20. Theorietag der GI-Fachgruppe AFS "Automaten und Formale Sprachen"

Baunatal bei Kassel

29.9.–1.10.2010

Friedrich Otto, Norbert Hundeshagen, Marcel Vollweiler (Hrsg.)

Vorwort

Die Fachgruppe AFS (früher Fachgruppe 0.1.5) der Gesellschaft für Informatik veranstaltet seit 1991 einmal im Jahr ein Treffen der Fachgruppe im Rahmen eines Theorietags, der traditionell eineinhalb Tage dauert, und auf dem auch die jährliche Fachgruppensitzung durchgeführt wird. Das erste solche Treffen fand 1991 in Magdeburg statt. Die weiteren Theorietage wurden in Kiel (1992), in Dagstuhl (1993), in Herrsching bei München (1994 und 2003), auf Schloß Rauischholzhausen (1995), in Cunnersdorf in der Sächsischen Schweiz (1996), in Barnstorf (1997), in Riveris bei Trier (1998), in Schauenburg-Elmshagen bei Kassel (1999), in Wien (2000 und 2006), in Wendgräben (2001), in der Lutherstadt Wittenberg (2002 und 2009), in Caputh bei Potsdam (2004), in Lauterbad bei Freudenstadt (2005), in Leipzig (2007) und in Wettenberg-Launsbach bei Giessen (2008) ausgerichtet. Seit dem Jahr 1996 wird dem eigentlichen Theorietag noch ein eintägiger Workshop zu speziellen Themen der theoretischen Informatik vorangestellt.

In diesem Jahr wird der Theorietag vom Fachgebiet "Theoretische Informatik" im Fachbereich Elektrotechnik/Informatik der Universität Kassel organisiert. Er findet statt vom 29.9. bis 1.10.2010 in Baunatal bei Kassel. Der Workshop am 29.9. steht in diesem Jahr unter dem allgemeinen Thema "Ausgewählte Themen der Theoretischen Informatik". Als Vortragende konnten

- Carsten Damm aus Göttingen,
- Markus Holzer aus Giessen,
- Peter Leupold aus Kassel,
- Martin Plátek aus Prag und
- Heribert Vollmer aus Hannover

gewonnen werden. Das Programm des eigentlichen Theorietags am 30.9. und 1.10. besteht aus 20 Vorträgen sowie der Sitzung der Fachgruppe AFS. In diesem Band finden sich die Zusammenfassungen aller Vorträge sowohl des Workshops als auch des Theorietags. Desweiteren enthält er das Programm und die Liste aller Teilnehmer.

Wir danken der Gesellschaft für Informatik für die finanzielle Unterstützung dieses Theorietags. Desweiteren danken wir Frau Djawadi ganz herzlich für ihre Hilfe bei der Organisation. Wir wünschen allen Teilnehmern einen interessanten und erfolgreichen Theorietag sowie einen angenehmen Aufenthalt in Baunatal.

Kassel, den 20.9.2010

Friedrich Otto
Norbert Hundeshagen
Marcel Vollweiler

Inhaltsverzeichnis

Programm des Workshops

Mittwoch, den 29.09.2010	
09:30 – 09:50	Anmeldung
09:50 – 10:00	Eröffnung: Friedrich Otto
Sitzung 1	Leitung: Friedrich Otto
10:00 – 11:00	Heribert Vollmer (Hannover): *Complexity of Satisfiability Problems*
11:00 – 11:30	Kaffeepause
Sitzung 2	Leitung: Markus Holzer
11:30 – 12:30	Carsten Damm (Göttingen): *On Applications of Information Theory in Molecular Phylogenetics*
12:30 – 14:00	Mittagspause
Sitzung 3	Leitung: Martin Kutrib
14:00 – 15:00	Markus Holzer (Giessen): *Descriptional Complexity of Regular(-Like) Expressions*
15:15 – 16:15	Martin Plátek (Prag): *On the Prague Group of Mathematical and Algebraic Linguists and Its Formal Tools*
16:15 – 16:45	Kaffeepause
Sitzung 4	Leitung: Bianca Truthe
16:45 – 17:45	Peter Leupold (Kassel): *Computing by Observing (Beobachtersysteme)*
18:30	Abendessen

Programm des Theorietags

Donnerstag, den 30.9.2010	
08:30 – 08:50	Anmeldung
08:50 – 09:00	Eröffnung
Sitzung 1	Leitung: Friedrich Otto
09:00 – 09:30	Rudolf Freund: *Determinism and Reversibility in P Systems with One Membrane*
09:30 – 10:00	Florin Manea: *Algorithmic Results on Hairpin Completion and Lengthening*
10:00 – 10:30	Andreas Malcher: *On Two-Party Watson-Crick Computations with Limited Communication*
10:30 – 11:00	Kaffeepause
Sitzung 2	Leitung: Rudolf Freund
11:00 – 11:30	Henning Bordihn: *Unentscheidbarkeiten und Hierarchien für parallel kommunizierende Automaten*
11:30 – 12:00	Marcel Vollweiler: *Centralized Versus Non-Centralized Parallel Communicating Systems of Restarting Automata*
12:00 – 12:30	Norbert Hundeshagen: *Transductions Computed by PC-Systems of Monotone Deterministic Restarting Automata*
12:30 – 14:00	Mittagspause
Sitzung 3	Leitung: Peter Leupold
14:00 – 14:30	Anna Kasprzik: *Generalizing over Several Learning Settings*
14:30 – 15:00	Dominik Freydenberger: *Inklusionsprobleme für Patternsprachen mit beschränkter Variablenzahl*
15:00 – 15:30	Markus Schmid: *Common Supersequences with Minimum Scope Coincidence Degree*
15:30 – 16:00	Kaffeepause

Sitzung 4	Leitung: Andreas Malcher
16:00 – 16:30	Marian Kogler: *Graph-Controlled Insertion-Deletion Systems*
16:30 – 17:00	Ingmar Meinecke: *Weighted Automata and Regular Expressions on Average and in the Long Run*
17:00 – 17:30	Torsten Stüber: *Monadic Datalog Tree Transducers*
17:30 – 18:00	Suna Bensch: *Algorithmische Eigenschaften von Millstream Systemen*
18:15 – 18:45	Fachgruppensitzung der GI-Fachgruppe AFS
19:00	Abendessen

Freitag, den 1.10.2010	
Sitzung 5	Leitung: Jürgen Dassow
08:45 – 09:00	Jens Doll: *Die Columbo-Architektur*
09:00 – 09:30	Marcus Gelderie: *Classifying Regular Languages via Cascade Products of Automata*
09:30 – 10:00	Stefan Gulan: *Kodierung von Graphen durch reguläre Ausdrücke*
10:00 – 10:30	Sebastian Jacobi: *The Magic Number Problem for Subregular Language Families*
10:30 – 11:00	Kaffeepause
Sitzung 6	Leitung: Henning Bordihn
11:00 – 11:30	Martin Lange: *The Complexity of the Emptiness and Word Problem for Visibly Pushdown Languages*
11:30 – 12:00	Bianca Truthe: *On the Descriptional Complexity of Limited Propagating Lindenmayer Systems*
12:00 – 12:30	Markus Holzer: *The Computational Complexity of RaceTrack*
12:30	Verabschiedung

On Applications of Information Theory in Molecular Phylogenetics

Extended Abstract

Carsten Damm
Institut für Informatik, Universität Göttingen
damm@informatik.uni-goettingen.de

Abstract

Molecular phylogenetics tries to infer evolutionary relationships on base of DNA and other molecular data. The resulting phylogenetic tree hopefully reflects the common ancestry of the species or individuals under consideration. So-called distance based methods first compute distances between the data and then search for a tree that gives a plausible explanation for the distance matrix. While the latter is an algorithmic problem of its own, already the distance measure is an issue. We discuss some of the mainstream approaches and then survey more recent approaches from the literature based on computational and statistical information theory. Finally we introduce an experimental web server implementation of these concepts.

There is strong scientific indication that all living on earth grew out of a single "primordial form" in an evolutionary process ([10, 28]). A phylogenetic tree depicts the resulting phylogenetic relatedness between extinct and extant life forms. Phylogeny construction is an unsupervised learning problem. Research in the field is nowadays mainly based on molecular data like DNA or protein sequences. The art of assembling and managing phylogenetic trees has become a broad and active field of research (sometimes dubbed "phyloinformatics", [9]). We survey only very few of it's many aspects. We shortly summarize character based methods of phylogeny construction and then concentrate on distance based methods.

1 Character based methods

Consider a set S of n *taxa* (e.g., species) whose phylogenetic history is to be estimated on base of assigned discrete string-shaped data items, called *characteristics*. These can be some typical DNA sequences or even vectors indicating presence/absence of certain morphological properties. Following [10] we consider the taxa as being the result of a (probably binary) branching process starting from the "primordial form". Taxa appear as leaves of a tree with unknown inner nodes corresponding to the extinct ancestors. Putting all biology away we state the structure of the problem like this: Given $\mathbf{s} = (s_1, ..., s_n)$, where $s_i \in \Sigma^m$ are characteristics[1] of the taxa, construct a binary tree $T(\mathbf{s})$ with n leaves and s_i assigned to leaf i that "best explains" the data s_i, when interpreting edges as character changing events.

Maximum Parsimony (MP) is applicable to discrete characteristics of any kind. Given some pre-designed $T(\mathbf{s})$ one determines a most parsimonious labeling of inner nodes by hypothetical characters. Hereby the *cost* is the overall number of necessary character changes along edges. Fitch's algorithm ([14]) solves this *small parsimony problem* in time $\mathcal{O}(n \cdot m \cdot k)$, where $k =$ alphabet size. The *big parsimony problem* consists of finding the most parsimonious among all trees and is NP-hard but admits polynomial time approximations ([15, 3]). A weakness of MP is that it almost certainly fails, when the amount of evolutionary change along edges significantly varies in the tree ([13]).

Maximum Likelihood (ML) phylogeny was introduced in [12]. Under certain optimistic technical assumptions it is statistically consistent[2] (see, e.g., [25]). ML is a generalization of MP in that character changes occur with certain probabilities governed by some statistical model of evolution. For a pre-designed $T(\mathbf{s})$ with symmetric probabilities[3] $\mathbf{p}$ assigned to character changes along edges let $L(T, \mathbf{s}, \mathbf{p})$ denote the *likelihood score* = likelihood of observing $\mathbf{s}$ at the leaves. *Small ML* consists of determining a probability vector $\mathbf{p}$ that maximizes $L(T, \mathbf{s}, \mathbf{p})$. *Big ML* consists of finding $T(\mathbf{s})$ and $\mathbf{p}$ maximizing the score over all trees. [7] proved this to be NP-hard. Further, by [6] the problem allows no arbitrary good polynomial time approximation schemes and remains NP-hard even under the assumption of a *molecular clock* (see below). The complexity of *Small ML* is still unknown.

2 Distance based methods

The idea was described in [4]: Given an appropriate distance measure $d(.,.)$ between characters and the matrix M of pairwise distances between taxa in S, construct a tree T_M with leaves set S and edge weights λ such that d equals or at least approximates distance in T_M, induced by λ. Thus T_M gives a plausible explanation for the distances.

The distance measure biologists aim for is *evolutionary distance* d_e. For molecular data

[1] using an alignment with gaps we can assume same-length characteristics

[2] with growing data sets the estimated tree statistically converges to the true one

[3] e.g., an '`X` for `U`' change is as probable as an '`U` for `X`' change

strings x, y this is the number of substitution events that have lead from their common ancestor string z to x and to y. This number is clearly unknown, so we look for a reasonable replacement. The main observation is that d_e is *additive* in the following sense: If edges in the true phylogenetic tree where weighted by evolutionary distance then for all nodes x, y of the tree $d_e(x, y)$ is the weight sum on the path $x \rightsquigarrow y$. More general, given a tree T with positive edge weights λ_e, the induced *additive metric* on its vertices is defined by $d_T(x, y) := \sum_{e \in E(x,y)} \lambda_e$, where $E(x, y)$ is the edge set along the path $x \rightsquigarrow y$. As proved in [5] a metric d on a finite set coincides with the additive metric of a weighted tree if and only if
(1) $\forall x, y, z, t : d(x, y) + d(z, t) \leq \max\{d(x, z) + d(y, t), d(x, t) + d(y, z)\}$ (*four-point condition*).
Moreover, if entries in the $n \times n$ distance matrix M satisfy (1), there is (up to isomorphism) a *unique* weighted tree T_M on n leaves such that d coincides on S with d_{T_M}. T_M can be computed in time $\mathcal{O}(n^2)$. This time bound is also achieved by Neighbor Joining (NJ) introduced by [26].

In an evolution model without insertions and deletions Hamming distance $h(x, y)$ (= number of symbol mismatches) looks like a nice approximation to evolutionary distance between $x, y \in \Sigma^m$. But since repeated changes in one place are not charged h is not additive. One remedy is to transform a given distance measure into an "almost" additive one. Under certain statistical assumptions on the evolution process it can be proved that the *corrected distance transformation* $\lambda(x, y) := -\frac{k-1}{k} \ln(1 - \frac{k}{k-1} h(x, y)/m))$ converges to an additive metric as m increases ([31]). This may also motivate techniques that simply apply NJ to distance matrices obtained from a heuristic approximation to evolutionary distance, even if it is not provably an approximation in the mathematical sense. A second group of heuristics is based on replacing the original distance matrix M by a nearest (e.g., in L_∞-norm) additive distance matrix and reconstruct the tree with that one instead. The exact solution is NP-hard but polynomial time approximable ([11, 1]).

NJ produces non-rooted trees. A widely-used method to find a root consists of identifying an *outgroup* g in S, i.e., a taxon on which by biologic reasoning is less related to the rest of S than any other pair in S. The root is inserted as a splitting node of the edge connecting g to the rest. If we assume that in a statistical sense evolution proceeded at the same pace along every edge (this *molecular-clock assumption* is under dispute), then all leaves (= extant species, i.e., living today) must have the same distance to the common ancestor (= root). Such trees are called *ultrametric* ([16, 17]) and their induced distances are characterised by:
(2) $\forall x, y, z : d(x, y) \leq \max\{d(x, z), d(y, z)\}$, (*three-point condition*—a strengthening of

(1)).
Similar as for additive distances, to any ultrametric distance matrix M there is a unique weighted *rooted* tree T_M, such that d_{T_M} coincides with d on the taxa. Several heuristics construct near-ultrametric trees from matrices that are *not* perfect ultrametric (as is the case for most distance measures in use), e.g., UPGMA [27]. Again, finding the nearest ultrametric tree is NP-hard, while polynomial time approximations are possible [2].

3 Information Distance and Compression Distance

As discussed above, Hamming distance is probably not an appropriate distance notion for string data. Variants of *edit distance* ([19]) are considered better alternatives. The dynamic programming approach to compute the edit distance between $x \in \Sigma^m, y \in \Sigma^n$ requires time and space $\mathcal{O}(nm)$ ([22]). As the space requirement is prohibitive in molecular biology applications, currently some linear space heuristics are in heavy use to compute so-called *alignment scores* by Clustal ([32, 18]). However, by the very nature of edit distance all these methods have difficulties to detect relatedness of strings that differ by large scale substring rearrangements instead of small edits (substitution, insertion, deletion). As a consequence, people seek for more appropriate notions of string distance ([30]).

It is essential to note that reasonable variants of string distance are related to *similarity*: The more similar strings x, y are in a certain sense, the smaller is their distance. By taking similarity as an unspecified but cognitive concept the following approach makes sense: x, y are considered similar (or close to each other) if low "effort" $E(x, y)$ is needed, to compute x given y and vice versa. This idea from [20] can be appropriately formalized by means of *Kolmogorov complexity* (*algorithmic information theory*), as we sketch now.

Let U be a fixed universal Turing machine. $K_U(x)$ denotes the length of the shortest program, such that starting it on U produces x on the empty input. The shortest length changes by at most an additive constant independent of x, when U is replaced by an other universal Turing machine V ([21]). So we simply fix U once and for all and speak of the *Kolmogorov complexity* $K(x)$ as being the length of the shortest program producing x etc. The *relative Kolmogorov complexity* $K(x|y)$ is the length of the shortest program producing x on input y. To any given string z let z^* denote a shortest program for z. Then $d_K(x, y) := \max\{K(x|y^*), K(y|x^*)\}/\max\{K(x), K(y)\}$ is called *normalized information distance* (NID) between x and y. The normalizing denominator was chosen to capture desirable technical properties[4]. This is not necessary, if the data are of similar length. In [20] it is shown that $d(x, y)$ is a *normalized metric "up to a vanishing additive term"*,

[4] e.g.: long strings differing in t positions should be more similar than short strings differing in t positions

i.e. it is between 0 and 1 and satisifies positivity, definiteness and symmetry and the triangle inequality for strings x, y, z is satisfied up to an additive error $\mathcal{O}(1/k)$, where k is the maximum involved Kolmogorov complexity, in this case $\max\{K(x), K(y), K(z)\}$. Moreover, $d(x, y)$ has the following *universality* property: $d(x, y)$ minorizes every "upper semi-computable" normalized distance measure $f(x, y)$ up to a vanishing additive term: $d(x, y) \leq f(x, y) + \mathcal{O}((\log k)/k)$. This can be interpreted as saying that, if x, y are similar in any computable way, they are at least as similar in terms of NID.

Thus, NID is a most natural distance function for strings. The problem is: Neither Kolmogorov complexity itself nor NID is computable. However, up to an additive constant, Kolmogorov complexity of a string x is majorized by the *compression length* $C(x)$ (length of the result of compressing x by some fixed lossless data compressing algorithm). This motivates to replace NID by a corresponding term using C instead of K. [8] suggest the *normalized compression distance* defined by $\mathrm{NCD}_C(x, y) = (\min\{C(xy), C(yx)\} - \min\{C(x), C(y)\})/\max\{C(x), C(y)\}$. Under certain technical assumptions on the compression algorithm (satisfied by standard stream compressors on typical data samples) the resulting NCD falls into the class of "similarity distances" advocated by the authors as well-founded basis for general purpose clustering based phylogeny constructions. This is supported by a large variety of successful experimental results, ranging from genomic data, via music samples to images of handwritten letters etc.

The same idea was in a slightly different and less formal setup also used in [23], where in place of $C(x)$ *Lempel-Ziv-complexity* (LZ) $c(x)$ was used. There are several slightly different definitions for $c(x)$. We define it to be the number N in the unique decomposition $x = x^1 \cdot x^2 \cdot ... \cdot x^N$ into *phrases*, characterized by the following property: For each $i < N$ holds $x_i \notin \{x_1, ..., x_{i-1}\}$ and there is some $j_i, 1 \leq j_i < i$ and some ending symbol $\alpha_i \in \Sigma$, such that $x^i = x^{j_i}\alpha_i$. Hence, the new phrase x^i can be referred to as $\langle j_i, \alpha_i\rangle$. This phrase decomposition is used in dictionary based compressors like `gzip`: Instead of $x = x^1x^2x^3, ...$ the encoder sends $x^1, \langle j_2, \alpha_2\rangle, \langle j_3, \alpha_3\rangle ...$, which in the long run leads to considerable compression. Among others the authors consider $d_c(x, y) := \max\{c(xy) - c(x), c(yx) - c(y)\}/\max\{c(x), c(y)\}$ as a distance measure and use it do derive biologically accepted phylogenies of a selection of mammals based on their mitochondrial genome data.

The approach taken in [29] is related to the latter. Here string distance is not directly measured by using a particular compression algorithm. Using ideas from *statistic information theory* the distance between strings is expressed by an statistical estimation for the compression performance of a dictionary build to optimally compress one string when using it to compress the other. This estimation is expressed in terms of the *average common substring* (ACS) length of two strings x, y: This is the average over all start positions i of the length of the longest match of the substring $x_i x_{i+1}...$ in y. Using the NJ algorithm the authors apply d_{ACS} among others to the same data set as [23] and obtain slightly better results.

4 Implementation and simulation results

To demonstrate the simplicity and at the same time effectiveness of information theory based phylogeny constructors we set up a web server[5] that implements the methods of Otu&Sayood and Ulitsky *et al.* ([23, 29]). The engine to compute LZ and ACS measures is the `gdist` library developed in our group. It uses suffix arrays as the underlying data structure which enables to compute the distance in linear time. The computation of a meaningful phylogeny (using the PHYLIP implementation[6] of NJ) from 20 mitochondrial genomes, each of length $\approx$ 16.000 bases, is a matter of seconds in our webserver environment. For larger requests (essentially longer and/or more sequences) it is more appropriate to use a desktop version which is also available.

As we cannot ask eyewitnesses of evolution, simulation experiments are important to validate phylogenetic tree construction methods. So we set up an "in vitro" version of string evolution based on random singular mutations (substitutions/insertions/deletions) occuring with probability p at every location and measured how well a depth 5 full binary tree can be reconstructed. The quality of reconstruction was measured by a version of the Robinson-Foulds tree distance ([24]). We observed that accuracy

- is satisfying already for string length of about 1000 symbols,
- increases with larger alphabets,
- decreases when p is too small or too large (resulting in random reconstructed tree structures)

[precise statements have to be postponed to the full version]. We did not observe essential differences between various "compression based distances" (CBD), instead we observed a strong correlation between any of the mentioned distance measures to mutation probability p and also to edit distance. By construction CBD should be robust under large scale substrings rearrangements, and indeed we measured this effect in a separate setup. In comparison, as expected, edit distance as computed by Clustal turned out to behave

[5] http://gdist-test.informatik.uni-goettingen.de/ and in a relaunched version with slightly different features http://gdist-test.informatik.uni-goettingen.de/n/

[6] http://evolution.genetics.washington.edu/phylip/general.html

chaotic under such changes. So CBD would detect if genomes are related by simple gene rearrangement, edit distance would probably not.

The latter is becoming more important, as more and more completely sequenced genomes become available. In summarizing, CBD seem adequate to quickly compute phylogenetic trees of string data, but the relevance to biological facts is still under investigation.

5 Acknowledgment

I would like to thank my students Sören Dammert, Jan Brinkmann, Felix Schindler, and Tim Rohlfs for developing the `gdist` library and related software and tests, and especially Tim for lots of simulation runs and inspiring discussions.

References

[1] Richa Agarwala and David F. Baca. A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. *SIAM J. Comput.*, 23(6):1216–1224, 1994.

[2] Richa Agarwala, Vineet Bafna, Martin Farach, Mike Paterson, and Mikkel Thorup. On the approximability of numerical taxonomy (fitting distances by tree metrics). *SIAM J. Comput.*, 28(3):1073–1085, 1999.

[3] Noga Alon, Benny Chor, Fabio Pardi, and Anat Rapoport. Approximate maximum parsimony and ancestral maximum likelihood. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(1):183–187, 2010.

[4] Peter Buneman. The recovery of trees from measures of dissimilarity. In D. G. Kendall and P. Tautu, editors, *Mathematics the the Archeological and Historical Sciences*, pages 387–395. Edinburgh University Press, 1971.

[5] Peter Buneman. A note on the metric properties of trees. *Journal of Combinatorial Theory (Series B)*, 17:48–50, 1974.

[6] Benny Chor and Tamir Tuller. Maximum likelihood of evolutionary trees: hardness and approximation. *Bioinformatics*, 21(suppl_1):i97–106, June 2005.

[7] Benny Chor and Tamir Tuller. Finding a maximum likelihood tree is hard. *Journal of the ACM*, 53:722–744, 2006.

[8] R. Cilibrasi and P. M. B. Vitanyi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, April 2005.

[9] Joel Cracraft. *Assembling the Tree of Life*. Oxford University Press, USA, July 2004.

[10] Charles Darwin. *The Origin of Species*. Gramercy, May 1995.

[11] Martin Farach, S. Kannan, and Tandy Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13:155–179, 1996.

[12] J. Felsenstein. Evolutionary trees from dna sequences: a maximum likelihood approach. *Journal of molecular evolution*, 17(6):368–376, 1981.

[13] Joseph Felsenstein. Cases in which parsimony or compatibility methods will be positively misleading. *Systematic Zoology*, 27:401–410, 1978.

[14] W. M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science (New York, N.Y.)*, 155(760):279–284, January 1967.

[15] L. R. Foulds and R. L. Graham. The steiner problem in phylogeny is np-complete. *Advances in Applied Mathematics*, 3:43–49, 1982.

[16] Felix Hausdorff. Über innere abbildungen. *Fundamentae Mathematicae*, 23:279–291, 1934.

[17] Marc Krasner. Nombres semi-reels et espaces ultrametriques. *Comptes-Rendus de'l Academie des Sciences, Tome II*, 219:433+, 1944.

[18] M. A. Larkin, G. Blackshields, N. P. Brown, R. Chenna, P. A. McGettigan, H. McWilliam, F. Valentin, I. M. Wallace, A. Wilm, R. Lopez, J. D. Thompson, T. J. Gibson, and D. G. Higgins. Clustal w and clustal x version 2.0. *Bioinformatics*, 23(21):2947–2948, November 2007.

[19] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707+, February 1966.

[20] M. Li, X. Chen, X. Li, B. Ma, and P. M. B. Vitanyi. The similarity metric. *IEEE Transactions on Information Theory*, 50(12):3250–3264, December 2004.

[21] Ming Li and Paul M. B. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, Berlin, 1993.

[22] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970.

[23] Hasan H. Otu and Khalid Sayood. A new sequence distance measure for phylogenetic tree construction. *Bioinformatics*, 19(16):2122–2130, November 2003.

[24] D. Robinson. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1-2):131–147, February 1981.

[25] James S. Rogers. Maximum likelihood estimation of phylogenetic trees is consistent when substitution rates vary according to the invariable sites plus gamma distribution. *Journal of Systematic Biology*, 50:713–722, 2001.

[26] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol Biol Evol*, 4(4):406–425, July 1987.

[27] P. H. SNEATH and R. R. SOKAL. Numerical taxonomy. *Nature*, 193:855–860, March 1962.

[28] Douglas L. Theobald. A formal test of the theory of universal common ancestry. *Nature*, 465(7295):219–222, May 2010.

[29] Igor Ulitsky, David Burstein, Tamir Tuller, and Benny Chor. The average common substring approach to phylogenomic reconstruction. *Journal of computational biology : a journal of computational molecular cell biology*, 13(2):336–350, March 2006.

[30] Li-San Wang and Tandy Warnow. Estimating true evolutionary distances between genomes. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 637–646, 2001.

[31] Tandy Warnow. Some combinatorial problems in phylogenetics. In *International Colloquium on Combinatorics and Graph Theory*, volume 7, pages 363–413, Budapest, 1999. Bolyai Society Mathematical Studies.

[32] W. J. Wilbur and D. J. Lipman. Rapid similarity searches of nucleic acid and protein data banks. *Proceedings of the National Academy of Sciences of the United States of America*, 80(3):726–730, February 1983.

The Complexity of Regular(-Like) Expressions[*]

Markus Holzer and Martin Kutrib

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

Regular expressions (RE) were originally introduced in [29] and allow a lovely set-theoretic characterization of languages accepted by deterministic (DFA) or nondeterministic finite automata (NFA). Compared to automata, regular expressions are better suited for human users and therefore are often used as interfaces to specify certain pattern or languages. For example, in the widely available programming environment Unix, regular(-like) expressions can be found in legion of software tools like, e.g., `awk`, `ed`, `emacs`, `egrep`, `lex`, `sed`, `vi`, etc., to mention a few of them. The syntax used to represent them may vary, but the concepts are very much the same everywhere.

Since the regular languages are closed under several more operations, the approach to add operations like intersection ($\cap$), complementation ($\sim$), or squaring (^{2}) does not increase the expressive power of regular expressions. However, the descriptional power, that is, the succinctness of such *regular-like* expressions can be increased. On the other hand, growing succinctness of the expressions can increase the computational complexity of decision problems. This motivates the investigation of regular and regular-like expressions (even with a set of operations capturing subregular language families only). In general $\mathrm{RE}(\Sigma, \varphi)$, where φ is a set of (regularity preserving) operations, denotes all regular(-like) expressions over alphabet Σ using only operations from φ. Hence $\mathrm{RE}(\Sigma, \{\cup, \cdot, {}^*\})$ refers to the set of all *ordinary* regular expressions, which is sometimes also denoted by $\mathrm{RE}(\Sigma)$ or simply RE, while, for example, $\mathrm{RE}(\Sigma, \{\cup, \cdot, \sim\})$ defines the star-free languages.

This gives rise to a tour on the following subjects of problems related to the descriptional and computational complexity of regular-like expressions: first we summarize some important measures on regular expressions [1, 4, 9, 10, 17, 22, 27, 31], since there is no general agreement in the literature about the proper measure. Then we focus on the

[*]This is an abstract of: M. Holzer and M. Kutrib. The complexity of regular(-like) expressions. In Y. Gao, H. Lu, S. Seki, and S. Yu, editors, *Proceedings of the* 14*th International Conference Developments in Language Theory*, number 6224 in LNCS, pages 16–30, London, Ontario, Canada, August 2010. Springer.

descriptional complexity of the conversion of regular expressions to equivalent finite automata [2, 3, 7, 15, 17, 23, 24, 25, 27, 30, 36] and *vice versa* [5, 8, 9, 10, 14, 18, 22, 31, 35], to the computational complexity of problems on regular-like expressions [6, 11, 26, 32, 33, 34, 37, 38] such as, e.g., membership, inequivalence, and non-emptiness of complement, and finally on the operation problem [9, 10, 12, 13, 14, 16, 18, 19, 20, 21, 28, 38] measuring the required size for transforming expressions with additional language operations (built-in or not) into equivalent ordinary regular expressions. Our tour on a fragment of the literature obviously lacks completeness and we give our view of what constitute the most recent interesting links to the considered problem areas.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] C. Allauzen and M. Mohri. A unified construction of the Glushkov, follow, and Antimirov automata. In *Mathematical Foundations of Computer Science (MFCS 2006)*, number 4162 of LNCS, pages 110–121, 2006. Springer.

[3] V. M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoret. Comput. Sci.*, 155:291–319, 1996.

[4] P. Bille and M. Thorup. Regular expression matching with multi-strings and intervals. In *21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1297–1308. SIAM, 2010.

[5] J. A. Brzozowski and E. J. McCluskey. Signal flow graph techniques for sequential circuit state diagrams. *IEEE Trans. Comput.*, C-12(2):67–76, 1963.

[6] J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11:481–494, 1964.

[7] J.-M. Champarnaud, F. Ouardi, and D. Ziadi. Normalized expressions and finite automata. *Int. J. Algebra Comput.*, 17:141–154, 2007.

[8] J. H. Conway. *Regular Algebra and Finite Machines.* Chapman and Hall, 1971.

[9] A. Ehrenfeucht and H. P. Zeiger. Complexity measures for regular expressions. *J. Comput. System Sci.*, 12:134–146, 1976.

[10] K. Ellul, B. Krawetz, J. Shallit, and M.-W. Wang. Regular expressions: New results and open problems. *J. Autom., Lang. Comb.*, 10:407–437, 2005.

[11] M. Fürer. The complexity of the inequivalence problem for regular expressions with intersection. In *International Colloquium on Automata, Languages and Programming (ICALP 1980)*, number 85 of LNCS, pages 234–245, 1980. Springer.

[12] W. Gelade. Succinctness of regular expressions with interleaving, intersection and counting. In *Mathematical Foundations of Computer Science (MFCS 2008)*, number 5162 of LNCS, pages 363–374, 2008. Springer.

[13] W. Gelade. *Foundations of XML: Regular Expressions Revisited.* PhD thesis, School voor Informatietechnologie, University of Hasselt, Belgium, and University of Maastricht, the Netherlands, 2009.

[14] W. Gelade and F. Neven. Succinctness of the complement and intersection of regular expressions. In *Symposium on Theoretical Aspects of Computer Science (STACS 2008)*, volume 08001 of *Dagstuhl Seminar Proceedings*, pages 325–336. IBFI, Schloss Dagstuhl, Germany, 2008.

[15] V. M. Glushkov. The abstract theory of automata. *Russian Math. Surveys*, 16:1–53, 1961.

[16] H. Gruber. *On the Descriptional and Algorithmic Complexity of Regular Languages.* PhD thesis, Institut für Informatik, Universität Giessen, Germany, 2010.

[17] H. Gruber and S. Gulan. Simplifying regular expressions. A quantitative perspective. In *Language and Automata Theory and Applications (LATA 2010)*, number 6031 of LNCS, pages 285–296, 2010. Springer.

[18] H. Gruber and M. Holzer. Finite automata, digraph connectivity, and regular expression size. In *International Colloquium on Automata, Languages and Programming (ICALP 2008)*, number 5126 of LNCS, pages 39–50, 2008. Springer.

[19] H. Gruber and M. Holzer. Provably shorter regular expressions from deterministic finite automata. In *Developments in Language Theory (DLT 2008)*, number 5257 of LNCS, pages 383–395, 2008. Springer.

[20] H. Gruber and M. Holzer. Language operations with regular expressions of polynomial size. *Theoret. Comput. Sci.*, 410:3281–3289, 2009.

[21] H. Gruber and M. Holzer. Tight bounds on the descriptional complexity of regular expressions. In *Developments in Language Theory (DLT 2009)*, number 5583 of LNCS, pages 276–287, 2009. Springer.

[22] H. Gruber and J. Johannsen. Optimal lower bounds on regular expression size using communication complexity. In *Foundations of Software Science and Computational Structures (FoSSaCS 2008)*, number 4962 of LNCS, pages 273–286, 2008. Springer.

[23] S. Gulan and H. Fernau. An optimal construction of finite automata from regular expressions. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2008)*, volume 08002 of *Dagstuhl Seminar Proceedings*, pages 211–222. IBFI, Schloss Dagstuhl, Germany, 2008.

[24] J. Hromkovič, S. Seibert, and T. Wilke. Translating regular expressions into small ϵ-free nondeterministic finite automata. *J. Comput. System Sci.*, 62:565–588, 2001.

[25] J. Hromkovič. Descriptional complexity of finite automata: Concepts and open problems. *J. Autom., Lang. Comb.*, 7:519–531, 2002.

[26] H. B. Hunt III. The equivalence problem for regular expressions with intersections is not polynomial in tape. Technical Report TR 73-161, Department of Computer Science, Cornell University, Ithaca, New York, 1973.

[27] L. Ilie and S. Yu. Follow automata. *Inform. Comput.*, 186(1):140–162, 2003.

[28] P. Kilpeläinen and R. Tuhkanen. Regular expressions with numerical occurrence indicators—Preliminary results. In *Symposium on Programming Languages and Software Tools*, pages 163–173. Department of Computer Science, University of Kuopio, Finland, 2003.

[29] S. C. Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–42. Princeton University Press, Princeton, N.J., 1956.

[30] Y. Lifshits. A lower bound on the size of ϵ-free NFA corresponding to a regular expression. *Inform. Process. Lett.*, 85(6):293–299, 2003.

[31] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IRE Trans. Elect. Comput.*, EC-9:39–47, 1960.

[32] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Symposium on Switching and Automata Theory (SWAT 1972)*, pages 125–129, 1972. IEEE.

[33] H. Petersen. Decision problems for generalized regular expressions. In *Descriptional Complexity of Automata, Grammars and Related Structures (DCAGRS 2000)*, pages 22–29, London, Ontario, Canada. 2000.

[34] H. Petersen. The membership problem for regular expressions with intersection is complete in LOGCFL. In *Symposium on Theoretical Aspects of Computer Science (STACS 2002)*, number 2285 of LNCS, pages 513–522, 2002. Springer.

[35] J. Sakarovitch. The language, the expression, and the (small) automaton. In *Conference on Implementation and Application of Automata (CIAA)*, number 3845 in LNCS, pages 15–30, 2005. Springer.

[36] G. Schnitger. Regular expressions and NFAs without ϵ-transitions. In *Symposium on Theoretical Aspects of Computer Science (STACS 2006)*, number 3884 of LNCS, pages 432–443, 2006. Springer.

[37] L. J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1974.

[38] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Symposium on Theory of Computing (STOC 1973)*, pages 1–9, 1973. ACM Press.

Computing by Observing (Beobachtersysteme)

Peter LEUPOLD
Fachbereich Elektrotechnik/Informatik,
Fachgebiet Theoretische Informatik
Universität Kassel, Germany
`Peter.Leupold@web.de`

Abstract

This is an introduction to the general architecture of Computing by Observing and an overview over the central results about this model of computation.

1 From Experiments to Computations

In recent years, much of the work in Formal Language Theory has stood in some relation to biochemistry. The original motivation for of this were hopes of building actual biocomputers based on the theoretical models that have been developed. So far, however, it has been mainly the theoretical side that has profited from this meeting by the way of numerous new formal mechanisms and models that were inspired by processes observed in nature or in laboratories. A very rich arsenal of devices has been defined. For just a glimpse one may consult for example the book by Păun et al. [6] or the article by Dassow et al. [5].

Nearly all of these models in the area of DNA computing follow the classical computer science paradigm of processing an input directly to an output, which is the result of the computation. Only the mechanisms of processing are different from conventional models; instead of a finite state control or a programming language it is biomolecular mechanisms that are used, or rather abstractions of such mechanisms.

