23. Theorietag

Automaten und formale Sprachen

25. – 27. September 2013

Fachgebiet Theoretische Informatik, TU Ilmenau

TECHNISCHE UNIVERSITÄT
ILMENAU

Workshop-Programm
Mittwoch, 25. September 2013

09.30	Benedikt Bollig (Cachan) *Automata and logic for concurrent systems*	S. 7
10.30	Kaffeepause	
11.00	Manfred Droste (Leipzig) *Weighted logics for unranked tree automata*	S. 9
12.00	Mittagspause	
13.30	Christof Löding (Aachen) *Constructions and algorithms for ω-automata*	S. 10
14.30	Pause	
14.45	Manfred Kufleitner (Stuttgart) *Logikbeschreibungen regulärer Sprachen*	S. 12
15.45	Kaffeepause	
16.15	Markus Lohrey (Siegen) *Isomorphie von endlich präsentierten Strukturen*	S. 13

Vormittagsprogramm
Donnerstag, 26. September 2013

09.00	Bianca Truthe *Ketten evolutionärer Prozessoren mit Random-Context-Filtern*	S. 14
	Norbert Hundeshagen *On restarting transducers*	S. 17
	Phoebe Buckheister, Georg Zetzsche *Recent advances on valence automata as a generalization of automata with storage*	S. 20
	Martin Kutrib, Andreas Malcher, Carlo Mereghetti, Beatrice Palano, Matthias Wendlandt *Input-driven queue automata*	S. 22
10.40	Kaffeepause	
11.10	Sebastian Jakobi, Katja Meckel, Carlo Mereghetti, Beatrice Palano *Queue automata of constant length*	S. 23
	Markus Holzer, Sebastian Jakobi *Nondeterministic biautomata and their descriptional complexity*	S. 26
	Dominik D. Freydenberger *Deskriptive Pattern und Ketten von Patternsprachen*	S. 29
12.25	Mittagspause	

Nachmittagsprogramm
Donnerstag, 26. September 2013

14.00	Markus Teichmann, Johannes Osterholzer *Characterizing tree valuation languages by multioperator weighted tree languages*	S. 32
	Johannes Osterholzer *A pushdown machine for context-free tree translation*	S. 35
	Matthias Büchse, Tobias Denkinger, Heiko Vogler *Construction of a bottom-up deterministic n-gram weighted tree automaton*	S. 38
	Toni Dietze *State-splitting for regular tree grammars*	S. 41
15.40	Kaffeepause	
16.10	Jens-D. Doll *Anforderungen an ein formales Esperanto (FE)*	S. 44
	Yvonne Meeres *Formalism for the usage of natural language*	S. 45
	Henning Fernau, Rudolf Freund, Sergiu Ivanov, Marion Oswald, Markus L. Schmid, K. G. Subramanian *Array insertion and deletion P systems*	S. 46
	Henning Fernau, Rudolf Freund, Markus L. Schmidt, K. G. Subramanian, Petra Wiederhold *Eine natürliche Membranhierachie*	S. 49

Programm
Freitag, 27. September 2013

09.00	Markus Lohrey, Georg Zetzsche *On Boolean closed full trios and rational Kripke frames*	S. 52
	Thomas Weidner *Characterizing probabilistic ω-recognizability by MSO logic and regular expressions*	S. 55
	Manfred Droste, Doreen Götze *The support of weighted unranked tree automata*	S. 58
	Achim Blumensath *Recognisability for infinite trees*	S. 59
10.40	Kaffeepause	
11.10	Jürgen Dassow, Florin Manea, Robert Mercaş, Mike Müller *Inner palindromic closure*	S. 62
	Paweł Gawrychowski, Florin Manea, Dirk Nowotka *Discovering hidden repetitions in words*	S. 65
	Peter Leupold *Unavoidability of primitive and palindromic words*	S. 68
	Martin Kutrib, Andreas Malcher, Matthias Wendlandt *Size of unary one-way multi-head finite automata*	S. 71

Automata and Logic for Concurrent Systems

Benedikt Bollig

Laboratoire Spécification et Vérification, École Normale Supérieure de Cachan &
Centre National de la Recherche Scientifique, France

Automata are a popular model of computer systems, making them accessible to formal methods and, in particular, synthesis and model checking. While classical finite-state automata are suitable to model *sequential* boolean programs, models of *concurrent* systems, involving several interacting processes, extend finite-state machines in several respects. Roughly, we may classify a system (or a system model) according to the following characteristics:

Form of Communication. Inter-process communication may be achieved, e.g., via shared variables or message passing. While boolean shared-variable programs usually give rise to finite-state systems so that classical methods are applicable for their analysis, message passing via a priori unbounded channels leads to undecidability of basic verification questions. However, putting some restrictions on the system (e.g., imposing a channel bound or restricting the system architecture) will allow us to infer positive results for both system synthesis and model checking.

System Architecture. The system architecture, connecting processes and arranging them in a certain way (e.g., as a pipeline or as a tree), may be static and known, or static but unknown, or it may change dynamically during a system execution. In the particular case where the topology is static but unknown, we deal with a parameterized setting. So, one will be interested in questions such as "Is the system correct no matter what the system architecture or the number of processes is?" or "Can one transform a specification into a system that is equivalent to the specification over all system architectures?". There has been a wide range of techniques for the verification of parameterized and dynamic systems. In the dynamic case, there are also close connections with the theory of words over infinite alphabets, where the alphabet may represent an unbounded supply of process identifiers.

Finite-State vs. Recursive Processes. Processes themselves can have finite state space (i.e., be modeled as finite-state automata) or recursive (i.e., be modeled as pushdown automata). Like processes communicating via message passing through unbounded channels, shared-variable recursive processes have an undecidable control-state reachability problem. However, under- and overapproximating the behavior of a system will still allow us to check certain system requirements.

In this talk, we survey automata models for some combinations of the above-mentioned features. We also present suitable specification formalisms (such as

monadic second-order logic, temporal logic, and high-level expressions). In particular, we will compare the expressive power of automata and logic, give translations of specifications into automata, and show, for some cases, how to solve the model-checking problem: "Does a given automaton satisfy its specification?".

Weighted logics for unranked tree automata

Manfred Droste

Institut für Informatik,
Universität Leipzig,
Leipzig, Germany
droste@informatik.uni-leipzig.de

We define a weighted monadic second order logic for unranked trees and the concept of weighted unranked tree automata, and we investigate the expressive power of these two concepts, with the weights being computed in any semiring. We show that weighted tree automata and a syntactically restricted weighted MSO-logic have the same expressive power in case the semiring is commutative or in case we deal only with ranked trees, but, surprisingly, not in general. This demonstrates a crucial difference between the theories of ranked trees and unranked trees in the weighted case. If time permits, we will also consider recent extensions of the weight structures to valuation monoids. These contain all semirings, but also average computations of real numbers as weights, recently considered by Henzinger and others.

References

1. M. Droste and H. Vogler. Weighted logics for unranked tree automata. *Theory of Computing Systems* 48 (2011), pp. 23-47.
2. M. Droste, D. Götze, S. Märcker and I. Meinecke. Weighted tree automata over valuation monoids and their characterizations by weighted logics. *Algebraic Foundations in Computer Science (eds. W. Kuich, G. Rahonis)*, Lecture Notes in Computer Sciences, vol. 7020, Springer, 2011, pp. 30–55.

Constructions and Algorithms for ω-Automata

Christof Löding

RWTH Aachen, Germany
loeding@cs.rwth-aachen.de

Automata on infinite words, or ω-automata, have their origin as a tool in a decision procedure for the monadic second-order (MSO) logic over the structure $(\mathbb{N}, +1)$ of the natural numbers with the successor function [3]. Each such formula can be translated into a nondeterministic Büchi automaton, which is syntactically the same as a standard nondeterministic finite automaton. However, the semantics refers to infinite words, namely an infinite word is accepted if there is a run that visits an accepting state infinitely often. The satisfiability problem for the MSO formulas then reduces to an emptiness test for the resulting automaton, which can be solved with standard graph algorithms.

With the rising interest in formal methods for verification, Büchi automata came back into focus. In [12] it was shown that linear temporal logic (LTL) can be translated into Büchi automata with only a single exponential blow-up (compared to the non-elementary complexity of the translation from MSO). Since LTL is a popular logic for specifying properties of system executions, Büchi automata have become part of verification tools like SPIN [5].

This new interest also stimulated new research for classical problems like the complementation problem for Büchi automata. As opposed to finite automata, the subset construction does not work for complementation (or determinization), and it can be shown that a blow-up of 2^n is not sufficient for the complementation of n-state Büchi automata [8, 13]. Many constructions and optimizations of existing constructions have been proposed and also evaluated experimentally (see, for example, [6, 9, 4, 2, 11]).

Another interesting logic that can be translated into ω-automata is the first-order (FO) logic over the structure $(\mathbb{R}, \mathbb{Z}, <, +)$, that is, the real numbers with a predicate for integers, the less than relation, and addition. As integers can be seen as finite words (e.g., in their decimal representation) real numbers naturally correspond to infinite words. An interesting aspect of $FO(\mathbb{R}, \mathbb{Z}, <, +)$ is that it can be handled by a subclass of ω-automata, called deterministic weak Büchi automata [1]. Deterministic weak automata have many good properties similar to standard deterministic finite automata on finite words. Most notably, their minimal automata can be characterized using a congruence on finite words [10], and a minimal automaton can be computed efficiently [7].

In this talk I will survey the connections between ω-automata and logics, as well as some of the central constructions and algorithmic problems like complementation and minimization, as discussed above.

References

1. Bernard Boigelot, Sébastien Jodogne, and Pierre Wolper. On the use of weak automata for deciding linear arithmetic with integer and real variables. In *Automated Reasoning, First International Joint Conference, IJCAR 2001*, volume 2083 of *Lecture Notes in Computer Science*, pages 611–625. Springer, 2001.
2. Stefan Breuers, Christof Löding, and Jörg Olschewski. Improved Ramsey-based Büchi complementation. In *FoSSaCS 2012*, volume 7213 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2012.
3. J. Richard Büchi. On a decision method in restricted second order arithmetic. In *International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.
4. Seth Fogarty, Orna Kupferman, Moshe Y. Vardi, and Thomas Wilke. Unifying Büchi complementation constructions. In *CSL*, 2011.
5. Gerard J. Holzmann. *The Spin Model Checker - Primer and Reference Manual.* Addison-Wesley, 2003.
6. Detlef Kähler and Thomas Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, ICALP 2008, Part I*, volume 5125 of *Lecture Notes in Computer Science*, pages 724–735. Springer, 2008.
7. Christof Löding. Efficient minimization of deterministic weak ω-automata. *Information Processing Letters*, 79(3):105–109, 2001.
8. Max Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.
9. Sven Schewe. Büchi complementation made tight. In *STACS*, volume 3 of *LIPIcs*, pages 661–672. Schloss Dagstuhl, 2009.
10. Ludwig Staiger. Finite-state ω-languages. *Journal of Computer and System Sciences*, 27:434–448, 1983.
11. Ming-Hsien Tsai, Seth Fogarty, Moshe Y. Vardi, and Yih-Kuen Tsay. State of Büchi complementation. In *CIAA*, volume 6482 of *Lecture Notes in Computer Science*, pages 261–271. Springer, 2010.
12. Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings, Symposium on Logic in Computer Science, 16-18 June 1986, Cambridge, Massachusetts, USA*, pages 332–344. IEEE Computer Society, 1986.
13. Qiqi Yan. Lower bounds for complementation of ω-automata via the full automata technique. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming, ICALP'06*, volume 4052 of *Lecture Notes in Computer Science*, pages 589–600. Springer, 2006.

Logikbeschreibungen regulärer Sprachen

Manfred Kufleitner

Formale Methoden der Informatik, Universität Stuttgart

Für reguläre Sprachen stehen viele verschiedene Beschreibungsmechanismen zur Verfügung. Die bekanntesten davon sind sicherlich reguläre Ausdrücke, endliche Automaten und rechtslineare Grammatiken. Anfang der 1960er konnte Büchi beweisen, dass monadische Logik zweiter Stufe (MSO) ebenfalls genau die regulären Sprachen beschreibt. Dieses Resultat ist auch heute noch vor allem im Zusammenhang mit der formalen Verifikation von sequentiellen Systemen hochrelevant. Es zeigt sich jedoch, dass viele Berechnungsprobleme wie das Leerheitsproblem oder das Inklusionsproblem bei Eingabe von MSO-Formeln nicht elementar lösbar sind. Andererseits ist für viele der zu überprüfenden Eigenschaften nicht die volle Ausdrucksstärke von MSO notwendig. Dies führt auf den Begriff des Logikfragments. Für eingeschränktere Fragmente stehen oft effizientere Verfahren zur Verfügung. Zum Verständnis eines gegebenen Fragments $\mathcal{F}$ stellen sich neben effizienten Algorithmen für das Leerheits- und das Inklusionsproblem die folgenden Fragen. Welche Eigenschaften lassen sich in $\mathcal{F}$ formulieren? Wie kann man entscheiden, ob sich eine gegebene Sprache (z.B. als Automat) in dem Fragment $\mathcal{F}$ definieren lässt? Welche Abschlusseigenschaften haben die $\mathcal{F}$-definierbaren Sprachen? In dem Vortrag werden sowohl klassische als auch aktuelle Entwicklungen aus diesem Bereich vorgestellt. Eine zentrale Rolle spielt hierbei der sogenannte algebraische Ansatz.

Isomorphie von endlich präsentierten Strukturen

Markus Lohrey

Universität Siegen

Die algorithmische Modelltheorie beschäftigt sich mit den logischen Eigenschaften von potentiell unendlichen Strukturen, die sich auf eine gewisse Art endlich repräsentieren lassen. Beispiele hierfür sind Pushdowngraphen, automatische Strukturen, sowie berechenbare Strukturen. Ein zentrales Problem in diesem Zusammenhang ist das Isomorphieproblem: Sind zwei durch endliche Beschreibungen repräsentierte Strukturen isomorph. So ist z.B. bekannt, dass das Isomorphieproblem für Pushdowngraphen (welche durch Kellerautomaten beschrieben sind) entscheidbar ist (Courcelle 1989), während das Isomorphieproblem für berechenbare Strukturen hochgradig unentscheidbar (Σ_1^1-vollständig) ist.

In dem Vortrag soll ein Überblick über das Isomorphieproblem von endlich präsentierten Strukturen gegeben werden. Einen Schwerpunkt bilden dabei automatische Strukturen und Varianten dieser. Eine Struktur ist automatisch, falls das Universum der Struktur eine reguläre Sprache ist, und alle Relationen durch synchrone Mehrbandautomaten erkannt werden können. Automatische Strukturen sind eine Teilklasse der berechenbaren Strukturen, haben jedoch im Gegensatz zu letzteren einige positive algorithmische Eigenschaften (insbesondere ist die Theorie 1. Stufe einer automatischen Struktur entscheidbar). Khoussainov, Nies, Rubin, und Stephan konnten jedoch zeigen, dass auch für automatische Strukturen das Isomorphieproblem unentscheidbar ist. In dem Vortrag werde ich auf wichtige Teilklassen von automatischen Strukturen (z.B. automatische Bäume, automatische lineare Ordnungen, automatische Äquivalenrelationen) eingehen.

Ketten evolutionärer Prozessoren mit Random-Context-Filtern*

Bianca Truthe

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

Zusammenfassung Es wird gezeigt, dass jede rekursiv aufzählbare Sprache von einem akzeptierenden Netz evolutionärer Prozessoren (ANEP) in der Form einer Kette (jeder Prozessor hat höchstens zwei Nachbarn), eines Ringes (jeder Prozessor hat genau zwei Nachbarn) oder eines Rades (ein Ring mit einem zusätzlichen Mittelknoten) akzeptiert wird, wobei die Kommunikation zwischen den Prozessoren durch Filter gesteuert wird, die die Ab- oder Anwesenheit gewisser Buchstaben in den zu kommunizierenden Wörtern überprüfen. Die Größe der konstruierten Netze hängt nicht von der akzeptierten Sprache ab. Hiermit werden einige Fragen beantwortet, die von J. Dassow und F. Manea in der auf der Konferenz „Descriptional Complexity of Formal Systems“ (DCFS) 2010 vorgestellten Arbeit ([3]) offen gelassen wurden.

1 Einleitung

Netze von Sprachprozessoren wurden von E. Csuhaj-Varjú und A. Salomaa eingeführt ([2]). Solch ein Netz kann als Graph angesehen werden, bei dem jeder Knoten Regeln und Wörter hat, die er entsprechend den Regeln ableitet, und die nach dem Passieren gewisser Filter über die Kanten zu anderen Knoten gelangen.

Von Punktmutationen in der Biologie inspiriert, haben J. Castellanos, C. Martín-Vide, V. Mitrana und J. Sempere in [1] Netze evolutionärer Prozessoren eingeführt. Dabei sind die verwendeten Regeln Ersetzen eines Buchstabens durch einen anderen, Einfügen eines Buchstabens und Löschen eines Buchstabens.

Akzeptierende Netze evolutionärer Prozessoren wurden von M. Margenstern, V. Mitrana und M. J. Pérez-Jiménez in [4] eingeführt. In einem solchen Netz gibt es einen sogenannten Eingabe-Knoten, der zu Beginn der Berechnung das zu untersuchende Wort (Eingabewort) enthält (alle anderen Knoten enthalten keine Wörter), und einen oder mehrere sogenannte Ausgabe-Knoten. Ein Eingabewort wird genau dann akzeptiert, wenn ein (oder jeder) Ausgabe-Knoten irgendwann im Laufe der Berechnung ein Wort enthält. Die Kommunikation zwischen den Prozessoren wird durch Filter gesteuert, die die Ab- oder

* Die Arbeit wurde unter dem Titel *Chains of Evolutionary Processors with Random Context Filters* auf der Konferenz NCMA 2013 vorgestellt [5].

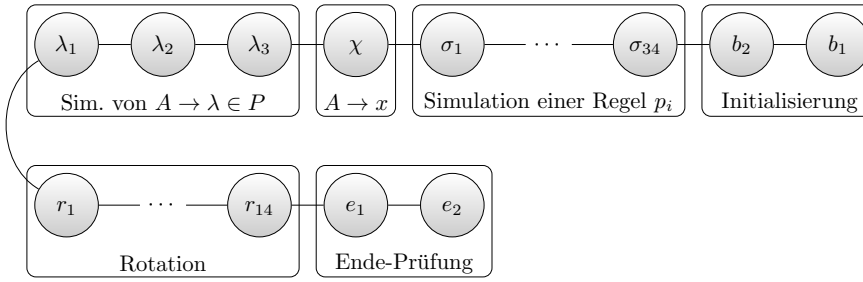
Anwesenheit gewisser Buchstaben in den zu kommunizierenden Wörtern überprüfen (Random-Context-Filter). Dabei wird zwischen Eingangs- und Ausgangsfiltern unterschieden. Jene Wörter die sich in einem Knoten K befinden, werden zu Beginn eines Kommunikationsschrittes vom Ausgangsfilter des Knotens K überprüft. Erfüllt ein Wort die Filterbedingungen, verlässt es den Knoten K und wird zu allen Nachbarn gesendet, andernfalls bleibt es im Knoten K und wird dort weiter verarbeitet. Jene Wörter, die von einem Knoten K zu einem Knoten K' übermittelt werden, werden vom Eingangsfilter des Knotens K' überprüft. Erfüllt ein Wort die Filterbedingungen, gelangt es in den Knoten K', andernfalls geht es verloren.

In der Arbeit [3] haben J. Dassow und F. Manea Untersuchungen zu akzeptierenden Netzen mit speziellen Formen hinsichtlich ihrer Mächtigkeit und Komplexität veröffentlicht. Insbesondere wurden Netze betrachtet, die die Form eines Sternes, Rades oder Gitters haben. In einem Stern gibt es einen zentralen Knoten, der mit jedem anderen Knoten in beide Richtungen verbunden ist, aber keine weiteren Kanten. In einem Rad gibt es einen zentralen Knoten, der mit jedem anderen Knoten in beide Richtungen verbunden ist, und jeder andere Knoten hat neben dem zentralen Knoten genau zwei weitere Nachbarn; der zentrale Knoten ist der Ausgabe-Knoten. Zu einem Gitternetz gibt es zwei natürliche Zahlen m und n und eine umkehrbar eindeutige Abbildung der Knoten auf die Paare (i, j) für $1 \leq i \leq m$ und $1 \leq j \leq n$ so, dass die Kanten gerade die Verbindungen zwischen den Knoten mit den Markierungen (i, j) und $(i+1, j)$, $(i-1, j)$, $(i, j+1)$ und $(i, j-1)$ darstellen (sofern $1 \leq i \pm 1 \leq m$ und $1 \leq j \pm 1 \leq n$ gilt).

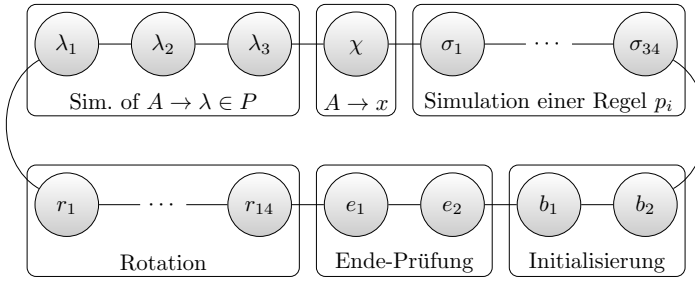
In der vorliegenden Arbeit werden die Untersuchungen fortgesetzt und einige offene Fragen aus der Arbeit [3] geklärt.

2 Ergebnisse

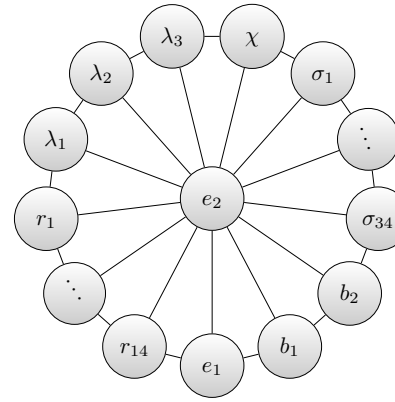
Jede rekursiv aufzählbare Sprache wird durch eine Kette von 56 evolutionären Prozessoren mit Random-Context-Filtern akzeptiert. Zu einer Grammatik in Kuroda-Normalform kann ein Netz mit folgender Struktur konstruiert werden, das genau die Sprache akzeptiert, die von der simulierten Grammatik erzeugt wird (die Linien stehen für Kanten in beide Richtungen):



Auf Grund des Eingangsfilters vom Ausgabe-Knoten N_{e_2} ändert ein Verbinden der Knoten N_{e_2} und N_{b_1} nichts an der akzeptierten Sprache, so dass die gleiche Sprache auch von einem Ring mit 56 Knoten akzeptiert wird:



Auch das Umformen zu dem nebenstehenden Rad ändert nichts an der akzeptierten Sprache. Der Ausgabe-Knoten N_{e_2} erwartet ein Symbol, das nur im Knoten N_{e_1} erzeugt wird. Folglich gelangen keine Wörter über die anderen Kanten hin zum Ausgabe-Knoten. Auch zwischen dem Eingabe-Knoten N_{b_1} und dem Knoten N_{e_1} findet keine direkte Kommunikation statt, da der Knoten N_{b_1} nur am Anfang ein Wort enthält und in diesem kein Hilfssymbol vorkommt, was aber vom Knoten N_{e_1} erwartet wird, und jedes Wort im Knoten N_{e_1} ein Hilfssymbol enthält, womit es nicht in den Knoten N_{b_1} gelangt.



