cooperating distributed grammar systems (CDGS, in short) has been introduced in [5] as a model of distributed problem solving. Context-free CDGS consist of several context-free grammars, the components of the system, which jointly derive a formal language, rewriting the sentential form in turns, according to some cooperation protocol (the so-called mode of derivation). In the present paper, two different cooperation protocols are taken into consideration, namely the $=m$-mode and the t-mode, in which a component, once started, has to perform exactly m and as many as possible derivation steps, respectively. This

*Most of the work was done while the first author was at the University of Potsdam, Institut für Informatik, August-Bebel-Stratße 89, D-14482 Potsdam, Germany.

†The second author was supported by a research scholarship of the Alexander von Humboldt Foundation.

approach enhances the descriptive power of (single) context-free grammars, as well; for a survey on CDGS, see the monograph [6] or the handbook chapter [8].

Our aim is to restrict context-free CDGS so that deterministic one pass no backtrack parsing in a top-down manner becomes possible. (For another approach in this direction, see [11].) In order to make them deterministic, we first restrict context-free CDGS to leftmost derivations. Two kinds of leftmostness are taken into consideration, which are appropriately defined for context-free CDGS. In contrast to context-free grammars, such restrictions can alter the generative capacity of CDGS; for an extensive investigation of leftmost context-free CDGS see [4]. At second, some sort of LL(k) condition is imposed to the systems. Given a context-free CDGS and an input string to be analyzed, the LL(k) property guarantees that for any sentential form of any leftmost derivation, the next k symbols of the input string (which have not been derived to the left of the leftmost nonterminal in the current sentential form), if they exist, determine the next step to be performed by the CDGS. That is, according to the derivation mode, a unique sequence of productions from a unique component is determined.

We first define the concept of LL(k) context-free CDGS working in the $=m$- and t-modes of derivation and using two different types of leftmost restrictions, and prove some first hierarchical properties for those systems satisfying the LL(k) condition. Then we focus on the $=m$-mode of derivation. It is shown that deterministic LL(k) CDGS working in the $=m$-mode can be simulated by deterministic LL(k) CDGS working in the =2-mode, if the so-called sw-type of leftmostness is imposed. Moreover, we show that systems of that type can generate non-semilinear languages and, given the lookup table, that a lookup string of length 1 is sufficient, that is, those systems can be simulated by deterministic LL(1) CDGS. We also provide a parsing algorithm for these systems which is of strictly sub-quadratic time complexity (see also [3]), if the lookup table is given as part of its input.

Finally, a condition is presented under which the lookup table can effectively be constructed.

References

[1] A. V. Aho, J. D. Ulmann, *The Theory of Parsing, Translation, and Compiling. Volume I.* Prentice-Hall, Englewood Cliffs, N.J., 1973.

[2] H. Bordihn: Mildly context-sensitive grammars. In: C. Matín-Vide, V. Mitrana, eds., *Formal Languages and Application, Studies in Fuzziness and Soft Computing 148*, 163–173. Springer-Verlag, Berlin, Heidelberg, 2004.

[3] H. Bordihn, Gy. Vaszil, CD grammar systems with LL(k) conditions. In: E. Csuhaj-Varjú, Gy. Vaszil, eds., *Proceedings of Grammar Systems Week*, MTA SZTAKI, Budapest, 95–112, 2004.

[4] H. Bordihn, Gy. Vaszil, On leftmost derivations in CD grammar systems. Accepted for LATA 2007.

[5] E. Csuhaj-Varjú, J. Dassow, On cooperating/distributed grammar systems, *Journal of Information Processing and Cybernetics* EIK, **26** (1990), 49–63.

[6] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun, *Grammar Systems. A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach, London, 1994.

[7] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*. Springer, 1989.

[8] J. Dassow, Gh. Păun, G. Rozenberg, Grammar Systems. Chapter 4. *Handbook of Formal Languages*, in: G. Rozenberg and A. Salomaa (eds.), Springer, Berlin, 1997, 155-213.

[9] R. W. Floyd: On the non-existence of a phrase-structure grammar for ALGOL 60. Comm. of the ACM **5**, 483–484.

[10] A. K. Joshi, How much context-sensitivity is necessary for characterizing structural descriptions: Tree adjoining grammars, in: D. Dowty, L. Karttunen, and A. Zwicky (eds.), *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives*, Cambridge University Press, New York, 1985.

[11] V. Mitrana, Parsability approaches in CD grammar systems, in: R. Freund and A. Kelemenová (eds.), *Proceedings of the International Workshop Grammar Systems 2000*, Silesian University at Opava, 2000, 165–185.

Zur Mächtigkeit von Netzwerken evolutionärer Prozessoren mit zwei Knotenarten

Jürgen Dassow
Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik
Universitätsplatz 2, D-39106 Magdeburg

und

Bianca Truthe*
Universitat Rovira i Virgili, Facultat de Lletres, GRLMC
Plaça Imperial Tàrraco 1, E-43005 Tarragona

Zusammenfassung

Gegenstand dieser Arbeit ist die Erzeugungskraft von Netzwerken evolutionärer Prozessoren, bei denen die Anzahl der vorkommenden Knotenarten auf zwei beschränkt ist. Wir weisen nach, dass (bis auf Durchschnitt mit einem Monoid) jede rekursiv aufzählbare Sprache von einem Netzwerk mit einem löschenden und zwei einfügenden Knoten erzeugbar ist, Netzwerke mit beliebig vielen einfügenden und ersetzenden Prozessoren kontextabhängige Sprachen erzeugen und zum Erzeugen jeder kontextabhängigen Sprache (bis auf Durchschnitt mit einem Monoid) nur ein einfügender und ein ersetzender Knoten nötig sind, sowie Netzwerke mit beliebig vielen ersetzenden und löschenden Prozessoren endliche Sprachen erzeugen und für jede endliche Sprache ein ersetzender oder löschender Knoten ausreichend ist.

1 Einleitung

Netzwerke von Sprachprozessoren wurden von E. CSUHAJ-VARJÚ und A. SALOMAA eingeführt ([4]). Solch ein Netzwerk kann als Graph angesehen werden, bei dem jeder Knoten Regeln und Wörter hat, die er entsprechend den Regeln ableitet, und die nach dem Passieren gewisser Filter über die Kanten zu anderen Knoten gelangen. Die von einem Netzwerk erzeugte Sprache besteht aus allen Wörtern, die irgendwann in einem festgelegten Knoten auftreten.

Von biologischen Prozessen inspiriert, haben J. CASTELLANOS, C. MARTIN-VIDE, V. MITRANA und J. SEMPERE in [2] Netzwerke evolutionärer Prozessoren eingeführt. Dabei sind die verwendeten Regeln Ersetzungen eines Buchstabens

*Die Forschung wurde unterstützt durch die Alexander-von-Humboldt-Stiftung.

durch einen anderen, Einfügen eines Buchstabens und Löschen eines Buchstabens. Diese Regeln modellieren Punktmutationen in der Biologie.

In [3] wurde gezeigt, dass Netzwerke evolutionärer Prozessoren rekursiv aufzählbare Sprachen erzeugen und für jede dieser Sprachen sechs Knoten genügen. Dieses Ergebnis wurde in [1] zu drei Knoten verbessert, wobei alle drei Typen vorkommen. In der vorliegenden Arbeit wird die Mächtigkeit von Netzwerken untersucht, in denen nur zwei Arten von Knoten vorkommen.

Wir geben im Folgenden einige der in dieser Arbeit verwendeten Begriffe und Notationen an. Für weitere Definitionen sei auf die Literatur verwiesen (z. B. [8]).

Zu einem Alphabet V bezeichnen wir mit V^* die Menge aller Wörter über V einschließlich dem Leerwort λ.

Eine Grammatik ist ein Quadrupel $G = (N, T, P, S)$ mit einem Alphabet N von Nichtterminalen, einem Alphabet T von Terminalen, einer endlichen und nichtleeren Menge P von Ersetzungsregeln der Form $\alpha \to \beta$ mit $\alpha \in (N \cup T)^* \setminus T^*$ und $\beta \in (N \cup T)^*$ und einem Startsymbol $S \in N$.

Eine Grammatik ist in Kuroda-Normalform, wenn alle ihre Ersetzungsregeln eine der folgenden Formen haben: $AB \to CD$, $A \to CD$, $A \to x$, $A \to \lambda$ mit $A, B, C, D \in N$ und $x \in N \cup T$.

Eine konditionale (monotone) Grammatik ist eine Grammatik, bei der die Regeln Paare von einer „herkömmlichen“ Ersetzungsregel und einer regulären Menge sind. Eine konditionale Regel (p, R) ist nur dann auf ein Wort w anwendbar, wenn das Wort w zur regulären Menge R gehört und die Ersetzungsregel p auf w anwendbar ist.

Eine Regel $\alpha \to \beta$ heißt
- ersetzend, wenn $|\alpha| = |\beta| = 1$ gilt, und
- löschend, wenn $|\alpha| = 1$ und $\beta = \lambda$ gelten.

Wir betrachten Einfügen als Gegenstück zu Löschen, schreiben $\lambda \to a$ für einen Buchstaben a und bezeichnen es ebenfalls als Regel. Das Einfügen $\lambda \to a$ liefert zu einem Wort w ein Wort $w_1 a w_2$ mit $w = w_1 w_2$ für zwei (möglicherweise leere) Wörter w_1 und w_2. Ersetzende, löschende und einfügende Regeln werden auch Evolutionsregeln genannt.

Wir definieren nun Netzwerke evolutionärer Prozessoren.

Definition 1.1

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

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

mit

- *einem Alphabet V,*
- *n Knoten $N_i = (M_i, A_i, I_i, O_i)$ (für $1 \leq i \leq n$), wobei*
  - *M_i eine sortenreine Menge von Evolutionsregeln ist,*
  - *A_i eine endliche Teilmenge von V^* ist und*

– I_i *und* O_i *reguläre Sprachen über* V *sind,*

- *einer Teilmenge* E *von* $\{1,2,\ldots,n\} \times \{1,2,\ldots,n\}$ *und*
- *einer natürlichen Zahl* j *mit* $1 \leq j \leq n$.

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

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

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

– *Kommunikationsschritt abgeleitet wird (geschrieben* $C \vdash C'$*), wenn für* $1 \leq i \leq n$

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

gilt.

Die Berechnung von $\mathcal{N}$ *ist eine Folge von Konfigurationen*

$$C_t = (C_t(1), C_t(2), \ldots, C_t(n)), \;\; t \geq 0,$$

so dass

– $C_0 = (A_1, A_2, \ldots, A_n)$ *gilt,*

– *für alle* $t \geq 0$ *die Konfiguration* C_{2t} *zu* C_{2t+1} *in einem Evolutionsschritt führt:* $C_{2t} \Longrightarrow C_{2t+1}$,

– *für alle* $t \geq 0$ *die Konfiguration* C_{2t+1} *zu* C_{2t+2} *in einem Kommunikationsschritt führt:* $C_{2t+1} \vdash C_{2t+2}$.

(iv) *Die von einem Netzwerk* $\mathcal{N}$ *erzeugte Sprache* $L(\mathcal{N})$ *ist*

$$L(\mathcal{N}) = \bigcup_{t \geq 0} C_t(j),$$

wobei $C_t = (C_t(1), C_t(2), \ldots, C_t(n))$, $t \geq 0$ *die Berechnung von* $\mathcal{N}$ *ist.*

Man stelle sich ein Netzwerk evolutionärer Prozessoren als einen gerichteten Graphen vor, dessen Knoten N_i ($1 \leq i \leq n$) Prozessoren sind, die über die Kanten (angegeben durch die Menge E) Wörter austauschen. Jeder Prozessor N_i hat eine Menge M_i von Evolutionsregeln, eine Menge von Wörtern (anfänglich A_i), einen Eingangsfilter I_i und einen Ausgangsfilter O_i. Der Prozessor heißt

– ersetzend, wenn $M_i \subseteq \{a \to b \mid a, b \in V\}$ gilt,
– einfügend, wenn $M_i \subseteq \{\lambda \to b \mid b \in V\}$ gilt, und

– löschend, wenn $M_i \subseteq \{a \to \lambda \mid a \in V\}$ gilt.

In einem Evolutionsschritt leitet jeder Prozessor seine Wortmenge entsprechend seiner Regeln ab. Die neue Wortmenge besteht dabei aus allen jenen Wörtern, die dadurch entstehen, dass eine Regel auf ein Wort der ursprünglichen Menge an einer möglichen Stelle angewendet wird, und jenen Wörten, auf die keine Regel anwendbar ist.

In einem Kommunikationsschritt sendet jeder Prozessor N_i alle Wörter, die seinen Ausgangsfilter passieren, zu allen benachbarten Prozessoren (zu denen eine Kante hinführt). Die Wörter, die der Ausgangsfilter nicht durchlässt, bleiben im Prozessor. Außerdem nimmt jeder Prozessor alle Wörter auf, die ihn auf einer ankommenden Kante erreichen und seinen Eingangsfilter passieren. Wörter, die einen Knoten verlassen und von keinem Knoten aufgenommen werden, gehen verloren (verschwinden aus dem Netzwerk).

Die Arbeit eines Netzwerkes beginnt mit einem Evolutionsschritt; danach wechseln sich Kommunikations- und Evolutionsschritte ab. Die erzeugte Sprache besteht aus allen Wörtern, die irgendwann zu dem ausgewiesenen Prozessor N_j gehören.

2 Netzwerke mit ersetzenden Prozessoren

In diesem Abschnitt wird die Erzeugungskraft von Netzwerken betrachtet, die nur aus ersetzenden und löschenden (aber keinen einfügenden) oder nur aus ersetzenden und einfügenden (aber keinen löschenden) Knoten bestehen.

Satz 2.1 *Die Klasse der von Netzwerken evolutionärer Prozessoren mit ausschließlich ersetzenden und löschenden Knoten erzeugten Sprachen stimmt mit der Klasse $\mathcal{L}(FIN)$ der endlichen Sprachen überein.*

Beweis. Ohne einfügende Regeln entstehen aus den (endlich vielen) Wörtern der Anfangsmengen keine längeren Wörter. Folglich gibt es zu jedem Netzwerk $\mathcal{N}$ eine natürliche Zahl $k_{\mathcal{N}}$, so dass jedes Wort der Sprache $L(\mathcal{N})$ von der Länge höchstens $k_{\mathcal{N}}$ ist.

Jede endliche Sprache F ist erzeugbar von einem Netzwerk mit genau einem löschenden oder ersetzenden Knoten, in dem F die Anfangswortmenge des Knotens ist und die Regeln so gewählt werden, dass sie nicht anwendbar sind (z. B. durch Hinzufügen von Buchstaben zum Alphabet des Netzwerkes, die nicht in Wörtern der Sprache F auftreten). □

Satz 2.2 *Die Klasse der von Netzwerken evolutionärer Prozessoren mit ausschließlich ersetzenden und einfügenden Knoten erzeugten Sprachen stimmt mit der Klasse $\mathcal{L}(CS)$ der kontextabhängigen Sprachen überein.*

Beweis. Die Arbeitsweise eines Netzwerks evolutionärer Prozessoren kann durch eine konditionale monotone Grammatik simuliert werden. Die Konstruktion einer solchen Grammatik zu einem Netzwerk ist ausführlich in der Arbeit [6] erläutert. Da konditionale monotone Grammatiken kontextabhängige Sprachen erzeugen ([5]), erzeugt auch jedes Netzwerk mit ersetzenden und einfügenden aber ohne löschende Prozessoren eine kontextabhängige Sprache.

Zu jeder kontextabhängigen Sprache L existieren eine Menge T und ein Netzwerk $\mathcal{N}$ mit genau einem einfügenden und einem ersetzenden Prozessor, so dass $L = L(\mathcal{N}) \cap T^*$ gilt. In der Arbeit [7] (und [6]) ist dargelegt, wie zu einer Grammatik G in Kuroda-Normalform, die die Sprache L erzeugt, ein Netzwerk konstruiert werden kann, das die Ableitungsschritte der Grammatik G simuliert. Der ersetzende Prozessor markiert in geeigneter Weise die Nichtterminale, auf die eine Regel der Grammatik angewendet werden soll (so, dass an der Markierung stets die Regel ablesbar ist) und ersetzt schließlich die Markierung durch neue Nichtterminale (entsprechend der betreffenden Regel); der einfügende Prozessor sorgt für das Verlängern von Wörtern (bei Regeln der Form $A \to BC$). Die Filter bewirken, dass nur jene Wörter mit richtig gesetzten Markierungen erhalten bleiben (die anderen verlassen einen Knoten und verschwinden). Wird als Menge T die Menge der Terminalsymbole der Grammatik G genommen, so bilden die vom Netzwerk erzeugten Wörter, die nur aus Terminalsymbolen bestehen, gerade die Sprache L.

Nimmt man einen weiteren Prozessor hinzu, der die terminalen Wörter „auffängt“, so erhält man ein Netzwerk $\mathcal{N}'$, das genau die Sprache L erzeugt. Dieser Prozessor darf nur Regeln haben, die auf kein Wort der Sprache L anwendbar sind, damit er die ankommenden Wörter nicht noch verändern kann. Da einfügende Regeln immer anwendbar sind, muss ein ersetzender Prozessor genommen werden. □

3 Netzwerke ohne ersetzende Prozessoren

In diesem Abschnitt wird die Erzeugungskraft von Netzwerken betrachtet, die nur aus löschenden und einfügenden (aber keinen ersetzenden) Knoten bestehen.

Satz 3.1 *Die Klasse der von Netzwerken evolutionärer Prozessoren mit ausschließlich löschenden und einfügenden Knoten erzeugten Sprachen stimmt mit der Klasse $\mathcal{L}(RE)$ der rekursiv aufzählbaren Sprachen überein.*

Beweis. Beliebige Netzwerke evolutionärer Prozessoren erzeugen rekursiv aufzählbare Sprachen, folglich auch Netzwerke ohne ersetzende Knoten ([3]).

Zu jeder rekursiv aufzählbaren Sprache L existieren eine Menge T und ein Netzwerk $\mathcal{N}$ mit genau einem löschenden und zwei einfügenden Prozessoren, so dass $L = L(\mathcal{N}) \cap T^*$ gilt. In der Arbeit [7] ist dargelegt, wie zu einer Grammatik G in Kuroda-Normalform, die die Sprache L erzeugt, ein Netzwerk konstruiert werden kann, das die Ableitungsschritte der Grammatik G simuliert. Ein Prozessor

fügt Symbole ein und markiert damit die Stelle, an der eine Regel der Grammatik angewendet werden soll. Der löschende Prozessor entfernt die zu überschreibenden Nichtterminale. Der zweite einfügende Prozessor fügt die Nichtterminale oder das Terminal der rechten Seite der betreffenden Regel ein. Der löschende Prozessor entfernt daraufhin die Markierungssymbole. In [6] ist die Arbeitsweise des Netzwerkes detaillierter erläutert.
Wird als Menge T die Menge der Terminalsymbole der Grammatik G genommen, so bilden die vom Netzwerk erzeugten Wörter, die nur aus Terminalsymbolen bestehen, gerade die Sprache L.

Nimmt man einen weiteren Prozessor hinzu, der die terminalen Wörter „auffängt“, so erhält man ein Netzwerk $\mathcal{N}'$, das genau die Sprache L erzeugt. Dieser Prozessor darf nur Regeln haben, die auf kein Wort der Sprache L anwendbar sind, damit er die ankommenden Wörter nicht noch verändern kann. Da einfügende Regeln immer anwendbar sind, muss ein löschender Prozessor genommen werden. □

Folgerung 3.2 *Es gibt ein Netzwerk $\mathcal{N}$ evolutionärer Prozessoren mit zwei einfügenden und einem löschenden Knoten, so dass $L(\mathcal{N})$ eine nicht-rekursive Sprache ist.*

Beweis. Da die Familie der rekursiven Sprachen unter Durchschnitt mit Mengen T^*, wobei T ein Alphabet ist, abgeschlossen ist, erzeugt das Netzwerk im Beweis von Satz 3.1 zu einer nicht-rekursiven Sprache L eine ebenfalls nicht-rekursive Sprache. □

Literatur

[1] A. ALHAZOV, C. MARTIN-VIDE und YU. ROGOZHIN, On the number of nodes in universal networks of evolutionary processors. *Acta Inf.* **43** (2006) 331–339.

[2] J. CASTELLANOS, C. MARTIN-VIDE, V. MITRANA und J. SEMPERE, Solving NP-complete problems with networks of evolutionary processors. In: *Proc. IWANN*, Lecture Notes in Computer Science **2084**, Springer-Verlag, Berlin, 2001, 621–628.

[3] J. CASTELLANOS, C. MARTIN-VIDE, V. MITRANA und J. SEMPERE, Networks of evolutionary processors. *Acta Informatica* **39** (2003) 517–529.

[4] E. CSUHAJ-VARJÚ und A. SALOMAA, Networks of parallel language processors. In: *New Trends in formal Language Theory* (Hrsg. GH. PĂUN und A. SALOMAA), Lecture Notes in Computer Science **1218**, Springer-Verlag, Berlin, 1997, 299–318.

[5] J. DASSOW und GH. PĂUN, *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin, 1989.

[6] J. DASSOW und B. TRUTHE, On the Power of Networks of Evolutionary Processors. Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik, Technical report, 2007.

[7] J. DASSOW und B. TRUTHE, On the Power of Networks of Evolutionary Processors. In: *Machines, Computations and Universality, MCU 2007, Orléans, France, September 10–13, 2007, Proceedings*. 2007.

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

Extended Spiking Neural P Systems with Spikes of Limited Lifetime

Rudolf FREUND[1], Mihai IONESCU[2], Marion OSWALD[1]

[1]Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9–11, A–1040 Vienna, Austria
E-mail: {rudi,marion}@emcc.at

[2] Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: armandmihai.ionescu@urv.cat

1 Introduction

Spiking neural P systems (in short SNP systems), introduced in [4], have as core concept the idea of neural communication through electrical pulses called *spikes*, or *action potential* (for details on spiking neurons see, e.g., [3], an introduction to membrane computing can be found in [5] and an up-to-date information on this area is available online at [7]).

SNP systems are represented as a graph with the neurons placed in the nodes of the graph. They are sending signals (spikes) along the *synapses* (the edges of the graph) if the *spiking rules* inside each neuron can be activated. The system is synchronized but it works sequentially at the level of the neurons: in every step at most one rule is used in each of them.

We here consider extended variants of spiking neural P systems with decaying spikes (i.e., the spikes have a limited lifetime) and/or total spiking (i.e., the whole contents of a neuron is erased when it spikes). Although we use the extended model of spiking neural P systems, these restrictions of decaying spikes and/or total spiking do not allow for the generation or the acceptance of more than regular sets of natural numbers.

2 Extended Spiking Neural P Systems

For the basic elements of formal language theory needed in the following, we refer to any monograph in this area, in particular, to [6].

The following definition is mainly taken from [1]:

An *extended spiking neural P system* (of degree $m \geq 1$) (in the following simply called *ESNP system*) is a construct

$$\Pi = (m, S, R)$$

where

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

A *computation* is a sequence of configurations starting with the initial configuration given by S. In every step of the computation, at most one rule is used in each neuron. A computation is called *successful* if it halts, i.e., for no neuron, a rule can be activated. For a more detailed description even incorporating delays in the rules as well as along the axons we refer to [2].

An ESNP system is called *finite* if all the regular checking sets in the rules are finite.

In this paper, we will consider the following variants of the above systems:

1. *ESNP systems with decaying spikes:*

   The spikes only have a limited lifetime before disappearing. In this case, a spike a is now written in the form (a, e), where $e \geq 1$ is the decay that, from the moment a spike (a, e) arrives in a neuron, is decremented by one in each step of the computation. As soon as $e = 0$, the corresponding spike is lost and cannot be used anymore.

2. *ESNP systems with total spiking:*

   In this case, the whole contents of the neuron is lost as soon as a spiking rule $(i, E/ \to P)$ (we omit specifying the number of spikes to be consumed when applying such a rule) can be applied in neuron i as the number of spikes present in the cell is in E.

For decaying spikes and total spiking and even a combination of these two, we will consider ESNP systems as generating as well as accepting devices, where the output (input in the case of accepting devices, respectively) is either given in a specified output (input) neuron, or else as the distance between the first two spikes exiting (entering) the system.

3 Results

When we consider the output to be the number of spikes at the end of a successful computation, then ESNP systems with decaying spikes can only generate finite sets, because the number of spikes that can be added in one step to the contents of a neuron is bounded, but the spikes in a neuron have a limited life-time, hence, at any moment the number of spikes in a neuron is bounded. On the other hand, every finite set of natural numbers can be generated by an extended spiking neural P system with spikes of minimal decay with only two neurons:

Example 1 *Let N be a finite set of natural numbers. We now construct the finite ESNP system Π that generates an element of N by the number of spikes contained in the output neuron 1 at the end of a successful computation:*

$$\begin{aligned} \Pi &= (2, S, R), \\ S &= \{(1, \lambda), (2, (a, 1))\}, \\ R &= \left\{ \left(2, \{(a,1)\} / (a,1) \rightarrow \left\{ \left(1, (a,1)^j\right) \right\}\right) \mid j \in N \right\}; \end{aligned}$$

after one step, every computation halts, the output neuron having received a number of spikes corresponding to a number from N. We could even add the feature of total spiking or minimal threshold 1 in order to obtain the same result; in this case, R would be written as $\left\{ \left(2, \{(a,1)\} / \rightarrow \left\{ \left(1, (a,1)^j\right) \right\}\right) \mid j \in N \right\}$ or $\left\{ \left(2, \geq 1 / \rightarrow \left\{ \left(1, (a,1)^j\right) \right\}\right) \mid j \in N \right\}$.

In sum, the ESNP systems with decaying spikes (even together with total spiking) generating the result as the number of spikes in the output neuron at the end of a successful computation characterize the finite sets of natural numbers.

Theorem 2 *Any finite set of natural numbers N can be generated by a ESNP system with spikes of minimal decay with only two neurons (even with total spiking, too). On the other hand, every set of natural numbers generated in the output neuron by an ESNP system with decaying spikes (even with total spiking, too) is finite.*

If we only consider the output to be the difference between the first two spikes arriving in the output neuron during a halting computation, then we

obtain a characterization of the regular sets of natural numbers even with ESNP systems with decaying spikes (an example is given in [2]).

In [1] it was shown that every ESNP system where the number of spikes remains bounded can only generate regular sets. The same arguments used to prove this result immediately show that ESNP systems with decaying spikes can only generate regular sets because the number of spikes is bounded in these cases and therefore the behaviour of the ESNP systems can be modeled by a (non-deterministic) finite automaton. Yet the same also holds true for ESNP systems with total spiking (see Theorem 4 in [2]).

Theorem 3 *Every language generated by an ESNP system with total spiking is regular.*

Now we consider the *accepting case* where the case of the input being given as the number of spikes in the input neuron (we always assume that the input neuron gets its input only from the environment) is quite trivial:

Example 4 *Let N be a regular set of natural numbers. Then N is accepted by the ESNP system with decaying spikes and/or total spiking*

$$\begin{aligned} \Pi(N) &= (2, \{(1,\lambda),(2,\lambda)\}, R(N)), \\ R(N) &= \{(1, L(\mathbb{N}-N) / \rightarrow \{(2,(a,1))\}), (2, \{a\} / \rightarrow \{(2,(a,1))\})\}. \end{aligned}$$

The input n is given by $(a,1)^n$ in the first neuron which fires if and only if $n \notin N$, hence the infinite loop in the second neuron is only started in this case, whereas for $n \in N$ the system immediately halts.

Again similar arguments as for the generating case can be applied when considering ESNP systems accepting a number given as this number of spikes in the input neuron or else as the difference between the (first) two spikes introduced in the input neuron from the environment.

In sum, we get the following results characterizing *REG* for the accepting cases:

Theorem 5 *Any regular set $N \in REG$ can be accepted by an ESNP system with decaying spikes and/or total spiking, the input either being given in the input neuron or else being taken as the difference between the first two spikes arriving in the input neuron during a successful computation. On the other hand, every language accepted by an ESNP system with decaying spikes and/or total spiking, the input either being given in the input neuron or else being taken as the difference between the first two spikes arriving in the input neuron from the environment during a successful computation, is regular.*

Acknowledgements

The work of Marion Oswald is supported by FWF-project T225-N04.

References

[1] A. Alhazov, R. Freund, M. Oswald, M. Slavkovik: Extended Spiking Neural P Systems Generating Strings and Vectors of Non-Negative Integers. In: H.J. Hoogeboom, Gh. Păun, G. Rozenberg: Workshop on Membrane Computing, WMC7, Leiden, The Netherlands 2006, LNCS 436, Springer, 2007, 123-134.

[2] R. Freund, M. Ionescu, M. Oswald: Extended Spiking Neural P Systems with Decaying Spikes and/or Total Spiking. Accepted for ACMC 2007.

[3] W. Gerstner, W Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge Univ. Press, 2002.

[4] M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae* **71**, 2–3 (2006), 279–308.

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

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

[7] The P Systems Web Page, `http://psystems.disco.unimib.it`

Symport/Antiport Systems with Partial Halting

Rudolf FREUND and Marion OSWALD
Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9–11, A–1040 Vienna, Austria
{rudi,marion}@emcc.at

Abstract

For symport/antiport systems, we consider *partial halting*, i.e., a computation is called halting if no multiset of rules containing a rule from each set of rules assigned to the membranes is applicable anymore. This new variant of partial halting is especially investigated for symport/antiport systems working in the minimally parallel derivation mode.

1 Introduction

In the first papers on membrane computing of Gheorghe Păun (see [6]) as well as of Jürgen Dassow and Gheorghe Păun (see [3]) membrane systems were introduced as systems with a hierarchical (tree-like) structure and the rules being applied in a maximally parallel manner; the results were taken as the contents of a specified output membrane in the final configurations of halting computations, i.e., at the end of computations to which no rule was applicable anymore. In this paper, we consider a variant of halting introduced just recently (see [4]), i.e., we consider a computation to halt as soon as no multiset of rules containing a rule from each set of rules assigned to the membranes is applicable anymore (*partial halting*), which reflects the idea that a (biological) system only stays alive as long as all its main components are still able to evolve in parallel. Moreover, we investigate the derivation mode of *minimal parallelism* (e.g., see [2]), where we may only apply a multiset of rules that includes at least one rule from every set of rules containing applicable rules.

2 P Systems with Symport/ Antiport Rules

$\mathbb{N}$ denotes the set of non-negative integers. By NRE and $NREG$ we denote the family of recursively enumerable sets and regular sets of non-negative integers, respectively.

A P system (of degree $d \geq 1$) *with symport/ antiport rules* (see [5]) is a construct

$$\Pi = (O, \mu, w_1, \cdots, w_d, R_1, \cdots, R_d, i_0) \quad \text{where}$$

- O is the alphabet of *objects*,
- μ is the *membrane structure* consisting of d membranes;
- w_i, $1 \leq i \leq d$, are strings over O representing the *initial* multiset of *objects* present in the membrane regions of the system,
- R_i, $1 \leq i \leq d$, are finite sets of *symport/ antiport rules* of the form x/y, for some $x, y \in O^*$, associated with membrane i; if $|x|$ or $|y|$ equals 0 then we speak of a *symport rule*, otherwise we call it an *antiport rule*; yet as this distinction is not relevant here, in the following we shall also speak of *antiport P systems* instead of P systems with symport/ antiport rules;
- i_o, $1 \leq i \leq d$, specifies the *output* membrane of Π.