However, in many experiments in biology and chemistry the setup is fundamentally different. The matter of interest is not some product of the system but rather the change observed in certain, selected quantities. To cite two simple examples that might be well-known from High School biology and chemistry classes: In the *predator-prey relationship*, it is not of much use to know the numbers of predators and prey in one single moment. The interesting feature here is how the increase or decrease in one of the two populations affects the other population. A chemical *reaction with a catalyst* often has the same product as without, but the energy curves during the reactions are different. So rather than an output, the interesting result of an experiment is the protocol of one or more quantities over a period of time.

The objective of Computing by Observing was to formalize this approach in a paradigm for computation. The resulting architecture consists of an underlying system, which evolves in discrete steps from one configuration to the next. The second element is an observer, which reads these configurations and transforms them into single letters; a type of classification, if we take the finite number of letters to represent a finite number of classes. In this way, a sequence of configurations is transformed into a simple sequence of symbols, i.e. a string. This corresponds to the protocol of an experiment in biology or chemistry, and for us it is the result of the computation. Figure 1 depicts this setup.

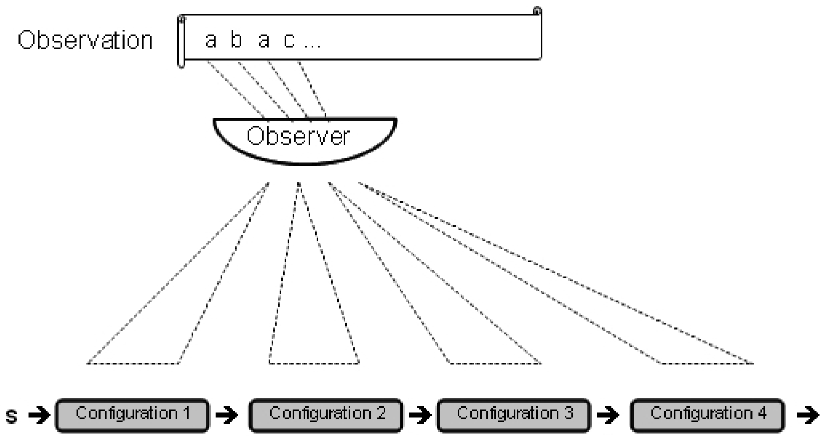


Figure 1: The Computing by Observing architecture.

With the motivation for Computing by Observing in mind, one might think that the most appropriate candidates for the underlying systems are models of biochemical systems, because these are closest to possible applications. And indeed several models of this type have been used, for example Membrane Systems [1].

However, at the current early stage of research, it seems most important to identify key features in the underlying system that make the whole system more powerful. Equally it would be useful to find out if certain features appear to be useless in this context. For such general questions, it seems very appropriate to use a system that is as general and elemental as possible. This is why we use string-rewriting systems as underlying systems here. Their very basic rules work on a very simple data structure, on strings.

2 Computing by Observing

Computing by Observing has been investigated in different variants. Here we will focus on the two main architectures with string-rewriting systems as underlying, evolving systems. On the one hand, there is a variant that generates languages: beginning from a specified starting symbol, a derivation runs until it reaches an irreducible string. Every intermediate string is mapped to one symbol, and the sequence of these symbols is the generated word [2]. Figure 1 depicts exactly this setup.

On the other hand, there are accepting observer systems [3]. The string-rewriting starts on an input word instead of a start symbol. This input is accepted, if the observed string belongs to a fixed regular language.

Technical details on these systems can be found in the referenced articles. Here we only want to give an idea of the power that derives from the interaction between the two components in an informal way. Instead of exact theorems we give these ideas in verbose statements.

Thus far, the standard observers have basically been deterministic regular automata. After reading the current configuration, they output one symbol according to the state that they have reached. These devices are called *monadic transducers*. The component that is varied to obtain observer systems of different power is the underlying string-rewriting system. If these systems contain rules of the form $a \to bc$ that allow the expansion of the working space and to some extent loops in a symbol, then the resulting systems generate or accept all recursively enumerable languages.

Statement 1. *With context-free string-rewriting and regular observers, observer systems are computationally complete in both the generating and the accepting variant.*

Thus the question is how to bound the resources that the systems have. If we only use rules of the form $a \to b$, then obviously no more space than that of the input word can be used. This means that only context-sensitive languages can be accepted. On the other hand, all languages of this type can be accepted, since a Turing Machine can still be simulated, only with a constant space bound.

Statement 2. *With only rules of the form $a \to b$ and regular observers, accepting observer systems have the same computational power as Linear Bounded Automata.*

For generating observer systems it is somewhat more complicated to bound the working space. Rules of the form $a \to b$ are not really an option, because here we start from a single symbol. Thus the entire system would only have a finite number of possible configurations.

A different idea is to oblige the observer to write in every step. Thus even with context-free rules we impose a linear space bound. Even more than that, there is a linear time bound, too. Therefore it is probable though not clear that less than all context-sensitive languages can be generated in this way.

Statement 3. *With context-free string-rewriting and regular observers that always write, generating observer systems only generate context-sensitive languages.*

Numerous further variants have been investigated. We only mention one more way of restraining the observer's power. Reading the entire configuration of unbounded size in every step does not seem to be a realistic feature. Instead, one could possibly detect what is changed, or more exactly: what type of change has occurred to the configuration. In our terms, this means to see which rule has been applied without seeing the current string. This variant has been called *observing change* [4]. Of course, some control over the derivation process is lost, if the position where a rule is applied is not known. However, with inverse context-free rules still all the languages generated by matrix grammars can be accepted.

Observing only change seems to be a restriction that takes Computing by Observing closer to a model that might actually be implementable in reality. Exploring more restrictions of this type and their effects on the computational power is an interesting task for the future.

References

[1] Cavaliere, M., and Leupold, P. Evolution and Observation: A New Way to Look at Membrane Systems. In: *Workshop on Membrane Computing* (2003), C. Martín-Vide, G. Mauri, G. Paun, G. Rozenberg, and A. Salomaa, Eds., vol. 2933 of *Lecture Notes in Computer Science*, Springer, pp. 70–87.

[2] Cavaliere, M., and Leupold, P. Evolution and Observation — A Non-Standard Way to Generate Formal Languages. *Theoretical Computer Science 321* (2004), 233–248.

[3] Cavaliere, M., and Leupold, P. Observation of String-Rewriting Systems. *Fundamenta Informaticae 74*, 4 (2006), 447–462.

[4] Cavaliere, M., and Leupold, P. Computing by Observing Changes. In: *IWNC* (2009), pp. 133–140.

[5] Dassow, J., Mitrana, V., and Salomaa, A. Operations and Language Generating Devices Suggested by the Genome Evolution. *Theoretical Comput. Sci. 270*, 1-2 (2002), 701–738.

[6] Păun, G., Rozenberg, G., and Salomaa, A. *DNA Computing – New Computing Paradigms.* Springer-Verlag, Berlin Heidelberg, 1998.

On the Prague Group of Mathematical and Algebraic Linguistics and Its Formal Tools*

Martin Plátek, Markéta Lopatková
Charles University in Prague,
Czech Republic
martin.platek@mff.cuni.cz

Introduction

Formal modeling of syntactic structure of a natural language, its syntactic analysis as well as synthesis, has an important impact on an insight into the characteristic features of the language and into the needs of its explicit description.

In this talk we focus on the basic tasks and methods developed within the *Functional Generative Description* of Czech (FGD), beginnings of which – connected with the name of Petr Sgall – date back to the 1960s. Both the 'classical' model based on pushdown automata (Section 1) as well as the current model adopting the framework of restarting automata (Section 2) are discussed.

1 The 'classical' model of Functional Generative Description

Functional Generative Description, as proposed by Petr Sgall in [12] and further developed by the *Group of algebraic linguistics* at Charles University in Prague, can be characterized as a *stratificational* and *dependency based* descriptive system for the Czech language.[1] The language description is split into layers, each layer providing a complete description of a (disambiguated) sentence and having its own vocabulary and syntax. Further, it adopts dependency formalism – syntactic information (both at the underlying and surface layers) is captured in a form of dependency trees: words are represented as nodes of

*The paper reports on the research supported by the grants of GAČR No. P202/10/1333 and 405/08/0681. It is carried under the project of MŠMT ČR No. MSM0021620838.

[1]This section is based on [6], which describes the experiments with testing the theoretical adequacy of FGD. The text can be found at the DBLP Bibliography Server.

the respective trees, each node being a complex unit with the lexical, morphological and syntactic features; relations among words are represented by oriented edges.

FGD was designed for *generating correct Czech sentences.* The 'classical' model consists of two components, a *generative component* and a *transducing component.*

The generative component

The generative component generates *tectogrammatical representations* (TR(s) in the sequel), i.e., "underlying representations on the level of linguistic meaning representing a specific patterning of extra-linguistic, ontological content, the set of generated strings surpassing only moderately the set of context-free languages", see [6, 11]. This component was originally based on a context free grammar combined with elements of dependency approach; later it was reformulated using exclusively pushdown store automata [12, 11].

This 'classical' model of FGD does not consider coordination and appositional constructions as these constructions go significantly beyond the straightforward concept of purely dependency-based approaches and require a more general formal model. A possible extended model (still based on pushdown store automaton) was introduced in [7].

The model imposes a significant constraint on a *projectivity* of a dependency tree, which allows for *linearized* representation of a dependency tree. Although this constraint conforms to the theoretical assumption on projectivity of TRs in FGD, it does not allow an adequate description of frequent non-projective surface constructions in Czech [1].

TRs describe all the linguistically relevant semantic information. These representations are disambiguated and identical for all synonymous surface variants. Thus the transducing component captures the relations of *synonymy* and *ambiguity* – it is able to generate all synonymous variants of a sentence from their common TR; on the other hand, an ambiguous sentence has several different TRs (one for each meaning).

The transducing component

The transducing component, which serves for translating the tectogrammatical representations of sentences to the lower layers of the language system, has the form of a sequence of pushdown store automata; the translation is split into steps that more or less correspond to the layers of language system (underlying and surface syntax, morphemics and phonemics/graphemics).

In [6], the model is described as follows: "The mathematical apparatus used for the transduction components of FGD is a sequence of pushdown store automata, transducing the TR into the surface representations (dependency trees) and the latter into morphemic ones (strings); then follows a finite automaton transducing the representation into the graphemic output form. [...]"

"Each transduction of the representation of the structure of the sentence to the adjacent level needs a pair of automata. The conditions constraining the transduction to the next level can be characterized as follows:

(a) In a given step only a single dependency syntagm (the governing word and its modification) is processed [...].

(b) A single pass through the sentence (in the text-to-rule order) is sufficient for every transducer.

(c) The process of transduction is based on the governing unit being handled by every pushdown automaton earlier that its modifications (dependent words). [...]"

The theoretical adequacy of the generative system and its practical usefulness were tested in an experimental implementation – a set of TRs of Czech sentences (mostly grammatically correct, though their meaningfulness could be doubted) were gained as a result of the procedure of random generation at the computer EC 1040, see esp. [6].

In 1980s a new system of translation schemes was designed, which made the interpretation in both directions possible; i.e., it worked as both a *generative* and an *analytical system* [8]. This model was based purely on dependency constructions.

2 The current framework for modeling FGD

Here we present our effort to formalize a basic linguistic method, an *analysis by reduction*, a method based on a stepwise simplification of an analyzed sentence. This method makes it possible to define formal dependency relations between particular sentence members – the (in)dependencies are obtained by correct reductions of Czech sentences – as well as to describe properly the complex word-order variants of a language with a high degree of free word order, including non-projective surface constructions. If we allow for rewriting analysis by reduction is also able to partially capture coordination and appositional constructions [4].

Analysis by reduction provided a crucial motivation for a new formal model of FGD based on the novel concept of *restarting automata* [2, 5]. Restarting automaton is a nondeterministic machine with a finite-state control unit, a finite characteristic vocabulary and a head which can read and process the symbols (words) of the sentence on a flexible working tape, marked by special symbols (the end-markers). This type of automaton starts its computation over an input sentence in the initial state with its head placed on the left end of the tape. A computation of a restarting automaton consists of cycles; the input sentence is processed – according to the transition relation of the automaton – until the sentence is accepted / rejected or until a restart operation is performed. Then the position of the head as well as the inner state of the control unit are 'forgotten' and the automaton starts processing the (already shortened) sentence from the beginning in a new cycle.

Modeling analysis by reduction (and consequently also syntactic analysis) with the use of restarting automata reflects the paradigm of FGD better than earlier models based on pushdown automata:

- Restarting automaton models adequately the syntactic relations determined by valency characteristics of lexical words. It makes it possible to perform several rewrite steps in a single cycle and thus to process a single verb (or noun, adjective or adverb) and all its valency complementations (as e.g., subject and object(s)) in a single computational cycle [4]. Therefore the restarting automaton reflects the *complete valency structures* as it is understood in the concept of dependency valency syntax; that distinguishes this model significantly from the models based on pushdown automata, which model *syntactic pairs* consisting of a governing and a dependent word.

- Restarting automaton makes it possible to capture the concept of *lexicalization* – the approach characteristic for dependency-based language description, which collects essential linguistic information in a lexicon.

- Restarting automaton reflects *non-local behavior of languages with free word order* – rewrite steps in such general models of automata are not restricted to the continuous substring of an input sentence; they can reduce several symbols with distant word-order positions (stored as discontinuous substrings on a working tape). Thus it can process words (and their complementations) with unbounded positions in a sentence as well as words forming non-projective (surface) constructions.

- Restarting automaton working in cycles models *recursive properties of a natural language* appropriately – first, the deepest embedded language constructions are processed, which results in the simplification of an analyzed sentence; then the language constructions embedded in such simplified sentence are processed. After each simplifying operation, a new cycle starts (i.e., the automaton restarts). The computation proceeds until the so-called core predicative structure is reached and accepted without any further restart or until the simplified sentence is rejected as an ill-formed sentence.

- The recent models of FGD based on restarting automata capture explicitly tree structures [9, 10] – restarting automaton is able to assign a set of parallel dependency structures to every reduction of an input sentence; these structures capture the correspondence of dependency trees on different layers of linguistic description, namely the tectogrammatical representation and the surface syntactic representation.

The talk at the TheorieTag will focus on the following issues:

(i) A linguistic background of FGD,

(ii) Formal tools connected with FGD and their adequacy,

(iii) A comparison between FGD and the 'Abhängigkeitsgrammatik' (developed by the group around Jürgen Kunze) [3], and

(iv) Current tasks in formal models of natural languages.

References

[1] Holan, T., Kuboň, V., Oliva, K., Plátek, M.: On Complexity of Word Order. *Les grammaires de dépendance – Traitement automatique des langues (TAL).* Vol. 41, No. 1 (q.ed. Kahane, S.), p. 273-300, 2000.

[2] Jančar, P., Mráz, F., Plátek, M. Vogel, J: On Monotonic Automata with a Restart Operation. *Journal of Automata, Languages and Combinatorics*, Vol. 4, No. 4, p. 287-311, 1999.

[3] Kunze, J.: *Abhängigkeitsgrammatik.* Volume XII of Studia Grammatica, Akademie Verlag, Berlin 1975.

[4] Lopatková, M., Plátek M., Kuboň, V.: Modeling Syntax of Free Word-Order Languages: Dependency Analysis by Reduction. In *Proceedings of TSD 2005*, LNCS 3658, Springer Verlag, Berlin Heidelberg, p. 140-147, 2005.

[5] Otto, F.: Restarting Automata. In *Recent Advances in Formal Languages and Applications, Studies in Computational Intelligence*, Studies in Computational Intelligence, Vol. 25, Springer-Verlag, Berlin Heidelberg, p. 269-303, 2006.

[6] Panevová, J.: Random Generation of Czech Sentences. In *Proceedings of COLING 1982*, p. 295–300, 1982.

[7] Petkevič, V.: A New Formal Specification of Underlying Structure. *Theoretical Linguistics* Vol.21, No.1, 1995.

[8] Plátek, M.: Composition of Translation with D - trees. In *Proceedings of COLING 1982*, p. 313-318, 1982.

[9] Plátek M., Mráz F., and Lopatková M.: Restarting Automata with Structured Output and Functional Generative Description, In *Proceedings of LATA 2010*, LNCS 6031, Springer-Verlag, Berlin Heidelberg, p. 500-511, 2010.

[10] Plátek M., Mráz F., and Lopatková M.: (In)dependencies in Functional Generative Description by Restarting Automata. Accepted for NCMA 2010 in Jena.

[11] Plátek, M., Sgall, P.: A Scale of Context-Sensitive Languages. *Aplication to Natural Language, Information and Control*, Vol. 38, No 1, 1978.

[12] Sgall P., Nebeský L., Goralčíková A., and Hajičová E.: *A Functional Approach to Syntax in Generative Description of Language.* Elsevier Science B.V., Amsterdam New York, 1969.

Complexity of Satisfiability Problems

A Survey of Recent and not so Recent Results

Heribert Vollmer
Leibniz Universität Hannover

Satisfiability problems are ubiquitous in computer science. The satisfiability problem for propositional formulae, SAT, is the historically first and most important NP-complete problem [Coo71, Lev73]. In more practical areas, extensions of propositional logic are used for various tasks: For instance, modal and temporal logics are used for modeling and verification of finite state hardware or software systems [CGP99]. In artificial intelligence, non-monotonic logics are used for the representation of knowledge and modeling of reasoning of intelligent agents [MT97, CS93]. Different dialects of description logics(which are, again, generalizations of propositional modal logic) are used in the semantic web [BCM^{+}03].

Since all these logics are more or less direct generalizations of propositional logic, their corresponding satisfiability problems are NP-hard. Their use in practice, however, makes a search for restrictions of the used propositional languages very interesting that on the one hand are still expressive enough to talk about "real" problems, but on the other hand allow a more efficient satisfiability problem.

Research in this direction started with the seminal paper by Harry Lewis [Lew79]. Fix a finite set of Boolean functions B, and let $\mathrm{SAT}(B)$ denote the satisfiability problems restricted to propositional formulae with connectives from B only. That is, $\mathrm{SAT}(\wedge, \vee)$, for example, is Monotone-SAT. Lewis proved that $\mathrm{SAT}(B)$ is NP-complete, if with connectives from B we can define or "implement" the Boolean function $x \wedge \neg y$, i. e., the negation of implication. An important tool in Lewis' proof is an exploitation of the structure of *Post's lattice*. For this, let $[B]$ denote the minimal set of all Boolean functions that can contains all functions from B, all projections, i. e., all functions I_k^n for $1 \leq k \leq n$ defined by $I_k^n(x_1, \ldots, x_n) = x_k$, and is closed under arbitrary functional composition. The set $[B]$ is called the *clone* generated by B, and B is called a *basis* for $[B]$. Emil Post [Pos41] identified the lattice of all clones and exhibited a finite basis for each of them, see Table 1. A quick discussion of the lattice can be found, e. g., in [Sip05] or [BCRV03]; a detailed account is [Lau06].

Clone	Definition	Base
BF	all Boolean functions	$\{x \wedge y, \neg x\}$
R_0	$\{f \in \mathsf{BF} \mid \mathsf{f}$ is 0-reproducing$\}$	$\{x \wedge y, x \oplus y\}$
R_1	$\{f \in \mathsf{BF} \mid \mathsf{f}$ is 1-reproducing$\}$	$\{x \vee y, x \leftrightarrow y\}$
R_2	$\mathsf{R}_0 \cap \mathsf{R}_1$	$\{x \vee y, x \wedge (y \leftrightarrow z)\}$
M	$\{f \in \mathsf{BF} \mid \mathsf{f}$ is monotone$\}$	$\{x \wedge y, x \vee y, 0, 1\}$
M_0	$\mathsf{M} \cap \mathsf{R}_0$	$\{x \wedge y, x \vee y, 0\}$
M_1	$\mathsf{M} \cap \mathsf{R}_1$	$\{x \wedge y, x \vee y, 1\}$
M_2	$\mathsf{M} \cap \mathsf{R}_2$	$\{x \wedge y, x \vee y\}$
S_0	$\{f \in \mathsf{BF} \mid \mathsf{f}$ is 0-separating$\}$	$\{x \rightarrow y\}$
S_0^n	$\{f \in \mathsf{BF} \mid \mathsf{f}$ is 0-separating of degree $n\}$	$\{x \rightarrow y, \mathrm{dual}(\mathrm{T}_\mathrm{n}^{\mathrm{n}+1})\}$
S_1	$\{f \in \mathsf{BF} \mid \mathsf{f}$ is 1-separating$\}$	$\{x \nrightarrow y\}$
S_1^n	$\{f \in \mathsf{BF} \mid \mathsf{f}$ is 1-separating of degree $n\}$	$\{x \nrightarrow y, \mathrm{T}_\mathrm{n}^{\mathrm{n}+1}\}$
$\mathsf{S}_{02}^\mathsf{n}$	$\mathsf{S}_0^\mathsf{n} \cap \mathsf{R}_2$	$\{x \vee (y \wedge \neg z), \mathrm{dual}(\mathrm{T}_\mathrm{n}^{\mathrm{n}+1})\}$
S_{02}	$\mathsf{S}_0 \cap \mathsf{R}_2$	$\{x \vee (y \wedge \neg z)\}$
$\mathsf{S}_{01}^\mathsf{n}$	$\mathsf{S}_0^\mathsf{n} \cap \mathsf{M}$	$\{\mathrm{dual}(\mathrm{T}_\mathrm{n}^{\mathrm{n}+1}), 1\}$
S_{01}	$\mathsf{S}_0 \cap \mathsf{M}$	$\{x \vee (y \wedge z), 1\}$
$\mathsf{S}_{00}^\mathsf{n}$	$\mathsf{S}_0^\mathsf{n} \cap \mathsf{R}_2 \cap \mathsf{M}$	$\{x \vee (y \wedge z), \mathrm{dual}(\mathrm{T}_\mathrm{n}^{\mathrm{n}+1})\}$
S_{00}	$\mathsf{S}_0 \cap \mathsf{R}_2 \cap \mathsf{M}$	$\{x \vee (y \wedge z)\}$
$\mathsf{S}_{12}^\mathsf{n}$	$\mathsf{S}_1^\mathsf{n} \cap \mathsf{R}_2$	$\{x \wedge (y \vee \neg z), \mathrm{T}_\mathrm{n}^{\mathrm{n}+1}\}$
S_{12}	$\mathsf{S}_1 \cap \mathsf{R}_2$	$\{x \wedge (y \vee \neg z)\}$
$\mathsf{S}_{11}^\mathsf{n}$	$\mathsf{S}_1^\mathsf{n} \cap \mathsf{M}$	$\{\mathrm{T}_\mathrm{n}^{\mathrm{n}+1}, 0\}$
S_{11}	$\mathsf{S}_1 \cap \mathsf{M}$	$\{x \wedge (y \vee z), 0\}$
$\mathsf{S}_{10}^\mathsf{n}$	$\mathsf{S}_1^\mathsf{n} \cap \mathsf{R}_2 \cap \mathsf{M}$	$\{x \wedge (y \vee z), \mathrm{T}_\mathrm{n}^{\mathrm{n}+1}\}$
S_{10}	$\mathsf{S}_1 \cap \mathsf{R}_2 \cap \mathsf{M}$	$\{x \wedge (y \vee z)\}$
D	$\{f \in \mathsf{BF} \mid \mathsf{f}$ is self-dual$\}$	$\{(x \wedge y) \vee (x \wedge \neg z) \vee (\neg y \wedge \neg z)\}$
D_1	$\mathsf{D} \cap \mathsf{R}_2$	$\{(x \wedge y) \vee (x \wedge \neg z) \vee (y \wedge \neg z)\}$
D_2	$\mathsf{D} \cap \mathsf{M}$	$\{(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)\}$
L	$\{f \in \mathsf{BF} \mid \mathsf{f}$ is affine$\}$	$\{x \oplus y, 1\}$
L_0	$\mathsf{L} \cap \mathsf{R}_0$	$\{x \oplus y\}$
L_1	$\mathsf{L} \cap \mathsf{R}_1$	$\{x \leftrightarrow y\}$
L_2	$\mathsf{L} \cap \mathsf{R}_2$	$\{x \oplus y \oplus z\}$
L_3	$\mathsf{L} \cap \mathsf{D}$	$\{x \oplus y \oplus z \oplus 1\}$
E	$\{f \in \mathsf{BF} \mid \mathsf{f}$ is constant or a conjunction$\}$	$\{x \wedge y, 0, 1\}$
E_0	$\mathsf{E} \cap \mathsf{R}_0$	$\{x \wedge y, 0\}$
E_1	$\mathsf{E} \cap \mathsf{R}_1$	$\{x \wedge y, 1\}$
E_2	$\mathsf{E} \cap \mathsf{R}_2$	$\{x \wedge y\}$
V	$\{f \in \mathsf{BF} \mid \mathsf{f}$ is constant or a disjunction$\}$	$\{x \vee y, 0, 1\}$
V_0	$\mathsf{V} \cap \mathsf{R}_0$	$\{x \vee y, 0\}$
V_1	$\mathsf{V} \cap \mathsf{R}_1$	$\{x \vee y, 1\}$
V_2	$\mathsf{V} \cap \mathsf{R}_2$	$\{x \vee y\}$
N	$\{f \in \mathsf{BF} \mid \mathsf{f}$ is essentially unary$\}$	$\{\neg x, 0, 1\}$
N_2	$\mathsf{N} \cap \mathsf{D}$	$\{\neg x\}$
I	$\{f \in \mathsf{BF} \mid \mathsf{f}$ is constant or a projection$\}$	$\{\mathrm{id}, 0, 1\}$
I_0	$\mathsf{I} \cap \mathsf{R}_0$	$\{\mathrm{id}, 0\}$
I_1	$\mathsf{I} \cap \mathsf{R}_1$	$\{\mathrm{id}, 1\}$
I_2	$\mathsf{I} \cap \mathsf{R}_2$	$\{\mathrm{id}\}$

Table 1: List of all clones with definition and bases

Lewis' result can now be rephrased as follows: SAT(B) is NP-complete, if $x \wedge \neg y \in [B]$ (equivalently: $\mathsf{S}_1 \subseteq [\mathsf{B}]$); and SAT($B$) is polynomial-time solvable in all other cases.

Lewis' approach has been taken up by Reith and Wagner [RW05] in the context of problems for Boolean circuits, and by different groups of authors in the context of satisfiability problems or similar algorithmic problems for generalizations of propositional logic, many of them in the area of nonmonotonic logic and artificial intelligence; we append a hopefully up to date list:

- Satisfiability for modal logic [HSS10] and hybrid modal logic [MMS+09, MMS+10]
- Satisfiability for linear temporal logic LTL [BSS+09] and branching time logic CTL and CTL* [MTVM09]
- Model checking for linear temporal logic [BMS+10] and branching time logic [BMM+09]
- Extension/expansion existence problem (a variant of satisfiability) and credulous and skeptical reasoning problems (variants of tautology) for default logic [BMTV09] and autoepistemic logic [CMTV10]
- Satisfiability and inference for propositional circumscription [KK01, Tho09]
- Propositional abduction [CST10] and logic-based argumentation [CSTW10]
- Satisfiability for modal dependence logic [LV10]
- Satisfiability for description logic [MS10]

Finally we would like to mention that many similar results have been obtained for classes of formulas restricted syntactically in a different way, namely for conjunctions of Boolean constraints. This leads into the context of so called (Boolean) *constraint satisfaction problems*. We do not aim to summarize the many results that have been published in the past few years in this research direction but refer the reader instead to [CKS01, CKV08].

References

[BCM+03] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

[BCRV03] E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with Boolean blocks I: Post's lattice with applications to complexity theory. *SIGACT News*, 34(4):38–52, 2003.

[BMM$^+$09] O. Beyersdorff, A. Meier, M. Mundhenk, T. Schneider, M. Thomas, and H. Vollmer. Model checking ctl is almost always inherently sequential. In *16th International Symposium on Temporal Representation and Reasoning*, pages 21–28. IEEE Computer Society Press, 2009.

[BMS$^+$10] M. Bauland, M. Mundhenk, T. Schneider, H. Schnoor, I. Schnoor, and H. Vollmer. The tractability of model-checking for LTL: The good, the bad, and the ugly fragments. *ACM Transactions on Computational Logic*, 12(2):XXX–YYY, 2010.

[BMTV09] O. Beyersdorff, A. Meier, M. Thomas, and H. Vollmer. The complexity of reasoning for fragments of default logic. In *Proc. 12th International Conference on Theory and Applications of Satisfiability Testing*, volume 5584 of *Lecture Notes in Computer Science*, pages 51–64, Berlin Heidelberg New York, 2009. Springer Verlag.

[BSS$^+$09] M. Bauland, T. Schneider, H. Schnoor, I. Schnoor, and H. Vollmer. The complexity of generalized satisfiability for linear temporal logic. *Logical Methods in Computer Science*, 5(1):1–21, 2009.

[CGP99] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.

[CKS01] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems.* Monographs on Discrete Applied Mathematics. SIAM, 2001.

[CKV08] Nadia Creignou, Phokion G. Kolaitis, and Heribert Vollmer, editors. *Complexity of Constraints - An Overview of Current Research Themes*, volume 5250 of *Lecture Notes in Computer Science*, Berlin Heidelberg New York, 2008. Springer.

[CMTV10] N. Creignou, A. Meier, M. Thomas, and H. Vollmer. The complexity of reasoning for fragments of autoepistemic logic. In *Circuits, Logic, and Games*, volume 10061 of *Dagstuhl Seminar Proceedings*, 2010.

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings 3rd Symposium on Theory of Computing*, pages 151–158. ACM Press, 1971.

[CS93] M. Cadoli and M. Schaerf. A survey on complexity results for non-monotonic logics. *Journal of Logic Programming*, 17:127–160, 1993.

[CST10] N. Creignou, J. Schmidt, and M. Thomas. Complexity of propositional abduction for restricted sets of Boolean functions. In *Proc. 12th Principles of Knowledge Representation and Reasoning*. AAAI, 2010.

[CSTW10] N. Creignou, J. Schmidt, M. Thomas, and S. Woltran. Sets of Boolean connectives that make argumentation easier. In *Proc. 12th European Conference on Logics in Artificial Intelligence*, Lecture Notes in Computer Science, Berlin Heidelberg New York, 2010. Springer-Verlag.

[HSS10] E. Hemaspaandra, H. Schnoor, and I. Schnoor. Generalized modal satisfiability. *Journal of Computer and System Sciences*, 76(7):561–578, 2010.

[KK01] L. M. Kirousis and P. Kolaitis. The complexity of minimal satisfiability in Post's lattice. Unpublished notes, 2001.

[Lau06] D. Lau. *Function Algebras on Finite Sets*. Monographs in Mathematics. Springer Verlag, Berlin Heidelberg, 2006.

[Lev73] L. A. Levin. Universal search problems. *Problemi Peredachi Informatsii*, 9(3):115–116, 1973. English translation: *Problems of Information Transmission*, 9(3):265–266.

[Lew79] H. Lewis. Satisfiability problems for propositional calculi. *Mathematical Systems Theory*, 13:45–53, 1979.

[LV10] P. Lohmann and H. Vollmer. Complexity results for modal dependence logic. In *Computer Science Logic*, Lecture Notes in Computer Science, Berlin Heidelberg New York, 2010. Springer-Verlag.

[MMS$^+$09] Arne Meier, Martin Mundhenk, Thomas Schneider, Michael Thomas, Volker Weber, and Felix Weiss. The complexity of satisfiability for fragments of hybrid logic – Part I. In *Proc. 34th International Symposium on Mathematical Foundations of Computer Science*, Springer Lecture Notes in Computer Science, Berlin Heidelberg New York, 2009. Springer Verlag.

[MMS$^+$10] A. Meier, M. Mundhenk, T. Schneider, M. Thomas, and F. Weiss. The complexity of satisfiability for fragments of hybrid logic — Part II. In *Proc. of the International Workshop on Hybrid Logic and Applications (HyLo 2010)*, 2010.

[MS10] A. Meier and T. Schneider. The Complexity of Satisfiability for Sub-Boolean Fragments of $\mathcal{ALC}$. In V. Haarslev, D. Toman, and G. Weddell, editors, *Proc. of the 23rd International Workshop on Description Logics (DL 2010)*, volume 573, pages 279–290, Waterloo, Canada, 2010.

[MT97] V. Marek and M. Truszczyński. Nonmonotonic reasoning – computational perspective. Lecture notes for ESSLLI-97, 1997.

[MTVM09] A. Meier, M. Thomas, H. Vollmer, and M. Mundhenk. The complexity of satisfiability for fragments of CTL and CTL*. *Int. J. Found. Comput. Sci.*, 20(5):901–918, 2009.

[Pos41] E. Post. The two-valued iterative systems of mathematical logic. *Ann. Math. Stud.*, 5:1–122, 1941.

[RW05] S. Reith and K. Wagner. The complexity of problems defined by Boolean circuits. In *Proc. MFI '99*, pages 141–156. World Science Publishing, 2005.

[Sip05] M. Sipser. *Introduction to the Theory of Computation.* Course Technology, 2nd edition, February 2005.

[Tho09] M. Thomas. The complexity of circumscriptive inference in Post's lattice. In *Proc. 10th Logic Programming and Non-Monotonic Reasoning*, volume 5753 of *Lecture Notes in Computer Science*, pages 290–302. Springer Verlag, 2009.

Determinism and Reversibility in P Systems with One Membrane

Artiom Alhazov [1,2], *Rudolf Freund* [3], *and Kenichi Morita* [1]

[1]IEC, Department of Information Engineering
Graduate School of Engineering, Hiroshima University
Higashi-Hiroshima 739-8527 Japan
`morita@iec.hiroshima-u.ac.jp`

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

[3]Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Wien, Austria
`rudi@emcc.at`

Abstract

We study aspects of determinism and reversibility in P systems with one membrane using specific control mechanisms or no control on the rules, and we show (non-)universality for specific variants of P systems working in the maximally parallel or in the sequential transition mode.

1 Introduction

P systems (e.g., see [13], [14]) with one membrane can be seen as a computational model of simple multiset processing. As further ingredients we use singleton promotors and inhibitors as well as priorities for the rules (we refer to [4] for the mechanisms of regulating rewriting). We say that systems are without control if none of these ingredients is used.

If a fixed enumeration of the elements of the alphabet is assumed, then multisets are isomorphic to vectors. In that sense, multiset processing in P systems with only one membrane also corresponds to vector addition systems (see, e.g., [6]). Alternatively, adding and removing symbols can be viewed as incrementing and decrementing counters, i.e., vector addition systems may be viewed as a variant of stateless counter machines, where for every instruction it is specified for each counter which integer is to be added to it. Such a variant is also equivalent to multiset processing systems (in this case, testing for zero corresponds to using inhibitors).

The main goal of this paper is to consider such properties of multiset processing mechanisms as determinism and reversibility as well as their strong versions. Reversibility is an important property of computational systems and has been well studied for circuits of logical elements ([5]), circuits of memory elements ([10]), cellular automata ([11]), Turing machines ([3], [12]), and register machines ([9]). Reversibility essentially is backward determinism. Reversible P systems have been considered in [7], in the energy-based model, simulating Fredkin gates and thus reversible circuits. For the features of (strong) reversibility and (strong) determinism, the case of P systems working in the maximally parallel transition mode was considered in [1], the case with the sequential mode in [2].

2 Definitions

$\mathbb{N}$ is the set of natural numbers (non-negative integers). Given a finite alphabet O, a multiset M over O is a mapping $M : O \to \mathbb{N}$; it may be represented by any string $w \in O^*$ such that $|w|_a = M(a)$ for $a \in O$.

A *(cooperative) multiset rewriting rule* is given by $r : u \to v$, where $u \to O^+$ and $v \to O^*$. This rule can be applied to a multiset w if $|w|_a \geq |u|_a$ for $a \in O$, and the result is w' if $|wv|_a = |w'u|_a$ for every $a \in O$; the sequential transition is written as $w \Longrightarrow_s w'$. In the case of maximal parallelism, a non-extendable set of such multiset rewriting rules is applied in parallel, and we write $w \Longrightarrow_{mp} w'$. If a rule has a promoter a, we write it as $u \to v|_a$. If a rule has an inhibitor a, we write it as $u \to v|_{\neg a}$. The priority relationship is denoted by $>$.

A *P system with one membrane* (a *multiset rewriting system*) – for short we shall simply speak of a *P system* in the following – is a tuple (O, T, w_0, R), where O is the alphabet, T is the terminal (input or output) alphabet, w_0 is the starting multiset, and R is a finite set of multiset rewriting rules. Throughout the rest of the paper, P systems working in the sequential mode and the maximally parallel mode will be called sequential and parallel, respectively. The space $\mathcal{C}$ of configurations (i.e., of multisets over O) essentially is an $|O|$-dimensional space with non-negative integer coordinates. The transition relations $\Longrightarrow_s$ and $\Longrightarrow_{mp}$ each induce an infinite graph on $\mathcal{C}$. The halting configurations (and only these) have out-degree zero. By *reachable* we mean reachable from the initial configuration. The strong variants of the properties *reversibility* and *determinism* are obtained by extending them from reachable configurations to all configurations, i.e., whereas the properties deterministic and reversible refer to the actual computations of the system, the strong variants do not depend on the initial configuration. In case of accepting systems, the initial configurations are obtained by adding arbitrary multisets over a fixed subalphabet to a fixed multiset.

