Symeon Bozapalidis
George Rahonis (Eds.)

LNCS 5725

Algebraic Informatics

Third International Conference, CAI 2009
Thessaloniki, Greece, May 2009
Proceedings

Springer

Symeon Bozapalidis George Rahonis (Eds.)

Algebraic Informatics

Third International Conference, CAI 2009
Thessaloniki, Greece, May 19-22, 2009
Proceedings

Springer

Volume Editors

Symeon Bozapalidis
George Rahonis
Aristotle University of Thessaloniki
Department of Mathematics
54124 Thessaloniki, Greece
E-mail:{bozapali, grahonis}@math.auth.gr

Library of Congress Control Number: Applied for

CR Subject Classification (1998): F.4, I.1.3, F.1.1, F.4.1, F.4.3, F.4.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-642-03563-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-03563-0 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12735013 06/3180 5 4 3 2 1 0

Preface

CAI 2009 was the Third International Conference on Algebraic Informatics. It was intended to cover the topics of algebraic semantics on graphs and trees, formal power series, syntactic objects, algebraic picture processing, finite and infinite computations, acceptors and transducers for strings, trees, graphs, arrays, etc., decision problems, algebraic characterization of logical theories, process algebra, algebraic algorithms, algebraic coding theory, algebraic aspects of cryptography.

CAI 2009 was dedicated to Werner Kuich on the occasion of his retirement. It was held in Thessaloniki, Greece, during May 19-22, 2009 and organized under the auspices of the Department of Mathematics of the Aristotle University of Thessaloniki. The opening lecture was given by Werner Kuich, the tutorials by Alessandra Cherubini and Wan Fokkink, and the other four invited lectures by Bruno Courcelle, Dietrich Kuske, Detlef Plump, and Franz Winkler. This volume contains 2 papers from the tutorials, 5 papers of the invited lectures, and 16 contributed papers. We received 25 submissions, the contributors being from 14 and countries, and the Program Committee selected 16 papers.

We are grateful to the members of the Program Committee for the evaluation of the submissions and the numerous referees who assisted in this work. We should like to thank all the contributors of CAI 2009 and especially the honorary guest Werner Kuich and the invited speakers who kindly accepted our invitation to present their important work. Special thanks are due to Alfred Hofmann the Editorial Director of LNCS, who gave us the opportunity to publish the proceedings of our conference in the LNCS series, as well as to Anna Kramer from Springer for the excellent cooperation. We are also grateful to the members of the Organizing Committee and a group of graduate students who helped us with several organizing jobs. Last but not least we want to express our gratitude to the members of the Steering Committee for their constant interest and especially to Arto Salomaa for his support at Springer.

The sponsors of CAI 2009, OPAP, Aristotle University of Thessaloniki, Attiko Metro S.A., Research Academic Computer Technology Institute (Fronts), and Ziti Publications are gratefully acknowledged.

July 2009

Symeon Bozapalidis
George Rahonis

Organization

Steering Committee

Jean Berstel, Marne-la-Vallée
Zoltan Ésik, Szeged
Werner Kuich, Vienna
Arto Salomaa, Turku

Program Committee

Jürgen Albert, Würzburg
Jos Baeten, Eindhoven
Symeon Bozapalidis, Thessaloniki (Chairman)
Flavio Corradini, Camerino
Erzsébet Csuhaj-Varjú, Budapest
Frank Drewes, Umeå
Manfred Droste, Leipzig
Ioannis Emiris, Athens
Dora Giammarresi, Rome
Masami Ito, Kyoto
Friedrich Otto, Kassel
Dimitrios Poulakis, Thessaloniki
Robert Rolland, Marseille
Kai Salomaa, Kingston Ontario
Paul Spirakis, Patras
Magnus Steinby, Turku
Sophie Tison, Lille
Heiko Vogler, Dresden
Sheng Yu, London Ontario

Referees

J. Albert
Y. Aubry
J. Baeten
J. Berstel
S. Bloom
S. Bozapalidis
M. Ćirić
F. Corradini
B. Courcelle
E. Csuhaj-Varjú
M. Domaratzki
F. Drewes
I. Emiris
Z. Fülöp
D. Giammarresi
A. Grammatikopoulou
J. Högberg
S. Jenei

H. Jonker
A. Kalampakas
D. Kuske
A. Lopes
A. Maletti
E. Mandrali
O. Matz
I. Meinecke
M. Mignotte
K. Ogata
F. Otto
A. Papistas
U. Prange
M. Pohst
D. Poulakis
R. Rabinovich
G. Rahonis
R. Rolland
Y. Roos
K. Salomaa
P. Spirakis
M. Steinby
S. Tison
N. Tzanakis
G. Vaszil
H. Vogler
S. Yu
S.-S. Yu

Organizing Committee

Archontia Grammatikopoulou
Antonios Kalampakas
Eleni Mandrali
Athanasios Papistas
Dimitrios Poulakis (Co-chairman)
George Rahonis (Chairman)

Sponsors

OPAP
Aristotle University of Thessaloniki
Attiko Metro S.A.
Research Academic Computer Technology Institute (Fronts)
Ziti Publications.

Table of Contents

Cycle-Free Finite Automata in Partial Iterative Semirings

Stephen L. Bloom[1], Zoltan Ésik[2,⋆], and Werner Kuich[3,⋆⋆]

[1] Dept. of Computer Science
Stevens Institute of Technology
Hoboken, NJ. USA

[2] Dept. of Computer Science
University of Szeged
Hungary

[3] Institut für Diskrete Mathematik und Geometrie
Technische Universität Wien
Austria

Abstract. We consider partial Conway semirings and partial iteration semirings, both introduced by Bloom, Ésik, Kuich [2]. We develop a theory of cycle-free elements in partial iterative semirings that allows us to define cycle-free finite automata in partial iterative semirings and to prove a Kleene Theorem. We apply these results to power series over a graded monoid with discounting.

1 Introduction

Cycle-free power series $r \in S\langle\langle \Sigma^* \rangle\rangle$, where S is a semiring and Σ is an alphabet, are defined by the condition that (r, ε), the coefficient of r at the empty word ε, is nilpotent. Transferring this notion via its transition matrix to a finite automaton assures that the behavior of a cycle-free finite automaton is well defined. This fact makes it possible to generalize classical finite automata with ε-moves to weighted cycle-free finite automata (see Kuich, Salomaa [11], Ésik, Kuich [9]).

In this paper, we take an additional step of generalization. We consider cycle-free elements in a partial iterative semiring and consider cycle-free finite automata. This generalization preserves all the nice results of weighted cycle-free finite automata and allows us to prove the usual Kleene Theorem stating the coincidence of the sets of recognizable and rational elements.

This paper consists of this and three more sections. In Section 2 we consider partial iterative semirings and partial Conway semirings, both introduced by Bloom, Ésik, Kuich [9]. Moreover, we define cycle-free elements in partial iterative semirings and prove several identities involving these cycle-free elements. In Section 3 we introduce cycle-free finite automata in partial iterative semirings,

⋆ Partially supported by grant no. K 75249 from the National Foundation of Scientific Research of Hungary, and by Stiftung Aktion Österreich-Ungarn.