Literatur

1. J. Castellanos, C. Martín-Vide, V. Mitrana, and J. M. Sempere. Solving NP-Complete Problems With Networks of Evolutionary Processors. In *IWANN 2001*, volume 2084 of *LNCS*, pages 621–628. Springer, 2001.
2. E. Csuhaj-Varjú and A. Salomaa. Networks of Parallel Language Processors. In *New Trends in Formal Languages – Control, Cooperation, and Combinatorics*, volume 1218 of *LNCS*, pages 299–318. Springer, 1997.
3. J. Dassow and F. Manea. Accepting Hybrid Networks of Evolutionary Processors with Special Topologies and Small Communication. In *DCFS 2010*, volume 31 of *EPTCS*, pages 68–77, 2010.
4. M. Margenstern, V. Mitrana, and M. J. Pérez-Jiménez. Accepting Hybrid Networks of Evolutionary Processors. In *10th Intern. Workshop on DNA Computing*, volume 3384 of *LNCS*, pages 235–246. Springer, 2004.
5. B. Truthe. Chains of Evolutionary Processors with Random Context Filters. In *NCMA 2013*, books@ocg.at. Österreichische Comp. Ges., Austria, 2013.

On Restarting Transducers*

Norbert Hundeshagen

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

Abstract. We study the computional power of restarting transducers, a recently introduced model for computing binary (word) relations. We show that the classes of relations defined by these machines are incomparable to the most common ones. Further, a restricted version, the so-called monotone restarting transducer, yield a class of relations that is almost equivalent to the pushdown relations.

1 Introduction

Restarting automata [7] were invented to model the so-called "analysis by reduction". Simply, this linguistic technique is a method to verify the (syntactical) correctness of a given sentence by a stepwise simplification, under the condition that every step preserves the correctness or incorrectness of the sentence processed. From a linguistic point of view, the verification of correctness (or incorrectness) is not the only goal of performing analysis by reduction, as it is also a useful tool to gain deeper information on the structure of sentences of natural languages, such as word dependency information [9] and morphological ambiguities [10]. Therefore, we are interested in transductions and in ways to compute them by restarting automata.

Several new computational models for realizing transductions, based on restarting automata, have been introduced in recent years. For instance, there are special types of restarting automata that are enhanced to produce tree structures that mirror dependency trees of sentences of natural languages (see e.g., [11]), or parallel communicating systems of restarting automata that realize binary relations [6].

Here we continue a more classical approach by investigating the computational power of restarting automata, extended by the capability to produce strings. These machines are called restarting transducers and were first introduced in [5]. There it was shown that the relations defined by types of restricted restarting transducers form a hierarchy inside the well-known class of rational relations.

* This extended abstract reports on results contained in [4].

2 Definitions and General Observations

A restarting transducer (RRWW-Td for short) consists of a finite-state control, a flexible tape with end markers, a read/write window of a fixed size working on that tape and a write-only oneway output tape. Formally, a restarting transducer is defined as a 9-tuple $T = (Q, \Sigma, \Delta, \Gamma, ¢, \$, q_0, k, \delta)$ where Q is the finite set of states, Σ and Γ are the finite input and tape alphabet, Δ is the finite output alphabet, $¢, \$ \notin \Gamma$ are the markers for the left and right border of the tape, $q_0 \in Q$ is the initial state, $k \geq 1$ is the size of the read/write window, and δ is the transition function.

Such a device works in cycles. In each cycle it moves the head right across the tape and at some point it performs a single rewrite operation that shortens the tape contents. Every cycle ends with a restart operation that forces the transducer to reset the internal state to the initial one and output a word over Δ. After a finite number of cycles, it halts and accepts (or rejects) while also producing some symbols. Thus, a binary relation $\mathrm{Rel}(T) \subseteq \Sigma^* \times \Delta^*$ is defined that consists of all pairs of words (u, v) for which there is an accepting computation started on u and finished with v on the output tape.

Obviously, the relations computed by restarting transducers are length-bounded: There is a constant c, such that for each pair (u, v) $(u \neq \varepsilon)$ in the relation, $|v| \leq c \cdot |u|$. This is for the reason that in every cycle the tape content has to be shortened while only a finite number of output symbols can be produced. Furthermore, we can show that for each recursively enumerable language L, there is a deterministic restarting transducer T such that L is the output language of T.

The latter immediately yields incomparability results to the most common classes of relations, the rational and pushdown relations (see e.g., [2, 3]).

3 Monotone Restarting Transducers

We turn to a more restricted type of restarting transducer by introducing the notion of monotonicity, known from the underlying automaton (see e.g., [8]). Informally, a restarting transducer is called monotone if in each of its computations that starts from an initial configuration, the distance between the right border and the rewrite position is never increased.

We show that the relations computed by transducers of this type are included in the pushdown relations. To establish this result we make use of the concept of input/output-relations associated to restarting automata (see [6]). Furthermore, by using a grammar-based characterization of the pushdown relations (see e.g., [1]) we derive an equivalence to a special subclass, the length-bounded pushdown relations.

References

1. A. V. Aho and J. D. Ullman. Properties of Syntax Directed Translations. *J. Comput. Syst. Sci.*, 3(3):319–334, 1969.

2. J. Berstel. *Transductions and Context-Free Languages.* Leitfäden der angewandten Mathematik und Mechanik. Teubner, 1979.
3. C. Choffrut and K. Culik II. Properties of Finite and Pushdown Transducers. *SIAM J. Comput.*, 12(2):300–315, 1983.
4. N. Hundeshagen. *Relations and Transductions Realized by Restarting Automata.* PhD thesis, Fachbereich Elektrotechnik/Informatik, Universität Kassel, 2013. To appear.
5. N. Hundeshagen and F. Otto. Characterizing the Rational Functions by Restarting Transducers. In A. H. Dediu and C. Martín-Vide, editors, *LATA*, volume 7183 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2012.
6. N. Hundeshagen, F. Otto, and M. Vollweiler. Transductions Computed by PC-Systems of Monotone Deterministic Restarting Automata. In M. Domaratzki and K. Salomaa, editors, *CIAA*, volume 6482 of *Lecture Notes in Computer Science*, pages 163–172. Springer, 2010.
7. P. Jancar, F. Mráz, M. Plátek, and J. Vogel. Restarting Automata. In H. Reichel, editor, *FCT*, volume 965 of *Lecture Notes in Computer Science*, pages 283–292. Springer, 1995.
8. T. Jurdzinski, F. Mráz, F. Otto, and M. Plátek. Degrees of Non-Monotonicity for Restarting Automata. *Theor. Comput. Sci.*, 369(1-3):1–34, 2006.
9. M. Lopatková, M. Plátek, and V. Kubon. Modeling Syntax of Free Word-Order Languages: Dependency Analysis by Reduction. In V. Matousek, P. Mautner, and T. Pavelka, editors, *TSD*, Lecture Notes in Computer Science, pages 140–147. Springer, 2005.
10. M. Plátek, M. Lopatková, and K. Oliva. Restarting Automata: Motivations and Applications. In M. Holzer, editor, *13. Theorietag 'Automaten und Formale Sprachen', Proc.*, pages 90 – 96, Technische Universität München, 2003.
11. M. Plátek, F. Mráz, and M. Lopatková. Restarting Automata with Structured Output and Functional Generative Description. In A. H. Dediu, H. Fernau, and C. Martín-Vide, editors, *LATA*, volume 6031 of *Lecture Notes in Computer Science*, pages 500–511. Springer, 2010.

Recent advances on valence automata as a generalization of automata with storage

Phoebe Buckheister and Georg Zetzsche

Technische Universität Kaiserslautern
zetzsche@cs.uni-kl.de

Abstract. A valence automaton over a monoid M is a finite automaton in which each edge carries an input word and an element of M. A word is then accepted if there is a run that spells the word such that the product of the monoid elements is the identity.
By choosing suitable monoids M, one can obtain various kinds of automata with storage as special valence automata. Examples include pushdown automata, blind multicounter automata, and partially blind multicounter automata. Therefore, valence automata offer a framework to generalize results on such automata with storage.
This talk will present recent advances in this direction. The addressed questions include: For which monoids do we have a Parikh's Theorem (as for pushdown automata)? For which monoids can we avoid silent transitions?

Let M be a monoid. We define a *valence automaton over* M to be a tuple $A = (Q, X, M, E, q_0, F)$, in which Q is a finite set of *states*, X is an alphabet, $E \subseteq Q \times X^* \times M \times Q$ is is a finite set of *edges*, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of *final states*.

A triple $(q, m, w) \in Q \times M \times X^*$ is called a *configuration*. For configurations $(q, m, w), (q', m', w')$, we write $(q, m, w) \Rightarrow_A (q', m', w')$ if there is an edge $(q, u, n, q') \in E$ such that $w' = wu$ and $m' = mn$. An edge (q, w, m, q') is called a *silent transition* (or *λ-transition*) if $w = \lambda$. The language *accepted by* A is then defined as

$$L(A) = \{w \in X^* \mid \exists q \in F : (q_0, 1, \lambda) \Rightarrow_A^* (q, 1, w)\}.$$

In other words, a word is accepted by A if there is a computation that reads the word and the product of the monoid elements on the edges is the identity. The class of languages accepted by valence automata over M will be denoted by $\mathsf{VA}(M)$. If we forbid silent transitions, the corresponding language class is denoted $\mathsf{VA}^+(M)$.

For many types of automata with storage, there is a monoid M such that valence automata behave like automata of the corresponding type. This is the case, for example, for *pushdown automata*, *blind counter automata*, *partially blind counter automata*, and *Turing machines*. For details on which monoids lead to these automata types, we refer the reader to [8].

Thus, valence automata allow us to ask how the structure of the storage mechanism influences the expressive power and decidability properties of automata: We can study for which monoids classical results about automata with storage still hold. This talk will present recent advances in this direction. The discussed questions include:

- For which monoids M does $\mathsf{VA}(M)$ contain only languages with semilinear Parikh image?
- For which monoids M do we have $\mathsf{VA}(M) \subseteq \mathsf{CF}$?
- For which monoids M can we eliminate λ-transitions, that is, when does $\mathsf{VA}(M) = \mathsf{VA}^+(M)$?
- Which language classes arise as $\mathsf{VA}(M)$?

The results mentioned in the talk appeared in [7, 8, 1].

References

1. P. Buckheister and G. Zetzsche. Semilinearity and context-freeness of languages accepted by valence automata, 2013. To appear in Proceedings of MFCS 2013. Full version available at `http://arxiv.org/abs/1306.3260`.
2. H. Fernau and R. Stiebe. Sequential grammars and automata with valences. *Theor. Comput. Sci.*, 276(1-2):377–405, 2002.
3. M. Kambites. Formal languages and groups as memory. *Communications in Algebra*, 37:193–208, 2009.
4. V. Mitrana and R. Stiebe. Extended finite automata over groups. *Discrete Applied Mathematics*, 108(3):287–300, 2001.
5. E. Render. *Rational Monoid and Semigroup Automata.* PhD thesis, University of Manchester, 2010.
6. E. Render and M. Kambites. Rational subsets of polycyclic monoids and valence automata. *Information and Computation*, 207(11):1329 – 1339, 2009.
7. G. Zetzsche. On the capabilities of grammars, automata, and transducers controlled by monoids. In *Proceedings of ICALP 2011*, volume 6756 of *LNCS*, pages 222–233. Springer Berlin Heidelberg, 2011.
8. G. Zetzsche. Silent transitions in automata with storage. In *Proceedings of ICALP 2013*, volume 7966 of *LNCS*, pages 434–445. Springer Berlin Heidelberg, 2013. Full version available at `http://arxiv.org/abs/1302.3798`.

Input-Driven Queue Automata* **

Martin Kutrib[1], Andreas Malcher[1], Carlo Mereghetti[2], Beatrice Palano[2], and Matthias Wendlandt[1]

[1] Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
{kutrib,malcher,matthias.wendlandt}@informatik.uni-giessen.de
[2] Dip. Informatica, Univ. degli Studi di Milano, v. Comelico 39, 20135 Milano, Italy
{mereghetti,palano}@di.unimi.it

We introduce and study the model of input-driven queue automata. On such devices, the input letters uniquely determine the operations on the memory store which is organized as a queue. In particular, we consider the case where only a finite number of turns on the queue is allowed. The resulting language families share with regular languages many desirable properties. We show that emptiness, finiteness, universality, inclusion, and equivalence are decidable. In contrast, all these problems are shown to be undecidable if the condition on a finite number of turns is dropped. Furthermore, we investigate closure under Boolean operations. Finally, the existence of an infinite and tight hierarchy depending on the number of turns is also proved.

* Partially supported by CRUI/DAAD under the project "Programma Vigoni: Descriptional Complexity of Non-Classical Computational Models", and MIUR under the project "PRIN: Automi e Linguaggi Formali: Aspetti Matematici e Applicativi".

** Summary of a paper presented at CIAA 2013, Halifax, Canada (M. Kutrib, A. Malcher, C. Mereghetti, B. Palano, M. Wendlandt: Input-Driven Queue Automata: Finite Turns, Decidability, and Closure Properties. In: S. Konstantinidis (Ed.), Implementation and Application of Automata (CIAA 2013), LNCS 7982, 2013, 232–243).

Queue Automata of Constant Length

Sebastian Jakobi[1,⋆] Katja Meckel[1,⋆]
Carlo Mereghetti[2,⋆,⋆⋆] Beatrice Palano[2,⋆,⋆⋆]

[1] Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
{jakobi,meckel}@informatik.uni-giessen.de
[2] Dip. Informatica, Univ. degli Studi di Milano, v. Comelico 39, 20135 Milano, Italy
{mereghetti,palano}@di.unimi.it

Abstract. We introduce and study the notion of *constant length queue automata*, as a formalism for representing regular languages. We show that their descriptional power outperforms that of traditional finite state automata, of constant height pushdown automata, and of straight line programs for regular expressions, by providing optimal exponential and double-exponential size gaps. Moreover, we prove that constant height pushdown automata can be simulated by constant length queue automata paying only by a linear size increase, and that removing nondeterminism in constant length queue automata requires an optimal exponential size blow-up, against the optimal double-exponential cost for determinizing constant height pushdown automata.

1 Introduction

It is well known that *computational power* can be tuned by restricting the way memory is accessed. To get a quick overview of this phenomenon, one may start from the traditional model of one-way Turing machines, where memory is modeled by a potentially unbounded working tape that can be freely accessed. If we impose a LIFO usage of the working tape, still keeping unboundedness, then we obtain *pushdown automata*, whose computational power (context-free languages) is strictly lower. On the other hand, by imposing a FIFO access policy, we get *queue automata*, whose power gets back to that of Turing machines.

In all cases, by fixing a *constant* bound — i.e., not depending on the input length — on the amount of available memory, the computational power boils down to that of finite state automata, regardless of memory usage mode. For *constant memory machines*, it is then worth investigating how the way in which memory is accessed affects their *descriptional power*, in other words, their capability of succinctly representing regular languages.

This line of research is settled in [7], where the notion of a *constant height pushdown automaton* is introduced and studied from a descriptional complexity

⋆ Partially supported by CRUI/DAAD under the project "Programma Vigoni: Descriptional Complexity of Non-Classical Computational Models".
⋆⋆ Partially supported by MIUR under the project "PRIN: Automi e Linguaggi Formali: Aspetti Matematici e Applicativi".

perspective. Roughly speaking, this device is a traditional pushdown automaton (see, e.g., [8]) with a built-in constant limit on the height of the pushdown. Optimal exponential and double-exponential gaps are proved between the size of constant height deterministic and nondeterministic pushdown automata (DPDAs and NPDAs, respectively), deterministic and nondeterministic finite state automata (DFAs and NFAs), and that of classical regular expressions. Moreover, also the notion of a straight line program for regular expressions (SLP) is introduced, as a formalism equivalent to a constant height NPDA from a size point of view. In [3, 2], the fundamental problem of removing nondeterminism in constant height NPDAs is tackled, and a double-exponential size blow-up for determinization is emphasized. Finally, the descriptional cost of boolean operations on constant height DPDAs and NPDAs is analyzed in [4, 6].

We investigate the descriptional advantages of substituting the pushdown with a *queue* storage of fixed size by introducing the notion of a *constant length queue automaton*. Basically, this device is a traditional queue automaton (see, e.g., [1, 5]), where the length of the queue cannot grow beyond a fixed constant limit.

2 Results

For the sake of conciseness, we will be using the designation "constant memory automaton" to denote either a constant height NPDA or a constant length NQA. For constant memory automata, a fair *size measure* (see, e.g., [4, 7]) takes into account all the components the device consists of, i.e.: (i) the number of states of its finite control, (ii) the size of the memory alphabet, and (iii) the memory limit.

As for constant height pushdown automata, we single out optimal exponential and double-exponential gaps between the size of constant height deterministic and nondeterministic queue automata (DQAs and NQAs, respectively) and DFAs and NFAs. However, differently from constant height pushdown automata, we prove that a queue storage enables a size-efficient handling of nondeterminism. Precisely, we show that NFAs can be simulated by constant length DQAs paying by only a *linear* size increase. This, in turn, leads us to prove that the optimal cost of removing nondeterminism in constant length NQAs is only *exponential*, in sharp contrast with the optimal double-exponential blow-up above pointed out for the pushdown case.

The higher descriptional power of a queue vs. a pushdown storage in a constant setting is also emphasized when we show that constant height NPDAs (resp., DPDAs) can be simulated by constant length NQAs (resp., DQAs), paying by only a *linear* size increase. On the other hand, the opposite simulations have optimal exponential costs; this is witnessed by proving, that constant length DQAs can be exponentially smaller than equivalent SLPs, and this gap is optimal.

For reader's ease of mind, we sum up in Figure 1 the main relations on the sizes of the different types of formalisms for regular languages considered in this paper:

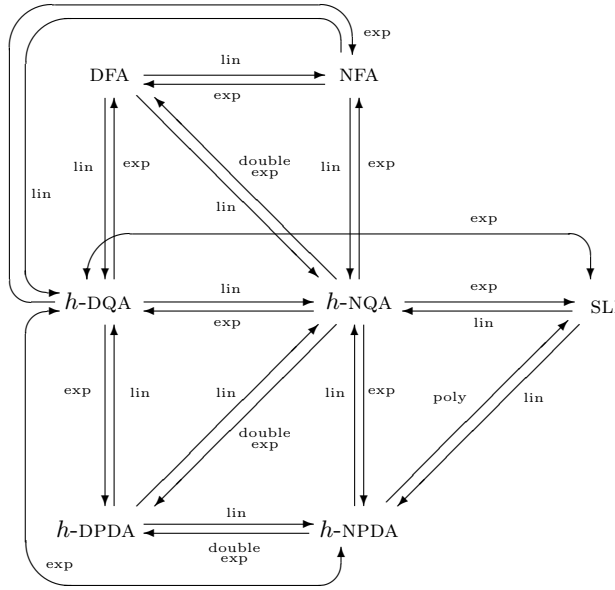


Fig. 1. Costs of simulations among different types of formalisms defining regular languages. Here h-DPDA (h-NPDA) denotes constant height DPDA (NPDA, respectively), while h-DQA (h-NQA) denotes constant length DQA (NQA, respectively). An arc labeled by lin (poly, exp, double exp) from a vertex A to a vertex B means that, given a representation of type A, we can build an equivalent representation of type B, paying by a linear (polynomial, exponential, double-exponential, respectively) increase in the size.

References

1. E. Allevi, A. Cherubini, and S. Crespi-Reghizzi. Breadth-first phrase structure grammars and queue automata. In *MFCS*, volume 324 of *Lecture Notes in Computer Science*, pages 162–170. Springer, 1988.
2. Z. Bednárová, V. Geffert, C. Mereghetti, and B. Palano. Removing nondeterminism in constant height pushdown automata. *Submitted for publication.*
3. Z. Bednárová, V. Geffert, C. Mereghetti, and B. Palano. Removing nondeterminism in constant height pushdown automata. In *DCFS*, volume 7386 of *Lecture Notes in Computer Science*, pages 76–88. Springer, 2012.
4. Z. Bednárová, V. Geffert, C. Mereghetti, and B. Palano. The size-cost of boolean operations on constant height deterministic pushdown automata. *Theor. Comput. Sci.*, 449:23–36, 2012.
5. A. Cherubini, C. Citrini, S. Crespi-Reghizzi, and D. Mandrioli. Qrt fifo automata, breath-first grammars and their relations. *Theor. Comput. Sci.*, 85(1):171–203, 1991.
6. V. Geffert, Z. Bednárová, C. Mereghetti, and B. Palano. Boolean language operations on nondeterministic automata with a pushdown of constant height. In *CSR*, volume 7913 of *Lecture Notes in Computer Science*, pages 100–111. Springer, 2013.
7. V. Geffert, C. Mereghetti, and B. Palano. More concise representation of regular languages by automata and regular expressions. *Inf. Comput.*, 208(4):385–394, 2010.
8. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to automata theory, languages, and computation - (2. ed.).* Addison-Wesley series in computer science. Addison-Wesley-Longman, 2001.

Nondeterministic Biautomata and Their Descriptional Complexity

(Extended Abstract)[⋆]

Markus Holzer and Sebastian Jakobi

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

Biautomata were recently introduced in [4] as a generalization of ordinary deterministic finite automata. A biautomaton consists of a *deterministic* finite control, a read-only input tape, and two reading heads, one reading the input from left to right, and the other head reading the input from right to left. An input word is accepted by a biautomaton, if there is an accepting computation starting with the heads on the two ends of the word and meeting somewhere in an accepting state. Although the choice of reading a symbol by either head is nondeterministic, the determinism of the biautomaton is enforced by two properties, which will be described later. Descriptional complexity issues for deterministic biautomata were addressed in [3]. We focus on the descriptional complexity of *nondeterministic* biautomata, which are defined as follows: a *nondeterministic biautomaton* is a sixtuple $A = (Q, \Sigma, \cdot, \circ, I, F)$, where Q is a finite set of *states*, Σ is an *alphabet*, $\cdot\colon Q \times \Sigma \to 2^Q$ is the *forward transition function*, $\circ\colon Q \times \Sigma \to 2^Q$ is the *backward transition function*, $I \subseteq Q$ is the set of *initial states*, and $F \subseteq Q$ is the set of *final states*. The transition functions $\cdot$ and $\circ$ are extended to words in the following way, for every word $v \in \Sigma^*$ and letter $a \in \Sigma$:

$$q \cdot \lambda = \{q\}, \quad q \cdot av = \bigcup_{p \in (q \cdot a)} p \cdot v, \quad \text{and} \quad q \circ \lambda = \{q\}, \quad q \circ va = \bigcup_{p \in (q \circ a)} p \circ v,$$

and further, both $\cdot$ and $\circ$ can be extended to sets of states $S \subseteq Q$, and $w \in \Sigma^*$ by $S \cdot w = \bigcup_{p \in S} p \cdot w$, and $S \circ w = \bigcup_{p \in S} p \circ w$. The biautomaton A *accepts* the word $w \in \Sigma^*$, if and only if $w = u_1 u_2 \ldots u_k v_k \ldots v_2 v_1$, for some words $u_i, v_i \in \Sigma^*$ with $1 \leq i \leq k$, such that $[((\ldots((((I \cdot u_1) \circ v_1) \cdot u_2) \circ v_2) \ldots) \cdot u_k) \circ v_k] \cap F \neq \emptyset$. The *language accepted* by A is defined as $L(A) = \{\, w \in \Sigma^* \mid A \text{ accepts } w \,\}$. Moreover, the biautomaton A is *deterministic*, if $|I| = 1$, and $|q \cdot a| = |q \circ a| = 1$ for all states $q \in Q$ and letters $a \in \Sigma$. The automaton A has the $\diamond$-*property*, if $(q \cdot a) \circ b = (q \circ b) \cdot a$, for every state $q \in Q$ and $a, b \in \Sigma$. Further, A has the F-property, if $q \cdot a \cap F \neq \emptyset$ if and only if $q \circ a \cap F \neq \emptyset$, for every state $q \in Q$ and letter $a \in \Sigma$. A deterministic biautomaton as defined above

⋆ This is an extended abstract of: M. Holzer, S. Jakobi. *Nondeterministic Biautomata and Their Descriptional Complexity.* In H. Jürgensen, R. Reis (eds.), *Proc.* 15*th DCFS*, volume 8031 in LNCS, pages 112–123, London, Ontario, Canada, July 2013. Springer.

that has both the $\diamond$- and the F-property is exactly what is called a biautomaton in [4]. Nondeterministic biautomata characterize the family of linear context-free languages [5], while nondeterministic biautomata that have the $\diamond$-property accept only regular languages. In fact, we are able to prove the following result.