Definition 1. *A P system* Π *is called* ***strongly reversible*** *if every configuration has in-degree at most one;* Π *is called* ***reversible*** *if every reachable configuration has in-degree at most one. We call* Π ***strongly deterministic*** *if every configuration has out-degree at most one;* Π *is called* ***deterministic*** *if every reachable configuration has out-degree at most one.*

A *computation* is a sequence of transitions, starting in the initial configuration, and ending in some halting configuration. The result of a halting computation is the number of terminal objects inside the system when it halts (or the number of input objects when the system starts, in the accepting case). The set $N_g(\Pi)/N_a(\Pi)$ of numbers generated/accepted by a P system Π is the set of natural numbers generated/accepted in any of its computations. The family of number sets generated by reversible P systems with features α and transition mode β is denoted by $N_gROP_1(\alpha,\beta)$, where $\alpha \subseteq \{coo, pro, inh, Pri\}$, with coo, pro, inh, and Pri meaning that we use cooperative rules, promotors, inhibitors, and priorities on the rules, respectively, as well as $\beta \in \{s, mp\}$ denoting the transition mode (sequential, maximally parallel). In the case of accepting systems, we write N_a instead of N_g. For strongly reversible systems, we replace R by R_s. For deterministic (strongly deterministic) systems, we replace R by D (D_s, respectively).

We denote the family of recursively enumerable sets of non-negative integers by NRE, and we call a class of systems generating or accepting it *universal*. $NREG$ is the family of regular sets of non-negative integers. A linear number set of natural numbers is a set S that can be defined by numbers $p_0, p_1, \cdots, p_k$ as $S = \{p_0 + \sum_{i=1}^{k} n_i p_i \mid n_i \geq 0,\ 1 \leq i \leq k\}$. Linear sets are a subclass of $NREG$. We call a class of sets *sublinear* if it is a proper subclass of linear sets.

Proving universality for classes of deterministic and reversible P systems is based on the simulation of deterministic and reversible register machines see [8] and [9].

3 Examples

Example 1. *The parallel P system* $\Pi_0 = (\{a,b\},\{a,b\},a,\{a \to ab\})$ *is strongly reversible (for a preimage, remove as many copies of b as there are copies of a); as no halting configuration is reachable,* $\emptyset \in N_gR_sOP_1(coo,mp)$.

Example 2. *Any sequential P system* $\Pi_1 = (O, O, w_0, \{u \to v\})$ *with only one rule is strongly reversible (to obtain a preimage, remove* v *and add* u*) and also strongly deterministic (there is no choice). If both* w_0 *and* v *contain* u*, then no halting configuration is reachable, otherwise a singleton (*u *or* v*) is generated. Therefore,* $\emptyset$ *as well as* $\{n\}$ *for any* $n \in \mathbb{N}$ *are in* $N_gR_sOP_1(coo, s)$*.*

Example 3. *The sequential P system* $\Pi_2 = (\{a, b, c\}, \{b\}, a, \{a \to ab,\ a \to c\})$ *generates the set of natural numbers since the reachable halting configurations are* cb^**, and it is reversible (for the preimage, replace* c *with* a *or* ab *with* a*), but not strongly reversible (e.g.,* $aab \Longrightarrow cab$ *and* $ca \Longrightarrow cab$*). Hence,* $\mathbb{N} \in N_gROP_1(coo, s)$*.*

Example 4. *Any parallel P system with some erasing rule* $u \to \lambda$ *is not reversible. Any sequential P system containing some erasing rule* $u \to \lambda$ *is not reversible unless all other rules are never applicable.*

Example 5. *Any sequential or parallel P system containing two rules* $x_1 \to y$ *and* $x_2 \to y$ *that may apply at least one of them in some computation is not reversible.*

4 Reversibility

Theorem 1. $N_gROP_1(coo, Pri, \beta) = N_gROP_1(coo, inh, \beta) = NRE$*,* $\beta \in \{s, mp\}$*.*

Corollary 1. *It is undecidable whether a (sequential or parallel) P system from the class of P systems with either inhibitors or priorities is reversible.*

Theorem 2. $N_gR_sOP_1(coo, \beta) = \{\emptyset\} \cup \{\{n\} \mid n \in \mathbb{N}\}$*,* $\beta \in \{s, mp\}$*.*

It is known that (e.g, see [6]) the generative power of sequential P systems equals $PsMAT$, hence, even without requiring additional properties like reversibility we cannot reach universality; it is an open problem to specify the exact generative power of this class.

Corollary 2. *Reversible sequential P systems without priorities and without inhibitors are not universal.*

5 Determinism

The concept of determinism as considered in the area of membrane computing essentially means that such a system, starting from a fixed configuration, has a unique computation; obviously, in this section we only deal with accepting systems.

Theorem 3. $N_aD_sOP_1(coo, Pri, \beta) = N_aDOP_1(coo, mp) = NRE$, $\beta \in \{s, mp\}$.

Corollary 3. *It is undecidable whether a system from the class of (sequential or parallel) P systems with priorities is deterministic.*

Theorem 4. $N_aYOP_1(coo, \beta) = \{\emptyset, \mathbb{N}\} \cup \{\{k \mid 0 \leq k \leq n\} \mid n \in \mathbb{N}\}$, $(Y, \beta) \in \{(D, s), (D_s, s), (D_s, mp)\}$.

Corollary 4. *A sequential P system (without control) is not strongly deterministic unless it has only one rule.*

6 Conclusions

We outlined the concepts of reversibility, strong reversibility, determinism, and strong determinism for sequential and parallel P systems. The results elaborated in [1] and [2] are summed up in Table 1.

Property	pure	Pri	inh	
D(acc)	L	U	U	
D_s(acc)	L	U	?	
R(gen)	N	U	U	
R_s(gen)	L	?	?	

Property	pure	Pri	inh	pro, inh
D(acc)	U	U	U	U
D_s(acc)	L	U	?	U
R(gen)	C	U	U	U
R_s(gen)	L	C	C	C

Table 1: The power of sequential (left) and parallel (right) P systems with different properties, depending on the features. U - universal, N - non-universal, L - sublinear, ? - open, C - conjectured to be non-universal.

References

[1] A. Alhazov, K. Morita: On Reversibility and Determinism in P Systems. Preproc. of *Membrane Computing - 10th International Workshop*, (Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez, Eds.), 129–139.

[2] A. Alhazov, R. Freund, K. Morita: Reversibility and Determinism in Sequential Multiset Rewriting. Proceedings *Unconventional Computation 9th International Conference, UC 2010*, Tokyo, Japan, June 21-25, 2010 (C.S. Calude, M. Hagiya, K. Morita, G. Rozenberg, J. Timmis, Eds.), LNCS 6079, 2010, DOI: 10.1007/978-3-642-13523-1, 21–31.

[3] C.H. Bennett: Logical Reversibility of Computation, *IBM Journal of Research and Development* **17**, 1973, 525–532.

[4] J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory.* EATCS Monographs in Theoretical Computer Science 18, Springer, 1989.

[5] E. Fredkin, T. Toffoli: Conservative Logic, *Int. J. Theoret. Phys.* **21**, 1982, 219–253.

[6] R. Freund, O.H. Ibarra, Gh. Păun, H.-C. Yen: Matrix languages, register machines, vector addition systems. Proc. *Third Brainstorming Week on Membrane Computing*, Sevilla, 2005, 155–168.

[7] A. Leporati, C. Zandron, G. Mauri: Reversible P Systems to Simulate Fredkin Circuits, *Fundam. Inform.* **74**(4), 2006, 529–548.

[8] M.L. Minsky: *Computation: Finite and Infinite Machines.* Prentice-Hall, Englewood Cliffs, NJ, 1967.

[9] K. Morita: Universality of a Reversible Two-Counter Machine, *Theoret. Comput. Sci.* **168** (1996) 303–320.

[10] K. Morita: A Simple Reversible Logic Element and Cellular Automata for Reversible Computing. Proc. *3rd Int. Conf. on Machines, Computations, and Universality*, LNCS **2055**, Springer-Verlag, 2001, 102–113.

[11] K. Morita: Simple Universal One-Dimensional Reversible Cellular Automata, *J. Cellular Automata* **2**, 2007, 159–165.

[12] K. Morita, Y. Yamaguchi: A Universal Reversible Turing Machine. Proc. *5th Int. Conf. on Machines, Computations, and Universality*, LNCS **4664**, Springer-Verlag, 2007, 90–98.

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

[14] P systems webpage. `http://ppage.psystems.eu/`.

Algorithmische Eigenschaften von Millstream Systemen

Suna Bensch, Henrik Björklund, Frank Drewes
Department of Computing Science
Umeå University
90187 Umeå, Sweden
{*suna,henrikb,drewes*}*@cs.umu.se*

Millstream Systeme [2, 3] sind ein formales generisches Modell für die Beschreibung natürlicher Sprache und bieten die Möglichkeit den Zusammenhang zwischen verschiedenen sprachlichen Ebenen wie Phonologie, Morphologie, Syntax und Semantik zu beschreiben.

Die Motivation der Millstream Systeme liegt in modernen linguistischen Theorien, die sich von der Tradition Chomsky distanzieren, in der sprachliche Ebenen als hierarchisch geordnet angesehen werden und wo die syntaktische Ebene die zentrale Rolle innehat. Die Autoren in [4, 5], zum Beispiel, schlagen vor, linguistische Ebenen als autonome Module zu betrachten, die miteinander durch Schnittstellen (Interfaces) verknüpft sind, welche die Interaktion und Interdependenz zwischen verschiedenen sprachlichen Ebenen beschreiben.

Ein Millstream System besteht aus verschiedenen Modulen, die Baumsprachen L_1,..., L_k beschreiben, und einer logischen Schnittstelle, welches die Bäume, die durch die Module erzeugt werden, miteinander verbindet. Eine Konfiguration ist ein Tupel $(t_1, \ldots, t_k) \in L_1 \times \ldots \times L_k$ mit Verknüpfungen (links), die von der Schnittstelle spezifiziert sind.

In diesem Vortrag betrachten wir algorithmische Eigenschaften von Millstream Systemen, die reguläre Baumgrammatiken als Module und Monadische Logik zweiter Stufe (MSO) als Interface-Logik haben [1]. Wir untersuchen das sogenannte Vervollständigungsproblem (completion problem): Gegeben Bäume die von einer Teilmenge der Module erzeugt wurden; können diese in eine Konfiguration des Millstream Systems vervollständigt werden?

References

[1] S. Bensch, H. Björklund, F. Drewes. Algorithmic properties of Millstream systems. In *Proceedings of the 14th Conference on Developments in Language Theory (DLT*

2010), LNCS 6224, pp. 54–65, Springer, 2010.

[2] S. Bensch, F. Drewes. Millstream systems - a formal model for linking language modules by interfaces. In *Proceedings of the ACL 2010 Workshop on Applications of Tree Automata in Natural Language Processing (ATANLP 2010)*, pp. 28-36. The Association for Computational Linguistics, 2010.

[3] S. Bensch, F. Drewes. Millstream systems. Report UMINF 09.21, Umeå University, 2009.

[4] R. Jackendoff. Foundations of Language: Brain, Meaning, Grammar, Evolution. Oxford University Press, Oxford, 2002.

[5] J. Sadock. Autolexical Syntax - A Theory of Parallel Grammatical Representations. The University of Chicago Press, Chicago & London, 1991.

Unentscheidbarkeiten und Hierarchien für parallel kommunizierende endliche Automaten

Henning Bordihn [1], *Martin Kutrib* [2], *and Andreas Malcher* [2]

[1] *Institut für Informatik, Universität Potsdam,*
August-Bebel-Straße 89, 14482 Potsdam, Germany
henning@cs.uni-potsdam.de

[2]*Institut für Informatik, Universität Giessen*
Arndtstraße 2, 35392 Giessen, Germany
{kutrib,malcher}@informatik.uni-giessen.de

Zusammenfassung

Parallel communicating finite automata (PCFAs) are systems of several finite state automata which process a common input string in a parallel way and are able to communicate by sending their states upon request. We consider deterministic and nondeterministic variants and distinguish four working modes. It is known that these systems in the most general mode are as powerful as one-way multi-head finite automata. It is additionally known that the number of heads corresponds to the number of automata in PCFAs in a constructive way. Thus, undecidability results as well as results on the hierarchies induced by the number of heads carry over from multi-head finite automata to PCFAs in the most general mode. Here, we complement these undecidability and hierarchy results also for the remaining working modes. In particular, we show that classical decidability questions are not semi-decidable for any type of PCFAs under consideration. Moreover, it is proven that the number of automata in the system induces infinite hierarchies for deterministic and nondeterministic PCFAs in three working modes.

In ihren Entstehungsjahren stützte sich die Informatik hauptsächlich auf die von-Neumann-Architektur, in der alle Prozesse aus einer sequentiellen Folge von Aktionen bestehen. Dieser Ansatz spiegelte sich in den ursprünglichen Modellen der Automatentheorie wider. Mit der stetig anwachsenden Bedeutung paralleler und verteilter Prozesse setzte auch ein Wandel in der theoretischen Informatik ein. So wurden in [19] endliche Automaten zu Mehrkopfautomaten verallgemeinert, bei denen mehrere Leseköpfe auf einem gemeinsamen Eingabeband die Eingabe lesen und von einer zentralen endlichen Kontrolle gesteuert werden. Endliche Mehrkopfautomaten wurden später u.a. zu Zweiweg- oder

Pushdownautomaten modifiziert [11, 9]. Daneben gab es auch eine Reihe von Arbeiten zu Systemen von Automaten, deren Definitionen von nebenläufigen Programmen oder Netzwerkprotokollen inspiriert waren [3, 4, 7, 8, 13].

Eine ähnliche Entwicklung setzte auch auf dem Gebiet der formalen Grammatiken ein, wo Systeme von Grammatiken in kooperativer Weise eine gemeinsame Sprache erzeugen. Einen Überblick über die verschiedenen Modelle findet sich z.B. in [6]. So besteht etwa ein parallel kommunizierendes Grammatiksystem aus einer festen Anzahl von (regulären oder kontextfreien) Grammatiken, die parallel aber synchronisiert Ableitungsschritte ausführen und auf Anfrage (nach Einführung eines ausgezeichneten Nichtterminals) die Satzform einer der anderen Grammatiken erhalten und in ihre eigene Satzform integrieren. Dieses Konzept, das an das so genannte Classroom-Modell der Künstlichen Intelligenz angelehnt ist, wurde im Jahr 2002 auf endliche Automaten übertragen und führte zum Begriff des parallel kommunizierenden endlichen Automaten (PCFA) [18].

Ein PCFA ist ein System aus einer gewissen Anzahl endlicher Automaten, im allgemeinen nichtdeterministisch mit spontanen Übergängen, die ein und dieselbe Eingabe unabhängig voneinander lesen, wobei ihre Transitionen einer globalen Uhr entsprechend synchronisiert sind. In speziellen Zuständen wird ein Kommunikationsschritt initiiert. Solche Zustände richten eine Anfrage an einen dem Zustand zugeordneten Automaten des Systems, der daraufhin seinen eigenen Zustand an den anfragenden Automaten sendet, mit dem dieser dann weiterrechnet. Hinsichtlich des Nachfolgezustandes der sendenden Komponente werden zwei Betriebsmodi unterschieden. Entweder ändert sich dort der Zustand nicht oder der Automat wird in seinen Anfangszustand zurück gesetzt, nachdem er die Anfrage beantwortet hat. Der erste Betriebsmodus wird "non-returning" genannt, der zweite heißt "returning" und ist vom Classroom-Modell inspiriert, wonach jede Komponente des Systems Teilaufgaben bearbeitet, bis die erreichten Ergebnisse, hier in Form des Zustandes, abgefordert werden und danach mit der Bearbeitung einer neuen Teilaufgabe, hier das Lesen der restlichen Eingabe, von Neuem begonnen wird. Daneben werden noch zentralisierte von nicht zentralisierten Systemen unterschieden, wobei in zentralisierten PCFAs nur ein ausgezeichneter Automat Anfragen an andere Komponenten des Systems richten darf, während in nicht zentralisierten Systemen alle Automaten Anfragezustände annehmen und somit Kommunikationsschritte einleiten dürfen. Somit unterscheiden wir insgesamt vier verschiedene Betriebsmodi, die zudem in Systemen deterministischer oder nichtdeterministischer endlicher Automaten betrachtet werden.

Eines der grundlegenden Resultate aus [18] ist die Äquivalenz von nicht zentralisierten non-returning PCFAs zu endlichen Mehrkopfautomaten, sowohl im deterministischen als auch im nichtdeterministischen Fall. In den Nachfolgearbeiten [5, 1] wurde gezeit, dass diese Äquivalenzen auch für nicht zentralisierte returning PCFAs gelten. In [1] wurden darüber hinaus einige Resultate zur Berechnungsstärke von zentralisierten PCFAs gezeigt. Es gelten die in Abbildung 1 dargestellten Inklusionsbeziehungen.

$$\begin{tikzcd}
 & & \mathrm{DCSL} & & \\
 & & \mathrm{NL} \arrow[u] & & \\
 & & \mathscr{L}(\mathrm{RPCFA}) = \mathscr{L}(\mathrm{PCFA}) \arrow[u] & & \\
\mathscr{L}(\mathrm{RCPCFA}) \arrow[urr, dashed] & & \mathscr{L}(\mathrm{DRPCFA}) = \mathscr{L}(\mathrm{DPCFA}) \arrow[u] & & \mathscr{L}(\mathrm{CPCFA}) \arrow[ull, dashed] \\
\mathscr{L}(\mathrm{DRCPCFA}) \arrow[u, dashed] \arrow[urr, dashed] & & & & \mathscr{L}(\mathrm{DCPCFA}) \arrow[ull] \arrow[u] \\
 & & \mathrm{REG} \arrow[ull] \arrow[urr] & &
\end{tikzcd}$$

Figure 1: Inklusionsdiagramm

Dabei stellen Pfeile Inklusionen dar, wobei Pfeile mit durchgezogenen Linien echte Inklusionen repräsentieren; $\mathscr{L}$(PCFA) ist die Familie der von PCFAs erkennbaren Sprachen; die Zusätze D, C und R bezeichnen die Einschränkung auf deterministische, zentralisierte bzw. returning PCFAs. Diese Ergebnisse werden in diesem Beitrag um zwei Gruppen von Resultaten ergänzt:

1. Die Anzahl der Komponenten, das heißt der Automaten im System, induziert unendliche, echte Hierarchien für PCFAs die nicht zentralisiert oder nicht returning sind.

2. Folgende Entscheidbarkeitsprobleme sind für die Klasse der PCFAs und alle hier betrachteten Unterklassen nicht semi-entscheidbar: Leerheit, Universalität, Inklusion, Äquivalenz, Endlichkeit, Unendlichkeit, Regularität und Kontextfreiheit.

Insbesondere sind die Ergebnisse für die zentralisierten Systeme neu. Erste Unentscheidbarkeitsergebnisse bezüglich Universalität, Inklusion und Äquivalenz für nichtdeterministische zentralisierte non-returning PCFAs aus [17] konnten hinsichtlich des Grades der PCFAs, d.h. bezüglich der Anzahl der Komponenten, verbessert werden. Alle Resultate konnten bereits für Systeme mit zwei Komponenten (mit drei Komponenten im Falle von zentralisierten returning PCFAs) nachgewiesen werden. Hierzu wurden die Entscheidbarkeitsprobleme auf nicht semi-entscheidbare Probleme für OCAs (one-way cellular automata) reduziert.

Es bleibt offen, ob die Anzahl der Komponenten echte, unendliche Hierarchien für zentralisierte returning PCFAs induziert. Eine andere weiterführende Frage ist die nach sinnvollen Teilklassen von PCFAs, für die einige der betrachteten Entscheidbarkeitsfragen entscheidbar sind. Für die zustandslose Variante der eng verwandten endlichen Mehrkopfautomaten wurden in [12] Entscheidbarkeitsresultate nachgewiesen.

Für eine formale Definition von PCFAs und die Beweise verweisen wir auf [2].

References

[1] Bordihn, H., Kutrib, M., Malcher, A.: On the computational capacity of parallel communicating finite automata. Developments in Language Theory (DLT 2008). Volume 5257 of LNCS, Springer (2008) 146–157

[2] Bordihn, H., Kutrib, M., Malcher, A.: Undecidability and hierarchy results for parallel communicating finite automata. Developments in Language Theory (DLT 2010). Volume 6224 of LNCS, Springer (2010) 88–99

[3] Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM **30** (1983) 323–342

[4] Buda, A.: Multiprocessor automata. Inform. Process. Lett. **25** (1987) 257–261

[5] Choudhary, A., Krithivasan, K., Mitrana, V.: Returning and non-returning parallel communicating finite automata are equivalent. RAIRO Inform. Théor. **41** (2007) 137–145

[6] Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, G.: Grammar Systems: A Grammatical Approach to Distribution and Cooperation. Gordon and Breach, Yverdon (1984)

[7] Csuhaj-Varjú, E., Martín-Vide, C., Mitrana, V., Vaszil, G.: Parallel communicating pushdown automata systems. Int. J. Found. Comput. Sci. **11** (2000) 633–650

[8] Ďuriš, P., Jurdziński, T., Kutyłowski, M., Loryś, K.: Power of cooperation and multihead finite systems. International Colloquium on Automata, Languages and Programming (ICALP 1998). Volume 1443 of LNCS, Springer (1998) 896–907

[9] Harrison, M.A., Ibarra, O.H.: Multi-tape and multi-head pushdown automata. Inform. Control **13** (1968) 433–470

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

[11] Ibarra, O.H.: On two-way multihead automata. J. Comput. System Sci. **7** (1973) 28–36

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

[13] Klemm, R.: Systems of communicating finite state machines as a distributed alternative to finite state machines. Phd thesis, Pennsylvania State University (1996)

[14] Kutrib, M.: Cellular automata – a computational point of view. New Developments in Formal Languages and Applications. Springer (2008) 183–227

[15] Kutrib, M.: Cellular automata and language theory. In: Encyclopedia of Complexity and System Science. Springer (2009) 800–823.

[16] Malcher, A.: Descriptional complexity of cellular automata and decidability questions. J. Autom., Lang. Comb. **7** (2002) 549–560

[17] Martín-Vide, C., Mitrana, V.: Some undecidable problems for parallel communicating finite automata systems. Inform. Process. Lett. **77** (2001) 239–245

[18] Martín-Vide, C., Mateescu, A., Mitrana, V.: Parallel finite automata systems communicating by states. Int. J. Found. Comput. Sci. **13** (2002) 733–749

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

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

Inklusionsprobleme für Patternsprachen mit beschränkter Variablenzahl[*]

Joachim Bremer,
Institut für Informatik,
Goethe-Universität,
60054 Frankfurt a.M.

Dominik D. Freydenberger,
Institut für Informatik,
Goethe-Universität,
60054 Frankfurt a.M.

Zusammenfassung

Wir betrachten das Inklusionsproblem für Patternsprachen, die von Pattern mit einer beschränkten Zahl von Variablen erzeugt werden. Diese Arbeit schließt an eine Arbeit von Freydenberger und Reidenbach (Information and Computation 208 (2010)) an und überträgt die dort gewonnenen Unentscheidbarkeitsresultate für allgemeine Patternsprachen auf die Sprachen die von Pattern mit beschränkter Variablenzahl erzeugt werden. Allerdings bewegen sich diese Schranken im vierstelligen Bereich; daher bringen wir außerdem das Inklusionsproblem für stärker beschränkte Klassen von Patternsprachen mit der Collatz-Vermutung in Verbindung. Zusätzlich dazu präsentieren wir den ersten Beweis für die Unentscheidbarkeit der Inklusion für NE-Patternsprachen, der (im Gegensatz zu früheren Beweisen) nicht auf der Nichtentscheidbarkeit der Inklusion für E-Patternsprachen aufbaut.

1 Einleitung

Pattern, d. h. endliche Wörter aus Variablen und Terminalsymbolen, stellen eine kompakte, elegante und natürliche Methode dar, gewisse kontextsensitive Sprachen zu repräsentieren. Ein Pattern erzeugt ein Wort durch eine Substitution, die alle Variablen im Pattern durch beliebige endliche Wörter über einem festen Terminalalphabet ersetzt. Die *Patternsprache* eines Pattern ist somit die Menge aller Wörter, die durch die Substitution des Pattern gebildet werden können; etwas formaler ist eine Patternsprache also die Menge aller Bilder des Pattern unter beliebigen terminalerhaltenden Homomorphismen. Wenn wir beispielsweise das Pattern $\alpha := x_1\,\mathtt{a}\,x_2\,\mathtt{b}\,x_1$ (mit Variablen x_1, x_2 und Terminalsymbolen $\mathtt{a}, \mathtt{b}$) betrachten, so liegen folglich u. a. $w_1 := \mathtt{aabbba}$, $w_2 := \mathtt{abababab}$ und $w_3 := \mathtt{aaabaa}$ in der

[*]Eine ausführlichere Darstellung dieser Arbeit findet sich in [2].

von α erzeugten Patternsprache, wohingegen die Beispielwörter $w_4 := \texttt{ba}$, $w_5 := \texttt{babbba}$ und $w_6 := \texttt{abba}$ nicht von α erzeugt werden können.

In der Literatur werden grundsätzlich zwei verschiedene Arten von Patternsprachen betrachtet: NE-Patternsprachen (eingeführt von Angluin [1]), bei denen die Variablen stets mit nichtleeren Wörtern substituiert werden müssen, und E-Patternsprachen (erstmals untersucht von Shinohara [11]), bei denen auch die Substitution mit leeren Wörtern erlaubt ist. Im obigen Beispiel ist somit das Wort w_3 zwar in der E-Patternsprache, nicht jedoch in der NE-Patternsprache von α enthalten.

Aufgrund ihrer einfachen Definition Patternsprachen besitzen Patternsprachen vielfältige Verbindungen zu anderen Konzepten in der Welt der formalen Sprachen (s. Mateescu und Salomaa [9]). Insbesondere ist das Inklusionsproblem von entscheidender Bedeutung (s. Freydenberger und Reidenbach [3]). Die vorliegende Arbeit untersucht daher dieses Entscheidungsproblem eingehender.

2 Definitionen

Wir bezeichnen das leere Wort als λ. Sei Σ ein Alphabet sogenannter *Terminalsymbole*, und X eine unendliche und zu Σ disjunkte Menge von *Variablen.* Ein Homomorphismus $\sigma : (\Sigma \cup X)^* \to \Sigma^*$ ist eine *Σ-Substitution*, wenn $\sigma(a) = a$ für alle $a \in \Sigma$, und *nichtlöschend*, wenn $\sigma(x) \neq \lambda$ für alle $x \in \Sigma \cup X$. Wir bezeichnen Wörter aus $(\Sigma \cup X)^*$ als *Pattern* und die Menge aller Pattern über Σ als Pat_Σ. Für jedes Pattern $\alpha \in \mathrm{Pat}_\Sigma$ bezeichnet $v(\alpha)$ die Menge der in α vorkommenden Variablen. Ein Pattern $\alpha \in \mathrm{Pat}_\Sigma$ erzeugt die *E-Patternsprache* $L_{\mathrm{E},\Sigma}(\alpha) := \{\sigma(\alpha) \mid \sigma \text{ ist eine } \Sigma\text{-Substitution}\}$ und die *NE-Patternsprache* $L_{\mathrm{NE},\Sigma}(\alpha) := \{\sigma(\alpha) \mid \sigma \text{ ist nichtlöschende } \Sigma\text{-Substitution}\}$.

Die *Collatzfunktion* C ist für alle $n \geq 1$ definiert durch $C(n) := \frac{1}{2}n$ falls n gerade, und $C(n) := 3n + 1$ falls n ungerade. Die Iteration der Collatzfunktion sei definiert durch $C^0(n) := n$ und $C^{i+1}(n) := C(C(n))$ für alle $n \geq 1$ und $i \geq 0$. Es hat sich herausgestellt, dass die Iteration der Collatzfunktion bei allen bisher überprüften Werten in den trivialen Zyklus 1, 4, 2, 1,. . .gerät. Die *Collatzvermutung* besagt, dass zu jedem $n \geq 1$ ein $i \geq 0$ mit $C^i(n) = 1$ existiert (dass also jeder Startwert die Collatziteration in den trivialen Zyklus führt). Diese Vermutung wurde zwar für viele Werte experimentell bestätigt, aber immer noch nicht bewiesen (s. Lagarias [6] und [7]). Stark vereinfacht ausgedrückt besteht außerdem nach Margenstern [8] begründeter Verdacht, dass jede Klasse von Turingmaschinen, die die Iteration der Collatzfunktion simulieren kann, ein nichtentscheidbares Halteproblem hat.

3 Das Inklusionsproblem für Patternsprachen

Laut Jiang, Salomaa, Salomaa und Yu [5] ist das *allgemeine (alphabetunabhängige) Inklusionsproblem für Patternsprachen* unentscheidbar:

Satz 1 (Jiang et al. [5])**.** *Sei* $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$*. Es gibt keine totale berechenbare Funktion* χ_{Z}*, die für jedes Alphabet* Σ *und jedes Paar von Pattern* $\alpha, \beta \in \mathrm{Pat}_\Sigma$ *entscheidet, ob* $L_{\mathrm{Z},\Sigma}(\alpha) \subseteq L_{\mathrm{Z},\Sigma}(\beta)$*.*

Jiang et al. zeigen dies, indem zunächst sie das unentscheidbare Leerheitsproblem für 2-Zählerautomaten (2ZA, s. Ibarra [4]) auf das allgemeine Inklusionproblem für E-Patternsprachen reduzieren. Dies geschieht durch Angabe einer effektiven Konstruktionsvorschrift, die zu einem 2ZA $\mathcal{A}$ ein Alphabet Σ sowie Pattern $\alpha, \beta \in \mathrm{Pat}_\Sigma$ erzeugt, so dass $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$ genau dann, wenn $\mathcal{A}$ bei keiner Angabe anhält. Der Beweis für die Unentscheidbarkeit des Inklusionsproblems für NE-Pattern erfolgt über eine Reduktion des *alphabetspezifischen* Inklusionsproblems für E-Patternsprachen über Alphabeten der Größe n auf das alphabetspezifische Inklusionsproblem für NE-Patternsprachen über Alphabeten der Größe $n + 2$.

Für den von Jiang et al. verwendeten Ansatz ist die Unbeschränktheit von Σ von essentieller Bedeutung – Σ enthält einen Buchstaben für jeden Zustand von $\mathcal{A}$ sowie sechs zusätzliche Sondersymbole. Eine Beschränkung von $|\Sigma|$ würde daher unmittelbar zu einer endlichen Menge von simulierbaren 2ZA und somit zu trivialer Entscheidbarkeit führen. Die meisten Anwendungen verwenden allerdings Klassen von Patternsprachen über festen Alphabeten (s. [3]), so dass Satz 1 dort keine Anwendung finden kann. Dieses Problem wurde schließlich durch das folgende Resultat behoben:

Satz 2 (Freydenberger und Reidenbach [3])**.** *Sei* $\{\mathrm{Z} \in \mathrm{E}, \mathrm{NE}\}$ *und sei* Σ *ein endliches Alphabet mit* $|\Sigma| \geq 2$ *falls* $\mathrm{Z} = \mathrm{E}$*, oder* $|\Sigma| \geq 4$ *falls* $\mathrm{Z} = \mathrm{NE}$*. Dann existiert keine totale berechenbare Funktion* $\chi_{\mathrm{Z},\Sigma}$*, die für jedes Paar von Pattern* $\alpha, \beta \in \mathrm{Pat}_\Sigma$ *entscheidet, ob* $L_{\mathrm{Z},\Sigma}(\alpha) \subseteq L_{\mathrm{Z},\Sigma}(\beta)$*.*

Die Autoren zeigen hierbei die Untentscheidbarkeit alphabetspezifischen Inklusionsproblems für E-Patternsprachen über allen endlichen, nicht-unären Alphabeten durch eine Modifikation des Beweises von Jiang et al., indem sie für jedes (endliche, nicht-unäre) Alphabet Σ eine effektive Konstruktionsvorschrift angeben, die bei Eingabe eines 2ZA $\mathcal{A}$ Pattern $\alpha, \beta \in \mathrm{Pat}_\Sigma$ konstruiert, so dass $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$ genau dann gilt, wenn $\mathcal{A}$ bei keiner Eingabe hält. Hierbei werden sämtliche Zustände von $\mathcal{A}$ unär codiert, während die sechs Sondersymbole der Konstruktion von Jiang et al. binär codiert bzw. durch

Kodierungstricks überflüssig gemacht werden. Dies beweist Satz 2 für den E-Fall, der NE-Fall folgt direkt mittels der Reduktion von Jiang et al.

Allerdings benötigen beide Konstruktionen eine unendliche Zahl von Variablen im Pattern β. Wir betrachten daher die Frage als naheliegend, ob auch Pattern mit einer beschränkten Zahl von Variablen zu unentscheidbaren Inklusionsproblemen führen, und befassen uns daher mit dem *alphabetspezifischen Inklusionsproblem für Pattern mit beschränkter Variablenzahl.* Wie unser erstes Hauptresultat zeigt, existieren obere Schranken, die dennoch zu Unentscheidbarkeit führen:

Satz 3. *Sei Σ ein binäres Alphabet, sei $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$ und seien $m, n \geq 2$. Es existiert kein $\chi_{\mathrm{Z},m,n}$, das bei Eingabe von $\alpha, \beta \in \mathrm{Pat}_\Sigma$ mit $|v(\alpha)| \leq m$, $|v(\beta)| \leq n$ entscheidet, ob $L_{\mathrm{Z},m,n}(\alpha) \subseteq L_{\mathrm{Z},m,n}(\beta)$, falls eine der folgenden Bedingungen erfüllt ist:*

1. $\mathrm{Z} = \mathrm{E}$, $m \geq 2$, $n \geq 2860$; *oder*
2. $\mathrm{Z} = \mathrm{E}$, $m \geq 3$, $n \geq 2854$; *oder*
3. $\mathrm{Z} = \mathrm{NE}$, $m \geq 2$, $n \geq 3549$; *oder*
4. $\mathrm{Z} = \mathrm{NE}$, $m \geq 3$, $n \geq 3541$.

Für beide Fälle von Z basiert der Beweis auf einer Modifikation des Beweises des E-Falls von Satz 2. Anders als die vorhergehenden Beweise verwenden wir allerdings keine Reduktion des Leerheitsproblems von 2ZA, sondern eine Reduktion des Halteproblems der universellen Turingmaschine $U_{2,15}$ von Neary und Woods [10]. Die verschiedenen Startwerte von $U_{2,15}$ werden hierbei im Terminalteil von α codiert, während β bei allen möglichen Eingaben stets das gleiche Pattern ist.

Es ist außerdem besonders hervorzuheben, dass der Beweis des NE-Falls ebenfalls nach diesem Prinzip verfährt und durch mehrere Kodierungstricks sämtliche Probleme umgeht, die daraus resultieren, dass bei NE-Pattern keine Variable gelöscht wird. Im Gegensatz zu den bisherigen Beweisen ist also keine Reduktion des Problems für den E-Fall notwendig, so dass keine zusätzlichen Buchstaben benötigt werden. Dies schließt die Lücke für den Beweis der Nichtentscheidbarkeit des alphabetspezifischen Inklusionsproblems für NE-Patternsprachen über binären und ternären Alphabeten.

Die in Satz 3 angegebenen Schranken sind zwar nicht scharf, lassen sich aber mit den uns bekannten Beweismethoden nicht signifikant senken. Allerdings ist es uns gelungen, durch eine geringfügige Modifikation der Konstruktion zu zeigen, dass sich mit deutlich geringerer Variablenzahl die Iteration der Collatzfunktion in der Inklusion von Patternsprachen simulieren lässt:

Satz 4. *Sei Σ ein binäres Alphabet und sei $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$. Jeder Algorithmus, der bei Eingabe von $\alpha, \beta \in \mathrm{Pat}_\Sigma$ mit*

1. $\mathrm{Z} = \mathrm{E}$, $|v(\alpha)| \leq 2$, $|v(\beta)| \leq 74$*; oder*
2. $\mathrm{Z} = \mathrm{NE}$, $|v(\alpha)| \leq 2$, $|v(\beta)| \leq 145$

entscheidet, ob $L_{\mathrm{Z},\Sigma}(\alpha) \subseteq L_{\mathrm{Z},\Sigma}(\beta)$, lässt sich in einen Algorithmus konvertieren, der für beliebige $N \geq 1$ entscheidet, ob ein $i \geq 0$ existiert, für das $C^i(N) = 1$.