⋆⋆ Partially supported by Stiftung Aktion Österreich-Ungarn.

define recognizable and rational elements and prove a Kleene Theorem: an element is recognizable iff it is rational. In Section 4 we apply the results to power series over a finitely generated graded monoid with discounting.

2 Cycle-Free Elements in Partial Iterative Semirings

Suppose that S is a semiring and I is an ideal of S, so that $0 \in I$, $I + I \subseteq I$ and $IS \cup SI \subseteq I$. According to Bloom, Ésik, Kuich [2], S is a *partial iterative semiring over* I if for all $a \in I$ and $b \in S$ the equation $x = ax + b$ has a unique solution in S. We denote this unique solution by a^*b.

Example. This is a running example for the whole paper. Let S be a semiring and Σ an alphabet, and consider the power series semiring $S\langle\langle \Sigma^* \rangle\rangle$. A power series $r \in S\langle\langle \Sigma^* \rangle\rangle$ is called *proper* if $(r, \varepsilon) = 0$. Clearly, the collection of proper power series forms an ideal $I = \{r \in S\langle\langle \Sigma^* \rangle\rangle \mid (r, \varepsilon) = 0\}$. By Theorem 5.1 of Droste, Kuich [4], $S\langle\langle \Sigma^* \rangle\rangle$ is a partial iterative semiring over the ideal I, where the * of a proper power series r is defined by $r^* = \sum_{j \geq 0} r^j$. □

In the rest of this section we suppose that S is a partial iterative semiring over I. Moreover, we let J denote the set of all $a \in S$ such that $a^k \in I$ for some $k \geq 1$. Note that if $a^k \in I$ then $a^m \in I$ for all $m \geq k$. When a^k is in I, we say that a is *cycle free with index* k. We clearly have $I \subseteq J$.

Proposition 1. *If $a \in I$ and $b \in J$ then $a + b \in J$. Moreover, if $a, b \in S$ with $ab \in J$ then $ba \in J$.*

Proof. If $a \in I$ and $b \in J$ with $b^k \in I$, then $(a + b)^k$ is a sum of terms which are k-fold products over $\{a, b\}$. Each such product is in I since it is either b^k or contains a as a factor. Since I is closed under sum, it follows that $(a + b)^k$ is in I and thus $a + b$ is in J.

Suppose now that $a, b \in S$ with $(ab)^k \in I$ for some $k \geq 1$. Then $(ba)^{k+1} = b(ab)^k b \in I$, proving that $ba \in J$. □

The following fact was shown in Bloom, Ésik, Kuich [2].

Proposition 2. *Suppose that $a \in J$ and $b \in S$. Then the equation $x = ax + b$ has a unique solution. Moreover, its unique solution is a^*b, where a^* is the unique solution of the equation $x = ax + 1$.*

Thus, we have a partial *-operation $S \to S$ defined on the set J of cycle-free elements.

Proposition 3. *Suppose that $a, b \in J$ and $c \in S$. If $ac = cb$, then $a^*c = cb^*$. In particular, $a(a^m)^* = (a^m)^*a$, for all $a \in J$ and $m \geq 1$.*

Proof. We have $acb^* + c = cbb^* + c = c(bb^* + 1) = cb^*$, so that $a^*c = cb^*$ by uniqueness. □

Proposition 4. *Suppose that $a, b \in S$ such that $ab \in J$. Then $ba \in J$, moreover, $(ab)^*a = a(ba)^*$ and $a(ba)^*b + 1 = (ab)^*$.*

Proof. Since $(ab)a = a(ba)$ and $ab, ba \in J$, we can apply Proposition [3] to get $(ab)^*a = a(ba)^*$. Using this, $a(ba)^*b + 1 = ab(ab)^* + 1 = (ab)^*$. □

Proposition 5. *Suppose that $a, b \in S$ such that $a, a+b$ and a^*b are all in J. Then $(a+b)^* = (a^*b)^*a^*$.*

Proof. We show that $(a^*b)^*a^*$ is a solution to the equation $x = (a+b)x + 1$:

$$\begin{aligned}(a+b)(a^*b)^*a^* + 1 &= a(a^*b)a^* + b(a^*b)a^* + 1\\ &= aa^*(ba^*)^* + (ba^*)^*\\ &= (aa^*+1)(ba^*)^*\\ &= a^*(ba^*)^*\\ &= (a^*b)^*a^*.\end{aligned}$$

□

Corollary 1. *If $a \in J$ and $b \in I$ then $(a+b)^* = (a^*b)^*a^*$.*

Proposition 6. *If $a \in J$ then $a^m \in J$ for all $m \geq 1$ and $a^* = (a^m)^*(a^{m-1} + \ldots + 1) = (a^{m-1} + \ldots + 1)(a^m)^*$.*

Proof. The fact that $(a^m)^*(a^{m-1}+\ldots+1) = (a^{m-1}+\ldots+1)(a^m)^*$ follows from Proposition 3. The fact that $a^* = (a^{m-1} + \ldots + 1)(a^m)^*$ follows by noting that

$$\begin{aligned}a(a^{m-1} + \ldots + 1)(a^m)^* + 1 &= a^m(a^m)^* + 1 + (a^{m-1} + \ldots + a)(a^m)^*\\ &= (a^m)^* + (a^{m-1} + \ldots + a)(a^m)^*\\ &= (a^{m-1} + \ldots + 1)(a^m)^*.\end{aligned}$$

□

The following fact is from Bloom, Ésik, Kuich [2].

Proposition 7. *If S is a partial iterative semiring over I, then $S^{n\times n}$ is a partial iterative semiring over $I^{n\times n}$.*

Below we will consider fixed point equations $X = AX + B$, where $A \in S^{n\times n}$ and $B \in S^{n\times m}$. We will assume that A and B are partitioned as

$$A = \begin{pmatrix} a & b\\ c & d\end{pmatrix} \quad \text{and} \quad \begin{pmatrix} e\\ f\end{pmatrix}$$

where $a \in S^{n_1\times n_2}$, $b \in S^{n_1\times n_2}$, $c \in S^{n_2\times n_1}$, $d \in S^{n_2\times n_2}$, $e \in S^{n_1\times m}$, $f \in S^{n_2\times m}$.

Corollary 2. *If A is cycle-free so that $A^k \in I^{n\times n}$ for some k, then the equation $X = AX + B$ has a unique solution.*

Again, this unique solution is A^*B, where A^* is the unique solution to the equation $X = AX + E_n$, where E_n denotes the unit matrix in $S^{n\times n}$.