An antiport rule of the form $x/y \in R_i$ means moving the objects specified by x from membrane i to the surrounding membrane j (to the environment, if i is the outermost membrane, i.e., if $i = 1$), at the same time moving the objects specified by y in the opposite direction; the weight of the antiport rule x/y is defined as $\max\{|x|, |y|\}$. We assume the environment to contain all objects in an unbounded number.

The computation starts with the multisets specified by $w_1, \ldots, w_m$ in the m membranes. In the *maximally parallel derivation mode*, in each time unit we choose a multiset of rules in such a way that, after identifying objects inside and outside the corresponding membranes to be affected by the selected multiset of rules, no objects remain to be subject to any additional rule at any membrane. In the case of *total halting,* the computation is successful if and only if no rule can be applied anymore; the output then is the number of symbols in the output membrane i_0. If instead of the total halting we take *partial halting*, we stop when no multiset of rules containing a rule from each set of rules assigned to the membranes is applicable anymore.

When using the *minimal derivation mode (min)*, we may only apply a multiset of rules that includes at least one rule from every set of rules containing applicable rules. In the *asynchronous (asyn)* and the *sequential derivation mode (sequ)*, in each derivation step we apply an arbitrary number of rules/ exactly one rule, respectively.

The family of sets of non-negative integers computed as above by antiport P systems with at most d membranes and antiport rules of weight at most k in the derivation mode X, $X \in \{max, min, asyn, sequ\}$, and the halting mode Y, $Y \in \{H, h\}$, is denoted by $NOP_d(anti_k, X, Y)$. When any of the parameters d and k is not bounded, it is replaced by $*$.

3 Results

After stating some general results for the new variant of *partial halting*, which immediately yield comparable computational completeness results for symport/antiport systems with total and with partial halting, we exhibit that working in the asynchronous or in the sequential derivation mode we can only obtain regular sets of non-negative integers with partial halting as with total halting. Moreover, symport/antiport P systems working in the minimally parallel derivation mode with partial halting only yield regular sets, too.

3.1 General Observations

Looking carefully into the definitions of the derivation modes as well as the halting modes explained above, we observe the following general results:

Theorem 1 *Any variant of P systems generating a set of non-negative integers M when working in any of the derivation modes max, min, $asyn$, or $sequ$, with only one set of rules assigned to a single membrane and stopping with total halting yields the same set M when stopping with partial halting, too.*

Theorem 2 *Any variant of P systems generating a set of non-negative integers M when working in the sequential or the asynchronous mode, with only one set of rules assigned to a single membrane and stopping with total or partial halting, respectively, yields the same set M when working in the minimally parallel derivation mode and stopping with the corresponding halting mode, too.*

3.2 Results for Symport/Antiport Systems

The following results are well known (e.g., see [7]):

Theorem 3 $NOP_1\,(anti_2, max, H) = NRE.$

Theorem 4 *For every $X \in \{asyn, sequ\}$,*

$$NOP_*\,(anti_*, X, H) = NOP_1\,(anti_2, X, H) = NREG.$$

Just recently, for minimal parallelism, the result established in [2] needing three membranes was improved to the following one in [1]:

Theorem 5 $NOP_2\,(anti_2, min, H) = NRE.$

The general result in Theorem 1 and the special result in Theorem 3 immediately yield the following result:

Corollary 6 $NOP_1\,(anti_2, max, h) = NRE.$

With the other derivation modes and partial halting, we only get regular sets:

Theorem 7 *For every* $X \in \{min, asyn, sequ\}$,

$$NOP_*\,(anti_*, X, h) = NREG.$$

Comparing the results for total and partial halting for the minimally parallel derivation mode, we realize that

$$NREG = NOP_*\,(anti_*, min, h) \subsetneqq NOP_*\,(anti_*, min, H) = NRE,$$

i.e., in the case of the minimally parallel derivation mode the halting condition – total in contrast to partial halting – makes the difference. Intuitively speaking, the requirement for a computation to continue only if there exists at least one applicable multiset of rules containing at least one rule from each set of rules, together with the minimally parallel derivation mode means that only multisets of rules containing at least one rule from each set of rules assigned to the membranes can be applied and therefore we do not have the possibility of appearance checking (as in matrix or programmed grammars) and thus cannot simulate the zero test for register machines, hence, we cannot obtain computational completeness.

Moreover, as from Theorems 1 and 7 we know that

$$NREG = NOP_1(anti_*, min, h) = NOP_1(anti_*, min, H),$$

we infer that the result from Theorem 5, i.e.,

$$NOP_2\,(anti_2, min, H) = NRE,$$

is optimal with respect to the number of membranes.

Acknowledgement

The work of Marion Oswald was supported by FWF-project T225-N04.

References

[1] A. Alhazov, R. Freund, M. Oswald, S. Verlan, Partial versus total halting in P systems, in: M. A. Gutiérrez-Naranjo, Gh. Păun, A. Romero-Jiménez, Agustín Riscos-Núñez (Eds.), *Proc. 5th Brainstorming Week on Membrane Computing*, Sevilla, 2007, 1–20.

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

[3] J. Dassow, Gh. Păun, On the power of membrane computing, *Journal of Universal Computer Science* **5** (2) (1999), 33–49.

[4] R. Freund, M. Oswald, Partial halting in P systems, *to appear*.

[5] A. Păun, Gh. Păun, The power of communication: P systems with symport/ antiport, *New Generation Computing* **20**, 3 (2002), 295–306.

[6] Gh. Păun, Computing with membranes, *J. of Computer and System Sciences* **61**, 1 (2000), 108–143, and TUCS Research Report 208 (1998) (`http://www.tucs.fi`).

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

Forgetting Automata and Closure Properties

Jens Glöckler
Institut für Informatik, Universität Giessen
Arndtstr. 2, D-35392 Giessen, Germany
Jens.Gloeckler@math.uni-giessen.de

Abstract

Forgetting automata are linear bounded automata which can only use the operations 'move', 'erase' (rewrite with a blank symbol) and 'delete' (remove completely). Beside the language families that coincide with the family REG of regular languages and the family CFL of context-free languages, there remain eight nondeterministic and ten deterministic language families for which the closure properties are investigated. Thereby we mainly focus on the language families between REG and CFL.

Introduction

Forgetting Automata have been introduced by Jančar, Mráz, and Plátek and have been studied in a number of papers in the 1990s [1, 2, 3, 4, 5, 6, 7] and recently in [8, 9]. The main purpose of these types of automata was to model linguistic principles, namely the *analysis by reduction* occurring in Slavic languages like Czech. The goal of the analysis by reduction is to shorten a sentence whilst retaining the syntactical correctness or incorrectness. If this process is iterated, there will finally remain a short sentence which can easily be tested for being correct or not.

The operations a forgetting automaton can use on its input tape are one or more of the following six: $\mathsf{MV_L}$, $\mathsf{MV_R}$ (move the head to the left/right), $\mathsf{ER_L}$, $\mathsf{ER_R}$ (erase, i.e. overwrite the current field with a blank symbol and move to the left/right), $\mathsf{DL_L}$ and $\mathsf{DL_R}$ (delete, i.e. completely remove the current field and move to the left/right).

In this paper we investigate the closure properties of the families of languages accepted by forgetting automata.

Preliminaries

Let A^* denote the set of all words over the finite alphabet A. The empty word is denoted by ε. The reversal of a word w is denoted by w^R and the length of w by $|w|$. The number of occurrences of an alphabet symbol $a \in A$

in a word $w \in A^*$ is denoted by $|w|_a$. Set inclusion and strict set inclusion are denoted by $\subseteq$ and $\subset$, respectively. We write $\mathscr{L}(X)$ for the family of languages accepted by devices of type X and $\mathscr{L}_{\det}(X)$ for the family of languages accepted by deterministic devices X.

Formal Definition

A *forgetting automaton* is a system $\mathcal{A} = \langle S, A, \triangleright, \triangleleft, \sqcup, O, \delta, s_0, F \rangle$, where S is a finite set of states, A is the input alphabet, $\triangleright, \triangleleft \notin A$ are the left and the right sentinels, $\sqcup \notin A$ is the blank symbol used for erasing, O is a set of operations (see below), $\delta : S \times (A \cup \{\triangleright, \triangleleft, \sqcup\}) \to 2^{S \times O}$ is the transition function, $s_0 \in S$ is the initial state and $F \subseteq S$ is the set of final states. O consists of one or more of the following operations:

$\mathsf{MV_L}$, $\mathsf{MV_R}$: move head to the left and to the right, resp.,

$\mathsf{DL_L}$, $\mathsf{DL_R}$: delete current field and move head to the left and to the right, resp.,

$\mathsf{ER_L}$, $\mathsf{ER_R}$: erase current field with $\sqcup$ and move head to the left and to the right, resp.

If $\mathcal{A}$ reads $\triangleright$ (resp. $\triangleleft$), it always implies an $\mathsf{MV_R}$-operation (resp. $\mathsf{MV_L}$-operation), even if $\mathsf{MV_R}, \mathsf{MV_L} \notin O$. Generally, a forgetting automaton is nondeterministic. A forgetting automaton is deterministic if $|\delta(s, x)| \leq 1$ for all $s \in S$ and $x \in (A \cup \{\triangleright, \triangleleft, \sqcup\})$.

A configuration of a forgetting automaton $\mathcal{A}$ is a string $w_1 s w_2$, where the word $w_1 w_2 \in \triangleright(A \cup \{\sqcup\})^* \triangleleft$ is the content of the list, s is the current state and $\mathcal{A}$ reads the first symbol of w_2. By $\vdash$ we denote the relation which describes the change of configurations according to δ; $\vdash^*$ is the reflexive, transitive closure of $\vdash$. An input word is accepted by $\mathcal{A}$ if there is a computation, starting in the initial configuration $s_0 \triangleright w \triangleleft$, which reaches a configuration with an accepting state.

In case O contains both versions X_L and X_R of an operation, we write X for short. A forgetting automaton with a certain set of operations, e.g. $\mathsf{MV_R}$ and DL, is called $(\mathsf{MV_R}, \mathsf{DL})$-automaton. For the family of languages accepted by such automata we write $\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$.

Closure Properties

We omit the trivial classes $\mathscr{L}(\mathsf{MV_L})$, $\mathscr{L}(\mathsf{ER_L})$ and $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER_L})$ below REG. The corresponding automata can only read the first symbol – if any – of an input string and therefore decide whether the first symbol is contained in some subset of the input alphabet.

For an overview of the closure properties between REG and CFL, see Table 1.

	union	marked union	union with regular set	intersection	intersection with regular set	complementation	reversal	concatenation	marked concatenation	iteration	marked iteration	(ε-free) homomorphism
REG $= \mathscr{L}(\mathsf{MV}) = \ldots$	+	+	+	+	+	+	+	+	+	+	+	+
$\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$	+	+	+	−	+	−	?	+	+	+	+	+
$\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$	−	+	+	−	+	+	−	−	+	−	+	−
$\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$	+	+	+	−	+	−	?	+	+	+	+	?
$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$	−	+	+	−	+	+	−	−	+	−	+	−
$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$	−	+	+	−	+	+	−	−	+	−	+	−
$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$	−	+	+	−	+	+	−	−	+	−	+	−
CFL $= \mathscr{L}(\mathsf{MV_R}, \mathsf{ER})$	+	+	+	−	+	−	+	+	+	+	+	+
DCFL	−	+	+	−	+	+	−	−	+	−	+	−

Table 1: Closure properties of language families up to CFL

References

[1] Jančar, P.: Nondeterministic forgetting automata are less powerful than deterministic linear bounded automata. Acta Mathematica et Informatica Universitatis Ostraviensis **1** (1993) 67–74

[2] Jančar, P., Mráz, F., Plátek, M.: Characterization of context-free languages by erasing automata. In: Proc. MFCS 1992. Volume 629 of Lecture Notes in Computer Science., Springer-Verlag (1992) 305–314

[3] Jančar, P., Mráz, F., Plátek, M.: Forgetting automata and the Chomsky hierarchy. In: Proc. SOFSEM 1992, Masaryk University, Brno, Institute of Computer Science (1992) 41–44

[4] Jančar, P., Mráz, F., Plátek, M.: A taxonomy of forgetting automata. In: Proc. MFCS 1993. Volume 711 of Lecture Notes in Computer Science., Springer-Verlag (1993) 527–536

[5] Jančar, P., Mráz, F., Plátek, M.: Forgetting automata and context-free languages. Acta Informatica **33** (1996) 409–420

[6] Mráz, F., Plátek, M.: A remark about forgetting automata. In: Proc. SOFSEM 1993, Masaryk University, Brno, Institute of Computer Science (1993) 63–66

[7] Mráz, F., Plátek, M.: Erasing automata recognize more than context-free languages. Acta Mathematica et Informatica Universitatis Ostraviensis **3** (1995) 77–85

[8] Glöckler, J.: Forgetting automata and unary languages. In: Proc. CIAA 2006. Volume 4094 of Lecture Notes in Computer Science., Springer-Verlag (2006) 186–197

[9] Glöckler, J.: Forgetting automata and unary languages. International Journal of Foundations of Computer Science **18** (2007) 813–827

On a Non-Context-Free Extension of PDL

Stefan Göller *
Universität Leipzig, Institut für Informatik, Germany
goeller@informatik.uni-leipzig.de

Dirk Nowotka
Universität Stuttgart, FMI, Germany
dirk.nowotka@informatik.uni-stuttgart.de

Abstract

Over the last 25 years, a lot of work has been done on seeking for decidable non-regular extensions of Propositional Dynamic Logic (PDL). Only recently, an expressive extension of PDL, allowing visibly pushdown automata (VPAs) as a formalism to describe programs, was introduced and proven to have a satisfiability problem complete for deterministic double exponential time. Lately, the VPA formalism was extended to so called k-phase multi-stack visibly pushdown automata (k-MVPAs). Similarly to VPAs, it has been shown that the language of k-MVPAs have desirable effective closure properties and that the emptiness problem is decidable. On the occasion of introducing k-MVPAs, it has been asked whether the extension of PDL with k-MVPAs still leads to a decidable logic. This question is answered negatively here. We prove that already for the extension of PDL with 2-phase MVPAs with two stacks satisfiability becomes Σ^1_1-complete.

Propositional Dynamic Logic (PDL) is a modal logic introduced by Fischer and Ladner [2] which allows to reason about regular programs. In PDL, there are two syntactic entities: formulas, built from boolean and modal operators and interpreted as sets of worlds of a Kripke structure; and programs, built from the operators test, union, composition, and Kleene star and interpreted as binary relations in a Kripke structure. Thence, the occuring programs can be seen as a regular language over an alphabet that consists of tests and atomic programs. However, the mere usage of regular programs limits the expressiveness of PDL as for example witnessed by the set of executions of well-matched calls and returns of a recursive procedure, cf. [3]. Therefore, non-regular extensions of PDL have been studied quite extensively [3, 4, 5, 6]. An extension of PDL by a class $\mathcal{L}$ of languages means that in addition to regular languages also languages in $\mathcal{L}$ may occur in modalities of formulas.

*The author is supported by the DFG project GELO.

One interesting result on PDL extensions, among many others as summarized in [3], is that already the extension of PDL with the *single* language $\{a^n b a^n \mid n \geq 1\}$ leads to an undecidable logic [4]. In contrast to this negative result, Harel and Raz proved that adding to PDL a *single* language accepted by a single-minded pushdown automaton yields a decidable logic [7]. A simple-minded pushdown automaton is a restricted pushdown automaton, where each input symbol determines the next control state, the stack operation and the stack symbol to be pushed, in case a push operation is performed. Generalizing this concept, Alur and Madhusudan proposed in [8] visibly pushdown languages which are defined as languages accepted by visibly pushdown automata (VPAs). A VPA is a pushdown automaton, where the stack operation is determined by the input in the following way; the alphabet is partitioned into letters that prompt a push, internal, or pop action, respectively. Note that it is well-known that visibly pushdown automata are strictly more powerful than simple-minded pushdown automata. Recently, also for the model of visibly pushdown languages, a PDL extension has been investigated by Löding, Lutz, and Serre [5]. They proved that satisfiability of this PDL extension is complete for deterministic double exponential time. Note that for this result, every visibly pushdown language occuring in a formula must be over the same partition of the alphabet.

Recently, k-phase multi-stack visibly pushdown automata (k-MVPAs), a natural extension of VPAs, have been introduced in [9]. A k-MVPA is an automaton equipped with n stacks where, again, the actions on the stacks are determined by the input, more precisely, every input symbol specifies on which stack a push or pop operation or whether an internal operation is done. Moreover, a k-MVPA is restricted to accept only words that can be obtained by concatenating at most k *phases*, where a phase is a sequence of input symbols that invoke pop actions from at most one stack. Note that k-MVPAs with one stack coincide with VPAs.

Due to the various effective closure properties and a decidable emptiness problem of the language class described by k-MVPAs, it is an interesting question to ask if the corresponding extension of PDL is still decidable. This question was raised in [9] and is answered negatively in this article. We prove Σ_1^1-completeness for this PDL extension. A Σ_1^1 lower bound already holds, if we restrict ourselves to deterministic 2-MVPAs with two stacks. This is the weakest possible instance of k-MVPAs that is still more powerful than VPAs. Our proof relies on the same technique of the Σ_1^1-hardness proof of undecidability of PDL extended with the single language $\{a^n b a^n \mid n \geq 1\}$, which is presented in [3]. Note however, that $\{a^n b a^n \mid n \geq 1\}$ is not recognized by any k-MVPA for any k.
The article can be found here [1].

References

[1] S. Göller, D. Nowotka, On a Non-Context-Free Extension of PDL, submitted. Arxiv-Link: `http://arxiv.org/abs/0707.0562`

[2] M. J. Fischer, R. E. Ladner, Propositional dynamic logic of regular programs, J. Comput. Syst. Sci.

[3] D. Harel, D. Kozen, J. Tiuryn, Dynamic Logic, Foundations of computing, The MIT Press, 2000.

[4] D. Harel, A. Pnueli, J. Stavi, Propositional dynamic logic of nonregular programs, J. Comput. System Sci. 26 (2) (1983) 222–243.

[5] C. Löding, C. Lutz, O. Serre, Propositional dynamic logic with recursive programs, J. Log. Algebr. Program.To appear.

[6] T. Koren, A. Pnueli, There exist decidable context-free propositional dynamic logics, in: Proceedings of the Carnegie Mellon Workshop on Logic of Programs, Springer-Verlag, London, UK, 1984, pp. 290–312.

[7] D. Harel, D. Raz, Deciding properties of nonregular programs, SIAM J. Comput. 22 (4) (1993) 857–874.

[8] R. Alur, P. Madhusudan, Visibly pushdown languages, in: Proceedings of the 36th Annual ACM Symposium on Theory of Computing, ACM, New York, 2004, pp. 202–211 (electronic).

[9] S. La Torre, P. Madhusudan, G. Parlato, A robust class of context-sensitive languages, in: Proceedings of the 22nd IEEE Symposium on Logic in Computer Science, IEEE, 2007.

Learning Finite-State Machines from Inexperienced Teachers

Olga Grinchtein*
Department of Computer Systems, Uppsala University, Sweden

Martin Leucker
Institut für Informatik, TU München, Germany

– Extended Abstract –

The general goal of query-based learning algorithms for finite-state machines is to identify a *machine*, usually of *minimum* size, that *agrees* with an *a priori* fixed (class of) machines. For this, *queries* on how the underlying system behaves may be issued.

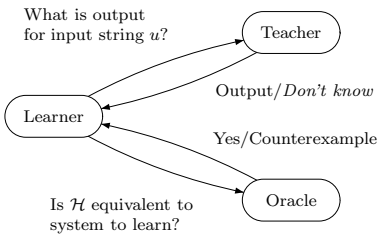
A popular setup is that of Angluin's L^* algorithm[Ang87], here adapted to the case of finite-state machines, in which a minimal deterministic finite-state machine for a regular language is learned based on so-called *membership* and *equivalence queries*. Using a pictorial language, we have a *learner* whose job is to come up with the automaton to learn, a *teacher* who may answer the output for a given input string as well an *oracle* answering whether the automaton $\mathcal{H}$ currently proposed by the learner is correct or not. This setting is depicted in Figure 1(a) (though assume that the *don't know* is not there).

In Angluin's setting, a teacher will always answer with the correct output symbol. In many application scenarios, however, parts of the machine to learn are not completely specified or not observable. Then, queries may be answered inconclusively, by *don't know*, also denoted by ?.

In the full version of this paper [GL06], we study a learning algorithm (and variants thereof), called ABSAT, ABSATi, and ABSATi2, that are designed to work with such an *inexperienced* teacher. The oracle, however, does not change its functionality in the setting discussed here (see Figure 1(a), the *don't know* is new).

In general, two types of learning algorithms for FSMs can be distinguished, so-called *online* and *offline* algorithms. Online algorithms, such as Angluin's L^* algorithm, query strings to the teacher. Offline algorithms get a fixed set of examples and no further queries are allowed before computing

*Part of the work has been done during the author's stay at TU München supported by the C F Liljewalchs fellowship, Uppsala University.



(a) L^* versus ABSAT

	bica	ABSAT	ABSATi	ABSATi2
	time	time	time	time
fo.70	n.a.	39.50	38.57	38.69
th.55	188.87	0.45	0.43	0.43
vmebus.master.m	59.93	1627.37	693.44	93.01
ifsm2	14.08	34.82	34.36	30.78
pe-send-ifc.fc	2.33	6.93	2.75	2.07
pe-send-ifc.fc.m	0.85	6.93	2.75	1.07
rubin1200	10.24	10.28	10.26	10.02
rubin2250	64.78	64.69	64.74	63.29
vbe4a	1.94	4.31	1.86	1.56
ex2	0.78	0.78	0.76	4.67

(b) Running times compared

Figure 1: The setup for the learning algorithms and their performance

a minimal FSM conforming to the examples. Typical algorithms of this type are based on a characterization in terms of a constraint satisfaction problem (CSP) over the natural numbers due to Biermann [BF72].

Faced with an inexperienced teacher, we cannot rely completely on Angluin's algorithm. We therefore define an algorithm that is a combination of an online algorithm and an offline algorithm and is based on [OS98]. Similar to Angluin's algorithm, we round off the information on the automaton in question by asking queries. As queries can be answered by *?*, we may not be able to complete the information as in Angluin's setting to compute an FSM directly. For this, we use Biermann's approach for obtaining an FSM based on the enriched information. Our combination is conservative in the sense that in case all queries are answered by either *yes* or *no*, we obtain the same efficiency as for Angluin's algorithm. Furthermore, the encoding in terms of CSP is optimized based on the information collected in Angluin's algorithm.

While in [OS01] an efficient implementation for solving the resulting CSP problem is explained, we give an encoding as a SAT problem featuring a simple yet—as the examples show—very efficient inference algorithm by employing powerful SAT solvers.

Actually, our approach is quite similar to the one proposed in [OS98] and [OS01]. The main difference are that we use SAT solvers for solving the corresponding CSP problem (which gives algorithm ABSAT) and that we additionally propose incremental consistency checks (ABSATi) and an incremental construction of the CSP problem (ABSATi2), which both improves the overall efficiency.

To validate our approach in practice, we have employed our techniques to the problem of reducing incompletely specified finite-state machines. We have implemented our extensions within the C++ program called BICA, used in [OS98] and have reexamined BICA as well as our three versions on the same set of examples studied in [OS98] (see Figure 1(b)).

The overall conclusion is that although the behavior of a SAT solver is not completely predictable, our algorithms ABSATi and ABSATi2 are, for

many examples, competitive alternatives to BICA, which especially work on examples that are too complex for BICA.

References

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[BF72] A. W. Biermann and J. A. Feldman. On the synthesis of finite-state machines from samples of their behaviour. *IEEE Transactions on Computers*, 21:592–597, 1972.

[GL06] Olga Grinchtein and Martin Leucker. Learning finite-state machines from inexperienced teachers. Technical Report TUM-I0613, TU München, 2006.

[OS98] A. L. Oliveira and J. P. M. Silva. Efficient search techniques for the inference of minimum size finite automata. In *String processing and information retrieval*, 1998.

[OS01] Arlindo L. Oliveira and João P. Marques Silva. Efficient algorithms for the inference of minimum size dfas. *Machine Learning*, 44(1/2):93–119, 2001.

On the Size of Higman-Haines Sets

Hermann Gruber*and Markus Holzer†and Martin Kutrib‡

A not so well known theorem in formal language theory is that of Higman [5, Theorem 4.4], which reads as follows:

> If X is any set of words formed from a finite alphabet, it is possible to find a *finite* subset X_0 of X such that, given a word w in X, it is possible to find w_0 in X_0 such that the letters of w_0 occur in w in their right order, though not necessarily consecutively.

In fact, this statement is a corollary to a more general theorem on well-partially-ordered sets. Here a partially ordered set is called well-partially-ordered, if every non-empty subset has at least one, but no more than a finite number of minimal elements (finite basis property). For instance, the set A^*, where A is a finite alphabet, under the scattered subword relation $\leq$, i.e., $v \leq w$ if and only if $v = v_1 \dots v_k$ and $w = w_1 v_1 \dots w_k v_k w_{k+1}$, for some integer k, where v_i and w_j are in A^*, for $1 \leq i \leq k$ and $1 \leq j \leq k+1$, is a well-partially-ordered set. Interestingly, the concept of well-partially-orders has been frequently rediscovered, for example, see [4, 5, 6, 7, 8]. Moreover, although Higman's result appears to be only of theoretical interest, it has some nice applications in formal language theory; see, e.g., [2, 3, 7]. It seems that one of the first applications has been given by Haines in [4, Theorem 3], where it is shown that the set of all scattered subwords, i.e., the *Higman-Haines sets*

$$\textsc{Down}(L) \;=\; \{\, v \in A^* \mid \text{there exists } w \in L \text{ such that } v \leq w \,\}$$

and

$$\textsc{Up}(L) \;=\; \{\, v \in A^* \mid \text{there exists } w \in L \text{ such that } w \leq v \,\},$$

are both regular for *any* language $L \subseteq A^*$. As pointed out in [4] this is an exceptional property, which is quite unexpected. It is worth mentioning that the regular languages $\textsc{Down}(L)$ and $\textsc{Up}(L)$ cannot be obtained constructively in general (in terms of finite automata).

*Institut für Informatik, Ludwig-Maximilians-Universität München, Oettingenstraße 67, D-80538 München, Germany. Email: gruberh@tcs.ifi.lmu.de

†Institut für Informatik, Technische Universität München, Boltzmannstraße 3, D-85748 Garching bei München, Germany. Email: holzer@in.tum.de

‡Institut für Informatik, Universität Giessen, Arndtstraße 2, D-35392 Giessen, Germany. Email: kutrib@informatik.uni-giessen.de

Although the basic results for Higman-Haines sets date back to the 1950s and 1960s, surprisingly less is known with respect to the (descriptional) size of these sets. To our knowledge the only paper dealing with effective constructibility issues is [8], where an open problem raised in [4] has been solved, i.e., $\textsc{Down}(L)$ can effectively be constructed for a given context-free grammar G with $L = L(G)$. Moreover, it was also shown that $\textsc{Up}(L)$ can be obtained effectively, if L is a context-free language. This immediately raises the question whether a similar result holds for the family of Church-Rosser languages. This language family lies in between the regular languages and the growing context-sensitive languages, but is incomparable to the family of context-free languages [1]. In fact, we show that for Church-Rosser languages the size of the Higman-Haines sets cannot be bounded by any recursive function; hence we obtain a non-recursive trade-off result, which implies that the Higman-Haines sets cannot effectively be constructed for Church-Rosser languages and all of its supersets. Moreover, we consider the problem of computing the Higman-Haines sets induced by the families of regular, context-free, and linear context-free languages. For the size of the Higman-Haines sets generated by regular languages upper and lower bounds are presented. That is, we prove that an exponential blow-up is sufficient and necessary in the worst case for a deterministic finite automaton to accept the Higman-Haines set $\textsc{Down}(L)$ or $\textsc{Up}(L)$ generated by some language that is represented by another deterministic finite automaton. This nicely contrasts the result about nondeterministic finite automata, where a linear matching upper and lower bound on the size of Higman-Haines sets can be shown. Furthermore, we investigate the descriptional complexity of the Higman-Haines sets when the underlying device is a context-free or linear context-free grammar.

References

[1] Gerhard Buntrock and Friedrich Otto. Growing context-sensitive languages and Church-Rosser languages. *Inform. Comput.*, 141(1):1–36, 1998.

[2] H. Fernau and F. Stephan. Characterizations of recursively enumerable sets by programmed grammars with unconditional transfer. *J. Autom., Lang. Comb.*, 4(2):117–152, 1999.

[3] Robert H. Gilman. A shrinking lemma for indexed languages. *Theoret. Comput. Sci.*, 163:277–281, 1966.

[4] Leonard H. Haines. On free monoids partially ordered by embedding. *J. Comb. Theory*, 6:94–98, 1969.

[5] Graham Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc., Series 2*, 2:326–336, 1952.

[6] Joseph B. Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *J. Comb. Theory*, 13:297–305, 1972.

[7] Jan van Leeuwen. A regularity condition for parallel rewriting systems. *SIGACT News*, 8(4):24–27, 1976.

[8] Jan van Leeuwen. Effective constructions in well-partially-ordered free monoids. *Discrete Mathematics*, 21:237–252, 1978.

Effizient chemisch rechnen durch deterministische Reaktionssysteme mit Regelpriorisierung

T. Hinze R. Faßler T. Lenser N. Matsumaru P. Dittrich

Friedrich-Schiller-Universität Jena
Arbeitsgruppe Biosystemanalyse
Ernst-Abbe-Platz 1–4, D-07743 Jena

{hinze,raf,thlenser,naoki,dittrich}@minet.uni-jena.de

Zusammenfassung

Vielteilchensysteme sind Forschungsgegenstand in Natur- und Lebenswissenschaften. Ihr molekulares Wechselwirkungsgefüge lässt sich sowohl strukturell untersuchen als auch zur Konstruktion chemischer Computer nutzen. Mathematische Beschreibungen erfolgen entweder algebraisch oder analytisch. Während die Abbildung der diskreten Arbeitsweise durch Termersetzungsmechanismen weitreichende Analogien zu Automatenmodellen aufzeigt, gestatten zeit- und wertkontinuierliche Ansätze eine effiziente Parametrisierbarkeit des dynamischen Verhaltens. Motiviert durch das Ziel, beide Sichtweisen in ein gemeinsames Modellgerüst zu integrieren, stellen wir zunächst eine rein algebraische Beschreibung reaktionsfähiger Vielteilchensysteme mittels P-Systemen Π_{PR} vor, die statt der maximal-parallelen Abarbeitung eine konsequente Priorisierung der Reaktionsregeln verwenden. Dadurch wird ein deterministisches Systemverhalten erreicht und zugleich die Möglichkeit einer Masseerhaltung geschaffen. Zusätzlich erleichtert dies den Zugang zur analytischen Methodik z.B. durch Reaktionskinetiken und Fixpunktbestimmungen. Am Beispiel des Rucksackproblems zeigen wir anschließend eine automatengestützte Transformation eines auf dynamischer Programmierung beruhenden Lösungsansatzes in Systeme des Typs Π_{PR}, deren Zeitverhalten für eine Probleminstanz simuliert wird.