Satz 4 zeigt, dass sich bereits mit einer deutlich kleineren Zahl von Variablen als in Satz 3 komplexe Fragestellungen ausdrücken lassen. Mit einer geringfügigen Modifikation dieses Beweises kann das folgende überraschende Resultat erzielt werden:

Satz 5. *Sei Σ ein binäres Alphabet und sei $\mathrm{Z} \in \{\mathrm{E}, \mathrm{NE}\}$. Jeder Algorithmus, der bei Eingabe von $\alpha, \beta \in \mathrm{Pat}_\Sigma$ mit*

1. $\mathrm{Z} = \mathrm{E}$, $|v(\alpha)| \leq 4$, $|v(\beta)| \leq 80$*; oder*
2. $\mathrm{Z} = \mathrm{NE}$, $|v(\alpha)| \leq 4$, $|v(\beta)| \leq 153$

entscheidet, ob $L_{\mathrm{Z},\Sigma}(\alpha) \subseteq L_{\mathrm{Z},\Sigma}(\beta)$, kann verwendet werden, um zu entscheiden, ob ein $N \geq 1$ existiert, so dass die Folge $C^i(N)$ in einen nichttrivialen Zyklus gerät.

Wäre eines dieser Inklusionsprobleme entscheidbar, so könnte der Algorithmus verwendet werden, um in endlicher Zeit eine Hälfte der schweren und seit langem offenen Collatzvermutung zu beweisen oder die gesamte Vermutung zu widerlegen. Wir betrachten Satz 5 daher als Indiz, dass auch diese Inklusionsprobleme unentscheidbar, oder zumindest nicht effizient entscheidbar, sind. Vergleichbare Resultate konnten auch für alle größeren endlichen Terminalalphabete gewonnen werden.

Literatur

[1] D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.

[2] J. Bremer and D.D. Freydenberger. Inclusion problems for patterns with a bounded number of variables. In *DLT 2010*, LNCS. Accepted.

[3] D.D. Freydenberger and D. Reidenbach. Bad news on decision problems for patterns. *Inform. and Comput.*, 208(1):83–96, 2010.

[4] O. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25:116–133, 1978.

[5] T. Jiang, A. Salomaa, K. Salomaa, and S. Yu. Decision problems for patterns. *J. Comput. Syst. Sci.*, 50:53–63, 1995.

[6] J.C. Lagarias. The 3x+1 problem: An annotated bibliography (1963–1999), Aug 2009. `http://arxiv.org/abs/math/0309224`.

[7] J.C. Lagarias. The 3x+1 problem: An annotated bibliography, II (2000–2009), Aug 2009. `http://arxiv.org/abs/math/0608208`.

[8] M. Margenstern. Frontier between decidability and undecidability: a survey. *Theor. Comput. Sci.*, 231(2):217–251, 2000.

[9] A. Mateescu and A. Salomaa. Patterns. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 4.6, pages 230–242. Springer, 1997.

[10] T. Neary and D. Woods. Four small universal Turing machines. *Fundam. Inform.*, 91(1):123–144, 2009.

[11] T. Shinohara. Polynomial time inference of extended regular pattern languages. In *Proc. RIMS Symposia on Software Science and Engineering, Kyoto*, volume 147 of *Lecture Notes in Computer Science*, pages 115–127, 1982.

Die Columbo-Architektur

Jens Doll
jens.doll@uni-hamburg.de

Zusammenfassung

Columbo ist die Bezeichnung für eine Architektur mit formalsprachlichen Schnittstellen, die sukzessive zu einem Rahmenwerk für fehlerfreie Software ausgebaut werden soll. Im Wesentlichen besteht sie aus Sprachen der Chomsky-Ebenen 1 bis 3 sowie aus zwei Termersetzungssystemen, welche die Korrektheit von While-Programmen berechnen. Es soll der aktuelle Stand der Entwicklung und die evolutionäre Idee des Gesamtkonzeptes dargestellt werden.

Weighted Automata and Regular Expressions on Average and in the Long Run

Manfred Droste and Ingmar Meinecke

Institut für Informatik, Universität Leipzig, D-04109 Leipzig, Germany

{droste,meinecke}@informatik.uni-leipzig.de

Abstract

Quantitative aspects of systems like consumption of resources, output of goods, or reliability can be modeled by weighted automata. Recently, objectives like the average cost or the longtime peak power consumption of a system have been modeled by weighted automata which are not semiring weighted anymore. Instead, operations like limit superior, limit average, or discounting are used to determine the behavior of these automata. Here, we introduce a new class of weight structures subsuming a range of these models as well as semirings. Our main result shows that such weighted automata and Kleene-type regular expressions are expressively equivalent both for finite and infinite words.

Recently, a new kind of weighted automata was established by Chatterjee, Doyen, and Henzinger [3, 4, 5, 6] which compute objectives like the long-run average cost or long-run maximal reward. These models enrich the automata toolbox for the modeling of quantitative aspects of systems which may be the consumption of some resource like time, power, or memory, or the production of some output like goods or pollution. Objectives like average or longtime peaks cannot be modeled by classical semiring weighted automata [1, 7, 16, 17, 21] or lattice automata [18]. Therefore, the theory of semiring weighted automata does not carry over to those new weighted automata.

Finite automata and regular expressions describe the same class of languages [15]. This result by Kleene was transfered by Schützenberger [22] to the semiring-weighted setting over finite words. For infinite words, the respective equivalence for ω-languages was shown by Büchi [2] and for the weighted setting by Ésik and Kuich [13, 14] for semiring-semimodule pairs, by Droste and Kuske [8] for discounting, and by Droste and Vogler [11] for bounded lattices. Here, we will establish that regular weighted expressions are expressively equivalent to the new kind of weighted automata computing average and longtime behavior.

In the weighted automata considered here, the weight of a finite run is calculated in a global way by means of a *valuation function* $\mathrm{val} : D^+ \to D$ where D is the basic set of the weight structure. For finite runs, examples of valuation functions are the average, the supremum, or the last value. As usual, non-determinism is resolved by a monoid operation $+$ (with a neutral element $\not\vdash$). Our automata model has features of classical finite automata and of the weighted automata from [3]. The use of a valuation function is due to [3]. But, moreover, we allow acceptance conditions like final states in the case of finite words and a Büchi condition for non-terminating behavior. The computation of the weight of an infinite run is realized by three ingredients: *(i)* the Büchi condition, *(ii)* a valuation function $\mathrm{val} : D^+ \to D$ for finite sequences, and *(iii)* an *ω-indexed valuation function* $\mathrm{val}^\omega : (\mathbb{N} \times D)^\omega \to D$ for infinite sequences. Hereby, the finite sequences of weights between two consecutive acceptance states are evaluated by the valuation function and then these infinitely many intermediate results are combined by the ω-indexed valuation function where also the lengths of the finite sequences can be taken into account. This procedure guarantees the necessary link between finite and ω-automata in order to establish a Kleene-like result also for infinite words. In this way, we may model for infinite runs the limit superior of the weights, the limit average, i.e., the longrun average weight, or a discounted sum.

Our main results are as follows. For finite words we show that weighted automata and regular weighted expressions are expressively equivalent. The weights are taken from *Cauchy valuation monoids* D which have, besides the sum and the valuation function, a family of products. Such a product is parameterized by two natural numbers representing the length of two words to be concatenated. By these products we can define the Cauchy product and the iteration of functions $S : \Sigma^+ \to D$ mapping a finite word over Σ to a weight from D. Cauchy valuation monoids generalize the valuation functions considered in [3], semirings, and more. Now the class of *regular weighted expressions* over an alphabet Σ and a Cauchy valuation monoid D is given by the grammar

$$E ::= d.a \mid E + E \mid E \cdot E \mid E^+$$

where $d \in D$ and $a \in \Sigma$, cf. [22, 7]. Now we have

Theorem 1. *Let D be a Cauchy valuation monoid and $S : \Sigma^+ \to D$.*

Then S is recognizable by a weighted finite automaton $\mathcal{A}$ if and only if $S = [\![E]\!]$ for some regular weighted expression E.

For infinite words, we present by *Cauchy ω-indexed valuation monoids* also a unified setting for the weight structure. They comprise a complete sum operation, a valuation and an ω-indexed valuation function, as well as a family of parameterized products. Now we have also products where the first parameter is a positive integer but the second one is ω which will be used for the concatenation of a finite and an infinite word. Moreover, we have to impose more restrictions on the interaction of the different operations. This is not surprising because one has to do so also in other settings, cf. [14]. However, instantiations of Cauchy ω-indexed valuation monoids are the structures with limit superior, limit average, or discounting as considered in [3] as well as the complete star-omega-semirings [13, 14] or the semirings used in [10, 20]. Using these parameterized products and the ω-indexed valuation function, we can define a Cauchy product and the ω-iteration and, thus, the semantics $[\![E]\!]$ of *ω-regular weighted expressions* E which are given by the grammar

$$E ::= E + E \mid F \cdot E \mid F^{\omega}$$

where F is any regular weighted expression.

Now we show that over Cauchy ω-indexed valuation monoids, every ω-regular weighted expression can be translated into an equivalent weighted Büchi automaton.

Theorem 2. *Let D be a Cauchy ω-indexed valuation monoid. For any ω-regular weighted expression E over D, the ω-series $[\![E]\!]$ is ω-recognizable by a weighted Büchi automaton $\mathcal{A}$.*

Conversely, under an additional assumption called the *partition property*, which governs the computation of an infinite sequence by using different partitions of the sequence, the behavior of every weighted Büchi automaton can be described by an ω-regular expression.

Theorem 3. *Let D be a Cauchy ω-indexed valuation monoid satisfying the partition property and let $S : \Sigma^{\omega} \to D$ be ω-recognized by a weighted Büchi automaton over D. Then there is an ω-regular weighted expression E with $[\![E]\!] = S$.*

Our setting for infinite words owes some ideas to the case of discounting [8]. The main difference to the setting of Ésik and Kuich [13, 14] is the absence of an infinitary associativity for the ω-indexed valuation function. But we have to drop this property in order to include the new models like limit superior or limit average [3].

Our results were published recently in [9].

References

[1] J. Berstel and C. Reutenauer. *Rational Series and Their Languages.* Springer, 1988.

[2] J. R. Büchi. On a decision method in restricted second order arithmetics. In E. Nagel et al., editors, *Proc. Intern. Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, Stanford, 1962.

[3] K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. In *CSL 2008*, volume 5213 of *LNCS*, pages 385–400. Springer, 2008.

[4] K. Chatterjee, L. Doyen, and T. A. Henzinger. Alternating weighted automata. In *FCT 2009*, volume 5699 of *LNCS*, pages 3–13. Springer, 2009.

[5] K. Chatterjee, L. Doyen, and T. A. Henzinger. Expressiveness and closure properties for quantitative languages. In *24th LICS 2009*, pages 199–208. IEEE Comp. Soc. Press, 2009.

[6] K. Chatterjee, L. Doyen, and T. A. Henzinger. Probabilistic weighted automata. In *CONCUR 2009*, volume 5710 of *LNCS*, pages 244–258. Springer, 2009.

[7] M. Droste, W. Kuich, and H. Vogler, editors. *Handbook of Weighted Automata*. EATCS Monographs in Theoretical Computer Science. Springer, 2009.

[8] M. Droste and D. Kuske. Skew and infinitary formal power series. *Theoretical Computer Science*, 366:199–227, 2006.

[9] M. Droste and I. Meinecke. Regular expressions on average and in the long run. In *CIAA 2010*, LNCS. Springer, 2010.

[10] M. Droste and U. Püschmann. Weighted Büchi automata with order-complete weights. *Int. J. of Algebra and Computation*, 17(2):235–260, 2007.

[11] M. Droste and H. Vogler. Kleene and Büchi theorems for weighted automata and multi-valued logics over arbitrary bounded lattices. In *DLT 2010*, LNCS. Springer, 2010.

[12] S. Eilenberg. *Automata, Languages, and Machines*, volume A. Academic Press, 1974.

[13] Z. Ésik and W. Kuich. A semiring-semimodule generalization of ω-regular languages I+II. *Journal of Automata, Languages and Combinatorics*, 10:203–242 and 243–264, 2005.

[14] Z. Ésik and W. Kuich. Finite automata. In Droste et al. [7], chapter 3.

[15] S. Kleene. Representations of events in nerve nets and finite automata. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–42. Princeton University Press, 1956.

[16] W. Kuich. Semirings and formal power series: their relevance to formal languages and automata. In *Handbook of Formal Languages, vol. 1: Word, Language, Grammar*, chapter 9, pages 609–677. Springer, New York, USA, 1997.

[17] W. Kuich and A. Salomaa. *Semirings, Automata, Languages.* EATCS Monographs in Theoretical Computer Science. Springer, 1986.

[18] O. Kupferman and Y. Lustig. Lattice automata. In *8th VMCAI 2007*, volume 4349 of *LNCS*, pages 199–213. Springer, 2007.

[19] S. Lombardy and J. Sakarovitch. Derivatives of rational expressions with multiplicity. *Theoretical Computer Science*, 332:141–177, 2005.

[20] G. Rahonis. Infinite fuzzy computations. *Fuzzy Sets and Systems*, 153:275–288, 2005.

[21] A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series.* Texts and Monographs in Computer Science. Springer, 1978.

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

Graph-Controlled Insertion-Deletion Systems

Rudolf Freund [1], *Marian Kogler* [1,2], *Yurii Rogozhin* [3], *Sergey Verlan* [4]

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

[2] *Institute of Computer Science,*
Martin Luther University Halle-Wittenberg
Von-Seckendorff-Platz 1, 06120 Halle (Saale), Germany
kogler@informatik.uni-halle.de

[3] *Institute of Mathematics and Computer Science*
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD-2028, Moldova
rogozhin@math.md

[4] *LACL, Département Informatique, UFR Sciences et Technologie*
Université Paris Est, 61, av. Général de Gaulle, 94010 Créteil, France
verlan@univ-paris12.fr

In this talk, we consider the operations of insertion and deletion working in a graph-controlled manner. We show that like in the case of context-free productions, the computational power is strictly increased when using a control graph: computational completeness can be obtained by systems with insertion or deletion rules involving at most two symbols in a contextual or in a context-free manner and with the control graph having only four nodes.

1 Insertion-deletion systems

An *insertion-deletion system* is a construct $ID = (V, T, A, I, D)$, where V is an alphabet; $T \subseteq V$ is the set of *terminal* symbols (in contrast, those of $V - T$ are called *non-terminal* symbols); A is a finite language over V, the strings in A are the *axioms*; I, D are finite sets of triples of the form (u, α, v), where u, α ($\alpha \neq \lambda$), and v are strings over V. The triples in I are *insertion rules*, and those in D are *deletion rules*. An insertion rule

$(u, \alpha, v) \in I$ indicates that the string α can be inserted between u and v, while a deletion rule $(u, \alpha, v) \in D$ indicates that α can be removed from between the context u and v. Stated in another way, $(u, \alpha, v) \in I$ corresponds to the rewriting rule $uv \to u\alpha v$, and $(u, \alpha, v) \in D$ corresponds to the rewriting rule $u\alpha v \to uv$. By $\Longrightarrow_{ins}$ we denote the relation defined by the insertion rules (formally, $x \Longrightarrow_{ins} y$ if and only if $x = x_1uvx_2, y = x_1u\alpha vx_2$, for some $(u, \alpha, v) \in I$ and $x_1, x_2 \in V^*$), and by $\Longrightarrow_{del}$ the relation defined by the deletion rules (formally, $x \Longrightarrow_{del} y$ if and only if $x = x_1u\alpha vx_2, y = x_1uvx_2$, for some $(u, \alpha, v) \in D$ and $x_1, x_2 \in V^*$). By $\Longrightarrow$ we refer to any of the relations $\Longrightarrow_{ins}, \Longrightarrow_{del}$, and by $\Longrightarrow^*$ we denote the reflexive and transitive closure of $\Longrightarrow$.

The language generated by ID is defined by

$$L(ID) = \{w \in T^* \mid x \Longrightarrow^* w \text{ for some x} \in \text{A}\}.$$

The complexity of an insertion-deletion system $ID = (V, T, A, I, D)$ is described by the vector $(n, m, m'; p, q, q')$ called *size*, where

$$\begin{aligned} n &= \max\{|\alpha| \mid (u, \alpha, v) \in I\}, & p &= \max\{|\alpha| \mid (u, \alpha, v) \in D\}, \\ m &= \max\{|u| \mid (u, \alpha, v) \in I\}, & q &= \max\{|u| \mid (u, \alpha, v) \in D\}, \\ m' &= \max\{|v| \mid (u, \alpha, v) \in I\}, & q' &= \max\{|v| \mid (u, \alpha, v) \in D\}. \end{aligned}$$

By $INS_n^{m,m'} DEL_p^{q,q'}$ we denote the families of insertion-deletion systems having the size $(n, m, m'; p, q, q')$.

If one of the parameters n, m, m', p, q, q' is not specified, then instead we write the symbol $*$. If one of numbers from the pairs m, m' and/or q, q' is equal to zero (while the other one is not), then we say that the corresponding families have a one-sided context. Finally we remark that the rules from I and D can be put together into one set of rules R by writing $(u, \alpha, v)_{ins}$ for $(u, \alpha, v) \in I$ and $(u, \alpha, v)_{del}$ for $(u, \alpha, v) \in D$.

1.1 Graph-controlled insertion-deletion systems

Like context-free grammars, insertion-deletion systems may be extended by adding some additional controls. We discuss here the adaptation of the idea of programmed and graph-controlled grammars for insertion-deletion systems.

A *graph-controlled insertion-deletion system with k components* is a construct

$$\Pi = (k, V, T, A, H, i_0, i_f, R) \text{ where}$$

- k is the number of components,
- V, T, A, H are defined as for graph-controlled insertion-deletion systems,

- $i_0 \in [1..k]$ is the initial component,
- $i_f \in [1..k]$ is the final component, and
- R is a finite set of rules of the form $l : (i, r, j)$ where r is an insertion or deletion rule over V and $i, j \in [1..k]$.

The set of rules R may be divided into sets R_i assigned to the *components* $i \in [1..k]$, i.e., $R_i = \{l : (r, j) \mid l : (i, r, j) \in R\}$; in a rule $l : (i, r, j)$, the number j specifies the *target component* where the string is sent from component i after the application of the insertion or deletion rule r. A configuration of Π is represented by a pair (i, w), where i is the number of the *current* component (initially i_0) and w is the current string. We also say that w is *situated* in component i. A transition $(i, w) \Rrightarrow (j, w')$ is performed as follows: first, a rule $l : (r, j)$ from component i (from the set R_i) is chosen in a non-deterministic way, the rule r is applied, and the string is moved to component j; hence, the new set from which the next rule to be applied will be chosen is R_j. More formally, $(i, w) \Rrightarrow (j, w')$ if there is $l : ((u, \alpha, v)_t, j) \in R_i$ such that $w \Longrightarrow_t w'$ by the rule $(u, \alpha, v)_t$; we also write $(i, w) \Rrightarrow_l (j, w')$ in this case. The result of the computation consists of all terminal strings situated in component i_f reachable from the axiom and the initial component, i.e.,

$$L(\Pi) = \{w \in T^* \mid (i_0, w') \Rrightarrow^* (i_f, w) \text{ for some } w' \in A\}.$$

We define the *communication graph* of a graph-controlled insertion-deletion system with k components to be the graph with nodes $1, \ldots, k$ having an edge from node i to node j if and only if there exists a rule $l : ((u, \alpha, v)_t, j) \in R_i$.

By $GCL_k(ins_n^{m,m'}, del_p^{q,q'})$ we denote the family of languages $L(\Pi)$ generated by graph-controlled insertion-deletion systems with at most k components and insertion and deletion rules of size at most $(n, m, m'; p, q, q')$. We replace k by $*$ if k is not fixed. Some results for the families of graph-controlled insertion-deletion systems $TCL_k(ins_n^{m,m'}, del_p^{q,q'})$ can directly be derived from the results presented in [1, 3] for the corresponding families of insertion-deletion P systems $ELSP_k(ins_n^{m,m'}, del_p^{q,q'})$, yet the results we present in the succeeding section either reduce the number of components for systems with an underlying tree structure or else take advantage of the arbitrary structure of the underlying communication graph thus obtaining computational completeness for new restricted variants of insertion and deletion rules.

2 Main results

For all the variants of insertion and deletion rules considered in this section, we know that the basic variants without using control graphs cannot achieve computational completeness (see [1], [2]). The computational completeness results from this section are based on simulations of derivations of a grammar in the special Geffert normal form. These simulations associate a group of insertion and deletion rules to each of the right- or left-linear rules $X \to bY$ and $X \to Yb$. The same holds for (non-context-free) erasing rules $AB \to \lambda$ and $CD \to \lambda$. We remark that during the derivation of a grammar in the special Geffert normal form, any sentential form contains at most one non-terminal symbol from N'.

We start with the following theorem where we even obtain a linear tree structure for the underlying communication graph.

Theorem 1. *$GCL_4(ins_1^{1,0}, del_2^{0,0}) = RE$.*

The next theorem uses one-sided contextual deletion rules.

Theorem 2. *$GCL_4(ins_1^{1,0}, del_1^{1,0}) = RE$.*

The result elaborated above also holds if the contexts for insertion and deletion rules are on different sides.

Theorem 3. *$GCL_4(ins_1^{1,0}, del_1^{0,1}) = RE$.*

Finally, we show a similar result that also holds in the case of context-free insertions.

Theorem 4. *$GCL_4(ins_2^{0,0}, del_1^{1,0}) = RE$.*

3 Conclusions

In this article we have investigated the application of the mechanism of a control graph to the operations of insertion and deletion. We gave a clear definition of the corresponding systems, which is simpler than the one obtained by using P systems. We investigated the case of systems with insertion and deletion rules of size $(1,1,0;1,1,0)$, $(1,1,0;1,0,1)$, $(1,1,0;2,0,0)$ and $(2,0,0;1,1,0)$ and we have shown that the corresponding graph-controlled insertion-deletion systems are computationally complete with only four components, i.e., with the underlying communication graph containing only four nodes. The case of rules of size $(2,0,0;2,0,0)$ is investigated in [1], where it is shown that such systems are not computationally complete.

We suggest two directions for the future research. The first one deals with the number of components needed to achieve computational completeness. The natural question is if it is possible to obtain similar results with only three components. The second direction is inspired from the area of P systems. We propose to further investigate systems where the communication graph has a tree structure as in Theorem 1. The only known results so far are to be found in [1], but there five nodes were used. Hence, the challenge remains to decrease these numbers of components.

References

[1] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. Computational power of insertion-deletion (P) systems with rules of size two. *Natural Computing*, 2010, (in publication), 2010.

[2] A. Matveevici, Y. Rogozhin, and S. Verlan. Insertion-deletion systems with one-sided contexts. In J. O. Durand-Lose and M. Margenstern, editors, *Machines, Computations, and Universality, 5th International Conference, MCU 2007, Orléans, France, September 10-13, 2007, Proceedings*, volume 4664 of *Lecture Notes in Computer Science*, pages 205–217. Springer, 2007.

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

Classifying Regular Languages via Cascade Products of Automata

Marcus Gelderie
RWTH Aachen University
Lehrstuhl für Informatik 7
Ahornstr. 55, 52056 Aachen, Germany
marcus.gelderie@rwth-aachen.de

Abstract

This abstract reports on ongoing work for the diploma thesis [6]. We characterize families of regular languages in terms of the cascade decompositions of the minimal automata accepting those languages. Our basis is the classical work of Krohn and Rhodes [5], Ginzburg [1] and Meyer [4] as well as similar work on the wreath and block product in [3, 2]. In [4] the decomposition results of [5] were used to characterize star-free languages and to give a proof of the celebrated Schützenberger Theorem.

We consider the cases of $\mathcal{R}$-trivial languages (i.e. languages L for which the syntactic monoid $M(L)$ is $\mathcal{R}$-trivial), piecewise testable languages and commutative languages.

Biased Reset Automata

Call a 2-state reset automaton (2-RA) $\mathfrak{R} = (\{0,1\}, \Sigma, \delta)$, $\delta : \Sigma \to Q^Q$, *1-biased* if the set $\Sigma_0 = \{\sigma \in \Sigma | \delta(\sigma) \equiv 0\}$ of all reset inputs inducing the constant 0 function is empty. Similarly, call $\mathfrak{R}$ 0-biased if $\Sigma_1 = \emptyset$. We show (not surprisingly):

Theorem. *A language L is $\mathcal{R}$-trivial iff L is recognized by a cascade product of biased resets.*

We call a cascade product $\mathfrak{A} := \mathfrak{R}_1 \circ \cdots \circ \mathfrak{R}_n$ of 1-biased 2-RAs $\mathfrak{R}_i = (\{0,1\}, \Sigma^i, \delta^{R_i})$ *strictly locally 1-triggered*, if for every $i = 2, \ldots, n$ and every $((x_1, \ldots, x_{i-1}), \sigma) \in \Sigma^i = \{0,1\}^{i-1} \times \Sigma^1$ we have $\delta^{R_i}((x_1, \ldots, x_{i-1}), \sigma) = \mathrm{id}$ if $x_{i-1} = 0$ and $\delta^{R_i}((x_1, \ldots, x_{i-2}, 1), \sigma) = \delta^{R_i}((y_1, \ldots, y_{i-2}, 1), \sigma)$ for all $(y_1, \ldots, y_{i-2}, 1), \sigma), (y_1, \ldots, y_{i-2}, 1), \sigma) \in \Sigma^i$. In other words: every 2-RA in the product $\mathfrak{A}$ reacts only on inputs, which are read when the immediately preceeding 2-RA is in state 1. Furthermore, the state of all other preceeding automata is irrelevant. Call $\mathfrak{A}$ *locally 1-triggered* if $\mathfrak{A}$ is a direct product of strictly locally 1-triggered cascade products. We show:

Theorem. *A language L is piecewise testable iff L is recognized by a locally 1-triggered cascade product of 1-biased 2-RAs.*

It is of course not essential that the resets are 1-biased, since by renaming the states we could just as well make them 0-biased. What is essential, however, is that if the resets are i-biased, the product is locally i-triggered.

Call a semiautomaton *1-semilinear*, if there exists exactly one input $\sigma \in \Sigma$, such that $\delta(\sigma) \neq \mathrm{id}$. 1-semilinear automata recognize commutative languages, which are defined by the number of occurrences of a single letter. We call a grouplike automaton $\mathfrak{G} = (Q^G, \Sigma, \delta^G)$ with associated group $G \cong M(\mathfrak{G})$ *cyclic* if G is cyclic. Furthermore, we call $|Q^G|$ the *order* of $\mathfrak{G}$. We show:

Theorem. *A language L is commutative iff L is recognized by a direct product of the form:*

$$\left(\bigtimes_{i=1}^{n} \mathfrak{R}_1^i \circ \cdots \circ \mathfrak{R}_{n_i}^i \right) \times \left(\bigtimes_{i=1}^{m} \mathfrak{G}_i \right)$$

where the $\mathfrak{R}_j^i$ are 1-semilinear biased 2-RAs and the $\mathfrak{G}_i$ are 1-semilinear cyclic grouplike automata. The 1-semilinear cyclic grouplike automata can in turn be decomposed into direct products of cascade products of cyclic grouplike automata of prime order.

The Scope of Cascade Products

We introduce the notion of the *scope* of a cascade product. Consider the cascade product $\mathfrak{A} := \mathfrak{A}_1 \circ \cdots \circ \mathfrak{A}_n$, where $\mathfrak{A}_1 = (Q_1, \Sigma, \delta_1)$ and $\mathfrak{A}_i = (Q_i, Q_1 \times \cdots \times Q_{i-1} \times \Sigma, \delta_i)$ for $i = 2, \ldots, n$. We say $\mathfrak{A}$ has *scope* $k \in \{1, \ldots, n\}$, if for $i \geq k$:

$$\delta_{i+1}((q_1, \ldots, q_{i-k}, q_{i-k+1}, \ldots, q_i), \sigma) = \delta_{i+1}((q'_1, \ldots, q'_{i-k}, q_{i-k+1}, \ldots, q_i), \sigma)$$

for all $(q_1, \ldots, q_i) \in Q_1 \times \cdots \times Q_i$, $(q'_1, \ldots, q'_{i-k}) \in Q_1 \times \cdots \times Q_{i-k}$ and all $\sigma \in \Sigma$. In other words: every automaton in the cascade is sensitive only to the states of the first k automata immediately preceeding it.

The scope of a cascade product seems interesting because it describes a certain locality in the cascade product. The "pace" by which information spreads through the cascade is bounded, which makes the product easier to handle in proofs. Furthermore, the "fan-in" of a given automaton is bounded by a constant.

The scope of a cascade product occurs naturally in some settings. Let $\Sigma = \{a, b\}$ and define

$$L_n = \{w \in \Sigma^* | \forall x \sqsubseteq w : 0 \leq |x|_a - |x|_b \leq n \wedge |w|_a = |w|_b\}$$

It can be shown that $L_n \in \mathcal{V}_n \setminus \mathcal{V}_{n-1}$ for all $n \geq 1$, where $\mathcal{V}_n$ is the n-th level of the Straubing Hierarchy. We prove:

Theorem. *Every cascade product of 2-RAs recognizing L_n has at least $n + 1$ factors.*

A very natural way to implement such a product with the minimal number of factors is to use n resets to count the excess in a's over b's, $\Delta(w) = |w|_a - |w|_b$, and one additional, biased 2-RA to permanently memorize if $\Delta(u) < 0$ or $\Delta(u) > n$ for some prefix u of w. We prove for all $0 < n \in \mathbb{N}$:

Theorem. *The language L_n can be recognized by a scope n cascade product of $n+1$ 2-RAs.*

Spending more 2-RAs, however, one can sometimes reduce the scope. We show:

Theorem. *All $\mathcal{R}$-trivial languages are recognized by a scope 2 cascade product of 2-RAs.*

This raises the question of how the number of factors and the scope of a cascade product are related. Presently, we study this tradeoff between the scope and the number of factors.

References

[1] A. Ginzburg. *Algebraic Theory of Automata.* Academic Press, New York, 1068.

[2] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity.* Birkhauser Verlag, Basel, Switzerland, 1994.

[3] H. Straubing. The Wreath Product and its Applications. In *Proceedings of the LITP Spring School on Theoretical Computer Science*, pages 15-24, London, UK, 1989. Springer Verlag.

[4] A. R. Meyer. A Note on Star-Free Events. *J. ACM*, 16(2):220–225, 1969.

[5] K. Krohn and J. Rhodes. Algebraic Theory of Machines. I. Prime Decomposition Theorem for Finite Semigroups and Machines. *Trans. Amer. Math. Soc*, 116:450464, 1965.

[6] M. Gelderie. *Classifying Regular Languages via Cascade Products of Automata.* Diploma Thesis. In preparation.

Kodierung von Graphen durch reguläre Ausdrücke

Stefan Gulan
gulan@informatik.uni-trier.de

Zusammenfassung

Die Klasse $\mathcal{SPS}$ erweitert die der gerichteten seriell-parallelen Graphen durch das Hinzufügen von Schleifen bzw. Kreisen. Elemente aus $\mathcal{SPS}$ lassen sich eindeutig umkehrbar auf reguläre Ausdrücke abbilden. Die verwendeten Konstruktionen werden um Kantenmarkierungen erweitert, was zu einer Kodierung endlicher Automaten mit derartiger Struktur durch sprachäquivalente reguläre Ausdrücke führt. Dabei verhalten sich die Größen von Automat und Ausdruck linear zueinander. Zuletzt wird eine Charakterisierung von $\mathcal{SPS}$ durch verbotene Teilgraphen vorgestellt.

1 Notation

Ein (gerichteter) *Graph* ist ein Tupel (V, A, h, t) mit Knotenmenge V, Kantenmenge A und Abbildungen $t : A \to V$ und $h : A \to V$. Zu einer Kante a ist $t(a)$ der *Schwanz* und $h(a)$ der *Kopf*, eine xy-Kante hat Schwanz x und Kopf y. Eine x-*Schleife* ist eine xx-Kante. Der *Eingangsgrad* eines Knotens x im Graphen G, wird $\mathrm{d}_G^-(y)$, der *Ausgangsgrad* $\mathrm{d}_G^+(x)$ notiert. Ein Graph ist *2-terminal* wenn er zwei verschiedene Knoten s, die *Quelle*, und t, die *Senke* besitzt, so dass zu jedem Knoten x ein sx- und ein xt-Pfad existiert. Die Klasse der 2-terminalen Graphen wird mit $\mathcal{TT}$ bezeichnet und wir schreiben (G, s, t) für $G \in \mathcal{TT}$ mit Quelle s und Senkte t. In (G, s, t) ist der Knoten x eine *Pforte* des Knotens y, wenn x auf jedem Pfad von s nach y und von y nach t liegt. Weiter ist x eine Pforte der yz-Kante a wenn x Pforte von y und z ist.

Syntax und Semantik von (regulären) *Ausdrücken* folgen [HU79]. Die Klasse der Ausdrücke über Σ ist $\mathrm{Reg}(\Sigma)$, immer sind r und s Ausdrücke. Die von r beschriebene Sprache wird $L(r)$ bezeichnet. Ein *erweiterter endlicher Automat* (FA) ist ein Tupel $A = (Q, \Sigma, \delta, I, F)$ mit Zuständen Q, Alphabet Σ, Transitionen $\delta \subseteq Q \times \mathrm{Reg}(\Sigma) \times Q$, Startzuständen I und Endzuständen F. Die Relation $\vdash_A$ ist auf $Q \times \Sigma^*$ definiert durch $(q, vw) \vdash_A (q', w)$ gdw. $(q, r, q') \in \delta$ und $v \in L(r)$. Die von A akzeptierte Sprache ist

$$L(A) = \{w \mid (q_i, w) \vdash_A^* (q_f, \varepsilon),\, q_i \in I,\, q_f \in F\}$$

Ein EFA ist *normalisiert* falls $|I| = |F| = 1$, $I \neq F$ und jeder Zustand erreichbar und co-erreichbar ist. Ein (herkömmlicher) endlicher Automat (FA) ist ein EFA der $\delta \subseteq Q \times (\Sigma \cup \varepsilon) \times Q$ erfüllt.

2 Seriell-Parallele Graphen mit Schleifen

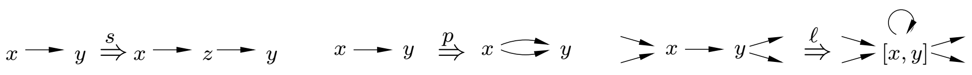


Figure 1: Expansion einer xy-Kante

Definition 1. Eine xy-Kante a kann wie folgt *expandiert* werden:

1. Serielle Expansion: Entferne a und füge einen neuen Knoten z sowie eine xz- und eine zy-Kante ein.
2. Parallele Expansion: Füge eine neue xy-Kante ein.
3. Schleifenexpansion: Gilt $\mathrm{d}^+(x) = \mathrm{d}^-(y) = 1$, vereine x und y.

Wir schreiben $G \overset{s}{\Rightarrow} H$, $G \overset{p}{\Rightarrow} H$ und $G \overset{\ell}{\Rightarrow} H$, wenn H aus G durch serielle, parallele oder Schleifenexpansion einer Kante entsteht und sagen, dass G zu H expandiert. Ausgehend von $\mathbf{P_1} := (\{x, y\}, \{a\}, \{(a, x)\}, \{(a, y)\})$, dem *Axiom*, konstruieren wir die Klasse $\mathcal{SPS}$:

Definition 2 ($\mathcal{SPS}$)**.**

- $\mathbf{P_1} \in \mathcal{SPS}$
- Mit $G \in \mathcal{SPS}$ und $G \overset{s}{\Rightarrow} H$ oder $G \overset{p}{\Rightarrow} H$ gilt $H \in \mathcal{SPS}$
- Mit $G \in \mathcal{SPS} \setminus \{\mathbf{P_1}\}$ und $G \overset{\ell}{\Rightarrow} H$ gilt $H \in \mathcal{SPS}$

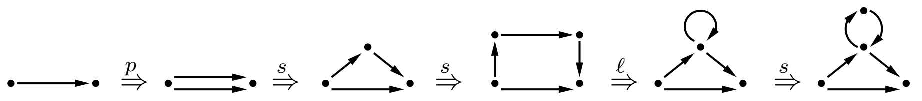


Figure 2: Konstruktion eines $\mathcal{SPS}$-Graphen aus $\mathbf{P_1}$.

Die Klasse $\mathcal{SPS}$ liegt echt zwischen den seriell-parallelen und den 2-terminalen Graphen. Um zu entscheiden, ob ein 2-terminaler Graph in $\mathcal{SPS}$ liegt, werden zu den Expansionen duale Operationen eingeführt.