Proposition 8. *Let $A \in S^{n\times n}$ be cycle-free and assume that $a, a + bd^*c, d, d + ca^*b$ are all cycle-free. Then*

$$A^* = \begin{pmatrix} (a+bd^*c)^* & (a+bd^*c)^*bd^*\\ (d+ca^*b)^*ca^* & (d+ca^*b)^*\end{pmatrix}. \qquad (1)$$

Proof. Consider the system of fixed point equations

$$x = ax + by + e \tag{2}$$

$$y = cx + dy + f \tag{3}$$

where x ranges over $S^{n_1 \times m}$ and y ranges over $S^{n_2 \times m}$. We show that it has a unique solution

$$x = (a + bd^*c)^*(e + bd^*f) \tag{4}$$

$$y = (d + ca^*b)^*(f + ca^*e) \tag{5}$$

Since a is cycle free, from (2) we have $x = a^*by + a^*e$. Substituting this for x in (3) gives

$$y = (d + ca^*b)y + ca^*e + f$$

Since $d + ca^*b$ is cycle-free, this gives (5). The proof of (4) is similar. □

Proposition 9. *Let $A \in S^{n\times n}$ and assume that a and d are cycle-free and $b \in I^{n_1\times n_2}$ or $c \in I^{n_2\times n_1}$. Then A is cycle-free and (1) holds.*

Proof. We only prove the case where $c \in I^{n_2\times n_1}$. The proof of the other case is similar. It is clear that for each $j \geq 1$,

$$A^j = \begin{pmatrix} a^j + x & y + z \\ u & d^j + v \end{pmatrix},$$

where the entries of x, y, u, v are all in I since they are finite sums of j-fold products containing at least one occurrence of the factor c. Moreover, z is a sum of j-fold products over $\{a, b, d\}$ having a single factor equal to b. Since a and d are cycle-free, it follows that for large enough j each such product is also a matrix with entries in I, so that each entry of z is in I. We have thus proved that when j is sufficiently large, then $A^j \in I^{n\times n}$ so that A^* is defined. Also, for each $j \geq 1$, $(a + bd^*c)^j = a^j + x$ and $(d + ca^*b)^j = d^j + y$ where x, y are matrices with entries in I. Since a and d are cycle-free, it follows again that when j is sufficiently large, then the entries of $(a + bd^*c)^j$ and $(d + ca^*b)^j$ are all in I, so that $a + bd^*c$ and $d + ca^*b$ are cycle-free and $(a + bd^*c)^*$ and $(d + ca^*b)^*$ exist. Thus, the assumptions of Proposition 8 are satisfied and our proposition is proved. □

Corollary 3. *Let $A \in S^{n\times n}$ and assume that a and d are cycle-free and $c = 0$. Then A is cycle-free and*

$$A^* = \begin{pmatrix} a^* & a^*bd^* \\ 0 & d^* \end{pmatrix}.$$

In Bloom, Ésik, Kuich [2], a *partial Conway semiring* is defined as a semiring S equipped with a distinguished ideal I and a partial operation $^* : S \to S$ defined on I which satisfies the *sum *-identiy*

$$(a + b)^* = (a^*b)^*a^*$$

for all $a, b \in I$ and *product *-identity*

$$(ab)^* = 1 + a(ba)^*b$$

for all $a, b \in S$ with $a \in I$ or $b \in I$. By Propositions 4 and 5 we have that each partial iterative semiring is a partial Conway semiring. It is known that when S is a partial Conway semiring with distinguished ideal I, then for each n, $S^{n\times n}$ is also a partial Conway semiring equipped with the ideal $I^{n\times n}$. Moreover, (1) holds for all decompositions of a matrix $A \in I^{n\times n}$. A *Conway semiring* (see Conway [3] and Bloom, Ésik [1]) is a partial Conway semiring S whose distinguished ideal is S, so that the *-operation is completely defined.

3 Cycle-Free Finite Automata

In this section we establish a Kleene Theorem in partial iterative semirings. To this end, we define a general notion of cycle-free finite automaton in partial iterative semirings. Defining the set of recognizable elements to be the set of behaviors of cycle-free finite automata, and the set of rational elements to be the least partial iterative semiring generated by some particular elements, the Kleene Theorem states that an element is recognizable iff it is rational.

In this section, S is a partial iterative semiring over the ideal I of S, Σ is a subset of I, and S_0 is a subsemiring of S. Moreover, $S_0\Sigma$ denotes the set of all finite linear combinations over Σ with coefficients in S_0, and $S_0 + S_0\Sigma$ denotes the set of sums of elements of S_0 with elements of $S_0\Sigma$. (See Bloom, Ésik, Kuich [2], Section 6.)

A *finite automaton in S and I over* (S_0, Σ) $\mathbf{A} = (\alpha, A, \beta)$ is given by

(i) a *transition matrix* $A \in (S_0 + S_0\Sigma)^{n\times n}$,
(ii) an *initial vector* $\alpha \in S_0^{1\times n}$,
(iii) a *final vector* $\beta \in S_0^{n\times 1}$.

The integer $n \geq 1$ is called the *dimension* of $\mathbf{A}$. Briefly, we call $\mathbf{A}$ finite automaton if S, I, S_0, Σ are understood.

The finite automaton $\mathbf{A} = (\alpha, A, \beta)$ is called *cycle-free* if A is cycle-free over $I^{n\times n}$. The *behavior* $|\mathbf{A}|$ of such a cycle-free finite automaton $\mathbf{A}$ is given by

$$|\mathbf{A}| = \alpha A^* \beta\,.$$

We say that $a \in S$ is *recognizable* if a is the behavior of some cycle-free finite automaton in S and I over (S_0, Σ). We let $\mathbf{Rec}_{S,I}(S_0, \Sigma)$ denote the set of all elements of S which are recognizable.

We say that $a \in S$ is *rational* if it is contained in the partial iterative semiring $\mathbf{Rat}_{S,I}(S_0, \Sigma)$ over $\mathbf{Rat}_{S,I}(S_0, \Sigma) \cap I$ generated by $S_0 \cup \Sigma$; i. e., if it is contained in the least set containing $S_0 \cup \Sigma$ and closed under the *rational operations* $+, \cdot, ^*$, where * is applied only to elements of I.

Observe that $\mathbf{Rat}_{S,I}(S_0, \Sigma)$ may be defined in an equivalent way as follows, due to Proposition 6: $\mathbf{Rat}_{S,I}(S_0, \Sigma)$ is the least set containing $S_0 \cup \Sigma$ which is closed under the operations $+, \cdot, ^*$, where * is applied only to cycle-free elements.

We will show that under a certain additional condition on S_0, $\mathbf{Rec}_{S,I}(S_0, \Sigma) = \mathbf{Rat}_{S,I}(S_0, \Sigma)$.

Example. We let S_0 be the subsemiring $S\langle\{\varepsilon\}\rangle = \{a\varepsilon \mid a \in S\}$ of $S\langle\langle\Sigma^*\rangle\rangle$. Then the finite automata in Subsection 2.1 of Ésik, Kuich [9] are essentially the finite automata in $S\langle\langle \Sigma^* \rangle\rangle$ and I over $(S\langle\{\varepsilon\}\rangle, \Sigma)$, where I is the ideal of proper series. (See Theorem 2.1 of Ésik, Kuich [9].)