Theorem 1. *The trade-off between deterministic or nondeterministic biautomata with or without the F-property and deterministic or nondeterministic biautomata that satisfy at least the $\diamond$-property is non-recursive.* □

By a straight-forward power-set construction, one can convert any n-state nondeterministic biautomaton into an equivalent deterministic biautomaton having at most 2^n states—this construction preserves both the $\diamond$- and the F-property. An exponential lower bound for the determinization of nondeterministic biautomata with both properties is given in the following result.

Theorem 2. *For all integers $n \geq 1$ there is a binary regular language L_n accepted by a nondeterministic biautomaton with $\diamond$-, and F-property that has $3n+2$ states, and for which every equivalent deterministic biautomaton with $\diamond$- and F-property needs at least $2^{2n} + 1$ states.* □

We also study the costs for the conversions from deterministic or nondeterministic finite automata (DFAs, NFAs), syntactic monoids, and regular expressions into nondeterministic biautomata that have the $\diamond$- and F-property. To prove lower bounds for such nondeterministic biautomata, the following generalization of the fooling set technique [1] is useful.

Theorem 3. *A set $S = \{ (x_i, y_i, z_i) \mid x_i, y_i, z_i \in \Sigma^*, 1 \leq i \leq n \}$ is a* bi-fooling set *for a language $L \subseteq \Sigma^*$ if the following two properties hold:*

1. *for $1 \leq i \leq n$ it is $x_i y_i z_i \in L$, and*
2. *for $1 \leq i, j \leq n$, with $i \neq j$, it is $x_i y_j z_i \notin L$ or $x_j y_i z_j \notin L$.*

If S is a bi-fooling set for the language L, then any nondeterministic biautomaton with both the $\diamond$-property and the F-property that accepts the language L has at least $|S|$ states. □

Table 1 summarizes our results on the costs of the above mentioned conversions. For comparison we also list the results from [3] for the conversions from DFAs, NFAs, and syntactic monoids to deterministic biautomata with both properties. Except for the conversion from regular expressions to nondeterministic biautomata, the indicated bounds are tight bounds, i.e., matching lower and upper bounds. We exemplarily present the results on the conversion from regular expressions to biautomata. We measure the size of a regular expression r by its *alphabetic width*, which is the number of occurrences of symbols from the underlying alphabet in the expression. The following upper bound is obtained by adapting the Glushkov construction [2] to biautomata.

Theorem 4. *Let r be a regular expression of alphabetic width n. Then there is a nondeterministic biautomaton A with $L(A) = L(r)$ that has $(n+1)^2$ states. Further, A has the $\diamond$- and the F-property.* □

Convert from ...	... to Biautomaton	
	deterministic, $\diamond$, F	nondeterministic, $\diamond$, F
DFA	$n \cdot 2^n - 2 \cdot (n-1)$	n^2
NFA	$2^{2n} - 2 \cdot (2^n - 1)$	n^2
syntactic monoid	n^2	n
regular expression		$n^2 \leq \cdot \leq (n+1)^2$

Table 1. Tight bounds for conversions from different models describing regular languages to deterministic or nondeterministic biautomata with the $\diamond$-property and the F-property. The results on deterministic biautomata are from [3]. For the conversions starting from a DFA or NFA, the integer n is the number of states of the finite automaton, when starting from a syntactic monoid, the number n is the size of the monoid, and for regular expressions, the integer n is the alphabetic width of the expression.

The following result provides a lower bound for this conversion.

Theorem 5. *For all integers $n \geq 1$ there is a binary language L_n with alphabetic width n, such that any nondeterministic biautomaton with the $\diamond$- and the F-property needs n^2 states to accept the language L_n.* □

The results and constructions for nondeterministic biautomata with $\diamond$- and F-property are evidence that this automaton model is a reasonable nondeterministic counterpart of the model of biautomata, as introduced in [4]. Concerning the F-property, its influence on the size of the biautomata is yet to be studied. By close inspection of the proof that biautomata with $\diamond$-property accept regular languages, one can deduce a quadratic upper bound for converting biautomata with $\diamond$-property into equivalent nondeterministic finite automata. With the conversions from finite automata to biautomata one can obtain upper bounds for enforcing the F-property, while preserving the $\diamond$-property—in case of nondeterministic biautomata, the bound is polynomial, and for deterministic biautomata it is exponential. The search for tight bounds for these conversions is left as an open problem.

References

1. J.-C. Birget. Intersection and union of regular languages and state complexity. *Inf. Process. Lett.*, 43:185–190, 1992.
2. V. M. Glushkov. The abstract theory of automata. *Russ. Math. Surv.*, 16:1–53, 1961.
3. G. Jirásková and O. Klíma. Descriptional complexity of biautomata. In M. Kutrib, N. Moreira, and R. Reis, editors, *Proc. 14th DCFS*, volume 7386 of *LNCS*, pages 196–208, Braga, Portugal, July 2012. Springer.
4. O. Klíma and L. Polák. On biautomata. RAIRO – Theo. Inf. Appl. 46(4):573–592, Oct. 2012.
5. R. Loukanova. Linear context free languages. In C. B. Jones, Z. Liu, and J. Woodcock, editors, *Proc. 4th ICTAC*, volume 4711 of *LNCS*, pages 351–365, Macau, China, Sept. 2007. Springer.

Deskriptive Pattern und Ketten von Patternsprachen

Dominik D. Freydenberger

Goethe-Universität, Frankfurt am Main

Zusammenfassung Die in diesem Vortrag vorgestellte Forschung setzt die Untersuchungen zur (Nicht-)Existenz deskriptiver Pattern fort, die von Freydenberger und Reidenbach begonnen wurden (*Existence and nonexistence of descriptive patterns*, Theor. Comput. Sci. 411 (2010)). Als technischer Hauptbeitrag können *Kettensysteme* betrachtet werden, ein neuer Mechanismus der die Arbeit mit Ketten von terminalfreien E-Patternsprachen erleichtert. Diese wiederum sind ein wichtiges Werkzeug in Beweisen zu deskriptiven Pattern. Anhand von Kettensystemen lässt such der Hauptbeweis aus Freydenberger und Reidenbach 2010 vereinfachen und verallgemeinern, zudem führen sie zu neuen Erkenntnisse über die Topologie der Klasse der terminalfreien E-Patternsprachen und zur Frage welche Sprachen sich nicht durch solche Patternsprachen approximieren lassen.

1 Ein kurzer Überblick

Ein *Pattern* ist ein Wort $\alpha \in (X \cup \Sigma)^+$, wobei X ein *Variablenalphabet* und Σ ein *Terminalalphabet* ist. Pattern können als kompakte und natürliche Sprachzeuger benutzt werden; ein Wort $w \in \Sigma^*$ gehört zur Sprache $L_{\mathrm{E}}(\alpha)$ eines Patterns α wenn es durch Ersetzen aller Variablen in α durch Terminalwörter erzeugt werden kann (wobei gleiche Variablen gleich ersetzt werden). Formaler gesprochen ist

$$L_{\mathrm{E}}(\alpha) := \{\sigma(\alpha) \mid \sigma \text{ ist eine Substitution}\},$$

wobei eine *Substitution* ein Homomorphismus $\sigma : (X \cup \Sigma)^* \to \Sigma^*$ ist, so dass $\sigma(a) = a$ für alle $a \in \Sigma$ gilt.

Zum Beispiel erzeugt das Pattern $\alpha := xx\,\mathtt{a}\,\mathtt{b}\,yy$ (mit $x, y \in X$ und $\mathtt{a}, \mathtt{b} \in \Sigma$) die Sprache $L_{\mathrm{E}}(\alpha) = \{uu\,\mathtt{a}\,\mathtt{b}\,vv \mid u, v \in \Sigma^*\}$. Gewöhnlich unterscheidet man in der Literatur zwischen *E-Patternsprachen* (bei denen das Ersetzen von Variablen durch das leere Wort *E*rlaubt ist) und *NE-Patternsprachen* (bei denen das Löschen von Variablen *N*icht *E*rlaubt ist). Im Gegensatz zu ihren ähnlichen Definitionen haben die Klasse der E- und die Klasse der NE-Patternsprachen oft sehr unterschiedliche Eigenschaften. Dieser Vortrag befasst sich ausschließlich mit E-Patternsprachen.

Patternsprachen wurden zuerst von Angluin [1] in der Lerntheorie eingeführt und, ausgehend von Jiang et al. [6], auch in der Sprachtheorie ausgiebig untersucht. Eine vergleichsweise aktuelle Übersicht zu Patternsprachen in diesen beiden Gebieten finden sich in Ng und Shinohara [7] und Salomaa [8].

Wir nennen ein Pattern δ *deskriptiv* für eine Sprache $L \subseteq \Sigma^*$, wenn

1. $L_{\mathrm{E}}(\delta) \supseteq L$ gilt, und außerdem
2. kein Pattern γ existiert, für das $L_{\mathrm{E}}(\delta) \supset L_{\mathrm{E}}(\gamma) \supseteq L$ gilt.

Ein Pattern, das für eine Sprache L deskriptiv ist, kann also als eine der besten Abschätzungen von L verstanden werden (im Sinne von inklusionsminimalen Generalisierungen), die innerhalb der Klasse der E-Patternsprachen möglich sind. Aus diesem Grunde spielen deskriptive Pattern eine wichtige Rolle in der Lerntheorie (bereits seit [1]). Dieser Ansatz wurde, zusammen mit Resultaten und Techniken aus Freydenberger und Reidenbach [4] zur Existenz deskriptiver Pattern, in Freydenberger und Reidenbach [5] weiterentwickelt zur *deskriptiven Generalisierung*, einem mächtigen Verfahren zum approximativen Lernen mittels deskriptiver Pattern.

Dieses Lernverfahren wiederum wurde kürzlich durch Freydenberger und Kötzing [3] zum Lernen von eingeschränkten regulären Ausdrücken verwendet, die von großer Bedeutung in der XML-Schemainferenz sind. Diese Entwicklung kann durchaus als überraschend betrachtet werden, da die Klasse der regulären Sprachen und die Klasse der Patternsprachen unvergleichbar sind und eine vollkommen unterschiedliche Topologie besitzen. Dennoch ließen sich viele der Techniken zu deskriptiven Pattern aus [5] (und damit indirekt auch [4]) auf die in [4] betrachteten regulären Ausdrücke übertragen.

Aus Sicht des Autors zeigt dies, dass die Untersuchung deskriptiver Pattern sowie der deskriptiven Generalisierung nicht nur im Kontext von Patternsprachen von Bedeutung ist, sondern auch darüber hinaus zu überraschenden Anwendungen in anderen Gebieten führen kann.

Im Gegensatz zur einfachen Definition von Patternsprachen sind viele kanonische Fragen zur Patternsprachen überraschend schwer. Dies gilt auch für Fragen zu deskriptiven Pattern. Insbesondere wurde die Frage, ob zu jeder Sprache ein deskriptives Pattern existiert, bereits von Jiang et al. [6] gestellt, aber erst von Freydenberger and Reidenbach [4] negativ beantwortet.

Damit zu einer Sprache L *kein* Pattern deskriptiv ist, muss zwischen jedem Pattern α mit $L_{\mathrm{E}}(\alpha) \supseteq L$ und L selbst eine unendliche absteigende *Kette* von Patternsprachen existieren. Dies ist leicht zu sehen: Falls kein Pattern deskriptiv für L ist, muss zu jedem Pattern α_i mit $L_{\mathrm{E}}(\alpha_i) \supset L$ ein weiteres Pattern α_{i+1} existieren, für das $L_{\mathrm{E}}(\alpha_i) \supset L_{\mathrm{E}}(\alpha_{i+1}) \supset L$ gilt. Dieser Prozess lässt sich unendlich fortsetzen, so dass die erwähnte Kette entsteht.

Eine solche Kette ist daher auch essentieller Bestandteil des Beweis in [4] zur Existenz einer Sprache, für die kein Pattern deskriptiv ist. Dieser Beweis gibt außerdem Grund zu der Vermutung, dass die Struktur jeder solchen Sprache stark mit der Struktur der entsprechenden Kette (oder auch Kette*n*) zusammenhängt. Nach eingehender Beschäftigung mit diesen Zusammenhängen könnte man zu dem Schluss kommen, dass sich auf diese Art eine Charakterisierung erstellen lässt, die die Sprachen beschreibt, für die kein Pattern deskriptiv ist.

Dies ist allerdings ein Trugschluss: Anhand der in diesem Vortrag vorgestellten Resultate lässt sich zeigen, dass eine solche Charakterisierung wahrscheinlich recht technisch sein dürfte, so sie überhaupt gefunden werden kann. Um dies zu

beweisen wird ein neues Spracherzeugungsmodell eingeführt, die sogenannten *Kettensysteme*.

Ein Kettensystem besteht aus einem Startpattern α_0 und einer unendlichen Folge von Homomorphismen $(\phi_i)_{i\in\mathbb{N}}$. Ein Kettensytem $C = (\alpha_0, (\phi_i)_{i\in\mathbb{N}})$ erzeugt durch $\alpha_{i+1} := \phi_i(\alpha_i)$ eine folge von Pattern, und mittels $L(C) := \bigcap_{i\in\mathbb{N}} L_{\mathrm{E}}(\alpha_i)$ eine Sprache. Hauptbeitrag der in diesem Vortrag vorgestellten Arbeit ist ein Sortiment von Resultaten zu Kettensystemen, die das Arbeiten mit diesem Modell erleichtern (insbesondere auch auf die Schwierigkeiten, die im Umgang mit einem Modell entstehen, dass über eine unendliche Folge von Homomorphismen definiert wird).

Anhand dieser Resultate wird der Hauptbeweis aus [4] vereinfacht, und es können mehrere schwere Gegenbeispiele definiert werden, die die Vermutung einer einfachen Charakterisierung zwar nicht endgültig widerlegen, jedoch sehr unplausibel erscheinen lassen. Obwohl die in diesen Beispielen verwendeten Sprachen auf den ersten Blick sehr kompliziert wirken mögen, wird gezeigt, dass sie allesamt EDT0L-Sprachen sind.

Die meisten der im Vortrag vorgestellten Resultate erschienen zuvor in [2], der Dissertation des Autors (dieser ist übrigens gerne bereit, Interessenten auf Nachfrage ein persönliches gedrucktes Exemplar zu überlassen).

Literaturverzeichnis

[1] D. Angluin. Finding patterns common to a set of strings. *J. Comput. Syst. Sci.*, 21:46–62, 1980.

[2] D. D. Freydenberger. *Inclusion of pattern languages and related problems.* PhD thesis, Institut für Informatik, Goethe-Universität Frankfurt am Main, 2011. Logos Verlag, Berlin.

[3] D. D. Freydenberger and T. Kötzing. Fast learning of restricted regular expressions and DTDs. In *Proc. ICDT 2013*, pages 45–56, 2013.

[4] D. D. Freydenberger and D. Reidenbach. Existence and nonexistence of descriptive patterns. *Theor. Comput. Sci.*, 411(34-36):3274–3286, 2010.

[5] D. D. Freydenberger and D. Reidenbach. Inferring descriptive generalisations of formal languages. *J. Comput. Syst. Sci.*, 79(5):622–639, 2013.

[6] T. Jiang, E. Kinber, A. Salomaa, K. Salomaa, and S. Yu. Pattern languages with and without erasing. *Int. J. Comput. Math.*, 50:147–163, 1994.

[7] Y. K. Ng and T. Shinohara. Developments from enquiries into the learnability of the pattern languages from positive data. *Theor. Comput. Sci.*, 397:150–165, 2008.

[8] K. Salomaa. Patterns. In C. Martin-Vide, V. Mitrana, and G. Păun, editors, *Formal Languages and Applications*, number 148 in Studies in Fuzziness and Soft Computing, pages 367–379. Springer, 2004.

Characterizing Tree Valuation Languages by Multioperator Weighted Tree Languages

Markus Teichmann** and Johannes Osterholzer*

Faculty of Computer Science, Technische Universität Dresden

Abstract. Weighted tree languages can be a) recognized by weighted tree automata over tree valuation monoids, b) defined by monadic second order logic over tree valuation monoids, c) recognized by weighted tree automata over multioperator monoids, and d) defined by multioperator expressions. We show that the classes a) and b) are characterized by c) and d), respectively, using a special multioperator monoid.

1 Introduction

The fundamental result of Büchi, Elgot, and Trakhtenbrot states the equivalence of *recognizability* by classical finite-state string automata and *definability* by means of formulas of monadic second-order logic (mso). This equivalence result has been extended, in particular, into two directions: (1) from string automata to finite-state tree automata and (2) from the unweighted to the weighted case. In this paper, we focus on weighted tree languages over *tree valuation monoids* (tv-monoid) and over *multioperator monoids* (m-monoid).

A tv-monoid is a commutative monoid $(D, +, 0)$ equipped with a valuation function Val which maps each unranked tree over elements of the carrier set D to an element of D. The weight of a run of a tv-monoid weighted tree automaton (tv-wta) is determined by applying the valuation function to a tree generated from the input tree by replacing every node by the weight of the transition taken at this position. In [1] the Büchi-like characterization has been proved for tv-wta using a syntactically restricted tree valuation weighted mso (tv-mso), i.e. $\mathrm{Rec}(\Sigma, \mathbb{D}) = \mathrm{Def}(\Sigma, \mathbb{D})$ where $\mathbb{D}$ is a product tv-monoid 'with multiplication'.

An m-monoid is a commutative monoid $(A, +, 0)$ equipped with a set Ω of (arbitrary) operations on A. To each transition of an m-monoid weighted tree automaton (m-wta) an operation is assigned which has the same arity as the transition. The weight of a run is obtained by evaluating the operations inductively according to their occurrences in that run. In [2] the Büchi-like characterization has been proved for m-wta using multioperator expressions (m-expressions), i.e. automata and logics are equally powerful: $\mathrm{Rec}(\Sigma, \mathcal{A}) = \mathrm{Def}(\Sigma, \mathcal{A})$.

In this paper, we construct (cf. Constr. 1) for any tv-monoid $\mathcal{D}$ (regular product tv-monoid $\mathbb{D}$) an m-monoid $\mathcal{A}_\mathcal{D}$ ($\mathcal{A}_\mathbb{D}$) s.t. the following holds (cf. Fig. 1):

1. $\mathrm{Rec}(\Sigma, \mathcal{D}) = \pi_1^2(\mathrm{Rec}(\Sigma, \mathcal{A}_\mathcal{D}))$ (cf. Thm. 3) and
2. $\mathrm{Def}(\Sigma, \mathbb{D}) = \pi_1^2(\mathrm{Def}(\Sigma, \mathcal{A}_\mathbb{D}))$ (cf. Thm. 5).

* (Partially) **Supported by DFG Graduiertenkolleg 1763 (QuantLA)

$$
\begin{tikzcd}
\text{tv-wta} \arrow[r, "{[\![\cdot]\!]}"] \arrow[d, leftrightarrow, "\text{related}"] & \mathrm{Rec}(\Sigma,\mathbb{D}) \arrow[r, equal, "{[1,\ \text{Thm. 5.5}]}"] \arrow[d, equal, "\text{Thm. 3}"'] & \mathrm{Def}(\Sigma,\mathbb{D}) \arrow[d, equal, "\text{Thm. 5}"] & \text{tv-mso formulas} \arrow[l, "{[\![\cdot]\!]}"'] \arrow[d, shift left, "t(\cdot)"] \\
\text{m-wta} \arrow[r, "{[\![\cdot]\!]}"'] & \pi_1^2(\mathrm{Rec}(\Sigma,\mathcal{A}_{\mathbb{D}})) \arrow[r, equal, "{[2,\ \text{Thm. 4.1}]}"'] & \pi_1^2(\mathrm{Def}(\Sigma,\mathcal{A}_{\mathbb{D}})) & \text{m-expressions} \arrow[l, "{[\![\cdot]\!]}"] \arrow[u, shift left, "t'(\cdot)"]
\end{tikzcd}
$$

Fig. 1. Overview of proven results (based on [2, p. 275])

2 Main Results

Construction 1. Given a tv-monoid $\mathcal{D} = (D, +, 0, \mathrm{Val})$ we construct the *corresponding m-monoid* $\mathcal{A}_{\mathcal{D}} = (D \times T_D^u, \oplus, (0,0), \Omega)$ as follows. Let $(T_D^u, \tilde{+}, 0)$ be the monoid over unranked trees labeled by elements of D where the operation $\tilde{+}$ is the additive superposition of two unranked trees from T_D^u using $+$. Then, $\oplus$ is the direct product of $+$ and $\tilde{+}$. We set $\Omega = \{\mathrm{valtop}_d^{(k)} \mid d \in D, k \in \mathbb{N}\}$ and, for $d \in D$, $k \in \mathbb{N}$, and $(d_1, \xi_1), \ldots, (d_k, \xi_k) \in D \times T_D^u$ with $\xi = d(\xi_1, \ldots, \xi_k)$, we define

$$
\mathrm{valtop}_d^{(k)}((d_1,\xi_1),\ldots,(d_k,\xi_k)) = \begin{cases} \big(\mathrm{Val}(\xi), \xi\big) & \text{if } d \neq 0 \text{ and} \\ & \xi_1, \ldots, \xi_k \text{ are 0-free,} \\ (0,0) & \text{otherwise.} \end{cases}
$$

Definition 2. *Let $\mathcal{N} = (Q, \mu, F)$ be a tv-wta over Σ and $\mathcal{D}$, and $\mathcal{M} = (Q', \delta, F')$ be an m-wta over Σ and $\mathcal{A}_{\mathcal{D}}$. We call $\mathcal{N}$ and $\mathcal{M}$* related *if*

- *$Q = Q'$ and $F = F'$;*
- *for every $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, and $q, q_1, \ldots, q_k \in Q$ it holds that $\delta_\sigma(q_1 \ldots q_k, q) = \mathrm{valtop}_d$ iff $\mu_\sigma(q_1 \ldots q_k, q) = d$.*

Related automata recognize the same weighted tree language (disregarding the projection) and since the definition can be seen as a construction we conclude:

Theorem 3. *Let $\mathcal{D}$ be a tv-monoid, $\mathcal{A}_{\mathcal{D}}$ the corresponding m-monoid, and Σ be a finite ranked alphabet. The following holds:*

1. *For every tv-wta $\mathcal{N}$ over Σ and $\mathcal{D}$ there is an m-wta $\mathcal{M}$ over Σ and $\mathcal{A}_{\mathcal{D}}$ such that $[\![\mathcal{N}]\!] = \pi_1^2([\![\mathcal{M}]\!])$.*
2. *For every m-wta $\mathcal{M}$ over Σ and $\mathcal{A}_{\mathcal{D}}$ there is a tv-wta $\mathcal{N}$ over Σ and $\mathcal{D}$ such that $\pi_1^2([\![\mathcal{M}]\!]) = [\![\mathcal{N}]\!]$.*

On the level of the logics we provide transformation functions $t(\cdot)$ and $t'(\cdot)$ between $\mathrm{Def}(\Sigma, \mathbb{D})$ and $\mathrm{Def}(\Sigma, \mathcal{A}_{\mathbb{D}})$ and vice versa, respectively. We only sketch the transformation $t(\cdot)$:

Construction 4. Let φ be a syntactically restricted[1] tv-mso formula over Σ and a regular product tv-monoid $\mathbb{D}$. We construct the *corresponding m-expression $t(\varphi)$ over Σ and $\mathcal{A}_{\mathbb{D}}$* inductively on the structure of φ:

[1] formally, φ is $\forall$-restricted and strongly $\wedge$-restricted

- For every Boolean tv-mso formula β: $t(\beta) = \beta \triangleright t(1)$.
- For every $d \in D$: Since $\mathbb{D}$ is regular, there is a tv-wta $\mathcal{N}_d = (Q, \mu, F)$ which assigns d to every input tree. The formula $t(d)$ simulates the semantics of this automaton.
- $t(\varphi_1 \vee \varphi_2) = t(\varphi_1) + t(\varphi_2)$.
- $\varphi = \varphi_1 \wedge \varphi_2$ is strongly $\wedge$-restricted:
  - φ_1 or φ_2 Boolean: Assume φ_1 is Boolean then $t(\varphi_1 \wedge \varphi_2) = \varphi_1 \triangleright t(\varphi_2)$
  - φ_1 and φ_2 are almost Boolean, i.e. $\mathrm{step}(\varphi_i) = (a_1^i, \psi_1^i) \dots (a_{n_i}^i, \psi_{n_i}^i)$ then we set $t(\varphi_1 \wedge \varphi_2) = \sum^+_{\substack{i \in [n_1] \\ j \in [n_2]}} (\psi_i^1 \wedge \psi_j^2) \triangleright t\big(a_i^1 \diamond a_j^2\big)$.
- $t(\exists x.\psi) = \sum_x t(\psi)$
- $t(\exists X.\psi) = \sum_X t(\psi)$
- $\varphi = \forall x.\psi$ is $\forall$-restricted: ψ almost Boolean, i.e. $\mathrm{step}(\psi) = (d_1, \psi_1) \dots (d_n, \psi_n)$. Let $\mathcal{U} = \{X_1, \dots, X_n\}$ and ω^ψ be a $\Sigma_\mathcal{U}$-family such that for every $U \subseteq \mathcal{U}$, $k \in \mathbb{N}$, and $\sigma \in \Sigma^{(k)}$ we have $(\omega^\psi)_{(\sigma,U)} = \mathrm{valtop}^{(k)}_{d_U}$ where $d_U = \sum_{\substack{i \in [n] \\ X_i \in U}} d_i$.