Definition 3. Sei $G \in \mathcal{TT}$. Dann gilt

- $G \overset{s}{\Leftarrow} H$, falls ein Knoten x in G lediglich inzident zu einer yx- und einer xz-Kante ist, und H aus G durch Löschen von x und Einfügen einer yz-Kante hervorgeht.
- $G \overset{p}{\Leftarrow} H$, falls a eine von mehreren xy-Kanten in G ist und H aus G durch Entfernen von a hervorgeht.
- $G \overset{\ell}{\Leftarrow} H$, falls a eine x-Schleife in G ist, wobei x Pforte keiner Kante ausser a ist, und H aus G hervorgeht indem ein Knoten y zugefügt wird, welcher neuer Kopf von a, und neuer Schwanz aller aus x ausgehenden Kanten wird.

Die Relationen $\overset{s}{\Leftarrow}$, $\overset{p}{\Leftarrow}$ und $\overset{\ell}{\Leftarrow}$ heissen serielle Reduktion, paralllele Reduktion und Schleifenreduktion, und G wird zu H *reduziert*, wenn G und H in einer der Relationen stehen. Die Pforten-Bedingung der Schleifenreduktion ist notwendig für

Lemma 2.1. *$G \in \mathcal{SPS}$ genau dann wenn G auf $\mathbf{P_1}$ reduzierbar ist.*

Tatsächlich ist keine besondere Reduktionsstrategie nötig, da gilt

Lemma 2.2. *Das Ersetzungssystem $\langle \mathcal{TT}, \overset{\int}{\Leftarrow}, \overset{\surd}{\Leftarrow}, \overset{\ell}{\Leftarrow} \rangle$ ist konfluent.*

Damit ist $G \in \mathcal{SPS}$ wie folgt entscheidbar: Zunächst wird Zugehörigkeit zu $\mathcal{TT}$ getestet, dazu muss G genau einen Knoten s mit $\mathrm{d}^-(s) = 0$ und einen Knoten t mit $\mathrm{d}^+(t) = 0$ enthalten, welche Quelle bzw. Senke sind (Tiefensuche). In dem Fall terminiert jede erschöpfende Reduktionsfolge von G in einem eindeutigen Graphen $\mathrm{R}(G)$ und es gilt

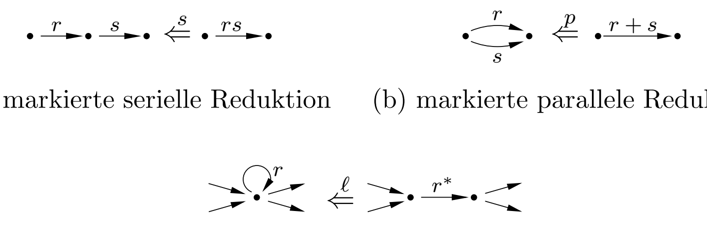
$$G \in \mathcal{SPS} \quad \text{gdw.} \quad \mathrm{R}(\mathcal{G}) = \mathbf{P_1}.$$

Der Linearzeit-Algorithmus für seriell-paralle Reduktion [VTL81], kann einfach um Schleifenreduktion erweitert werden. Der Test, ob eine x-Schleife reduzierbar ist, also ob x Pforte ist, ist linear, was quadratische Gesamtlaufzeit bedingt.

2.1 Kantenmarkierte Graphen

Die betrachteten Graphen bzw. die Expansions- und Reduktionsregel werden um Kantenmarkierungen erweitert, was zu *markierter Reduktion* führt. Als Markierung werden Ausdrücke gewählt; dabei folgen wir der Intuition, dass ein Produkt seriellen Charakter, eine Summe parallelen Charakter und eine Iteration Schleifencharakter hat.

Vollständige markierte Reduktion von $G \in \mathcal{SPS}$ mit Markierungen aus M (wobei $M \cap \{+, *\} = \emptyset$) ergibt das Axiom, welches mit einem Ausdruck $\mathrm{c}(G) \in \mathrm{Reg}(M)$ markiert ist.



(a) markierte serielle Reduktion (b) markierte parallele Reduktion

(c) markierte Schleifenreduktion

Figure 3: Reduktionen mit Ausdrücken als Kantenmarkierung

Lemma 2.3. *Zu $G \in \mathcal{SPS}$ ist der Ausdruck* $\mathrm{c}(G)$ *modulo Assoziativität und Kommutativität der regulären Operatoren eindeutig bestimmt.*

Kehrt man die markierten Reduktionen um, ergeben sich markierte Expansionen. Expandiert man derart das mit $\mathrm{c}(G)$ markierte Axiom bis in keiner Markierung Operatoren vorkommen, ergibt sich wieder G.

Die inneren Knoten im Syntaxbaum von $\mathrm{c}(G)$ kodieren die Struktur von $G \in \mathcal{SPS}$, während Blätter die Kantenmarkierungen enthalten. Unmarkierte $\mathcal{SPS}$-Graphen lassen sich als Ausdrücke über beliebigen nichtleeren Alphabeten kodieren, etwa über $\{\varepsilon\}$ oder der Kantenmenge des Graphen. Beispielsweise kodiert $\varepsilon+(\varepsilon(\varepsilon\varepsilon)^*\varepsilon)$ den in Abb. 2 konstruierten Graphen.

Ein normalisierter FA A mit Startzustand q_0 und Endzustand q_f lässt sich als 2-terminaler Graph mit Quelle q_0, Senke q_f und Markierungen aus Σ auffassen. Dieser lässt sich unter Erhalt der akzeptierten Sprache zu einem eindeutig bestimmten EFA $\mathrm{R}(A)$ reduzieren; liegt der A zugrundeliegende Graph in $\mathcal{SPS}$, so ergibt sich ein einzelner Ausdruck $\mathrm{c}(A)$ als Markierung von $\mathrm{R}(A) = \mathbf{P_1}$ und es gilt

$$L(\mathrm{c}(A)) = L(A)$$

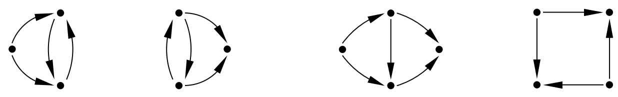
Markierte Expansion von $\mathbf{P_1}$ mit $\mathrm{c}(A)$ als Markierung ergibt wieder A. Beide Richtungen der Konstruktion ergeben eine eindeutige Zuordnung von Transitionen in A und Literalvorkommen in $\mathrm{c}(A)$. Demnach ist die Größe von A linear in der von $\mathrm{c}(A)$ und umgekehrt.

Mit etwas Mehraufwand lässt sich markierte Expansion auf beliebige Ausdrücke, die zunächst keinen FA mit $\mathcal{SPS}$-Struktur kodieren, erweitern. In einem Ausdruck werden dazu alle — ohnehin redundanten — Vorkommen von r^{**}, εr, $r\varepsilon$ und ε^* entfernt. Bei Schleifenexpansion einer xy-Kante werden abhängig von $\mathrm{d}^+(x)$ und $\mathrm{d}^-(y)$ zusätzlich ε-Transitionen eingeführt. Die dabei auftretenden Fälle entsprechen den in [GF08] beschriebenen.

3 Charakterisierung durch verbotene Minoren

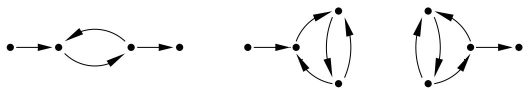
Eine *Unterteilung* von F entsteht durch eine Folge serieller Expansionen aus F. F ist ein *Minor* von G, wenn G eine Unterteilung von F als Teilgraphen enthält. Weiter ist F ein *freier* Minor von G, wenn F Minor von G ist und je zwei innendisjunkten Pfaden zwischen Knoten einer F-Unterteilung in G zwei verschiedene Kanten in F entsprechen.

In Erweiterung der Minorencharakterisierung seriell-paralleler Graphen [VTL81] lässt sich $\mathcal{SPS}$ über die Mengen $\boldsymbol{\mathcal{F}} = \{\mathbf{C}, \mathbf{C}^{\mathrm{R}}, \mathbf{N}, \mathbf{Q}\}$ und $\boldsymbol{\mathcal{B}} = \{\boldsymbol{\Phi}, \boldsymbol{\Psi}, \boldsymbol{\Psi}^{\mathrm{R}}\}$ vollständig charakterisieren (Abbn. 4, 5).



(a) $\mathbf{C}$ (b) $\mathbf{C}^{\mathrm{R}}$ (c) $\mathbf{N}$ (d) $\mathbf{Q}$

Figure 4: Menge $\boldsymbol{\mathcal{F}}$ verbotener Minoren



(a) $\boldsymbol{\Phi}$ (b) $\boldsymbol{\Psi}$ (c) $\boldsymbol{\Psi}^{\mathrm{R}}$

Figure 5: Menge $\boldsymbol{\mathcal{B}}$ verbotetener freier Minoren

Satz 3.1. *$G \in \mathcal{SPS}$ genau dann wenn $G \in \mathcal{TT}$ und*

1. *kein $F \in \boldsymbol{\mathcal{F}}$ ist Minor von G, und*
2. *kein $B \in \boldsymbol{\mathcal{B}}$ ist freier Minor von G*

References

[GF08] Stefan Gulan and Henning Fernau. An optimal construction of finite automata from regular expressions. In R. Hariharan, M. Mukund, and V. Vinay, editors, *FSTTCS*, pages 211–222, 2008.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[VTL81] Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1981.

The Computational Complexity of RaceTrack*

Markus Holzer [1]†*and Pierre McKenzie* [2]

[1]Institut für Informatik, Universität Giessen,

Arndtstraße 2, D-35392 Giessen, Germany

`holzer@informatik.uni-giessen.de`

[2]Département d'I.R.O., Université de Montréal, C.P. 6128,

succ. Centre-Ville, Montréal (Québec), H3C 3J7 Canada

`mckenzie@iro.umontreal.ca`

RACETRACK is a popular multi-player simulation pencil-paper game of car racing. The origin of the game is not clear, but most people remember this game from high school, where great effort and time was spent to become the RACETRACK champion, even at the expense of the said champion's school results. Variants of the game appeared all over the world under various names like, e.g., *Le Zip* in France or *Vektorrennen* in Germany. Here is the game description, literally taken from Gardner [2]:

> The game is played on math-paper where a racetrack is drawn. Then the cars are lined up at a grid position at the start line, and at each turn a player moves his car along the track to a new grid position subject to the following rules:
>
> 1. The new grid point and the straight line segment joining it to the preceding grid point must lie entirely within the track.
> 2. No two cars may simultaneously occupy the same grid point, i.e., no collisions are allowed.
> 3. Acceleration and deceleration are simulated as follows: a car maintains its speed in either direction or it can change its speed by only one distance unit per move—see Figure 2 for illustration. The first move following this rule is one unit horizontally or vertically, or both.

*This is an extended abstract of: M. Holzer and P. McKenzie. The computational complexity of RACETRACK. In P. Boldi and L. Gargano, editors, *Proceedings of the 5th International Conference on FUN with Algorithms*, number 6099 in LNCS, pages 260–271, Ischia Island, Italy, June 2010. Springer.

†Part of the work was done while the author was at the Département d'I.R.O., Université de Montréal, C.P. 6128, succ. Centre-Ville, Montréal (Québec), H3C 3J7 Canada, and the Institut für Informatik, Technische Universität München, Boltzmannstraße 3, D-85748 Garching bei München, Germany.

The first car to cross the finish line (point) wins. A car that collides with another car or leaves the track is out of the race.

A sample game—taken from Gardner [2]—is depicted in Figure 1 and shows a race of two cars. In his collection of *Scientific American* columns Gardner [3] described a

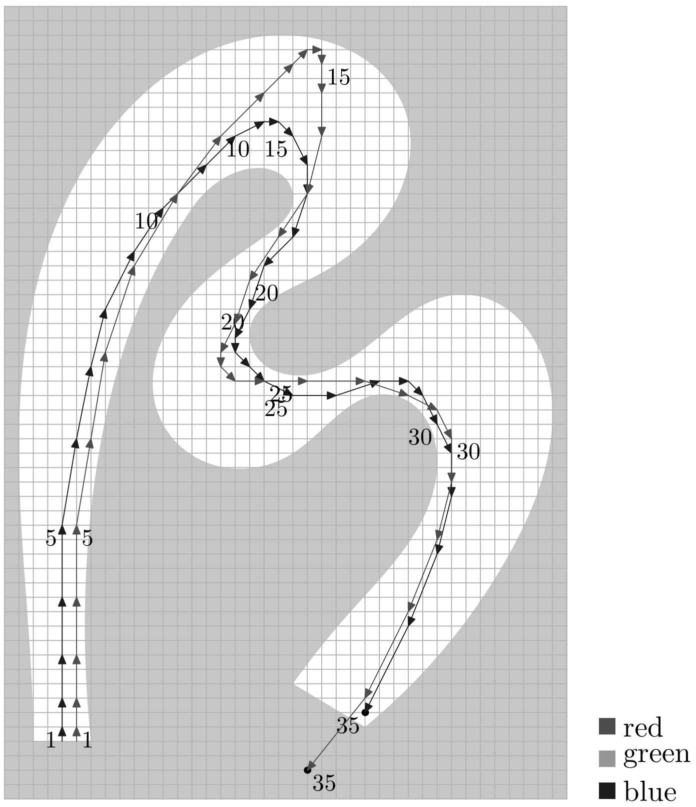


Figure 1: A sample RaceTrack game of two cars—red wins against blue in 35 moves.

variety of modifications to the original RaceTrack game such as, e.g., fast acceleration and power braking moves, allowing cars to occupy the same point at the same time, etc. Even more complicated modifications like for instance patches representing oil slicks that require cars to move at a constant speed and direction were suggested.

Here we investigate the complexity of RaceTrack when played as a 1-player or 2-player game. Before describing our results, we note that the shape of the racetrack border is *a priori* arbitrary. Thus, in some far-fetched settings, merely verifying whether a move is valid, i.e., merely checking whether the new grid point and the straight line segment joining it to the preceding grid point lies entirely within the track, could be undecidable. To avoid such complications, we stick to a discrete version of the racetrack border, where the track is drawn along grid lines—see Figure 2. This is a reasonable restriction, which does not change the practical appeal of the game.

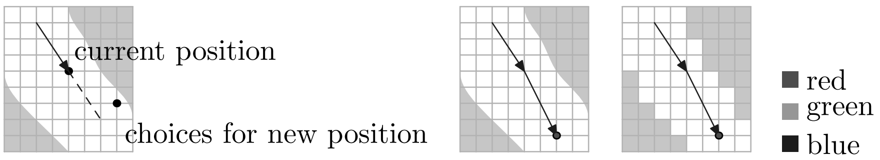


Figure 2: (Left:) The arrow depicts the last move of the car—two squares east and three squares south—on the racetrack (drawn in white); the gray shaded area is outside of the racetrack. If the car maintains its speed it will follow the dashed line and go two squares east and three squares south again, but it can also reach one extra square north, south, east, or west of this point by changing speed. These extra points are marked by black dots. (Middle:) One out of nine legal moves are shown. (Right:) Discrete version of the RaceTrack game. Observe that certain movements of the car are not possible anymore if the border is *not* allowed for driving.

Solitaire RaceTrack will refer to the problem of deciding whether a single player can reach the finish line within an input-specified number of legal moves. By RaceTrack *reachability*, we will mean the simpler problem of deciding whether the single player can reach the finish line at all. First we reduce the *touching* variant of RaceTrack reachability (i.e., with the track border considered part of the driving area) to the undirected grid graph reachability problem (UGGR). More precisely we show the following result:

Theorem 1. RaceTrack *reachability, where the track boundary can be used for driving, is equivalent to UGGR under* AC^0 *reducibility.*

From [1] we deduce that this touching variant is NC^1-hard and can be solved in deterministic logarithmic space. By contrast, and to our initial surprise, we then show that the *non-touching* reachability variant is NL-complete, hence that the complexity of the RaceTrack reachability problem crucially depends on whether the car is allowed to touch the racetrack border or not.

Theorem 2. RaceTrack *reachability, where the track boundary* cannot *be used for driving, is* NL*-complete. Moreover, solitaire* RaceTrack *is* NL*-complete, too, regardless of whether the track boundary can be used for driving or not.*

Finally, we turn to the 2-player game and prove that checking whether the first player has a winning strategy is P-complete.

Theorem 3. *Deciding if the first player has a winning strategy in the* 2*-player* RaceTrack *game, regardless of whether the track boundary can be used for driving or not, is* P*-complete.*

In particular, the 2-player game is efficiently solvable (in polynomial time). Consider the following "popular conjecture:"

Conjecture. *All "fun and interesting" (2-player) games are* NP*-hard.*

This "conjecture" attempts to capture when a "game" makes a game and it is wildly accepted in the algorithmic game theory community: in order to be interesting, a game purportedly needs enough complexity to be able to encode interesting (NP-hard) computational problems. And a game in P supposedly becomes boring because a player can quickly learn "the trick" to perfect play. The 2-player RaceTrack game, being fun and interesting to play, yet polynomial time solvable, is a rare example of a game that violates the implication of this conjecture.

References

[1] E. Allender, D. A. Mix Barrington, T. Chakraborty, S. Datta, and S. Roy. Planar and grid graph reachability problems. *Theory of Computing Systems*, 45(4):675–723, July 2009.

[2] M. Gardner. Mathematical games—Sim, Chomp and Race Track: new games for the intellect (and not for Lady Luck). *Scientific American*, 228(1):108–115, January 1973.

[3] M. Gardner. *Knotted Doughnuts and Other Mathematical Entertainments.* W. H. Freeman and Company, 1986.

The Magic Number Problem for Subregular Language Families[*]

Markus Holzer, Sebastian Jakobi and Martin Kutrib

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

It is well-known that, given some n-state nondeterministic finite automaton (NFA), one can always construct an equivalent deterministic finite automaton (DFA) with at most 2^n states [15]. This so-called *powerset construction* turned out to be optimal, in the sense that for an arbitrary n there is always some n-state NFA which cannot be simulated by any DFA with less than 2^n states [11]. On the other hand, there are cases where nondeterminism does not help for the succinct representation of a language compared to DFAs. A decade ago a very fundamental question on the well known subset construction was raised in [4]: Does there always exist a minimal n-state NFA whose equivalent minimal DFA has α states, for all n and α with $n \leq \alpha \leq 2^n$? A number α not satisfying this condition is called a *magic number* for n. The answer to this simple question turned out not to be so easy. For NFAs over a two-letter alphabet various non-magic numbers were identified in [2, 4, 5, 7, 8, 13], but the range from n to 2^n is not totally classified yet. The problem becomes easier if one allows more input letters. First it was shown that for exponentially growing alphabets there are no magic numbers at all [7]. This result was improved to constant alphabets of size four [6], and very recently to three-letter alphabets [10]. Magic numbers for unary NFAs were recently studied in [3]. In the same paper also a brief historical summary of the magic number problem can be found. Further results on the magic number problem (in particular in relation to the operation problem on regular languages) can be found, for example, in [8, 9].

To our knowledge the magic number problem was not systematically studied for subregular languages families (except for unary languages). Several of these subfamilies are

[*]This is an abstract and slightly extended version of: M. Holzer, S. Jakobi and M. Kutrib. *The Magic Number Problem for Sugreagular Language Families.* In I. McQuillan and G. Pighizzini, editors, Proceedings of the 12th Annual Workshop on Descriptional Complexity of Formal Systems, pp. 110–119, Saskatoon, Canada, August 2010.

well motivated by their representations as finite automata or regular expressions: finite languages (are accepted by acyclic finite automata), star-free languages or regular non-counting languages (which can be described by regular-like expressions using only union, concatenation, and complement), star languages (which are expressed by the Kleene star of a regular language), definite languages (can be realized by a register and a combinational circuit), suffix-free languages (are accepted by non-returning automata, that is, automata where the initial state does not have any in-transition), prefix-free languages (are accepted by non-exiting automata, that is, automata where all out-transitions of every accepting state go to a rejecting sink state), infix-free languages (are accepted by non-returning and non-exiting automata, where these conditions are necessary, but not sufficient), prefix-closed, suffix-closed and infix-closed languages (are accepted by automata where all states are accepting, initial and both accepting and initial, respectively).

We study all mentioned families with respect to the magic number problem, and show—except for finite languages, where only some partial results will be presented—that there are only trivial magic numbers, whenever they exist. That is, given n and α, we construct a minimal n-state NFA whose equivalent minimal DFA has α states. The basis for most of our constructions is an automaton from [6] by Jirásek, Jirásková, and Szabari that solves the general magic number problem using a four letter alphabet. In the following their automaton will be refered to as the JJS-automaton. We adapt this construction to show that in the aforementioned language families, up to their specific upper bounds for the NFA to DFA conversion [1], there are no magic numbers.

An observation from [1] shows that the magic number problem for *elementary* and *combinational* languages is trivial, since automata accepting such (non-trivial) lanugages always need two or three states. For *star-free languages* we obtain the following result:

Theorem 1. *For all integers n and α such that $n \leq \alpha \leq 2^n$, there exists an n-state nondeterministic finite automaton accepting a* star-free *language whose equivalent minimal deterministic finite automaton has exactly α states.*

This is shown by proving the deterministic JJS-automaton to be *permutation-free*, which is a characterization of star-free languages given in [14]. By some minor modifications for the JJS-automaton, we show that also for star languages, all numbers from n to 2^n are non-magic, which generalizes to *(two-sided) comets*, since these contain the star languages.

For some variants of definite language families, we show that from n up to the language families' specific upper bounds for the NFA to DFA conversion, there are no magic numbers. Numbers above the bounds are trivially magic. According to [1], these bounds are 2^{n-1} for *ultimate definite languages*, $2^{n-1}+1$ for *reverse ultimate definite languages* and $2^{n-2}+1$ for *central definite languages*. Furthermore, for *symmetric definite languages*, which contain all previous three language families, we prove a tight upper bound of $2^{n-1}+1$ states for the NFA to DFA conversion, which can be improved to 2^{n-1} when considering symmetric definite but *not* reverse ultimate definite languages. This solves an open problem from [1].

For subword-specific languages, there are also only trivial magic numbers which are either n or greater than the language families' upper bounds for the NFA to DFA conversion. In detail, the numbers from $n+1$ to $2^{n-1}+1$ are non-magic for *prefix-free* and *suffix-free languages*, while for *infix-free languages* the upper bound is $2^{n-2}+2$. If $n>1$, then n is a magic number for *prefix-closed* and *infix-closed languages*, and for the latter and for *suffix-closed languages*, the numbers from $2^{n-1}+2$ to 2^n are magic due to the upper bounds for the NFA to DFA conversion in these language families. All this is proven by suitable modifications of the JJS-automaton.

The magic number problem for *finite languages* is more challenging. First notice that n is magic for every nonempty finite language, since any DFA needs a sink state to accept a finite language and the NFA does not. Further we get the following results:

Theorem 2. *For all integers n and α such that $n+1 \leq \alpha \leq (\frac{n}{2})^2 + \frac{n}{2} + 1$ if n is even, and $n+1 \leq \alpha \leq (\frac{n-1}{2})^2 + n + 1$ if n is odd, there exists an n-state nondeterministic finite automaton accepting a* finite *language over a binary alphabet whose equivalent minimal deterministic finite automaton has exactly α states.*

And finally by adapting results from [12] we obtain the following:

Theorem 3. *For all integers n and α such that $\alpha = 3 \cdot 2^{(n/2)-1} + \beta$ if n is even and $\alpha = 2^{(n+1)/2} + \beta$ if n is odd, with $\beta = 2^i - 1$ for some integer $1 \leq i \leq \lceil \frac{n-1}{2} \rceil$, there exists an n-state nondeterministic finite automaton accepting a* finite *language over a binary alphabet whose equivalent minimal deterministic finite automaton has exactly α states.*

References

[1] H. Bordihn, M. Holzer & M. Kutrib (2009): *Determinization of Finite Automata Accepting Subregular Languages. Theoret. Comput. Sci.* 410, pp. 3209–3222.

[2] V. Geffert (2005): *(Non)determinism and the size of one-way finite automata.* In: *Descriptional Complexity of Formal Systems (DCFS 2005)*, Universita degli Studi di Milano, pp. 23–37.

[3] V. Geffert (2007): *Magic numbers in the state hierarchy of finite automata.* Inform. Comput. 205, pp. 1652–1670.

[4] K. Iwama, Y. Kambayashi & K. Takaki (2000): *Tight bounds on the number of states of DFAs that are equivalent to n-state NFAs.* Theoret. Comput. Sci. 237, pp. 485–494.

[5] K. Iwama, A. Matsuura & M. Paterson (2003): *A family of NFAs which need $2^n - \alpha$ deterministic states.* Theoret. Comput. Sci. 301, pp. 451–462.

[6] Jozef Jirásek, Galina Jirásková & Alexander Szabari (2007): *Deterministic Blow-Ups of Minimal Nondeterministic Finite Automata over a Fixed Alphabet.* In: Developments in Language Theory (DLT 2007), number 4588 in LNCS, Springer, pp. 254–265.

[7] G. Jirásková (2001): *Note on minimal finite automata.* In: Mathematical Foundations of Computer Science (MFCS 2001, number 2136 in LNCS, Springer, pp. 421–431.

[8] G. Jirásková (2008): *On the State Complexity of Complements, Stars, and Reversal of Regular Languages.* In: Developments in Language Theory (DLT 2008), number 5257 in LNCS, Springer, pp. 431–442.

[9] G. Jirásková (2009): *Concatenation of Regular Languages and Descriptional Complexity.* In: Computer Science Symposium in Russia, number 5675 in LNCS, Springer, pp. 203–214.

[10] G. Jirásková (2009): *Magic Numbers and Ternary Alphabet.* In: Developments in Language Theory (DLT 2009), number 5583 in LNCS, Springer, pp. 300–311.

[11] O. B. Lupanov (1963): *A Comparison of two types of Finite Sources.* Problemy Kybernetiki 9, pp. 321–326, (in Russian). German translation (1966): *Über den Vergleich zweier Typen endlicher Quellen.* Probleme der Kybernetik 6, pp. 328-335.

[12] R. Mandl (1973): *Precise bounds associated with the subset construction on various classes of nondeterministic finite automata.* In: Conference on Information and System Sciences, pp. 263–267.

[13] A. Matsuura & Y. Saito (2008): *Equivalent Transformation of Minimal Finite Automata over a Two-Letter Alphabet.* IPSJ SIG Notes AL-117, pp. 75–82. (in Japanese).

[14] R. McNaughton & S. Papert (1971): *Counter-free automata.* Number 65 in Research monographs. MIT Press.

[15] M. O. Rabin & D. Scott (1959): *Finite Automata and Their Decision Problems. IBM J. Res. Dev.* 3, pp. 114–125.

Transductions Computed by PC-Systems of Monotone Deterministic Restarting Automata*

Norbert Hundeshagen, Friedrich Otto, Marcel Vollweiler
Fachbereich Elektrotechnik/Informatik,
Universität Kassel, 34109 Kassel,Germany
{*hundeshagen,otto,vollweiler*}*@theory.informatik.uni-kassel.de*

Abstract

We associate a *transduction* (that is, a binary relation) with the characteristic language of a restarting automaton, and we prove that in this way monotone deterministic restarting automata yield a characterization of pushdown transductions. Then we study the class of transductions that are computed by *parallel communicating systems* (PC-systems) of monotone deterministic restarting automata. We will see that this class includes all transductions that are computable.

1 Introduction

Automata with a restart operation were introduced originally to describe a method of grammar-checking for the Czech language (see, e.g., [5]). These automata started the investigation of restarting automata as a suitable tool for modeling the so-called *analysis by reduction*, which is a technique that is often used (implicitly) for developing formal descriptions of natural languages based on the notion of *dependency* [6, 12]. In particular, the Functional Generative Description (FGD) for the Czech language (see, e.g., [7]) is based on this method.

FGD is a dependency based system, which translates given sentences into their underlying tectogrammatical representations, which are (at least in principle) disambiguated. Thus, the real goal of performing analysis by reduction on (the enriched form of) an input sentence is not simply to accept or reject this sentence, but to extract information from that sentence and to translate it into another form (be it in another natural language or a formal representation). Therefore, we are interested in *transductions* (that is, binary relations) and in ways to compute them by certain types of restarting automata.

*This work is a summary of a paper presented at CIAA 2010.

Here we study two different approaches. First we associate a binary relation with the *characteristic language* of a restarting automaton, motivated by the way in which the so-called *proper language* of a restarting automaton is defined. In this way we obtain a characterization for the class of *pushdown transductions* in terms of monotone deterministic restarting automata. Then we introduce *parallel communicating systems* (PC-systems, for short) that consist of two monotone deterministic restarting automata, using them to compute transductions. In this way we obtain a characterization for the class of all computable transductions. In addition we consider the *input-output transductions* computed by these two types of restarting automata.

2 Transductions Computed by Restarting Automata

A large variety of types of restarting automata has been developed over the years. Here we are only interested in the deterministic RRWW-automaton. Such an automaton consists of a finite-state control, a single flexible tape with end markers, and a read/write window of fixed size. Formally, it is described by an 8-tuple $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite tape alphabet containing Σ, the symbols $¢, \$ \notin \Gamma$ are used as markers for the left and right border of the work space, respectively, $q_0 \in Q$ is the initial state, $k \geq 1$ is the size of the *read/write window*, and δ is the *transition function*.

Without looking at details (an overview can be found in [10]) different languages can be associated with these types of automata: $L_{\mathrm{C}}(M)$ denotes the *characteristic language* of M (roughly speaking: the language of words over Γ), while the set $L(M) = L_{\mathrm{C}}(M) \cap \Sigma^*$ of all input sentences accepted by M is the *input language recognized* by M. Further, $L_{\mathrm{P}}(M) = \mathsf{Pr}^{\Sigma}(L_{\mathrm{C}}(M))$[1] is the *proper language* of M.

We study transductions that are computed by RRWW-automata. Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an RRWW-automaton with tape alphabet Γ, which contains the input alphabet Σ and the *output alphabet* Δ. Here we assume that Σ and Δ are disjoint. With M we associate two transductions, where $\mathrm{sh}(u, v)$ denotes the *shuffle* of the words u and v:

$$R_{\mathrm{io}}(M) = \{ (u, v) \in \Sigma^* \times \Delta^* \mid L_{\mathrm{C}}(M) \cap \mathrm{sh}(u, v) \neq \emptyset \},$$
$$R_{\mathrm{P}}(M) = \{ (u, v) \in \Sigma^* \times \Delta^* \mid \exists w \in L_{\mathrm{C}}(M) : u = \mathsf{Pr}^{\Sigma}(w) \wedge v = \mathsf{Pr}^{\Delta}(w) \}.$$

Here $R_{\mathrm{io}}(M)$ is the *input-output transduction* of M, and $R_{\mathrm{P}}(M)$ is the *proper transduction* of M. By $\mathcal{R}el_{\mathrm{io}}(\mathsf{RRWW})$ ($\mathcal{R}el_{\mathrm{P}}(\mathsf{RRWW})$) we denote the class of input-output transductions (proper transductions) of RRWW-automata.

[1]If Σ is a subalphabet of an alphabet Γ, then by Pr^{Σ} we denote the projection from Γ^* onto Σ^*. For a language $L \subseteq \Gamma^*$, $\mathsf{Pr}^{\Sigma}(L) = \{ \mathsf{Pr}^{\Sigma}(w) \mid w \in L \}$.

3 Transformations Computed by PC-Systems of Monotone Deterministic RRWW-Automata

Instead of computing a transduction R by a single restarting automaton, we propose to compute it by a pair of restarting automata that have the ability to communicate with each other. To formalize this idea, we introduce the so-called *parallel communicating system of restarting automata* (or PC-RRWW-system, for short). A PC-RRWW-system consists of a pair $\mathcal{M} = (M_1, M_2)$ of RRWW-automata $M_i = (Q_i, \Sigma_i, \Gamma_i, ¢, \$, q_0^{(i)}, k, \delta_i)$, $i = 1, 2$. Here it is required that, for each $i \in \{1, 2\}$, the set of states Q_i of M_i contains finite subsets Q_i^{req} of *request states* of the form (q, req), Q_i^{res} of *response states* of the form $(q, \mathrm{res}(l))$, Q_i^{rec} of *receive states* of the form $(q, \mathrm{rec}(l))$, and Q_i^{ack} of *acknowledge states* of the form $(q, \mathrm{ack}(l))$. Further, in addition to the move-right, rewrite and restart steps, M_1 and M_2 have so-called *communication steps.*

Different types of transductions are associated with PC-RRWW-systems similar to the definitions used for restarting automata. Here $R_{\mathrm{C}}(\mathcal{M})$ is the *characteristic transduction* of $\mathcal{M}$, $R_{\mathrm{io}}(\mathcal{M}) = R_{\mathrm{C}}(\mathcal{M}) \cap (\Sigma_1^* \times \Sigma_2^*)$ is the *input-output transduction* of $\mathcal{M}$, and $R_{\mathrm{P}}(\mathcal{M}) = \{ (\mathsf{Pr}^{\Sigma_1}(w_1), \mathsf{Pr}^{\Sigma_2}(w_2)) \mid (w_1, w_2) \in R_{\mathrm{C}}(\mathcal{M}) \}$ is the *proper transduction* of $\mathcal{M}$.

4 Results

The definitions above lead to the results shown in the figure below, where arrows denote proper inclusions.

$$\begin{tikzcd}
\text{Pushdown Transductions} \arrow[d, equal] & & \text{Computable Transductions} \arrow[d, equal] \\
\mathcal{R}el_{\mathrm{P}}(\text{det-mon-RRWW}) \arrow[rr] & & \mathcal{R}el_{\mathrm{P}}(\text{det-mon-PC-RRWW}) \\
\mathcal{R}el_{\mathrm{io}}(\text{det-mon-RRWW}) \arrow[u] & & \mathcal{R}el_{\mathrm{io}}(\text{det-mon-PC-RRWW}) \arrow[u] \\
& \text{RatRel} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Fig. 1. Taxonomy of classes of transductions computed by various types of monotone deterministic restarting automata. Here RatRel denotes the class of rational transductions.

References

[1] A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation, and Compiling, Vol. I: Parsing.* Prentice-Hall, Englewood Cliffs, N.J., 1972.

[2] J. Berstel. *Transductions and Context-free Languages.* Teubner Studienbücher, Teubner, Stuttgart, 1979.

[3] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. On monotonic automata with a restart operation. *J. Autom. Lang. Comb.* 4 (1999) 283-292.

[4] T. Jurdziński and K. Loryś. Church-Rosser languages vs. UCFL. In: P. Widmayer, F. Triguero, R. Morales, M. Hennessy, S. Eidenbenz, and R. Conejo (eds.), *ICALP'02, Proc.*, *LNCS* 2380, Springer, Berlin, 2002, 147–158.

[5] V. Kuboň and M. Plátek. A grammar based approach to a grammar checking of free word order languages. In: *COLING'94, Proc., Vol. II*, Kyoto, Japan, 1994, 906–910.

[6] M. Lopatková, M. Plátek, and V. Kuboň. Modeling syntax of free word-order languages: Dependency analysis by reduction. In: V. Matoušek, P. Mautner, and T. Pavelka (eds.), *TSD 2005, Proc.*, *LNCS* 3658, Springer, Berlin, 2005, 140–147.

[7] M. Lopatková, M. Plátek, and P. Sgall. Towards a formal model for functional generative description: Analysis by reduction and restarting automata. *The Prague Bulletin of Mathematical Linguistics* 87 (2007) 7–26.

[8] H. Messerschmidt and F. Otto. On deterministic CD-systems of restarting automata. *Intern. J. Found. Comput. Sci.* 20 (2009) 185–209.

[9] H. Messerschmidt and F. Otto. A hierarchy of monotone deterministic non-forgetting restarting automata. *Theory of Computing Systems*, online: DOI 10.1007/s00224-009-9247-x, published on Nov. 21, 2009.

[10] F. Otto. Restarting Automata. In: Z. Esik, C. Martin-Vide, and V. Mitrana (eds.), *Recent Advances in Formal Languages and Applications*, Vol. 25 of *Studies of Computational Intelligence*, Springer (2006) 269–303.

[11] F. Otto. On proper languages and transformations of lexicalized types of automata. *AFLAS 2008, Proc.*, World Scientific, Singapore, to appear.

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

Generalizing over Several Learning Settings

Anna Kasprzik
`kasprzik@informatik.uni-trier.de`

Introduction. We recapitulate inference from membership and equivalence queries, positive and negative samples. Regular languages cannot be learned from one of those information sources only [12, 3, 4]. Combinations of two sources allowing regular (polynomial) inference are MQs and EQs [2], MQs and positive data [1, 6], positive and negative data [16, 8]. We sketch a meta-algorithm fully presented in [14] that generalizes over as many combinations of those sources as possible. This includes a survey of pairings for which there are no well-studied algorithms.