1 Einführung

Konzepte des Molecular Computing werden seit mehr als zehn Jahren untersucht. Einen Schwerpunkt bildet hierbei die Frage nach Programmierparadigmen, die die massive Datenparallelität weitmöglichst zur Zeiteffizienzsteigerung einsetzen. Während spezifische Ansätze wie das DNA-Computing submolekulare Strukturen in die Algorithmierung einbeziehen [3], verfolgen künstliche Chemien eine allgemeinere Herangehensweise [2], die auch durch P-Systeme, Modelle des Membrane Computing, aufgegriffen wird [4].

Der nichtdeterministischen Natur chemischer Reaktionsnetzwerke stehen jedoch analytisch-deterministische Beschreibungsmodelle ihres dynamischen Verhaltens gegenüber, die sich auf Reaktionskinetiken und daraus resultierende Differentialgleichungssysteme stützen. P-Systeme, die diskret nach dem Vorbild regelgesteuerter Termersetzungen funktionieren, zeichnen zumeist alle potentiell möglichen Ersetzungspfade nach und betrachten diese als gleichwertig. Diese maximal-parallele Arbeitsweise impliziert Nichtdeterminismus. Die Aufgabe, nichtdeterministische Berechnungsmodelle geeignet zu determinisieren, erweist sich als sinnvoll, um analytische und algebraische Beschreibungen rechentechnisch ineinander transformieren zu können und für die darauf jeweils aufsetzende Untersuchungsmethodik zugänglich zu machen. Bereits das Determinisieren endlicher Automaten zählt zur Klasse der NP-vollständigen Probleme.

Vor diesem Hintergrund soll eine algebraisch-deterministische Beschreibung reaktionsfähiger Vielteilchensysteme im Kontext von P-Systemen entstehen. Es bieten sich zwei Determinisierungsstrategien an: Stochastizität und Regelpriorisierung. Stochastische Modelle ermöglichen die Auswahl des wahrscheinlichsten Pfades. Eine konsequente Regelpriorisierung durch eine vollständige Ordnung in Fortführung der Ideen aus [5] abstrahiert den Aspekt, dass chemische Reaktionen mit unterschiedlichen Aktivierungsenergien behaftet sein können. Zugleich beseitigt man damit eine Unzulänglichkeit maximal-parallel arbeitender P-Systeme, die auftreten kann, wenn die Anzahl der Moleküle nicht ausreicht, um trotz vorhandener Ausgangsstoffe alle darauf anwendbaren Reaktionen gleichzeitig absättigen zu können. Werden diese dennoch ausgeführt, wird dem System mehr Material entnommen als darin vorrätig ist, was das Kriterium der Masseerhaltung verletzt. Nachfolgend werden P-Systeme vom Typ Π_{PR} eingeführt, eine Transformationsvorschrift endlicher Automaten angegeben und eine Lösung des Rucksackproblems demonstriert. Wir zeigen damit über Implementierungen logischer Gatter hinaus, wie man rechenintensive Probleme durch chemische Prinzipien lösen kann, was Anwendungsperspektiven in synthetischer Biologie, Molecular und Organic Computing eröffnet.

2 Multimengenbasierte Systemdefinition

Sei A eine beliebige Menge und $\mathbb{N}$ die Menge der natürlichen Zahlen einschl. null. $\mathcal{P}(A)$ verkörpert die Potenzmenge von A. Eine Multimenge über A ist eine Abbildung $F : A \longrightarrow \mathbb{N} \cup \{\infty\}$. $F(a)$ gibt die Vielfachheit des Vorkommens von $a \in A$ in F an. Multimengen lassen sich als elementweise Aufzählung der Form $\{(a_1, F(a_1)), (a_2, F(a_2)), \ldots\}$ darstellen, wobei $\forall (a, b_1), (a, b_2) \in F : b_1 = b_2$. Der Support $\mathrm{supp}(F) \subseteq A$ von F ist definiert durch $\mathrm{supp}(F) = \{a \in A \mid F(a) > 0\}$. Eine Multimenge F über A ist leer gdw. $\forall a \in A : F(a) = 0$. Seien F_1 und F_2 Multimengen über A. F_1 ist

eine Teilmenge von F_2, notiert $F_1 \subseteq F_2$, gdw. $\forall a \in A : (F_1(a) \leq F_2(a))$. Die Multimengen F_1 und F_2 sind gleich gdw. $F_1 \subseteq F_2 \wedge F_2 \subseteq F_1$. Die Schnittmenge $F_1 \cap F_2 = \{(a, F(a)) \mid a \in A \wedge F(a) = \min(F_1(a), F_2(a))\}$, die Multimengensumme $F_1 \uplus F_2 = \{(a, F(a)) \mid a \in A \wedge F(a) = F_1(a) + F_2(a)\}$ sowie die Multimengendifferenz $F_1 \ominus F_2 = \{(a, F(a)) \mid a \in A \wedge F(a) = \max(F_1(a) - F_2(a), 0)\}$ bilden Multimengenoperationen. Der Term $\langle A \rangle = \{F : A \longrightarrow \mathbb{N} \cup \{\infty\}\}$ beschreibt die Menge aller Multimengen über A.

Ein P-System zur Beschreibung von reaktionsfähigen Vielteilchensystemen mit konsequenter Regelpriorisierung ist ein Tupel

$$\Pi_{\mathrm{PR}} = (V, T, [_1]_1, L_0, R),$$

in welchem V das Arbeitsalphabet und $T \subseteq V$ das Terminalalphabet bilden. $[_1]_1$ bezeichnet den einzigen Reaktionsraum in Anlehnung an die bei P-Systemen übliche Notation. Die Multimenge $L_0 \subset V \times (\mathbb{N} \cup \{\infty\})$ erfasst den Initialinhalt des Systems. Jedes Molekülexemplar wird hierbei als Symbolobjekt ohne räumliche Position betrachtet. Symbolobjekte implizieren keine zusätzlichen Restriktionen für weiterführende Systemimplementierungen, da sie keine Vorgaben für innere Strukturen setzen. Die endliche Menge $R = \{r_1, \ldots, r_k\}$ fasst den verfügbaren Satz chemischer Reaktionen zusammen, wobei sich jede Reaktion $r_i \in \langle E_i \rangle \times \langle P_i \rangle$ aus der endlichen Multimenge der Ausgangsstoffe $E_i \subset V \times \mathbb{N}$ sowie der endlichen Multimenge der Reaktionsprodukte $P_i \subset V \times \mathbb{N}$ ergibt. Die Vielfachheiten der Elemente entsprechen jeweils den stöchiometrischen Faktoren. Eine Reaktionsregel $r_i = (\{(A_1, a_1), \ldots, (A_h, a_h)\}, \{(B_1, b_1), \ldots, (B_v, b_v)\})$ wird nachfolgend in der chemischen Notation $r_i : a_1\, A_1 + \ldots + a_h\, A_h \longrightarrow b_1\, B_1 + \ldots + b_v\, B_v$ geschrieben. Der Index i jeder Reaktionsregel definiert ihre Priorität bei der Ausführung, die für r_1 am höchsten und für r_k am niedrigsten ist.

Die Arbeitsweise von Π_{PR} wird induktiv durch Fortschreibung der Systemkonfigurationen L_t über die Zeit t ausgehend von L_0 nach folgendem Schema definiert:

$$\begin{aligned} L_{t+1} \;=\; \big\{(a, \alpha_{a,k}) \mid \forall a \in V \;\wedge\; \alpha_{a,0} = |L_t \cap \{(a,\infty)\}| \;\wedge\; \beta_{a,i} = |E_i \cap \{(a,\infty)\}| \\ \wedge\; \gamma_{a,i} = |P_i \cap \{(a,\infty)\}| \;\wedge\; i \in \{1, \ldots, k\} \\ \wedge\; \alpha_{a,i} = \begin{cases} \alpha_{a,i-1} + \gamma_{a,i} - \beta_{a,i} & \text{falls } \forall a \in V : \alpha_{a,i-1} \geq \beta_{a,i} \\ \alpha_{a,i-1} & \text{sonst} \end{cases} \big\} \end{aligned}$$

Die Anwendung einer Reaktionsregel r_i erfolgt dadurch, dass alle benötigten Ausgangsstoffe in der geforderten Molekülzahl aus L_t entnommen und die zugehörigen Reaktionsprodukte hinzugefügt werden. Hierzu geschieht eine Überprüfung, ob die Reaktion abgesättigt werden kann, d.h., ob alle benötigten Ausgangsstoffe auch vorliegen. Zu diesem Zweck werden für jeden Stoff $a \in V$ die Maßzahlen $\alpha_{a,j}$ und $\beta_{a,j}$ bestimmt, die angeben, wieviele Moleküle vorhanden ($\alpha_{a,j}$) bzw. wieviele zur Umsetzung der Reaktion notwendig sind

($\beta_{a,j}$). $\gamma_{a,j}$ spezifiziert entsprechend die im Falle einer Reaktion hinzukommende Molekülanzahl des Stoffes a. Bei der Absättigungsüberprüfung geht das Iterationsschema hierarchisch vor und testet zuerst die Reaktion r_1, führt diese ggf. aus, prüft danach die Reaktion r_2 usw. bis zur Reaktion r_k. Die Abarbeitung von Π_{PR} endet, sobald keine Reaktionsregel mehr anwendbar ist. Allgemein lässt sich die von Π_{PR} erzeugte Ausgabe durch die Elementarsprache

$$L(\Pi_{\mathrm{PR}}) = \mathrm{supp}\left(\biguplus_{t=0}^{\infty}(L_t \cap \{(w,\infty) \mid w \in T\})\right) \subseteq T$$

beschreiben, wobei die Frage $L(\Pi_{\mathrm{PR}}) = \emptyset$? zum Akzeptorverhalten korrespondiert.

3 Transformation endlicher Automaten in Π_{PR}

Gegeben sei ein nichtdeterministischer endlicher Automat (NEA) $M = (Z, \Sigma, \delta, z_0, F)$ mit nichtleerer endlicher Zustandsmenge Z, Alphabet Σ, Überführungsrelation $\delta \subseteq (Z \times \Sigma) \times Z$, Startzustand $z_0 \in Z$ und Finalzustandsmenge $F \subseteq Z$. Zusätzlich gelte: $Z \cap \Sigma = \emptyset$. M akzeptiere die formale Sprache $L(M)$. Aus M lässt sich wie folgt ein verhaltensadäquates P-System Π_{PR} konstruieren:

$$\begin{aligned}
\Pi_{\mathrm{PR}} &= (V, T, [_1]_1, L_0, \{r_1, \ldots, r_{|\delta|}\}) \\
V &= Z \cup \Sigma \cup \{\Phi, C\}, \text{ wobei o.B.d.A. } \Phi, C \notin Z \cup \Sigma \\
T &= \{\Phi\} \\
L_0 &= \{(z_0, q) \mid q = |\{(z_0, x, q') \in \delta\}|\} \cup \\
&\quad \{(w, a_w) \mid w \in \Sigma \ \wedge \ a_w = |\{(z_0, w, q') \in \delta\}|\} \cup \\
&\quad \{(C, \tau)\}, \text{ wobei } \tau \in \mathbb{N} \cup \{\infty\} \text{ (wählbarer Laufzeitparameter)} \\
r_{1\ldots m} &: \ q + A + C \longrightarrow f + \Phi + A \quad \forall (q, A, f) \in \delta \,:\, f \in F \\
r_{m+1\ldots p} &: \ z_0 + A + C \longrightarrow q' + z_0 + A \quad \forall (z_0, A, q') \in \delta \\
r_{p+1\ldots|\delta|} &: \ q + A + C \longrightarrow q' + A \quad \forall (q, A, q') \in \delta \,:\, q \in Z \setminus (F \cup \{z_0\})
\end{aligned}$$

Es gilt: $L(\Pi_{\mathrm{PR}}) = \emptyset$ gdw. $L(M) = \emptyset$. Zusätzlich zu den Molekülen, die die Zustände und Alphabetzeichen von M kodieren, werden zwei neue Molekültypen eingeführt. Φ dient als Marker, der das Erreichen eines Finalzustandes signalisiert. C steht für Clock-Moleküle, deren anfänglicher Vorrat mit jeder Transition schrittweise abgebaut wird. Bei endlicher Initialvorgabe besteht so die Möglichkeit, die Systemkonfigurationen L_t durch absteigendes Sortieren nach der Anzahl C in eine zeitliche Abfolge zu bringen und die während eines Zeitschrittes abgelaufenen Transitionen zu rekonstruieren.

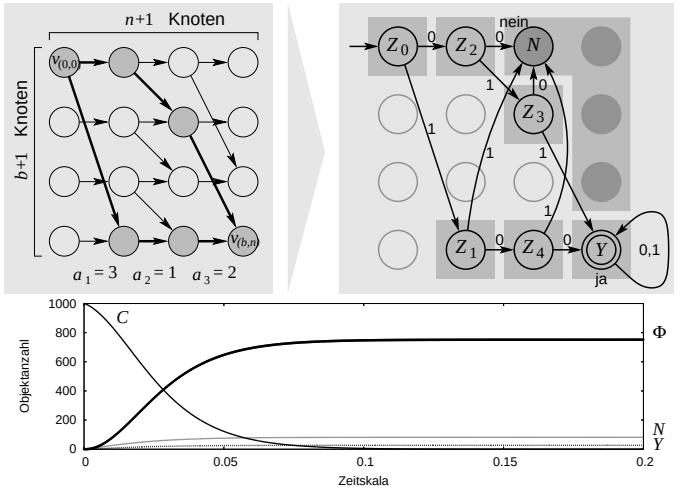


Abbildung 1: Konstruktion von M (oben rechts) aus dem Ansatz zur Lösung des Rucksackproblems mittels dynamischer Programmierung nach [1] (oben links), Simulation des dynamischen Systemverhaltens von Π_{PR} mit Copasi (unten)

4 Beispielanwendung: Rucksackproblem

Das als NP-vollständig bekannte Rucksackproblem ist durch n natürliche Zahlen $a_1, \ldots, a_n$, welche die Gewichte der korrespondierenden Gegenstände $1, \ldots, n$ repräsentieren, und ein Referenzgewicht $b \in \mathbb{N}$ definiert. Es wird gefragt, ob es eine Auswahl $I \subseteq \{1, \ldots, n\}$ aus diesen Gegenständen gibt, so dass $\sum_{i \in I} a_i = b$. Der in [1] vorgestellte Ansatz zur Lösung des Rucksackproblems mittels dynamischer Programmierung greift auf den gerichteten Graphen $\mathcal{G} = (V, E)$ mit einem $(b+1) \cdot (n+1)$-Knotenraster zurück, so dass $V = \{v_{(i,k)} \mid \forall i = 0, \ldots, b \forall k = 0, \ldots, n\}$. Die Kantenrelation $E \subset V \times V$ wird definiert durch $E = \{(v_{(i,k)}, v_{(i,k+1)}) \mid \forall i = 0, \ldots, b \forall k = 0, \ldots, n-1\} \cup \{(v_{(i,k)}, v_{(i+a_i,k+1)}) \mid \forall i = 0, \ldots, b : i + a_i \leq b \forall k = 0, \ldots, n-1\}$. Das Ergebnis lautet „ja“ gdw. es in $\mathcal{G}$ einen Pfad von $v_{(0,0)}$ nach $v_{(b,n)}$ gibt. Abbildung 1 veranschaulicht $\mathcal{G}$ (oben links) und einen daraus resultierenden Automaten M (oben rechts) für die Probleminstanz $n = 3, a_1 = 3, a_2 = 1, a_3 = 2, b = 3$. Ein daraus resultierendes P-System Π_{PR} besitzt die Gestalt:

$\Pi_{\mathrm{PR}} = (\{Z_0, Z_1, Z_2, Z_3, Z_4, N, Y, 0, 1, \Phi, C\}, \{\Phi\}, [\,]_1, L_0, \{r_1, \ldots, r_{12}\})$

$L_0 = \{(Z_0, 2), (0, 1), (1, 1), (C, \tau)\}$

$r_1 : Z_3 + 1 + C \longrightarrow Y + \Phi + 1$
$r_2 : Z_4 + 0 + C \longrightarrow Y + \Phi + 0$
$r_3 : Y + 1 + C \longrightarrow Y + \Phi + 1$
$r_4 : Y + 0 + C \longrightarrow Y + \Phi + 0$
$r_5 : Z_0 + 1 + C \longrightarrow Z_1 + 1 + Z_0$
$r_6 : Z_0 + 0 + C \longrightarrow Z_2 + 0 + Z_0$
$r_7 : Z_1 + 1 + C \longrightarrow N + 1$
$r_8 : Z_1 + 0 + C \longrightarrow Z_4 + 0$
$r_9 : Z_2 + 1 + C \longrightarrow Z_3 + 1$
$r_{10} : Z_2 + 0 + C \longrightarrow N + 0$
$r_{11} : Z_3 + 0 + C \longrightarrow N + 0$
$r_{12} : Z_4 + 1 + C \longrightarrow N + 1$

Abbildung 1 (unten) zeigt die Anreicherung der Markermoleküle Φ (Problemlösung „ja“) sowie den Abbau der Clock-Moleküle (initial: $\tau = 1000$). Der Vorrat an alphabetzeichenkodierenden Molekülen bleibt konstant.

Danksagung. Die vorliegende Arbeit wurde aus Mitteln des 6. EU-Rahmenprogrammes für das Forschungsprojekt ESIGNET (Evolving Cell Signalling Networks in Silico, Projektnr. 12789) und aus Mitteln des BMBF (0312704A) gefördert.

Literatur

[1] E. Baum, D. Boneh. *Running dynamic programming algorithms on a DNA computer.* Proc. DNA2, DIMACS **44**:77-86, 1999

[2] P. Dittrich, J. Ziegler, W. Banzhaf. *Artificial Chemistries – A Review.* Artificial Life **7(3)**:225-275, 2001

[3] T. Hinze, M. Sturm. *Rechnen mit DNA – Eine Einführung in Theorie und Praxis.* Oldenbourg-Wissenschaftsverlag München, Wien, 2004

[4] G. Păun. *Computing with Membranes.* Journal of Computer and System Sciences **61(1)**:108-143, 2000

[5] G. Păun. *Membrane Computing: An Introduction.* Springer-Verlag Berlin 2002

Equivalence of Match-, Change- and Inverse Match-Boundedness for Length-Preserving String Rewriting

Dieter Hofbauer
Berufsakademie Nordhessen
Eichlerstraße 25, D-34537 Bad Wildungen, Germany.

Johannes Waldmann
Hochschule für Technik, Wirtschaft und Kultur (FH) Leipzig
Fb IMN, PF 30 11 66, D-04251 Leipzig, Germany.

1 Introduction

String rewriting is a model of non-deterministic, step-wise computation. To get more detailed information on derivations, positions in strings can be annotated by natural numbers, called heights. That means switching from an alphabet Σ to the annotated alphabet $\Sigma \times \mathbb{N}$. For each rewrite derivation, there is a corresponding annotated derivation. It starts with all annotations equal to zero, and proceeds by computing annotations in the contractum from annotations in the redex (and leaving annotations unchanged elsewhere).

One example of annotated rewriting was given by Ravikumar [Rav04], where the annotations in the contractum are obtained as successors of the annotations at corresponding positions in the redex. These annotations are called *change heights*. This definition makes sense only for length-preserving systems, i.e., rewriting systems where for each rule, the left- and the right-hand side have equal length.

As an example, take the rewriting system $R = \{abb \to bba, bbb \to aaa\}$ over the alphabet $\Sigma = \{a, b\}$, and the R-derivation $\underline{abb}bb \to bb\underline{abb} \to b\underline{bbb}a \to baaaa$, where for each step, the redex is underlined. If we annotate this derivation with change heights, we get

$$\underline{a_0b_0b_0}b_0b_0 \to b_1b_1\underline{a_1b_0b_0} \to b_1\underline{b_1b_2b_1}a_1 \to b_1a_2a_3a_2a_1.$$

The concept of change heights has been generalized by Geser and the present authors [GHW04] to *match heights*: here the annotations in the contractum are defined as the successor of the minimal annotation in the redex. If we annotate the above derivation with match heights, we get

$$\underline{a_0b_0b_0}b_0b_0 \to b_1b_1\underline{a_1b_0b_0} \to b_1\underline{b_1b_1b_1}a_1 \to b_1a_2a_2a_2a_1.$$

If there is an upper bound on the heights of all annotated R-derivations that start from strings annotated by zero, then the string rewriting system R is called annotation-bounded (e.g., change-bounded, match-bounded).

Match-bounded string rewriting systems effectively preserve regularity of languages, and they are terminating. This concept can be used for automated termination provers, also for term rewriting [GHWZ07, KM07].

The inverse of a rewrite system flips all its arrows; in the example, $R^- = \{bba \to abb, aaa \to bbb\}$. Then the above R-derivation corresponds to the R^--derivation $b\underline{aaa}a \to bb\underline{bba} \to \underline{bba}bb \to abbbb$. If we annotate this inverted derivation with match heights, we get

$$b_0\underline{a_0a_0a_0}a_0 \to b_0b_1\underline{b_1b_1a_0} \to \underline{b_0b_1a_1}b_1b_1 \to a_1b_1b_1b_1b_1.$$

These match heights of the inverse derivation are called *inverse match heights* of the original derivation. Systems that are inverse match-bounded have been treated in [GHW05], where it is shown that they effectively preserve context-freeness, and that the termination and the uniform termination problem are both decidable for this class. There, the relation between match-boundedness of the original system and the inverse system has been stated as an open question. Here, we give a partial answer: for length-preserving systems R, the following statements are equivalent:

1. R is match-bounded,
2. R is change-bounded,
3. R is inverse change-bounded,
4. R is inverse match-bounded.

The following implications are obvious: (2) $\Rightarrow$ (1) and (2) $\Leftrightarrow$ (3). So we have to show (1) $\Rightarrow$ (2), and the rest follows by symmetry.

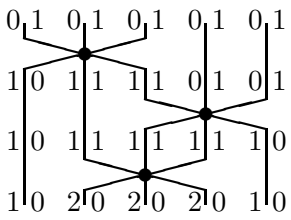
The proof uses linear algebra in the (min,plus) semi-ring to compute match heights. (A similar observation is that the Tetris computer game can be modelled by (max,plus) matrices [GP97].) For lack of space, we omit the proof here, and only illustrate the claim.

2 Match Heights and Inverse Match Heights

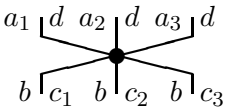
For our considerations, we can completely ignore the rules of R and the letters of Σ and consider *only* their height annotations instead. Then an annotated derivation is just a sequence of strings over $\mathbb{N}$. In our running example, rules have width 3 and we apply them on a string of length 5 at positions 0, 2, and 1 successively. We get the following sequences (redexes underlined):

Derivation:	$\underline{abbbb} \to bb\underline{abb} \to b\underline{bbba} \to baaaa,$
Change heights:	$\underline{00000} \to 11\underline{100} \to 1\underline{121}1 \to 12321,$
Match heights:	$\underline{00000} \to 11\underline{100} \to 1\underline{111}1 \to 12221,$
Inverse match heights:	$11111 \leftarrow \underline{01111} \leftarrow 01\underline{110} \leftarrow 0\underline{0000}0.$

We now visualize this rewrite sequence, together with its match heights and inverse match heights, as a (directed) graph. Each rewrite step corresponds to a node, each edge (orientet top-down) stands for an (annotated) position in a string. Edges are annotated by the pair of match height and inverse match height at the corresponding position. Note that match height annotations start at zero in the top (north) row, and inverse match height annotations start at zero in the bottom (south) row.



general pattern for one node:



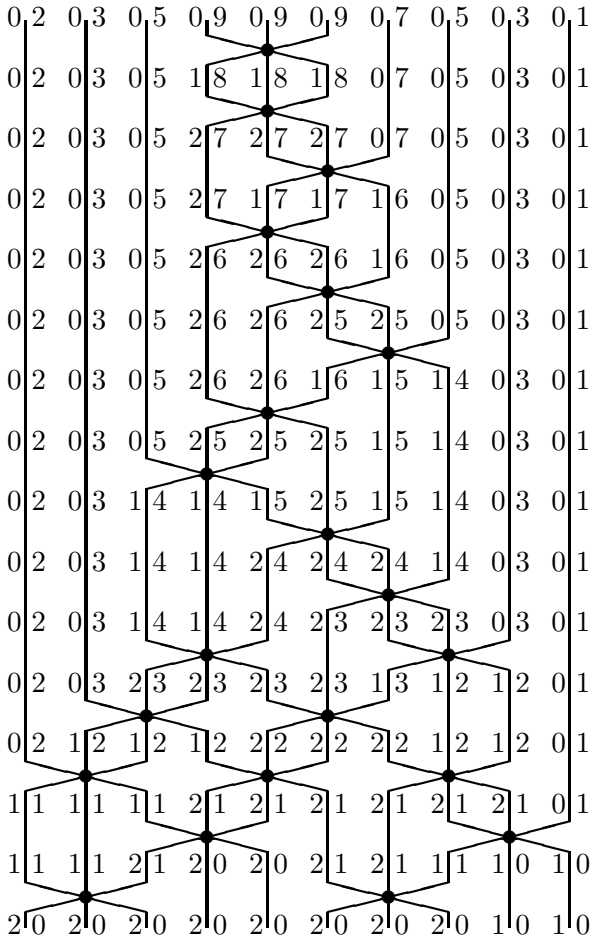
where $b = 1 + \min\{a_1, a_2, a_3\}$
and $d = 1 + \min\{c_1, c_2, c_3\}$

The question about the relation between bounds for match heights and inverse match heights can be rephrased as a problem on planar directed acyclic graphs as in the picture. Assume that all nodes have equal in- and outdegree w. If each node has distance at most M from the top, is there a bound M' on the distances from the bottom?

The answer is yes, and we can prove $M' \leq (2Mw + 1)(w + 1)^M$. We do not know how sharp this bound is. Via computer experiments we found derivations of the following inverse match heights M' (for given match height M and rule width w):

M'	$w = 1$	2	3	4	5
$M = 1$	1	2	2	2	2
2	2	6	9	12	15
3	3	12	20		
4	4	19			
5	5	26			

The following example illustrates rule width $w = 3$, match-bound $M = 2$ and inverse match-bound $M' = 9$.



References

[GHW04] Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Match-bounded string rewriting systems. *Appl. Algebra Eng. Commun. Comput.*, 15(3-4):149–171, 2004.

[GHW05] Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Termination proofs for string rewriting systems via inverse match-bounds. *J. Autom. Reasoning*, 34(4):365–385, 2005.

[GHWZ07] Alfons Geser, Dieter Hofbauer, Johannes Waldmann, and Hans Zantema. On tree automata that certify termination of left-linear term rewriting systems. *Inf. Comput.*, 205(4):512–534, 2007.

[GP97] Stephane Gaubert and Max Plus. Methods and applications of (max, +) linear algebra. In Rüdiger Reischuk and Michel Morvan, editors, *STACS*, volume 1200 of *Lecture Notes in Computer Science*, pages 261–282. Springer, 1997.

[KM07] Martin Korp and Aart Middeldorp. Proving termination of rewrite systems using bounds. In Franz Baader, editor, *RTA*, volume 4533 of *Lecture Notes in Computer Science*, pages 273–287. Springer, 2007.

[Rav04] Bala Ravikumar. Peg-solitaire, string rewriting systems and finite automata. *Theor. Comput. Sci.*, 321(2-3):383–394, 2004.

Operationelle Semantiken und Simulationskonzepte DNA-basierter Systeme

Christian Hofmann

Technische Universität Dresden
Institut für Theoretische Informatik
e-mail: hofmann@tcs.inf.tu-dresden.de

Zusammenfassung

DNA-basierte Systeme beschreiben einerseits alternative Berechnungsparadigmen, andererseits Abläufe biochemischer Prozesse durch Anwendung molekularbiologischer Operationen auf DNA-Molekülen, und werden formal als Spracherzeugungssysteme betrachtet. Die Analyse der durch ein System erzeugten Sprache ist jedoch nicht hinreichend, um temporale Aussagen über ein System treffen zu können. Deshalb werden in diesem Beitrag operationelle Semantiken für einen Vertreter des in-vitro sowie des in-vivo-Computing definiert und Simulationskonzepte der Prozesstheorie adaptiert.

1 Einführung

Sowohl die verteilten kommunizierenden Splicing-Systeme (TT-Systeme), als auch die Splicing-P-Systeme basieren auf der Berechnung mittels molekularbiologischer Operationen unter Ausnutzung der massiven Datenparallelität, einerseits in vitro, andererseits in vivo.

Mittels struktureller operationeller Semantiken (SOS) werden formale Beschreibungen dieser Berechnungssysteme definiert, welche u. a. die Analyse von (Bi-)Simulationen zwischen zwei gegebenen Systemen sowie die formale Prüfung von temporalen Eigenschaften mittels Model-Checking in einem System ermöglichen.

2 Mathematische Grundlagen

Die Menge der natürlichen Zahlen $\{0, 1, \dots\}$ wird mit $\mathbb{N}$, die Menge $\mathbb{N} \setminus \{0\}$ mit $\mathbb{N}^+$ notiert. Sei V ein Alphabet, V^* beschreibt das durch V und die Wortverkettung erzeugte freie Monoid mit dem leeren Wort ε als neutralem Element und $V^+ = V^* \setminus \{\varepsilon\}$. Eine beliebige Teilmenge von Zeichenketten aus V^* wird als (formale) Sprache bezeichnet, jedes Element dieser Menge

als Wort. Die Funktionen $\mathrm{pre}, \mathrm{suf} : V^* \to \mathcal{P}(V^*)$ berechnen die Menge aller Präfixe bzw. Suffixe und $|\_| : V^* \to \mathbb{N}$ die Länge eines Wortes $w \in V^*$.

Ein beschriftetes Transitionssystem (LTS) über einer Aktionsmenge $\mathcal{A}ct$ ist ein Tupel $(\mathcal{Q}, \mathcal{T})$ mit der Zustandsmenge $\mathcal{Q}$ der Transitionsrelation $\mathcal{T} \subseteq \mathcal{Q} \times \mathcal{A}ct \times \mathcal{Q}$, wobei (q, α, q') mit $q, q' \in \mathcal{Q}$ und $\alpha \in \mathcal{A}ct$ durch $q \xrightarrow{\alpha} q'$ notiert wird. Eine Ableitung von q_0 nach q_n ist eine endliche Sequenz $q_0 \xrightarrow{\alpha_1} q_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} q_n$ mit den Aktionen $\alpha_1, \ldots \alpha_n$.

Eine Aktion $\xrightarrow{\alpha}$ ist beobachtbar, falls $\alpha \in Act$, bzw. unbeobachtbar, falls $\xrightarrow{\tau}$. Ein Experiment e ist eine endliche Sequenz $e = \alpha_1 \ldots \alpha_n$ von beobachtbaren Aktionen $\alpha_1, \ldots, \alpha_n$. Die Relation $\Rightarrow$ ist definiert als die möglicherweise leere Sequenz von unbeobachtbaren Aktionen $p \xrightarrow{\tau} \cdots \xrightarrow{\tau} q$, formal $\Rightarrow \stackrel{\mathrm{def}}{=} \xrightarrow{\tau}{}^*$. Für $e = \alpha_1 \ldots \alpha_n$ ist die Relation $\stackrel{e}{\Rightarrow} \stackrel{\mathrm{def}}{=} \Rightarrow \xrightarrow{\alpha_1} \Rightarrow \cdots \Rightarrow \xrightarrow{\alpha_n} \Rightarrow$ definiert.