  We define, similarly to the proof in [2, Lemma 5.10],

$$t(\forall x.\psi) = \sum_{X_1} \dots \sum_{X_n} \big(\forall x.(\bigwedge_{i \in [n]} (x \in X_i) \Leftrightarrow \psi_i)\big) \triangleright H(\omega^\psi) \ .$$

The transformation $t'(\cdot)$ in the opposite direction uses similar considerations for common syntactic constructs like disjunction. The semantics of the structure $H(\omega)$ is simulated using a more complex formula enumerating symbol-variable combinations. Using both transformation functions we can conclude:

Theorem 5. *Let $\mathbb{D}$ be a regular product tv-monoid, $\mathcal{A}_\mathbb{D}$ the corresponding m-monoid, and Σ a finite alphabet. Then the following holds:*

1. *For every $\forall$-restricted and strongly $\wedge$-restricted tv-mso formula φ there is an m-expression $t(\varphi)$ over Σ and $\mathcal{A}_\mathbb{D}$ such that $[\![\varphi]\!] = \pi_1^2([\![t(\varphi)]\!])$.*
2. *For every m-expression e over Σ and $\mathcal{A}_\mathbb{D}$ there is a tv-mso formula $t'(e)$ over Σ and $\mathbb{D}$ such that $\pi_1^2([\![e]\!]) = [\![t'(e)]\!]$.*

An alternative proof for the Büchi-like theorem [1, Thm. 5.5] arises utilizing our result and the Büchi-characterization [2, Thm. 4.1] (cf. Fig. 1).

References

1. M. Droste, D. Götze, S. Märcker, and I. Meinecke. Weighted tree automata over valuation monoids and their characterization by weighted logics. In *Algebraic Foundations in Computer Science*, volume 7020 of *Lecture Notes in Comput. Sci.*, pages 30–55. Springer Berlin Heidelberg, 2011.
2. Z. Fülöp, T. Stüber, and H. Vogler. A Büchi-like theorem for weighted tree automata over multioperator monoids. *Theory Comput. Syst.*, 50(2):241–278, 2012. Published online 28.10.2010.

A Pushdown Machine for Context-Free Tree Translation

Johannes Osterholzer*

Institute of Theoretical Computer Science
Technische Universität Dresden

Abstract. We identify a syntactic restriction of *synchronous context-free tree grammars.* The notion of (linear and non-deleting) *pushdown extended top-down tree transducers* is introduced and we prove that the transformations of the former coincide with those of the latter.

1 Introduction

In syntax-based machine translation of natural languages, an input sentence s is translated by applying a tree transformation to a parse tree ξ of s, given a grammar for the input language. The transformation is often performed by formalisms such as extended top-down tree transducers (XTT) [7, 5]. These, however, can not capture certain phenomena that occur in natural language [4].

Hence a number of more powerful formalisms has been introduced, among those *synchronous context-free tree grammars (SCFTG)* [6]. They can be considered as context-free tree grammars where both an input and output tree are derived synchronously (cf. Fig. 1). Synchronous derivation leaves open the problem of *parsing* a given input tree. But it may prove beneficial, just as for string languages, to research formalisms which take parsing into account. Hence, we propose a formalism with a unidirectional derivation semantics, called *pushdown extended top-down tree transducers (PDXTT).* These can be understood as extended top-down tree transducers where the finite state control is equipped with a tree pushdown storage [7, 2] that allows the recognition of an input context-free tree language, or as (extended input) pushdown tree automata [3] with output.

The class of transformations that are computed by linear and non-deleting PDXTT will be proven to coincide with the class of transformations of *simple* SCFTG, which are a certain syntactic restriction of SCFTG. Actually, this is a generalization of the characterization of simple syntax-directed translation schemata by pushdown transducers [1] to tree transformations.

2 Synchronous Context-Free Tree Grammars

Context-free tree grammars (CFTG) are a generalization of CFG to trees. Roughly, a CFTG $\mathcal{G}$ consists of two ranked alphabets Σ and N of *terminal* resp. *nonterminal symbols.* The productions of $\mathcal{G}$ allow to rewrite a nonterminal A of rank k *within* a tree into a tree over $N \cup \Sigma$, i.e., in $T_{N\cup\Sigma}(X_k)$, cf. [7].

* Partially supported by DFG Graduiertenkolleg 1763 (QuantLA)

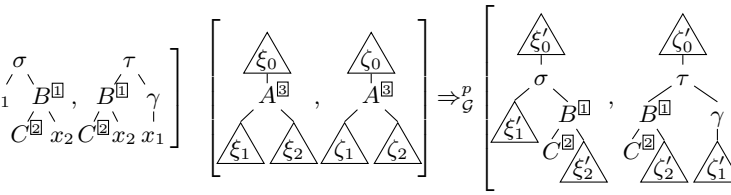


Fig. 1. An SCFTG production p and its application to a sentential form

The right-hand sides of the productions of *SCFTG* now are *pairs* of such trees, such that there is a bijection between the contained nonterminal symbols, and both trees are linear and non-deleting in a common set of variables. The sentential forms of SCFTG are also pairs of trees over $N \cup \Sigma$, where the nonterminals are correlated as above. A production may only be applied to such a correlated pair of nonterminals. For an example, see Fig. 1, where the bijective relation between the nonterminal symbols A resp. B is denoted by a boxed superscript number (an *index*), and ξ_0' etc. are just the result of a consistent renaming of these indices in ξ_0 etc. to avoid clashes. This SCFTG transforms a tree ξ into ζ if both ξ and ζ contain only terminals and the pair $[\xi, \zeta]$ is derived from an initial pair of nonterminals in a finite number of steps.

Intuitively, a production is *simple* if both the input and the output tree from its right-hand side exhibit the same call structure of nonterminal symbols and variables: for every occurring nonterminal $A^{\boxed{i}}$ of rank k, and for every $j \in \{1, \ldots, k\}$, the sets of indexed nonterminals and variables contained in the j-th child subtree of $A^{\boxed{i}}$ must be equal in the right-hand side's input and output component. Hence, the right-hand side of the production p in Fig. 1 is simple. In contrast, the right-hand side $\left[D^{\boxed{1}}(D^{\boxed{2}}(x_1)), D^{\boxed{2}}(D^{\boxed{1}}(x_1))\right]$ is not simple, since $D^{\boxed{1}}$ dominates $D^{\boxed{2}}$ in the input, but not in the output tree. The right-hand side $\left[A^{\boxed{1}}(x_1, x_2), A^{\boxed{1}}(x_2, x_1)\right]$ is not simple either, since x_1 appears as the nonterminal's first argument in the input, but as the second one in the output. An SCFTG is called simple if all its productions are simple.

3 Pushdown Extended Top-Down Tree Transducers

In contrast to the productions of SCFTG, the rules of PDXTT are asymmetric, and permit a state-based rewriting of input into output trees. Just as for XTT, every rule allows to match the input tree with a context of finite but arbitrary height. Their right-hand sides are trees, at whose frontiers the state-based rewriting may continue on the remaining subtrees of the input. Unlike XTT however, the derivation process of PDXTT is controlled by a tree pushdown. Thus, a rule can additionally check the top symbol of the tree pushdown for the current input tree, and push further information that controls the derivation of the remaining subtrees. A PDXTT $\mathcal{M}$ transforms ξ into ζ if ζ consists entirely of output symbols and if it is a normal form of $\langle q_0, \gamma_0, \xi \rangle$ with regard to $\Rightarrow_{\mathcal{M}}$, where q_0 and γ_0 are the initial state, resp. initial pushdown symbol, of $\mathcal{M}$.

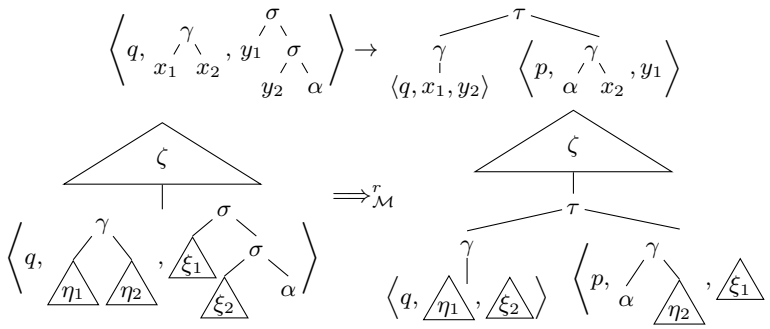


Fig. 2. A PDXTT rule r and its application

For an example rule r and its application to a configuration, see Fig. 2. The tree ζ has already been produced as output, while the input (sub)tree $\sigma(\xi_1, \sigma(\xi_2, \alpha))$ has yet to be rewritten by state q. Since the tree pushdown is of the form $\gamma(\eta_1, \eta_2)$, r can be applied, producing some output, with the remaining inputs ξ_1 and ξ_2 marked for processing, controlled by the pushdowns η_1 and η_2, where moreover $\gamma(\alpha, x_1)$ has been pushed onto η_2.

If, for every rule of a PDXTT $\mathcal{M}$, each variable x_i and y_j on its left-hand side appears exactly once on its right-hand side, then $\mathcal{M}$ is *linear and non-deleting.* Rule r in Fig. 2 is of this form.

Theorem 1. *Let τ be a tree transformation. The following are equivalent:*

1. *There is a simple SCFTG s.t. τ is its transformation.*
2. *There is a linear and nondeleting PDXTT s.t. τ is its transformation.*

Proof. By a close correspondence between simple SCFTG in (a certain) normal form and linear and nondeleting one-state PDXTT in (another) normal form.

References

1. A. V. Aho and J. D. Ullman. *The Theory of Parsing, Translation, and Compiling.* Prentice-Hall, 1972.
2. J. Engelfriet and H. Vogler. Pushdown Machines for the Macro Tree Transducer. *Theoret. Comput. Sci.*, 42(3):251–368, 1986.
3. I. Guessarian. Pushdown Tree Automata. *Math. Syst. Theory*, 16(1):237–263, 1983.
4. L. Kallmeyer. *Parsing Beyond Context-Free Grammars.* Springer, 2010.
5. A. Maletti, J. Graehl, M. Hopkins, and K. Knight. The Power of Extended Top-Down Tree Transducers. *SIAM J. Comput.*, 39(2):410–430, 2009.
6. M.-J. Nederhof and H. Vogler. Synchronous Context-Free Tree Grammars. In *Proc. of TAG+11*, pages 55–63, 2012.
7. W. C. Rounds. Mappings and Grammars on Trees. *Theory Comput. Syst.*, 4(3):257–287, 1970.

Construction of a Bottom-Up Deterministic n-Gram Weighted Tree Automaton

Matthias Büchse, Tobias Denkinger, and Heiko Vogler

Department of Computer Science
Technische Universität Dresden

Abstract. We propose a novel construction to lift an n-gram model to trees. The resulting weighted tree automaton is bottom-up deterministic in contrast to the weighted tree automaton constructed using the Bar-Hillel, Perles, Shamir algorithm.

1 Introduction

Recent approaches to machine translation are mostly statistical [4]. Researchers define a class of translation functions, and they use training algorithms to select a function that fits a given set of existing translations. Translation functions that are considered in research are often syntax-directed, i.e., the grammatical structure of a sentence, represented by a tree, is of special interest.

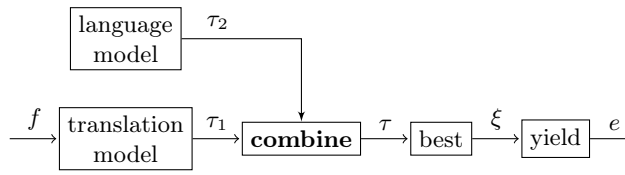


Fig. 1. Translation function with translation and language model.

A typical translation function is shown in Fig. 1. The translation model consumes the input sentence f and emits a weighted tree language (WTL) τ_1 over $(\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$, in which each translation of f is assigned a real number (as weight). The language model provides a weight for every sentence of the target language by means of a WTL τ_2. Both WTLs are then combined into one WTL τ. Best followed by yield outputs the string e of the best tree ξ in τ.

Extended top-down tree transducers [3], synchronous tree-adjoining grammars [6], and synchronous context-free grammars [2] are some of the most prominent examples of translation models. Examples of language models are n-gram models, hidden Markov models, weighted string automata (WSA), and probabilistic context-free grammars.

All language models mentioned above generate weighted string languages (WSL). But in order to make the combination of τ_1 and τ_2 possible, τ_2 must be lifted to a WTL. In this paper we show a construction that lifts the n-gram model

to a WTL by constructing a weighted tree automaton (WTA), called n-gram WTA.

The classical approach to construct the n-gram WTA is the following: for the n-gram model N, we can construct a WSA $\mathcal{A}$ that recognizes N. Then we construct the product of $\mathcal{A}$ and the WTA that recognizes every tree with weight 1. For this, we employ the extension [5, Section 4] of the Bar-Hillel, Perles, Shamir algorithm [1, Lemma 4.1]. The constructed product is the n-gram WTA.

We propose a direct construction for the n-gram WTA. We show that the resulting WTA is bottom-up deterministic, which is in contrast to the n-gram WTA produced by the classical approach. Our construction is inspired by [2] where it appears interleaved with the other steps shown in Fig. 1.

An efficient implementation of the translation function in Fig. 1 computes the two functions best and combine interleaved, where best is usually computed via dynamic programming, i.e., bottom-up. Thus such an algorithm can profit when τ_2 is specified in a bottom-up deterministic manner.

2 Preliminaries

We let Γ^* denote the set of all words over an alphabet Γ. For $w \in \Gamma^*$ and $k \in \mathbb{N}$, $\mathrm{fst}_k(w)$ and $\mathrm{lst}_k(w)$ denote the sequences of the first and the last k symbols of w, respectively. A *ranked alphabet* is a tuple (Σ, rk) where Σ is an alphabet and $\mathrm{rk}\colon \Sigma \to \mathbb{N}$ is a *rank mapping*. In the following, we assume that Γ is an alphabet and (Σ, rk), or just Σ, is a ranked alphabet with $\Gamma \subseteq \mathrm{rk}^{-1}(0)$.

Let Q be an alphabet, the set of *unranked trees over* Q is denoted by $\mathcal{U}_Q$. The set of *positions of* ξ is denoted by $\mathrm{pos}(\xi)$. For $p \in \mathrm{pos}(\xi)$, the *symbol of* ξ *at* p is denoted by $\xi(p)$. The set of *(ranked) trees over* Σ is denoted by T_Σ. The Γ*-yield of* ξ is the mapping $\mathrm{yield}_\Gamma : T_\Sigma \to \Gamma^*$ where $\mathrm{yield}_\Gamma(\xi)$ is the sequence of all symbols σ in ξ with $\sigma \in \Gamma$ read from left to right.

A *weighted tree automaton (WTA)* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \nu)$ where Q is an alphabet, δ is a Σ-family of functions $\delta_\sigma\colon Q^{\mathrm{rk}(\sigma)} \times Q \to \mathbb{R}_{\geq 0}$, and $\nu\colon Q \to \mathbb{R}_{\geq 0}$. The set of all *runs of* $\mathcal{A}$ *on* ξ is the set $R_\mathcal{A}(\xi) = \{\kappa \in \mathcal{U}_Q \mid \mathrm{pos}(\kappa) = \mathrm{pos}(\xi)\}$. For $\kappa \in R_\mathcal{A}(\xi)$, the *weight of* κ is $\mathrm{wt}(\kappa) = \prod_{p \in \mathrm{pos}(\xi)} \delta_{\xi(p)}(\kappa(p1), \ldots, \kappa(p\,\mathrm{rk}(\xi(p))), \kappa(p))$. The *semantics of* $\mathcal{A}$ is the mapping $[\![\mathcal{A}]\!]\colon T_\Sigma \to \mathbb{R}_{\geq 0}$ where for every $\xi = \sigma(\xi_1, \ldots, \xi_{\mathrm{rk}(\sigma)}) \in T_\Sigma$ we define $[\![\mathcal{A}]\!](\xi) = \sum_{\kappa \in R_\mathcal{A}(\xi)} \mathrm{wt}(\kappa) \cdot \nu(\kappa(\varepsilon))$. We call $\mathcal{A}$ *bottom-up deterministic* if for every $\sigma \in \Sigma$ and $q_1, \ldots, q_{\mathrm{rk}(\sigma)} \in Q$ there exists at most one $q \in Q$ such that $\delta_\sigma(q_1, \ldots, q_{\mathrm{rk}(\sigma)}, q) > 0$.

In the following we assume that $n \geq 1$. An *n-gram model over* Γ is a tuple $N = (\Gamma, \mu)$ where $\mu\colon \Gamma^n \to \mathbb{R}_{\geq 0}$ is a mapping (n-gram weights). The *semantics of an n-gram model* N is the mapping $[\![N]\!]\colon \Gamma^* \to \mathbb{R}_{\geq 0}$ where for every $l \geq 0$ and $w_1, \ldots, w_l \in \Gamma$ we define $[\![N]\!](w_1 \cdots w_l) = \prod_{i=0}^{l-n} \mu(w_{i+1} \cdots w_{i+n})$ if $l \geq n$, and $[\![N]\!](w_1 \cdots w_l) = 0$ otherwise. In the following, N denotes an n-gram model.

Proposition 1. *Let $u, v \in \Gamma^*$, $|u| \geq n$, and $|v| \geq n$. We have*

$$[\![N]\!](uv) = [\![N]\!](u) \cdot [\![N]\!](\mathrm{lst}_{n-1}(u)\,\mathrm{fst}_{n-1}(v)) \cdot [\![N]\!](v) \ .$$

3 Direct Construction

In order to define a WTA $\mathcal{A}_{N,\Sigma}$ with $[\![\mathcal{A}_{N,\Sigma}]\!] = [\![N]\!] \circ \mathrm{yield}_\Gamma$ we have to compute $[\![N]\!] \circ \mathrm{yield}_\Gamma(\xi)$ while traversing a given tree ξ bottom-up. At each node in ξ, we only see the current symbol and the states of the computations in the subtrees. A closer look at Proposition 1 suggests to (1) compute the semantics of the currently visible substrings under N and (2) propagate the left and right $n-1$ symbols of the substring in the state. In the following construction, parts (1) and (2) are handled by the functions g and f, respectively.

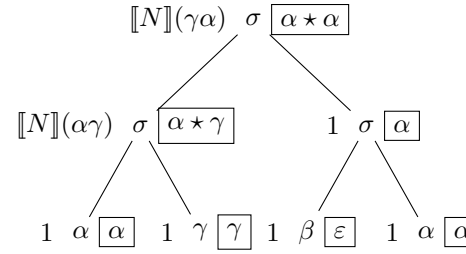


Fig. 2. Tree over $\Sigma = \{\sigma^{(2)}, \beta^{(0)}\} \cup \Gamma$ and $\Gamma = \{\alpha^{(0)}, \gamma^{(0)}\}$, with a run (states appear in boxes) and transition weights due to a 2-gram model N over Γ.

Let $\star$ be a new symbol, i.e., $\star \notin \Sigma$. We define $f\colon (\Gamma \cup \{\star\})^* \to (\Gamma \cup \{\star\})^*$ and $g\colon (\Gamma \cup \{\star\})^* \to \mathbb{R}_{\geq 0}$ as follows. Let $w \in (\Gamma \cup \{\star\})^*$. Then $f(w) = \mathrm{fst}_{n-1}(w) \star \mathrm{lst}_{n-1}(w)$ if $|w| \geq n$, and $f(w) = w$ otherwise. Note that there are $u_0, \ldots, u_k \in \Gamma^*$ such that $w = u_0 \star u_1 \cdots \star u_k$. We define $g(w) = \prod_{i=0}^{k} N'(u_i)$ where $N'(u_i) = [\![N]\!](u_i)$ if $|u_i| \geq n$, and $N'(u_i) = 1$ otherwise.