The sets $S^{\mathrm{rec}}\langle\langle \Sigma^* \rangle\rangle$ and $S^{\mathrm{rat}}\langle\langle \Sigma^* \rangle\rangle$ in Ésik, Kuich [9] are then the specializations of the sets of recognizable and rational elements of $S\langle\langle \Sigma^* \rangle\rangle$, respectively; i. e., $S^{\mathrm{rec}}\langle\langle \Sigma^* \rangle\rangle = \mathbf{Rec}_{S\langle\langle \Sigma^* \rangle\rangle, I}(S\langle\{\varepsilon\}\rangle, \Sigma)$ and $S^{\mathrm{rat}}\langle\langle \Sigma^* \rangle\rangle = \mathbf{Rat}_{S\langle\langle \Sigma^* \rangle\rangle, I}(S\langle\{\varepsilon\}\rangle, \Sigma)$. Then $\mathbf{Rec}_{S,I}(S_0, \Sigma) = \mathbf{Rat}_{S,I}(S_0, \Sigma)$ is the Kleene-Schützenberger Theorem, usually written as $S^{\mathrm{rec}}\langle\langle \Sigma^* \rangle\rangle = S^{\mathrm{rat}}\langle\langle \Sigma^* \rangle\rangle$, and the theory of cycle-free finite automata developed in this section is a generalization of Subsection 2.1 of Ésik, Kuich [9]. □

Two cycle-free finite automata $\mathbf{A}$ and $\mathbf{A}'$ are *equivalent* if $|\mathbf{A}| = |\mathbf{A}'|$. A finite automaton $\mathbf{A} = (\alpha, A, \beta)$ of dimension n is called *normalized* if $n \geq 2$ and

(i) $\alpha_1 = 1$, $\alpha_i = 0$, for all $2 \leq i \leq n$;
(ii) $\beta_n = 1$, $\beta_i = 0$, for all $1 \leq i \leq n-1$;
(iii) $A_{i,1} = A_{n,i} = 0$, for all $1 \leq i \leq n$.

(See also Ésik, Kuich [9], below Theorem 2.9.)

Proposition 10. *Each cycle-free finite automaton is equivalent to a normalized cycle-free finite automaton.*

Proof. Let $\mathbf{A} = (\alpha, A, \beta)$ be a cycle-free finite automaton of dimension n. Define the finite automaton

$$\mathbf{A}' = ((1\ 0\ 0), \begin{pmatrix} 0 & \alpha & 0 \\ 0 & A & \beta \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix})$$

of dimension $n+2$. Then $\mathbf{A}'$ is normalized. Applying Corollary 3 twice on the transition matrix of $\mathbf{A}'$ proves that $\mathbf{A}'$ is cycle-free and

$$|\mathbf{A}'| = (\begin{pmatrix} 0 & \alpha & 0 \\ 0 & A & \beta \\ 0 & 0 & 0 \end{pmatrix}^*)_{1,n+2} = \alpha A^* \beta = |\mathbf{A}|\,.$$

□

We now show that, under an additional condition on S_0, each cycle-free finite automaton is equivalent to one where the entries of the transition matrix are in the ideal I. (See condition (23) in Section 6 of Bloom, Ésik, Kuich [2].)

Definition 1. *Suppose that S is a partial iterative semiring over the ideal I, S_0 is a subsemiring of S. We say (S, S_0, I)* ***is cycle-free*** *if for all $a \in S_0$ and all $b \in I$, if*

$$a + b \in I$$

then $a = 0$.

Thus, when (S, S_0, I) is cycle-free, we understand that S, S_0, I satisfy the assumptions of Definition 1.

Proposition 11. *Suppose (S, S_0, I) is cycle-free and $\Sigma \subseteq I$. Then each cycle-free finite automaton in S and I over (S_0, Σ) is equivalent to a cycle-free automaton $\mathbf{A}' = (\alpha', A', \beta')$ in S and I over (S_0, Σ), where $A' \in (S_0\Sigma)^{n\times n}$, and $\alpha'_1 = 1$, $\alpha'_i = 0$ for all $2 \le i \le n$.*

Proof. For each cycle-free finite automaton there exists, by Proposition 10, an equivalent normalized cycle-free automaton $\mathbf{A} = (\alpha, A, \beta)$. The definition of the transition matrix A implies that it can be written (not necessarily in a unique way) in the form $A = A_0 + A_1$, where $A_0 \in S_0^{n\times n}$ and $A_1 \in (S_0\Sigma)^{n\times n}$. Assume that A is cycle-free of index k. Then $A^k = A_0^k + B \in I^{n\times n}$, where $A_0^k \in S_0^{n\times n}$ and $B \in I^{n\times n}$. By the additional condition on S_0 we obtain $A_0^k = 0$ and $A_0^* = A_0^{k-1} + \ldots + E \in S_0^{n\times n}$. Hence $A_0^*A_1 \in (S_0\Sigma)^{n\times n}$ and $A_0^*\beta \in S_0^{n\times n}$.

We now define the finite automaton $\mathbf{A}'$ by $A' = A_0^*A_1$, $\alpha' = \alpha$, $\beta' = A_0^*\beta$ and show the equivalence of $\mathbf{A}$ and $\mathbf{A}'$:

$$|\mathbf{A}'| = \alpha(A_0^*A_1)^*A_0^*\beta = \alpha(A_0 + A_1)^*\beta = \alpha A^*\beta = |\mathbf{A}| \,.$$

Here we have applied Corollary 1 in the second equality. □

We now define, for given finite automata $\mathbf{A} = (\alpha, A, \beta)$ and $\mathbf{A}' = (\alpha', A', \beta')$ of dimensions n and n', respectively, the finite automata $\mathbf{A} + \mathbf{A}'$ and $\mathbf{A} \cdot \mathbf{A}'$ of dimension $n + n'$:

$$\mathbf{A} + \mathbf{A}' = ((\alpha\ \alpha'), \begin{pmatrix} A & 0 \\ 0 & A' \end{pmatrix}, \begin{pmatrix} \beta \\ \beta' \end{pmatrix}),$$

$$\mathbf{A} \cdot \mathbf{A}' = ((\alpha\ 0), \begin{pmatrix} A & \beta\alpha' \\ 0 & A' \end{pmatrix}, \begin{pmatrix} 0 \\ \beta' \end{pmatrix}).$$

Since the entries of $\beta\alpha'$ are in S_0, the entries of the transition matrices of $\mathbf{A}+\mathbf{A}'$ and $\mathbf{A} \cdot \mathbf{A}'$ are in $S_0 + S_0\Sigma$. If $\mathbf{A}$ and $\mathbf{A}'$ are cycle-free then, by Corollary 3, the transition matrices of $\mathbf{A} + \mathbf{A}'$ and $\mathbf{A} \cdot \mathbf{A}'$ are cycle-free. Hence, $\mathbf{A} + \mathbf{A}'$ and $\mathbf{A} \cdot \mathbf{A}'$ are then again cycle-free finite automata.