Für zwei gegebene LTS $(\mathcal{Q}, \mathcal{T})$ und $(\mathcal{Q}', \mathcal{T}')$ ist eine nichtleere binäre Relation $\mathcal{S} \subseteq \mathcal{Q} \times \mathcal{Q}'$ eine starke Simulation [schwache Simulation], gdw. für $p, p' \in \mathcal{Q}, q, q' \in \mathcal{Q}', \alpha \in \mathcal{A}ct$, ein Experiment $e \in \mathcal{A}ct^*$ und $p \mathcal{S} q$ gilt, dass aus $p \xrightarrow{\alpha} p'$ $[p \stackrel{e}{\Rightarrow} p']$ die Existenz eines $q' \in \mathcal{Q}'$ mit $q \xrightarrow{\alpha} q'$ $[q \stackrel{e}{\Rightarrow} q']$ und $p' \mathcal{S} q'$ folgt, bzw. eine starke Bisimulation [schwache Bisimulation], falls $\mathcal{S}$ und $\mathcal{S}^{-1}$ jeweils starke Simulationen [schwache Simulationen] sind. Der Zustand $q \in \mathcal{Q}'$ simuliert stark [schwach] $p \in \mathcal{Q}$, falls $\exists \mathcal{S}. p \mathcal{S} q$; p, q sind *stark bisimular* $(p \sim q)$ [*schwach bisimular* $(p \approx q)$], falls $\mathcal{S}$ eine Bisimulation beschreibt.

3 TT-Systeme

Ein TT-System Γ vom Grad $n \geq 1$ wird formal durch $(V, \Sigma, (A_1, R_1, F_1), \ldots, (A_n, R_n, F_n))$ mit den in [3] spezifizierten Komponenten definiert.

Eine *Konfiguration* $\mathcal{C}_\Gamma$ eines TT-Systems Γ ist ein Tupel $(\mathcal{L}'_1, \ldots, \mathcal{L}'_n)$ und definiert durch $(A_1, \ldots, A_n) \vdash_\Gamma^* (\mathcal{L}'_1, \ldots, \mathcal{L}'_n)$, wobei die Relation $\mathcal{C}_\Gamma \vdash_\Gamma \mathcal{C}'_\Gamma$ ein Berechnungsschritt eines TT-Systems Γ mit der massiv parallelen Anwendung der Splicing-Operation in jedem Reagenzglas und der anschließenden Verteilung der Ergebnisse unter Beachtung der jeweiligen Filterregelmengen beschreibt. Eine formale Definition ist in [3] gegeben. Wir bezeichnen weiterhin mit $\mathcal{C}_\Gamma^{(i)}$ die $i$te Konfiguration, initial $\mathcal{C}_\Gamma^{(0)} = (A_1, \ldots, A_n)$.

Beispiel 3.1 $\Gamma = (\{X, Y, Z, Z', Y', n\}, \{n\}, (\{ZnnY', ZnnY, XnY\}, \{\varepsilon \# Y \$ Z \# nnY', \varepsilon \# Y \$ Z \# nnY\}, \{(X, Y)\}), (\{XnY', Z'\}, \{\varepsilon \# Y' \$ Z' \# \varepsilon, X \# \varepsilon \$ \varepsilon \# Z'\}, \{(X, Y')\}), (\emptyset, \emptyset, \{(n, n)\}))$ *erzeugt die formale Sprache* $L(\Gamma) = \{w \,|\, w \in \{n\}^+ \wedge |\, w \,| = (2 \cdot i) + 1 \wedge i \in \mathbb{N}\}$.

3.1 SOS von TT-Systemen

Für die Definition der SOS von TT-Systemen wird die Syntax von Konfigurationen leicht modifiziert. Für ein gegebenes System Γ vom Grad n

über dem Alphabet V ist eine Konfiguration $\mathcal{C}_\Gamma$ gegeben durch eine Sequenz $\langle L, K\rangle^n$, wobei $L \in \{1, \ldots, n\}$ ein Label und $K \subseteq V^*$ ist. Eine Zwischenkonfiguration $\mathcal{C}'_\Gamma$ ist gegeben durch eine Sequenz $\langle L, K'\rangle^n$ mit einem Label L und der Menge der Kommunikationsregeln $K' \subseteq L \times V^+$ der Form (l, w), wobei l die Zielkomponente und w das zu kommunizierende Wort beschreibt.

Die SOS ist durch ein LTS mit der Aktionsmenge $\mathcal{A}ct \subseteq V^*$, der Menge der erreichbaren Konfigurationen $\mathcal{C}_\Gamma$ des Systems Γ und den folgenden Transitionsregeln mit $K'_i = \bigcup_{j=1,j\neq i}^{n}\{(j, w) \,|\, (w \in K_i \wedge w \in S_j) \vee (\exists x, y \in K_i.\exists r \in R_i.((((x, y) \vdash_r (w, z)) \vee ((x, y) \vdash_r (z, w))) \wedge w \in S_j))\} \cup \{(i, w) \,|\, (w \in K_i \wedge w \notin \bigcup_{j=1,j\neq i}^{n} S_j) \vee (\exists x, y \in K_i.\exists r \in R_i.((((x, y) \vdash_r (w, z)) \vee ((x, y) \vdash_r (z, w))) \wedge w \notin \bigcup_{j=1,j\neq i}^{n} S_j))\}$, $\mathrm{com}_i : L \times V^+ \to V^*$ mit $\mathrm{com}_i(K') = \{w \,|\, (i, w) \in K'\}$, $\pi : \mathbb{N} \times \langle L, K\rangle^n \to V^*$ mit $\pi(i, \langle L_1, K_1\rangle, \ldots, \langle L_n, K_n\rangle) = K_i$ und $S_i = \{w \in V^* \,|\, \exists (p_a, p_e) \in F_i.(p_a \in \mathrm{pre}(w) \wedge p_e \in \mathrm{suf}(w))\}$ gegeben.

$$\text{(Trans)}\quad \frac{\mathcal{C}_\Gamma \rightharpoonup \mathcal{C}'_\Gamma \qquad \mathcal{C}'_\Gamma \rightharpoondown \mathcal{C}''_\Gamma \qquad \begin{array}{r}\zeta = (\pi(n, \mathcal{C}''_\Gamma) \setminus \pi(n, \mathcal{C}_\Gamma)) \cap \Sigma^+ \\ \xi = (\pi(n, \mathcal{C}_\Gamma) \setminus \pi(n, \mathcal{C}''_\Gamma)) \cap \Sigma^+\end{array}}{\mathcal{C}_\Gamma \xrightarrow{(\zeta,\xi)} \mathcal{C}''_\Gamma}$$

$$\text{(τ)}\quad \frac{\mathcal{C}_\Gamma \xrightarrow{(\emptyset,\emptyset)} \mathcal{C}''_\Gamma}{\mathcal{C}_\Gamma \xrightarrow{\tau} \mathcal{C}''_\Gamma} \qquad \text{($\rightharpoonup_1$)}\quad \frac{\langle 1, K_1\rangle \rightharpoonup \langle 1, K'_1\rangle \qquad \ldots \qquad \langle n, K_n\rangle \rightharpoonup \langle n, K'_n\rangle}{\langle 1, K_1\rangle, \ldots, \langle n, K_n\rangle \rightharpoonup \langle 1, K'_1\rangle, \ldots, \langle n, K'_n\rangle}$$

$$\text{($\rightharpoonup_2$)}\quad \frac{}{\langle i, K_i\rangle \rightharpoonup \langle i, K'_i\rangle} \qquad \text{($\rightharpoondown$)}\quad \frac{K'_i = \bigcup_{j=1}^{n} \mathrm{com}_i(K_j)}{\langle 1, K_1\rangle, \ldots, \langle n, K_n\rangle \rightharpoondown \langle 1, K'_1\rangle, \ldots, \langle n, K'_n\rangle}$$

Jede Transition wird durch die Regel Trans in eine Splicing-Transition ($\rightharpoonup$) und eine Kommunikationstransition ($\rightharpoondown$) unterteilt, wobei der beobachtbare Teil aus den Mengen von Wörtern über Σ besteht, die zur $n$ten Komponente hinzugefügt (ζ) bzw. herausgefiltert (ξ) werden, da diese o. B. d. A. die Berechnungsergebnisse des TT-Systems beinhaltet. Eine Transition ist unbeobachtbar, falls $\mathcal{C}_\Gamma \xrightarrow{(\emptyset,\emptyset)} \mathcal{C}''_\Gamma$ gilt, was durch die Regel (τ) inferiert wird.

4 Splicing-P-Systeme

Ein extendiertes Splicing-P-System Π vom Grad $n, n \geq 1$ wird durch $\Pi = (V, \Sigma, \mu_0, M_1, \ldots, M_n, R_1, \ldots, R_n)$ mit der in [1] gegebenen Syntax und Semantik definiert. Ein gegebenes Splicing-P-System Π ist *geschlossen*, wenn $[_{\mathrm{Skin}}\mu_0]_{\mathrm{Skin}}$ und $M_{\mathrm{Skin}} = R_{\mathrm{Skin}} = \emptyset$ gilt. Die durch ein geschlossenes Splicing-P-Systeme erzeugte Sprache sind alle Wörter über Σ, die in die Skin-Membran kommuniziert werden.

Beispiel 4.1 *Das System* $\Pi = (\{X, Y, Z', Z, n\}, \{n\}, [_{Skin}[_1[_2]_2]_1]_{Skin}, \{Xn, ZnnY, Z', XnY\}, \{Z'\}, \{(n\#Y\$\varepsilon\#nnY; in_2, out), (X\#\varepsilon\$\varepsilon\#Z'; out, out)\}, \{(\varepsilon\#Y\$Z'\#\varepsilon; out, here), (n\#Y\$n\#Y; out, out)\})$ *erzeugt die formale Sprache* $L(\Pi) = \{w \mid w \in \{n\}^+ \wedge |w| = (2 \cdot i) + 1 \wedge i \in \mathbb{N}\}$.

4.1 SOS von geschlossenen Splicing-P-Systemen

Analog zur SOS von TT-Systemen und [2] wird die SOS für geschlossene Splicing-P-Systeme eingeführt. Für ein System $\Pi = (V, \Sigma, \mu, M_1, \ldots, M_n, R_1, \ldots, R_n)$ ist eine Konfiguration $\mathcal{C}_\Pi$ als Membranstruktur induktiv definiert mit $\langle L \mid w \rangle$ für eine elementare Membran und $\langle L \mid w; M_1, \ldots, M_n \rangle$ für eine Membran, welche n Membranen enthält, wobei L ein Label und $w \subseteq V^*$ die Elemente der Membran definiert. Eine endliche Folge von Membranstrukturen $M_1, \ldots, M_n$ wird mit M_* bzw. M_+, falls die die Folge mindestens ein Element besitzt, beschrieben.

Für ein Alphabet V, eine Sprache $\mathcal{L} \subseteq V^*$ und eine endliche Menge von Entwicklungs-Regeln $R \subset (V^* \cdot \{\#\} \cdot V^* \cdot \{\$\} \cdot V^* \cdot \{\#\} \cdot V^*) \times (Tar \times Tar)$ mit $Tar = \{here, in, out\}$ und $\#, \$ \notin V$ ist die Splicing-Operation ς für Splicing-P-Systeme definiert mit $\varsigma(\mathcal{L}, R) = \{(w, tar_1) \mid \exists x, y \in \mathcal{L}. \exists (r; tar_1, tar_2) \in R. (x, y) \vdash_r (w, z)\} \cup \{(z, tar_2) \mid \exists x, y \in \mathcal{L}. \exists (r; tar_1, tar_2) \in R. (x, y) \vdash_r (w, z)\}$ mit $(x, y) \vdash_r (w, z)$ gdw. $r = u_1\#u_2\$u_3\#u_4 \wedge x = x_1u_1u_2x_2 \wedge y = y_1u_3u_4y_2 \wedge w = x_1u_1u_4y_2 \wedge z = y_1u_3u_2x_2$. Weiterhin werden die Funktionen $\text{here}(w) = \{x \mid (x, here) \in w\}, \text{out}(w) = \{x \mid (x, out) \in w\}, \text{in}_\text{L}(w) = \{x \mid (x, in_L) \in w\}, \text{label}(\langle L \mid w; M_* \rangle) = L$ und $\text{obj}(\langle L \mid w; M_* \rangle) = w$ definiert.

Die SOS ist durch ein LTS mit der Aktionsmenge $\mathcal{A}ct \subseteq V^*$, der Menge der erreichbaren Konfigurationen $\mathcal{C}_\Pi$ des Systems Π und den folgenden Transitionsregeln mit $w' = \varsigma(w, R_L) \cup \{(x, here) \mid x \in w\}$ in $\rightharpoonup_1$, $w' = \text{here}(w)$ in $\rightharpoonup_1$, $w'' = \text{out}(\text{obj}(M)) \cup \text{here}(w)$ in $\rightharpoonup_2$ und für die Membranen $M_1, \ldots, M_m, m \leq n$, $w'' = \text{out}(\text{obj}(M_1)) \cup \cdots \cup \text{out}(\text{obj}(M_m)) \cup \text{here}(w)$ sowie für alle $i \in \{1, \ldots, m\}$ $\text{obj}(M_i'') = \text{obj}(M_i') \cup \text{in}_{\text{label}(M_i')}(w)$ in $\rightharpoonup_3$ gegeben.

$$\frac{}{w \rightharpoonup_L w'}\ (\rightharpoonup_1) \qquad \frac{w \rightharpoonup_L w'}{\langle L \mid w \rangle \rightharpoonup \langle L \mid w' \rangle}\ (\rightharpoonup_2) \qquad \frac{w \rightharpoonup_L w' \qquad M_+ \rightharpoonup M_+'}{\langle L \mid w; M_+ \rangle \rightharpoonup \langle L \mid w'; M_+' \rangle}\ (\rightharpoonup_3)$$

$$\frac{M \rightharpoonup M' \qquad M_+ \rightharpoonup M_+'}{M, M_+ \rightharpoonup M', M_+'}\ (\rightharpoonup_4) \qquad \frac{M_1, \ldots, M_n \rightharpoondown M_1', \ldots, M_n'}{\langle L \mid w; M_1, \ldots, M_n \rangle \rightharpoondown \langle L \mid w''; M_1'', \ldots, M_n'' \rangle}\ (\rightharpoondown_3)$$

$$\frac{L \neq \text{Skin}}{\langle L \,|\, w\rangle \rightharpoonup \langle L \,|\, w'\rangle}\ (\rightharpoonup_1) \qquad \frac{M \rightharpoonup M' \qquad (\text{out}(\text{obj}(M)) \setminus \text{here}(w)) \cap \Sigma^+ = \zeta}{\langle \text{Skin} \,|\, w; M\rangle \overset{(\zeta,\emptyset)}{\rightharpoonup} \langle \text{Skin} \,|\, w''; M'\rangle}\ (\rightharpoonup_2)$$

$$\frac{M \rightharpoonup M' \qquad M_+ \rightharpoonup M'_+}{M, M_+ \rightharpoonup M', M'_+}\ (\rightharpoonup_4) \qquad \frac{\mathcal{C}_\Pi \rightharpoonup \mathcal{C}'_\Pi \qquad \mathcal{C}'_\Pi \overset{(\zeta,\emptyset)}{\rightharpoonup} \mathcal{C}''_\Pi}{\mathcal{C}_\Pi \overset{(\zeta,\emptyset)}{\longrightarrow} \mathcal{C}''_\Pi}\ (\textsc{Trans}) \qquad \frac{\mathcal{C}_\Pi \overset{(\zeta,\emptyset)}{\longrightarrow} \mathcal{C}''_\Pi}{\mathcal{C}_\Pi \overset{\tau}{\longrightarrow} \mathcal{C}''_\Pi}\ (\tau)$$

Die Definition der strukturellen operationellen Semantik von TT- bzw. Splicing-P-Systemen ermöglicht die Definition der schwachen bzw. starken Simulation bzw. Bisimulation, also den Nachweis von Verhaltensgleichheit zwischen in-vitro und in-vivo-Systemen.

Beispiel 4.2 *Es gilt* $\Gamma \approx \Pi$*, d. h.* $\exists \mathcal{S}.\mathcal{C}_\Gamma^{(0)} \approx_\mathcal{S} \mathcal{C}_\Pi^{(0)}$*. Für* Γ *und* $i \in \mathbb{N}, i \geq 3$ *gilt* $\mathcal{C}_\Gamma^{(0)} \overset{\tau}{\longrightarrow} \mathcal{C}_\Gamma^{(1)} \overset{(\{n\},\emptyset)}{\longrightarrow} \cdots \overset{(\{n^{2\cdot i-5}\},\emptyset)}{\longrightarrow} \mathcal{C}_\Gamma^{(i-1)} \overset{(\{n^{2\cdot i-3}\},\emptyset)}{\longrightarrow} \mathcal{C}_\Gamma^{(i)} \overset{(\{n^{2\cdot i-1}\},\emptyset)}{\longrightarrow} \cdots$ *und für* $i \in \mathbb{N}, i \geq 3, i \bmod 2 = 0$ *und das System* Π $\mathcal{C}_\Pi^{(0)} \overset{(\{n\},\emptyset)}{\longrightarrow} \mathcal{C}_\Pi^{(1)} \overset{\tau}{\longrightarrow} \cdots \overset{\tau}{\longrightarrow} \mathcal{C}_\Pi^{(i)} \overset{(\{n^{2\cdot i-1}\},\emptyset)}{\longrightarrow} \mathcal{C}_\Pi^{(i+1)} \overset{\tau}{\longrightarrow} \cdots$ *und somit gilt für* $j \in \{0,1\}$ *und alle* $i \in \mathbb{N}, i \geq 2$*, dass* $\mathcal{S} = \left\{\left(\mathcal{C}_\Gamma^{(j)}, \mathcal{C}_\Pi^{(0)}\right), \left(\mathcal{C}_\Gamma^{(i)}, \mathcal{C}_\Pi^{(2\cdot i-2)}, \right), \left(\mathcal{C}_\Gamma^{(i)}, \mathcal{C}_\Pi^{(2\cdot i-3)}, \right)\right\}$*. Es existiert jedoch keine starke Bisimulation zwischen den Zuständen der beiden Systeme, d. h.* $\Gamma \not\sim \Pi$*.*

Literatur

[1] G. Păun, *Membrane Computing – An Introduction*, Springer Verlag, 2002.

[2] O. Andrei, G. Ciobanu and D. Lucanu, *A rewriting logic framework for operational semantics of membrane systems*, Theoretical Computer Science, 373(3), 2007.

[3] T. Hinze and M. Sturm, *Rechnen mit DNA - Eine Einführung in Theorie und Praxis*, Oldenbourg Wissenschaftsverlag München, 2004.

Concurrent Finite Automata

Matthias Jantzen* Manfred Kudlek* Georg Zetzsche*

Abstract

We present a generalization of finite automata using Petri nets as control. Acceptance is defined by final markings of the Petri net. The class of languages obtained by λ-free concurrent finite automata contains both the class of regular sets and the class of Petri net languages defined by final marking.

1 Introduction

In classical finite automata, the input is read by one head that moves across one symbol in every step. In order to investigate the impact of concurrency on automata accepting languages, our generalization of finite automata allows arbitrarily many heads that can move concurrently. These heads are distributed across the input and in particular, different parts of the input can be processed at the same time. A similar model, that applies the idea of multiple independent heads to Turing machines instead of finite automata is investigated in [FKR06].

The concurrency is achieved by using a Petri net that describes the movement of the heads. For every position on the input, there is a multiset of heads, which is interpreted as a marking of the Petri net. If a transition fires at some position of the input, the corresponding preset is removed from that position and the postset is added at the next position. In contrast to multi-head automata, there is no global state, since the ability to fire only depends on the heads at the respective position. Therefore, different parts of the word can be processed concurrently.

It turns out that this model is equivalent to an automaton which, for every symbol on the input, solves a system of algebraic equations and applies some homomorphism to the solution to obtain the next configuration. These two definitions are presented in section 2. Section 3 and 4 give an overview of the results obtained so far concerning the languages accepted by CFA.

*Department Informatik, Universität Hamburg, E-Mail: {jantzen,kudlek,3zetzsch}@informatik.uni-hamburg.de

2 Definitions

Definition 1. *A set M with an operation $+ : M \times M \to M$ is a* monoid*, if the operation is associative and there is a neutral element $0 \in M$ for which $0+x = x+0 = x$ for every $x \in M$. M is called* commutative*, if $x+y = y+x$ for all $x, y \in M$. For $x, y \in M$, let $x \sqsubseteq y$ iff there is a $z \in M$ with $y = x+z$.*

For every set A, we have the set $A^{\oplus}$ of mappings $\mu : A \to \mathbb{N}$. The elements of $A^{\oplus}$ are called multisets *over A. With the operation $\oplus$, defined by $(\mu \oplus \nu)(a) = \mu(a) + \nu(a)$, $A^{\oplus}$ becomes a commutative monoid with the neutral element $\mathbf{0}$, $\mathbf{0}(a) := 0$ for every $a \in A$. In the case $\mu \sqsubseteq \nu$ we can define $(\nu \ominus \mu)(a) := \nu(a) - \mu(a)$ for all $a \in A$. If A is finite, let $|\mu| := \sum_{a \in A} \mu(a)$. These definitions are carried over to $\mathbb{N}^k$ by noting that $\mathbb{N}^k \cong \{a_1, \ldots, a_k\}^{\oplus}$.*

A concurrent finite automaton is given by the following data.

Definition 2. *A* CFA *is a sextuple $C = (\Sigma, N, \sigma, \mu_0, \mathcal{F}, \#)$, where*

- *Σ is an alphabet and $\# \notin \Sigma$ is the* end marker *symbol,*
- *$N = (P, T, \partial_0, \partial_1)$ is a Petri net, where P (T) is the set of places (transitions) and $\partial_0, \partial_1 : T^{\oplus} \to P^{\oplus}$ are homomorphisms that specify the pre- and post-multisets. Furthermore, $\partial_0(t) \neq \mathbf{0}$ for every $t \in T$. In the case that $\partial_1(t) \neq \mathbf{0}$ for every $t \in T$, C is called* non-erasing*.*
- *$\sigma : T \to \Sigma \cup \{\lambda\}$ defines the corresponding symbol for every transition. A transition $t \in T$ is called* λ-transition*, if $\sigma(t) = \lambda$. C is called* λ-free*, if it does not contain λ-transitions.*
- *$\mu_0 \in P^{\oplus}$ is the* initial marking *and $\mathcal{F}$ is a finite set of* final markings*.*

Now we give two definitions of the accepted language that are equivalent, that is, the same language classes result from these definitions. The first one directly describes the firing of the transitions in the underlying net.

Definition 3. *Let $C = (\Sigma, N, \sigma, \mu_0, \mathcal{F}, \#)$ a CFA, where $N = (P, T, \partial_0, \partial_1)$. Then a* configuration *is a tuple $(\nu_0, a_1, \nu_1, \ldots, a_n, \nu_n, \#, \nu_{n+1})$, where $\nu_0, \ldots, \nu_{n+1} \in P^{\oplus}$ and $a_1, \ldots, a_n \in \Sigma$. On the set of configurations of C, we define the binary relation $\underset{C}{\longrightarrow}$. It describes the firing of one transition. Let*

$$(\nu_0, a_1, \nu_1, \ldots, a_n, \nu_n, \#, \nu_{n+1}) \underset{C}{\longrightarrow} (\nu'_0, a_1, \nu'_1, \ldots, a_n, \nu'_n, \#, \nu'_{n+1})$$

iff there are $t \in T, j \in \{1, \ldots, n+1\}$ and one of the following conditions holds:

- *$\sigma(t) = a_j$ (where $a_{n+1} = \#$), $\nu'_{j-1} = \nu_{j-1} \ominus \partial_0(t)$, $\nu'_j = \nu_j \oplus \partial_1(t)$, and $\nu'_i = \nu_i$ for every $i \neq j$.*

Operator	$\mathcal{C}_0$	$\mathcal{C}'_0$	$\mathcal{C}^\lambda_0$
$\cap R$	+	+	+
$L_1 \cdot L_2$	+	?	+
$\cup$	+	+	+
$\cap$	+	+	+
h^{-1}	+	+	?
h	-	-	-
λ-free h	?	?	+

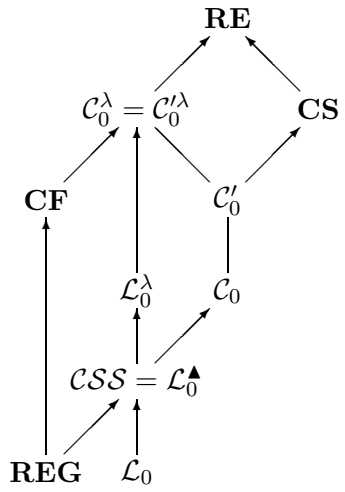


Figure 1: Closure properties and relations of the language classes.

- $\sigma(t) = \lambda$ *and* $\partial_0(t) \sqsubseteq \nu_j$, $\nu'_j = \nu_j \ominus \partial_0(t) \oplus \partial_1(t)$ *and* $\nu'_i = \nu_i$ *for every* $i \neq j$.

So the firing of a non-λ-transition moves tokens across one symbol whereas λ-transitions work like ordinary Petri net transitions on the multiset at one position. Then for $w \in \Sigma^*$, $w = a_1 \cdots a_n$, $a_1, \ldots, a_n \in \Sigma$, it is $w \in L_1(C)$ iff $(\mu_0, a_1, \mathbf{0}, \ldots, a_n, \mathbf{0}, \#, \mathbf{0}) \xrightarrow[C]{*} (\mathbf{0}, a_1, \mathbf{0}, \ldots, a_n, \mathbf{0}, \#, \mu)$ for some $\mu \in \mathcal{F}$, where $\xrightarrow[C]{*}$ denotes the reflexive transitive closure of $\xrightarrow[C]{}$. Thus, a word is accepted if all the heads are on the rightmost position and a final marking has been reached.

One possible variant is to accept a word also if the final marking is reached in a distributed manner. So for $w \in \Sigma^*$, $w = a_1 \cdots a_n$, $a_1, \ldots, a_n \in \Sigma$, let $w \in L_2(C)$ iff $(\mu_0, a_1, \mathbf{0}, \ldots, a_n, \mathbf{0}, \#, \mathbf{0}) \xrightarrow[C]{*} (\nu_0, a_1, \nu_1, \ldots, a_n, \nu_n, \#, \nu_{n+1})$, where $\nu_0 \oplus \cdots \oplus \nu_{n+1} \in \mathcal{F}$ and $\nu_{n+1} \neq \mathbf{0}$.

Now we present another definition, where the automaton solves a system of algebraic equations and obtains the next configuration by applying a homomorphism to the solution.

Definition 4. *Using the notation $T_x := \{t \in T \mid \sigma(t) = x\}$ for $x \in \Sigma \cup \{\lambda\}$, we define for every CFA C the binary relation $\Longrightarrow_C$ on $\Sigma^* \times P^\oplus$ by*

$$(w, \mu) \Longrightarrow_C (wa, \mu') \text{ if } \exists v \in T_a^\oplus : \partial_0(v) = \mu \wedge \partial_1(v) = \mu',$$

for $w \in (\Sigma \cup \{\#\})^*$, $a \in \Sigma$, $\mu, \mu' \in P^\oplus$ and

$$(w, \mu) \Longrightarrow_C (w, \mu') \text{ if } \exists t \in T_\lambda : \partial_0(t) \sqsubseteq \mu \wedge \mu' = \mu \ominus \partial_0(t) \oplus \partial_1(t),$$

for $w \in (\Sigma \cup \{\#\})^*$ *and* $\mu, \mu' \in P^{\oplus}$. *If* $\overset{*}{\underset{C}{\Longrightarrow}}$ *denotes the reflexive transitive closure of* $\underset{C}{\Longrightarrow}$, *then the accepted language of* C *is*

$$L_3(C) := \{w \in \Sigma^* \mid \exists \mu \in \mathcal{F} : (\lambda, \mu_0) \overset{*}{\underset{C}{\Longrightarrow}} (w\#, \mu)\}.$$

Now it is not hard to see that $L_3(C) = L_1(C)$ for every CFA C. Furthermore, it can be shown that the following definitions of language classes accepted by different types of CFA do not depend on whether one chooses $L_1(C)$ or $L_2(C)$ as the language of C. By $\mathcal{C}_0$ we denote the class of languages accepted by non-erasing λ-free CFA. $\mathcal{C}_0'$ is the class of languages accepted by λ-free CFA, $\mathcal{C}_0^{\lambda}$ the class of languages accepted by non-erasing CFA, and $\mathcal{C}_0'^{\lambda}$ the class of languages accepted by arbitrary CFA. **REG**, **CF**, **CS**, **RE** denotes the class of regular, context-free, context-sensitive, recursively enumerable languages, respectively. Then let $\mathcal{L}_0^{\lambda}$, $(\mathcal{L}_0)$ be the class of Petri net languages generated by (λ-free) Petri nets with final marking, as defined in [Hack76]. For alphabets Σ, Γ, a homomorphism $h : \Sigma^* \to \Gamma^*$ is called *coding*, if $h(a) \in \Gamma$ for all $a \in \Sigma$. For a language class $\mathcal{L}$, let $\hat{\mathcal{H}}(\mathcal{L})$ $(\mathcal{H}^{cod}(\mathcal{L}))$ be the class of all languages $h(L)$ with $L \in \mathcal{L}$, where h is an arbitrary homomorphism (coding).

3 Relations To Other Language Classes

By a simple construction, one obtains $\mathcal{C}_0' \subseteq \mathcal{C}_0^{\lambda}$ and $\mathcal{C}_0^{\lambda} = \mathcal{C}_0'^{\lambda}$. Furthermore, it is easy to see that (λ-free) Petri nets can be simulated by (λ-free) CFA. These inclusion is even proper, so we have $\mathcal{L}_0 \subset \mathcal{C}_0$ and $\mathcal{L}_0^{\lambda} \subset \mathcal{C}_0^{\lambda}$. While it is still open whether all context-free languages are accepted by CFA, the application of codings allows a construction similar to one used for push-down-automata: For every context-free language L, there is a coding h and a non-erasing λ-free CFA C such that $L = h(L(C))$. Thus, $\mathbf{CF} \subseteq \mathcal{H}^{cod}(\mathcal{C}_0)$. Applying arbitrary homomorphisms to languages accepted by CFA leads to the whole class of recursively enumerable languages: For every recursively enumerable language L, there is a (possibly erasing) homomorphism h and a non-erasing λ-free CFA C such that $L = h(L(C))$. Thereby, h and C are effectively constructible. In particular, it is $\mathbf{RE} = \hat{\mathcal{H}}(\mathcal{C}_0)$ and the emptiness problem is undecidable even for non-erasing λ-free CFA. In contrast, the word problem is decidable for every type of CFA and therefore we have the proper inclusion $\mathcal{C}_0 \subset \mathbf{RE}$. For every language of a λ-free CFA, there is an algorithm accepting it in linear space and quadratic time, so we have $\mathcal{C}_0' \subseteq \mathrm{NTimeSpace}(n^2, n) \subset \mathbf{CS}$.

4 Closure Properties

An easy consequence from $\mathcal{C}_0 \subseteq \mathcal{C}_0' \subseteq \mathcal{C}_0^\lambda \subset \mathbf{RE}$ and $\mathbf{RE} = \hat{\mathcal{H}}(\mathcal{C}_0)$ is the fact that $\mathcal{C}_0, \mathcal{C}_0', \mathcal{C}_0^\lambda$ are not closed under arbitrary homomorphisms. Nevertheless, at least the class $\mathcal{C}_0^\lambda$ is closed under non-erasing homomorphisms. This and $\mathbf{CF} \subseteq \mathcal{H}^{cod}(\mathcal{C}_0)$ imply $\mathbf{CF} \subseteq \mathcal{C}_0^\lambda$.