The *n-gram WTA over* Σ is the WTA $A_{N,\Sigma} = (Q, \Sigma, \delta, \nu)$ where $Q = Q_1 \cup Q_2$ with $Q_1 = \bigcup_{i=0}^{n-1} \Gamma^i$ and $Q_2 = \Gamma^{n-1} \times \{\star\} \times \Gamma^{n-1}$, $\nu(q) = 1$ if $q \in Q_2$, otherwise $\nu(q) = 0$, and for every $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, and $q_1, \ldots, q_k, q \in Q$ (cf. Fig. 2 for an example):

$$\delta_\sigma(q_1, \ldots, q_k, q) = \begin{cases} g(q) & \text{if } k = 0 \text{ and } q = \mathrm{yield}_\Gamma(\sigma) \\ g(q_1 \cdots q_k) & \text{if } k \geq 1 \text{ and } q = f(q_1 \cdots q_k) \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 2. *Let N be an n-gram model over Γ and Σ be a ranked alphabet. Then $[\![\mathcal{A}_{N,\Sigma}]\!] = [\![N]\!] \circ \mathrm{yield}_\Gamma$ and the WTA $\mathcal{A}_{N,\Sigma}$ is bottom-up deterministic.*

References

1. Y. Bar–Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. *Z. Phonetik. Sprach. Komm.*, 14:143–172, 1961.
2. D. Chiang. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228, 2007.
3. K. Knight and J. Graehl. An overview of probabilistic tree transducers for natural language processing. In *Computational Linguistics and Intelligent Text Processing*, volume 3406 of *LNCS*, pages 1–24. Springer, 2005.
4. A. Lopez. Statistical machine translation. *ACM Comp. Surv.*, pages 8:1–8:49, 2008.
5. A. Maletti and G. Satta. Parsing algorithms based on tree automata. In *Proc. of IWPT '09*, pages 1–12. ACL, 2009.
6. S. M. Shieber and Y. Schabes. Synchronous tree-adjoining grammars. In *Proceedings of the 13th Int. Conf. on Comp. Ling.*, volume 3, pages 253–258, 1990.

State-Splitting for Regular Tree Grammars

Toni Dietze

Institute of Theoretical Computer Science, Technische Universität Dresden

Abstract. We formalize the state-splitting algorithm of [2].

1 Introduction

The expectation-maximization algorithm (EM algorithm) [1] is a well-known procedure, which can be used, e.g., to estimate probabilities for the rules of a regular tree grammar (RTG) G on the basis of a corpus c of derived trees. The algorithm starts with an arbitrary probability assignment p_0 and iterates an expectation and a maximization step; in each iteration the likelihood of the corpus c under the current probabilistic RTG (PRTG) increases or stays the same. We denote the resulting PRTG by $\mathrm{EM}((G, p_0), c)$.

In [2], multiple calls to EM were interleaved with modifications of the underlying RTG (cf. SPLIT, MERGE in Fig. 1). This may improve the likelihood, because the state behavior of the current RTG may be adapted to characteristics of the trees in c. In [2], this approach was presented in an informal style, and no properties were proved. In our current work, we have formalized this approach and call it state-splitting algorithm.

2 Preliminaries

Let Σ be an alphabet. We denote the set of all trees over Σ by T_Σ. A *regular tree grammar (RTG)* is a tuple $(Q, \Sigma, q_\mathrm{s}, R)$ where Q and Σ are alphabets of *states* and *terminal symbols*, respectively, with $Q \cap \Sigma = \emptyset$, $q_\mathrm{s} \in Q$ is the *initial state*, and R is a finite set of *rules* of the form $q_0 \to \sigma(q_1, \ldots, q_n)$ where $\sigma \in \Sigma$, $n \geq 0$, and $q_i \in Q$. We denote the left-hand side of a rule $r \in R$ by $\mathrm{lhs}(r)$. A *probabilistic RTG (PRTG)* is a pair (G, p) where $G = (Q, \Sigma, q_\mathrm{s}, R)$ is an RTG and $p\colon R \to [0, 1]$ is a probability assignment which is *proper*, i.e., the probabilities of rules with the same left-hand side sum up to 1. A *corpus* is a mapping $c\colon \mathrm{T}_\Sigma \to \mathbb{R}_{\geq 0}$ such that $\{a \in A \mid c(A) > 0\}$ is finite. The *likelihood of c under* (G, p) is defined as $\mathrm{L}(c \mid (G, p)) = \prod_{t \in \mathrm{T}_\Sigma} p(t)^{c(t)}$.

3 The State-Splitting Algorithm

The state-splitting algorithm (see Fig. 1) uses the functions SPLIT and MERGE.

Input: alphabet Σ, corpus c over T_Σ, PRTG G_0 over Σ with initial state q_s and $\mathrm{L}(c \mid G_0) > 0$, $\mu \in [0,1]$.
Output: sequence of PRTG.

for $i \leftarrow 1, 2, \ldots$ **do**
  $G' \leftarrow \mathrm{EM}(\textsc{split}(G_{i-1}), c)$
  $G_i \leftarrow \mathrm{EM}(\textsc{merge}(G'), c)$
function $\textsc{split}(G)$
  $\pi \leftarrow G$-splitter splitting every state q in G into q^1 and q^2 except q_s
  return a proper π-split of G
function $\textsc{merge}(G')$
  $\pi \leftarrow$ identity mapping
  for all states q s.t. q^1, q^2 in G' **do**
    $\widehat{\pi} \leftarrow$ identity mapping
    $\widehat{\pi}(q^1) \leftarrow q$ and $\widehat{\pi}(q^2) \leftarrow q$
    $\lambda \leftarrow$ a good $\widehat{\pi}$-distributor
    if $\frac{\mathrm{L}(c \mid \mathrm{merge}_{\widehat{\pi}}^{\lambda}(G'))}{\mathrm{L}(c \mid G')} \geq \mu$ **then**
      $\pi(q^1) \leftarrow q$ and $\pi(q^2) \leftarrow q$
  $\lambda \leftarrow$ a good π-distributor
  return $\mathrm{merge}_\pi^\lambda(G')$

Fig. 1. The state-splitting algorithm.

Splitting Let $G = (Q, \Sigma, q_\mathrm{s}, R)$ be an RTG and let $q \in Q \setminus \{q_\mathrm{s}\}$. We can split q into new states, e.g., q^1 and q^2. Then we also have to split rules which use this state, e.g., we would split the rule $q_\mathrm{s} \to \sigma(q, q)$ into $q_\mathrm{s} \to \sigma(q^1, q^1)$, $q_\mathrm{s} \to \sigma(q^1, q^2)$, $q_\mathrm{s} \to \sigma(q^2, q^1)$, and $q_\mathrm{s} \to \sigma(q^2, q^2)$ (cf. Fig. 2), i.e., we create a rule for every possible combination of the split states.

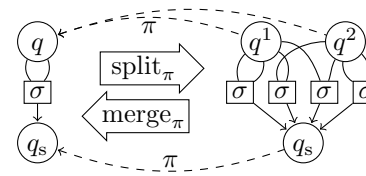


Fig. 2. Visualization of split/merge.

Formally, let Q' be an alphabet such that $q_\mathrm{s} \in Q'$, and let $\pi \colon Q' \to Q$ be a surjective mapping such that $\pi^{-1}(q_\mathrm{s}) = \{q_\mathrm{s}\}$. We call π a *G-splitter (w.r.t. Q')*. We use π to map a split state to its (unsplit) source. Hence, we can use π^{-1} to split a state. We will denote the splitting based on π by split_π and we will use this notion for the splitting of states, rules, sets of those, and RTG. Thus the *π-split of G* is the RTG $\mathrm{split}_\pi(G) = (\mathrm{split}_\pi(Q), \Sigma, q_\mathrm{s}, \mathrm{split}_\pi(R))$.

Now let us turn to PRTG. Let (G, p) and (G', p') be PRTG over the same terminal alphabet. We say (G', p') is a *proper π-split of* (G, p), if $G' = \mathrm{split}_\pi(G)$, and $p(r) = \sum_{r' \in \mathrm{split}_\pi(r) \colon \mathrm{lhs}(r') = q'} p'(r')$ for every rule r in G and for every $q' \in \mathrm{split}_\pi(\mathrm{lhs}(r))$. One can show that the probability of a tree in (G, p) is the same as in (G', p'), if (G', p') is a proper π-split of (G, p).

Merging The merge operation undoes splits. Formally, let $G' = (Q', \Sigma, q_\mathrm{s}, R')$ be an RTG, Q an alphabet such that $q_\mathrm{s} \in Q$, and π a surjective mapping such that $\pi^{-1}(q_\mathrm{s}) = \{q_\mathrm{s}\}$. We call π a *G'-merger (w.r.t. Q)*. We will denote the merging based on π by merge_π and we will use this notion for the merging of states, rules, sets of those, and RTG. Thus the merging results in the RTG $\mathrm{merge}_\pi(G') = (\mathrm{merge}_\pi(Q'), \Sigma, q_\mathrm{s}, \mathrm{merge}_\pi(R'))$.

Note that splitting and merging fit together nicely: Let G be an RTG, π be a G-splitter, and $G' = \mathrm{split}_\pi(G)$; then π is also a G'-merger and $\mathrm{merge}_\pi(G') = G$. The other direction is a bit more subtle: Let $G' = (Q', \Sigma, q_\mathrm{s}, R')$ be an RTG and π a G'-merger; then $Q' = \mathrm{split}_\pi(\mathrm{merge}_\pi(Q'))$, but $R' \subseteq \mathrm{split}_\pi(\mathrm{merge}_\pi(R'))$.

Now let us turn to PRTG again. Let (G', p') be a PRTG and π a G'-merger with $G' = (Q', \Sigma, q_s, R')$ and $\mathrm{merge}_\pi(G') = (Q, \Sigma, q_s, R)$. Roughly speaking, we need to merge p' now. Let $\lambda\colon Q' \to [0,1]$ be a mapping such that for every $q \in Q$ we have $\sum_{q' \in \mathrm{split}_\pi(q)} \lambda(q') = 1$. We call λ a *π-distributor*. We define the probability assignment $\mathrm{merge}_\pi^\lambda(p')$ to be the mapping $p\colon R \to [0,1]$ such that $p(r) = \sum_{r' \in R'\colon \mathrm{merge}_\pi(r')=r} \lambda(\mathrm{lhs}(r'))\cdot p'(r')$ for every $r \in R$. One can show that p is proper, which is guaranteed by the properties of λ. Yet, the λ provides enough flexibility for later tasks. We define $\mathrm{merge}_\pi^\lambda(G', p') = (\mathrm{merge}_\pi(G'), \mathrm{merge}_\pi^\lambda(p'))$.

In the MERGE function in Fig. 1, the algorithm decides which states to merge. For this purpose, it compares the likelihoods of the unmerged grammar and a grammar where only two states are merged. If this merge does not harm the likelihood too much, these two states will also be merged in the result. The parameter μ lets us configure the maximally accepted loss in the likelihood to find a trade-off between the loss of likelihood and the complexity of the grammar.

4 Further Research

We have to choose a "good" π-distributor λ for every merge. Our idea is, to do an EM iteration for the merged grammar, and since we do not have an initial probability assignment, we use the probability assignment of the split grammar to compute the counts needed for an EM iteration.

While merging, it is not clear how the likelihood changes in general. In [2], the ratio between the likelihoods (cf. Fig. 1, Line 13) was only approximated. Maybe the exact likelihood for the merged grammar can be calculated efficiently, if the likelihood of the unmerged grammar has already been calculated.

Also, in SPLIT we have to choose a proper π-split of G; it remains to investigate how this choice affects the outcome of the algorithm exactly. Additionally, good values for the parameter μ need to be found, which may be done empirically. Another idea is to get along without merging at all, if we split states only selectively, e.g., only the most frequent states in derivations of the trees in the corpus.

References

1. A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.
2. S. Petrov, L. Barrett, R. Thibaux, and D. Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 433–440, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

Anforderungen an ein formales Esperanto

Jens-D. Doll
jens.doll@studium.uni-hamburg.de

In diesem Vortrag werden pseudoformale Anforderungen vorgestellt, die es Sprachentwicklern erlauben, die Definition von Detail- und Metasprachen mit einem universellen Gerüst zu verbinden, das vollständig und unbeschränkt ist.

1 Umfang

Ein formales Esperanto umfasst neben allen Skript- Konfigurations- und Programmiersprachen jedes technische oder konzeptionelle Artefakt.

2 Äquivalenz

Zu jeder definierbaren Sprache findet sich ein semantisches Äquivalent in FE.

3 Reichweite

FE enthält alle erkennbaren Symbole und jede denkbare Abstraktion.

4 Webseite

Die Arbeit an FE wird auf http://cococo.de/Esperanto dokumentiert.

Formalism for the Usage of Natural Language

Yvonne Meeres

Max Planck Institute for Meteorology

Abstract. Defining a formal language not for natural language itself but for the speaking of natural language, we gain insights into new algorithms and into foreign language teaching.

Motivation

My motivation is twofold:

1. problems mastering a foreign language
2. imitate natural language algorithms for robust communication in technical systems.

Learners of foreign languages have problems to master certain aspects of the new language even if they spend years learning it. The presented formalism for grammar teaching books sheds light on the possible reasons for the learning problem.

Apart from that natural language uses interesting algorithms to make communication robust. Technical systems can use this mechanism to achieve a robust communication.

Methodology

The teaching material for foreign languages will be formalized with a formal language. Complexity measurements are defined to make hypothesizing possible. With the help of these formalisms the grammars can be optimized concerning different complexity measurements.

Future Outcome

A deep understanding of the robustness and the complexity of natural language will be achieved. This research is in progress. The formalisation as well as the complexity measurements are in progress with already some of the desired insights gained.

Outlook

The application of the above formalism is widespread, ranging from cognitive science and natural language research to algorithms for a variety of technical systems.

Array Insertion and Deletion P Systems

Henning Fernau[1], Rudolf Freund[2], Sergiu Ivanov[3]
Marion Oswald[2], Markus L. Schmid[1], and K.G. Subramanian[4]

[1] Fachbereich 4 – Abteilung Informatikwissenschaften, Universität Trier
[2] Technische Universität Wien, Institut für Computersprachen
[3] Laboratoire d'Algorithmique, Complexité et Logique, Université Paris Est
[4] School of Computer Sciences, Universiti Sains Malaysia

Array insertion grammars have already been considered as *contextual array grammars* in [5], whereas the inverse interpretation of a contextual array rule as a deletion rule has newly been introduced in [2] and [3]. The results described in this extended abstract were elaborated in [3] for one-dimensional arrays and in [2] for two-dimensional arrays.

Sequential Grammars

A *(sequential) grammar* G (see [4]) is a construct $(O, O_T, w, P, \Longrightarrow_G)$ where O is a set of *objects*, $O_T \subseteq O$ is a set of *terminal objects*, $w \in O$ is the *axiom (start object)*, P is a finite set of *rules*, and $\Longrightarrow_G \subseteq O \times O$ is the *derivation relation* of G induced by the rules in P.

$L_*(G) = \left\{ v \in O_T \mid w \overset{*}{\Longrightarrow}_G v \right\}$ is the *language generated by* G (in the $*$-mode); $L_t(G) = \left\{ v \in O_T \mid \left(w \overset{*}{\Longrightarrow}_G v \right) \wedge \nexists z \left(v \Longrightarrow_G z \right) \right\}$ is the *language generated by G in the t-mode,* i.e., the set of all terminal objects derivable from the axiom in a halting computation. The family of languages generated by grammars of type X in the derivation mode δ, $\delta \in \{*, t\}$, is denoted by $\mathcal{L}_\delta(X)$. The family of recursively enumerable d-dimensional array languages is denoted by $\mathcal{L}_*(d\text{-}ARBA)$.

Arrays and array grammars

Let $d \in \mathbb{N}$; then a *d-dimensional array* $\mathcal{A}$ over an alphabet V is a function $\mathcal{A} : \mathbb{Z}^d \to V \cup \{\#\}$, where $shape(\mathcal{A}) = \left\{ v \in \mathbb{Z}^d \mid \mathcal{A}(v) \neq \# \right\}$ is finite and $\# \notin V$ is called the *background* or *blank symbol.* The set of all d-dimensional arrays over V is denoted by V^{*d}. For $v \in \mathbb{Z}^d$, $v = (v_1, \ldots, v_d)$, the norm of v is $\|v\| = \max\{|v_i| \mid 1 \leq i \leq d\}$. The *translation* $\tau_v : \mathbb{Z}^d \to \mathbb{Z}^d$ is defined by $\tau_v(w) = w + v$ for all $w \in \mathbb{Z}^d$. For any array $\mathcal{A} \in V^{*d}$, $\tau_v(\mathcal{A})$, the corresponding d-dimensional array translated by v, is defined by $(\tau_v(\mathcal{A}))(w) = \mathcal{A}(w - v)$ for all $w \in \mathbb{Z}^d$. For a (non-empty) finite set $W \subset Z^d$ the norm of W is defined as $\|W\| = \max\{\, \|v - w\| \mid v, w \in W \,\}$.

$[\mathcal{A}] = \left\{ \mathcal{B} \in V^{*d} \mid \mathcal{B} = \tau_v(\mathcal{A}) \text{ for some } v \in \mathbb{Z}^d \right\}$ is the equivalence class of arrays with respect to linear translations containing $\mathcal{A}$. The set of all equivalence

classes of d-dimensional arrays over V with respect to linear translations is denoted by $\left[V^{*d}\right]$ etc.

$G_A = \left((N \cup T)^{*d}, T^{*d}, \mathcal{A}_0, P, \Longrightarrow_{G_A}\right)$ is called a *d-dimensional array grammar*, where N is the alphabet of *non-terminal symbols*, T is the alphabet of *terminal symbols*, $N \cap T = \emptyset$, $\mathcal{A}_0 \in (N \cup T)^{*d}$ is the *start array*, P is a finite set of *d-dimensional array rules* over V, $V := N \cup T$, and $\Longrightarrow_{G_A} \subseteq (N \cup T)^{*d} \times (N \cup T)^{*d}$ is the derivation relation induced by the array rules in P.

A *d-dimensional contextual array rule* (see [5]) over the alphabet V is a pair of finite d-dimensional arrays $(\mathcal{A}_1, \mathcal{A}_2)$ with $dom\,(\mathcal{A}_1) \cap dom\,(\mathcal{A}_2) = \emptyset$ and $shape\,(\mathcal{A}_1) \cup shape\,(\mathcal{A}_2) \neq \emptyset$; we also call it an *array insertion rule*, as its effect is that in the context of $\mathcal{A}_1$ we insert $\mathcal{A}_2$; hence, we write $I\,(\mathcal{A}_1, \mathcal{A}_2)$. The pair $(\mathcal{A}_1, \mathcal{A}_2)$ can also be interpreted as having the effect that in the context of $\mathcal{A}_1$ we delete $\mathcal{A}_2$; in this case, we speak of an *array deletion rule* and write $D\,(\mathcal{A}_1, \mathcal{A}_2)$. For any (contextual, insertion, deletion) array rule we define its norm by $\|dom\,(\mathcal{A}_1) \cup dom\,(\mathcal{A}_2)\|$. The types of d-dimensional array grammars using array insertion rules of norm $\leq k$ and array deletion rules of norm $\leq m$ are denoted by d-$D^m I^k A$; without specific bounds, we write d-DIA.

(Sequential) P Systems

For the the area of P systems, we refer the reader to [6] and the P page [7].

A *(sequential) P system of type X with tree height n* is a construct $\Pi = (G, \mu, R, i_0)$ where $G = (O, O_T, A, P, \Longrightarrow_G)$ is a sequential grammar of type X; μ is the membrane (tree) structure of the system with the height of the tree being n, the membranes are uniquely labelled by labels from a set Lab; R is a set of rules of the form (h, r, tar) where $h \in Lab$, $r \in P$, and tar, called the *target indicator*, is taken from the set $\{here, in, out\} \cup \{in_h \mid h \in Lab\}$; i_0 is the initial membrane containing the axiom A.

A configuration of Π is a pair (w, h) where w is the current object (e.g., string or array) and h is the label of the membrane currently containing the object w. A sequence of transitions between configurations of Π, starting from the initial configuration (A, i_0), is called a *computation* of Π. A *halting computation* is a computation ending with a configuration (w, h) such that no rule from R_h can be applied to w anymore; w is called the *result* of this halting computation if $w \in O_T$. The language generated by Π, $L_t\,(\Pi)$, consists of all terminal objects from O_T being results of a halting computation in Π.

By $\mathcal{L}_t\,(X\text{-}LP)$ $(\mathcal{L}_t\left(X\text{-}LP^{\langle n \rangle}\right))$ we denote the family of languages generated by P systems (of tree height at most n) using grammars of type X. If only the targets *here*, *in*, and *out* are used, then the P system is called *simple*, and the corresponding families of languages are denoted by $\mathcal{L}_t\,(X\text{-}LsP)$ $(\mathcal{L}_t\left(X\text{-}LsP^{\langle n \rangle}\right))$.

Undecidability and Computational Completeness Results

An instance of the *Post Correspondence Problem (PCP)* is a pair of sequences of non-empty strings $(u_1, \ldots, u_n)$ and $(v_1, \ldots, v_n)$ over an alphabet T. A solution of

this instance is a sequence of indices $i_1, \ldots, i_k$ such that $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$; we call $u_{i_1} \ldots u_{i_k}$ the result of this solution. Let $L\left(\left(u_1, \ldots, u_n\right), \left(v_1, \ldots, v_n\right)\right)$ denote the set of results of all solutions of the instance $\left(\left(u_1, \ldots, u_n\right), \left(v_1, \ldots, v_n\right)\right)$ of the PCP, and let the homomorphism h_Σ be defined by $h_\Sigma : \Sigma \to \Sigma^+$ with $h_\Sigma(a) = aa'$ for all $a \in \Sigma$.

Lemma 1. *([3]) Let $I = \left(\left(u_1, \ldots, u_n\right), \left(v_1, \ldots, v_n\right)\right)$ be an instance of the PCP. Then we can effectively construct a one-dimensional array insertion P system Π such that*

$$[L(\Pi)] = \left\{LL'h_T(w)\,RR' \mid w \in L\left(\left(u_1, \ldots, u_n\right), \left(v_1, \ldots, v_n\right)\right)\right\}.$$

Hence, the emptiness problem for $\mathcal{L}_t\left(1\text{-}DIA\text{-}LP^{\langle k\rangle}\right)$ is undecidable.

For $d \geq 2$, even the emptiness problem for $\mathcal{L}_t(d\text{-}CA)$ is undecidable, see [1].

Theorem 2. *([3]) $\mathcal{L}_t\left(1\text{-}D^1I^1A\text{-}LsP^{\langle 2\rangle}\right) = \mathcal{L}_t\left(1\text{-}D^2I^2A\right) = \mathcal{L}_*(1\text{-}ARBA)$.*

It remains as an interesting question for future research whether this result for array grammars only using array insertion and deletion rules with norm at most two can also be achieved in higher dimensions; at least for dimension two, in [2] the corresponding computational completeness result has been shown for 2-dimensional array insertion and deletion P systems using rules with norm at most two.

Theorem 3. *([2]) $\mathcal{L}_t\left(2\text{-}D^2I^2A\text{-}LsP^{\langle 2\rangle}\right) = \mathcal{L}_*(2\text{-}ARBA)$.*

References

1. H. Fernau, R. Freund, and M. Holzer, Representations of recursively enumerable array languages by contextual array grammars, *Fundamenta Informaticae* **64** (2005), pp. 159–170.
2. H. Fernau, R. Freund, S. Ivanov, M. L. Schmid, and K. G. Subramanian, Array insertion and deletion P systems, in G. Mauri, A. Dennunzio, L. Manzoni, and A. E. Porreca, Eds., *UCNC 2013*, Milan, Italy, July 1–5, 2013, LNCS **7956**, Springer 2013, pp. 67–78.
3. R. Freund, S. Ivanov, M. Oswald, and K. G. Subramanian, One-dimensional array grammars and P systems with array insertion and deletion rules, *accepted for MCU 2013.*
4. R. Freund, M. Kogler, and M. Oswald, A general framework for regulated rewriting based on the applicability of rules, in J. Kelemen and A. Kelemenová, Eds., *Computation, Cooperation, and Life - Essays Dedicated to Gheorghe Păun on the Occasion of His 60th Birthday*, LNCS **6610**, Springer, 2011, pp. 35-53.
5. R. Freund, Gh. Păun, and G. Rozenberg, Contextual array grammars, in K.G. Subramanian, K. Rangarajan, and M. Mukund, Eds., *Formal Models, Languages and Applications*, Series in Machine Perception and Artificial Intelligence **66**, World Scientific, 2007, pp. 112–136.
6. Gh. Păun, G. Rozenberg, A. Salomaa, Eds., *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.
7. The P systems Web page, `http://ppage.psystems.eu/`.