Proposition 12. *Let $\mathbf{A}$ and $\mathbf{A}'$ be cycle-free finite automata. Then $\mathbf{A}+\mathbf{A}'$ and $\mathbf{A} \cdot \mathbf{A}'$ are again cycle-free finite automata and*

$$|\mathbf{A} + \mathbf{A}'| = |\mathbf{A}| + |\mathbf{A}'| \qquad \text{and} \qquad |\mathbf{A} \cdot \mathbf{A}'| = |\mathbf{A}||\mathbf{A}'| \,.$$

Proof. For the proof of the equalities we apply Corollary 3:

$$|\mathbf{A} + \mathbf{A}'| = (\alpha\ \alpha') \begin{pmatrix} A & 0 \\ 0 & A' \end{pmatrix}^* \begin{pmatrix} \beta \\ \beta' \end{pmatrix} =$$

$$(\alpha\ \alpha') \begin{pmatrix} A^* & 0 \\ 0 & A'^* \end{pmatrix} \begin{pmatrix} \beta \\ \beta' \end{pmatrix} = \alpha A^*\beta + \alpha' A'^*\beta' = |\mathbf{A}| + |\mathbf{A}'| \,;$$

$$|\mathbf{A} \cdot \mathbf{A}'| = (\alpha\ 0) \begin{pmatrix} A & \beta\alpha' \\ 0 & A' \end{pmatrix}^* \begin{pmatrix} 0 \\ \beta' \end{pmatrix} =$$

$$(\alpha\ 0) \begin{pmatrix} A^* & A^*\beta\alpha' A'^* \\ 0 & A'^* \end{pmatrix} \begin{pmatrix} 0 \\ \beta' \end{pmatrix} = \alpha A^*\beta\alpha' A'^*\beta' = |\mathbf{A}||\mathbf{A}'| \,.$$

□

Proposition 13. *Let $a \in S_0 + S_0\Sigma$. Then $a \in \mathbf{Rec}_{S,I}(S_0, \Sigma)$.*

Proof. Consider the following finite automaton $\mathbf{A}_a$, $a \in S_0 + S_0\Sigma$, of dimension 2:

$$\mathbf{A}_a = ((1\ 0), \begin{pmatrix} 0 & a \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}).$$

Clearly, $\mathbf{A}_a$ is cycle-free of index 2 and we obtain

$$|\mathbf{A}_a| = (1\ 0) \begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = a\,.$$

□

Corollary 4. $\mathbf{Rec}_{S,I}(S_0, \Sigma)$ *is a subsemiring of S containing $S_0 \cup \Sigma$.*

We define, for a given finite automaton $\mathbf{A} = (\alpha, A, \beta)$ the finite automaton $\mathbf{A}^+ = (\alpha, A + \beta\alpha, \beta)$. Since the entries of $\beta\alpha$ are in S_0, the entries of the transition matrix of $\mathbf{A}^+$ are in $S_0 + S_0\Sigma$.

Proposition 14. *Suppose that (S, S_0, I) is cycle-free and $\Sigma \subseteq I$. Then, for $a \in \mathbf{Rec}_{S,I}(S_0, \Sigma) \cap I$, $a^* \in \mathbf{Rec}_{S,I}(S_0, \Sigma)$.*

Proof. Let $a \in \mathbf{Rec}_{S,I}(S_0, \Sigma) \cap I$. Then, by Proposition 11, there exists a finite automaton $\mathbf{A} = (\alpha, A, \beta)$ with $A \in (S_0\Sigma)^{n\times n}$, $\alpha \in S_0^{1\times n}$ and $\beta \in S_0^{n\times 1}$ such that $a = |\mathbf{A}|$. Since $a = \alpha A^*\beta = \alpha\beta + \alpha AA^*\beta$, where $\alpha\beta \in S_0$ and $\alpha AA^*\beta \in I$, we infer by the additional condition on S_0 that $\alpha\beta = 0$.

Considering the transition matrix of the finite automaton $\mathbf{A}^+$, we obtain $(A + \beta\alpha)^2 = A^2 + A\beta\alpha + \beta\alpha A + \beta\alpha\beta\alpha = A^2 + A\beta\alpha + \beta\alpha A \in I^{n\times n}$. Hence, the transition matrix of $\mathbf{A}^+$ is cycle-free of index 2. Observe that $A^*\beta\alpha = \beta\alpha + AA^*\beta\alpha$ is cycle-free for a similar reason; thus we can in the following computation apply Proposition 5 in the second equality and Proposition 4 in the third equality and obtain

$$|\mathbf{A}^+| = \alpha(A + \beta\alpha)^*\beta = \alpha(A^*\beta\alpha)^*A^*\beta = (\alpha A^*\beta)(\alpha A^*\beta)^* = |\mathbf{A}||\mathbf{A}|^*\,.$$

Hence, $aa^* \in \mathbf{Rec}_{S,I}(S_0, \Sigma)$.

Consider now the cycle-free finite automaton $\mathbf{A}_1 + \mathbf{A}^+$. It has the behavior $1 + |\mathbf{A}||\mathbf{A}|^* = |\mathbf{A}|^*$. Hence, $a^* \in \mathbf{Rec}_{S,I}(S_0, \Sigma)$. □

Corollary 5. *Suppose that (S, S_0, I) is cycle-free and $\Sigma \subseteq I$. Then $a^* \in \mathbf{Rec}_{S,I}(S_0, \Sigma)$ if $a \in \mathbf{Rec}_{S,I}(S_0, \Sigma)$ is cycle-free.*

Corollary 6. *Suppose that (S, S_0, I) is cycle-free and $\Sigma \subseteq I$. Then $\mathbf{Rec}_{S,I}(S_0, \Sigma)$ is a partial iterative subsemiring of S (and hence, a partial Conway subsemiring of S) containing $S_0 \cup \Sigma$ over the ideal $\mathbf{Rec}_{S,I}(S_0, \Sigma) \cap I$ of $\mathbf{Rec}_{S,I}(S_0, \Sigma)$.*

Corollary 4 and Propositions 13, 14 show that, under an additional condition on S_0, $\mathbf{Rat}_{S,I}(S_0, \Sigma) \subseteq \mathbf{Rec}_{S,I}(S_0, \Sigma)$. We now prove the converse.

Proposition 15. $\mathbf{Rec}_{S,I}(S_0, \Sigma) \subseteq \mathbf{Rat}_{S,I}(S_0, \Sigma)$.

Proof. Let $\mathbf{A} = (\alpha, A, \beta)$ be a cycle-free finite automaton, where A is cycle-free of index k. Then $|\mathbf{A}| = \alpha A^* \beta = \alpha (A^k)^* (A^{k-1} + \ldots + E)\beta = \alpha(A^{k-1} + \ldots + E)\beta + \alpha A^k (A^k)^* (A^{k-1} + \ldots + E)\beta$. By a proof analogous to that of Lemma 6.8 of Bloom, Ésik, Kuich [2], the entries of $A^k(A^k)^*$ are in $\mathbf{Rat}_{S,I}(S_0, \Sigma)$. Since the entries of $\alpha, \beta, A^{k-1}, \ldots, E$ are also in $\mathbf{Rat}_{S,I}(S_0, \Sigma)$, the behavior $|\mathbf{A}|$ is in $\mathbf{Rat}_{S,I}(S_0, \Sigma)$. □

Corollary 7. *Suppose that (S, S_0, I) is cycle-free and $\Sigma \subseteq I$. Then*

$$\mathbf{Rec}_{S,I}(S_0, \Sigma) = \mathbf{Rat}_{S,I}(S_0, \Sigma).$$

Corollary 8. *Let (S, S_0, I) be cycle-free, and suppose that $\Sigma \subseteq I$.Then $\mathbf{Rec}_{S,I}(S_0, \Sigma)$ is the least partial iterative subsemiring of S (and hence, the least partial Conway subsemiring of S) containing $S_0 \cup \Sigma$ over the ideal $\mathbf{Rec}_{S,I}(S_0, \Sigma) \cap I$ of $\mathbf{Rec}_{S,I}(S_0, \Sigma)$.*

Corollary 4.11 and Corollary 6.13 of Bloom, Ésik, Kuich [2] show that under the conditions of Corollary 8, our set $\mathbf{Rec}_{S,I}(S_0, \Sigma)$ coincides with the set $\mathbf{Rec}_S(S_0, \Sigma)$ of Bloom, Ésik, Kuich [2].