In order to prove that the CFA-languages are closed unter inverse homomorphisms, a lemma about finitely generated monoids was needed. For a commutative monoid M, a subset $S \subseteq M$ is called *quasi-invertible* iff for every $a, b \in M$, $a \in S$ and $a + b \in S$ imply $b \in S$. For example, kernels of homomorphisms are quasi-invertible. Now the lemma states that quasi-invertible submonoids of finitely generated monoids are finitely generated as well. With this lemma, one can prove that $\mathcal{C}_0$ and $\mathcal{C}_0'$ are closed unter inverse homomorphisms.

An overview of the closure properties and the known relations of the language classes is given in figure 1. Thereby, $\cap R$, $L_1 \cdot L_2$, $\cup$, $\cap$, h^{-1}, h, λ-free h stand for intersection with regular languages, concatenation, union, intersection, inverse homomorphism, arbitrary homomorphism and non-erasing homomorphism, respectively.

References

[FKR06] Berndt Farwer, Manfred Kudlek, and Heiko Rölke. *Concurrent turing machines.* Fundamenta Informaticae, 79(3-4):303-317, 2007.

[Hack76] M. Hack. *Petri Net Languages.* Cambridge, Mass.: MIT, Laboratory Computer Science, TR-159, 1976.

Schützenberger's Theorem on Formal Power Series Follows from Kleene's Theorem

Dietrich Kuske
Institut für Informatik, Universität Leipzig

Schützenberger's theorem shows that the behaviors of weighted finite automata over an arbitrary semiring are precisely the rational formal power series. Considering the Boolean semiring, one obtains as a corollary Kleene's classical result on the equivalence of the rationality and regularity of languages of finite words. Looking at the proofs of these two theorems, the similarity becomes even more evident: the proof of Schützenberger's theorem follows the same ideas as that of Kleene's theorem but adds several nontrivial details. In particular, in order to show recognizability of all rational languages or formal power series, one presents a list of constructions on (weighted) automata that demonstrate the closure of the set of recognizable languages (or formal power series) under the rational operations. Differently, we will transform a rational expression (over formal power series) into a rational language expression. Then, by Kleene's theorem, the language of this rational language expression is recognizable. The minimal deterministic automaton for this automaton can then be transformed into a weighted automaton for the original formal power series. This proof technique even works for rational expressions over formal power series extended by the Hadmard product (that corresponds to the intersection of languages). The converse implication of Schützenberger's theorem states that the behavior of every weighted finite automaton can be described by a rational expression. Its proof is based on an analysis of the paths in the automaton and dynamic programming arguments as in the proof of Kleene's theorem. Following Heiko Vogler's ideas in a more elementary presentation, we will not explicitely use the techniques of Kleene's proof, but the theorem itself to analyse the set of paths.

It is possible that this new proof technique can also be applied to other settings (like infinite words, trees, sp-pomsets, and pictures) where Kleene- and Schützenberger-type results have been shown.

Regulated Nondeterminism in Pushdown Automata*

Martin Kutrib, Larissa Werlein

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

Andreas Malcher

Institut für Informatik, Johann Wolfgang Goethe-Universität
60054 Frankfurt am Main, Germany
a.malcher@em.uni-frankfurt.de

Abstract

A generalization of pushdown automata towards regulated nondeterminism is studied. The nondeterminism is governed in such a way that the decision, whether or not a nondeterministic rule is applied, depends on the whole content of the stack. More precisely, the content of the stack is considered as a word over the stack alphabet, and the pushdown automaton is allowed to act nondeterministically, if this word belongs to some given set R of control words. Otherwise its behavior is deterministic. The computational capacity of such R-PDAs depends on the complexity of R. It turns out that non-context-free languages are accepted even if R is a linear, deterministic context-free language. On the other hand, regular control sets R do not increase the computational capacity of nondeterministic pushdown automata. This raises the natural question for the relations between the structure and complexity of regular sets R on one hand and the computational capacity of the corresponding R-PDA on the other hand. Clearly, if R is empty, the deterministic context-free languages are characterized. For $R = \{a, b\}^*$ one obtains all context-free languages. Furthermore, if R is finite, then the regular closure of the deterministic context-free languages is described. We investigate these questions, and discuss closure properties of the language classes in question under AFL operations.

*Summary of a paper presented at CIAA 2007, Prague, Czech Republic (cf. [9]).

1 Introduction

In order to explore the power of nondeterminism in bounded-resource computations, in [3] the study of nondeterminism as a measurable resource has been initiated. The well-known proper inclusion between the deterministic and nondeterministic real-time multitape Turing machine languages is refined by showing an infinite hierarchy between the deterministic real-time Turing machine languages and the languages acceptable by real-time Turing machines whose number of nondeterministic steps is logarithmically bounded. Extensive investigations are also made on limited nondeterminism in the context of finite automata [4, 5]. The quantitative study of nondeterminism in context-free languages originates from [14], and is continued in [12, 13]. The so-called branching as measure of nondeterminism, introduced for finite automata [4], is studied in [7] in connection with pushdown automata, where infinite hierarchies in between the deterministic context-free and context-free languages depending on the amount of nondeterminism or on the amount of ambiguity are shown. In [6] lower bounds for the minimum amount of nondeterminism to accept certain context-free languages are established. Pushdown automata with limited nondeterminism were investigated in [8] from the viewpoint of context-dependent nondeterminism. One important result obtained there is an automata characterization of the regular closure of deterministic context-free languages (DCFL). This is an interesting language class, since it properly extends DCFL but still has a linear-time membership problem [1]. Thus, the limitation of nondeterminism increases the generative capacity but preserves the computational complexity of the model.

Another cornerstone concept in formal language theory is that of regulated rewriting. Roughly speaking, that is, given some grammar, to impose restrictions on how to use the productions. The restrictions are usually realized by some control device. Extensive investigations of this concept in many areas of formal language theory have been done. There are too many fundamental approaches to mention them in an introduction. A valuable source for results and references is [2].

The concept of regulated rewriting has been adapted to automata in [10, 11]. Basically, the idea is to limit the computations in such a way that the sequence of transition steps has to form some words of a given control language. Even for very simple context-free control languages the power of one-turn regulated pushdown automata suffices to characterize the recursively enumerable languages.

The main goal of this paper is to investigate pushdown automata with regulated nondeterminism. Again, we want to achieve that regulation increases the generative capacity and avoids additional complexity. The use of transition rules is controlled in a weak sense. We provide two independent transition functions, where one is deterministic and the other one is nonde-

terministic. The regulation concerns the application of the nondeterministic function. Moreover, the control device is not directly defined by the words formed by sequences of transition steps. Rather the content of the stack is used. More precisely, the content of the stack is considered as a word over the stack alphabet, and the pushdown automaton is allowed to act nondeterministically, if this word belongs to some given set R of control words. Otherwise, the deterministic transition function is applied. This mechanism extends the context dependent nondeterminism of [8]. Context-dependence means that nondeterministic transition steps may appear only within certain contexts, i.e., in configurations that meet particular conditions. When these conditions concern the stack content only, there is a bridge to the approach considered here.

2 Pushdown Automata with Regulated Nondeterminism

Now we introduce the concept of regulated nondeterminism for pushdown automata. In detail, we provide some control language R such that nondeterministic steps are only allowed when the current content of the stack forms a word belonging to R. Recall that the bottom-of-stack symbol appears at the bottom of the stack only. A formal definition is as follows.

Definition 1 *Let $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ be a PDA and $R \subseteq (\Gamma \setminus Z_0)^*$ be some control language. Then $\mathcal{M}$ is called an R-*PDA *if*

(i) for all $q \in Q$, $a \in \Sigma_\lambda$, and $Z \in \Gamma$, δ can be decomposed as

$$\delta(q, a, Z) = \delta_d(q, a, Z) \cup \delta_n(q, a, Z),$$

where $\langle Q, \Sigma, \Gamma, \delta_d, q_0, Z_0, F\rangle$ is a DPDA and $\langle Q, \Sigma, \Gamma, \delta_n, q_0, Z_0, F\rangle$ is a PDA,

(ii) for all $q, q' \in Q$, $a \in \Sigma_\lambda$, $w \in \Sigma^*$, $Z \in \Gamma$, and $\gamma \in \Gamma^*$,

(a) $(q, aw, Z\gamma) \vdash (q', w, \gamma'\gamma)$, if $(q', \gamma') \in \delta_n(q, a, Z)$ and $Z\gamma = \gamma'' Z_0$ with $\gamma'' \in R$,

(b) $(q, aw, Z\gamma) \vdash (q', w, \gamma'\gamma)$, if $\delta_d(q, a, Z) = (q', \gamma')$ and $Z\gamma = \gamma'' Z_0$ with $\gamma'' \notin R$.

Examples

- $R = \emptyset$ means no nondeterminism. Thus, $\mathscr{L}(\emptyset\text{-PDA}) = \text{DCFL}$.
- If $R = (\Gamma \setminus Z_0)^*$, then $\mathscr{L}(R\text{-PDA}) = \text{CFL}$. Also, $\mathscr{L}(\{a, b\}^*\text{-PDA}) = \text{CFL}$.

- $\mathscr{L}(\{\lambda\}\text{-PDA}) = \Gamma_{\text{REG}}(\text{DCFL})$
- The one-counter languages are a proper subset of $\mathscr{L}(\{a\}^*\text{-PDA})$.

Constructions

- Let R be a regular set and $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ an R-PDA. Then an equivalent PDA $\mathcal{M}'$ can effectively be constructed.
- Let $R \neq \{\lambda\}$ be not empty. Then the families $\mathscr{L}((R \cup \{\lambda\})\text{-PDA})$ and $\mathscr{L}((R \setminus \{\lambda\})\text{-PDA})$ are equal.
- Let R be finite and not empty. Then the families $\mathscr{L}(R\text{-PDA})$ and $\mathscr{L}(\{\lambda\}\text{-PDA})$ are equal.

Hierarchy

$$\mathscr{L}(\emptyset\text{-PDA}) \subset \mathscr{L}(\{\lambda\}\text{-PDA}) \subset \mathscr{L}(\{a\}^*\text{-PDA}) \subset \mathscr{L}(\{a,b\}^*\text{-PDA})$$

Closure Properties

Language Class	$\cup$	$\bullet$	$*$	h	h^{-1}	$\cap_{reg}$	$\sim$
$\mathscr{L}$(1-counter)	+	+	+	+	+	+	−
$\mathscr{L}(\emptyset\text{-PDA})$	−	−	−	−	+	+	+
$\mathscr{L}(\{\lambda\}\text{-PDA})$	+	+	+	−	+	+	−
$\mathscr{L}(R\text{-PDA})$	+	?	?	−	+	+	−
CFL	+	+	+	+	+	+	−

Table 1: Closure properties of pushdown automata languages with regulated nondeterminism, where R is a non-empty regular set such that $\mathscr{L}(R\text{-PDA}) \neq \text{CFL}$.

References

[1] Bertsch, E., Nederhof, M.J.: Regular closure of deterministic languages. SIAM J. Comput. **29** (1999) 81–102.

[2] Dassow, J., Păun, G.: Regulated Rewriting in Formal Language Theory. Springer, Berlin (1989).

[3] Fischer, P.C., Kintala, C.M.R.: Real-time computations with restricted nondeterminism. Math. Systems Theory **12** (1979) 219–231.

[4] Goldstine, J., Kintala, C.M.R., Wotschke, D.: On measuring nondeterminism in regular languages. Inform. Comput. **86** (1990) 179–194.

[5] Goldstine, J., Leung, H., Wotschke, D.: On the relation between ambiguity and nondeterminism in finite automata. Inform. Comput. **100** (1992) 261–270.

[6] Goldstine, J., Leung, H., Wotschke, D.: Measuring nondeterminism in pushdown automata. J. Comput. System Sci. **71** (2005) 440–466.

[7] Herzog, C.: Pushdown automata with bounded nondeterminism and bounded ambiguity. Theoret. Comput. Sci. **181** (1997) 141–157.

[8] Kutrib, M., Malcher, A.: Context-dependent nondeterminism for pushdown automata. Theoret. Comput. Sci. **376** (2007) 101–111.

[9] Kutrib, M., Malcher, A., Werlein, L.: Regulated nondeterminism in pushdown automata. In: Jan Holub, J. and Žďárek, J. (eds.): Preproceedings of CIAA 2007, Prague (2006) 66–76.

[10] Meduna, A., Kolář, D.: Regulated pushdown automata. Acta Cybernet. **14** (2000) 653–664.

[11] Meduna, A., Kolář, D.: One-turn regulated pushdown automata and their reduction. Fund. Inform. **51** (2002) 399–405.

[12] Salomaa, K., Yu, S.: Limited nondeterminism for pushdown automata. Bull. EATCS **50** (1993) 186–193.

[13] Salomaa, K., Yu, S.: Measures of nondeterminism for pushdown automata. J. Comput. System Sci. **49** (1994) 362–374.

[14] Vermeir, D., Savitch, W.: On the amount of nondeterminism in pushdown automata. Fund. Inform. **4** (1981) 401–418.

Weighted Logic for Nested Words

Christian Mathissen
Institut für Informatik, Universität Leipzig
D-04009 Leipzig, Germany
mathissen@informatik.uni-leipzig.de

In order to verify recursive programs it is necessary to model them as pushdown systems rather than finite automata. But, unfortunately, language inclusion of context-free languages is undecidable. This has motivated the definition of nested word languages and visibly pushdown languages [AM06], a proper subclass of context-free languages exceeding regular languages having nice properties such as decidable language inclusion problem.

Functional validity, however, does not suffice for many applications as, for example, fault-tolerant systems show. There one tolerates some incorrect functioning provided the probability of its occurrence is sufficiently small. Or real-time systems where one has to guarantee that an event takes place at specific time.

In order to model such real-time or probabilistic systems weighted automata have been used. However, as in the unweighted case, they are too restricted to model sequential programs with recursive procedure calls. Therefore extended models such as probabilistic pushdown automata have been proposed [EKM04]. Here we define and investigate weighted nested word automata. Due to the fact that we define them over arbitrary semirings they are quite flexible in the sense that they can model timed or real-time systems as well as probabilistic and stochastic systems. We characterize their expressiveness using weighted logics as introduced by Droste and Gastin [DG05], but we aviod the standard proof by giving an interpretation in so-called sp-biposets for which a similar result has been shown [Mat07].

References

[AM06] R. Alur and P. Madhusudan. Adding nesting structure to words. In *Proc. of the 10th DLT 2006, Santa Barbara*, volume 4036 of *Lecture Notes in Computer Science*, pages 1–13, 2006.

[DG05] M. Droste and P. Gastin. Weighted automata and weighted logics. In *Proc. of the 32nd ICALP 2005, Lisbon*, volume 3580 of *Lecture Notes in Computer Science*, pages 513–525, 2005.

[EKM04] J. Esparza, A. Kučera, and R. Mayr. Model checking probabilistic pushdown automata. In *Proc. of the 19th LICS 2004, Turku*, pages 12–21. IEEE Computer Society, 2004.

[Mat07] Ch. Mathissen. Definable transductions and weighted logics for texts. In *Proc. of the 11th DLT 2007, Turku*, volume 4588 of *Lecture Notes in Computer Science*, pages 324–336, 2007.

A Generalisation of Hausdorff-Dimension in Terms of ω–Languages

Jöran Mielke
Institut für Informatik
Martin–Luther–Universität Halle–Wittenberg
email: mielke@informatik.uni-halle.de

Abstract

In this work an idea of Felix Hausdorff, presented in [Ha19], is applied to ω–languages. Instead of using just a number as fractal dimension, parameters of a family of functions will be used. By this more general way to define a dimension, non–zero–measure with respect to its generalised dimension can be achieved for certain classes of ω–languages of measure zero with respect to its HAUSDORFF–dimension.

1 Preliminaries

Let X be an alphabet of cardinality $|X| = r$. By X^* we denote the set of finite words on X, X^ω is the set of infinite sequences (ω-words). This set is considered as metric space (X^ω, ϱ) with the metric defined as

$$\varrho(\xi, \eta) := \inf\{r^{-|w|} \mid w \sqsubset \xi \wedge w \sqsubset \eta\}$$

For a language $W \subseteq X^\omega$ let $\mathbf{s}_W : \mathbb{N} \to \mathbb{N}$ be its structure function, i.e. $\mathbf{s}_W(n) = |W \cap X^n|$. In the case of $F \subseteq X^\omega$ it is $\mathbf{s}_F(n) = \mathbf{s}_{A(F)}(n)$, where $A(F)$ is the set of all prefixes of ω–words of F. The following equation defines the α–dimensional measure of $F \subseteq X^\omega$ on X^ω

$$\mathbb{L}_\alpha(F) := \lim_{n\to\infty} \inf\left\{ \sum_{w\in W} r^{-\alpha\cdot|w|} \mid F \subseteq W \cdot X^\omega \wedge l(W) \geq n \right\},$$

where $l(W) = \min\{|w| \mid w \in W\}$. It satisfies

Corollary 1 *If $\mathbb{L}_\alpha(F) < \infty$, then $\mathbb{L}_{\alpha+\varepsilon}(F) = 0$ for all $\varepsilon > 0$.*

If we, for fixed F, consider $\mathbb{L}_\alpha(F)$ as a function of α there is an α_0 such that $\mathbb{L}_\alpha(F) = \infty$ for all $\alpha < \alpha_0$ and $\mathbb{L}_\alpha(F) = 0$ for all $\alpha > \alpha_0$. At this point the function may have any value between 0 and ∞. This "changeover"–point is called HAUSDORFF–dimension. It can be defined as

$$\dim F := \sup\{\alpha \mid \alpha = 0 \vee \mathbb{L}_\alpha(F) = \infty\} = \inf\{\alpha \mid \mathbb{L}_\alpha(F) = 0\}.$$

In the particular case $\alpha = 0$ the measure $\mathbb{L}_0$ is the counting measure and in the case $\alpha = 1$ the measure $\mathbb{L}_1$ is the usual LEBESGUE–measure. The idea of HAUSDORFF–dimension has been used to study (infinite) ω–languages with LEBESGUE–measure zero. But still there are examples such that the $\dim F$–dimensional measure of F has value zero or ∞.

Example 2 *Let* $F := \{a,b\} \cdot \prod_{i=0}^{\infty}(\{a,b\}^{2^i-1} \cdot a)$, *with* $X = \{a,b\}$ *and* $\mathbf{s}_F(n) = 2^{n-\lfloor \log_2 n \rfloor}$. *Since this language is closed in* (X^ω, ϱ) *it is known from theorem 4 of [St89] that* $\dim F = \liminf_{n\to\infty} \frac{\log_2 \mathbf{s}_F(n)}{n} = 1$. *Now we can estimate* $\mathbb{L}_1(F) \leq 2^{n-\lfloor \log_2 n \rfloor} \cdot |X|^{-n} = 0$.

By a generalisation we want to find out relations between languages that have HAUSDORFF–measure zero or ∞ respectively. In particular we want to find functions in such a manner that these languages have measure strictly between zero and infinity with respect to this function.

2 Generalisation and basic Properties

We go back to HAUSDORFF's original definition as stated in [Ha19], that is, instead of using an exponent α that corresponds with the function t^α, the following class of functions is used

$$h(t) = \prod_{i\in\mathbb{N}} \left(\frac{1}{\log_r^i t^{-1}} \right)^{p_i},$$

where $\log_r^i(t) = \underbrace{\log_r \ldots \log_r}_{i \text{ times}}(t)$. For technical reasons we consider the function $\log_r^k$ in the following way

$$\log_r^k(t) = \begin{cases} \log_r^k(t) & \text{, if } \log_r^{k-1}(t) > 1 \\ 1 & \text{, else} \end{cases}$$

Definition 3 *The* $(\alpha, p_1, \ldots, p_n)$*–dimensional measure is defined as follows*

$$\mathcal{H}^{(\alpha,p_1,\ldots,p_n)}(F) = \lim_{n\to\infty} \inf \left\{ \sum_{v\in V} h(r^{-|w|}) \mid F \subseteq V \cdot X^\omega \wedge l(V) \geq n \right\},$$

with $h(t) = t^\alpha \cdot \prod_{i\in\mathbb{N}} \left(\frac{1}{\log_r^i t^{-1}} \right)^{p_i}$ *and the restriction, that the first value* $\neq 0$ *in* $(\alpha, p_1, \ldots, p_n)$ *is greater than 0. Now the generalised dimension can be defined:*

$$\dim_{GH}(F) = \sup \left\{ (\alpha, p_1, \ldots, p_n) \mid \mathcal{H}^{(\alpha,p_1,\ldots,p_n)}(F) = \infty \right\},$$

where the tuples are lexicographically ordered.

Now some basic properties for this generalised definition of dimension are developed. First the mass distribution principle (e.g. [Fa03, 4.2]) is stated for this new kind of dimension functions. It is an important instrument to obtain lower bounds for the measure of languages.

Lemma 4 *Let μ a mass distribution on $F \subseteq X^\omega$ (i.e. $0 < \mu(F) < \infty$ and $\mu(U) = 0$, if $U \cap F = \emptyset$) and h a function as above with parameters $(\alpha, p_1 \ldots, p_n)$. If there are constants $c, \delta > 0$ with*

$$\mu(U) \leq c \cdot h(\operatorname{diam}(U))$$

for all $U \subseteq X^\omega$ fulfilling $\operatorname{diam}(U) < \delta$, then $\mathcal{H}^{(\alpha,p_1\ldots,p_n)} \geq \frac{\mu(F)}{c}$.

Another property of the "usual" HAUSDORFF–dimension is shift–invariance, that is, the languages F and $w \cdot F$ (for some word $w \in X^*$) have the same dimension. This is because for every $w \in X^*$ there is a constant c such that $\mathbb{L}_\alpha(w \cdot F) = c \cdot \mathbb{L}_\alpha(F)$. In the case of the generalised measure one can only estimate:

Lemma 5 *Let $w \in X^*$ and h a function as mentioned above with parameters $(\alpha, p_1 \ldots, p_n)$. Then there exist constants $c^{(1)}_{|w|}, c^{(2)}_{|w|} > 0$ only depending on $|w|$ such that for all $F \subseteq X^\omega$*

$$c^{(1)}_{|w|} \cdot \mathcal{H}^{(\alpha,p_1\ldots,p_n)}(F) \leq \mathcal{H}^{(\alpha,p_1\ldots,p_n)}(w \cdot F) \leq c^{(2)}_{|w|} \cdot \mathcal{H}^{(\alpha,p_1\ldots,p_n)}(F)$$

This provides the shift–invariance of this kind of dimension, too.

Lemma 6 *Let $w \in X^*$ and $F \subseteq X^\omega$. Then $\dim_{GH}(F) = \dim_{GH}(w \cdot F)$.*

3 Some Results on ω–Languages

We start with a case where we cannot achieve an improvement, that is, it is not possible to find a function h such that the measure with respect to h is strictly between zero and infinity.

Lemma 7 *Let $F \subseteq X^\omega$ with $\mathbb{L}_{\dim F}(F) = \infty$. If $F = \bigcup_{i\in\mathbb{N}} F_i$ such that $\mathbb{L}_{\dim F}(F_i) < \infty$ then there is no function as specified in the previous section with $0 < \mathcal{H}^{(\alpha,p_1,\ldots,p_n)}(F) < \infty$, where $(\alpha, p_1, \ldots, p_n)$ are the parameters of h.*

Regular and context-free ω-languages can be written as a union of ω-power languages W^ω (see [RS97]) and from [St93] it is known, that $\mathbb{L}_{\dim W^\omega}(W^\omega) < \infty$. And as $\dim F = \sup_i \dim F_i$ we derive

Theorem 8 *Let $F \subseteq X^\omega$ regular or context–free language with $\mathbb{L}_{\dim F}(F) = \infty$. There is no function h, such that $0 < \mathcal{H}^{(\alpha,p_1,\ldots,p_n)}(F) < \infty$, where $(\alpha, p_1, \ldots, p_n)$, $n > 0$, are the parameters of h.*

An obvious general upper bound for the dimension can be achieved by the structure function.

Lemma 9 *Let $h(r^{-n}) \in O(\mathbf{s}_F^{-1}(n))$ a function with parameters $(\alpha, p_1, \ldots, p_n)$. Then $\mathcal{H}^{(\alpha,p_1,\ldots,p_n)} < \infty$, and, consequently, $\dim_{GH} F \leq (\alpha, p_1, \ldots, p_n)$.*

Let us now take a look at example 2 again.

Example 2 (continued) *We have $F := \{a,b\} \cdot \prod_{i=0}^{\infty}(\{a,b\}^{2^i-1} \cdot a)$, with $X = \{a,b\}$ and $\mathbf{s}_F(n) = 2^{n-\lfloor \log_2 n \rfloor} = 2^n \cdot \frac{1}{n} = 2^n \cdot \left(\frac{1}{\log_2 2^n}\right)$. And we can estimate $\mathcal{H}^{(1,-1)}(F) \leq \lim_{n\to\infty} \mathbf{s}_F(n) \cdot (2^{-n}) 1 \cdot \left(\frac{1}{\log_2 2^n}\right)^{-1} = \lim_{n\to\infty} \mathbf{s}_F(n) \cdot 2^{-n} \cdot n = 1$. The converse inequality is also true, and thus we have $\dim_{GH} F = (1,-1)$ and $\mathcal{H}^{(1,-1)}(F) = 1$.*

Next we give a generalisation of Lemma 3.10 of [St93]. To this end we introduce the δ-limit of a language $V \subseteq X^*$

$$V^\delta = \{\xi \mid \xi \in X^\omega \wedge |A(\xi) \cap V| = \infty\}$$

Theorem 10 *Let $F \subseteq X^\omega$ and h a function with parameters $(\alpha, p_1, \ldots, p_n)$. Then $\mathcal{H}^{(\alpha,p_1,\ldots,p_n)}(F) = 0$ if and only if there is a $V \subseteq X^*$ with $F \subseteq V^\delta$ and and $\sum_{v\in V} h(r^{-|v|}) < \infty$.*

It is well known that the lower box counting dimension $\dim_B F := \liminf_{n\to\infty} \frac{\log \mathbf{s}_F(n)}{n}$ is an upper bound for $\dim F$. Here we derive a similar result on upper bounds, where another limit is used instead of the box counting dimension.

Lemma 11 *Let $F \subseteq X^\omega$, $\beta(n) = \log_r \prod_{i\in\mathbb{N}} \left(\log_r^i r^{-n}\right)^{-p_i}$.*
If $\liminf_{n\to\infty} \frac{\log_r \mathbf{s}_F(n) - \alpha\cdot n}{\beta(n)} \leq c < \infty$, then $\dim_{GH} F \leq (\alpha, c\cdot p_1, c\cdot p_2 \ldots)$.

4 Concluding Remarks

The examples found so far are structured very uniformly, i.e. the number of sequences with a certain prefix depend only on the length of the prefix. Those languages are called balanced (see [St93]). This leads to the conjecture that for every balanced language with HAUSDORFF–measure zero one can find a suitable function of the considered class that leads to a measure strictly between zero and infinity for this language.

On the other hand it seems to be difficult to treat languages having HAUSDORFF–measure ∞. An indicator for this is theorem 4.10 of [Fa03], which states that a set of infinite measure has compact subsets of finite non-null measure.

References

[Fa03] Falconer, Kenneth, Fractal Geometry, *John Wiley & Sons*, 2003

[Ha19] Hausdorff, Felix, Dimension und äußeres Maß, *Math. Annalen*, **79**, 157–179

[RS97] Rozenberg, G., Salomaa, A. (Eds.), Handbook of Formal Languages, Vol. 3, Beyond Words, *Springer*, 1997

[St89] Staiger, L.,Combinatorial properties of the Hausdorff dimension, *J. Statist. Plann. Inference*, 1989, **23**, 95-100

[St93] Staiger, L.,Kolmogorov Complexity and Hausdorff dimension, *Information and Computation*, 1993, **103**, 159-194

Eingeschränkte Restart-Baumautomaten

Friedrich Otto und **Heiko Stamer**
Fachbereich Elektrotechnik/Informatik, Universität Kassel
34109 Kassel, Germany
{otto,stamer}@theory.informatik.uni-kassel.de

Zusammenfassung

Wir betrachten zwei Einschränkungen für Restart-Baumautomaten. Einerseits limitieren wir mit der sogenannten *single-path* Variante die parallele Verzweigungsmöglichkeit, so dass sowohl reguläre Kontrolle als auch Ersetzungen nur entlang eines Pfades möglich sind. Ferner betrachten wir *ground-rewrite* Restart-Baumautomaten, die nur Grundterme ersetzen dürfen.

Kombiniert man beide Einschränkungen, dann erhält man ein Modell der Restart-Baumautomaten, das zumindest alle regulären Baumsprachen umfasst. Allerdings ist noch unklar, ob diese speziellen Automaten die regulären Baumsprachen genau charakterisieren.

1 Introduction

The restarting automaton, which was introduced to model the so-called *analysis by reduction* used in linguistics, has been extended in [SO07a] from strings to trees. Actually, several different variants of restarting tree automata have been defined that correspond to certain basic types of restarting automata (on strings). In [SO07a] some fundamental results on the expressive power of these types of restarting tree automata are derived, and some closure properties are given for the families of tree languages recognized by them. In [SO07b] this work is continued by proving that all linear context-free tree languages are recognized by restarting tree automata.

Here we study two restrictions of this automaton: the *single-path restarting tree automaton* and the *ground-rewrite restarting tree automaton*.

2 Restarting Tree Automata

Formally, a *top-down restarting tree automaton* (RRWWT) is described by a six-tuple $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$, where $\mathcal{F}$ is a ranked input alphabet, $\mathcal{G} \supseteq \mathcal{F}$ is a ranked working alphabet, $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ is a finite set of

states such that $\mathcal{Q}_1 \cap \mathcal{Q}_2 = \emptyset$, $q_0 \in \mathcal{Q}_1$ is the initial state and simultaneously the restart state, $k \geq 1$ is the height of the read/write-window, and $\Delta = \Delta_1 \cup \Delta_2$ is a finite term rewriting system on $\mathcal{G} \cup \mathcal{Q}$. The rule set Δ_1 only contains k-height bounded top-down transitions of the form $q(t) \to t[q_1(x_1), \ldots, q_m(x_m)]$, where $m \geq 1$, $t \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$, $x_1, \ldots, x_m \in \mathcal{X}_m$, and $q, q_1, \ldots, q_m \in \mathcal{Q}_1$, and k-height bounded final transitions of the form $q(t) \to t$, where $t \in \mathcal{T}(\mathcal{G})$ and $q \in \mathcal{Q}_1$. The rule set Δ_2 only contains size-reducing top-down *rewrite transitions*, that is, linear rewrite rules of the form $q(t) \to t'[q_1(x_1), \ldots, q_m(x_m)]$, where $m \geq 1$, $t \in \mathcal{T}(\mathcal{G}, \mathcal{X}_m)$, $t' \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$, $x_1, \ldots, x_m \in \mathcal{X}_m$, $q \in \mathcal{Q}_1$, and $q_1, \ldots, q_m \in \mathcal{Q}_2$, and size-reducing *final rewrite transitions* of the form $q(t) \to t'$, where $q \in \mathcal{Q}_1$ and $t, t' \in \mathcal{T}(\mathcal{G})$. For both these types of transitions it is required that $||t|| > ||t'||$ and $\mathrm{Hgt}(t) \leq k$. Furthermore, Δ_2 contains k-height bounded top-down transitions of the form $q(t) \to t[q_1(x_1), \ldots, q_m(x_m)]$, where $m \geq 1$, $t \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$, $x_1, \ldots, x_m \in \mathcal{X}_m$, and $q, q_1, \ldots, q_m \in \mathcal{Q}_2$, and k-height bounded final transitions of the form $q(t) \to t$, where $t \in \mathcal{T}(\mathcal{G})$ and $q \in \mathcal{Q}_2$.