Eine natürliche Membranhierachie

Henning Fernau[1], Rudolf Freund[2], Markus L. Schmid[1], K.G. Subramanian[3]& Petra Wiederhold[4]

[1] Fachbereich 4 – Abteilung Informatikwissenschaften, Universität Trier
D-54296 Trier, Germany
[2] Technische Universität Wien, Institut für Computersprachen, A-1040 Wien, Austria
[3] School of Computer Sciences, Universiti Sains Malaysia, 11800 Penang, Malaysia
[4] Department of Automatic Control,
Centro de Investigación y de Estudios Avanzados (CINVESTAV-IPN),
Av. I.P.N. 2508, Col. San Pedro Zacatenco, México 07000 D.F., México

Zusammenfassung Wir zeigen, dass kontextuelle Array-Grammatiken einen natürlichen Mechanismus liefern, über den eine beweisbar echte, durch die Anzahl von Membranen gegebene Hierarchie von Sprachfamilien angegeben werden kann.

1 Einführung

In der Theorie der P Systeme galt es längere Zeit als offene Frage, ein nicht-universelles Modell von P Systemen anzugeben, welches eine unendliche Hierarchie von Sprachfamilien anzugeben gestattet, die jeweils über eine Schranke auf die Anzahl der Membranen definiert ist [2, 3]. Geklärt wurde dies prinzipiell von Ibarra [1] durch Definition einer P System Variante, die Bezüge zu Zählerautomaten aufweist, wodurch sich die genannten Resultate erklären.

Kontextuelle Array-Regeln über dem Alphabet V haben die Gestalt $p = (\alpha, \beta)$, wobei α und β Abbildungen sind der Art $\mathbb{Z}^2 \to V$ mit endlichen aber disjunkten Definitionsbereichen U_α (dem sogenannten Selektor) bzw. U_β (dem Kontext). Für Arrays $\mathcal{C}_1, \mathcal{C}_2 \in V^{+2}$ gilt (intuitiv) $\mathcal{C}_1 \Longrightarrow_p \mathcal{C}_2$, falls wir in $\mathcal{C}_1$ und $\mathcal{C}_2$ ein Teilarray finden können, das dem Selektor (U_α, α) entspricht und falls die dem Kontext (U_β, β) entsprechenden Positionen nur mit dem Lückensymbol # in $\mathcal{C}_1$ gefüllt sind, sodass wir hier den Kontext (U_β, β) einfügen können, was somit $\mathcal{C}_2$ beschreibt. Wir betrachten im Wesentlichen den t-Modus bei der Ableitung; Regeln werden also solange ausgeführt, bis es nicht mehr geht.

2 P Systeme über kontextuelle Arrays

P Systeme über kontextuelle Arrays sind wie üblich aufgebaut, enthalten aber kontextuelle Array-Regeln in den einzelnen Membranen.

Anstelle einer formalen Definition betrachten wir die Sprache $L_{\text{star},4}$ der Sterne mit vier gleichlangen Armen über dem Alphabet$\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$:

$$L_{\text{star},4} := \{\mathtt{b\,a\,b}\begin{array}{c}\mathtt{c}\\\mathtt{a}\\\mathtt{b}\\\mathtt{a}\\\mathtt{b}\\\mathtt{a}\\\mathtt{b}\end{array}\mathtt{a\,b\,a\,b},\ \mathtt{b\,a\,b\,a\,b}\begin{array}{c}\mathtt{c}\\\mathtt{a}\\\mathtt{b}\\\mathtt{a}\\\mathtt{b}\\\mathtt{a}\\\mathtt{b}\end{array}\mathtt{a\,b\,a\,b\,a\,b},\ \mathtt{b\,a\,b\,a\,b\,a\,b}\begin{array}{c}\mathtt{c}\\\mathtt{a}\\\mathtt{b}\\\mathtt{a}\\\mathtt{b}\\\mathtt{a}\\\mathtt{b}\end{array}\mathtt{a\,b\,a\,b\,a\,b\,a\,b},\ldots\}$$

Wir zeigen, wie ein kontextuelles P System mit linearer Membranstruktur (und fünf Membranen) diese Sprache beschreibt.

$$\Pi_{\text{star},4} = (\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, \#, \mu, A_1, \ldots, A_5, P_1, \ldots, P_5, 5)\,,$$

wobei $\mu = [_1\,[_2\,[_3\,[_4\,[_5\,]_5\,]_4\,]_3\,]_2\,]_1$. Die Axiomenmengen sind gegeben durch:

$$A_1 = \left\{\mathtt{a\,b}\begin{array}{c}\mathtt{a}\\\mathtt{b}\\\mathtt{a}\\\mathtt{b}\\\mathtt{a}\end{array}\mathtt{b\,a}\right\} \text{ and } A_2 = A_3 = A_4 = A_5 = \emptyset\,,$$

Die Regeln sind definiert als:

$$\begin{aligned}
P_1 &:= \{p_{1,1}, p_{1,2}\} := \left\{\left(\begin{array}{l}\mathtt{b}\\\boxed{\mathtt{a}},\ \text{in}\\\boxed{\mathtt{b}}\end{array}\right), \left(\begin{array}{l}\mathtt{c}\\\boxed{\mathtt{a}},\ \text{in}\\\boxed{\mathtt{b}}\end{array}\right)\right\},\\
P_2 &:= \{p_{2,1}, p_{2,2}\} := \left\{\left(\boxed{\mathtt{b}}\,\boxed{\mathtt{a}}\,\mathtt{b},\ \text{in}\right), \left(\boxed{\mathtt{a}}\,\boxed{\mathtt{b}}\,\mathtt{a},\ \text{out}\right)\right\},\\
P_3 &:= \{p_{3,1}, p_{3,2}\} := \left\{\left(\begin{array}{l}\boxed{\mathtt{b}}\\\boxed{\mathtt{a}},\ \text{in}\\\mathtt{b}\end{array}\right), \left(\begin{array}{l}\boxed{\mathtt{a}}\\\boxed{\mathtt{b}},\ \text{out}\\\mathtt{a}\end{array}\right)\right\},\\
P_4 &:= \{p_{4,1}, p_{4,2}\} := \left\{\left(\mathtt{b}\,\boxed{\mathtt{a}}\,\boxed{\mathtt{b}},\ \text{in}\right), \left(\mathtt{a}\,\boxed{\mathtt{b}}\,\boxed{\mathtt{a}},\ \text{out}\right)\right\},\\
P_5 &:= \{p_{5,1}\} := \left\{\left(\begin{array}{l}\mathtt{a}\\\boxed{\mathtt{b}},\ \text{out}\\\boxed{\mathtt{a}}\end{array}\right)\right\}.
\end{aligned}$$

Die Idee ist, dass die Arrays während einer Berechnung ständig zwischen den den Membranen 1 und 5 hin- und herpendeln. c dient lediglich zum Anhalten dieser Berechnung.

3 Das Hierarchie-Resultat

In ähnlicher Weise ist auch die Kammsprache $L_{\mathrm{comb},k}$ aufgebaut, welche k "Zähne" enthält in folgender Weise:

$$
\begin{array}{ccccccccc}
 & & & & & & & \mathtt{X} & \\
 & & & & & & & \mathtt{b}_3 & \\
 & & & & & & & \mathtt{b}_2 & \\
 & & & & & & & \mathtt{b}_1 & \\
 & & & & & & & \vdots & \\
\mathtt{a}_1 & & \mathtt{a}_2 & & \mathtt{a}_3 & \dots & \mathtt{a}_k & & \\
\vdots & & \vdots & & \vdots & \vdots & \vdots & \mathtt{b}4 & \\
 & & & & & & & \mathtt{b}3 & \\
\mathtt{a}_1 & & \mathtt{a}_2 & & \mathtt{a}_3 & \dots & \mathtt{a}_k & \mathtt{b}_2 & \\
\mathtt{a}_1 & \mathtt{X} & \mathtt{a}_2 & \mathtt{X} & \mathtt{a}_3 & \dots & \mathtt{a}_k & \mathtt{X}\ \mathtt{b}_1 &
\end{array}
.
$$

Konkreter gilt: $L_{\mathrm{comb},4}$ ist gleich

$$
\left\{
\begin{array}{ccccccccccccccccccccccccccccccccc}
 & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & \mathtt{X} & & & & \\
 & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & \mathtt{b}_3 & & & & \\
 & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & & \mathtt{b}_2 & & & & \\
 & & & & & & & & & & & & & & & & & & & & \mathtt{X} & & & & & & & & & & & & \mathtt{b}_1 & & & & \\
 & & & & & & & & & & & & & & & & & & & & \mathtt{b}_3 & & & & & & & & & & & & \mathtt{b}_4 & & & & \\
 & & & & & & & & & & & & & & & & & & & & \mathtt{b}_2 & & & & & & & & & & & & \mathtt{b}_3 & & & & \\
 & & & & & & & & \mathtt{X} & & & & & & & & & & & & \mathtt{b}_1 & & & & & & & & & & & & \mathtt{b}_2 & & & & \\
 & & & & & & & & \mathtt{b}_3 & & & & & & & & & & & & \mathtt{b}_4 & & & & & & & & & & & & \mathtt{b}_1 & & & & \\
\mathtt{a}_1 & & \mathtt{a}_2 & & \mathtt{a}_3 & & \mathtt{a}_4 & & \mathtt{b}_2 & , & & & \mathtt{a}_1 & & \mathtt{a}_2 & & \mathtt{a}_3 & & \mathtt{a}_4 & & \mathtt{b}_3 & , & & & & & & & & & & & & \mathtt{b}_4 & , & \dots & \\
\mathtt{a}_1 & \mathtt{X} & \mathtt{a}_2 & \mathtt{X} & \mathtt{a}_3 & \mathtt{X} & \mathtt{a}_4 & \mathtt{X} & \mathtt{b}_1 & & & & \mathtt{a}_1 & & \mathtt{a}_2 & & \mathtt{a}_3 & & \mathtt{a}_4 & & \mathtt{b}_2 & & & & \mathtt{a}_1 & & \mathtt{a}_2 & & \mathtt{a}_3 & & \mathtt{a}_4 & & \mathtt{b}_3 & & & \\
 & & & & & & & & & & & & \mathtt{a}_1 & \mathtt{X} & \mathtt{a}_2 & \mathtt{X} & \mathtt{a}_3 & \mathtt{X} & \mathtt{a}_4 & \mathtt{X} & \mathtt{b}_1 & & & & \mathtt{a}_1 & & \mathtt{a}_2 & & \mathtt{a}_3 & & \mathtt{a}_4 & & \mathtt{b}_4 & & & \\
 & & & & & & & & & & & & & & & & & & & & & & & & \mathtt{a}_1 & & \mathtt{a}_2 & & \mathtt{a}_3 & & \mathtt{a}_4 & & \mathtt{b}_3 & & & \\
 & & & & & & & & & & & & & & & & & & & & & & & & \mathtt{a}_1 & & \mathtt{a}_2 & & \mathtt{a}_3 & & \mathtt{a}_4 & & \mathtt{b}_2 & & & \\
 & & & & & & & & & & & & & & & & & & & & & & & & \mathtt{a}_1 & \mathtt{X} & \mathtt{a}_2 & \mathtt{X} & \mathtt{a}_3 & \mathtt{X} & \mathtt{a}_4 & \mathtt{X} & \mathtt{b}_1 & & & &
\end{array}
\right\}
$$

Wir konnten zeigen:

- Es gibt ein kontextuelles Array P System mit $k+1$ Membranen, welches $L_{\mathrm{comb},k}$ beschreibt.
- $L_{\mathrm{comb},k}$ lässt sich jedoch nicht durch ein kontextuelles Array P System mit nur $\lfloor \frac{k}{3} \rfloor$ vielen Membranen erzeugen.

Dies zeigt die behauptete unendliche Hierarchie von Sprachfamilien. Der Beweis von Punkt zwei arbeitet mit dem Schubfachprinzip, aufbauend auf einer Kategorisierung der möglichen Regeln.

Eine interessante offene Frage ist, ob die unendliche Hierarchie auch von Stufe zu Stufe echt ist. Dies konnten wir an einzelnen Beispielen nur für kleine Anzahlen von Membranen beweisen.

Wir möchten abschließend noch darauf hinweisen, dass die Ergebnisse anders ausfallen, wenn Selektoren das Lückensymbol enthalten dürfen. Dann kann die Sprache $L_{\mathrm{comb},k}$ bereits mit zwei Membranen erzeugt werden. Ebenso ist noch offen, inwieweit die Alphabetgröße auf eine Konstante beschränkt werden kann.

Wir danken der Förderung des vierten Autors durch das Projekt "Universität der Großregion", welches einen Forschungsaufenthalt an der Universität Trier ermöglichte.

Literatur

1. O. H. Ibarra. The number of membranes matters. In C. Martín-Vide, G. Mauri, Gh. Paun, G. Rozenberg, and A. Salomaa, editors, *Workshop on Membrane Computing, WMC 2003*, volume 2933 of *LNCS*, pages 218–231. Springer, 2004.
2. C. Martín-Vide and Gh. Păun, editors. *Recent Topics in Mathematical and Computational Linguistics.* Editura Academiei Române, 2000.
3. The P systems Web page, `http://ppage.psystems.eu/`.

On Boolean closed full trios and rational Kripke frames

Markus Lohrey[1] and Georg Zetzsche[2]

[1] Universität Siegen
lohrey@informatik.uni-leipzig.de
[2] Technische Universität Kaiserslautern
zetzsche@cs.uni-kl.de

Abstract. A Boolean closed full trio is a class of languages that is closed under Boolean operations (union, intersection, and complement) and rational transductions. It is well-known that the regular languages constitute such a Boolean closed full trio. We present a result stating that every such language class that contains any non-regular language already contains the whole arithmetical hierarchy.
Our construction also shows that there is a fixed rational Kripke frame such that assigning an arbitrary non-regular language to some variable allows the interpretation of any language from the arithmetical hierarchy in the corresponding Kripke structure.

Given alphabets X and Y, a *rational transduction* is a rational subset of the monoid $X^* \times Y^*$. For a language $L \subseteq Y^*$ and a rational transduction R, we write $RL = \{x \in X^* \mid \exists y \in L : (x, y) \in R\}$.

A class $\mathcal{C}$ of languages is called a *full trio* if it is closed under (arbitrary) homomorphisms, inverse homomorphisms, and regular intersections. It is well-known [2] that a class $\mathcal{C}$ is a full trio if and only if it is closed under rational transductions, i.e., for every $L \in \mathcal{C}$ and every rational transduction R, we have $RL \in \mathcal{C}$. We call a language class *Boolean closed* if it is closed under all Boolean operations (union, intersection, and complementation).

For any language class $\mathcal{C}$, we write $\mathsf{RE}(\mathcal{C})$ for the class of languages accepted by some Turing machine with an oracle $L \in \mathcal{C}$. Furthermore, let REC denote the class of recursive languages. Then the *arithmetical hierarchy* (see, for example, [3]) is defined as

$$\Sigma_0 = \mathsf{REC}, \quad \Sigma_{n+1} = \mathsf{RE}(\Sigma_n) \text{ for } n \geq 0.$$

Languages in $\bigcup_{n \geq 0} \Sigma_n$ are called *arithmetical*. It is well-known that the class of regular languages constitutes a Boolean closed full trio. These closure properties of the regular languages allow for a rich array of applications. Aside from the insights gained from the individual closure properties, this particular collection is exploited, for example, in the theory of automatic structures, since it implies that in such structures, every first-order definable relation can be represented by a regular language. Since emptiness is decidable for regular languages, one can therefore decide the first-order theory of these structures.

Hence, the question arises whether there are languages classes beyond the regular languages that enjoy these closure properties. Our first main result states that every such language class already contains the whole arithmetical hierarchy and thus loses virtually all decidability properties.

Theorem 1. *Let $\mathcal{C}$ be a Boolean closed full trio. If $\mathcal{C}$ contains any non-regular language, then $\mathcal{C}$ contains the arithmetical hierarchy.*

Actually, it turns out that a fixed set of rational transductions suffice to construct all arithmetical languages from any non-regular language:

Theorem 2. *There exist a fixed alphabet X and a list $R_1, \ldots, R_n \subseteq X^* \times X^*$ of rational transductions such that for every non-regular language $L \subseteq X^*$, every arithmetical language can be constructed from L using Boolean operations and applications of the rational transductions $R_1, \ldots, R_n$.*

It should be noted that Theorem 1 and 2 do not mean that there is no way of developing a theory of automatic structures beyond regular languages. It might well be that some smaller collection of closure properties suffices to obtain all first-order definable relations and still admits a decision procedure for the emptiness problem.

A large number of grammar and automata models is easily seen to produce only recursively enumerable languages. Hence, Theorem 1 also implies that the corresponding language classes are never Boolean closed full trios.

Theorem 2 can be also restated in terms of multimodal logic. A *Kripke structure* (or edge- and node-labeled graph) is a tuple

$$\mathcal{K} = (V, (E_a)_{a \in A}, (U_p)_{p \in P}),$$

where V is a set of nodes (also called worlds), A and P are finite sets of actions and propositions, respectively, for every $a \in A$, $E_a \subseteq V \times V$, and for every $p \in P$, $U_p \subseteq V$. The tuple $\mathcal{F} = (V, (E_a)_{a \in A})$ is then also called a *Kripke frame*. We say that $\mathcal{K}$ (and $\mathcal{F}$) is *word-based* if $V = X^*$ for some finite alphabet X. Formulas of *multimodal logic* are defined by the following grammar, where $p \in P$ and $a \in A$:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box_a \varphi \mid \Diamond_a \varphi.$$

The semantics $[\![\varphi]\!]_{\mathcal{K}} \subseteq V$ of formula φ in $\mathcal{K}$ is inductively defined as follows:

$$\begin{aligned}
[\![p]\!]_{\mathcal{K}} &= U_p, \\
[\![\neg\varphi]\!]_{\mathcal{K}} &= V \setminus [\![\varphi]\!]_{\mathcal{K}}, \\
[\![\varphi \wedge \psi]\!]_{\mathcal{K}} &= [\![\varphi]\!]_{\mathcal{K}} \cap [\![\psi]\!]_{\mathcal{K}}, \\
[\![\varphi \vee \psi]\!]_{\mathcal{K}} &= [\![\varphi]\!]_{\mathcal{K}} \cup [\![\psi]\!]_{\mathcal{K}}, \\
[\![\Box_a \varphi]\!]_{\mathcal{K}} &= \{v \in V \mid \forall u \in V : (v, u) \in E_a \to u \in [\![\varphi]\!]_{\mathcal{K}}\}, \\
[\![\Diamond_a \varphi]\!]_{\mathcal{K}} &= \{v \in V \mid \exists u \in V : (v, u) \in E_a \wedge u \in [\![\varphi]\!]_{\mathcal{K}}\}.
\end{aligned}$$

A word-based Kripke frame $\mathcal{F} = (X^*, (E_a)_{a \in A})$ is called *rational* if every E_a is a rational transduction. A word-based Kripke structure $\mathcal{K} = (X^*, (E_a)_{a \in A}, (U_p)_{p \in P})$

is called *rational* if every relation E_a is a rational transduction and every U_p is a regular language. The closure properties of regular languages imply that for every rational Kripke structure $\mathcal{K}$ and every multimodal formula φ, the set $[\![\varphi]\!]_{\mathcal{K}}$ is a regular language that can be effectively constructed from φ and (automata describing the structure) $\mathcal{K}$. Using this fact, Bekker and Goranko [1] proved that the model-checking problem for rational Kripke structures and multimodal logic is decidable. Our reformulation of Theorem 2 in terms of multimodal logic is:

Theorem 3. *There exist a fixed alphabet X and a fixed rational Kripke frame $\mathcal{F} = (X^*, R_1, \ldots, R_n)$ such that for every non-regular language $L \subseteq X^*$ and every arithmetical language $A \subseteq X^*$ there exists a multimodal formula φ such that $A = [\![\varphi]\!]_{\mathcal{K}}$, where $\mathcal{K} = (X^*, R_1, \ldots, R_n, L)$.*

References

1. W. Bekker and V. Goranko. Symbolic model checking of tense logics on rational kripke models. In M. Archibald, V. Brattka, V. Goranko, and B. Löwe, editors, *ILC*, volume 5489 of *Lecture Notes in Computer Science*, pages 2–20. Springer, 2007.
2. J. Berstel. *Transductions and context-free languages.* Teubner, Stuttgart, 1979.
3. D. C. Kozen. *Automata and computability.* Springer-Verlag, New York, 1997.

Characterizing Probabilistic ω-Recognizability by MSO Logic and Regular Expressions

Thomas Weidner

Universität Leipzig

1 Introduction

Probabilistic automata, introduced already by Rabin [12], form a flourishing field. Their applications range from speech recognition [13] over prediction of climate parameters [10] to randomized distributed systems [9]. For surveys of theoretical results see the books [11,4]. Recently, the concept of probabilistic automata has been transferred to infinite words by Baier and Grösser [1]. This concept led to further research [2,5,6,7,8,14].

Though probabilistic automata admit a natural quantitative behavior, namely the acceptance probability of each word, the main research interest has been towards qualitative properties (for instance the language of all words with positive acceptance probability). We consider the behavior of a probabilistic automaton as function mapping finite or infinite words to a probability value.

On the other hand, there are two classical characterizations of recognizable languages: Büchi and Elgot established the characterization using monadic second order logic, and Kleene used regular expressions as a model expressively equivalent to recognizable languages. Both results were a paramount success with applications in all fields of theoretical computer science. We establish both results in the context of probabilistic ω-automata.

2 Definitions and Results

For the following, let Σ always denote an alphabet and $w \in \Sigma^\omega$ a word. For a set X, we denote the set of all probability distributions on X by $\Delta(X)$. Given a $p \in (0,1)$, we use the Bernoulli measure B_p on $\{0,1\}^\omega$ which is uniquely defined by $\mathrm{B}_p(u_1 \cdots u_n\{0,1\}^\omega) = p^{|\{i|u_i=1\}|}(1-p)^{|\{i|u_i=0\}|}$ for $u_1, \ldots, u_n \in \{0,1\}$. As there is a natural bijection from $\{0,1\}^\omega$ to $2^\mathbb{N}$, we regard B_p also as a measure on $2^\mathbb{N}$.

Definition 1 (Probabilistic Muller automata). *A* probabilistic Muller automaton *is a tuple* $A = (Q, \delta, \mu, \mathrm{Acc})$ *where* $-$ Q *is a non-empty, finite* set of states $-$ $\delta\colon Q \times \Sigma \to \Delta(Q)$ *the* transition probability function $-$ $\mu \in \Delta(Q)$ *the* initial probabilities $-$ $\mathrm{Acc} \subseteq 2^Q$ *a* Muller acceptance condition. *For a word* $w = w_1 w_2 \cdots \in \Sigma^\omega$*, the behavior of* A *is defined using the unique measure* $\mathbb{P}_A^w$ *on* Q^ω *defined by* $\mathbb{P}_A^w(q_0 \cdots q_n Q^\omega) = \mu(q_0) \prod_{i=1}^n \delta(q_{i-1}, w_i, q_i)$*. The* behavior $\|A\|$ *of* A *is then given by* $\|A\|(w) = \mathbb{P}_A^w(\rho \in Q^\omega \ ; \ \inf(\rho) \in \mathrm{Acc})$*, where* $\inf(\rho)$ *designates the set of states occurring infinitely often in* ρ*.*

Next, we define the syntax and semantics of probabilistic MSO logic and probabilistic ω-regular expressions.