Definition 1. *$T = \langle S, E, obs\rangle$ ($S, E \subseteq \Sigma^*$) is an* observation table *if S is prefix-closed and $obs(s, e) = 1$ if $se \in L$, 0 if $se \notin L$, $*$ if unknown. Let $row(s) := \{(e, obs(s, e))|e \in E\}$. S is partitioned into* RED *and* BLUE. *We call $r, s \in S$* obviously different *(OD; $r <> s$) iff $\exists e \in E$ with $obs(r, e) \neq obs(s, e)$ and $obs(r, e), obs(s, e) \in \{0, 1\}$. T is* closed *iff $\neg\exists s \in$* BLUE *$: \forall r \in$* RED *$: r <> s$.*

Let $r \equiv_L s$ iff $re \in L \Leftrightarrow se \in L$ for all $r, s, e \in \Sigma^*$. Let $I_L := |\{[s_0]_L | s_0 \in \Sigma^*\}|$. Due to the Myhill-Nerode theorem there is a unique total state-minimal DFA $\mathcal{A}_L$ with I_L states, each recognizing an equivalence class.
From a closed and consistent (see [2]) table $T = \langle S, E, obs\rangle$ with $\varepsilon \in E$ we derive a DFA $\mathcal{A}_T = \langle \Sigma, Q_T, q_T, F_T, \delta_T\rangle$ with $Q_T = row(\text{RED})$, $q_T = row(\varepsilon)$, $F_T = \{row(s)|s \in \text{RED}, obs(s, \varepsilon) = 1\}$, and $\delta_T = \{(row(s), a) \mapsto q|\neg(q <> row(sa)),\ s \in \text{RED},\ a \in \Sigma,\ sa \in S\}$. $\mathcal{A}_T$ has at most I_L states (see [2]).

Definition 2. *A finite $X \subseteq L$ is* representative *for L with min. DFA $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta\rangle$ iff $[\forall(q_1, a) \mapsto q_2 \in \delta : a \in \Sigma \Rightarrow \exists w \in X : \exists u, v \in \Sigma^* : w = uav \wedge (q_0, u) \mapsto q_1 \in \delta] \wedge [\forall q \in F : \exists w \in X : (q_0, w) \mapsto q \in \delta]$. A finite $X \subseteq \Sigma^* \setminus L$ is* separative *iff $\forall q_1 \neq q_2 \in Q : \exists w \in X : \exists u, v \in \Sigma^* : w = uv \wedge [\delta(q_L, u) = q_1 \vee \delta(q_L, u) = q_2] \wedge \exists(q_1, v) \mapsto q_a, (q_2, v) \mapsto q_b \in \delta : [(q_a \in F \wedge q_b \in (Q \setminus F)) \vee (q_b \in F \wedge q_a \in (Q \setminus F))]$.*

All learning algorithms we consider can be seen to start out with a provisional set of classes and converge to the partition by $\equiv_L$ by splitting or merging them according to obtained information. In a table S contains strings whose rows are candidates for *states* in the minimal DFA, and E *experiments* ('*contexts*') proving that two strings belong to distinct classes and represent different states.

Algorithm GENMODEL. The input is a tuple $IP = \langle EQ, MQ, X_+, X_- \rangle$ with Boolean values stating if EQs or MQs can be asked, a positive, and a negative finite sample of L. After initializing T we enter a loop checking if T is closed and, if it is, if we can still find states that should be split up, until there is no more information to process. The algorithm is composed of the following procedures:
INIT initializes the oracle (MQORACLE returns a blackbox for L if $MQ = \mathit{1}$ and otherwise the prefix automaton for X_+ as an imperfect oracle improved during the process) and T. The set RED contains candidates that were fixed to represent a state in the output, and is initialized by ε (start state), and BLUE contains candidates representing states to which there is a transition from one of the states in RED. WHITE is the set of remaining candidates from which BLUE is filled up. The set of (initial) candidates is given by **POOL**: If $X_+ \neq \emptyset$ POOL returns $\mathit{Pref}(X_+)$, otherwise all strings up to length 2. If $X_- \neq \emptyset \wedge MQ = \mathit{1}$ POOL builds X_+ from X_-: Let $n_- := |\mathit{Suff}(X_-)|$. In a worst case, every suffix in a separative X_- distinguishes a different of the $(I_L^2 - I_L)/2$ pairs of states. From $n_- \leq (I_L^2 - I_L)/2$ we compute an upper bound for I_L and take all strings up to that length as X_+ as the longest shortest representative of a state in $\mathcal{A}_L$ is at most of length I_L. Note that $|X_+|$ can be exponential with respect to $|X_-|$.
We also have **UPDATE** which clears the elements that were moved to BLUE out of WHITE and fills in the cells of T if we have a perfect membership oracle which for $MQ = \mathit{1}$ is true at any time and for $MQ = \mathit{0}$ when we have processed all available information, provided that it was sufficient. For the cases with empty samples but $MQ = \mathit{1}$ we fill up WHITE with all one-symbol extensions of BLUE.
CLOSURE is straightforward, it successively finds all elements preventing the closedness of T, moves them to RED, and calls UPDATE to fill the table.
NEXTDIST calls FINDNEXT to look for a candidate to be fixed as another state of the output. Then T is modified by MAKEOD such that CLOSURE will move this string to RED. If no such candidate is found FINDNEXT returns $\langle \varepsilon, \varepsilon \rangle$ (this can be seen as a test for the termination criterion). In that case WHITE is emptied if we use queries only, for all other cases the remaining candidates are moved to BLUE in order not to lose the information contained in the pool.
If $MQ = \mathit{1}$ **FINDNEXT** exploits a counterexample. $EQ = \mathit{1}$: c is given by the oracle. Else if $X_+ \neq \emptyset$ the learner tries to build c from $T_{ext} = \langle S \cup \text{WHITE}, E \cup \mathit{Suff}(X_+), obs \rangle$. This succeeds if X_+ is representative (see [14]). At least one prefix of c must be a distinct state of the output, but as it may not be in BLUE MINIMIZE is called to replace the BLUE

prefix of c until it finds $s'e'$ with $s' \in$ BLUE and e' distinguishes s' from all RED elements: FINDNEXT returns $\langle s'e' \rangle$. If *MQ* = *0* we continue merging states unless there is information preventing it. After the call of MERGENEXT either all BLUE strings correspond to states resulting from a merge or there is s which is a non-mergeable state. FINDNEXT returns $\langle s, \varepsilon \rangle$ as s should be a distinct state of the solution. In cases not covered by these distinctions we cannot reliably find another candidate to move and return $\langle \varepsilon, \varepsilon \rangle$.
MAKEOD is called if FINDNEXT returns $\langle s, e \rangle$ with $s \neq \varepsilon$, i.e., s is to be moved to RED by CLOSURE. If *MQ* = *1* there is a single $r \in$ RED *not* OD from s (RED elements are pairwise OD, and rows of S are complete), and e separates s from r, so add e to E. If *MQ* = *0* $row(s)$ consists of '$*$'s – we have to make s OD from all $r \in$ RED "by hand": Find $c \in X_-$ preventing the merge of q_r and q_s via PREVENTMERGE and a suffix e_r of c leading from q_r or q_s to a final state ($X_- \neq \emptyset$ as FINDNEXT returns $\langle \varepsilon, \varepsilon \rangle$ for *MQ* = *0* otherwise). As c should not be accepted e_r separates s from r. Add e_r to E and fill the two cells of T with differing values – note that they do not have to be correct as they are used only once by CLOSURE, and T will be updated completely just before termination.

GENMODEL is intended as a generalization of algorithms for settings where polynomial one-shot inference is possible, which also implies that it is deterministic and does not guess/backtrack. However, note that it behaves in an "intuitively appropriate" way when (polynomial) inference is not possible as well.

An information source is *non-void* for queries if *MQ* = *1*/*EQ* = *1*, for a positive sample if it is representative, for a negative sample if it is separative.

Theorem 1. *(a) Let L be the regular target. GENMODEL terminates for any input after at most $2I_L - 1$ main loop executions and returns a DFA.*
(b) For any input that includes at least two non-void information sources except for $\langle 1, 0, X_+, X_- \rangle$ with X_+ or X_- void the output is a minimal DFA for L.

See [14] for the proof. Note that Theorem 1b can also be seen from the proofs of the algorithms in [2, 6, 8]. We comment on the following three cases because to our knowledge there are no such well-studied algorithms for these settings.

$\langle 0, 1, \emptyset, X_- \rangle$: As $\langle 0, 1, X_+, \emptyset \rangle$. We build a positive sample from X_- (see above) which however may be exponential in size with respect to $|X_-|$ so that the number of MQs is not polynomial with respect to the size of the given data.

$\langle 1, 0, X_+, \emptyset \rangle$: Suppose we wanted to handle this case analogously: We would have to test state mergeability in O via EQs. For X_+ representative a positive counterexample reveals the existence of states that should be merged, a negative one of states that should not have been. When we query the result of a merge (even without repairing non-determinism by further merges) and get a positive counterexample we could either repeat the EQ and wait for a negative one but the number of positive ones may be infinite. Or we could query the next merge but when (if) we eventually get a negative one we do not know which of the previous merges was illegitimate. So this strategy is no less complex than ignoring all counterexamples and asking an EQ for the result of every possible *set* of merges, of which there are exponentially many. Therefore, since we cannot proceed as in the cases where inference is possible with a polynomial number of steps or queries this case is eclipsed from GENMODEL by the corresponding case distinctions.

$\langle 1, 0, \emptyset, X_- \rangle$: If X_- is separative negative counterexamples do not carry new information, and the number of negative counterexamples may be infinite. The set of positive counterexamples so far may not be representative so that we cannot reliably detect an illegitimate merge as there may be final states that are not even represented in the current O such that a compatibility check is too weak. If we make the merge we might have to undo it because of another positive counterexample, a situation we want to avoid. Hence we eclipse this case as well.

Note: For input with more than two non-empty sources the algorithm chooses one of the two-source options according to the priority hierarchy MQs&EQs $>$ MQ&X_+ $>$ X_+&X_-.

Conclusion. We have aimed to design GENMODEL as modular as possible as an inventory of the essential procedures in existing and conceivable polynomial one-shot regular inference algorithms of the considered kind. This may help to give clearer explanations for the interchangeability of information sources. Practically, an extended GENMODEL (see below) could be used as a template from which individual algorithms for hitherto unstudied scenarios can be instantiated. We have chosen observation tables as an abstract and flexible means to perform and document the process, from which various descriptions can be derived.

GENMODEL offers itself to be extended in several directions. We could try to generalize over the type of objects, such as trees (see [9, 6, 15, 13]), graphs, matrices, or infinite strings. Then there are other kinds of information sources which might be integratable, such as correction queries [18], active exploration [17], or distinguishing functions [11]. The third direction concerns an extension of the learned language class beyond regularity (for example by using strategies as in [10] for even linear languages, or [5] for languages recognized by DFA with infinite transition graphs) and even beyond context-freeness [10, 19]. The development of GENMODEL may be of use in the concretization of an even more general model of learning in the sense of polynomial one-shot inference as considered here – also see the very interesting current work of Clark [7].

References

[1] D. Angluin. A note on the number of queries needed to identify regular languages. *Inf. & Contr.*, 51:76–87, 1981.

[2] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[3] D. Angluin. Queries and concept learning. *Mach. L.*, 2:319–342, 1988.

[4] D. Angluin. Negative results for equivalence queries. *Mach. L.*, 5:121–150, 1990.

[5] P. Berman and R. Roos. Learning one-counter languages in polynomial time. In *SFCS*, pages 61–67, 1987.

[6] J. Besombes and J.-Y. Marion. Learning tree languages from positive examples and membership queries. In *ALT 2003*, pages 440–453, 2004.

[7] A. Clark. Three learnable models for the description of language. LATA 2010.

[8] C. de la Higuera. *Grammatical Inference: Learning Automata and Grammars.* Cambridge University Press, 2010.

[9] F. Drewes and J. Högberg. Learning a regular tree language from a teacher. In *DLT*, pages 279–291, 2003.

[10] H. Fernau. Even linear simple matrix languages: Formal language properties and grammatical inference. *Theoretical Computer Science*, 289(1):425–456, 2002.

[11] H. Fernau. Identification of function distinguishable languages. *Theoretical Computer Science*, 290(3):1679–1711, 2003.

[12] E.M. Gold. Language identification in the limit. *Inf. & Contr.*, 10(5):447–474, 1967.

[13] A. Kasprzik. A learning algorithm for multi-dimensional trees, or: Learning beyond context-freeness. In *ICGI*, pages 111–124, 2008.

[14] A. Kasprzik. Generalizing over several learning settings. Technical report, University of Trier, 2009.

[15] J. Oncina and P. Garcia. Inference of recognizable tree sets. Technical report, DSIC II/47/93, Universidad de Valencia, 1993.

[16] J. Oncina and P. Garcia. Identifying regular languages in polynomial time. volume 5 of *Machine Perception and Artificial Intelligence*, pages 99–108. World Scientific, 2002.

[17] L. Pitt. Inductive inference, DFAs, and computational complexity. In *AII*, 1989.

[18] C. Tîrnăucă. A note on the relationship between different types of correction queries. In *ICGI*, pages 213–223, 2008.

[19] R. Yoshinaka. Learning mildly context-sensitive languages with multidimensional substitutability from positive data. In *ALT*, pages 278–292, 2009.

On Two-Party Watson-Crick Computations with Limited Communication[*]

Martin Kutrib

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

Andreas Malcher

Institut für Informatik, Universität Gießen
Arndtstraße 2, 35392 Gießen, Germany
malcher@informatik.uni-giessen.de

Abstract

We investigate synchronous systems consisting of two finite automata running in opposite directions on a shared read-only input. The automata communicate by sending messages. The communication is quantitatively measured by the number of messages sent during a computation. It is shown that even the weakest non-trivial devices in question, that is, systems that are allowed to communicate constantly often only, accept non-context-free languages. We investigate the computational capacity of the devices in question and prove a strict four-level hierarchy depending on the number of messages sent. The strictness of the hierarchy is shown by means of Kolmogorov complexity. For systems with unlimited communication several properties are known to be undecidable. A question is to what extent communication has to be reduced in order to regain decidability. Here, we derive that the problems remain non-semidecidable even if the communication is reduced to a limit close to the logarithm of the length of the input. Furthermore, we show that the border between decidability and undecidability is crossed when the communication is reduced to be constant. In this case only semilinear languages can be accepted.

Introduction

Watson-Crick automata were introduced in [3] as a formal model for DNA computing. Their definition has been inspired by processes observed in nature and laboratories. The

[*]Summary of a paper presented at CIAA 2010, Winnipeg, Canada.

idea is to have an automaton with two reading heads running on either strand of a double stranded DNA-molecule. Since in nature enzymes that actually move along DNA strands may obey the biochemical direction of the single strands of the DNA sequence, so-called $5' \to 3'$ Watson-Crick automata have been introduced in [9] after an idea presented in [10]. Basically, these systems are two-head finite automata where the heads start at opposite ends of a strand and move in opposite physical directions. If the complementarity relation of the double stranded sequence is known to be one-to-one, no additional information is encoded in the second strand. Then $5' \to 3'$ Watson-Crick automata share a common input sequence.

Whenever several heads of a device are controlled by a common finite-state control, one may suppose that the heads are synchronous and autonomous finite automata that communicate their states in every time step. Here we add a new feature to $5' \to 3'$ Watson-Crick automata. It seems to be unlikely that in reality enzymes moving along and acting on DNA molecules communicate in every time step. So, we consider $5' \to 3'$ Watson-Crick systems where the components may but don't need to communicate by broadcasting messages. We are interested in the impact of communication in such devices, where the communication is quantitatively measured by the total number of messages sent during a computation. The role of message complexity in conventional one-way and two-way multi-head finite automata has been studied in [4, 5], where deep results have been obtained. According to the notations given there and in order to differentiate the notation from 'conventional' $5' \to 3'$ Watson-Crick automata we call the devices in question two-party Watson-Crick systems. Recently, deterministic Watson-Crick automata have been studied from a descriptional complexity point of view in [2].

The idea of another related approach is based on so-called parallel communicating finite automata systems which were introduced in [7]. In this model, the input is read and processed in parallel by several one-way (left-to-right) finite automata. The communication is defined in such a way that an automaton can request the current state from another automaton, and is set to that state after receiving it whereby its former state is lost. One can distinguish whether each automaton which sends its current state is reset to its initial state or not. The degree of communication in such devices was studied in [8]. Without considering the message complexity, the concept of one-way parallel communicating finite automata systems was investigated for conventional Watson-Crick automata in [1].

Definitions

We denote the set of nonnegative integers by $\mathbb{N}$. We write Σ^* for the set of all words over the finite alphabet Σ. The empty word is denoted by λ, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The reversal of a word w is denoted by w^R and for the length of w we write $|w|$. We use $\subseteq$ for inclusions and $\subset$ for strict inclusions.

A two-party Watson-Crick system is a device of two finite automata working independently and in opposite directions on a common read-only input data. The automata communicate by broadcasting messages. The transition function of a single automaton depends on its current state, the currently scanned input symbol, and the message currently received from the other automaton. Both automata work synchronously and the messages are delivered instantly. Whenever the transition function of (at least) one of the single automata is undefined the whole systems halts. The input is accepted if at least one of the automata is in an accepting state. A formal definition is as follows.

Definition 1. *A* deterministic two-party Watson-Crick system (*DPWK*) *is a construct* $\mathcal{A} = \langle \Sigma, M, \rhd, \lhd, A_1, A_2, \rangle$*, where* Σ *is the finite set of* input symbols*,* M *is the set of possible* messages*,* $\rhd \notin \Sigma$ *and* $\lhd \notin \Sigma$ *are the* left and right endmarkers*, and each* $A_i = \langle Q_i, \Sigma, \delta_i, \mu_i, q_{0,i}, F_i \rangle$*,* $i \in \{1,2\}$*, is basically a* deterministic finite automaton *with* state set Q_i*,* initial state $q_{0,i} \in Q_i$*, and set of* accepting states $F_i \subseteq Q_i$*. Additionally, each* A_i *has a* broadcast function $\mu_i : Q_i \times (\Sigma \cup \{\rhd, \lhd\}) \to M \cup \{\bot\}$ *which determines the message* to be sent*, where* $\bot \notin M$ *means* nothing to send*, and a* (partial) transition function $\delta_i : Q_i \times (\Sigma \cup \{\rhd, \lhd\}) \times (M \cup \{\bot\}) \to Q_i \times \{0, 1\}$*, where* 1 *means to move the head one square and* 0 *means to keep the head on the current square.*

The automata A_1 and A_2 are called *components* of the system $\mathcal{A}$, where the so-called *upper* component A_1 starts at the left end of the input and moves from left to right, and the *lower* component A_2 starts at the right end of the input and moves from right to left. A *configuration* of $\mathcal{A}$ is represented by a string $\rhd v_1 \overrightarrow{p} x v_2 y \underleftarrow{q} v_3 \lhd$, where $v_1 x v_2 y v_3$ is the input and it is understood that component A_1 is in state p with its head scanning symbol x, and component A_2 is in state q with its head scanning symbol y. System $\mathcal{A}$ starts with component A_1 in its initial state scanning the left endmarker and component A_2 in its initial state scanning the right endmarker. So, for input $w \in \Sigma^*$, the initial configuration is $\overrightarrow{q_{0,1}} \rhd w \lhd \underleftarrow{q_{0,2}}$. A computation of $\mathcal{A}$ is a sequence of configurations beginning with an initial configuration. One step from a configuration to its successor configuration is denoted by $\vdash$. Let $w = a_1 a_2 \cdots a_n$ be the input, $a_0 = \rhd$, and $a_{n+1} = \lhd$, then we set $a_0 \cdots a_{i-1} \overrightarrow{p} a_i \cdots a_j \underleftarrow{q} a_{j+1} \cdots a_{n+1} \vdash a_0 \cdots a_{i'-1} \overrightarrow{p_1} a_{i'} \cdots a_{j'} \underleftarrow{q_1} a_{j'+1} \cdots a_{n+1}$, for $0 \le i \le j \le n+1$, and

$$a_0 \cdots a_j \underleftarrow{q} a_{j+1} \cdots a_{i-1} \overrightarrow{p} a_i \cdots a_{n+1} \vdash a_0 \cdots a_{j'} \underleftarrow{q} a_{j'+1} \cdots a_{i'-1} \overrightarrow{p} a_{i'} \cdots a_{n+1},$$

for $0 \le j \le i \le n+1$, iff $\delta_1(p, a_i, \mu(q, a_j)) = (p_1, d_1)$ and $\delta_2(q, a_j, \mu(p, a_i)) = (q_1, d_2)$, $i' = i + d_1$ and $j' = j - d_2$. As usual we define the reflexive, transitive closure of $\vdash$ by $\vdash^*$.

A computation *halts* when the successor configuration is not defined for the current configuration. This may happen when the transition function of one component is not defined. The language $L(\mathcal{A})$ accepted by a DPWK $\mathcal{A}$ is the set of inputs $w \in \Sigma^*$ such that there is some computation beginning with the initial configuration for w and halting with at least one component being in an accepting state.

In the following, we study the impact of communication in deterministic two-party Watson-Crick systems. The communication is measured by the total number of messages sent during a computation, where it is understood that $\perp$ means no message and, thus, is not counted.

Let $f : \mathbb{N} \to \mathbb{N}$ be a mapping. If all $w \in L(\mathcal{A})$ are accepted with computations where the total number of messages sent is bounded by $f(|w|)$, then $\mathcal{A}$ is said to be *communication bounded by f*. We denote the class of DPWKs that are communication bounded by f by DPWK(f).

In general, the *family of languages accepted* by devices of type X is denoted by $\mathscr{L}(X)$.

Lemma 2. *The language $\{\, a^n b^n c^n \mid n \geq 1 \,\}$ belongs to $\mathscr{L}(DPWK(2))$.*

Computational Capacity

First we note that any DPWK can be simulated by a two-way two-head finite automaton in a straightforward manner. Therefore, the family $\mathscr{L}$(DPWK) is a proper subclass of the complexity class L. From Lemma 2 we can immediately derive the construction of a DPWK(1) that accepts the non-regular language $\{\, a^n b^n \mid n \geq 1 \,\}$. Together with the obvious inclusion of the regular languages in $\mathscr{L}$(DPWK(1)) we obtain that just one communication suffices to accept all regular languages and, additionally, also some non-regular languages, whereas two communications allow to accept non-context-free languages. However, the witness languages are semilinear. In [6] it has been shown that DPWKs that communicate in every time step accept non-semilinear languages. So, the question arises how much communication is necessary to accept a non-semilinear language.

Lemma 3.

1. *Language $L_{expo} = \{\, a^{2^0} b a^{2^2} b \cdots b a^{2^{2m}} c a^{2^{2m+1}} b \cdots b a^{2^3} b a^{2^1} \mid m \geq 1 \,\}$ belongs to $\mathscr{L}(DPWK(O(\log(n))))$.*

2. *Language $L_{poly} = \{\, a b a^5 b a^9 b \cdots b a^{4m+1} c a^{4m+3} b \cdots b a^{11} b a^7 b a^3 \mid m \geq 0 \,\}$ belongs to $\mathscr{L}(DPWK(O(\sqrt{n})))$.*

3. *Language $\{\, wcw^R \mid w \in \{0,1\}^* \,\}$ belongs to $\mathscr{L}(DPWK(O(n)))$.*

By definition we have a finite hierarchy of language classes as follows. Furthermore, the inclusions can be shown to be strict by using arguments from Kolmogorov complexity.

$$\mathscr{L}(\mathrm{DPWK}(O(1))) \subset \mathscr{L}(\mathrm{DPWK}(O(\log(n)))) \subset \mathscr{L}(\mathrm{DPWK}(O(\sqrt{n}))) \subset \mathscr{L}(\mathrm{DPWK}(O(n)))$$

Decidability Problems

Theorem 4. *The problems of testing emptiness, finiteness, infiniteness, inclusion, equivalence, regularity, and context-freeness are not semidecidable for DPWK*$(\log(n) \cdot \log\log(n))$.

Theorem 5. *For all $k \geq 0$ there are constants r and s such that any DPWK(k) can effectively be simulated by a deterministic two-way 4-counter machine which performs in every accepting computation at most r reversals of the input head and at most s turns in each counter.*

Theorem 6. *Let $k \geq 0$ be a constant. Then the problems of testing emptiness, finiteness, inclusion, and equivalence are decidable for DPWK(k).*

References

[1] Czeizler, E., Czeizler, E.: On the power of parallel communicating Watson-Crick automata systems. Theoret. Comput. Sci. **358** (2006) 142–147

[2] Czeizler, E., Czeizler, E., Kari, L., Salomaa, K.: On the descriptional complexity of Watson-Crick automata. Theoret. Comput. Sci. **410** (2009) 3250–3260

[3] Freund, R., Păun, G., Rozenberg, G., Salomaa, A.: Watson-Crick finite automata. In: DIMACS Workshop on DNA Based Computers, University of Pennsylvania (1997) 305–317

[4] Jurdziński, T., Kutyłowski, M.: Communication gap for finite memory devices. In: International Colloquium on Automata, Languages and Programming (ICALP 2001). Volume 2076 of LNCS, Springer (2001) 1052–1064

[5] Jurdziński, T., Kutyłowski, M., Loryś, K.: Multi-party finite computations. In: Computing and Combinatorics (COCOON 1999). Volume 1627 of LNCS, Springer (1999) 318–329

[6] Leupold, P., Nagy, B.: $5' \to 3'$ Watson-Crick automata with several runs. In: Non-Classical Models of Automata and Applications (NCMA 2009). Volume 256 of books@ocg.at, Austrian Computer Society (2009) 167–180

[7] Martín-Vide, C., Mateescu, A., Mitrana, V.: Parallel finite automata systems communicating by states. Int. J. Found. Comput. Sci. **13** (2002) 733–749

[8] Mitrana, V.: On the degree of communication in parallel communicating finite automata systems. J. Autom., Lang. Comb. **5** (2000) 301–314

[9] Nagy, B.: On $5' \to 3'$ sensing Watson-Crick finite automata. In: DNA Computing. Volume 4848 of LNCS, Springer (2007) 256–262

[10] Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing: New Computing Paradigms. Texts in Theoretical Computer Science. Springer (1998)

The Complexity of the Emptiness and Word Problem for Visibly Pushdown Languages (Extended Abstract)

Martin Lange
Dept. of Elect. Eng. and Computer Science
University of Kassel, Germany

1 Introduction

Context-free languages (CFL) are an important formalism for the specification of the syntax of programming or natural languages. However, the class of context-free languages lacks some important closure properties, namely determinisability and closure under complement and intersection. The class of deterministic context-free languages (DCFL) trivially admits determinisability and is therefore closed under the complement operation but still not under intersections.

Recently, a subclass of CFL and in fact also DCFL has been introduced which does possess all of these three properties and which is therefore particularly useful in the specification of runs of recursive programs: visibly pushdown languages (VPL) [1]. These are recognised by visibly pushdown automata (VPA) which are nondeterministic pushdown automata (PDA) in which the input letter determines the type of stack action that the automaton carries out when reading this letter.

VPA have very quickly become on interesting object of study because of their nice algorithmic properties. For instance, they can be used in Propositional Dynamic Logic [4] instead of regular expression whilst retaining decidability [8] which is not true for DCFL for instance [5].

Disclaimer. This extended abstract describes work in progress which is partially done together with Friedrich Otto, University of Kassel, and Markus Latte, University of Munich.

2 Visibly Pushdown Automata and Languages

A *visibly pushdown alphabet* is an alphabet Σ in the usual sense, which is then partitioned into three parts $(\Sigma_{\mathsf{push}}, \Sigma_{\mathsf{int}}, \Sigma_{\mathsf{pop}})$. These are the letters which, respectively, force a VPA to push a stack symbol, leave the stack untouched, and pop a stack symbol. The *size* of the alphabet Σ is a triple $(|\Sigma_{\mathsf{push}}|, |\Sigma_{\mathsf{int}}|, |\Sigma_{\mathsf{pop}}|)$. When comparing alphabet sizes we use the pointwise order.

A VPA is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \delta, F)$ where Q is a finite set of states, Σ as above, Γ is a finite stack alphabet containing the special bottom-of-stack symbol $\bot$, $q_0 \in Q$ is the designated initial state, and $F \subseteq Q$ is a set of final states. The transition relation δ is partitioned into three parts $\delta = \delta_{\mathsf{push}} \cup \delta_{\mathsf{int}} \cup \delta_{\mathsf{pop}}$ where

$$\begin{aligned}
\delta_{\mathsf{push}} &\subseteq Q \times \Sigma_{\mathsf{push}} \times Q \times (\Gamma \setminus \{\bot\}) \\
\delta_{\mathsf{int}} &\subseteq Q \times \Sigma_{\mathsf{int}} \times Q \\
\delta_{\mathsf{pop}} &\subseteq Q \times \Sigma_{\mathsf{push}} \times \Gamma \times Q
\end{aligned}$$

A *configuration* is a $(q, \gamma, w) \in Q \times \Gamma^* \times \Sigma^*$. The computational behaviour of the VPA $\mathcal{A}$ is explained by a relation $\vdash$ on configurations. It is defined as follows. Let $q \in Q$, $\gamma \in \Gamma^*$, $a \in \Sigma$, $w \in \Sigma^*$, $B \in \Gamma$ with $B \neq \bot$.

$$\begin{array}{rcll}
(q, \gamma, aw) &\vdash& (q', B\gamma, w) & \text{if } (q, a, q', B) \in \delta_{\mathsf{push}} \\
(q, \gamma, aw) &\vdash& (q', \gamma, w) & \text{if } (q, a, q') \in \delta_{\mathsf{int}} \\
(q, B\gamma, aw) &\vdash& (q', \gamma, w) & \text{if } (q, a, B, q') \in \delta_{\mathsf{pop}} \\
(q, \bot, aw) &\vdash& (q', \bot, w) & \text{if } (q, a, \bot, q') \in \delta_{\mathsf{pop}}
\end{array}$$

As usual, let $\vdash^*$ be the reflexive-transitive closure of $\vdash$. Also, we write $\vdash^n$ to denote the n-fold iteration of this relation. The language accepted by $\mathcal{A}$ is then

$$L(\mathcal{A}) \;:=\; \{w \in \Sigma^* \mid \exists q \in F.\ \exists \gamma \in \Gamma^*.\ (q_0, \bot, w) \vdash^* (q, \gamma, \epsilon)\}$$

It is very easy to see that the language $\{a^n b^n \mid n \in \mathbb{N}\}$ for instance is a VPL over the visibly pushdown alphabet $(\{a\}, \emptyset, \{b\})$. Equally, every Dyck language is a VPL over a suitable partitioning of the alphabet. On the other hand, $\{a^n b a^n \mid n \in \mathbb{N}\}$ is not a VPL over any alphabet since a PDA would ultimately have to perform push actions when reading the first a's, and pop actions when reading the latter a's.

Finally, consider the alphabet $(\{a\}, \{b\}, \{c\})$ and the language L_0 over this alphabet, given by the following context-free grammar.

$$S \;\to\; \epsilon \mid bS \mid aScS$$

Again, it is not difficult to see that this is a VPL.

3 The Emptiness Problem

Since every VPA is also a PDA, upper bounds on decision problems for PDA trivially carry over to VPA. For instance, the emptiness problem or the membership problem are solvable in deterministic polynomial time [7]. In case of CFL, this is also known to be optimal: the emptiness problem for CFL is P-hard. This result seems to be folklore and is indeed easy to prove by a reduction from the alternating graph reachability problem which is an abstracted version of the word problem for alternating, logarithmic space bounded Turing Machines. This is one of the standard P-hard decision problems [2].

A natural question to ask is whether or not the emptiness problem for VPA is P-hard already. The answer is "yes". It is possible to reduce the alternating graph reachability problem to the emptiness problem for VPA (instead of PDA).

Proposition 1. *The emptiness problem for VPL is P-complete.*

The lower bound already holds for visibly pushdown alphabets of size $(1, 0, 1)$ [6].

4 The Universal Word Problem

Another interesting problem is the (universal) word problem: given a VPA $\mathcal{A}$ and a word w over the same visibly pushdown alphabet, is $w \in L(\mathcal{A})$? For PDA this is simply a generalisation of the emptiness problem. Context-free languages are closed under homomorphisms with logspace reductions, therefore this problem reduces to the question whether or not the empty word ϵ belongs to a given context-free language. However, if the description of the CFL does not use any alphabet symbols then this is the same as the emptiness problem.

Visibly pushdown languages are not closed under homomorphisms though, and therefore the universal word problem requires an explicit study. It is unlikely to be P-hard because it can be shown to belong to LOGCFL.

Proposition 2 (Otto)**.** *The universal word problem for VPL is in LOGCFL.*

Proof. It is known that a language belongs to LOGCFL iff it can be recognised by a nondeterministic auxiliary pushdown automaton using a logarithmic worktape and polynomial time [10]. Such a machine can easily decide the word problem for a given VPA $\mathcal{A}$ and an input w. It simulates $\mathcal{A}$ on w using the worktape in order to memorise the current position in the word and the current state of the VPA $\mathcal{A}$, and its stack in order to simulate $\mathcal{A}$'s stack. Nondeterminism is used to follow the nondeterministic choices made by $\mathcal{A}$. Polynomial runtime is guaranteed by the fact that $\mathcal{A}$ accepts w iff it accepts w after exactly $|w|$ many steps. □

Note that – even though VPL can be recognised by deterministic VPA – this does not necessarily yield inclusion of the universal word problem in LOGDCFL. The determinisation procedure for VPA incurs an exponential blow-up (for every NFA is also a VPA). Thus, the universal word problem cannot necessarily be solved by a deterministic auxiliary pushdown automaton using logarithmic space only.

On the other hand, the problem is clearly NLOGSPACE-hard which is inherited from the word problem for nondeterministic finite automata. We will quickly give some intuitive evidence of why the problem is unlikely to be NLOGSPACE-complete or LOGCFL-complete.

The problem is in NLOGSPACE for the subclass of VPA which use a singleton stack alphabet. Since the stack cannot grow beyond the length of the word, a stack over a singleton alphabet can be encoded binarily as a number of at most logarithmic length. Then the word problem reduces to the reachability problem in a graph made up of the product of the input word, the VPA's state space, and a number encoding the stack height. On the other hand, NLOGSPACE-completeness would entail that the word problem could always logarithmically be reduced to the word problem for a VPA over a singleton stack alphabet. This seems very unlikely, though.

Now for LOGCFL-hardness. There is another characterisation: a language belongs to LOGCFL iff it can be recognised by an alternating Turing Machine with a logarithmic worktape and polynomially sized computation trees [9]. This characterisation is indeed very close to the proof of P-hardness of the emptiness problem. This is not surprising since one characterises the complexity class P if the restriction on the size of the computation trees is dropped [9]. The problem in proving LOGCFL-hardness using this characterisation is then to construct deterministically a VPA and a word from such a given alternating Turing Machine. It is natural and possible to construct the VPA such that it searches for an accepting computation tree using its stack to back-track on branches in the tree. However, the word to be recognised would have to encode the structure of the computation tree, namely a depth-first traversal of the entire tree using push- and pop-symbols. But note that all that is known about the tree is its overall size. Thus, it would be possible to construct a word of polynomial length if the exact structure of the tree was known. Since it is unknown, it seems like one can only construct a word of exponential length which encodes an (exponential) overapproximation of this computation tree.

A possibility to characterise the complexity more precisely is to introduce a new complexity class for which the problem under consideration is trivially hard.

Definition 1. *A language $L \subseteq \Sigma^*$ belongs to the class LOGVPL iff there is a logspace computable function $f : \Sigma^* \to \Delta^*$ and a VPA $\mathcal{A}$ over the visibly pushdown alphabet Δ s.t. for all $w \in \Sigma^*$ we have $w \in L$ iff $f(w) \in L(\mathcal{A})$.*

This class clearly sits between the two classes mentioned above. Also, we can now use determinisability of VPA and put this new class into the complexity hierarchy as follows.

Proposition 3. *NLOGSPACE $\subseteq$ LOGVPL $\subseteq$ LOGDCFL $\subseteq$ LOGCFL.*

The third inclusion is trivial and of course well-known. The first inclusion follows from the fact that the word problem for NFA is NLOGSPACE-hard under logspace reductions, and every NFA is also a VPA. Thus, this inclusion is almost equally trivial. The second inclusion is a consequence of the fact that the visibly or deterministic pushdown language that some other language is reduced to in the definition of LOGVPL and LOGDCFL, is existentially quantified. It need not be constructed by the reduction. Thus, if a language can be reduced in logarithmic space to a VPL, then it can also be reduced in logarithmic space to a DCFL. In other words: VPL $\subseteq$ DCFL, therefore LOGVPL $\subseteq$ LOGDCFL.

Furthermore, LOGVPL-hardness of the word problem for VPA is easy, too.

Proposition 4. *The word problem for VPA is LOGVPL-hard.*

Proof. Suppose $L \in$ LOGVPL. By definition, there is a logspace computable f, and a VPA $\mathcal{A}$ s.t. $w \in L$ iff $f(w) \in L(\mathcal{A})$. Now consider the function f' defined by $f'(w) = \langle f(w), enc(\mathcal{A})\rangle$ for some suitable encoding function *enc*. Since the second component of this pair is just a constant, f' is also logspace computable. But then f' realises a logspace reduction to the word problem for VPA which is, hence, LOGVPL-hard. □

The question of whether or not the word problem is complete for LOGVPL boils down to finding – in some sense – a hardest visibly pushdown language in analogy to the ones for CFL [3].