The *partial move relation* $\to_\Delta$ and its reflexive transitive closure $\to_\Delta^*$ are induced by the TRS Δ, while the *final move relation* $\to_{\Delta_1}$ and its reflexive transitive closure $\to_{\Delta_1}^*$ are induced by Δ_1. We use the notation $u \hookrightarrow_{\mathcal{A}} v$ $(u, v \in \mathcal{T}(\mathcal{G}))$ to express the fact, that $q_0(u)(\to_\Delta^* \setminus \to_{\Delta_1}^+)v$. The relation $\hookrightarrow_{\mathcal{A}}^*$ is the reflexive transitive closure of $\hookrightarrow_{\mathcal{A}}$. The *tree language* recognized by the RRWWT-automaton $\mathcal{A}$ is

$$L(\mathcal{A}) = \left\{ t_0 \in \mathcal{T}(\mathcal{F}) \mid \exists t' \in \mathcal{T}(\mathcal{G}) \text{ such that } t_0 \hookrightarrow_{\mathcal{A}}^* t' \text{ and } q_0(t') \to_{\Delta_1}^* t' \right\}.$$

Also some restricted variants of restarting tree automata have been introduced. A restarting tree automaton is called an RWWT-*automaton*, if all its top-down rewrite transitions are of the special form $q(t) \to t'[x_1, \ldots, x_m]$, where $m \geq 1$, $q \in \mathcal{Q}_1$, $t \in \mathcal{T}(\mathcal{G}, \mathcal{X}_m)$, and $t' \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$ such that $||t'|| < ||t||$ and $\mathrm{Hgt}(t) \leq k$. A restarting tree automaton is an RRWT-*automaton*, if its working alphabet $\mathcal{G}$ coincides with its input alphabet $\mathcal{F}$, that is, no auxiliary symbols are available. It is an RRT-*automaton*, if it is an RRWT-automaton for which the right-hand side of every rewrite transition is a scattered subterm of the corresponding left-hand side. Analogously, we obtain the RWT- and the RT-*automaton* from the RWWT-automaton.

Example 1 *The language* $L_1 = \{ f(g^n(a), g^n(a)) \mid n \geq 0 \} \in \mathscr{L}(\mathsf{CFTG}) \setminus \mathscr{L}(\mathsf{RTG})$ *is recognized by the* RT-*automaton* $\mathcal{A}_1 = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$, *where* $\mathcal{F} = \{f(\cdot,\cdot), g(\cdot), a\}$, $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ *with* $\mathcal{Q}_1 = \{q_0\}$, $\mathcal{Q}_2 = \emptyset$, $k = 2$, *and* Δ *is given by the rewrite rules* $q_0(f(g(x_1), g(x_2))) \to f(x_1, x_2)$ *and* $q_0(f(a, a)) \to f(a, a)$.

Finally, we introduce the *single-path top-down tree automaton* (spNF↓T), which will serve as the basis for the single-path restarting tree automaton in the next section. An spNF↓T-automaton is given through a five-tuple

$\mathcal{A} = (\mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$, where $\mathcal{F}$ is a ranked alphabet, $\mathcal{Q}$ is a finite set of states, $q_0 \in \mathcal{Q}$ is the initial state, $k \geq 1$ is the height of the look-ahead window, and Δ is a finite term rewriting system that contains k-height bounded top-down transitions of the form $q(t) \to t[x_1, \ldots, x_{i-1}, q'(x_i), x_{i+1} \ldots, x_m]$, where $m \geq 1$, $t \in \mathrm{Ctx}(\mathcal{F}, \mathcal{X}_m)$, $x_1, \ldots, x_m \in \mathcal{X}_m$, $q, q' \in \mathcal{Q}$, and $i \in \{1, \ldots, m\}$, and k-height bounded final transitions of the form $q(t) \to t$, where $t \in \mathcal{T}(\mathcal{F})$ and $q \in \mathcal{Q}$. The *tree language recognized* by $\mathcal{A}$ is $L(\mathcal{A}) = \{\, t \in \mathcal{T}(\mathcal{F}) \mid q_0(t) \to_\Delta^* t \,\}$, that is, it consists of those ground terms $t \in \mathcal{T}(\mathcal{F})$ for which $\mathcal{A}$ has an accepting run starting from the configuration $q_0(t)$.

3 Restrictions for Restarting Tree Automata

Definition 1 *A* single-path restarting tree automaton (spRRWWT) *is formally described by a six-tuple* $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$*, where* $\mathcal{F}$ *is a ranked input alphabet,* $\mathcal{G} \supseteq \mathcal{F}$ *is a ranked working alphabet,* $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ *is a finite set of states such that* $\mathcal{Q}_1 \cap \mathcal{Q}_2 = \emptyset$*,* $q_0 \in \mathcal{Q}_1$ *is the initial state and simultaneously the restart state,* $k \geq 1$ *is the height of the read/write-window, and* $\Delta = \Delta_1 \cup \Delta_2$ *is a finite term rewriting system on* $\mathcal{G} \cup \mathcal{Q}$*, where* $(\mathcal{G}, \mathcal{Q}_1, q_0, k, \Delta_1)$ *is an* spNF↓T*-automaton on* $\mathcal{G}$*, and the rule set* Δ_2 *only contains the following types of transitions:*

1. *Size-reducing top-down* rewrite transitions*, that is, linear rewrite rules of the form* $q(t) \to t'[x_1, \ldots, x_{i-1}, q'(x_i), x_{i+1}, \ldots, x_m]$*, where* $m \geq 1$*,* $t \in \mathcal{T}(\mathcal{G}, \mathcal{X}_m)$*,* $t' \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$*,* $i \in \{1, \ldots, m\}$*,* $q \in \mathcal{Q}_1$*, and* $q' \in \mathcal{Q}_2$*, and size-reducing* final rewrite transitions *of the form* $q(t) \to t'$*, where* $q \in \mathcal{Q}_1$ *and* $t, t' \in \mathcal{T}(\mathcal{G})$*. For both these types of transitions it is required that* $||t|| > ||t'||$ *and* $\mathrm{Hgt}(t) \leq k$*.*

2. *k-height bounded top-down transitions of the form* $q(t) \to t[x_1, \ldots, x_{i-1}, q'(x_i), x_{i+1}, \ldots, x_m]$*, where* $m \geq 1$*,* $t \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$*,* $i \in \{1, \ldots, m\}$*, and* $q, q' \in \mathcal{Q}_2$*, and k-height bounded final transitions of the form* $q(t) \to t$*, where* $t \in \mathcal{T}(\mathcal{G})$ *and* $q \in \mathcal{Q}_2$*.*

Essentially, an spRRWWT-automaton $\mathcal{A}$ works just like an RRWWT-automaton, the only difference is the fact that in the current tree $\mathcal{A}$ walks down a single path only. The *tree language* accepted by $\mathcal{A}$ is defined as

$$L(\mathcal{A}) = \left\{\, t_0 \in \mathcal{T}(\mathcal{F}) \mid \exists t' \in \mathcal{T}(\mathcal{G}) \text{ such that } t_0 \hookrightarrow_{\mathcal{A}}^* t' \text{ and } q_0(t') \to_{\Delta_1}^* t' \,\right\},$$

that is, it consists of those trees $t_0 \in \mathcal{T}(\mathcal{F})$ that can be reduced by the relation $\hookrightarrow_{\mathcal{A}}^*$ to a tree recognized by the spNF↓T-automaton $(\mathcal{G}, \mathcal{Q}_1, q_0, k, \Delta_1)$.

For the single-path restriction we have obtained the following results:

- $\mathscr{L}(\mathsf{RTG}) \subsetneq \mathscr{L}(\mathsf{spRT})$, $\mathscr{L}(\mathsf{lin\text{-}CFTG}) \subsetneq \mathscr{L}(\mathsf{spRWWT})$, and

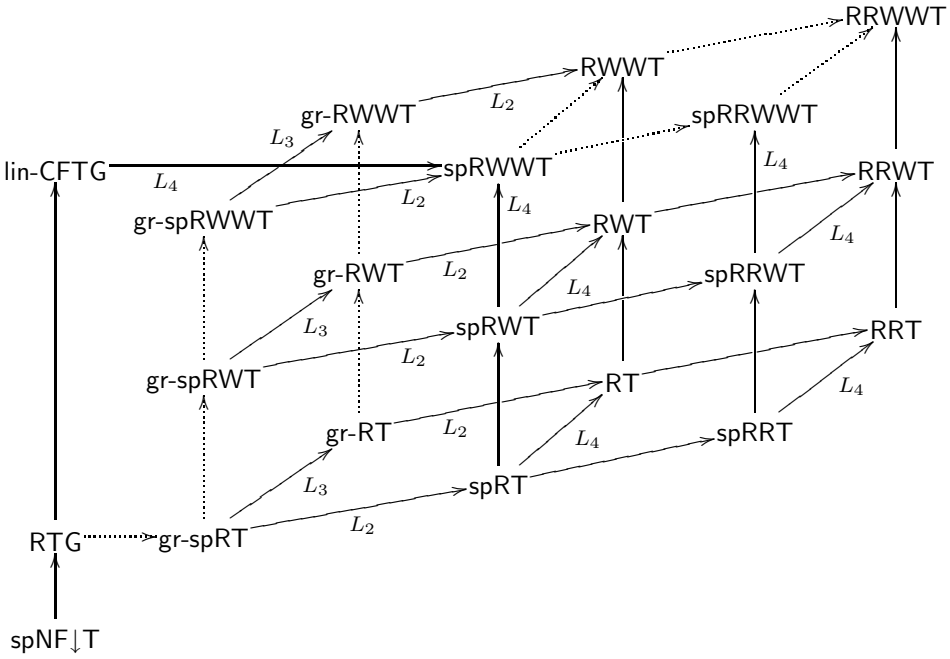


Figure 1: Inclusions between language classes defined by restricted restarting tree automata and classes generated by various tree grammars resp. tree automata. An arrow denotes a proper inclusion, while a dotted arrow denotes an inclusion that is not known to be proper.

- $\mathscr{L}(\mathsf{spRWWT})$ contains tree languages that are not context-free, e.g., the tree language $L_4 := \{\, f(g^n(h^n(a)), g^n(h^n(a))) \mid n \geq 1 \,\}$.

Definition 2 *A* ground-rewrite restarting tree automaton (gr-RWWT) *is an* RWWT*-automaton* $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ *for which* Δ_2 *only contains final rewrite transitions.*

Obviously, for ground-rewrite restarting tree automata we only need to consider the RWWT-, the RWT-, and the RT-variant. Even gr-RT-automata can still recognize all regular tree languages, as the subsystem of a gr-RT-automaton that is described by Δ_1 is (essentially) an NF↓T-automaton.

It is easily seen that $L_2 := \{\, g^n(h(g^n(a))) \mid n \geq 0 \,\} \in \mathscr{L}(\mathsf{spRT}) \setminus \mathscr{L}(\mathsf{gr\text{-}RT})$. In fact, L_2 is not even recognized by any gr-RWWT-automaton. On the other hand, $L_3 := \{\, f(g^n(a), g^n(a)), f(g^n(a), g^{2n}(b)) \mid n \geq 0 \,\} \in \mathscr{L}(\mathsf{gr\text{-}RT}) \setminus \mathscr{L}(\mathsf{spRWT})$. In particular, it follows that $\mathscr{L}(\mathsf{gr\text{-}RT})$ and $\mathscr{L}(\mathsf{spRT})$ are incomparable under set inclusion.

Definition 3 *A* ground-rewrite spRWWT-automaton (gr-spRWWT) *is an* spRWWT*-automaton* $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ *for which* Δ_2 *only contains final rewrite transitions.*

For the above combination of both restrictions we also get the inclusion $\mathscr{L}(\mathsf{RTG}) \subseteq \mathscr{L}(\mathsf{gr\text{-}spRT})$, but it remains open whether gr-spRT-automata recognize any non-regular tree language.

References

[CDG+] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. September 2005.

[GS97] F. Gécseg and M. Steinby. Tree Languages. In G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages*, vol. 3, pp. 1–68. Springer, 1997.

[SO07a] H. Stamer and F. Otto. Restarting Tree Automata. In J. van Leeuwen, G.F. Italiano, W. van der Hoek, C. Meinel, H. Sack, and F. Plášil (eds.), *SOFSEM 2007, Proc.*, *Lect. Notes Comput. Sci.* **4362**, pp. 510–521. Springer, 2007.

[SO07b] H. Stamer and F. Otto. Restarting Tree Automata and Linear Context-Free Tree Languages. `http://web.auth.gr/cai07/Lectures/StamerCAI07.pdf`. Talk presented at *CAI 2007*, Thessaloniki, May 2007.

Simulationen durch zeitbeschränkte Zählerautomaten

Holger Petersen

Univ. Stuttgart, FMI
Universitätsstraße 38
D-70569 Stuttgart
petersen@informatik.uni-stuttgart.de

Ein klassisches Resultat von Minsky [5] besagt, dass Automaten mit zwei Zählern universell sind. Vom Standpunkt der Berechenbarkeit aus ändert sich daher nichts an der Mächtigkeit solcher Automaten, wenn ihnen weitere Zähler zur Verfügung stehen. Allerdings kann sich die Effizienz, mit der andere Speichertypen durch Zähler simuliert werden, deutlich verändern. Die bekannte Simulation eines Bandes durch zwei Zähler erfordert einen doppelt exponentiellen Zeitaufwand [6, Theorem 2.1 (3)], während zwei Keller in einfach exponentieller Zeit durch drei Zähler simuliert werden können [3, Theorem 8.14], was eine entsprechend effiziente Simulation eines Bandes erlaubt. Für die Simulation mehrerer Zähler durch drei gibt Greibach eine explizite polynomielle Schranke an [2], vermutet aber, dass kein entsprechendes Resultat für einen Simulator mit zwei Zählern möglich ist.

Es ist daher eine natürliche Frage, wie die Anzahl an Zählern die Zeiteffizienz von Simulationen beeinflusst. Hierbei konzentrieren wir uns auf deterministische Automaten mit Einweg-Eingabe und die Simulation von Zählern und Kellern, weil komplexere Speicher wie Bänder oder Queues leicht durch Keller simuliert werden können.

Sind obere Schranken für Simulationen bekannt, dann liegt es nahe, die Optimalität der Verfahren zu untersuchen. In den ersten Veröffentlichungen über zeitbeschränkte Zählerautomaten [4, 1] werden Sprachen angegeben, die von Automaten mit $k \geq 2$ Zählern in Realzeit akzeptiert werden, von Automaten mit $k-1$ Zählern aber nicht in der gleichen Schranke erkannt werden können. Eine allgemeine Simulation von k durch $k-1$ Zähler ist daher in Realzeit nicht möglich. Für Palindromerkennung wird in [1] eine exponentielle untere Schranke bewiesen, welche aber deutlich schlechter als die entsprechende obere Schranke ist und keine exakte Trennung erlaubt. Wesentlich komplexer als die Trennung von Automaten, die in Realzeit arbeiten, ist die Trennung in polynomieller Zeit. Greibach zeigt in

[2, Theorem 3.3], dass $s + 1$ zusätzliche Zähler die Mächtigkeit von Automaten erhöhen, welche in Zeit n^s arbeiten. Das intuitive Hindernis für eine schärfere Trennung ist die notwendige Kodierung der Information, die auf den Zählern gespeichert wird.

Wir geben für jede Anzahl $k \geq 3$ eine optimale Simulation eines Kellers mit Hilfe von k Zählern an. Außerdem verbessern wir die Zeitschranke der Simulation von polynomiell zeitbeschränkten Zählerautomaten durch Automaten mit einer kleineren Anzahl an Zählern aus [2] und zeigen ein Hierarchieresultat für diese Art von Automaten, das nicht von der Zeitschranke abhängt.

References

[1] P. C. Fischer, A. R. Meyer, and A. L. Rosenberg. Counter machines and counter languages. *Mathematical Systems Theory*, 2:265–283, 1968.

[2] S. A. Greibach. Remarks on the complexity of nondeterministic counter languages. *Theoretical Computer Science*, 1:269–288, 1976.

[3] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Pearson Education, 3rd edition, 2007.

[4] R. Laing. Realization and complexity of commutative events. Technical Report 03105-48-T, University of Michigan, 1967.

[5] M. L. Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *Annals of Mathematics*, 74:437–455, 1961.

[6] P. van Emde Boas. Machine models and simulations. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 1–66. Elsevier, Amsterdam, 1990.

A Kleene-Schützenberger Theorem for Weighted Timed Automata

Karin Quaas, Manfred Droste
Institut für Informatik
Universität Leipzig
04109 Germany
{quaas,droste}@informatik.uni-leipzig.de

During the last years, weighted timed automata (wta) have received much interest in the real-time community. Weighted timed automata are an extension of timed automata [2] and allow to assign weights (costs) to both locations and edges. This model has been introduced independently by Alur et al. [3] and Behrmann et al. [4]. It allows the modelling of continuous consumption of resources, and thus, enables to represent e.g. scheduling and planning problems. Consequently, there has been much research on problems as optimal reachability and model checking [7], [9], [1].However, there has been no algebraic characterization of the behaviour of wta so far. We attempt to fill this gap by providing a Kleene-Schützenberger theorem for wta. We apply the theory of weighted finite automata [6], [13], [12], and define wta over a semiring, resulting in a model that subsumes previous definitions in the literature, e.g. [3], where costs for reaching a location are computed by taking the infimum of the running weights of all runs, or [8], a multi-priced variant of a wta. For giving a Kleene-Schützenberger theorem, we combine the approach of Schützenberger [14] as well as a recent approach of a Kleene-like theorem for (unweighted) timed automata by Bouyer and Petit [10]. Our main result also implies Kleene-like theorems for several subclasses of wta, i.e., weighted finite automata, timed automata, timed automata with stopwatch observers [11].

References

[1] R. Alur, M. Bernadsky, and P. Madhusudan. Optimal reachability in weighted timed games. In J. Díaz, J. Karhumäki, A. Lepistö, and D. Sannella, editors, *ICALP*, volume 3142 of *LNCS*, pages 122–133. Springer, 2004.

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

[3] R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In Benedetto and Sangiovanni-Vincentelli [5], pages 49–62.

[4] G. Behrmann, A. Fehnker, T. Hune, K. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager. Minimum-cost reachability for priced timed automata. In Benedetto and Sangiovanni-Vincentelli [5], pages 147–161.

[5] M. D. Di Benedetto and A. Sangiovanni-Vincentelli, editors. *Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001, Rome, Italy, March 2001, Proceedings*, volume 2034 of *LNCS*. Springer, 2001.

[6] J. Berstel and C. Reutenauer. *Rational Series and their Languages*. Springer-Verlag New York, Inc., New York, NY, USA, 1988.

[7] P. Bouyer, T. Brihaye, V. Bruyère, and J.-F. Raskin. On the optimal reachability problem on weighted timed automata. *Formal Methods in System Design*, 2007. To appear.

[8] P. Bouyer, E. Brinksma, and K. G. Larsen. Optimal infinite scheduling for multi-priced timed automata. *Formal Methods in System Design*, 2007. To appear.

[9] P. Bouyer, K. G. Larsen, and N. Markey. Model-checking one-clock priced timed automata. In Helmut Seidl, editor, *FoSSaCS*, volume 4423 of *LNCS*, pages 108–122. Springer, 2007.

[10] P. Bouyer and A. Petit. A Kleene/Büchi-like theorem for clock languages. *J. Autom. Lang. Comb.*, 7(2):167–186, 2001.

[11] T. Brihaye, V. Bruyère, and J.-F. Raskin. On model-checking timed automata with stopwatch observers. *Inf. Comput.*, 204(3):408–433, 2006.

[12] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*, volume 5 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1986.

[13] A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer New York, 1978.

[14] M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.

An Automata Theoretic Approach to Rational Tree Relations

Frank G. Radmacher
Lehrstuhl für Informatik 7, RWTH Aachen, Germany
radmacher@automata.rwth-aachen.de

Abstract

We investigate rational relations over trees. Our starting point is the definition of rational tree relations via rational expressions by Raoult [5]. We develop a new class of automata, called asynchronous tree automata, which recognize exactly these relations. The automata theoretic approach is convenient for the solution of algorithmic problems (like the emptiness problem). Our second contribution is a new subclass of the rational tree relations, called separate-rational tree relations, defined via a natural restriction on asynchronous tree automata. These relations are closed under composition, preserve regular tree languages, and generate precisely the regular sets in the unary case (all these properties fail for the general model), and they are still more powerful than, for instance, the automatic tree relations.

1 Background

Automata definable relations over words are widely investigated. Recognizable, automatic, deterministic rational, and (non-deterministic) rational relations result in a well-known hierarchy [2]. Proper generalizations of these theories to trees have been established over the past years in the case of recognizable relations and automatic relations. However, it is still debatable how to obtain a reasonable generalization of rational word relations to trees.

Rational relations over words can be introduced in several equivalent ways: First, they are definable via rational expressions (a generalization of regular expressions), which means that rational relations are generated from the finite relations by closure under union, componentwise concatenation, and Kleene star. On the other hand rational relations are recognized by a generalized model of finite automata, so-called *asynchronous automata* (sometimes also called *multi-tape automata*) [1].

Generalizing rational relations to trees (resp. terms) is not straightforward. Attractive results on rational word relations which one would also like for rational tree relations are the following:

- Applied to unary trees, the rational word relations should be generated (also in the case of n-ary relations).
- A characterization via rational expressions should exist.
- A natural automata theoretic characterization should exist.
- Restricted to unary relations the class of regular tree languages should be generated.
- Binary rational tree relations should be closed under composition.
- Binary rational tree relations (transductions) should preserve regular tree languages.

Towards a generalization of rational relations to trees, Raoult suggests in [5] defining relations over trees by tree grammars in which non-terminals are represented by tuples of letters (called *multivariables*), so that a synchronization between the productions is possible. Raoult calls these relations *rational tree relations* and gives also a characterization in terms of rational expressions.

Complementary Raoult's grammars, our first contribution are so-called *asynchronous tree automata* which recognize exactly the rational tree relations. With our automata theoretic approach it is possible to enter the topic of deterministic tree relations, and to address certain properties and (un-) decidability results of rational tree relations.

Rational tree relations in the mentioned format have a few drawbacks. They do not coincide with regular tree languages in the unary case, they are not closed under composition, and if considered as transductions they do not preserve regular tree languages. In [5] Raoult proposes a restriction of his tree grammars to so-called *transduction grammars* which resolve these problems. But these have the disadvantage that, when applied to unary trees, they can only be considered as a generalization of binary rational word relations, but not of the n-ary case. Furthermore, Raoult's restriction is difficult to adapt to tree automata, i. e. it misses a natural automata theoretic characterization. To take account of these problems our second contribution is such a natural restriction of rational tree relations (which semantically differs from Raoult's one). These so-called *separate-rational tree relations* meet all the properties demanded above and are still more powerful than automatic tree relations.

2 Rational Tree Relations

The starting point of our research is Raoult's definition of rational expressions via rational expressions [5].

Example 1. Consider the rational expression

$$(cx_1y_1, cbx_2y_2)^{*_{y_1y_2}} \cdot_{y_1y_2} (a, a) \cdot_{x_1x_2} (bz_1, bz_2)^{*_{z_1z_2}} \cdot_{z_1z_2} (a, a)$$

over the ranked alphabet $\Sigma = \{a^{(0)}, b^{(1)}, c^{(2)}\}$. In this example we use "multivariables" x_1x_2, y_1y_2, z_1z_2 (written also as X, Y, Z) which are subject to simultaneous substitution. Here, each instance of the multivariable X becomes substituted with two unary trees of same height.

Note, that unary relations of this class, called *rational tree languages* in the following, do not coincide with the class of regular tree languages:

Example 2. The rational expression $(fx_1x_2) \cdot_{x_1x_2} (gy_1, gy_2)^{*_{y_1y_2}} \cdot_{y_1y_2} (aa)$ over the ranked alphabet $\Sigma = \{f^{(2)}, g^{(1)}, a^{(0)}\}$ describes the tree language $T_{\text{sim}} = \{f(g^n a, g^n a) \mid n \in \mathbb{N}\}$ which is not regular.

We present *asynchronous tree automata* which recognize exactly Raoult's rational tree relations. The both considered Examples show that these automata have to provide following three mechanisms:

(1) Certain transitions are supposed to be used simultaneously. We achieve this by combining states to tuples of states which we call *macro states*. In the runs of our automata all states of a macro state have to be reached and left simultaneously.

(2) In addition we require some mechanism to allow asynchronous moves. We achieve this by the addition of ε-transitions. This enables the automaton to do a bottom-up step in one component and to stay in place in another component (possibly just changing the state).

(3) An unbounded number of instances of macro states has to be distinguished. In a run we have to distinguish whether states belong to the same or to different instances.

The Kleene-Theorem for tree languages can be adapted to asynchronous tree automata: A relation R of n-tuples of trees is rational if and only if there exists an asynchronous tree automaton $\mathcal{A}$ with $R(\mathcal{A}) = R$.

Some elementary closure properties and all undecidability results of rational word relations can be extended to trees easily: The class of n-ary rational tree relations is closed under union, not closed under intersection, and not closed under complementation. For rational tree relations R_1, R_2 it is undecidable to determine whether $R_1 \cap R_2 = \emptyset$, $R_1 \subseteq R_2$, and $R_1 = R_2$. But unlike binary rational relations over words, the class of binary rational tree relations is not closed under composition [5].

The *membership problem* (whether $(t_1, \ldots, t_n) \in R(\mathcal{A})$), the *emptiness problem* (whether $R(\mathcal{A}) = \emptyset$), and the *infinity problem* (whether $|R(\mathcal{A})|$ is infinite) are decidable for asynchronous tree automata.

Binary rational relations over words are also called *(rational) transductions*. They preserve regular and context-free languages, i. e. the image and the inverse image of a regular (resp. a context-free) language under a

transduction is again a regular (resp. context-free) language [1]. But binary rational tree relations do not even preserve regularity [5].

3 Separate-Rational Tree Relations

We present a subclass, called *separate-rational tree relations*, which overcome the mentioned "defects" of rational tree relations. The idea is to define a class of relations which can be computed by asynchronous tree automata which have all their macro states separated between the components, i. e. each state of a macro state can only occur in one component. For *separate-rational expressions* we demand variables of the same multivariable to be in different components of the generated tuple and that relations are generated by tuple of trees of height 2 at most (this assures that tuples can be computed by separate-asynchronous tree automata of the proposed form).

Example 3. (a) The relation from Example 1 is separate-rational. The rational expression can be rewritten as $(cx_1y_1, cx_2y_2)^{*_{y_1y_2}} \cdot_{y_1y_2} (a, a) \cdot_{x_1x_2} (x_1, bx_2) \cdot_{x_1x_2} (bz_1, bz_2)^{*_{z_1z_2}} \cdot_{z_1z_2} (a, a)$. It is generated by trees of height 2 at most, and all multivariables are separated between the components.

(b) T_{sim} from Example 2 is *not* separate-rational, because two different variables of one multivariable have to occur in the same component.

The Equivalence Theorem can be reformulated for separate-rational relations. We also obtain the same undecidability results and most of the closure properties as in the previous Section. Beyond this, separate-rational relations resolve the mentioned issues of rational tree relations: (a) The class of separate-rational tree languages is the class of regular tree languages. (b) The class of binary separate-rational tree relations is closed under composition. (c) The image and the inverse image of a regular tree language under a binary separate-rational tree relation R are again regular tree languages.

4 Outlook

Rational tree relations are more powerful than *linear tree transducers* and some cases of *term rewriting systems* [3]. These results do not hold for the separate-rational restriction. More expressive extensions of separate-rational relations with such features need to be investigated.

Asynchronous tree automata allow the definition of rational relations over unranked trees and the definition of deterministic rational tree relations (both over ranked and unranked trees). These theories were started in [4]. There, for determinism a top-down model was considered. A deterministic bottom-up model seems to be more challenging.

Acknowledgements. This work contains some results of my diploma thesis [4]. Special thanks go to Wolfgang Thomas for supervising this work and for his numerous helpful suggestions.

References

[1] Jean Berstel. *Transductions and Context-Free Languages*. Number 38 in Leitfäden der angewandten Mathematik und Mechanik. Teubner, Stuttgart, 1979.

[2] Olivier Carton, Christian Choffrut, and Serge Grigorieff. Decision problems among the main subfamilies of rational relations. *Theoretical Informatics and Applications*, 40(2):255–275, 2006.

[3] Antoine Meyer. On term rewriting systems having a rational derivation. In *Proceedings of FoSSaCS*, volume 2987 of *Lecture Notes in Computer Science*, pages 378–392. Springer, 2004.

[4] Frank G. Radmacher. Automatendefinierbare Relationen über Bäumen. Diploma thesis (revised version), RWTH Aachen, 2007. Available at `http://www.automata.rwth-aachen.de`.

[5] Jean-Claude Raoult. Rational tree relations. *Bulletin of the Belgian Mathematical Society*, 4(1):149–176, 1997.

Ein alternativer Primitivitätsbegriff für Wörter*

Daniel Reidenbach
Dept. of Computer Science,
Loughborough University,
Loughborough, LE11 3TU,
England
D.Reidenbach@lboro.ac.uk

Johannes C. Schneider§
Fachbereich Informatik,
Techn. Univ. Kaiserslautern,
Postfach 3049, 67653 K'lautern,
Deutschland
jschneider@informatik.uni-kl.de

Zusammenfassung

Die vorliegende Arbeit führt einen alternativen Primitivitätsbegriff für endliche Wörter ein, der darauf basiert, wie Wörter durch Homomorphismen aufeinander abgebildet werden können. Ein Wort w wird dabei als *m-imprimitiv* bezeichnet, wenn es ein kürzeres Wort v und Homomorphismen h, h' gibt, so dass $h(v) = w$ und $h'(w) = v$ gilt – ansonsten nennen wir es *m-primitiv* (engl. *morphically primitive*, vgl. [4]). Wir zeigen, dass diese Begrifflichkeit gut gewählt ist, da zum einen die m-primitiven Wörter eine Reihe typischer Eigenschaften von Primitivität zeigen und zum anderen dieses Attribut in einer Vielzahl kombinatorischer Gebiete, deren Definition auf Wörtern und Homomorphismen fußt, eine zentrale Rolle einnimmt. Ferner untersuchen wir einige grundlegende Eigenschaften von m-(im)primitiven Wörtern.

1 Motivation und Einleitung

Die Definition von primitiven Wörtern, also Wörtern, die sich nicht als nichttriviales Vielfaches eines anderen Wortes darstellen lassen, ist wohletabliert in der Wortkombinatorik und zahlreiche grundlegende Eigenschaften von Wörtern basieren darauf.