Definition 2 (Probabilistic MSO logic). *The syntax of a probabilistic MSO φ is given in BNF by*

$$\varphi ::= \psi \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathbb{E}_p X.\varphi,$$
$$\psi ::= \mathrm{P}_a(x) \mid x \in X \mid x \leq y \mid \psi \wedge \psi \mid \neg\psi \mid \forall x.\psi \mid \forall X.\psi,$$

where ψ denotes a Boolean PMSO formula*, $a \in \Sigma$, and x, y (X) are first-order (resp. second-order) variables.*

The semantics $[\![\varphi]\!]$ are defined inductively: Given a word $w \in \Sigma^\omega$ and an assignment $\alpha\colon \mathcal{V} \to \mathbb{N} \cup 2^{\mathbb{N}}$, for a Boolean formula ψ we define $[\![\psi]\!](w,\alpha) = 1$ if (w,α) satisfies ψ in the classical sense and $[\![\varphi]\!](w,\alpha) = 0$ otherwise. For probabilistic MSO formulas φ the semantics are given by

$$[\![\varphi_1 \wedge \varphi_2]\!](w,\alpha) = [\![\varphi_1]\!](w,\alpha) \cdot [\![\varphi_2]\!](w,\alpha), \qquad [\![\neg\varphi]\!](w,\alpha) = 1 - [\![\varphi]\!](w,\alpha),$$
$$[\![\mathbb{E}_p X.\varphi]\!](w,\alpha) = \int_{2^{\mathbb{N}}} [\![\varphi]\!](w, \alpha[X \mapsto M])\ \mathrm{B}_p(\mathrm{d}M).$$

Definition 3 (Probabilistic ω-regular expressions). *The set $p\omega RE$ of all probabilistic ω-regular expressions is the smallest set satisfying*

- *$\Sigma^\omega \in p\omega RE$*
- *If $\emptyset \neq A \subseteq \Sigma$ and $(E_a)_{a\in A} \in p\omega RE$, then $\sum_{a\in A} aE_a \in p\omega RE$*
- *If $p \in [0,1]$ and $E, F \in p\omega RE$, then $pE + (1-p)F \in p\omega RE$ and $pE \in p\omega RE$*
- *If $E\Sigma^\omega \in p\omega RE$ and $F \in p\omega RE$, then $EF \in p\omega RE$*
- *If $E\Sigma^\omega + F \in p\omega RE$, then $E^*F + E^\omega \in p\omega RE$, $E^\omega \in p\omega RE$ and $E^*F \in p\omega RE$*
- *The set $p\omega RE$ is closed under distributivity of $\cdot$ over $+$, associativity, and commutativity of $+$ and multiplication by real numbers.*

The semantics of probabilistic ω-regular expressions are inductively defined by $\|\Sigma^\omega\|(w) = \mathbb{1}_{\Sigma^\omega}(w)$ and

$$\|a\|(w) = \begin{cases} 1 & \text{if } w = a \\ 0 & \text{otherwise} \end{cases} \qquad \|EF\|(w) = \sum_{uv=w} \|E\|(u) \cdot \|F\|(v)$$
$$\|p\|(w) = \begin{cases} p & \text{if } w = \varepsilon \\ 0 & \text{otherwise} \end{cases} \qquad \|E^*\|(w) = \sum_{u_1\cdots u_n = w} \|E\|(u_1)\cdots\|E\|(u_n)$$
$$\|E+F\|(w) = \|E\|(w) + \|F\|(w) \qquad \|E^\omega\|(w) = \lim_{n\to\infty} \sum_{u_1\cdots u_n v = w} \|E\|(u_1)\cdots\|E\|(u_n)$$

where $a \in \Sigma$ and $p \in [0,1]$.

Note that the semantics of $E + F$, E^*, and E^ω are well-defined because of the syntax restrictions imposed on pωRE.

Theorem 4. *Let $f\colon \Sigma^\omega \to [0,1$. The following statements are equivalent*

1. *$f = \|A\|$ for a probabilistic Muller automaton A*
2. *$f = [\![\varphi]\!]$ for a probabilistic MSO sentence φ*
3. *$f = \|E\|$ for a probabilistic ω-regular expression E*

All equivalences given in Theorem 1 are shown using effective constructions. Hence decidability results for probabilistic ω-automata transfer to probabilistic MSO logic and probabilistic ω-regular expressions. For example it is undecidable, given an automaton A, a sentence φ, or an expression E, if there is a word $w \in \Sigma^\omega$ with $\|A\|(w) > 0$, resp. $[\![\varphi]\!](w) > 0$, resp. $\|E\|(w) > 0$.

If an expression E is *ω-deterministic*, i.e. no sub-expression F^ω in E contains probability constants other than 0 or 1, then the above problem is decidable.

References

1. Baier, C., Grösser, M.: Recognizing ω-regular languages with probabilistic automata. In: Proc. LICS. pp. 137 – 146. IEEE (2005)
2. Baier, C., Bertrand, N., Grösser, M.: On decision problems for probabilistic Büchi automata. In: FoSSaCS, LNCS, vol. 4962, pp. 287–301. Springer (2008)
3. Bollig, B., Gastin, P., Monmege, B., Zeitoun, M.: A probabilistic kleene theorem. In: ATVA, pp. 400–415. LNCS, Springer Berlin Heidelberg (2012)
4. Bukharaev, R.G.: Theorie der stochastischen Automaten. Teubner (1995)
5. Chadha, R., Sistla, A.P., Viswanathan, M.: Probabilistic Büchi automata with non-extremal acceptance thresholds. In: VMCAI. pp. 103–117. Springer (2011)
6. Chatterjee, K., Doyen, L., Henzinger, T.A.: Probabilistic weighted automata. In: Proc. CONCUR. LNCS, vol. 5710, pp. 244–258. Springer (2009)
7. Chatterjee, K., Henzinger, T.: Probabilistic automata on infinite words: Decidability and undecidability results. In: ATVA, LNCS, vol. 6252, pp. 1–16. Springer (2010)
8. Chatterjee, K., Tracol, M.: Decidable problems for probabilistic automata on infinite words. In: LICS. pp. 185–194. IEEE Computer Society (2012)
9. Cheung, L., Lynch, N., Segala, R., Vaandrager, F.: Switched PIOA: Parallel composition via distributed scheduling. TCS 365(1–2), 83 – 108 (2006)
10. Mora-López, L., Morales, R., Sidrach de Cardona, M., Triguero, F.: Probabilistic finite automata and randomness in nature: a new approach in the modelling and prediction of climatic parameters. Proc. International Environmental Modelling and Software Congress pp. 78–83 (2002)
11. Paz, A.: Introduction to Probabilistic Automata. Computer science and applied mathematics, Academic Press, Inc. (1971)
12. Rabin, M.O.: Probabilistic automata. Information and Control 6(3), 230 – 245 (1963)
13. Ron, D., Singer, Y., Tishby, N.: The power of amnesia: Learning probabilistic automata with variable memory length. Machine Learning 25, 117–149 (1996)
14. Tracol, M., Baier, C., Größer, M.: Recurrence and transience for probabilistic automata. In: FSTTCS. LIPIcs, vol. 4, pp. 395–406. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2009)

The Support of Weighted Unranked Tree Automata

Manfred Droste and Doreen Götze

Institut für Informatik, Universität Leipzig, D-04109 Leipzig, Germany
{droste,goetze}@informatik.uni-leipzig.de

Unranked trees are a common concept in computer science. For instance (fully structured) XML-documents can be seen as unranked trees. With the help of unranked tree automata, one can investigate qualitative questions on XML-documents.

To allow the study of quantitative aspects, Droste and Vogler (TOCS 2011) recently proposed and investigated bottom-up weighted unranked tree automata over semirings. These automata relate each of their states $q \in Q$ and each letter a to a weighted finite string automaton $\mathcal{A}_{q,a}$ over the same semiring and the set Q as alphabet. For the definition of the behavior of weighted unranked tree automata, Droste and Vogler employed two distinct approaches, which for semirings coincide. The first one can be viewed as a conceptual one; it uses runs, which associate each position of an input tree to a state, and proceeds by calling for every position of the tree the weighted string automaton associated with the state and the letter at that position. The second definition is based on extended runs and could be viewed as an operational model; extended runs consist of global trees composed of local runs of the automata associated with the state and the letter assigned to each position of a given input tree.

We wish to investigate the support of weighted unranked tree automata; it is defined as the language of all trees evaluated to non-zero.

We will consider strong bimonoids as weight structures which can be viewed as 'semirings without requiring distributivity'. These form a much larger class of weight structures than semirings. For instance, they contain all bounded (non-distributive) lattices, which occur in multi-valued logics, and also weight structures recently investigated e.g. by Chatterjee, Doyen, and Henzinger for modeling peak power consumption of energy.

The main results of our paper are the following:

- We introduce weighted unranked tree automata over strong bimonoids by using valid extended runs.
- We show that for zero-sum free, commutative strong bimonoids, the support of a weighted unranked tree automaton is recognizable.
- An unranked tree automaton for the support can be effectively constructed if Kirsten's zero generation problem (DLT 2009) is decidable, even if the bimonoid operations are not computable.

For our proofs, we heavily use the methods of Kirsten (DLT 2009), but adjusted for constructing (the technically more involved) unranked tree automata.

Recognisability for infinite trees

Achim Blumensath

TU Darmstadt

Abstract. We develop an algebraic language theory for languages of infinite trees. This theory is based on so-called *ω-hyperclones.* We show that a language of infinite trees is regular if, and only if, it is recognised by a finitary path-continuous ω-hyperclone. Aiming at applications to decision procedures we also introduce the corresponding notion of a Wilke algebra and we prove that the omega-power of a Wilke algebra uniquely determines the infinite product. As an application we give a purely algebraic proof of Rabin's Tree Theorem.

Instead of using finite automata to develop the theory of regular languages, one can also employ semigroup theory. By now this approach has a long tradition and there exists an extended structure theory connecting varieties of languages with finite semigroups. This theory is particularly effective if one is interested in characterising subclasses of the class of all regular languages. For instance, the only known decidable characterisation of the class of first-order definable languages is based on semigroup theory.

Naturally, there have been attempts to generalise this theory to other notions of regularity. For languages of ω-words, such a generalisation has largely been achieved. A detailed account can be found in the book of Pin and Perrin [11]. There also have been several contributions to an algebraic theory for languages of finite trees [15, 18, 16, 5, 12, 7, 6, 13, 3, 1, 8]. But the resulting theory is still fragmentary with several competing approaches and formalisations. Our own work has been influenced in particular by the following two articles: Ésik and Weil [7] have developed an approach using *preclones,* while Bojańczyk and Walukiewicz [3] use *forest algebras.* As far as the algebraic setting is concerned, the formalisation in the present article most closely resembles the work on *clones* by Ésik [5].

So far, an algebraic theory for languages of *infinite trees* is still missing. The main obstacle is the lack of appropriate combinatorial tools, like Ramseyan factorisation theorems for infinite trees. In particular, a purely combinatorial proof that every nonempty regular language of infinite trees contains a regular tree is still missing. There is recent work of Bojańczyk and Idziaszek [2] on characterisation results for classes of infinite trees that manages to circumvent these problems by a technical trick: since every regular language of infinite trees is determined by the regular trees it contains, it is sufficient to consider only regular trees.

In this talk I provide steps in the development of an algebraic theory for recognisability of classes of infinite trees. Inspired by the work of Ésik and Weil

on preclones, we define suitable algebras of infinite trees called *ω-hyperclones.* We can show that every regular language is recognised by some homomorphism into such a (finitary, path-continuous) ω-hyperclone.

The proof is performed in two steps. First, we define a special class of ω-hyperclones called *path-hyperclones* that directly correspond to tree-automata. The problem with path-hyperclones is that their definition is not axiomatic, but syntactic. That is, given an arbitrary ω-hyperclone we cannot tell from the definition whether or not this ω-hyperclone is isomorphic to some path-hyperclone.

In the second step, we therefore give an algebraic characterisation of the main properties of such path-hyperclones (they are *path-continuous*). Using this result we can transfer our characterisation from path-hyperclones to path-continuous ω-hyperclones.

Finally, we prove that the class of path-continuous ω-hyperclones is closed under products and a certain power-set operation. From these results we can deduce a second (equivalent) version of our main theorem: recognisability by finitary path-continuous ω-hyperclones is the same as definability in monadic second-order logic.

Historically, one of the main advantages of algebraic characterisations of regularity has been their suitability for deriving decision procedures for subclasses of regular languages. We hope that the theory developed in this talk will also be useful in this respect. One of the prerequisites for obtaining decision procedures is that the algebras one is dealing with are finitely representable. For the algebras introduced above this is, in fact, the case.

As an application of this result, we present a new proof of Rabin's Tree Theorem, which states that the monadic second-order theory of the infinite complete binary tree is decidable. The standard proof of this result is based on the translation of monadic second-order formulae into tree automata. The required automata-theoretic machinery in turn rests on two main results: (i) the positional determinacy of parity games and (ii) either a determinisation construction for Büchi automata, or an analogous result for tree automata (see [9, 10]).

For Büchi's Theorem – the corresponding result for the natural numbers with successor relation – there exists, besides the usual automata-theoretic proof, an alternative proof due to Shelah [14, 17]. It is purely combinatorial in nature and it is based on Feferman-Vaught like composition arguments for monadic second-order theories.

For a long time it has been an open problem to extend these results to the theory of the binary tree. What was missing in order to transfer Shelah's proof was a suitable variant of Ramsey's Theorem for trees. Such a theorem has recently been provided by Colcombet [4]. In this article we use Colcombet's result to prove that our algebras have finite representations. This fact is then used to give an alternative proof of Rabin's theorem without references to automata or games.

References

1. P. Baltazar. M-Solid Varieties of Languages. *Acta Cynbernetica*, 18:719–731, 2008.
2. M. Bojańczyk and T. Idziaszek. Algebra for Infinite Forests with an Application to the Temporal Logic EF. In *Proc. 20th International Conference on Concurrency Theory, CONCUR, LNCS 5710*, pages 131–145, 2009.
3. M. Bojańczyk and I. Walukiewicz. Forest Algebras. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives*, pages 107–132. Amsterdam University Press, 2007.
4. T. Colcombet. On Factorization Forests. Technical Report hal-00125047, Irisa Rennes, 2007.
5. Z. Ésik. A variety theorem for trees and theories. *Publ. Math. Debrecen*, 54:711–762, 1999.
6. Z. Ésik. Characterizing CTL-like logics on finite trees. *Theoretical Computer Science*, 356:136–152, 2006.
7. Z. Ésik and P. Weil. Algebraic recognizability of regular tree languages. *Theoretical Computer Science*, 340:291–321, 2005.
8. Z. Ésik and P. Weil. Algebraic Characterization of Logically Defined Tree Languages. *International Journal of Algebra and Computation*, 20:195–239, 2010.
9. E. Grädel, W. Thomas, and T. Wilke. *Automata, Logic, and Infinite Games*. LNCS 2500. Springer-Verlag, 2002.
10. D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141:69–107, 1995.
11. D. Perrin and J.-E. Pin. *Infinite Words – Automata, Semigroups, Logic and Games*. Elsevier, 2004.
12. T. Petkovic and S. Salehi. Positive varieties of tree languages. *Theoretical Computer Science*, 347:1–35, 2005.
13. S. Salehi and M. Steinby. Tree algebras and varieties of tree languages. *Theoretical Computer Science*, 377:1–24, 2007.
14. S. Shelah. The Monadic Second Order Theory of Order. *Annals of Mathematics*, 102:379–419, 1975.
15. M. Steinby. A theory of tree language varieties. In *Tree Automata and Languages*, pages 57–82. North-Holland, 1992.
16. M. Steinby. General Varieties of Tree Languages. *Theoretical Computer Science*, 205:1–43, 1998.
17. W. Thomas. Ehrenfeucht Games, the Composition Method, and the Monadic Theory of Ordinal Words. *LNCS*, 1261:118–143, 1997.
18. T. Wilke. Algebras for Classifying Regular Tree Languages and an Application to Frontier Testability. In *Proc. 20th Int. Colloquium on Automata, Languages and Programming, ICALP 1993, LNCS 700*, pages 347–358, 1993.

Inner Palindromic Closure

Jürgen Dassow[1], Florin Manea[2], Robert Mercaş[2], and Mike Müller[2]

[1] Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik
[2] Christian-Albrechts-Universität zu Kiel, Institut für Informatik

Abstract. We introduce the inner palindromic closure as a new operation $\spadesuit$, which consists in expanding a factor u to the left or right by v such that vu or uv, respectively, is a palindrome of minimal length. We investigate several language theoretic properties of the iterated inner palindromic closure $\spadesuit^*(w) = \bigcup_{i\geq 0} \spadesuit^i(w)$ of a word w.

1 Introduction

The investigation of repetitions of factors in a word is a very old topic in formal language theory. For instance, already in 1906, Thue proved that there exists an infinite word over an alphabet with three letters which has no factor of the form ww. Since the eighties a lot of papers on combinatorial properties concerning repetitions of factors were published (see [4] and the references therein).

The duplication got further interest in connection with its importance in natural languages and in DNA sequences and chromosomes. Motivated by these applications, grammars with derivations consisting in "duplications" (more precisely, a word $xuwvy$ is derived to $xwuwvy$ or $xuwvwy$ under certain conditions for w, u, and v) were introduced. Combining the combinatorial, linguistic and biological aspect, it is natural to introduce the duplication language $D(w)$ associated to a word $w \in \Sigma^+$, which is the language containing all words that double some factor of w, i. e., $D(w) = \{xuuy \mid w = xuy, x, y \in \Sigma^*, u \in \Sigma^+\}$ and its iterated version $D^*(w) = \bigcup_{i\geq 0} D^i(w)$. In several papers, the regularity of $D^*(w)$ was discussed; for instance, it was shown that, for any word w over a binary alphabet, $D^*(w)$ is regular and that $D^*(abc)$ is not regular. Further cases of bounded duplication, i. e., the length of the duplicated word is bounded by a constant, were also investigated.

It was noted that words w containing hairpins, i. e., $w = xuyh(u^R)z$, and words w with $w = xuy$ and $u = h(u^R)$, where u^R is the mirror image of u and h is a letter-to-letter isomorphism, are of interest in DNA structures, where the Watson-Crick complementarity gives the isomorphism). Therefore, operations leading to words with hairpins as factors were studied (see [2]).

In this work, we consider the case where the operation leads to words which have palindromes (words with $w = w^R$) as factors (which is a restriction to the identity as the isomorphism). An easy step would be to obtain xuu^Ry from a word xuy in analogy to the duplication. But then all newly obtained palindromes are of even length. Thus it seems to be more interesting to consider the palindrome closure defined by de Luca [3]. Here a word is extended to a palindrome

of minimal length. We allow this operation to be applied to factors and call it inner palindromic closure. We also study the case of iterated applications and a restriction bounding the increase of length.

For more details on basic concepts and definitions see [4].

An alphabet Σ has the cardinality denoted by $\|\Sigma\|$. A sequence of elements of Σ, called letters, constitute a word w, and we denote the empty word by ε.

For $i \geq 0$, the i-fold catenation of a word w with itself is denoted by w^i and is called the ith power of w. When $i = 2$, we call $w^2 = ww$ a square.

For a word $w \in \Sigma^*$, we denote its mirror image (or reversal) by w^R and say that w is a palindrome if $w = w^R$. For a language L, let $L^R = \{w^R \mid w \in L\}$.

2 Results

The following operation from [3] considers extensions of words into palindromes.

Definition 1. *For a word u, the left (right) palindromic closure of u is a word vu (uv) which is a palindrome for some non-empty word v such that any other palindromic word having u as proper suffix (prefix) has length greater than $|uv|$.*

As for duplication and reversal, we can now define a further operation.

Definition 2. *For a word w, the left (right) inner palindromic closure of w is the set of all words $xvuy$ ($xuvy$) for any factorisation $w = xuy$ with possibly empty x, y and non-empty u, v, such that vu (uv) is the left (right) palindromic closure of u. We denote these operations by $\spadesuit_\ell(w)$ and $\spadesuit_r(w)$, respectively, and define the inner palindromic closure $\spadesuit(w)$ as the union of $\spadesuit_\ell(w)$ and $\spadesuit_r(w)$.*

The operation is extended to languages and an iterated version is introduced.

Definition 3. *For a language L, let $\spadesuit(L) = \bigcup_{w \in L} \spadesuit(w)$. We set $\spadesuit^0(L) = L$, $\spadesuit^n(L) = \spadesuit(\spadesuit^{n-1}(L))$ for $n \geq 1$, $\spadesuit^*(L) = \bigcup_{n \geq 0} \spadesuit^n(L)$. Any set $\spadesuit^n(L)$ is called a finite inner palindromic closure of L, and we say that $\spadesuit^*(L)$ is the iterated inner palindromic closure of L.*

For the inner palindromic closure operation on binary alphabets, we get a result similar to that in [1].

Theorem 4. *The iterated inner palindromic closure of a language over a binary alphabet is regular.* □

Obviously, the finite inner palindromic closure of a finite language is always regular. However, when considering the entire class of regular languages the result is not necessarily regular.

Theorem 5. *The finite inner palindromic closure of a regular language is not necessarily regular.* □

It remains an *open problem* whether or not the iterated inner palindromic closure of a regular language L, where $\|\mathrm{alph}(L)\| \geq 3$, is also regular.

Definition 6. *For a word u and integers $m \geq 0$ and $n > 0$, we define the sets $L_{m,n}(w) = \{u \mid u = u^R, u = xw \text{ for } x \neq \varepsilon, |x| \geq n,\ m \geq |w| - |x| \geq 0\}$, $R_{m,n}(w) = \{u \mid u = u^R, u = wx \text{ for } x \neq \varepsilon, |x| \geq n,\ m \geq |w| - |x| \geq 0\}$. The left (right) (m,n)-palindromic closure of w is the shortest word of $L_{m,n}(w)$ (resp., $R_{m,n}(w)$), or undefined if $L_{m,n}(w)$ (resp., $R_{m,n}(w)$) is empty.*

Next we investigate the (m,n)-palindromic closure, for positive integers m, n.

Theorem 7. *There exist infinitely long binary words avoiding both palindromes of length 6 and longer, and squares of words with length 3 and longer.* □

Theorem 8. *There exist infinitely long ternary words avoiding both palindromes of length 3 and longer, and squares of words with length 2 and longer.* □

We associate to a $k \geq 2$ a pair (p_k, q_k) if an infinite k-letter word avoiding palindromes of length $\geq q_k$ and squares of words of length $\geq p_k$ exists.

Theorem 9. *Let $m > 0$ and $k \geq 2$ be two integers and define $n = \max\{\frac{q_k}{2}, p_k\}$. Let Σ be a k-letter alphabet with $a \notin \Sigma$ and $w = a^m y_1 a y_2 \cdots a y_{r-1} a y_r$ be a word such that $alph(w) = \Sigma \cup \{a\}$, $r > 0$, $y_i \in \Sigma^*$ for all $1 \leq i \leq r$, and there exists j with $1 \leq j \leq r$ and $|y_j| \geq n$. Then $\spadesuit^*_{(m,n)}(w)$ is not regular.* □

The following theorem follows immediately from the previous results.