Another task is of course to find other complete problems or, equivalently, other characterisations of this class.

References

[1] R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. 36th Ann. ACM Symp. on Theory of Computing, STOC'04*, pages 202–211, New York, 2004. ACM Press.

[2] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[3] S. Greibach. The hardest context-free language. *SIAM Journal on Computing*, 2:304–310, 1973.

[4] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* MIT Press, 2000.

[5] D. Harel, A. Pnueli, and J. Stavi. Propositional dynamic logic of nonregular programs. *Journal of Computer and System Sciences*, 26(2):222–243, 1983.

[6] M. Lange. P-hardness of the emptiness problem for visibly pushdown languages. Submitted for publication, 2010.

[7] M. Lange and H. Leiß. To CNF or not to CNF? An efficient yet presentable version of the CYK algorithm. *Informatica Didactica*, 8, 2009.

[8] C. Löding and O. Serre. Propositional dynamic logic with recursive programs. In *Proc. 9th Int. Conf. on Foundations of Software Science and Computation Structures, FOSSACS'06*, volume 3921 of *LNCS*, pages 292–306. Springer, 2006.

[9] W. L. Ruzzo. Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21(2):218–235, 1980.

[10] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *JACM*, 25(3):405–414, 1978.

[11] H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43(2):380–404, 1991.

Algorithmic Results on Hairpin Completion and Lengthening [*]

Florin Manea
Otto-von-Guericke-University Magdeburg,
Faculty of Computer Science
PSF 4120, D-39016 Magdeburg, Germany.
flmanea@fmi.unibuc.ro

This paper reviews a series of works started with [2] (based on some ideas from [1]), where, inspired by the DNA manipulation, a new formal operation on words, called hairpin completion, was introduced. The initial work was followed by a series of related papers ([10, 11, 13, 14, 6, 12]), where the algorithmic properties of the hairpin completion, as well as its inverse operation, the hairpin reduction, and two other variants of this operation (bounded hairpin completion and hairpin lengthening) were further investigated.

Single-stranded DNA molecules (ssDNA) are composed by nucleotides which differ from each other by their bases: A (adenine), G (guanine), C (cytosine), and T (thymine). Therefore each ssDNA may be viewed as a finite string over the four-letter alphabet $\{A, C, G, T\}$. Two single strands can bind to each other, forming the secondary structure of DNA, if they are pairwise Watson-Crick complementary: A is complementary to T, and C to G. The binding of two strands is also called *annealing*. Similarly, RNA molecules are chains of nucleotides having the bases A, G, C and U (uracyl), with A complementary to U, and C to G. An intramolecular base pairing, known as *hairpin*, is a pattern that can occur in single-stranded DNA or RNA molecules. Hairpin or hairpin-free structures have numerous applications to DNA-computing and molecular genetics. In many DNA-based algorithms, these DNA molecules cannot be used in the subsequent computations. Therefore, it is important to design methods for constructing sets of DNA sequences which are unlikely to lead to such "bad" hybridizations. This problem was considered in a series of papers, see e.g. [15, 4, 5, 8] and the references therein.

In [2] a new formal operation on words is introduced, namely the *hairpin completion*. It consists of three biological principles. Besides the Watson-Crick complementarity and

[*]This work was supported by the *Alexander von Humboldt Foundation*. The author is on leave from the Faculty of Mathematics and Computer Science, University of Bucharest.

annealing, the third biological phenomenon is that of *lengthening DNA by polymerases*. In our case the phenomenon produces a new molecule as follows: one starts with a hairpin - which is, here, a single-stranded molecule, such that one of its ends (a prefix or, respectively, a suffix) is annealed to another part of itself by Watson-Crick complementarity -, and a *polymerization buffer* with many copies of the four basic nucleotides. Then, the initial hairpin is prolongated by polymerases (thus adding a suffix or, respectively, a prefix), until a complete hairpin structure is obtained (the beginning of the strand is annealed to the end of the strand).

Further, two variants of the hairpin completion were considered, as they seem more appropriate for practical implementation: hairpin lengthening and bounded hairpin completion. The first variant concerns the prolongation of a strand which forms a hairpin, similarly to the process described above, but not necessarily until a complete hairpin structure is obtained; the main motivation in introducing this operation is that, in practice, it may be a difficult task to control the completion of a hairpin structure, and it seems easier to model only the case when such a structure is extended. The second variant, respectively the bounded hairpin completion, assumes that the length of the prefix and suffix added by the hairpin completion is bounded by a constant.

As most of the unary operations on words, the hairpin completion/ lengthening operations can be extended canonically to operations on languages, and, then, their iterated version can be defined. Nevertheless, the iterated hairpin completion (lengthening) has also a biological motivation. Since such an operation can be seen as an operation by which one single stranded molecule evolves into a new single stranded molecule, it is natural to consider the situation when multiple evolution steps occur, thus the initial word is transformed by multiple hairpin completion/ lengthening steps. In this context several natural algorithmic questions appear: given two words, can we decide if the smaller one evolved (by iterated hairpin completion/ lengthening) into the longer one, or, given two words, can we decide if they both evolved from the same word - called usually common ancestor. Moreover, one can be also interested in finding how many steps are needed to transform one word into another by iterated application of hairpin completion, or, if we consider our biological motivation, how many evolution steps are needed to transform a single stranded molecule into another. In this way, the *hairpin completion distance* between two words was defined as the minimum number of times we must iterate the hairpin completion operation, starting from one of the two words, in order to obtain the other (of course, similar definitions were given in the case of bounded hairpin completion and hairpin lengthening). Further, one can also be interested in finding, for two words, a common ancestor that minimizes, or respectively maximizes, the sum of the hairpin distances to the two words

- called minimum, respectively maximum, distance common ancestor. Such common ancestors have also a biological motivation: they can be seen as extreme common ancestors of the given single-stranded molecules. The maximum distance common ancestor of two given molecules can be seen as the simplest (less evolved) molecule from which the given molecules could have evolved; on the other hand, the minimum distance common ancestor is the most complex (most evolved) molecule from which the both given molecules could have evolved.

In the following we review some of the results regarding the problems mentioned above.

1 Preliminaries

Let Ω be a "superalphabet", that is an infinite set such that any alphabet considered in this paper is a subset of Ω. In other words, Ω is the *universe* of the languages in this paper, i.e., all words and languages are over alphabets that are subsets of Ω. An *involution* over a set S is a bijective mapping $\sigma : S \longrightarrow S$ such that $\sigma = \sigma^{-1}$. Any involution σ on Ω such that $\sigma(a) \neq a$ for all $a \in \Omega$ is said to be, in this paper's context, a *Watson-Crick involution* (we prefer this terminology since the hairpin lengthening defined later is inspired by the DNA lengthening by polymerases, where the Watson-Crick complementarity plays an important role). Let $\bar{\cdot}$ be a Watson-Crick involution fixed for the rest of the paper. The Watson-Crick involution is extended to a morphism from Ω^* to Ω^* in the usual way. We say that the letters a and $\overline{a}$ are complementary to each other. For an alphabet V, we set $\overline{V} = \{\overline{a} \mid a \in V\}$. Note that V and $\overline{V}$ could be disjoint or intersect or be equal. We denote by $(\cdot)^R$ the mapping defined by ${}^R : V^* \longrightarrow V^*$, $(a_1a_2 \ldots a_n)^R = a_n \ldots a_2a_1$. Note that ^{R} is an involution and an *anti-morphism* ($(xy)^R = y^Rx^R$ for all $x, y \in V^*$). Note also that the two mappings $\bar{\cdot}$ and $\cdot^R$ commute, namely, for any word x, $(\overline{x})^R = \overline{x^R}$ holds.

Let V be an alphabet. For any $w \in V^+$ we define the *k-hairpin completion* of w, denoted by $(w \rightarrow_k)$, for some $k \geq 1$, as follows:

$$\begin{aligned}
w \rightharpoonup_k &= \{\overline{\gamma^R}w | w = \alpha\beta\overline{\alpha^R}\gamma, |\alpha| = k,\ \alpha, \beta \in V^+, \gamma \in V^*\} \\
w \rightharpoondown_k &= \{w\overline{\gamma^R} | w = \gamma\alpha\beta\overline{\alpha^R}, |\alpha| = k,\ \alpha, \beta \in V^+, \gamma \in V^*\} \\
w \rightarrow_k &= w \rightharpoonup_k \cup w \rightharpoondown_k
\end{aligned}$$

The *hairpin completion* of w is defined by: $(w \rightarrow) = \bigcup_{k\geq1}(w \rightarrow_k)$. This operation is schematically illustrated in Figure 1. Clearly, $(w \rightarrow_{k+1}) \subseteq (w \rightarrow_k)$ for any $w \in V^+$ and $k \geq 1$, hence $(w \rightarrow) = (w \rightarrow_1)$. The hairpin completion is naturally extended to languages by $(L \rightarrow_k) = \bigcup_{w\in L}(w \rightarrow_k)$.

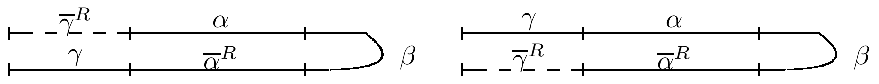


Fig. 1: Hairpin completion

For any $w \in V^+$ we define the *k-hairpin lengthening* of w, denoted by $HL_k(w)$, for some $k \geq 1$, as follows:

$$\begin{aligned}
HLP_k(w) &= \{\overline{\delta^R}w | w = \alpha\beta\overline{\alpha^R}\gamma, |\alpha| = k,\ \alpha, \beta, \gamma \in V^+ \text{ and } \delta \text{ is a prefix of } \gamma\},\\
HLS_k(w) &= \{w\overline{\delta^R} | w = \gamma\alpha\beta\overline{\alpha^R}, |\alpha| = k,\ \alpha, \beta, \gamma \in V^+ \text{ and } \delta \text{ is a suffix of } \gamma\},\\
HL_k(w) &= HLP_k(w) \cup HLS_k(w).
\end{aligned}$$

The *hairpin lengthening* of w is defined by $HL(w) = \bigcup_{k \geq 1} HL_k(w)$. Clearly, $HL_{k+1}(w) \subseteq HL_k(w)$ for any $w \in V^+$ and $k \geq 1$. The hairpin lengthening is naturally extended to languages by $HL_k(L) = \bigcup_{w \in L} HL_k(w)$.

This operation is schematically illustrated in Figure 2.

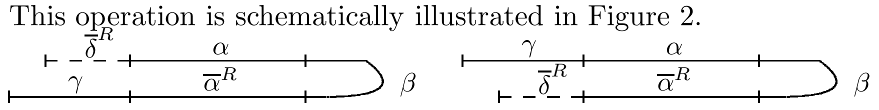


Fig. 2: Hairpin lengthening

Finally, for any $w \in V^+$ we define the p-bounded k-hairpin completion of w, denoted by $pHC_k(w)$, for some $k, p \geq 1$, as follows:

$$\begin{aligned}
pHC \leftarrow_k (w) &= \{\overline{\gamma^R}w | w = \alpha\beta\overline{\alpha^R}\gamma, |\alpha| = k,\ \alpha, \beta \in V^+, \gamma \in V^*, |\gamma| \leq p\}\\
pHC \rightarrow_k (w) &= \{w\overline{\gamma^R} | w = \gamma\alpha\beta\overline{\alpha^R}, |\alpha| = k,\ \alpha, \beta \in V^+, \gamma \in V^*, |\gamma| \leq p\}\\
pHC_k(w) &= pHC \leftarrow_k (w) \cup pHC \rightarrow_k (w).
\end{aligned}$$

The p-bounded k-hairpin completion is naturally extended to languages by $pHC_k(L) = \bigcup_{w \in L} pHC_k(w)$.

The iterated version of the hairpin completion is defined as usual by:

$$w(\rightarrow_k)^0 = \{w\}, w(\rightarrow_k)^{n+1} = (w(\rightarrow_k)^n) \rightarrow_k,\ w(\rightarrow_k)^* = \bigcup_{n \geq 0} w(\rightarrow_k)^n$$

and $L(\rightarrow_k)^* = \bigcup_{w \in L} w(\rightarrow_k)^*$.

The iterated version of the hairpin lengthening and of the p-bounded k-hairpin completion are defined similarly.

2 Results

In the following we present the main algorithmic results regarding the above defined operations

First, the non-iterated version of the operations is analyzed.

Theorem 1. *([10, 6, 12]) For every $k \geq 1$ and every language L recognizable in $\mathcal{O}(f(n))$ time, the k-hairpin completion, as well as the k-hairpin lengthening and the p-bounded k-hairpin completion, of L are recognizable in $\mathcal{O}(nf(n))$ time. If L is regular (context-free), $L(\rightarrow_k)$, $HL_k(L)$ and $pHC_k(L)$ are recognizable in $\mathcal{O}(n)$ (respectively, $\mathcal{O}(n^3)$) time.*

We recall that the hairpin completion and the hairpin lengthening of a regular (context-free) language are not necessarily regular (context-free), while the bounded hairpin completion of a regular (context-free) language is always regular (context-free). Also, the algorithms used to show the above results are based on different strategies, depending on the operation.

In the iterated case one can show that:

Theorem 2. *([10, 12, 9]) For every $k \geq 1$ and every language L recognizable in $\mathcal{O}(f(n))$ time, the iterated k-hairpin completion (or lengthening) of L is recognizable in $\mathcal{O}(n^2 f(n))$ time. Also, the iterated k-hairpin completion (or lengthening) of L can be recognized in $\mathcal{O}(\max(f(n), n^2))$, provided that all the subwords of a word w (with $|w| = n$) contained in L can be found also in $\mathcal{O}(f(n))$ time; this is the case of context-free languages (if we take $f(n) = n^3$) and regular languages (for $f(n) = n^2$). The iterated p-bounded k-hairpin completion of a regular (respectively, context-free) language L is regular (respectively, context-free), thus it can be recognized in $\mathcal{O}(n)$ time (respectively, $\mathcal{O}(n^3)$).*

Although the results are similar for all the three operations, there are major differences between the algorithms used to show this in the case of the completion operations (both bounded and unbounded) and in the case of the lengthening operations. Also, note that, different from the case of bounded completion, the classes of regular and context-free languages are not closed to hairpin completion (it is not known what happens in the case of lengthening).

The *k-hairpin completion distance* between two words x and y is defined as the minimal number of hairpin completions which can be applied either to x in order to obtain y or to y in order to obtain x. If none of them can be obtained from the other by iterated hairpin completion, then the distance is ∞. Formally, the k-hairpin completion distance between x and y, denoted by $HCD_k(x, y)$, is defined by:

$$HCD_k(x,y) = \begin{cases} \min\{p \mid x \in y(\rightarrow_k)^p \text{ or } y \in x(\rightarrow_k)^p\}, \\ \infty, \text{ if neither } x \in y(\rightarrow_k)^* \text{ nor } y \in x(\rightarrow_k)^* \end{cases}$$

Similarly, one can define the *k-hairpin lengthening distance* (denoted HLD_k) and the *p-bounded k-hairpin completion distance* (denoted $pHCD_k$) between two words.

We stress from the very beginning that the functions defined above, applied on pairs of words, are not distance functions in the strict mathematical sense, since they do not necessarily verify the triangle inequality. Rather, they can be seen as similarity measures between strings, or, if we consider our biological motivation, as measures that tell us how many evolution steps are needed to transform a string into the other. However, we prefer to call them distances for sake of uniformity, as many similar measures are called distances in the literature.

Theorem 3. *([14, 12, 6]) Let x and w be two words, and $n = \max(|x|, |w|)$. The k-hairpin completion distance between x and w can be computed in $\mathcal{O}(n^2 \log n)$, using $\mathcal{O}(n^2)$ space. The k-hairpin lengthening distance between x and w can be computed in $\mathcal{O}(n^2)$, using $\mathcal{O}(n^2)$ space. The p-bounded k-hairpin completion distance x and w can be computed in $\mathcal{O}(n^2 \log p)$ time and space.*

Note, once again, that the algorithms used to obtain these bounds are different, in each case. Although they are all based on a similar dynamic programming strategy, their efficient implementation depends on particular strategies: the usage of efficient data structures (segment trees for the general completion and range minimum queries for bounded completion) or greedy arguments (in the case of lengthening).

A word w is called k-hairpin completion ancestor of two words x and y if $\{x, y\} \subseteq w(\rightarrow_k)^*$. A word w is called minimum distance common k-hairpin completion ancestor of two words x and y if w is a k-hairpin completion ancestor of x and y, and $HCD_k(w, x) + HCD_k(w, y) \leq HCD_k(w', x) + HCD_k(w', y)$, for all w' such that $\{x, y\} \subseteq w'(\rightarrow_k)^*$. Similarly, one can define the maximum distance common k-hairpin completion ancestors of two words. The above notions can be defined for hairpin lengthening and bounded hairpin completion.

Theorem 4. *([14]) Let x and w be two words, and $n = \max(|x|, |w|)$. A minimum (maximum) distance common k-hairpin completion (k-hairpin lengthening, p-bounded k-hairpin completion) ancestor of x and w can be computed in time $\mathcal{O}(n^2 \log n)$, using $\mathcal{O}(n^2)$ space. A common k-hairpin completion (k-hairpin lengthening, p-bounded k-hairpin completion) ancestor of the words x and w can be computed in $\mathcal{O}(n^2)$ time and space.*

References

[1] P. Bottoni, A. Labella, V. Manca, V. Mitrana. Superposition based on Watson-Crick-like complementarity. *Theory of Computing Systems* 39(4):503–524, 2006.

[2] D. Cheptea, C. Martin-Vide, V. Mitrana. A new operation on words suggested by DNA biochemistry: hairpin completion, *Proc. Transgressive Computing*, 216–228, 2006.

[3] T.H. Cormen, C.E. Leiserson, R.R Rivest. *Introduction to Algorithms*, MIT Press, 1990.

[4] R. Deaton, R. Murphy, M. Garzon, D.R. Franceschetti, S.E. Stevens. Good encodings for DNA-based solutions to combinatorial problems, *Proc. of DNA-based computers II*, DIMACS Series, vol. 44:247–258, 1998.

[5] M. Garzon, R. Deaton, L.F. Nino, S.E. Stevens Jr., M. Wittner. Genome encoding for DNA computing, *Proc. Third Genetic Programming Conference*, Madison, MI, 684–690, 1998.

[6] M. Ito, P. Leupold, F. Manea, V. Mitrana. Bounded Hairpin Completion, *Inform. and Comput.*, to appear.

[7] D.E. Knuth, J.H. Morris, V.R. Pratt. Fast pattern matching in strings, *SIAM Journal of Computing* 6:323–350, 1977.

[8] L. Kari, S. Konstantinidis, P. Sosik, G. Thierrin. On hairpin-free words and languages, *Proc. Developments in Language Theory 2005*, LNCS 3572:296-307, 2005.

[9] S. Kopecki. On the Iterated Hairpin Completion. *Developments in Language Theory 2010*, LNCS 6224:438-439, 2010.

[10] F. Manea, C. Martín-Vide, V. Mitrana. On some algorithmic problems regarding the hairpin completion, *Discr. App. Math.* 157(9):2143-2152, 2009.

[11] F. Manea, V. Mitrana. Hairpin completion versus hairpin reduction, *Computation in Europe CiE 2007*, LNCS 4497:532-541, 2007.

[12] F. Manea, C. Martín-Vide, V. Mitrana. Hairpin lengthening, *Computation in Europe CiE 2010*, LNCS 6158:296–306, 2010.

[13] F. Manea, V. Mitrana, T. Yokomori. Two complementary operations inspired by the DNA hairpin formation: completion and reduction, *Theor. Comput. Sci.* 410(4-5):417-425, 2009.

[14] F. Manea. A series of algorithmic results related to the iterated hairpin completion. Submitted.

[15] K. Sakamoto, H. Gouzu, K. Komiya, D. Kiga, S. Yokoyama, T. Yokomori, and M. Hagiya. Molecular computation by DNA hairpin formation, *Science* 288:1223–1226, 2000.

Common Supersequences with Minimum Scope Coincidence Degree

Daniel Reidenbach and Markus L. Schmid [*]

Department of Computer Science, Loughborough University,
Loughborough, Leicestershire, LE11 3TU, United Kingdom
{D.Reidenbach,M.Schmid}@lboro.ac.uk

1 Introduction

The *Common Supersequence Problem* is to construct a word w for a given finite set of words $\{u_1, u_2, \dots, u_n\}$ such that w contains every u_i, $1 \leq i \leq n$, as a (scattered) subword. A word u is a scattered subword of a word v if we can obtain u by deleting symbols from v, e. g. `ab` is a scattered subword of `bcadcbd`. The word $w = u_1 \cdot u_2 \cdot \dots \cdot u_n$ is obviously a trivial and uninteresting solution to that problem, and therefore one usually requires w to meet certain additional constraints. The most prominent example of the Common Supersequence Problem is the variant where we look for the *shortest* word w that contains every u_i, $1 \leq i \leq n$, which is called the *Shortest Common Supersequence Problem.* This NP-complete problem has been thoroughly investigated (see, e. g., [6, 2, 4, 1]) and is a topic of ongoing research (see, e. g., [3]).

The variant of the Common Supersequence Problem to be introduced in this paper is mainly motivated by a problem of scheduling memory accesses. Let us assume we have k processes and m values stored in memory cells, and all these processes need to access the stored values at some points during their execution. A process does not necessarily need all the m values at the same time, so a process might get along with less than m memory cells by, for example, first using a memory cell for a value x and then, as soon as x is not needed anymore, using the same cell for another, and previously unneeded, value y. As an example, we assume that process w_1 uses the values `a`, `b` and `c` in the order `abacbc`. This process only needs two memory cells: In the first cell, `b` is permanently stored, and the second cell first stores `a` until it is not required anymore and then stores

[*]Corresponding author.

value c. This is possible, since the part of w_1 where a occurs and the part where c occurs can be completely separated from each other. If we now assume that the k processes cannot access the shared memory simultaneously, then the question arises how we can sequentially arrange all memory accesses such that a minimum overall number of memory cells is required. For example, if we assume that, in addition to process $w_1 =$ abacbc, there is another process $w_2 :=$ abc, then we can of course first execute w_1 and afterwards w_2, which results in the memory access sequence abacbcabc. It is easy to see that this requires a memory cell for each value a, b and c. On the other hand, we can first execute aba of process w_1, then process $w_2 =$ abc, and finally the remaining part cbc of w_1. This results in abaabccbc, which allows us to use a single memory cell for both values a and c as before.

This scheduling problem can directly be formalised as a Common Supersequence Problem. To this end, we merely have to interpret each of the k processes as a word over an alphabet of cardinality m, where m is the number of different values to be stored. Hence, our problem of finding the best way to organise the memory accesses of all processes directly translates into a Common Supersequence Problem. Unfortunately, even for $k = 2$, there is an exponential number of possible ways to schedule the memory accesses. However, we can present an algorithm solving this problem for arbitrary input words and a fixed alphabet size in polynomial time.

Our practical motivation and the definition of this problem result from an application of a new automata model with two input heads [5]. In our application, these two input heads need to travel over factors of the input word; to this end, they need to know the lengths of these factors. Thus, each input head movement can be interpreted as a process that needs to access lengths of factors in a certain order. Within the scope of [5], the overall number of values that need to be stored simultaneously does not only affect the memory usage of the automaton, but it also has a significant impact on the runtime of its computations. Thus, our variant of the Common Supersequence Problem is crucial in this context.

Although we consider this nontrivial problem fundamental and believe that it might occur in other practical situations as well, we are not aware of any related literature.

2 Basic Definitions

In the following, let Σ be a finite alphabet. A *word* (*over* Σ) is a finite sequence of symbols from Σ, and ε stands for the *empty word*. The symbol Σ^+ denotes the set of all nonempty words over Σ, and $\Sigma^* := \Sigma^+ \cup \{\varepsilon\}$. For the *concatenation* of two strings w_1, w_2 we write

$w_1 \cdot w_2$ or simply $w_1 w_2$. We say that a string $v \in \Sigma^*$ is a *factor* of a string $w \in \Sigma^*$ if there are $u_1, u_2 \in \Sigma^*$ such that $w = u_1 \cdot v \cdot u_2$. If $u_1 = \varepsilon$ (or $u_2 = \varepsilon$), then v is a *prefix* of w (or a *suffix*, respectively). The notation $|K|$ stands for the size of a set K or the length of a string K. The term $\mathrm{alph}(w)$ denotes the set of all symbols occurring in w and, for each $\mathtt{a} \in \mathrm{alph}(\mathtt{w})$, $|w|_{\mathtt{a}}$ refers to the number of occurrences of $\mathtt{a}$ in w. If we wish to refer to the symbol at a certain position j, $1 \leq j \leq n$, in a word $w = \mathtt{a_1} \cdot \mathtt{a_2} \cdot \ldots \cdot \mathtt{a_n}$, $\mathtt{a_i} \in \Sigma$, $1 \leq i \leq n$, we use $w[j] := \mathtt{a_j}$. Furthermore, for each j, j', $1 \leq j < j' \leq |w|$, let $w[j, j'] := \mathtt{a_j} \cdot \mathtt{a_{j+1}} \cdot \ldots \cdot \mathtt{a_{j'}}$ and $w[j, -] := w[j, |w|]$. In case that $j > |w|$, we define $w[j, -] = \varepsilon$.

In order to model the Common Supersequence Problem relevant to this paper, we use the notion of a shuffle. The *shuffle operation*, denoted by $\sqcup$, is a binary operation on words, defined inductively by

- $u \sqcup \varepsilon = \varepsilon \sqcup u = \{u\}$, for each $u \in \Sigma^*$,
- $\mathtt{a} \cdot \mathtt{u} \sqcup \mathtt{b} \cdot \mathtt{v} = \mathtt{a} \cdot (\mathtt{u} \sqcup \mathtt{b} \cdot \mathtt{v}) \cup \mathtt{b} \cdot (\mathtt{a} \cdot \mathtt{u} \sqcup \mathtt{v})$, for all $u, v \in \Sigma^*$ and $\mathtt{a}, \mathtt{b} \in \Sigma$.

We extend the definition of the shuffle operation to the case of more than two words in the obvious way. Furthermore, for arbitrary words $w_1, w_2, \ldots, w_k \in \Sigma^*$, we call $\Gamma := w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ the *shuffle* of $w_1 \ldots, w_k$ and each word $w \in \Gamma$ is a *shuffle word* of $w_1 \ldots, w_k$. For example, $\mathtt{bcaabac} \in \mathtt{abc} \sqcup \mathtt{ba} \sqcup \mathtt{ca}$.

Finally, we introduce a special property of words that is important for our central problem. For an arbitrary $w \in \Sigma^*$ and any $\mathtt{b} \in \mathrm{alph}(\mathtt{w})$ let l, r, $1 \leq l, r \leq |w|$, be chosen such that $w[l] = w[r] = \mathtt{b}$ and there exists no k, $k < l$, with $w[k] = \mathtt{b}$ and no k', $r < k'$, with $w[k'] = \mathtt{b}$. Then the *scope of* $\mathtt{b}$ *in* w ($\mathrm{sc}_w(\mathtt{b})$ for short) is defined by $\mathrm{sc}_w(\mathtt{b}) := (\mathtt{l}, \mathtt{r})$. Note, that in the case that for some word w we have $w[j] = \mathtt{b}$ and $|w|_{\mathtt{b}} = 1$, the scope of $\mathtt{b}$ in w is (j, j). Now we are ready to define the so called *scope coincidence degree*: Let $w \in \Sigma^*$ be an arbitrary word and, for each i, $1 \leq i \leq |w|$, let

$$\mathrm{scd}_i(w) := |\{\mathtt{b} \in \Sigma \mid \mathtt{b} \neq \mathtt{w}[\mathtt{i}], \mathrm{sc}_{\mathtt{w}}(\mathtt{b}) = (\mathtt{l}, \mathtt{r}) \text{ and } \mathtt{l} < \mathtt{i} < \mathtt{r}\}| \ .$$

We call $\mathrm{scd}_i(w)$ the scope coincidence degree of position i in w. Furthermore, the scope coincidence degree of the word w is defined by $\mathrm{scd}(w) := \max\{\mathrm{scd}_i(w) \mid 1 \leq i \leq |w|\}$. We now illustrate this definition with a brief example.

Example 1. *Let* $\Sigma := \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}\}$. *We consider the following word over* Σ, $w := \mathtt{acacbbdeabcedefdeff} \in \Sigma^*$. *It can easily be verified that* $\mathrm{scd}_8(w) = \mathrm{scd}_9(w) = 4$ *and* $\mathrm{scd}_i(w) < 4$ *if* $i \notin \{8, 9\}$. *Hence,* $\mathrm{scd}(w) = 4$.

We are now ready to state our central problem CSminSCD$_\Sigma$, which is the problem of finding a shuffle word $w \in w_1 \sqcup \ldots \sqcup w_k$, for arbitrary $w_1, w_2, \ldots, w_k \in \Sigma^*$, with minimum scope coincidence degree.

3 Examples

In the following, we illustrate the problem CSminSCD$_\Sigma$ with the help of a few examples. For all of these examples we use the alphabet $\Sigma := \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}, \mathtt{h}\}$. First, we consider the following two words over Σ: $w_1 := \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c} \cdot \mathtt{d} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{e} \cdot \mathtt{f}$ and $w_2 := \mathtt{g} \cdot \mathtt{h} \cdot \mathtt{a}$. We note that $\mathrm{scd}(w_1) = 2$ and $\mathrm{scd}(w_2) = 0$ which implies that $\mathrm{scd}(w') \geq 2$ for all $w' \in w_1 \sqcup w_2$. Furthermore, all symbols in w_2, except $\mathtt{a}$, do not occur in w_1, so in a shuffle word $w' \in w_1 \sqcup w_2$ the position of the symbol $\mathtt{a}$ from w_2 is crucial for $\mathrm{scd}(w')$. A shuffle word $w' \in w_1 \sqcup w_2$ that is minimal with respect to the scope coincidence degree can be easily constructed. To this end, we consider the numbers $\mathrm{scd}_i(w_1)$, $1 \leq i \leq 8$, and observe that $\mathrm{scd}_i(w_1) \leq 1$, $i \neq 4$, and $\mathrm{scd}_4(w_1) = 2$. If we shuffle word w_2 and w_1, the only symbol from w_2 that can cause one of these numbers to increase is the symbol $\mathtt{a}$. Consequently, to obtain a minimal, shuffle word it is sufficient to make sure that $\mathrm{scd}_4(w_1)$ does not increase. So we only have to assure that the $\mathtt{a}$ from w_2 is located to the left of the single occurrence of $\mathtt{d}$ in w_1. In the following shuffle words we mark the symbols from w_2 (i. e. instead of $\mathtt{a}$ we write $\overline{\mathtt{a}}$ etc.) to illustrate the way shuffle words are created. The marking does not affect the scope coincidence degree. As pointed out before, the following shuffle word w' of w_1 and w_2 is minimal with respect to the scope coincidence degree, i. e. $\mathrm{scd}(w') = 2$:

$$w' = \mathtt{a} \cdot \overline{\mathtt{g}} \cdot \mathtt{b} \cdot \overline{\mathtt{h}} \cdot \mathtt{c} \cdot \overline{\mathtt{a}} \cdot \mathtt{d} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{e} \cdot \mathtt{f} \ .$$

We now consider the situation where a third word, namely $w_3 := \mathtt{e} \cdot \mathtt{f} \cdot \mathtt{g} \cdot \mathtt{h}$, has to be shuffled with w_1 and w_2, i. e. we examine $w_1 \sqcup w_2 \sqcup w_3$. Here, the problem of finding a shuffle word with minimum scope coincidence degree becomes more complex, which is caused by the fact that all the symbols in w_3 also occur in either w_1 or w_2. If we try to insert w_3 somehow into w' (which obviously results in a shuffle word of w_1, w_2 and w_3), we encounter the following problem. If $\mathtt{e} \cdot \mathtt{f}$ occurs to the left of $\mathtt{d}$, then $\mathtt{d}$ is in the scope of $\mathtt{e}$ and $\mathtt{f}$. On the other hand, if they occur to the right of $\mathtt{d}$, then so do the symbols $\mathtt{g}, \mathtt{h}$, which implies that $\mathtt{d}$ is in the scope of $\mathtt{g}$ and $\mathtt{h}$, and this increases the scope coincidence degree of the resulting word by 2. So in this new situation, inserting the symbol $\mathtt{a}$ from w_2 to the left of $\mathtt{d}$ (in order to keep $\mathtt{d}$ out of the scope of $\mathtt{a}$) is not a good idea anymore. Inserting $\mathtt{a}$ from w_2 to the right of $\mathtt{d}$ still puts $\mathtt{d}$ in the scope of $\mathtt{a}$, but, at the same time, allows us to keep $\mathtt{d}$ out of the scopes of $\mathtt{e}, \mathtt{f}, \mathtt{g}$ and $\mathtt{h}$, which results in a shuffle word with smaller scope coincidence degree. We consider an optimal shuffle word for w_1, w_2 and w_3 where the symbols from w_3 are marked (i. e. $\widehat{\mathtt{g}}$ instead of $\mathtt{g}$):

$$w'' := \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c} \cdot \mathtt{d} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{e} \cdot \widehat{\mathtt{e}} \cdot \mathtt{f} \cdot \widehat{\mathtt{f}} \cdot \overline{\mathtt{g}} \cdot \widehat{\mathtt{g}} \cdot \overline{\mathtt{h}} \cdot \widehat{\mathtt{h}} \cdot \overline{\mathtt{a}} \ .$$

It is easy to verify that $\mathrm{scd}(w'') = 3$. The following considerations demonstrate that this is indeed a shuffle word with minimum scope coincidence degree. We assume that there exists a $v \in w_1 \sqcup w_2 \sqcup w_3$ with $\mathrm{scd}(v) = 2$ (recall that 2 is a lower bound for the scope coincidence degree). This implies that the complete word w_2 occurs to the left of $\mathtt{d}$. Now consider the word w_3. In order to ensure $\mathrm{scd}(v) = 2$, the $\mathtt{e} \cdot \mathtt{f}$ factor of w_3 must occur to the right and the $\mathtt{g} \cdot \mathtt{h}$ factor of w_3 must occur to the left of $\mathtt{d}$. Clearly, by definition of a shuffle word, this is not possible.

The previous example also points out that it seems inappropriate to solve $\mathrm{CSminSCD}_\Sigma$ on input $w_1, w_2, \dots, w_k$ by first computing a minimal shuffle word w' of w_1, w_2 (ignoring $w_3, \dots, w_n$) and then solve $\mathrm{CSminSCD}_\Sigma$ on the smaller input $w', w_3 \dots, w_n$ and so on.

4 Outline of the Algorithm

Our algorithm is based on the observation that it is not necessary to search the whole shuffle $w_1 \sqcup w_2 \sqcup \dots \sqcup w_k$ in order to solve $\mathrm{CSminSCD}_\Sigma$, on arbitrary input words $w_1, w_2, \dots, w_k \in \Sigma^*$. It turns out that at least one shuffle word with minimum scope coincidence degree is included in the set of so-called *greedy shuffle words*, which is a proper subset of the whole shuffle $w_1 \sqcup w_2 \sqcup \dots \sqcup w_k$.

In general, we can construct any shuffle word by gradually cutting off prefixes of the words $w_1, w_2, \dots, w_k$, appending these prefixes to a new word w, and repeating this step until all w_i, $1 \leq i \leq k$, are empty. So at each step of this method, we maintain a word w, which is a prefix of the shuffle word that is constructed, and for each w_i, $1 \leq i \leq k$, a (possibly empty) suffix of w_i. A shuffle word is called greedy, if it can be constructed from $w_1, w_2, \dots, w_k$ by this method with the following condition. In each step, we always first cut off the longest prefixes of the w_i, $1 \leq i \leq k$, that only consist of symbols that already occur in w. In other words, if a w_i, $1 \leq i \leq k$, starts with a symbol not occurring in w, then we can only cut off a prefix of that w_i and append it to w if all the words w_i, $1 \leq i \leq k$, either start with such a symbol or are empty. This construction is then considered greedy with respect to those symbols already occurring in w.

In order to solve $\mathrm{CSminSCD}_\Sigma$, we only need to consider all greedy shuffle words to find one with minimum scope coincidence degree. The number of these greedy shuffle words is polynomial with respect to the number and length of the input words.

References

[1] M. R. Garey and D. S. Johnson. *Computers And Intractability.* W. H. Freeman and Company, 1979.

[2] D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25:322–336, 1978.

[3] F. Nicolasa and E. Rivals. Longest common subsequence problem for unoriented and cyclic strings. *Theoretical Computer Science*, 370:1–18, 2007.

[4] K.-J. Räihä and E. Ukkonen. The shortest common supersequence problem over binary alphabet is NP-complete. *Theoretical Computer Science*, 16:187–198, 1981.