Innerhalb dieser Arbeit möchten wir einen neuen Begriff der Primitivität für Wörter einführen. Im Sinne einer möglichst allgemeinen Formulierung betrachten wir Wörter über einem unendlichen Alphabet, das wir durch die Menge $\mathbb{N}$ der natürlichen Zahlen repräsentieren. Um etwaige Verwechslungen zu vermeiden, trennen wir die Buchstaben (also die Elemente aus $\mathbb{N}$) eines Wortes durch das Zeichen "$\cdot$". Für ein Wort $w \in \mathbb{N}^*$ bezeichnen wir mit $\mathrm{symb}(w)$ die Menge der in w vorkommenden Symbole, d. h. $\mathrm{symb}(w) \subseteq \mathbb{N}$.

*Eine ausführlichere Darstellung der Resultat dieser Arbeit findet sich in [4].
§Korrespondierender Autor.

Unser Primitivitätsbegriff basiert nun darauf, wie Wörter von Homomorphismen aufeinander abgebildet werden können. So lassen sich z. B. die Wörter $w_1 := 1{\cdot}2{\cdot}2{\cdot}3{\cdot}3{\cdot}1{\cdot}2{\cdot}2$ und $w_2 := 1{\cdot}1{\cdot}2{\cdot}3{\cdot}3{\cdot}1{\cdot}1{\cdot}2$ durch die (löschenden) Homomorphismen $h_1, h_2 : \mathbb{N}^* \to \mathbb{N}^*$, definiert durch $h_1(1) := 1 \cdot 1 \cdot 2$, $h_1(2) := \varepsilon$, $h_1(3) := 3$ und $h_2(1) := \varepsilon$, $h_2(2) := 1 \cdot 2 \cdot 2$, $h_2(3) := 3$, ineinander überführen, d. h. es gilt $h_1(w_1) = w_2$ und $h_2(w_2) = w_1$. Wir verallgemeinern das und definieren eine *Äquivalenzrelation* $\equiv_*$ für beliebige Wörter $v, w \in \mathbb{N}^*$: Es sei $v \equiv_* w$ genau dann, wenn Homomorphismen $h, h' : \mathbb{N}^* \to \mathbb{N}^*$ existieren, so dass $h(v) = w$ und $h'(w) = v$ gilt. Neben $w_1 \equiv_* w_2$ gilt z. B. auch $v_1 \equiv_* w_2$ für $v_1 := 1 \cdot 3 \cdot 3 \cdot 1$, da $h_2(v_1) = w_2$ ist und ein Homomorphismus $h_3 : \mathbb{N}^* \to \mathbb{N}^*$, definiert durch $h_3(1) := \varepsilon$, $h_3(2) := 1$, $h_3(3) := 3$, das Wort w_2 auf v_1 abbildet.

Die kürzesten Elemente der durch $\equiv_*$ definierten Äquivalenzklassen bezeichnen wir als m-primitiv: Ein $w \in \mathbb{N}^*$ heißt genau dann *m-primitiv*, wenn kein $v \in \mathbb{N}^*$ existiert mit $|v| < |w|$ und $v \equiv_* w$. Andernfalls nennen wir w *m-imprimitiv*. Zu gegebenem $w \in \mathbb{N}^*$ bezeichnen wir ein m-primitives $v \in \mathbb{N}^*$ mit $v \equiv_* w$ als *m-primitive Wurzel (von w)*. Im obigen Beispiel ist v_1 m-primitiv und eine m-primitive Wurzel von w_1 und w_2.

Die folgenden Tatsachen verdeutlichen, dass unsere Terminologie sinnvoll gewählt ist. Sei $\mathrm{IMPRIM}(v) := \{w \in \mathbb{N}^* \mid |w| > |v| \text{ und } w \equiv_* v\}$ die *imprimitive Hülle* eines m-primitiven Wortes v sowie PRIM die Menge aller m-primitiven Wörter über $\mathbb{N}$. Dann gilt:

Proposition 1 $\bigcup_{v \in \mathrm{PRIM}} (v \cup \mathrm{IMPRIM}(v)) = \mathbb{N}^*$.

Durch Anwendung geeigneter Homomorphismen (geeignet im Sinne von $\equiv_*$) auf m-primitive Wörter kann man also *alle* Wörter erzeugen.

Für die folgende Überlegung benötigen wir einen weiteren Begriff: Wir bezeichnen einen Homomorphismus $r : \mathbb{N}^* \to \mathbb{N}^*$ als *Umbenennung*, wenn er injektiv ist und $|r(x)| = 1$ gilt für alle $x \in \mathbb{N}$, und sagen außerdem, ein Wort $v \in \mathbb{N}^*$ ist eine *Umbenennung* eines $w \in \mathbb{N}^*$, wenn eine Umbenennung r existiert mit $r(w) = v$.

Proposition 2 *Seien v und v' m-primitive Wörter. Es gilt* $\mathrm{IMPRIM}(v) \cap \mathrm{IMPRIM}(v') = \emptyset$ *genau dann, wenn v' keine Umbenennung von v ist.*

Hinreichend unterschiedliche m-primitive Wörter erzeugen also disjunkte imprimitive Hüllen. Um mittels einer Menge $\mathrm{PRIMBASE} \subseteq \mathbb{N}^*$ durch Homomorphismen alle Wörter erzeugen zu können (wie durch ganz PRIM in Proposition 1), muss also von jedem m-primitiven Wort mindestens eine Umbenennung in PRIMBASE vorhanden sein. Außerdem folgt:

Folgerung 1 *Sei $w \in \mathbb{N}^*$. Wenn v und v' m-primitive Wurzeln von w sind, so ist v' eine Umbenennung von v.*

Die m-primitiven Wurzeln eines Wortes sind also bis auf Umbenennung *eindeutig bestimmt*. Proposition 1 und 2 rechtfertigen in unseren Augen die Terminologie der Primitivität.

M-primitive Wörter sind als kombinatorisch reichhaltige Struktur aber nicht nur per se interessant, sondern spielen auch in anderen Gebieten eine wichtige Rolle, wie der folgende Charakterisierungssatz zeigt:

Satz 1 *Sei $v \in \mathbb{N}^*$. Die folgenden Aussagen sind äquivalent:*

1. *v ist m-primitiv.*
2. *v ist kein* Fixpunkt[1] *eines nichttrivialen Homomorphismus $h : \mathbb{N}^* \to \mathbb{N}^*$.*
3. *v ist ein* prägnantes Pattern[2]*.*
4. *Es existiert ein* eindeutiger[3] *injektiver Homomorphismus $\sigma : \mathbb{N}^* \to \{\mathtt{a}, \mathtt{b}\}^*$ bzgl. v.*

Wir können also zum einen von den Erkenntnissen aus der Forschung zu Fixpunkten, Patternsprachen und eindeutigen Homomorphismen profitieren und zum anderen durch die nähere Untersuchung von m-primitiven Wörtern zu neuen Ergebnissen auch in diesen Forschungszweigen beitragen.

2 Die imprimitive Hülle

Wir wollen uns nun zuerst denjenigen Homomorphismen zuwenden, welche die imprimitive Hülle eines m-primitiven Wortes erzeugen. Für ein m-primitives Wort $v \in \mathbb{N}^*$ nennen wir einen solchen Homomorphismus $h : \mathbb{N}^* \to \mathbb{N}^*$ mit $|h(v)| > v$ und $h(v) \equiv_* v$ einen *Imprimitivitätshomomorphismus für v*. Beispielsweise ist h_3 aus dem vorherigen Kapitel ein Imprimitivitätshomomorphismus.

Der folgende Satz charakterisiert die Imprimitivitätshomomorphismen:

Satz 2 *Sei $v \in \mathbb{N}^*$ m-primitiv. Ein Homomorphismus $h : \mathbb{N}^* \to \mathbb{N}^*$ ist genau dann ein Imprimitivitätshomomorphismus für v, wenn*

1. *zu jedem $x \in \mathrm{symb}(v)$ ein x_h existiert, das genau einmal in $h(x)$ vorkommt und für das $x_h \notin \mathrm{symb}(h(y))$, $y \in \mathrm{symb}(v) \setminus \{x\}$, gilt und*
2. *ein $x \in \mathrm{symb}(v)$ existiert mit $|h(x)| \geq 2$.*

[1] v heißt *Fixpunkt (eines nichttrivialen Homomorphismus $h : \mathbb{N}^* \to \mathbb{N}^*$)*, falls $h(v) = v$ gilt sowie $h(x) \neq x$ für ein $x \in \mathrm{symb}(v)$. Für weitere Informationen zu endlichen Fixpunkten konsultiere man z. B. Hamm und Shallit [2].

[2] Englischer Begriff: *succinct pattern*. Vgl. dazu Literatur zu Patternsprachen und für aktuelle Resultate Reidenbach [3].

[3] Ein Homomorphismus $\sigma : \mathbb{N}^* \to \mathbb{N}^*$ ist *eindeutig* bzgl. v, wenn es keinen Homomorphismus $\tau : \mathbb{N}^* \to \mathbb{N}^*$ gibt mit $\tau(v) = \sigma(v)$ und $\tau(x) \neq \sigma(x)$ für ein $x \in \mathrm{symb}(v)$ (vgl. Freydenberger, Reidenbach, Schneider [1]).

Daraus ergibt sich eine direkte Folgerung für die Anzahl der Zeichen eines Wortes und einer seiner m-primitiven Wurzeln:

Folgerung 2 *Seien $v, w \in \mathbb{N}^*$, so dass v eine m-primitive Wurzel von w ist. Dann gilt $|\mathrm{symb}(v)| \leq |\mathrm{symb}(w)|$ und, falls w m-imprimitiv ist, sogar $|\mathrm{symb}(v)| < |\mathrm{symb}(w)|$.*

3 Die m-primitiven Wurzeln

Bisher sind wir nicht weiter auf die Frage eingegangen, wie man zu einem gegebenen Wort w herausfinden kann, ob es sich um ein m-primitives oder m-imprimitives Wort handelt, und wie man im letzteren Fall eine m-primitive Wurzel von w findet. Bei einfachen, kurzen Wörtern w kann man leicht (bis auf Umbenennung) alle kürzeren Wörter v auf die Existenz von Homomorphismen prüfen, die $w \equiv_* v$ erfüllen. Bei längeren Wörtern ist dieses Verfahren allerdings sehr aufwändig – man kann sich hier jedoch Satz 1 zunutze machen und nach bestimmten Strukturen in w suchen, die w als Fixpunkt bzw. nicht-prägnantes Pattern charakterisieren (vgl. [2] bzw. [3]). Diese Strukturen sind wie folgt definiert:

Definition 1 *Sei $w \in \mathbb{N}^*$. Eine* Imprimitivitätsfaktorisierung für w *ist eine Abbildung $f : \mathbb{N}^+ \to \mathbb{N}^n \times (\mathbb{N}^+)^n$, $n \in \mathbb{N}$, so dass $u_0, u_1, \ldots, u_n \in \mathbb{N}^*$ existieren und für $f(w) = (x_1, x_2, \ldots, x_n; v_1, v_2, \ldots, v_n)$ gilt, dass $w = u_0\, v_1\, u_1\, v_2\, u_2 \ldots v_n\, u_n$ ist, und außerdem:*

(i) *Für alle $i \in \{1, 2, \ldots, n\}$ ist $|v_i| \geq 2$,*

(ii) *für alle $i \in \{0, 1, \ldots, n\}$ und für alle $j \in \{1, 2, \ldots, n\}$ ist $\mathrm{symb}(u_i) \cap \mathrm{symb}(v_i) = \emptyset$,*

(iii) *für alle $i \in \{1, 2, \ldots, n\}$ kommt x_i genau einmal in v_i vor, und wenn $x_i \in \mathrm{symb}(v_{i'})$ für ein $i' \in \{1, 2, \ldots, n\}$, so gilt $v_i = v_{i'}$ und $x_i = x_{i'}$.*

Es gilt dann:

Satz 3 *Sei $w \in \mathbb{N}^*$. Es ist w genau dann m-imprimitiv, wenn eine Imprimitivitätsfaktorisierung für w existiert.*

So ist z. B. das Wort $w_3 := 1{\cdot}2{\cdot}3{\cdot}3{\cdot}4{\cdot}3{\cdot}5{\cdot}5{\cdot}4{\cdot}3{\cdot}1{\cdot}2{\cdot}3{\cdot}3{\cdot}5$ m-imprimitiv, da eine Imprimitivitätsfaktorisierung $f(w_3) = (2, 4, 4, 2; 2{\cdot}3{\cdot}3, 4{\cdot}3, 4{\cdot}3, 2{\cdot}3{\cdot}3)$ existiert, welche wie folgt veranschaulicht werden kann:

$$w_3 = 1 \cdot \underbrace{②\cdot 3\cdot 3}_{v_1} \cdot \underbrace{④\cdot 3}_{v_2} \cdot 5 \cdot 5 \cdot \underbrace{④\cdot 3}_{v_3} \cdot 1 \cdot \underbrace{②\cdot 3\cdot 3}_{v_4} \cdot 5.$$

Man beachte, dass w_3 noch eine Reihe anderer Imprimitivitätsfaktorisierungen besitzt. Kürzt man im obigen Beispiel die v_i-Teile auf die (eingekreisten) Symbole x_i, so erhält man mit $v_3 := u_0 x_1 u_1 x_2 u_2 \ldots x_n u_n = 1{\cdot}2{\cdot}4{\cdot}5{\cdot}5{\cdot}4{\cdot}1{\cdot}2{\cdot}5$

ein kürzeres Wort mit $v_3 \equiv_* w_3$. Dieses Vorgehen kann man für jedes Wort anwenden, das über eine Imprimitivitätsfaktorisierung verfügt. Das gekürzte Wort ist allerdings nicht notwendigerweise m-primitiv. So ist z. B. v_3 m-imprimitiv; man kann es weiter kürzen auf $v_3' = 1 \cdot 4 \cdot 5 \cdot 5 \cdot 4 \cdot 1 \cdot 5$ und erhält so mit v_3' eine m-primitive Wurzel von w_3. In [4] wird diese Kürzungsoperation ausführlich untersucht. Außerdem wird ein hinreichendes Kriterium für Imprimitivitätsfaktorisierungen $f(w) = (x_1, x_2, \ldots, x_n; v_1, v_2, \ldots, v_n)$ angegeben, so dass das gekürzte Wort $u_0 x_1 u_1 x_2 u_2 \ldots x_n u_n$ m-primitiv ist.

4 Offene Probleme

Nicht zuletzt wegen der in Satz 1 aufgezeigten Querbezüge erachten wir die Frage, ob die m-Primitivität eines Wortes schnell entscheidbar ist, als sehr interessant. Auch Untersuchungen zur Häufigkeit von m-primitiven Wörtern (modulo Umbenennung bei fester Länge) halten wir für aufschlussreich.

Literatur

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

[2] D. Hamm und J. Shallit. Characterization of finite and one-sided infinite fixed points of morphisms on free monoids. Technical Report CS-99-17, Dept. of Computer Science, University of Waterloo, 1999. `http://www.cs.uwaterloo.ca/~shallit/papers.html`.

[3] D. Reidenbach. Discontinuities in pattern inference. *Theoretical Computer Science*. Zur Publikation angenommen.

[4] D. Reidenbach und J.C. Schneider. Morphically Primitive Words. In *Proc. 6th International Conference on Words, WORDS 2007*. Im Druck.

Recognizability of Iterative Picture Languages

Sibylle Schwarz Renate Winter
Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg
email: {schwarzs,winter}@informatik.uni-halle.de

1 Motivation

Pictures are rectangular arrays of colors from a finite alphabet. A set of pictures is called *picture language*. Picture languages are studied in various contexts such as pattern recognition and digital image processing [7].

A special class of picture languages can be defined by word languages over an alphabet $A_k = \{0, \ldots, k-1\}^2$ of coordinates [7]. We call such languages *iterative* picture languages. Iterative picture languages that are generated by regular word languages are used, for example, in digital image compression [7] and generation of fractal pictures [2].

We show that every iterative picture language generated by a regular word language is recognizable by a tiling system [1]. Furthermore, we present a non-regular word language that generates a recognizable picture language.

2 Iterative picture languages

Some languages of black and white pictures can be defined by word languages. We call such languages *iterative* picture languages. Every word $w = w_1 \cdots w_n$ over an alphabet $A_k = \{0, \ldots, k-1\}^2$ addresses a position $\mathsf{Pos}(w) \in \{0, \ldots, k^n - 1\}^2$ by

$$\mathsf{Pos}(w) = \left(\sum_{i=0}^{|w|-1} \pi_1(w_i) k^{|w|-i-1}, \sum_{i=0}^{|w|-1} \pi_2(w_i) k^{|w|-i-1} \right)$$

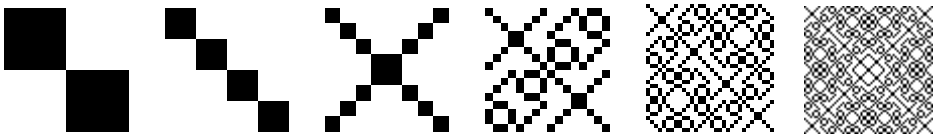
where $\pi_1(w)$ and $\pi_2(w)$ are the projections of letters $(m, n) \in \{0, \ldots, k-1\}$ to its first and second component. The inverse Pos^{-1} maps every position in a $k^n \times k^n$-square to a word in A_k^n. Every word language $L \subseteq A_k^+$ defines a set $\{p_i \mid i \in \mathbb{N}_+\}$ of black and white pictures where

$$p_i(m, n) = \begin{cases} 1 & \text{if } \mathsf{Pos}^{-1}(m, n) \in L \\ 0 & \text{otherwise} \end{cases}$$

For example, the regular word language

$$L = (((0,0)+(1,1))^*((0,1)+(1,0)))^{3*}((0,0)+(1,1))^*$$

defines the picture language $\mathsf{picture}(L) = \{p_i \mid i \in \mathbb{N}_+\}$ where



p_1 p_2 p_3 p_4 p_5 p_6

By definition, every iterative picture language P generated by a word language $L \subseteq A_k^+$ satisfies the following property:

(PS) for every $i \in \mathbb{N}_+$, P contains exactly one picture $p_i : \{0,\dots,k^i-1\}^2 \to \{0,1\}$.

3 Recognizable picture languages

Recognizability of picture languages is defined by tiling systems [1]. This definition coincides with recognizability of picture languages by several other computational devices (nondeterministic 4-way-automata, on-line tessellation automata).

For a picture language P, the set $B_{2,2}(P)$ contains all 2×2-sub-pictures of pictures in P. Given a set T of 2×2-pictures, the set $E_{2,2}(T)$ contains all pictures p satisfying $B_{2,2}(\{p\}) \subseteq T$. Note that in general, $B_{2,2}$ and $E_{2,2}$ are not inverse to each other.

For every picture $p : \{0,\dots,m\}\times\{0,\dots,m\} \to C$, the framed picture $\hat{p} : \{0,\dots,m+2\}\times\{0,\dots,m+2\} \to C\cup\{\#\}$ is defined by

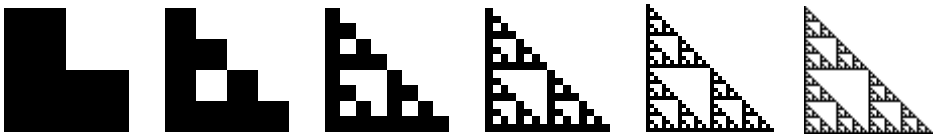
$$\hat{p}(i,j) = \begin{cases} p(i-1,j-1) & \text{if } (i-1,j-1) \in \{0,\dots,m\}\times\{0,\dots,n\} \\ \# & \text{otherwise} \end{cases}$$

This is extended to picture languages by $\hat{P} = \{\hat{p} \mid p \in P\}$.

A picture language P is called *local* if $\hat{P} = E_{2,2}(B_{2,2}(\hat{P}))$. A picture language P is called *recognizable* if P is a homomorphic image of a local language.

For example, the language P_S of finite Sierpinski triangles $P_S = \{p_i : \{0,\dots,2^i-1\}^2 \to \{0,1\} \mid i \in \mathbb{N}_+\}$ where for all $i \in \mathbb{N}_+$ and

$$k \in \{0,\dots,2^i-1\} : p_i(k,0) = 1$$
$$l \in \{1,\dots,2^i-1\} : p_i(0,l) = 0$$
$$(k,l) \in \{1,\dots,2^i-1\}^2 : p_i(k,l) = \begin{cases} 1 & \text{if } p_i(k-1,l-1)+p_i(k-1,l) = 1 \bmod 2 \\ 0 & \text{otherwise} \end{cases}$$



p_1 p_2 p_3 p_4 p_5 p_6

is local [4].

The set P_2 of all monochrome white squares of side 2^n where $n \in \mathbb{N}_+$ is not local but recognized by the local language P_S defined above and the mapping $\varphi(0) = \varphi(1) = 0$.

4 Picture languages generated by regular languages

Picture languages defined by regular languages are used for example in digital image compression [7] and generation of fractal pictures [2]. Every regular word language $L \subseteq A^*$ is recognized by a complete deterministic finite automaton $\mathcal{A} = (A, Q, \delta, s, F)$. By stepwise computation, this automaton defines a language $\mathsf{picture}(\mathcal{A}) = \{p_i \mid i \in \mathbb{N}_+\}$ of pictures with colors from Q by

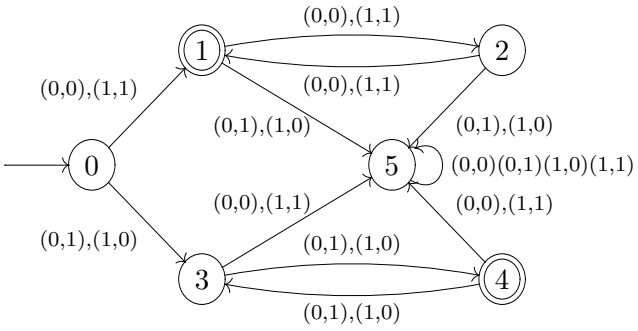
$$p_i(j, l) = \delta(s, \mathsf{Pos}^{-1}(j, l)).$$

Every complete deterministic finite automaton $\mathcal{A} = (A, Q, \delta, s, F)$ defines a two-dimensional D0L-System (Q, R, s) where

$$\text{for every state } q \in Q: \quad R(q) = \begin{array}{|ccc|}\hline \delta(q,(0,0)) & \cdots & \delta(q,(0,k-1)) \\ \vdots & \ddots & \vdots \\ \delta(q,(k-1,0)) & \cdots & \delta(q,(k-1,k-1)) \\ \hline \end{array}$$

For every D0L-system, iterated simultaneous application of the substitution of $q \in Q$ by $R(q)$ generates a picture language. The D0L-system defined above generates the picture language $\mathsf{picture}(\mathcal{A})$.

For example, the automaton



generates the picture language

$\mathsf{picture}(\mathcal{A}) = \{p_i : \{0,\ldots,2^i-1\}^2 \to \{0,\ldots,5\} \mid i \in \mathbb{N}_+\}$ where

for all $j \in \mathbb{N}_+$: $p_{2j-1} = \begin{array}{|ccccc|}\hline \mathbf{1} & 5 & \cdots & 5 & 3 \\ 5 & \mathbf{1} & \cdots & 3 & 5 \\ \vdots & & \ddots & & \vdots \\ 5 & 3 & \cdots & \mathbf{1} & 5 \\ 3 & 5 & \cdots & 5 & \mathbf{1} \\ \hline\end{array}$ and $p_{2j} = \begin{array}{|ccccc|}\hline 2 & 5 & \cdots & 5 & \mathbf{4} \\ 5 & 2 & \cdots & \mathbf{4} & 5 \\ \vdots & & \iddots & & \vdots \\ 5 & \mathbf{4} & \cdots & 2 & 5 \\ \mathbf{4} & 5 & \cdots & 5 & 2 \\ \hline\end{array}$

Application of the characteristic function of the set of final states results in a language of white squares with black diagonals in alternating direction.

In [4], it was stated (with a proof sketch) that every picture language generated by a two-dimensional D0L-system is recognizable. The idea of the proof relies on recognizability of an auxiliary information structure [6]. A similar structure is used in [5].

Pointwise application of the characteristic function $\chi_F : Q \to \{0,1\}$ of the set F of final states of the automaton maps every $p_i : \{0,\ldots,k^i-1\}^2 \to Q \in \mathsf{picture}(\mathcal{A}, n)$ to a picture $q_i : \{0,\ldots,k^i-1\}^2 \to \{0,1\} \in \mathsf{picture}(L_{\mathcal{A}}, n)$ Hence we have

$$\mathsf{picture}(L) = \chi_F(\mathsf{picture}(\mathcal{A}))$$

By the above considerations, we get the following result.

Theorem 1 *For every regular word language $L \subseteq A_k^+$, the picture language* $\mathsf{picture}(L)$ *is recognizable.*

Since the property (PS) is a strong restriction to a picture language, one might conjecture that every recognizable picture language satisfying this property (PS) is $\mathsf{picture}(L)$ for a regular word language L. The following picture language is a counterexample:

The word language $L = \{A_2^{2^n} \mid n \in \mathbb{N}_+\}$ is not regular by a simple pumping argument. $\mathsf{picture}(L)$ is a set of monochrome pictures p_i. p_i is a black square of side 2^{2^n} if $i = 2^n$ for an $n \in \mathbb{N}_+$ and otherwise a white square.

Kari and Moore showed in [3] that for every recursively enumerable set M of natural numbers, the set of all monochrome squares of side $m \in M$ is recognizable. Both $M_1 = \{2^{2^n} \mid n \in \mathbb{N}\}$ and $M_2 = \{2^n \mid n \in \mathbb{N}\} \setminus M_1$ are recursively enumerable. P_1 is the set of all black squares of side from M_1 and P_2 is the set of all white squares of side from M_2. By the result from [3], P_1 and P_2 are recognizable and so is their union $\mathsf{picture}(L) = P_1 \cup P_2$.

References

[1] Dora Giammaresi and Antonio Restivo. Two-dimensional languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume III, pages 215–268. Springer-Verlag, 1997.

[2] Helmut Jürgensen, Ludwig Staiger, and Hideki Yamasaki. Finite automata encoding geometric figures. *Theor. Comput. Sci.*, 381(1-3):33–43, 2007.

[3] Jarkko Kari and Cristopher Moore. Rectangles and squares recognized by two-dimensional automata. In Juhani Karhumäki, Hermann A. Maurer, Gheorghe Paun, and Grzegorz Rozenberg, editors, *Theory Is Forever*, volume 3113 of *Lecture Notes in Computer Science*, pages 134–144. Springer, 2004.

[4] Kristian Lindgren, Cristopher Moore, and Mats Nordahl. Complexity of two-dimensional patterns. Working Papers 97-03-023, Santa Fe Institute, March 1997. available at http://ideas.repec.org/p/wop/safiwp/97-03-023.html.

[5] Klaus Reinhardt. The #a = #b pictures are recognizable. In Afonso Ferreira and Horst Reichel, editors, *STACS*, volume 2010 of *Lecture Notes in Computer Science*, pages 527–538. Springer, 2001.

[6] Raphael M. Robinson. Undecidability and Nonperiodicity for Tilings of the Plane. *Inventiones Mathematicae*, 12(3):177–209, 1971.

[7] Karel Čulik II and Jarkko Kari. Digital images and formal languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume III, pages 599–616. Springer-Verlag, 1997.

Weighted Monadic Datalog

Torsten Stüber, Heiko Vogler
Department of Computer Science
Technische Universität Dresden
{stueber, vogler}@tcs.inf.tu-dresden.de

1 Introduction

Data on the Web are commonly stored in semistructured databases which are represented by means of XML documents, typically abstracted by unranked trees. The popularity of XML led to an immense increase of research concerning information extraction techniques for semistructured databases or, equivalently, unranked trees. Common concepts for recognizing or defining languages of unranked trees are unranked tree automata [7, 2] and MSO logic over unranked trees [8]. A more recent approach is monadic datalog that has been studied by Gottlob and Koch [5]. A monadic datalog program is a finite set of Horn-clauses that do not allow for function symbols; moreover, predicates that occur in head atoms, have to be nullary or unary. Monadic datalog has been shown to have the same expressiveness as MSO logic on unranked trees and, thus, unranked tree automata. Moreover, it can be evaluated in linear time with respect to the size of the input tree and the size of the monadic datalog program.

The notions of tree automata and MSO logic have been extended by weights taken from a semiring; the behavior of weighted tree automata and weighted MSO logic are characterized as formal tree series, i.e., mappings from the set of trees to the semiring (cf. [3]). In this paper we will introduce a weighted version of monadic datalog. Our work is based on semiring-based constraint logic programming (studied by Bistarelli et al. [1]), an extension of (constraint) logic programming in the sense that weights, i.e., semiring elements, are allowed to occur in Horn clauses. In particular, we will discuss how the aforementioned expressiveness and evaluation complexity results of monadic datalog carry over to weighted monadic datalog.

2 Basic Definitions

We will make use of the following notations: for every $n \in \mathbb{N}$ the set $\{1, \ldots, n\}$ is denoted by $[n]$. For an alphabet Σ the set U_Σ denotes the *set of unranked trees* over Σ and for a $t \in U_\Sigma$ the *set of positions in* t is denoted

by $pos(t)$ (e.g., $pos(\alpha(\beta(\beta), \gamma)) = \{\varepsilon, 1, 11, 2\}$). For a position $w \in pos(t)$, $t|_w$ denotes the subtree, $t(w)$ denotes the label, and $\mathrm{rk}_t(w)$ denotes the rank of t at w. For a set C and a semiring S we denote by $S\langle\langle C \rangle\rangle$ the *set of all formal power series* φ from C to S, i.e., mappings of the form $\varphi : C \to S$, and for every $c \in C$ we write (φ, c) instead of $\varphi(c)$. A *ranked set* B is a set where every element $b \in B$ is associated with a natural number, its *rank*. If the rank of b is n, then we also write $b^{(n)}$. We call B *monadic* iff the rank of no elements is greater than 1. For a ranked set B and a set C we denote by $A_{B,C}$ the set $\{b(c_1, \ldots, c_k) \mid b \in B, k \text{ is the rank of } b, \text{ and } c_1, \ldots, c_k \in C\}$.

Throughout this paper, V denotes the set of *variable symbols*, P stands for a monadic ranked set (of *user-defined predicates*), Σ for an arbitrary alphabet, and $S = (S, \oplus, \otimes, 0, 1)$ for an arbitrary semiring. The alphabet Σ induces a ranked set $sp(\Sigma)$ of *predefined predicates* defined as $sp(\Sigma) = \{\mathrm{root}^{(1)}, \mathrm{leaf}^{(1)}, \mathrm{firstchild}^{(2)}, \mathrm{nextsibling}^{(2)}, \mathrm{lastsibling}^{(1)}\} \cup \{\mathrm{label}_\sigma^{(1)} \mid \sigma \in \Sigma\}$.

The set A of *weighted monadic datalog atoms* is the set $A = A_{P,V} \cup A_{sp(\Sigma),V} \cup S$. By $Var(a)$ we denote the set of variables that occur in the atom $a \in A$. We can instantiate atoms with positions of a tree $t \in U_\Sigma$ as follows: let $\rho : Var(a) \to pos(t)$; then the *ρ-instance of a*, denoted by $\rho(a)$, is obtained from a by replacing every variable v that occurs in a by $\rho(v)$. Thus, the set of all atom instances for a given tree t is the set $A_{P,pos(t)} \cup A_{sp(\Sigma),pos(t)} \cup S$.