Theorem 10. *Let $w = a^p y_1 a \cdots y_{r-1} a y_r$, where $a \notin alph(y_i)$ for $1 \leq i \leq r$.*
(1) If $\|alph(w)\| \geq 3$ and $|y_j| \geq 3$ for some $1 \leq j \leq r$, then for every positive integer $m \leq p$ we have that $\spadesuit^*_{(m,3)}(w)$ is not regular.
(2) If $\|alph(w)\| \geq 4$ and $|y_j| \geq 2$ for some $1 \leq j \leq r$, then for every positive integer $m \leq p$ we have that $\spadesuit^*_{(m,2)}(w)$ is not regular.
(3) If $\|alph(w)\| \geq 5$, then for every positive integer $m \leq p$ we have that $\spadesuit^*_{(m,1)}(w)$ is not regular.
(4) For every positive integers m and n there exists u with $\spadesuit^*_{(m,n)}(u)$ not regular. □

In general, the regularity of the languages $\spadesuit^*_{(m,n)}(w)$ for positive integers m and n, and binary words w, $|w| \geq n$, is *left open*. We only show the following.

Theorem 11. *For any $w \in \{a,b\}^+$ and integer $m \geq 0$, $\spadesuit^*_{(m,1)}(w)$ is regular.* □

References

1. D. P. Bovet and S. Varricchio. On the regularity of languages on a binary alphabet generated by copying systems. *Inf. Process. Lett.*, 44:119–123, 1992.
2. D. Cheptea, C. Martín-Vide, and V. Mitrana. A new operation on words suggested by DNA biochemistry: Hairpin completion. *Trans. Comput.*, pages 216–228, 2006.
3. A. de Luca. Sturmian words: Structure, combinatorics, and their arithmetics. *Theor. Comput. Sci.*, 183:45–82, 1997.
4. G. Rozenberg and A. Salomaa. *Handbook of Formal Languages.* Springer-Verlag New York, Inc., 1997.

Discovering Hidden Repetitions in Words

Paweł Gawrychowski[1], Florin Manea[2], Dirk Nowotka[2]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany,
gawry@cs.uni.wroc.pl
[2] Christian-Albrechts-Universität zu Kiel, Institut für Informatik, Kiel, Germany,
{flm,dn}@informatik.uni-kiel.de

Abstract. Pseudo-repetitions are a natural generalization of the classical notion of repetitions in sequences: they are the repeated concatenation of a word and its encoding under a certain morphism or antimorphism. We approach the problem of deciding whether there exists an anti-/morphism for which a word is a pseudo-repetition. In other words, we try to discover whether a word has a hidden repetitive structure. We show that some variants of this problem are efficiently solvable, while some others are NP-complete. This manuscript is an abstract of [3].
Keywords: repetition, pseudo-repetition, pattern matching, stringology.

1 Definitions

Let V be a finite alphabet. We denote by V^* the set of all words over V and by V^k the set of all words of length k. The *length* of a word $w \in V^*$ is denoted by $|w|$. The *empty word* is denoted by λ. Moreover, we denote by $\text{alph}(w)$ the alphabet of all letters that occur in w. In the problems discussed in this paper we are given as input a word w of length n and we assume that the letters of w are in fact integers from $\{1, \ldots, n\}$ and w is seen as a sequence of integers. This is a common assumption in algorithmic on words (see, e.g., [4]).

A word u is a *factor* of a word v if $v = xuy$, for some x, y; also, u is a *prefix* of v if $x = \lambda$ and a *suffix* of v if $y = \lambda$. We denote by $w[i]$ the symbol at position i in w and by $w[i..j]$ the factor $w[i]w[i+1]\ldots w[j]$ of w starting at position i and ending at position j. For simplicity, we assume that $w[i..j] = \lambda$ if $i > j$. A word u occurs in w at position i if u is a prefix of $w[i..|w|]$. The powers of a word w are defined recursively by $w^0 = \lambda$ and $w^n = ww^{n-1}$ for $n \geq 1$. If w cannot be expressed as a power of another word, then w is *primitive*. If $w = u^n$ with $n \geq 2$ and u primitive, then u is called the primitive root of w. A *period* of a word w over V is a positive integer p such that $w[i] = w[j]$ for all i and j with $i \equiv j \pmod p$. By $per(w)$ we denote the smallest period of w.

A function $f : V^* \to V^*$ is a morphism if $f(xy) = f(x)f(y)$ for all $x, y \in V^*$; f is an antimorphism if $f(xy) = f(y)f(x)$ for all $x, y \in V^*$. Note that to define an anti-/morphism it is enough to give the definitions of $f(a)$, for all $a \in V$. We say that f is *uniform* if there exists a number k with $f(a) \in V^k$, for all $a \in V$; if $k = 1$ then f is called *literal*. If $f(a) = \lambda$ for some $a \in V$, then f is called *erasing*, otherwise *non-erasing*. The vector T_f of $|V|$ natural numbers with

$T_f[a] = |f(a)|$ is called the length-type of the anti-/morphism f in the following. If $V = \{a_1, \ldots, a_n\}$, T is a vector of n natural numbers $T[a_1], \ldots, T[a_n]$, and $x = b_1 \cdots b_k$ with $b_i \in V$ for all i, we denote by $T(x) = \sum_{i \leq k} T[b_i]$, the length of the image of x under any anti-/morphism of length type T defined on V.

We say that a word w is an *f-repetition*, or, alternatively, an f-power, if w is in $t\{t, f(t)\}^+$, for some prefix t of w; for simplicity, if $w \in t\{t, f(t)\}^+$ then w is called an f-power of root t. If w is not an f-power, then w is *f-primitive*.

For example, the word *abcaab* is primitive from the classical point of view (i.e., $\mathbf{1}$-primitive, where $\mathbf{1}$ is the identical morphism) as well as f-primitive, for the morphism f defined by $f(a) = b$, $f(b) = a$ and $f(c) = c$. However, when considering the morphism $f(a) = c$, $f(b) = a$ and $f(c) = b$, we get that *abcaab* is the concatenation of ab, $ca = f(ab)$, and ab, thus, being an f-repetition.

2 Overview

In [2], an efficient solution for the problem of deciding, given a word w and an anti-/morphism f, whether w is an f-repetition was given. Here we approach a more challenging problem. Namely, we are interested in deciding whether there exists an anti-/morphism f for which a given word w is an f-repetition. Basically, we check whether a given word has an intrinsic (yet hidden) repetitive structure. Note that in the case approached in [2] the main difficulty was to find a prefix x of w such that $w \in x\{x, f(x)\}^*$. The case we discuss here seems more involved: not only we need to find two factors x and y such that $w \in x\{x, y\}^*$, i.e., a suitable decompositions of w, but we also have to decide the existence of an anti-/morphism f with $f(x) = y$. The problem is defined in the following.

Problem 1. Given $w \in V^+$, decide whether there exists an anti-/morphism $f : V^* \to V^*$ and a prefix t of w such that $w \in t\{t, f(t)\}^+$.

The unrestricted version of the problem is, however, trivial. We can always give a positive answer for input words of length greater than 2. It is enough to take the (non-erasing) anti-/morphism f that maps the first letter of w, namely $w[1]$, to $w[2..n]$, where $n = |w|$. Clearly, $w = w[1]f(w[1])$, so w is indeed an f-repetition. When the input word has length 1 or 0, the answer is negative.

On the other hand, when we add a series of simple restrictions to the initial statement, the problem becomes more interesting. The restrictions we define are of two types: either we restrict the desired form of f, and try to find anti-/morphisms of given length type, or we restrict the repetitive structure of w by requiring that it consists in at least three repeating factors or that the root of the pseudo-repetition has length at least 2.

In the first case, when the input consists both in the word w and the length type of the anti-/morphism we are trying to find, we obtain a series of polynomial time solutions for Problem 1. More precisely, in the most general case we can decide whether there exists an anti-/morphism f such that w is an f-repetition in $\mathcal{O}(n(\log n)^2)$ time. Note that deciding whether a word is an f-repetition when f is known took only $\mathcal{O}(n \log n)$ time [2]. When we search for an uniform morphism

we solve the problem in optimal linear time time. This matches the complexity of deciding, for a given uniform anti-/morphism f, whether a given word is an f-repetition, obtained in [2]. This result covers also the case of literal anti-/morphism, extensively approached in the literature (see, e.g., [1, 5]). Our solutions are based both on combinatorial results regarding the structure of pseudo-repetitions and on the usage of efficient data-structures.

For the second kind of restrictions, the length type of f is no longer given. In this case, we want to check, for instance, whether there exist a prefix t and an anti-/morphism f such that w is an f-repetition that consists in the concatenation of at least 3 factors t or $f(t)$. The most general case as well as the case when we add the supplementary restriction that f is non-erasing are NP-complete; the case when f is uniform (but of unknown length type) is tractable. The problem of checking whether there exists a prefix t, with $|t| \geq 2$, and a non-erasing anti-/morphism f such that $w \in t\{t, f(t)\}^+$ is also NP-complete; this problem becomes tractable for erasing or uniform anti-/morphisms.

Our two main theorems are:

Theorem 2. *Given a word w and a vector T of $|V|$ numbers, we decide whether there exists an anti-/morphism f of length type T such that $w \in t\{t, f(t)\}^+$ in $\mathcal{O}(n(\log n)^2)$ time. If T defines uniform anti-/morphisms we need $\mathcal{O}(n)$ time.*

Theorem 3. *For a word $w \in V^+$, deciding the existence of an anti-/morphism $f : V^* \to V^*$ and a prefix t of w such that $w \in t\{t, f(t)\}^+$ with $|t| \geq 2$ (respectively, $w \in t\{t, f(t)\}\{t, f(t)\}^+$) is solvable in linear time (respectively, NP-complete) in the general case, is NP-complete for f non-erasing, and is solvable in $\mathcal{O}(n^2)$ time for f uniform.*

References

1. E. Czeizler, L. Kari, and S. Seki. On a special class of primitive words. *Theoretical Computer Science*, 411:617–630, 2010.
2. P. Gawrychowski, F. Manea, R. Mercas, D. Nowotka, and C. Tiseanu. Finding pseudo-repetitions. In N. Portier and T. Wilke, editors, *STACS*, volume 20 of *LIPIcs*, pages 257–268. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.
3. P. Gawrychowski, F. Manea, and D. Nowotka. Discovering hidden repetitions in words. In P. Bonizzoni, V. Brattka, and B. Löwe, editors, *CiE*, volume 7921 of *Lecture Notes in Computer Science*, pages 210–219. Springer, 2013.
4. J. Kärkkäinen, P. Sanders, and S. Burkhardt. Linear work suffix array construction. *J. ACM*, 53:918–936, 2006.
5. F. Manea, M. Müller, and D. Nowotka. The avoidability of cubes under permutations. In H.-C. Yen and O. H. Ibarra, editors, *Developments in Language Theory*, volume 7410 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 2012.

Unavoidability of Primitive and Palindromic Words

Peter Leupold

Institut für Informatik,
Universität Leipzig,
Leipzig, Germany
Peter.Leupold@web.de

1 Primitive and Palindromic Words

A word is *primitive*, iff it is not a non-trivial (i.e. with exponent one) power of any word. Thus u is primitive, if $u = v^k$ implies $u = v$ and $k = 1$. We denote the language of all primitive words by Q. It is a well-known fact that for every non-empty word w there exists a unique primitive word p such that $w \in p^+$; this primitive word is called the *(primitive) root* of w and we will denote it by $\sqrt{w}$. The unique integer i such that $\sqrt{w}^i = w$ is called the *degree* of w. The notion of root is extended to languages in the canonical way such that $\sqrt{L} := \bigcup_{w \in L} \sqrt{w}$.

Primitive Words play an important role in the Theory of codes [1]. The most important open question concerning primitive words is, whether the language of all primitive words is context-free. This question was first raised by Dömösi, Horváth and Ito [2], and has so far resisted all attempts to answer it.

A second concept central to the work presented here is *palindromicity*. First off, for a word w by w^R we denote its reversal, that is $w[|w| \ldots 1]$. If $w = w^R$, the word is called a palindrome. For words of even length, this means there is some word u such that $w = uuR$; these are called *even* palindromes. On the other hand *odd* palindromes are of odd length, and they are of the form $w = uxuR$ with a letter x at their center. The set of all palindromes of a language L is denoted by $\mathcal{P}al(L) = \mathcal{P}al \cap L$.

2 Unavoidable Languages

Unavoidability of languages formalizes the following intuitive concept: if a language L shares some words with every language from a given class $\mathcal{C}$, then it is unavoidable in $\mathcal{C}$, because parts of it appear in some sense everywhere. Depending on the size of these parts we define also a strong version of unavoidability.

Definition 1 [5] A language $U \subseteq \Sigma^*$ is called *unavoidable* in the language class $\mathcal{C}$, iff $U \cap L \neq \emptyset$ for all infinite languages $L \in \mathcal{C}$. U is *strongly unavoidable*, iff $U \cap L$ is infinite for all infinite languages $L \in \mathcal{C}$.

Notice that this concept is different from unavoidable sets or languages as they are used in Combinatorics on Words [6]; there, a set of words U is unavoidable, if there exists an integer k such that every word longer than k must have a word from U (or a morphic image of such a word) as a factor. Thus unavoidability is an absolute property of languages, not one relative to a language class as in our case. A further difference is that we demand that words of U be elements of all languages in $\mathcal{C}$, and not just that they occur as factors.

Trivially, Σ^* is strongly unavoidable for all possible language classes over the alphabet Σ, because it has an infinite intersection with any infinite language. Two less trivial examples can be derived from the Pumping Lemmata for regular and context-free languages.

Example 2 Let L_{sq} be the language of all words that contain a square. From the two Pumping Lemmata we can see that every infinite regular language has a subset of the form $w_1 w_2 w_2^+ w_3$ and that every infinite context-free language has a subset of the form $\{w_1 w_2^i w_3 w_4^i w_5 : i \geq 2\}$. Both sets contain only words with squares and are thus infinite subsets of L_{sq}. Thus the latter is strongly unavoidable for regular and context-free languages.

□

The central result from the first work on unavoidability is the following:

Theorem 3 *[5] The language of primitive words is strongly unavoidable for $CF \setminus LIN$.*

Like Example 2 this follows from the pumping properties, but in a much less direct manner. Basically one needs to use the fact that pumping a factor of a word either produces powers of the same word or infintely many primitive words, see for example the results of Shyr and Yu [8], Kászonyi and Katsura [4] or Păun et al. [7].

When we additionally require palindromicity of words a careful analysis leads to the following result:

Theorem 4 *[3] The language $Q^{(2)}$ is strongly unavoidable for palindromic context-free languages that are not regular and only have finitely many primitive words.*

Unfortunately, these results have not yet helped to get closer to a solution for the question about the context-freeness of Q. For example, the avoidability of primitive words for non-deterministic and/or inherently ambiguous context-free languages would provide a negative answer.

References

1. J. Berstel and D. Perrin. *Theory of Codes.* Academic Press, Orlando, 1985.
2. P. Dömösi, S. Horváth, and M. Ito. On the connection between formal languages and primitive words. *Analele Univ. din Oradea, Fasc. Mat.*, pages 59–67, 1991.

3. S. Z. Fazekas, P. Leupold, and K. Shikishima-Tsuji. On non-primitive palindromic context-free languages. *International Journal of Foundations of Computer Science*, 23(6):1277–1290, 2012.
4. L. Kászonyi and M. Katsura. On the context-freeness of a class of primitive words. *Publicationes Mathematicae Debrecen*, 51:1–11, 1997.
5. P. Leupold. Primitive words are unavoidable for context-free languages. In A. H. Dediu, H. Fernau, and C. Martín-Vide, editors, *LATA*, volume 6031 of *Lecture Notes in Computer Science*, pages 403–413. Springer, 2010.
6. M. Lothaire. *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics and Its Applications*. Addison-Wesley, Reading, Massachusetts, 1983.
7. G. Păun, N. Santean, G. Thierrin, and S. Yu. On the robustness of primitive words. *Discrete Applied Mathematics*, 117:239–252, 2002.
8. H. Shyr and S. Yu. Non-primitive words in the language p^+q^+. *Soochow J. Math.*, 4, 1994.

Size of Unary One-Way Multi-Head Finite Automata

Martin Kutrib, Andreas Malcher, and Matthias Wendlandt

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

1 Introduction

One of the main topics of descriptional complexity is the question of how the size of the description of a formal language varies when being described by different formalisms. A fundamental result is the exponential trade-off between the number of states of nondeterministic (NFA) and deterministic finite automata (DFA) (see, for example, [12]). Additional exponential and double-exponential trade-offs are known, for example, between unambiguous and deterministic finite automata, between alternating and deterministic finite automata, between deterministic pushdown automata and DFA, and between the complement of a regular expression and conventional regular expressions. Beside these *recursive* trade-offs, bounded by recursive functions, it is known that there also *non-recursive* trade-offs, which are not bounded by any recursive function. Such trade-offs have at first been shown to exists between context-free grammars generating regular languages and finite automata [12]. For a survey on recursive and non-recursive trade-offs we refer to [3, 5].

Unary languages, that is, languages defined over a singleton alphabet, are of particular interest, since in this case often better or more precise results than in the case of arbitrary alphabets can be obtained. For example, the trade-off of 2^n between an n-state NFA and DFA, is reduced to $e^{\Theta(\sqrt{n \cdot \ln(n)})}$ in the unary case [1]. The descriptional complexity of unary regular languages has been studied in many ways. On the one hand, many automata models such as one-way finite automata, two-way finite automata, pushdown automata, or context-free grammars for unary languages are investigated and compared to each other with respect to simulation results and the size of the simulation (see, for example, [2, 11, 13, 15]). On the other hand, many results concerning the state complexity of operations on unary languages have been obtained (see, for example, [4, 7, 10, 14]).

Here, we consider deterministic one-way multi-head finite automata accepting unary languages. Since it is known that every unary language accepted by a one-way multi-head finite automaton is semilinear and thus regular [6, 16], it is of interest to investigate the descriptional complexity of such devices in comparison with the models mentioned above. In detail, we establish upper and lower bounds for the conversion of k-head DFA to one-head DFA and one-head

NFA. Moreover, we investigate the size costs for simulating one-head NFA by k-head DFA and the computational complexity of decidability questions for k-head DFA. Unary deterministic one-way multi-head finite automata have already been studied in [9]. The main results obtained there are infinite proper hierarchies with respect to the number of states as well as to the number of heads. It should be noted that the trade-offs between general k-head DFA and one-head DFA are non-recursive for all $k \geq 2$ [8].

2 Results

As is often the case in connection with unary languages, the function

$$F(n) = \max\{\, \mathrm{lcm}(c_1, c_2 \ldots, c_l) \mid c_1, c_2, \ldots, c_l \geq 1 \text{ and } c_1 + c_2 + \cdots + c_l = n \,\},$$

which gives the maximal order of the cyclic subgroups of the symmetric group of n symbols, plays a crucial role, where lcm denotes the *least common multiple*.

Theorem 1. *For any integers $k, n \geq 2$ so that n is prime, there is a unary n-state DFA(k) M, such that $n \cdot F(n)^{k-1}$ states are necessary for any DFA to accept the language $L(M)$.*

Theorem 2. *Let $k, n \geq 1$ and M be a unary n-state DFA(k). Then there is a constant t depending only on k so that $O(n \cdot F(t \cdot n)^{k-1})$ states are sufficient for a DFA to accept the language $L(M)$. The DFA can effectively be constructed from M.*

Theorem 3. *Let $k, n \geq 2$ be constants and M be a unary n-state DFA(k). Then $O(n^{2k})$ states are sufficient for an NFA to accept the language $L(M)$. The NFA can effectively be constructed from M.*

Theorem 4. *For any integers $k, n \geq 2$, there is a unary n-state DFA(k) M, such that $\Omega(n^k)$ states are necessary for any NFA to accept the language $L(M)$.*

Theorem 5. *Let $k \geq 1$, $n \geq 2$ be constants, $t = \lfloor \frac{-3+\sqrt{8n+1}}{2} \rfloor$, and M be a unary n-state NFA. Then*

$$n' \leq \begin{cases} n^2 - 2 + F(n), & \text{if } k = 1; \\ n^2 - 2 + \left(n - \frac{t^2+t}{2}\right)^{\lceil \frac{t}{k} \rceil}, & \text{if } 1 < k < t/2; \\ 2n^2, & \text{if } k \geq t/2. \end{cases}$$

states are sufficient for a DFA(k) to accept the language $L(M)$. The DFA(k) can effectively be constructed from M.

Theorem 6. *Let $k \geq 1$ be a constant. For any integer $m \geq 1$ there is an integer $n > m$ and a unary n-state NFA M, such that $c_2 \cdot \sqrt[k]{e^{\frac{\sqrt{2n}}{\sqrt{c_1 \ln(\sqrt{2n})}}}}$ states are necessary for any DFA(k) to accept the language $L(M)$, where $c_1, c_2 > 0$ are two constants.*

Lemma 7. *Let $k \geq 1$ and M be an n-state DFA(k). Then there exists an n-state DFA(k) M' accepting the complement of $L(M)$. The DFA(k) M' can effectively be constructed from M.*

Theorem 8. *Let $k \geq 1$ be an integer. Then the problems to decide emptiness, universality, finiteness, inclusion, and equivalence for unary DFA(k) are* LOGSPACE*-complete.*

References

1. Chrobak, M.: Finite automata and unary languages. Theoret. Comput. Sci. 47, 149–158 (1986)
2. Geffert, V., Mereghetti, C., Pighizzini, G.: Converting two-way nondeterministic unary automata into simpler automata. Theoret. Comput. Sci. 295, 189–203 (2003)
3. Goldstine, J., Kappes, M., Kintala, C.M.R., Leung, H., Malcher, A., Wotschke, D.: Descriptional complexity of machines with limited resources. J. UCS 8, 193–234 (2002)
4. Holzer, M., Kutrib, M.: Unary language operations and their nondeterministic state complexity. In: Developments in Language Theory (DLT 2002). LNCS, vol. 2450, pp. 162–172. Springer (2003)
5. Holzer, M., Kutrib, M.: Descriptional complexity – An introductory survey. In: Scientific Applications of Language Methods, pp. 1–58. Imperial College Press (2010)
6. Ibarra, O.H.: A note on semilinear sets and bounded-reversal multihead pushdown automata. Inform. Process. Lett. 3, 25–28 (1974)
7. Kunc, M., Okhotin, A.: State complexity of operations on two-way finite automata over a unary alphabet. Theoret. Comput. Sci. 449, 106–118 (2012)
8. Kutrib, M.: The phenomenon of non-recursive trade-offs. Int. J. Found. Comput. Sci. 16, 957–973 (2005)
9. Kutrib, M., Malcher, A., Wendlandt, M.: States and heads do count for unary multi-head finite automata. In: Developments in Language Theory (DLT 2012). LNCS, vol. 7410, pp. 214–225. Springer (2012)
10. Mera, F., Pighizzini, G.: Complementing unary nondeterministic automata. Theoret. Comput. Sci. 330, 349–360 (2005)
11. Mereghetti, C., Pighizzini, G.: Optimal simulations between unary automata. SIAM J. Comput. 30, 1976–1992 (2001)
12. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: Symposium on Switching and Automata Theory (SWAT 1971). pp. 188–191. IEEE (1971)
13. Pighizzini, G.: Deterministic pushdown automata and unary languages. Int. J. Found. Comput. Sci. 20, 629–645 (2009)
14. Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthal's function. Int. J. Found. Comput. Sci. 13, 145–159 (2002)
15. Pighizzini, G., Shallit, J., Wang, M.W.: Unary context-free grammars and pushdown automata, descriptional complexity and auxiliary space lower bounds. J. Comput. System Sci. 65, 393–414 (2002)
16. Sudborough, I.H.: Bounded-reversal multihead finite automata languages. Inform. Control 25, 317–328 (1974)