[5] D. Reidenbach and M.L. Schmid. A polynomial time match test for large classes of extended regular expressions. In *Proc. 15th International Conference on Implementation and Application of Automata, CIAA 2010*, Lecture Notes in Computer Science, 2010. To appear.

[6] J. D. Ullman, A. V. Aho, and D. S. Hirschberg. Bounds on the complexity of the longest common subsequence problem. *Journal of the ACM*, 23:1–12, 1976.

Monadic Datalog Tree Transducers

Torsten Stüber

Faculty of Computer Science, Technische Universität Dresden

D-01062 Dresden, Germany

`stueber@tcs.inf.tu-dresden.de`

Abstract

Attributed tree transducers, an abstract form of attribute grammars, are a formal model of syntax-directed semantics, that is, a model for specifying tree transformations. Monadic datalog, a syntactically restricted fragment of standard datalog, is a means of formally specifying node selection queries on trees.

We introduce a tree transducer model combining aspects of both attributed tree transducers and monadic datalog, thereby allowing to specify in one rule information transport for non-adjacent nodes. We show that our model is strictly more powerful than attributed tree transducers, and we identify a large syntactic subclass which is as powerful as attributed tree transducers. This is shown by an effective construction.

On the Descriptional Complexity of Limited Propagating Lindenmayer Systems

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

Abstract

We investigate the descriptional complexity of limited propagating Lindenmayer systems and their deterministic and tabled variants with respect to the number of rules and the number of symbols. We determine the decrease of complexity when the generative capacity is increased. For incomparable families, we give languages that can be described more efficiently in either of these families than in the other.

1 Introduction

Several generating devices for formal languages have been studied in the literature with respect to the size of their descriptions (e. g., [2]). For sequentially deriving grammars, the measures number of productions, number of nonterminal symbols, and number of all symbols have been investigated.

In 1968, Lindenmayer systems (L-systems) have been introduced ([4]). In order to model the development of organisms, these devices work in parallel (in one derivation step, not only one symbol is rewritten as in a sequential grammar but all symbols are rewritten). For L-systems, the number of tables, the number of active symbols, and the degree of nondeterminism have been studied as measures of complexity. In [1], the measures number of rules and number of symbols were introduced for L-systems.

Twenty years after the introduction of L-systems, a restricted variant of L-systems with a partially parallel derivation process has been proposed in [7]. In these so-called k-limited L-systems, only k occurrences of each symbol are replaced according to some rule. First results on the descriptional complexity of k-limited L-systems can be found in [3].

We continue this work and study the relations that were left open in [3] or that have not been optimal yet. In this paper, we confine ourselves to propagating limited systems.

2 Definitions

We assume that the reader is familiar with the basic concepts of formal language theory (see e. g. [5]). We recall here some notations used in the paper.

We denote the set of all positive integers by $\mathbb{N}$ and the set of all non-negative integers by $\mathbb{N}_0$.

For an alphabet V (a finite set of symbols), we denote by V^* the set of all words over V, by V^+ the set of all non-empty words over V, and by V^n for a natural number $n \in \mathbb{N}_0$ the set of all words which have the length n. We denote the empty word by λ, the length of a word w by $|w|$, and the number of occurrences of a letter a in a word w by $|w|_a$. Furthermore, we denote the cardinality of a set A by $|A|$.

Two sets X and Y are called incomparable, if neither $X \subseteq Y$ nor $Y \subseteq X$ holds. They are called disjoint if the intersection is empty.

A tabled interactionless Lindenmayer system (L-system for short), abbreviated as T0L system, is a triple $G = (V, \mathcal{P}, \omega)$ where V is an alphabet, $\omega \in V^+$ is called the axiom, and $\mathcal{P}$ is a finite, non-empty set $\{ P_1, P_2, \ldots, P_n \}$ where P_i (called a table), for $1 \leq i \leq n$, is a finite subset of $V \times V^*$ such that there is at least one element $(a, w) \in P_i$ for each letter $a \in V$. The elements (a, w) in some table are called productions or rules and are written as $a \to w$.

A T0L system $G = (V, \mathcal{P}, \omega)$ is called an 0L system if $\mathcal{P}$ contains only one table. It is called a DT0L system if every table $\mathcal{P}$ contains only one rule for each letter in V and it is called a D0L system if $\mathcal{P}$ contains only one table and the table consists of only one rule for each letter in V.

Such an L-system is called propagating, if there is no erasing rule $a \to \lambda$ in the system (all rules have the form $a \to w$ with $a \in V$ and $w \in V^+$).

A word $v \in V^+$ directly derives a word $w \in V^*$ by a system G, written as $v \Longrightarrow_G w$ (we omit the index if it is clear from the context), if $v = x_1 x_2 \cdots x_m$ with $m \in \mathbb{N}$, $x_i \in V$ for $1 \leq i \leq m$ and $w = y_1 y_2 \cdots y_m$ with $y_i \in V^*$ for $1 \leq i \leq m$ such that the system G contains a table P which contains all the rules $x_i \to y_i$ for $1 \leq i \leq m$. Hence, in parallel, every letter of a word is replaced by a word according to the rules of a table. By $\Longrightarrow^*$, we denote the reflexive and transitive closure of $\Longrightarrow$. The language generated by a system G is defined as

$$L(G) = \{ z \mid \omega \Longrightarrow_G^* z \} .$$

In [7], a limitation of the parallel rewriting was introduced. For a natural number $k \in \mathbb{N}$, a k-limited T0L system (shortly written as $k\ell$T0L system) is a quadruple $G = (V, \mathcal{P}, \omega, k)$ where $(V, \mathcal{P}, \omega)$ is a T0L system. In a k-limited system, exactly $\min\{k, |w|_a\}$ occurrences of any letter a in the word w under consideration are rewritten in a derivation step (hence, the number of occurrences of a letter that are replaced in each step is limited by k).

We only say a T0L system is limited (shortly written as ℓT0L system) if it is a k-limited system for some number $k \in \mathbb{N}$.

The class of all k-limited T0L systems is written as $k\ell$T0L. The restricted and propagating variants thereof are denoted by $k\ell$PD0L, $k\ell$P0L, $k\ell$PDT0L, $k\ell$PT0L, and without k if the limit is arbitrary. For a class X of L-systems, we write $\mathcal{L}(X)$ for the family of languages that is generated by an L-system from X.

As measures of descriptional complexity, we consider the number of rules and the number of symbols. For an L-system G over an alphabet V with tables $P_1, P_2, \ldots, P_n$ with $n \in \mathbb{N}$ and an axiom ω, we set

$$\mathrm{Prod(G)} = \sum_{\mathrm{i=1}}^{\mathrm{n}} |\mathrm{P_i}| \quad \text{and} \quad \mathrm{Symb(G)} = |\omega| + \sum_{\mathrm{i=1}}^{\mathrm{n}} \sum_{\mathrm{a \to w \in P_i}} (|\mathrm{w}| + 2).$$

Let X be a class of L-systems. For a language $L \in \mathcal{L}(X)$, we set

$$\mathrm{Prod_X(L)} = \min\{\, \mathrm{Prod(G)} \mid \mathrm{G \in X \text{ with } L(G) = L} \,\} \text{ and}$$
$$\mathrm{Symb_X(L)} = \min\{\, \mathrm{Symb(G)} \mid \mathrm{G \in X \text{ with } L(G) = L} \,\}.$$

Hence, the complexity of a language L with respect to a class X of L-systems is the complexity of a smallest L-system $G \in X$ that generates the language L. If we extend a class X to a class Y then the complexity can only become smaller: If $X \subseteq Y$, then $K_X(L) \geq K_Y(L)$ for any language $L \in \mathcal{L}(X)$ and complexity measure $K \in \{\, \mathrm{Prod}, \mathrm{Symb} \,\}$.

We now define the complexity relations considered in this paper. Let X and Y be two classes of L-systems such that the language families $\mathcal{L}(X)$ and $\mathcal{L}(Y)$ are not disjoint and let $K \in \{\, \mathrm{Prod}, \mathrm{Symb} \,\}$ be a complexity measure. We write

(a) $X =^K Y$ if $K_X(L) = K_Y(L)$ holds for any language $L \in \mathcal{L}(X) \cap \mathcal{L}(Y)$ (the complexities are equal),

(b) $X >^K Y$ if there is a sequence of languages $L_m \in \mathcal{L}(X) \cap \mathcal{L}(Y)$, $m \in \mathbb{N}$, such that $K_X(L_m) - K_Y(L_m) \geq c \cdot m$ for a constant $c \in \mathbb{N}$ (the difference of the complexities can be arbitrarily large),

(c) $X \gg^K Y$ if there is a sequence of languages $L_m \in \mathcal{L}(X) \cap \mathcal{L}(Y)$, $m \in \mathbb{N}$, such that $\lim\limits_{m \to \infty} \frac{K_Y(L_m)}{K_X(L_m)} = 0$ (asymptotically, the complexity using X grows faster than using Y),

(d) $X \ggg^K Y$ if there is a sequence of languages $L_m \in \mathcal{L}(X) \cap \mathcal{L}(Y)$, $m \in \mathbb{N}$, and a constant $c \in \mathbb{N}$ such that $K_Y(L_m) \leq c$ and $K_X(L_m) \geq m$.

From these definitions, we obtain that $X \ggg^K Y$ implies $X \gg^K Y$ and that also $X \gg^K Y$ implies $X >^K Y$ for $K \in \{\,\mathrm{Prod}, \mathrm{Symb}\,\}$.

For each natural number c, there are only finitely many L-systems G (upto renaming the symbols) for which Symb(G) $\leq$ c holds. Hence, there is no class X of L-systems that generates infinitely many languages L_n with $\mathrm{Symb_X(L_n)} \leq \mathrm{c}$. Thus, there are no two classes X and Y with the relation $Y \ggg^{\mathrm{Symb}} X$.

In all cases throughout this paper, we obtain the relation $X \gg^{\mathrm{Symb}} Y$ whenever we obtain $X \ggg^{\mathrm{Prod}} Y$. Then, we also shortly write $X \ggg Y$. Further, if two classes X and Y are in the same relation $\rhd$ with respect to both measures Prod and Symb, hence, $X \rhd^{\mathrm{Prod}} Y$ and $X \rhd^{\mathrm{Symb}} Y$ for a symbol $\rhd \in \{\gg, >, =\}$, then we write $X \rhd Y$.

3 Results

We first give some sequences of languages that are used as witnesses for the relations given later.

For $m \geq 1$, let

$$\begin{aligned}
L_m^{(1)} &= \{e\} \cup \{\, a^n x_1 x_2 \cdots x_m d^n \mid n \geq 1,\ x_i \in \{b, c\},\ 1 \leq i \leq m \,\}, \\
L_m^{(2)} &= \{\, a_1^{n_1} a_2^{n_2} \cdots a_m^{n_m} \mid n_i \geq 1,\ 1 \leq i \leq m \,\}, \\
L_m^{(3)} &= \{e\} \cup \left\{\, a^{kn} w d^{kn} \mid n \geq 1, w \in \{b, c\}^{km}, |w|_b = kj \text{ for } 0 \leq j \leq m \,\right\}, \\
L_m^{(4)} &= \{c\} \cup \{\, x_1 x_2 \cdots x_{km} \mid x_i \in \{a, bb\},\ 1 \leq i \leq km \,\}, \\
L_m^{(5)} &= \left\{\, a^{m+im^2} b \mid i \geq 0 \,\right\} \cup \left\{\, a^{m+im^2+1} c \mid i \geq 0 \,\right\} \cup \{\, d \,\}, \\
L_m^{(6)} &= \{\, d \,\} \cup \{\, w \mid w = x_1 x_2 \cdots x_{2m},\ x_i \in \{a, bb, ccc\}, 1 \leq i \leq 2m, \\
&\qquad\qquad\qquad\qquad\qquad\qquad |w|_a = 2n,\ 0 \leq n \leq m \,\}, \\
L_m^{(7)} &= \{a^{5mk(1+nm)} \mid n \geq 0\}, \\
L_m^{(8)} &= \{\, c \,\} \cup \{\, w \mid w = x_1 x_2 \cdots x_{(k+1)m},\ x_i \in \{a, bb\}, 1 \leq i \leq (k+1)m, \\
&\qquad\qquad\qquad\qquad\qquad\qquad |w|_a = j(k+1), 0 \leq j \leq m \,\}.
\end{aligned}$$

Regarding 1-limited propagating L-systems, we obtain the hierarchy shown in the following figure.

In brackets behind a relation, you find a witness sequence of languages or a link to the corresponding proof in the literature.

$$\begin{tikzcd}
 & 1\ell\mathrm{PT0L} & \\
1\ell\mathrm{P0L} \arrow[ur, "\ggg (L_m^{(1)})"] \arrow[rr, bend left=10, "\ggg (L_m^{(1)})"] & & 1\ell\mathrm{PDT0L} \arrow[ul, "\gg {[3]}"'] \arrow[ll, bend left=10, "\ggg (L_m^{(2)})"] \\
 & 1\ell\mathrm{PD0L} \arrow[ul, "\substack{=^{\mathrm{Prod}} {[3]} \\ =^{\mathrm{Symb}} {[6]}}"] \arrow[ur, "\substack{=^{\mathrm{Prod}} {[3]} \\ =^{\mathrm{Symb}} {[6]}}"'] &
\end{tikzcd}$$

Figure 1: Results for 1-limited systems

If a sequence of languages L_m is generated by 1ℓP0L systems or 1ℓPT0L systems with a constant number of rules, then the languages L_m can also be generated by 1ℓPDT0L systems with a constant number of rules. As a consequence, all relations mentioned above are tight.

Regarding k-limited propagating L-systems with $k \geq 2$, we obtain the hierarchy shown in the following figure.

$$\begin{tikzcd}
 & k\ell\mathrm{PT0L} & \\
k\ell\mathrm{P0L} \arrow[ur, "\ggg {[3]}"] \arrow[rr, bend left=10, "\ggg (L_m^{(3)})"] & & k\ell\mathrm{PDT0L} \arrow[ul, "\ggg {[3]}"'] \arrow[ll, bend left=10, "\ggg (L_m^{(4)})"] \\
 & k\ell\mathrm{PD0L} \arrow[ul, "\substack{=^{\mathrm{Prod}} {[3]} \\ =^{\mathrm{Symb}} {[6]}}"] \arrow[ur, "\substack{=^{\mathrm{Prod}} {[3]} \\ =^{\mathrm{Symb}} {[6]}}"'] &
\end{tikzcd}$$

Figure 2: Results for higher limited systems

If one considers classes of limited propagating L-systems, one cannot only add nondeterminism when increasing the generative power but the limit of a minimal system can change. This has especially an impact on the relations between the classes ℓPD0L and ℓP0L or ℓPDT0L with respect to the number of symbols. For example, a minimal limited PD0L-system for generating the language $L_m^{(5)}$ for a number $m \geq 1$ has the limit $k = 1$ and

$$\mathrm{Symb}_{\ell\mathrm{PD0L}}(\mathrm{L}_\mathrm{m}^{(5)}) = \mathrm{m}^2 + \mathrm{m} + 12$$

symbols. However, using an ℓP0L-system or an ℓPDT0L-system, the number of symbols can be reduced to $3m + 17$ or $2m + 26$, respectively, while the limit is increased to m.

The hierarchy obtained regarding limited propagating L-systems is shown in the following figure.

$$\begin{tikzcd}
 & \ell\mathrm{PT0L} & \\
\ell\mathrm{P0L} \arrow[ur, "\gg\ {[3]}"] \arrow[rr, bend left, "\gg\ (L_m^{(1)})"] & & \ell\mathrm{PDT0L} \arrow[ul, "\gg\ {[3]}"'] \arrow[ll, bend left, "\gg\ (L_m^{(6)})"] \\
 & \ell\mathrm{PD0L} \arrow[ul, "=^{\mathrm{Prod}}\ {[3]}\ \gg^{\mathrm{Symb}}\ (L_m^{(5)})"] \arrow[ur, "=^{\mathrm{Prod}}\ {[3]}\ \gg^{\mathrm{Symb}}\ (L_m^{(5)})"'] &
\end{tikzcd}$$

Figure 3: Results for arbitrarily limited systems

The relations between k-limited and arbitrarily limited propagating L-systems can be seen in the following figure.

$$\begin{tikzcd}
 & & & & & \ell\mathrm{PT0L} & \\
 & k\ell\mathrm{PT0L} \arrow[rrrrd, dashed, "\gg\ (L_m^{(8)})"] & & & \ell\mathrm{P0L} \arrow[ur] & & \ell\mathrm{PDT0L} \arrow[ul] \\
k\ell\mathrm{P0L} \arrow[ur] & & k\ell\mathrm{PDT0L} \arrow[ul] & & & \ell\mathrm{PD0L} \arrow[ul] \arrow[ur] & \\
 & k\ell\mathrm{PD0L} \arrow[ul] \arrow[ur] \arrow[rrrru, dashed, "=^{\mathrm{Prod}}\ {[3]}\ \gg^{\mathrm{Symb}}\ (L_m^{(7)})"'] & & & & &
\end{tikzcd}$$

Figure 4: Relations between k-limited and arbitrarily limited systems

Proofs of all relations presented here can be found in either [3] or [6].

References

[1] J. Dassow, On the descriptional complexity of Lindenmayer systems. *International Journal of Foundations of Computer Science* **15** (2004) 4, 663–672.

[2] J. Gruska, Descriptional Complexity (of Languages) – A Short Survey. In: *MFCS*. Lecture Notes in Computer Science 45, Springer, 1976, 65–80.

[3] R. Harbich, *Beschreibungskomplexität k-limitierter und limitierter Lindenmayer-Systeme*. Diplomarbeit, Otto-von-Guericke-Universität Magdeburg, 2009.

[4] A. LINDENMAYER, Mathematical models for cellular interaction in development: Parts I and II. *Journal of Theoretical Biology* **18** (1968) 3, 280–315.

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

[6] B. TRUTHE, On the Descriptional Complexity of Limited Propagating L-Systems. In: I. MCQUILLAN, G. PIGHIZZINI, B. TROST (eds.), *DCFS 2010*. Technical Report 2010-02, Department of Computer Science, University of Saskatchewan, Canada, 2010, 215–226.

[7] D. WÄTJEN, k-limited 0L systems and languages. *Journal of Information Processing and Cybernetics EIK* **24** (1988) 6, 267–285.

Centralized Versus Non-Centralized Parallel Communicating Systems of Restarting Automata

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

Abstract

Recently we introduced parallel communicating (PC) systems of restarting automata [5]. With regard to the communication structure a comparison between centralized and non-centralized versions seems obvious. In this contribution it will be shown that in case of PC systems of restarting automata centralized and non-centralized systems have the same computational power.

1 Introduction

The concept of parallel communicating (PC for short) components was already applied to different kinds of formal models: PC grammar systems [8], PC systems of finite automata [6], PC systems of pushdown automata [3], PC systems of Watson-Crick automata [4] etc. Further, in [5] we introduced PC systems of restarting automata. Regarding distributed computation the question of an appropriate communication structure seems obvious. One crucial aspect is the issue of whether the communication is centralized or non-centralized. All components of a centralized system are only allowed to communicate with a distinguished master component, whereas in a non-centralized communication structure each component can communicate with all other components.
That non-centralized systems are at least as powerful as centralized systems seems clear (see e.g., [3, 6]). But does the centralization as a restriction of the communication structure yield a proper decrease of computational power? For example, this holds true for deterministic parallel communicating finite automata [1] and PC grammar systems with regular components [2]. Within this contribution it will be shown that in case of PC systems of restarting automata centralized and non-centralized systems have the same

computational power independent of the component's type.
Basic notations and definitions of restarting automata can be found in [7].

2 PC Systems of Restarting Automata

A system $\mathcal{M}$ of parallel communicating restarting automata (*PCRA* for short) of degree n is an n-tuple

$$\mathcal{M} = (M_1, M_2, \ldots, M_n),$$

where $M_1, M_2, \ldots, M_n$ are restarting automata which are the components of the system:

$$\begin{aligned} M_1 &= (Q_1, \Sigma, \Gamma_1, ¢, \$, q_1, s_1, \delta_1), \\ M_2 &= (Q_2, \Sigma, \Gamma_2, ¢, \$, q_2, s_2, \delta_2), \\ &\vdots \\ M_n &= (Q_n, \Sigma, \Gamma_n, ¢, \$, q_n, s_n, \delta_n). \end{aligned}$$

Here $Q_1, \ldots, Q_n$ are the sets of states, Σ is an input alphabet, $\Gamma_1, \ldots, \Gamma_n$ are tape alphabets, ¢ and \$ are the left and the right sentinels, $q_1, \ldots, q_n$ are the initial states, $s_1, \ldots, s_n$ are the sizes of the read/write windows, and $\delta_1, \ldots, \delta_n$ are the transition functions.

The communication between the components is realized by special kinds of communication states contained in $Q_1, Q_2, \ldots, Q_n$: request states (req^i_d), response states ($res^i_{d,c}$), receive states ($rec^i_{d,c}$), and acknowledge states ($ack^i_{d,c}$). The superscript i denotes the addressee (generally the index of the communication partner), the subscript d stands for an optional local information, and the subscript c is the information that is exchanged during the communication. Observe that d and c must be of constant length. The communication process is schematically illustrated in Figure 1 and formalized in the definition of a computation step (see below).

A *configuration* K of a *PCRA* of degree n is an n-tuple, which contains a configuration of each component:

$$K = (k_1, k_2, \ldots, k_n).$$

Here k_i ($1 \le i \le n$) is of the form $u_i q_i v_i$ where $q_i \in Q_i$, and $u_i = ¢\alpha$ and $v_i = \beta\$$, or $u_i = \varepsilon$ and $v_i = ¢\beta\$$ ($\alpha, \beta \in \Gamma_i^*$, ε denotes the empty word). For an input word w and initial states q_i ($1 \le i \le n$) the initial configuration of the system is:

$$K_0 = (q_1¢w\$, q_2¢w\$, \ldots, q_n¢w\$).$$

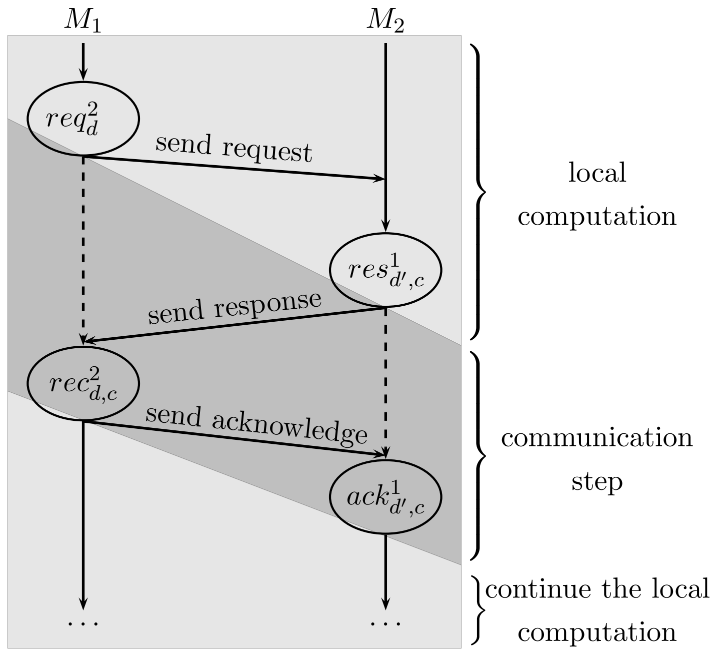


Figure 1: A communication between two components.

A *computation step* of the system $\mathcal{M} = (M_1, M_2, \ldots, M_n)$ can be described by the binary relation $\vdash_{\mathcal{M}}$. Let K and K' be two configurations with $K = (k_1, k_2, \ldots, k_n)$ and $K' = (k'_1, k'_2, \ldots, k'_n)$. Then $K \vdash_{\mathcal{M}} K'$ iff for all $i \in \{1, 2, \ldots, n\}$ one of the following conditions holds:

1. $k_i \vdash_{M_i} k'_i$ (local computation step)
2. $\exists j \in \{1, 2, \ldots, n\} \setminus \{i\} : k_i = u_i req^j_{d_i} v_i, k_j = u_j res^i_{d_j,c} v_j, k'_i = u_i rec^j_{d_i,c} v_i,$
   $k'_j = u_j ack^i_{d_j,c} v_j$ (communication)
3. $\exists j \in \{1, 2, \ldots, n\} \setminus \{i\} : k_i = u_i res^j_{d_i,c} v_i, k_j = u_j req^i_{d_j} v_j, k'_i = u_i ack^j_{d_i,c} v_i,$
   $k'_j = u_j rec^i_{d_j,c} v_j$ (communication)
4. $k_i = k'_i$ and no local operation (MVR, MVL, replacement, restart) or communication of M_i is possible.

The reflexive and transitive closure of the relation $\vdash_{\mathcal{M}}$ is expressed by $\vdash^*_{\mathcal{M}}$ and describes a *computation* of $\mathcal{M}$. Then the *accepted language* of a PC system $\mathcal{M}$ over an input alphabet Σ is

$$L(\mathcal{M}) = \{w \in \Sigma^* \mid (q_1 ¢w\$, q_2 ¢w\$, \ldots, q_n ¢w\$) \vdash^*_{\mathcal{M}} (k_1, k_2, \ldots, k_n),$$
$$\{k_1, k_2, \ldots, k_n\} \cap \{Accept\} \neq \emptyset\}$$

where $q_1, q_2, \ldots, q_n$ are the initial states of the components.

A *PCRA* is called *nondeterministic* if there is at least one component that is nondeterministic. If all components are deterministic, the system is called *locally deterministic* and gets the prefix *det-local*. In general there is no restriction according to the accepting component. That allows the system to accept the input with whatever component reaches the accepting configuration. A more strict definition of determinism in *PCRA* is that the system accepts if and only if the first component accepts. Such systems are called *globally deterministic* and get the prefix *det-global*.

The types of the components determine the type of the system. If the components are restarting automata of type *RRWW*, the system is of the type *PC-RRWW*.

3 Centralized Versus Non-Centralized PC Systems of Restarting Automata

A *centralized PC-RRWW*-system (*cPC-RRWW*-system for short) is a *PC-RRWW*-system in which every component is only allowed to communicate with the first component (the master component). The set of languages accepted by *cPC-RRWW*-systems is denoted by $\mathcal{L}(cPC\text{-}RRWW)$.

Theorem 1. $\mathcal{L}(cPC\text{-}RRWW) = \mathcal{L}(PC\text{-}RRWW)$.

Proof outline. $\mathcal{L}(cPC\text{-}RRWW) \subseteq \mathcal{L}(PC\text{-}RRWW)$ is clear.
Consider $\mathcal{L}(PC\text{-}RRWW) \subseteq \mathcal{L}(cPC\text{-}RRWW)$. The two main aspects for this proof are the following. First a new component M (the master component) is inserted that controls all communications, and then the original components are modified in a way that they send all information about communication wishes, of being stuck, and acceptance to the master. Observe that these are the only three situations that can appear at the end of a component's local computation.
To reach all components the master communicates cyclically with the components. Furthermore it is important that the system does not get stuck if a single component is stuck (if an appropriate transition for the current configuration of a component is missing). This is realized by sending an according information to the master at the end of all local computations. Hence the master knows at all times in which situation all components are and can decide whether to communicate with a special component or not. If a component sends the accepting information to the master, the latter and thus the system accepts

like the original system. If there is no component within the original system that accepts, then there is no component within the modified system sending the master the accepting information. Hence the modified system does not accept, either. □

Theorem 2. $\mathcal{L}(cPC\text{-}RLWW) = \mathcal{L}(PC\text{-}RLWW)$

Proof outline. MVL-operations together with MVR-operations can cause infinite loops within a component's computation. In the proof outline above this may lead to a deadlock of the whole system, if the master communicates with a component being in such a loop and hence cannot answer. To avoid this kind of situation a component communicates with the master after every MVL-operation. Like before the master can now cyclically communicate with all components, thus all components can continue their local computations, and if there is any component reaching the accepting configuration, the whole system accepts. If there is no component that reaches the accepting configuration and some component is catched in a loop, then the system is also in a loop and does not accept the input just like the original non-centralized system. □

Theorem 3. *For every PC-RLWW-system $\mathcal{M}$ there exists a cPC-RLWW-system $\mathcal{M}'$ with $L(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$ such that $\mathcal{M}'$ has the same number of components as $\mathcal{M}$.*

Proof outline. The centralized system considered in Theorem 1 and extended in Theorem 2 contains one additional component (the master component) in comparison to the non-centralized system. But it can be constructed in a way that the master component applies only communications but no local operations (no MVL, MVR, replacing, restart). In other words the computation of the master component never depends on the content of the tape. Thus, the idea seems obvious to merge the master component with some other component and to build the product automaton in a broader sense. Then both work mainly simultaneously. In the case of a restart of the product automaton a third component is needed for storing the information and give it back after the restart of the product automaton (observe that a system of degree one or two is already centralized per definition). □

The system considered in Theorems 1, 2, and 3 is not only centralized but can be constructed with the property that the system accepts iff the first component accepts. Thus an interesting conclusion of Theorem 1, 2, and 3 is that for every locally deterministic $PCRA$ there exists an equivalent (centralized) globally deterministic system (the opposite direction is clear):

Corollary 1. $\mathcal{L}(det\text{-}global\text{-}PC\text{-}X) = \mathcal{L}(det\text{-}local\text{-}PC\text{-}X)$ *for all* $X \in \{R, RR, RW, RRW, RWW, RRWW, RL, RLW, RLWW\}$.

References

[1] Henning Bordihn, Martin Kutrib, and Andreas Malcher. On the computational capacity of parallel communicating finite automata. In Masami Ito and Masafumi Toyama, editors, *Developments in Language Theory*, volume 5257, pages 146–157. Springer, 2008.

[2] Erzsébet Csuhaj-Varjú, Josef Kelemen, Gheorghe Păun, and Jürgen Dassow, editors. *Grammar Systems: A Grammatical Approach to Distribution and Cooperation.* Gordon and Breach Science Publishers, Inc., Newark, NJ, USA, 1994.

[3] Erzsébet Csuhaj-Varjú, Carlos Martín-Vide, Victor Mitrana, and György Vaszil. Parallel Communicating Pushdown Automata Systems. *Int. J. Found. Comput. Sci.*, 11(4):633–650, 2000.

[4] Elena Czeizler and Eugen Czeizler. Parallel communicating Watson-Crick automata systems. *Acta Cybern.*, 17(4):685–700, 2006.

[5] Marcel Goehring. PC-Systems of Restarting Automata. In Jöran Mielke, Ludwig Staiger, and Renate Winter, editors, *Theorietag Automaten und Formale Sprachen 2009*, pages 26–27, Universität Halle-Wittenberg, Institut für Informatik, 2009. Technical Report 2009/03.

[6] Carlos Martín-Vide, Alexandru Mateescu, and Victor Mitrana. Parallel Finite Automata Systems Communicating by States. *Int. J. Found. Comput. Sci.*, 13(5):733–749, 2002.

[7] Friedrich Otto. Restarting Automata. In Zoltán Ésik, Carlos Martín-Vide, and Victor Mitrana, editors, *Recent Advances in Formal Languages and Applications*, volume 25 of *Studies in Computational Intelligence*, pages 269–303. Springer, 2006.

[8] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages, vol. 2: linear modeling: background and application.* Springer-Verlag New York, Inc., New York, NY, USA, 1997.

Teilnehmerliste

1	Bensch, Suna Umeå University, Department of Computer Science SE-90187 Umeå, Sweden `suna@cs.umu.se`
2	Bordihn, Henning Universität Potsdam, Institut für Informatik August-Bebel-Straße 89, D-14482 Potsdam `henning@cs.uni-potsdam.de`
3	Damm, Carsten Universität Göttingen, Institut für Informatik Fakultät für Mathematik und Informatik Goldschmidtstraße 7, D-37077 Göttingen `damm@informatik.uni-goettingen.de`
4	Dassow, Jürgen Universität Magdeburg, Fachbereich Informatik Universitätsplatz 2, D-39106 Magdeburg `dassow@iws.cs.uni-magdeburg.de`
5	Doll, Jens Context IT GmbH Postfach 1616, D-22906 Ahrensburg `jd@cococo.de`
6	Freund, Rudolf TU Wien, Institut für Computersprachen Favoritenstraße 9, A-1040 Wien `rudi@emcc.at`
7	Freydenberger, Dominik D. Universität Frankfurt, Institut für Informatik Robert-Mayer-Straße 11–15, D-60054 Frankfurt am Main `freydenberger@em.uni-frankfurt.de`
8	Gelderie, Marcus RWTH-Aachen, Lehrstuhl Informatik 7 Ahornstraße 55, D-52074 Aachen `marcus.gelderie@rwth-aachen.de`
9	Gulan, Stefan Universität Trier, Fachbereich Informatik Campus II, D-54286 Trier `Gulan@informatik.uni-trier.de`

10	Harbich, Ronny Universität Magdeburg, Fachbereich Informatik Universitätsplatz 2, D-39106 Magdeburg `ronny.harbich@student.uni-magdeburg.de`
11	Holzer, Markus Universität Giessen, Institut für Informatik Arndtstraße 2, D-35392 Giessen `holzer@informatik.uni-giessen.de`
12	Hundeshagen, Norbert Universität Kassel, Fachbereich Elektrotechnik/Informatik Wilhelmshöher Allee 71–73, D-34121 Kassel `hundeshagen@theory.informatik.uni-kassel.de`
13	Jacobi, Sebastian Universität Giessen, Institut für Informatik Arndtstraße 2, D-35392 Giessen `Sebastian.Jacobi@informatik.uni-giessen.de`
14	Kasprzik, Anna Universität Trier, Fachbereich Informatik Campus II, D-54286 Trier `kasprzik@informatik.uni-trier.de`
15	Kogler, Marian Universität Halle-Wittenberg, Institut für Informatik Von Seckendorff Platz 1, D-06120 Halle (Saale) `kogler@informatik.uni-halle.de`
16	Kutrib, Martin Universität Giessen, Institut für Informatik Arndtstraße 2, D-35392 Giessen `kutrib@informatik.uni-giessen.de`
17	Lange, Martin Universität Kassel, Fachbereich Elektrotechnik/Informatik Wilhelmshöher Allee 71–73, D-34121 Kassel `martin.lange@uni-kassel.de`
18	Leupold, Peter Universität Kassel, Fachbereich Elektrotechnik/Informatik Wilhelmshöher Allee 71–73, D-34121 Kassel `Peter.Leupold@web.de`

19	Malcher, Andreas Universität Giessen, Institut für Informatik Arndtstraße 2, D-35392 Giessen `Andreas.Malcher@informatik.uni-giessen.de`
20	Manea, Florin Universität Magdeburg, Fachbereich Informatik Universitätsplatz 2, D-39106 Magdeburg `flmanea@gmail.com`
21	Meckel, Katja Universität Giessen, Institut für Informatik Arndtstraße 2, D-35392 Giessen `Katja.Meckel@math.uni-giessen.de`
22	Meinecke, Ingmar Universität Leipzig, Institut für Informatik Fakultät für Mathematik und Informatik Johannisgasse 26, D-04103 Leipzig `meinecke@informatik.uni-leipzig.de`
23	Otto, Friedrich Universität Kassel, Fachbereich Elektrotechnik/Informatik Wilhelmshöher Allee 71–73, D-34121 Kassel `otto@theory.informatik.uni-kassel.de`
24	Plátek, Martin Charles University, Faculty of Math. and Physics Department of Computer Science Malostranské nám. 25, CZ-11800 Prague 1 `Martin.Platek@mff.cuni.cz`
25	Schmid, Markus Loughborough University, Department of Computer Science Leics, UK LE11 3TU `M.Schmid@lboro.ac.uk`
26	Stüber, Thorsten TU Dresden, Fakultät Informatik Nöthnitzer Straße 46, D-01187 Dresden `stueber@tcs.inf.tu-dresden.de`
27	Truthe, Bianca Universität Magdeburg, Fachbereich Informatik Universitätsplatz 2, D-39106 Magdeburg `truthe@iws.cs.uni-magdeburg.de`

28	Vollmer, Heribert Universität Hannover, Institut für Theoretische Informatik Fakultät für Elektrotechnik und Informatik Appelstraße 4, D-30167 Hannover `vollmer@thi.uni-hannover.de`
29	Vollweiler, Marcel Universität Kassel, Fachbereich Elektrotechnik/Informatik Wilhelmshöher Allee 71–73, D-34121 Kassel `vollweiler@theory.informatik.uni-kassel.de`
30	Wagner, Klaus Universität Würzburg, Institut für Informatik Am Hubland, D-97074 Würzburg `wagner@informatik.uni-wuerzburg.de`
31	Wendlandt, Matthias Universität Giessen, Institut für Informatik Arndtstraße 2, D-35392 Giessen `matthias.wendlandt@informatik.uni-giessen.de`
32	Wotschke, Detlef Universität Giessen, Institut für Informatik Arndtstraße 2, D-35392 Giessen `wotschke@em.uni-frankfurt.de`