A *weighted monadic datalog rule* (for short: *wmd rule*) is of the form $a \leftarrow b_1, \ldots, b_n$, where $a \in A_{P,V}$, $n \geq 1$, and $b_1, \ldots, b_n \in A$. We extend the notions of occurring variables and ρ-instances to rules: let $r = a \leftarrow b_1, \ldots, b_n$; then $Var(r) = Var(a) \cup \bigcup_{i \in [n]} Var(b_i)$ and for a mapping $\rho : Var(r) \to pos(t)$ we call $\rho(r) = \rho(a) \leftarrow \rho(b_1), \ldots, \rho(b_n)$ the *ρ-instance of r*. Note, that in the body of $\rho(r)$ there can be atom instances over predefined predicates that are not satisfied by t[1]. We are not interested in such rule instances and define $\Phi_{r,t} = \{\rho : Var(r) \to pos(t) \mid t \text{ satisfies every } a \in A_{sp(\Sigma),pos(t)} \text{ in the body of } \rho(r)\}$. A *weighted monadic datalog program* (for short: *wmd program*) is a finite set R of wmd rules. A *weighted monadic datalog query* (for short: *wmd query*) from Σ to S is a pair (R, q), where R is a wmd program and q is a nullary user-defined predicate.

Next we are going to define the semantics of weighted monadic datalog. From now on, R denotes an arbitrary wmd program and q an arbitrary nullary user-defined predicate. Let $t \in U_\Sigma$. A *t-interpretation* is a mapping $I : A_{P,pos(t)} \to S$. The set of all t-interpretations is denoted by $\mathcal{I}_t$. For every $s \in S$ we denote by $I^s \in \mathcal{I}_t$ the mapping $I^s(a) = s$ for every $a \in A_{P,pos(t)}$. By abuse of notation we extend interpretations to all kinds of atom instances:

[1]The meaning of the predefined predicates is obvious, e.g., $\mathrm{firstchild}(w, v)$ is true iff v is the first child of w; $\mathrm{nextsibling}(w, v)$ is true iff w, v have the same parent and v is the right neighbor of w; $\mathrm{lastsibling}(w)$ is true iff w is the youngest child of its parent node.

for $I \in \mathcal{I}_t$ we define $I(s) = s$ for every $s \in S$ and for every $a \in A_{sp(\Sigma),pos(t)}$ we define $I(a) = 1$ if a is true in t and $I(a) = 0$ if a is false in t. We define a mapping $\mathcal{T}_{R,t} : \mathcal{I}_t \to \mathcal{I}_t$, called the *immediate consequence operator of R and t*, as follows for every $I \in \mathcal{I}_t$ and $a \in A_{P,pos(t)}$:

$$\mathcal{T}_{R,t}(I)(a) = \bigoplus_{r \in R} \bigoplus_{\substack{\rho \in \Phi_{r,t} \\ \rho(r) = a \leftarrow b_1, \ldots, b_n}} I(b_1) \otimes \cdots \otimes I(b_n) \ .$$

In accordance with conventional horn calculus we are interested in determining the least fixpoint of $\mathcal{T}_{R,t}$ w.r.t some order on the set of interpretations. We provide this order through the semiring S: a semiring S together with a partial order $\leq$ is called an *ω-cpo semiring* iff S is a semiring, $(S, \leq)$ is an ω-cpo[2], and both semiring operations are monotone and continuous with respect to $\leq$. Examples of ω-cpo semirings are the Boolean semiring, the arctical semiring, every complete semiring that is infinitary idempotent, and every c-semiring (cf. [1]). For the following discussion assume that S is an ω-cpo semiring.

We extend the order on S to $\mathcal{I}_t$ by defining for every $I_1, I_2 \in \mathcal{I}_t$ that $I_1 \leq I_2$ iff $I_1(a) \leq I_2(a)$ for every $a \in A_{P,pos(t)}$. We obtain that $(\mathcal{I}_t, \leq)$ is an ω-cpo and that the immediate consequence operator $\mathcal{T}_{R,t}$ is monotone and continuous. Hence, we can apply the fixpoint theorem of Knaster-Tarski and obtain that the supremum of the set $\{\mathcal{T}^n_{R,t}(I^\perp) \mid n \in \mathbb{N}\}$[3] is the least fixpoint of $\mathcal{T}_{R,t}$. We denote this least fixpoint by $\mathcal{T}^\omega_{R,t}$. The *unranked tree series $[\![(R,q)]\!] \in S\langle\langle U_\Sigma \rangle\rangle$ defined by the wmd query (R,q)* is given by $([\![(R,q)]\!], t) = \mathcal{T}^\omega_{R,t}(q())$ for every $t \in U_\Sigma$. The set of all unranked tree series defined by wmd queries from Σ to S is denoted by $uMD(\Sigma, S)$. Two wmd queries are *semantically equivalent* iff they define the same unranked tree series. The definition of the semantics yields that monadic datalog can be imbedded in weighted monadic datalog by using the Boolean semiring.

The definition of the immediate consequence operator makes it obvious that for every interpretation I the value of an atom instance $a \in A_{P,pos(t)}$ under $\mathcal{T}_{R,t}(I)$ depends on the value of some atom instances $b_1, \ldots, b_n$ under I. These dependencies might be circular. Programs that do not allow for such circular dependencies will be called *non-circular*, more precisely, R is called non-circular iff there is no $t \in U_\Sigma$ such that the *dependency graph* of R and t is cyclic. The dependency graph of R and t is the graph $(A_{P,pos(t)}, E)$ such that for every $r \in R$ and $\rho \in \Phi_{r,t}$ there is an edge from the head atom instance of $\rho(r)$ to every body atom instance over user-defined predicates of $\rho(r)$.

For the remainder of this section let S be an arbitrary semiring and let R be non-circular. Let $n = |P| \cdot |pos(t)|$. A consequence of the non-

[2] An ω-cpo $(S, \leq)$ is a partially ordered set $(S, \leq)$ such that there is a least element $\perp$ and every countable chain in S has a supremum.

[3] $\mathcal{T}^n_{R,t}(I^\perp)$ denotes the n-fold application of $\mathcal{T}_{R,t}$ to $I^\perp$.

circularity of R is that for every $I, I' \in \mathcal{I}_t$ we have $\mathcal{T}^n_{R,t}(I) = \mathcal{T}^n_{R,t}(I')$ and that $\mathcal{T}^n_{R,t}(I)$ is the *unique* fixpoint of $\mathcal{T}_{R,t}$. Since $\mathcal{T}_{R,t}$ has exactly one fixpoint, we can associate a well-defined semantics with non-circular programs based on arbitrary (i.e., not necessarily ordered) semirings: the *unranked tree series* $[\![(R,q)]\!]^{nc} \in S\langle\langle U_\Sigma \rangle\rangle$ *nc-defined by the wmd query* (R,q) is given by $([\![(R,q)]\!]^{nc}, t) = \mathcal{T}^n_{R,t}(I^0)(q())$ for every $t \in U_\Sigma$. The set of all unranked tree series nc-defined by wmd queries from Σ to S is denoted by $uMD^{nc}(\Sigma, S)$.

If S is an ω-cpo semiring, then both semantics $[\![(R,q)]\!]$ and $[\![(R,q)]\!]^{nc}$ are applicable to (R,q). It can be shown that they coincide, i.e., $[\![(R,q)]\!] = [\![(R,q)]\!]^{nc}$. Hence, we obtain the following lemma.

Lemma 1. *If S is an ω-cpo semiring, then $uMD^{nc}(\Sigma, S) \subseteq uMD(\Sigma, S)$. However, there is an alphabet Σ and a commutative ω-cpo semiring S such that $uMD^{nc}(\Sigma, S) \subset uMD(\Sigma, S)$.*

3 Expressiveness

Here we will show how weighted monadic datalog relates to weighted tree automata on unranked trees (cf. [4]) with regard to their expressiveness . An *unranked weighted tree automaton* (for short: *uwta*) over Σ and S is a tuple $M = (Q, \delta, \gamma)$, where Q is a finite set, $\delta : Q \times \Sigma \to S^{rec}\langle\langle Q^* \rangle\rangle$[4], and $\gamma : Q \to S$. The automaton M induces a mapping $h_M : U_\Sigma \to S^Q$ that is defined as follows for every $t \in U_\Sigma$ and $q \in Q$: $h_M(t)_q = \bigoplus_{q_1,\ldots,q_k \in Q} (\delta(q, \sigma), q_1 \cdots q_k) \otimes h_M(t|_1)_{q_1} \otimes \cdots \otimes h_M(t|_k)_{q_k}$, where $k = \mathrm{rk}_t(\varepsilon)$ and $\sigma = t(\varepsilon)$. The *unranked tree series* $[\![M]\!] \in S\langle\langle U_\Sigma \rangle\rangle$ *recognized by* M is defined for every $t \in U_\Sigma$ as $([\![M]\!], t) = \bigoplus_{q \in Q} \gamma(q) \otimes h_M(t)_q$. The set of all unranked tree series recognized by uwta over Σ and S is denoted by $uTA(\Sigma, S)$.

Theorem 2. *For every alphabet Σ and commutative semiring S we have $uTA(\Sigma, S) \subseteq uMD^{nc}(\Sigma, S)$. There is an alphabet Σ and a commutative semiring S such that $uTA(\Sigma, S) \subset uMD^{nc}(\Sigma, S)$.*

4 Combined Complexity

We will express the evaluation complexity of weighted monadic datalog in terms of the number of applications of semiring operations depending on the size of the input tree and the size of the wmd query (called *combined complexity*). We call a rule $r \in R$ *connected* iff the graph $(Var(r), E_r)$

[4] $S^{rec}\langle\langle Q^* \rangle\rangle$ is the set of recognizable formal power series over the alphabet Q and the semiring S (cf. [6])

with $E_r = \{(v_1, v_2) \mid$ there is a body atom a in r with $v_1, v_2 \in Var(a)\}$ is connected. The wmd program R is called *connected* iff every $r \in R$ is connected.

Theorem 3. *Let (R, q) be a wmd query and $t \in U_\Sigma$. If R is non-circular and connected, we can compute $([\![(R, q)]\!]^{nc}, t)$ by $\mathcal{O}(size(R) \cdot |pos(t)|)$ applications of semiring operations. If S is commutative and R is non-circular, then $([\![(R, q)]\!]^{nc}, t)$ can be computed by $\mathcal{O}(size(R) \cdot |pos(t)|)$ applications of semiring operations. If S is a finite commutative ω-cpo semiring, then $([\![(R, q)]\!], t)$ can be computed by $\mathcal{O}(|S| \cdot (size(R) \cdot |pos(t)|)^2)$ applications of semiring operations.*

References

[1] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.

[2] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets: Version 1, April 2001. Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology, 2001.

[3] M. Droste and H. Vogler. Weighted tree automata and weighted logics. *Theoret. Comput. Sci.*, 366:228–247, 2006.

[4] M. Droste and H. Vogler. Weighted logics for XML. manuscript, 2007.

[5] G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for web information extraction. *J. ACM*, 51(1):74–113, 2004.

[6] M.P. Schützenberger. On the definition of a family of automata. *Inf. and Control*, 4:245–270, 1961.

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

[8] J.W. Thatcher and J.B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Math. Syst. Theory*, 2(1):57–81, 1968.

On a Knuth-like 0-1-2-Principle for Parallel Prefix Computation
(Extended Abstract)

Janis Voigtländer

Institut für Theoretische Informatik
Technische Universität Dresden
01062 Dresden, Germany

voigt@tcs.inf.tu-dresden.de

Abstract

Parallel prefix computation is the task to compute, given inputs $x_1, \ldots, x_n$ and an associative operation $\oplus$, the outputs $x_1, x_1 \oplus x_2, \ldots, x_1 \oplus x_2 \oplus \cdots \oplus x_n$. It has numerous applications in the hardware and algorithmics fields. There is a wealth of solutions, employing the associativity of $\oplus$ in different ways to realize different trade-offs between certain characteristics of the resulting "circuits", and more keep coming up. An obvious concern is that for correctness of such new, and increasingly complex, methods. Of course, any formal analysis or testing of new solution candidates must be sufficiently generic, given that in the problem specification neither the type of the inputs $x_1, \ldots, x_n$, nor any specifics (apart from associativity) of $\oplus$ are fixed.

Here Knuth's 0-1-Principle comes to mind. It states that if an oblivious sorting algorithm is correct on boolean valued input sequences, then it is correct in general. This greatly eases the analysis of such algorithms. Is something similar possible for parallel prefix computation?

1 The Problem

We cast the problem and our analysis in the purely functional programming language Haskell. Since it is universal, it allows us to precisely capture the notion of an "algorithm" (that may, or may not, be a correct solution to the parallel prefix computation task) in the most general way. It also provides the mathematical expressivity and reasoning techniques that we need. Since Haskell's notation is quite intuitive, we do not pause for a detailed introduction to the language, instead introducing concepts as we go.

Haskell's standard library provides a function

$$\mathit{foldl1} :: (\alpha \to \alpha \to \alpha) \to [\alpha] \to \alpha$$

such that

$$\mathit{foldl1}\ \oplus\ [x_1, \ldots, x_n] \quad = \quad (\cdots(x_1 \oplus x_2) \oplus \cdots) \oplus x_n$$

for every binary operation and type-conforming, non-empty input list.[1] While Haskell allows empty lists as well, we from now on consider $[\alpha]$ to be the type of non-empty, finite lists. The variable α indicates that the function *foldl1* can be used at arbitrary (but then fixed) type.

Now we can specify our computation task as follows:

$$\begin{array}{l}\mathit{scanl1} :: (\alpha \to \alpha \to \alpha) \to [\alpha] \to [\alpha] \\ \mathit{scanl1}\ \oplus\ \mathit{xs} = \mathit{map}\ (\lambda k \to \mathit{foldl1}\ \oplus\ (\mathit{take}\ (k+1)\ \mathit{xs}))\ [0..\mathit{length}\ \mathit{xs} - 1]\end{array}$$

by using Haskell's syntactic sugar $[0..n]$ for the list $[0, 1, 2, \ldots, n]$ (with $n :: \mathit{Int}$ and $n \geq 0$) and some further standard functions, whose semantics should be clear from their names and types:

$$\begin{array}{ll}\mathit{length} & :: [\alpha] \to \mathit{Int} \\ \mathit{take} & :: \mathit{Int} \to [\alpha] \to [\alpha] \\ \mathit{map} & :: (\alpha \to \beta) \to [\alpha] \to [\beta]\end{array}$$

The correctness issue we are interested in is whether a given function

$$\mathit{candidate} :: (\alpha \to \alpha \to \alpha) \to [\alpha] \to [\alpha]$$

is semantically equivalent to *scanl1* for associative operations as first argument. The perfect analogue to Knuth's 0-1-Principle would be if this were so provided one can establish that equivalence at least on the boolean type. Unfortunately, it does not hold. Exhaustive testing shows that $x_1 \oplus (x_1 \oplus (x_1 \oplus x_2)) = x_1 \oplus x_2$ for every $x_1, x_2 :: \mathit{Bool}$ and $\oplus :: \mathit{Bool} \to \mathit{Bool} \to \mathit{Bool}$ (associative or not). Given this, it is easy to manufacture a *candidate* function that is equivalent to *scanl1* at type *Bool*, but not in general (not even for associative $\oplus$).

2 The Claim

We claim that parallel prefix computation enjoys a 0-1-2-Principle. That is, using the following Haskell datatype:

$$\textbf{data}\ \mathit{Three} = \mathit{Zero} \mid \mathit{One} \mid \mathit{Two}$$

the following theorem holds for every *candidate* function.

Theorem 1 *If for every associative* $\oplus :: \mathit{Three} \to \mathit{Three} \to \mathit{Three}$ *and* $\mathit{xs} :: [\mathit{Three}]$, *candidate* $\oplus$ *xs and scanl1* $\oplus$ *xs give equal results, then the same holds for every type* τ, *associative* $\oplus :: \tau \to \tau \to \tau$, *and* $\mathit{xs} :: [\tau]$.

[1] Note that function application is denoted by juxtaposition, just as in the λ-calculus. Also, we abuse syntax a bit here and in what follows, by using $\oplus$ for both infix and stand-alone occurrences, whereas the latter would have to be given as $(\oplus)$ in proper Haskell.

3 The Proof Outline

Rather than relating correctness at type *Three* to correctness at arbitrary type directly, we perform an indirection via the type $[Int]$ (essentially because that type can be used to embed freely generated semigroups over arbitrarily large finite generator sets). To formulate the indirection, we need another standard function

$$(+\!\!+) :: [\alpha] \to [\alpha] \to [\alpha]$$

that concatenates lists of equal type, as well as a function

$$wrap :: \alpha \to [\alpha]$$

that wraps an element into a singleton list, and the following function:

$$\begin{array}{l} ups :: Int \to [[Int]] \\ ups\ n = map\ (\lambda k \to [0..k])\ [0..n] \end{array}$$

Then, we prove two propositions. Theorem 1 arises by combining these.

Proposition 1 *If for every associative* $\oplus :: Three \to Three \to Three$ *and* $xs :: [Three]$*, candidate* $\oplus$ *xs and scanl1* $\oplus$ *xs give equal results, then for every* $n :: Int$*,*

$$candidate\ (+\!\!+)\ (map\ wrap\ [0..n]) \quad = \quad ups\ n$$

Proposition 2 *If for every* $n :: Int$*,*

$$candidate\ (+\!\!+)\ (map\ wrap\ [0..n]) \quad = \quad ups\ n$$

then for every type τ*, associative* $\oplus :: \tau \to \tau \to \tau$*, and* $xs :: [\tau]$*, candidate* $\oplus$ *xs and scanl1* $\oplus$ *xs give equal results.*

A slightly different formulation of the second proposition was already known to Sheeran in the context of her work on systematically searching for new solutions to the parallel prefix computation task [She07]. In fact, that was our inspiration for investigating a Knuth-like principle for parallel prefix computation, trying to bring the discrimination between good and bad candidates down from the infinite type $[Int]$ to as small a type as possible.

The key to connecting the behaviour of *candidate* at different types is relational parametricity [Rey83], in the form of free theorems [Wad89]. It allows to derive the following lemma just from the polymorphic type of *candidate* (i.e., without having to know the function's actual definition).

Lemma 1 *For every choice of types* τ_1, τ_2 *and functions* $f :: \tau_1 \to \tau_2$, $\otimes :: \tau_1 \to \tau_1 \to \tau_1$, *and* $\oplus :: \tau_2 \to \tau_2 \to \tau_2$, *if for every* $x, y :: \tau_1$,

$$f\ (x \otimes y) \quad = \quad (f\ x) \oplus (f\ y)$$

then for every $z :: [\tau_1]$,

$$map\ f\ (candidate \otimes z) \quad = \quad candidate \oplus (map\ f\ z)$$

There is no need to do any proof for this lemma, as this is already accounted for in the general methodology of free theorems. The above is a simple produce of that methodology, and in fact can be obtained as result of a completely automated process, with just the type of *candidate* as input. The crucial point now is to cleverly instantiate types and functions quantified over in the above lemma in such a way that the instantiated versions lend themselves to proving our desired propositions.

The key insights on why a three-valued type suffices to distinguish good from bad candidates for parallel prefix computation are encapsulated in the following lemma. Quite nicely, it can be formulated in terms of *foldl1*, i.e., essentially in terms of just one element of the output list of *scanl1* at a time.

Lemma 2 *Let* $js :: [Int]$ *and* $k :: Int$ *with* $k \geq 0$. *If for every* $h :: Int \to Three$ *and associative* $\oplus :: Three \to Three \to Three$,

$$foldl1\ \oplus\ (map\ h\ js) \quad = \quad foldl1\ \oplus\ (map\ h\ [0..k])$$

then $js = [0..k]$.

We do not spell out the proof here, but mention that only two associative functions of type $Three \to Three \to Three$ are actually needed in it:

$$\begin{array}{ll}
\oplus_1 :: Three \to Three \to Three & \oplus_2 :: Three \to Three \to Three \\
x \quad \oplus_1 Zero = x & x \oplus_2 Zero = x \\
Zero \oplus_1 One = One & x \oplus_2 One = One \\
x \quad \oplus_1 y \quad = Two & x \oplus_2 Two = Two
\end{array}$$

The functions of type $Int \to Three$ to be used in combination with these are $(h_1\ i)$ for every $0 \leq i \leq k$ and $(h_2\ i)$ for every $0 \leq i < k$, respectively, where h_1 and h_2 are given as follows:

$$\begin{array}{ll}
h_1 :: Int \to Int \to Three & h_2 :: Int \to Int \to Three \\
h_1\ i\ j \mid i \equiv j \quad\quad\quad = One & h_2\ i\ j \mid i \equiv j \quad\quad = One \\
\quad\quad \mid 0 \leq j\ \&\&\ j \leq k = Zero & \quad\quad \mid i \equiv j - 1 \ = Two \\
\quad\quad \mid otherwise \quad\quad = Two & \quad\quad \mid otherwise = Zero
\end{array}$$

Using some laws about list-manipulating functions, most of which can be obtained automatically by using the machinery of free theorems, the

previous lemma can be lifted to a statement on the overall computation of *scanl1*, and finally to Proposition 1. The proof of the latter additionally uses Lemma 1 instantiated as follows: $\tau_1 = [Int]$, $\tau_2 = Three$, $f = foldl1\ \oplus \cdot map\ h$, $\otimes = (+\!\!+)$, and $z = map\ wrap\ [0..n]$. One further statement we need in that proof is that for every type τ, associative $\oplus :: \tau \to \tau \to \tau$, and $x, y :: [\tau]$,

$$foldl1\ \oplus\ (x +\!\!+ y) \quad = \quad (foldl1\ \oplus\ x) \oplus (foldl1\ \oplus\ y)$$

In fact, the above can be seen as the very definition of associativity.

We do not go into detail here about proving the remaining half of our 0-1-2-Principle, i.e., Sheeran's original observation. Suffice it to say that the proof of Proposition 2 again uses Lemma 1, this time instantiated as follows: $\tau_1 = [Int]$, $\tau_2 = \tau$, $f = foldl1\ \oplus \cdot map\ (xs!!)$, $\otimes = (+\!\!+)$, and $z = map\ wrap\ [0..length\ xs - 1]$, where $(xs!!)$ is a function that expects an argument of type *Int* and will return the element at the corresponding position in xs, counting from 0.

References

[Rey83] J.C. Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing, Proceedings*, pages 513–523. Elsevier Science Publishers B.V., 1983.

[She07] M. Sheeran. Searching for prefix networks to fit in a context using a lazy functional programming language. Talk at *Hardware Design and Functional Languages*, 2007.

[Wad89] P. Wadler. Theorems for free! In *Functional Programming Languages and Computer Architecture, Proceedings*, pages 347–359. ACM Press, 1989.

XML-Like Grammars and Languages

Matthias Wendlandt
Institut für Informatik
Universität Giessen
Arndtstr. 2

Abstract

XML-like grammars are an abstraction of XML description systems which have the aim to structure the content of XML documents. There are a lot of different types of XML description systems used in practice: DTD, XML schema, TREX, RELAX NG,... They can be translated into grammars which are called XML-like grammars. The XML description systems which are used in practice can be classified in three different grammar types [LC00]: balanced grammars [BB02], XML grammars [BB00] and single-type balanced grammars. The motivation for studying these grammars is twofold: first from the theoretical view, they relate to three other types of grammars which are studied in the past: parenthesis grammars [McN67], bracketed grammars [GH67] and Chomsky-Schützenberger grammars [CS63]. Second there is a practical requirement to study these grammars. The closure properties as well as the decision problems are important for XML software products and the classification in relation to the Chomsky Hierarchy could be important for parsers and parser generators. Besides it could be necessary to search for new XML-like grammars, which have better closure properties or have more decision problems which are decidable or rather are more efficient to parse. The restrained competition balanced grammars [MLM01] and the RegDyck grammars are two new ones.

1 Introduction

XML (e**X**tensible **M**arkup **L**anguage) is a W3C initiative that allows information and services to be encoded with meaningful structure and semantics that computers and humans can understand.
The development of XML 1.0 was leaded by C. M. Sperberg McQueen, John Bosak, Tim Bray and James Clark and finished in 1998.
XML is a markup language, which is based on tags, similar to HTML. It has a tree-like structure. Every tag has a corresponding endtag and consists of text and other tags.
Normally there exists a description system for every XML document. These

description systems are simular to grammars of the Chomsky Hierarchy.
The description systems which are used in practice can be classified in three different grammar types: balanced grammars, XML grammars and single-type balanced grammars. The balanced grammars are the suptype of the other grammars. In addition there are two more types of XML-like grammars. One type is called restrained competition balanced grammars and is based on an idea of Murata, Mani and Lee [MLM01]. This type is close to the other two subtypes of balanced grammars. The other type is called RegDyck grammars and describes the intersection between dyckprimes and regular languages. RegDyck grammars are also a subtype of balanced grammars.
The next section shows the definition of the different types of XML-like grammars. XML grammars and balanced grammars were defined in [BB00] and [BB02]. Based on an idea of [MLM01], we define the other two types. We show some basic results in relation to the Chomsky Hierarchy.
In section three the closure properties and some decision problems are summarized.

2 XML-like grammars

Let A be an alphabet. Then $Reg(A)$ is the set of regular expressions over A.

Definition 2.1
A *balanced grammar* is a 4-tupel $G = (N, T, S, P)$, where N is a finite set of nonterminals, T is a finite set of terminals, with $T = A \cup \overline{A}$, $\overline{A} = \{\overline{x} \mid x \in A\}$ is a disjoint copy of A, $S \in N$ is the start symbol, P is the set of productions which have the form $X \longrightarrow xm\overline{x}$, $m \in Reg(N)$ is called content model. For any pair $(X, x) \in N \times A$ exists exactly one production $X \longrightarrow xm_{(X,x)}\overline{x}$.

There are different subtypes of balanced grammars: XML grammars, single-type balanced grammars, restrained competition balanced grammars and RegDyck grammars. The first three types have the common base that their restrictions in relation to balanced grammars are associated with competing nonterminals and that their content models are an 1-lookahead deterministic regular expression [HW07] [BKW92] [BKW98] over nonterminals.

Definition 2.2
Let $G = (N, T, S, P)$ be a balanced grammar.
Two nonterminals X and Y compete if:
$\exists m, m_0 \in Reg(N), x \in A$:

$$(X \longrightarrow xm\overline{x} \in P \wedge Y \longrightarrow xm_0\overline{x} \in P \wedge X \neq Y)$$

An XML grammar is a balanced grammar if there are competing nonterminals and if there is a bijective relation between nonterminals and terminals.

Definition 2.3
A balanced grammar $G = (N, T, S, P)$ is said to be an *XML grammar* if every production in P has the form:

$$X_a \longrightarrow am\overline{a}, \quad m \in Reg(N), \quad a \in A,$$

and further
$\forall X, Y \in N, m, m_0 \in Reg(N)$:

$$\neg\exists x \in A : (X \longrightarrow xm\overline{x} \in P \wedge Y \longrightarrow xm_0\overline{x} \in P \wedge X \neq Y).$$

Definition 2.4
A balanced grammar $G = (N, T, S, P)$ is said to be a *single-type balanced grammar* if
$\forall X, Y, Z \in N, z, \overline{z} \in T, m, m_0, \alpha, \beta, \gamma \in Reg(N)$:

$$\neg\exists x \in A : (X \longrightarrow xm\overline{x} \in P \wedge Y \longrightarrow xm_0\overline{x} \in P \wedge X \neq Y \wedge Z \longrightarrow z\alpha X\beta Y\gamma\overline{z} \in P).$$

Definition 2.5
A balanced grammar $G = (N, T, S, P)$ is said to be a *restrained competition balanced grammar* if
$\forall \alpha, \beta, \gamma, m, m_0 \in Reg(N), X, Y, Z \in N, z \in A, U, V, W \in N^*$:

$$\neg\exists x \in A : (X \longrightarrow xm\overline{x} \in P \wedge Y \longrightarrow xm_0\overline{x} \in P \wedge X \neq Y \wedge Z \longrightarrow z\alpha X\beta Y\gamma\overline{z} \in P \wedge UXW \in L(\alpha X\beta Y\gamma) \wedge UYV \in L(\alpha X\beta Y\gamma)).$$

The languages of the grammars are called balanced languages, restrained competition balanced languages, single-type balanced languages and XML languages.
Now let us come to our results. There is a proper hierarchy formed by the balanced languages, the restrained competition balanced languages, the single-type balanced languages and the XML languages.
It is straightforward that the balanced languages are a subtype of context-free languages and that they are incomparable with regular languages. Moreover they belong to the deterministic context-free languages. The subtypes restrained competition balanced grammars, single-type balanced grammars and XML grammars belong to the LL(1) grammars.

3 Properties of XML-like grammars

The following table shows the closure properties of XML-like languages. For XML languages and balanced languages intersection, union and complement was investigated in [BB00] and [BB02].

Closure properties		REG	XMLL	STBL	RCBL	BL	DCFL
	Union	+	-	-	-	+	-
	Intersection	+	+	+	+	+	-
	Complement	+	-	-	-	+	+
	Reversal	+	+	+	+	+	+
	ext. Concatenation	+	-	-	+	+	-
	ext. Kleene Closure	+	+	+	+	+	-
	ext. Substitution	+	-	-	-	-	-
	Homomorphism	+	-	-	-	-	-

REG - regular languages, **XMLL** - XML languages, **STBL** - single-type balanced languages, **RCBL** - restrained competition balanced languages **BL** - balanced languages, **DCFL** - deterministic contextfree languages, ext. - extended

Now let us consider some decision problems. As usual it is assumed, that languages are given in an effective way, in general, by a grammar, according to the assumption of the statement.
The inclusion for balanced languages is decidable because equivalence is decidable for deterministic context-free languages and balanced grammars are closed under intersection. Moreover it is decidable whether a balanced grammar is an XML grammar, a single-type balanced grammar or a restrained competition balanced grammar.
The balanced grammars cannot define ambiguous languages because the balanced languages belong to deterministic context-free languages. Nevertheless one can define ambiguous balanced grammars. So parsing can be complicated. It is decidable whether a balanced grammar is ambiguous.

References

[BB00] J. Berstel and L. Boasson. XML grammars. In *Mathematical Foundations of Computer Science, LNCS*, volume 1893, pages 182–191. Springer, 2000.

[BB02] J. Berstel and L. Boasson. Balanced grammars and their languages. In *Formal and Natural Computing, LNCS*, volume 2300, pages 3–25. Springer, 2002.

[BKW92] A. Brüggemann-Klein and D. Wood. Deterministic regular languages. In *9th Annual Symposium on Theoretical Aspects of Computer Science, LNCS*, volume 577, pages 173–184. Springer, 1992.

[BKW98] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.

[CS63] N. Chomsky and M.P. Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 118–161. North-Holland, 1963.

[GH67] S. Ginsburg and M. Harrison. Bracketed context-free languages. *Journal of Computer and System Sciences*, 1:1–23, 1967.

[HW07] Yo-Sub Han and D. Wood. Generalizations of one-deterministic regular languages. In *Conference on Language and Automata Theory and Applications (LATA'07), Universitat Rovira I Virgili*, pages 273–285, 2007.

[LC00] D. Lee and W. W. Chu. Comparative analysis of six XML schema languages. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 29(3):76–87, 2000.

[McN67] R. McNaughton. Paranthesis grammars. *Journal of the ACM*, 14:490–500, 1967.

[MLM01] M. Murata, D. Lee, and M. Mani. Taxonomy of XML schema languages using formal language theory. Technical report, IBM Almaden Research Center, 2001.