4 Cycle-Free Finite Automata with Discounting

In this section we apply our results to a generalization of the usual power series semiring: to power series semirings over a graded monoid with discounting. We reprove a result of Droste, Sakarovitch, Vogler [6].

A monoid $\langle M, \cdot, e\rangle$ is called *graded* if it is equipped with a length function $|\;| : M \to \mathbb{N}$ that is an additive morphism. (See Sakarovitch [12,13].)

For a semiring S, we denote by $\mathrm{End}(S)$ the monoid of all endomorphisms of S, with composition as monoid operation and the identity morphism as unit.

For the rest of this section, let $\langle M, \cdot, e\rangle$ be a finitely generated graded monoid with length function $|\;|$, let $\langle S, +, \cdot, 0, 1\rangle$ be a semiring and let $\phi : M \to \mathrm{End}(S)$ be a monoid morphism.

A *formal power series over M and S* is a mapping $r : M \to S$, written as $r = \sum_{m \in M} (r, m)m$, where $(r, m) = r(m)$ is the *coefficient* of m. The set of all these power series is denoted by S^M. Let $r, s \in S^M$. *Addition* of r, s is defined pointwise by letting $(r + s, m) = (r, m) + (s, m)$ for all $m \in M$. *Multiplication* of r, s is defined by the *ϕ-Cauchy product* $r \cdot_\phi s$ of r and s by letting

$$(r \cdot_\phi s, m) = \sum_{m=uv} (r, u)\phi(u)(s, v) \qquad \text{for all } m \in M.$$

The usual definitions on power series over Σ^* and S, Σ an alphabet, can be easily transferred to power series in S^M.

Theorem 1 (Droste, Kuske [5], Droste, Sakarovitch, Vogler [6])**.** *The algebra $S_\phi\langle\langle M\rangle\rangle = \langle S^M, +, \cdot_\phi, 0, e\rangle$ is a semiring. Moreover, the algebra $S_\phi\langle M\rangle$ of polynomials is a subsemiring of $S_\phi\langle\langle M\rangle\rangle$.*

In the sequel we write $S_\phi\langle\langle M\rangle\rangle$ for the set S^M of formal power series over M and S.

Theorem 2. *Let S be a partial iterative semiring over the ideal I'. Then $S_\phi\langle\langle M\rangle\rangle$ is a partial iterative semiring over the ideal $\{r \in S_\phi\langle\langle M\rangle\rangle \mid (r,e) \in I'\}$.*

Proof. Consider the equation $y = ry + s$, $r, s \in S_\phi\langle\langle M\rangle\rangle$ with $(r,e) \in I'$. Let $r^* = \sum_{j\geq 0} r^j$. Here $r^0 = 1$ and $r^{j+1} = r \cdot_\phi r^j = r^j \cdot_\phi r$, $j \geq 0$. Clearly, $\{r^j \mid j \geq 0\}$ is locally finite and hence, r^* is well defined.

By an argument similar to that of Theorem 5.6 of Kuich [10], r^* satisfies

$$\begin{aligned}(r^*, e) &= (r,e)^*\,,\\ (r^*, m) &= \textstyle\sum_{uv=m,\ u\neq e}(r^*, e)(r,u) \cdot_\phi (r^*, v)\,.\end{aligned}$$

Let $t \in S_\phi\langle\langle M\rangle\rangle$ be any solution of $y = ry + s$. Then, for all $m \in M$,

$$(t,m) = \sum_{uv=m} (r,u) \cdot_\phi (t,v) + (s,m)\,.$$

We claim that $(t,m) = (r^* \cdot_\phi s, m)$ for all $m \in M$ and prove it by induction on $|m|$.

Let $m = e$. Then $(t,e) = (r,e)(t,e) + (s,e)$. Hence, $(t,e) = (r,e)^*(s,e) = (r^* \cdot_\phi s, e)$.

Let now $|m| > 1$. Then

$$(t,m) = (r,e)(t,m) + \sum_{uv=m,\ u\neq e} (r,u) \cdot_\phi (r^* \cdot_\phi s, v) + (s,m)$$

implies

$$\begin{aligned}&(t,m) = (r^*, e)\textstyle\sum_{uv_1v_2=m,\ u\neq e}(r,u) \cdot_\phi (r^*, v_1) \cdot_\phi (s, v_2) + (r^*, e)(s,m) =\\ &\textstyle\sum_{u_1v_2=m,\ u_1\neq e}(r^*, u_1) \cdot_\phi (s, v_2) + (r^*, e)(s,m) = (r^* \cdot_\phi s, m)\,.\end{aligned}$$

Hence, $r^* \cdot_\phi s$ is the unique solution of $y = ry + s$. □

In the sequel, $S_\phi\langle\{e\}\rangle$ denotes the subsemiring $\{ae \mid a \in S\}$ of $S_\phi\langle\langle M\rangle\rangle$ and I an ideal of $S_\phi\langle\langle M\rangle\rangle$. A *finite automaton in $S_\phi\langle\langle M\rangle\rangle$ and I over $(S_\phi\langle\{e\}\rangle, M)$*

$$\mathbf{A} = (\alpha, A, \beta)$$

is given by

(i) a *transition matrix* $A \in (S_\phi\langle M\rangle)^{n\times n}$,
(ii) an *initial vector* $\alpha \in (S_\phi\langle\{e\}\rangle)^{1\times n}$,
(iii) a *final vector* $\beta \in (S_\phi\langle\{e\}\rangle)^{n\times 1}$.

This definition is a specialization of the definition of finite automaton in Section 3. The finite automaton $\mathbf{A} = (\alpha, A, \beta)$ is called *proper* or *cycle-free* if A is proper or cycle-free, respectively. The *behavior* $|\mathbf{A}|$ of a cycle-free finite automaton $\mathbf{A}$ is given by

$$|\mathbf{A}| = \alpha \cdot_\phi A^* \cdot_\phi \beta\,.$$

Let now $S^{\mathrm{rec}}_{I,\phi}\langle\langle M\rangle\rangle$ and $S^{\mathrm{rat}}_{I,\phi}\langle\langle M\rangle\rangle$ denote the sets $\mathbf{Rec}_{S_\phi\langle\langle M\rangle\rangle,I}(S_\phi\langle\{e\}\rangle, M)$ and $\mathbf{Rat}_{S_\phi\langle\langle M\rangle\rangle,I}(S_\phi\langle\{e\}\rangle, M)$, respectively. (Here the definition of **Rec** and **Rat** is adjusted from Σ^* to M.)

Corollary 8 implies the next theorem.

Theorem 3. *Let S be a partial iterative semiring over the ideal I' and $I = \{r \in S_\phi\langle\langle M\rangle\rangle \mid (r, e) \in I'\}$. Suppose $(S_\phi\langle\langle M\rangle\rangle, S, I)$ is cycle-free. Then*

$$S^{\mathrm{rec}}_{I,\phi}\langle\langle M\rangle\rangle = S^{\mathrm{rat}}_{I,\phi}\langle\langle M\rangle\rangle$$

is the least partial iterative subsemiring of $S_\phi\langle\langle M\rangle\rangle$ (and hence, the least Conway subsemiring of $S_\phi\langle\langle M\rangle\rangle$) containing $S_\phi\langle\{e\}\rangle \cup M$ over the ideal $S^{\mathrm{rec}}_{I,\phi}\langle\langle M\rangle\rangle \cap I$.

This theorem generalizes the Kleene-Schützenberger Theorem of Schützenberger [14].

The finite S-automata over M in Droste, Sakarovitch, Vogler [6] are nothing other than our finite automata in $S_\phi\langle\langle M\rangle\rangle$ and $I = \{r \in S_\phi\langle\langle M\rangle\rangle \mid (r, e) = 0\}$ over $(S_\phi\langle\{e\}\rangle, M - \{e\})$ with proper transition matrix.

Corollary 9 (Droste, Sakarovitch, Vogler [6])**.** *Let $I = \{r \in S_\phi\langle\langle M\rangle\rangle \mid (r, e) = 0\}$. Then $S^{\mathrm{rec}}_{I,\phi}\langle\langle M\rangle\rangle = S^{\mathrm{rat}}_{I,\phi}\langle\langle M\rangle\rangle$ is the least partial iterative subsemiring of $S_\phi\langle\langle M\rangle\rangle$ (and hence, the least Conway subsemiring of $S_\phi\langle\langle M\rangle\rangle$) containing $S_\phi\langle\{e\}\rangle \cup M$ over the ideal $S^{\mathrm{rec}}_{I,\phi}\langle\langle M\rangle\rangle \cap I$.*

We now assume, for the rest of this section, that S is a partial Conway semiring.

Theorem 4. *If S is a Conway semiring then so is $S_\phi\langle\langle M\rangle\rangle$.*

Proof. In the definition of r^*, $r \in S_\phi\langle\langle M\rangle\rangle$, and in the proof of Corollary 2.4 of Kuich [10] replace $\varphi^{|w|}$ by $\phi(w)$, $w \in \Sigma^*$ by $w \in M$, and ε by e. □

In the next theorem, we assume S is a Conway semiring, and $S_\phi\langle\langle M\rangle\rangle$ is a partial Conway semiring over the ideal $S_\phi\langle\langle M\rangle\rangle$ and apply Corollary 6.12 of Bloom, Ésik, Kuich [2] or Theorem 3.2 of Ésik, Kuich [7].

Corollary 10. *Let S be a Conway semiring. Then*

$$S^{\mathrm{rec}}_{S_\phi\langle\langle M\rangle\rangle,\phi}\langle\langle M\rangle\rangle = S^{\mathrm{rat}}_{S_\phi\langle\langle M\rangle\rangle,\phi}\langle\langle M\rangle\rangle$$

is the least Conway subsemiring of $S_\phi\langle\langle M\rangle\rangle$ which contains $S\langle\{e\}\rangle \cup M$.

Theorem 5. *If S is a partial Conway semiring over the ideal I' then $S_\phi\langle\langle M\rangle\rangle$ is a partial Conway semiring over the ideal $I = \{r \in S_\phi\langle\langle M\rangle\rangle \mid (r, e) \in I'\}$.*

Proof. In a first step, change the proof of Corollary 2.4 of Kuich [10] according to the proof of Theorem 4. Now inspect this proof and assume that the power series r and s are in I. We have to check, whether the * of all power series, taken in the proof of Theorem 4, does exist; i. e., we have to check that the *-operation is applied only to power series t where $(t, e) \in I'$. Inspection shows that this is the case and the * of all used power series is defined. □

Corollary 6.13 of Bloom, Ésik, Kuich [2] implies our next result.

Corollary 11. *Let S be a partial Conway semiring with distinguished ideal I' and $I = \{r \in S_\phi\langle\langle M\rangle\rangle \mid (r,e) \in I'\}$. Suppose $(S_\phi\langle\langle M\rangle\rangle, S, I)$ is cycle-free. Then*

$$S^{\mathrm{rec}}_{I,\phi}\langle\langle M\rangle\rangle = S^{\mathrm{rat}}_{I,\phi}\langle\langle M\rangle\rangle$$

is the least partial Conway subsemiring of $S_\phi\langle\langle M\rangle\rangle$ containing $S_\phi\langle\{e\}\rangle \cup M$ with distinguished ideal $S^{\mathrm{rec}}_{I,\phi}\langle\langle M\rangle\rangle \cap I$.

References

1. Bloom, S.L., Ésik, Z.: Iteration Theories: The Equational Logic of Iterative Processes. EATCS monographs on Theoretical Computer Science. Springer, Heidelberg (1993)
2. Bloom, S.L., Ésik, Z., Kuich, W.: Partial Conway and iteration semirings. Fundamenta Informaticae 86, 19–40 (2008)
3. Conway, J.H.: Regular Algebra and Finite Machines. Chapman and Hall, Boca Raton (1971)
4. Droste, M., Kuich, W.: Semirings and formal power series. In: Droste, M., Kuich, W., Vogler, H. (eds.) Handbook of Weighted Automata. EATCS Monographs on Theoretical Computer Science, ch. 1. Springer, Heidelberg (to appear, 2009)
5. Droste, M., Kuske, D.: Skew and infinitary formal power series. Theoretical Computer Science 366, 199–227 (2006)
6. Droste, M., Sakarovitch, J., Vogler, H.: Weighted automata with discounting. Information Processing Letters 108, 23–28 (2008)
7. Esik, Z., Kuich, W.: Equational axioms for a theory of automata. In: Martin-Vide, C., Mitrana, V., Paun, G. (eds.) Formal Languages and Applications. Studies in Fuzziness and Soft Computing, 148, pp. 183–196. Springer, Heidelberg (2004)
8. Esik, Z., Kuich, W.: Modern Automata Theory (2007), www.dmg.tuwien.ac.at/kuich
9. Ésik, Z., Kuich, W.: Finite Automata. In: Droste, M., Kuich, W., Vogler, H. (eds.) Handbook of Weighted Automata. EATCS Monographs on Theoretical Computer Science, ch. 3. Springer, Heidelberg (2009)
10. Kuich, W.: Kleene Theorems for skew formal power series. Acta Cybernetica 17, 719–749 (2006)
11. Kuich, W., Salomaa, A.: Semirings, Automata, Languages. EATCS Monographs on Theoretical Computer Science. Springer, Heidelberg (1986)
12. Sakarovitch, J.: Éléments de théorie des automates. Vuibert (2003)
13. Sakarovitch, J.: Rational and recognizable power series. In: Droste, M., Kuich, W., Vogler, H. (eds.) Handbook of Weighted Automata. EATCS Monographs on Theoretical Computer Science, ch. 4. Springer, Heidelberg (to appear, 2009)
14. Schützenberger, M.P.: On the definition of a family of automata. Inf. Control 4, 245–270 (1961)

Picture Languages: From Wang Tiles to 2D Grammars*

Alessandra Cherubini[1] and Matteo Pradella[2]

[1] Politecnico di Milano
[2] CNR IEIIT-MI
P.zza L. da Vinci, 32, 20133 Milano, Italy
{alessandra.cherubini,matteo.pradella}@polimi.it

Abstract. The aim of this paper is to collect definitions and results on the main classes of 2D languages introduced with the attempt of generalizing regular and context-free string languages and in same time preserving some of their nice properties. Almost all the models here described are based on tiles. So we also summarize some results on Wang tiles and its applications.

1 Introduction

The interest for a robust theory of two-dimensional (2D) languages (or picture languages) comes from the increasing relevance of pattern recognition and image processing. The main attempt of the research in this area is to generalize the richness of the theory of 1D languages to two dimensions. First focus was on definitions of classes of picture languages that are the analogue of the classes of Chomsky's hierarchy for 1D languages, in sense that, restricting to pictures of size $(1, n)$, picture and string languages at each level of the hierarchy coincide and that the new definitions for pictures inherit as many as possible properties from the corresponding definitions for strings.

Several different approaches were considered in the whole literature on the topic. The generalizations that seem to be the best answers to previous requests for the two lower levels of Chomsky's hierarchy are essentially based on Wang tiles and in this paper we aim to give a survey of classical and new results on these picture languages. Wang tiles, introduced in 1961, are squares whose all edges are colored. A finite set of Wang tiles admits a valid tiling of the plane if copies of the tiles can be arranged one by one, without rotations or reflections, to fill the plane so that all shared edges between tiles have matching colors. In 1966, Berger [8] proved that the problem of determining whether a given finite set of Wang tiles can tile the plane is undecidable, and constructed the first example of an aperiodic set of Wang tiles, i.e. a finite set of tiles whose all valid tilings have no periodic behavior. Several papers are devoted to the problem of determining small aperiodic set of Wang tiles but recently the main interest in Wang tiles was motivated by applications which, besides computer graphics, start to involve appealing areas in the frameworks of nanotechnologies and so called life sciences.

* Work partially supported by ESF *Automata: from Mathematics to Applications (AutoMathA)*, CNR RSTL 760 *Grammatiche 2D per la descrizione di immagini*, and by MIUR PRIN project *Mathematical aspects and emerging applications of automata and formal languages*.

For the ground level of Chomsky's hierarchy a robust definition of recognizable picture languages was proposed in 1991 by Giammarresi and Restivo. They defined the family REC of recognizable picture languages *by projection of local properties*, [31]. This class is considered *the* generalization of the class of regular 1D languages because it unifies several approaches to define the two dimension analogue of regular languages via finite automata, grammars, logic and regular expressions.

In 2005 Crespi Reghizzi and Pradella [18] introduced tile grammars, a model of grammars that extends the context-free (CF) grammars for 1D languages to two dimensions. The right hand part of each rule of a tile grammar is a set of tiles determining a local picture language. A rule is applied to the current picture replacing a rectangular subpicture, completely filled by the left hand side of the rule, with an isometric rectangle belonging to the local picture language determined by the right hand part of the rule. The generative power of these grammars exceeds REC languages. More recently a simplified version of tiling in the right hand part of the rules was considered in [15], giving raise to a new model of grammars called regional tile grammars. The new model includes several models of grammars proposed as generalizations of CF 1D grammars, the membership problem is solved by a polynomial time algorithm that naturally extends the classical CKY algorithm for strings, but it generates a family of languages incomparable with REC.

The first section of the paper contains some basic notions on pictures and picture languages. Then, some information on Wang tiles is given in second section, third and forth sections are devoted to collect results respectively on REC family and on several types of grammars proposed as generalization of CF 1D languages included in the family generated by tile grammars. In the last section, some open problems and some hints on different approaches to picture grammars are given.

2 Basic Definitions

In this section some standard definitions of pictures, picture languages and operations on pictures are recalled.

Let Σ be a finite alphabet. A *picture* over Σ is a 2D array of elements of Σ called *pixels*. The *size* $|p|$ of a picture p is the pair $(|p|_{row}, |p|_{col})$ of its number of rows (its height) and columns (width). The indices grow from top to bottom for the rows and from left to right for the columns. The set of all pictures over Σ is denoted by $\Sigma^{+,+}$. $\Sigma^{*,*}$ is $\Sigma^{+,+} \cup \{\lambda\}$, where λ is the empty picture. For $h, k \geq 1$, $\Sigma^{h,k}$ (resp. $\Sigma^{h,+}$, $\Sigma^{+,k}$) is the set of all pictures of size (h,k) (resp. with h rows, with k columns). A *picture language* over Σ is a subset of $\Sigma^{*,*}$. # $\notin \Sigma$ is used when needed as a *boundary symbol*; $\hat{p}$ refers to the bordered version of picture p. That is, for $p \in \Sigma^{h,k}$, $\hat{p}$ is

$$\hat{p} = \begin{array}{ccccc} \# & \# & \dots & \# & \# \\ \# & p(1,1) & \dots & p(1,k) & \# \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \# & p(h,1) & \dots & p(h,k) & \# \\ \# & \# & \dots & \# & \# \end{array}$$

The *domain* of a picture p is the set $\mathrm{dom}(p) = \{1, \ldots, |p|_{row}\} \times \{1, \ldots, |p|_{col}\}$ and $\mathrm{dom}(\hat{p}) = \{0, \ldots, |p|_{row} + 1\} \times \{0, \ldots, |p|_{col} + 1\}$ is the domain of the bordered picture $\hat{p}$.

A *subdomain* of $\mathrm{dom}(p)$ is a set d of the form $\{x, \ldots, x'\} \times \{y, \ldots, y'\}$ where $1 \leq x \leq x' \leq |p|_{row},\ 1 \leq y \leq y' \leq |p|_{col}$; the size of d is $(x' - x + 1, y' - y + 1)$. We will often denote a subdomain by using its top-left and bottom-right coordinates, in the previous case the quadruple $(x, y; x', y')$[1]. Subdomains of $\mathrm{dom}(\hat{p})$ are defined analogously. Each subdomain of $\mathrm{dom}(\hat{p})$ of size $(1, 1)$ is called a *position* of p. The *translation* of a subdomain $d = (x, y; x', y')$ by displacement $(a, b) \in \mathbb{Z}^2$ is the subdomain $d' = (x + a, y + b; x' + a, y' + b)$: we will write $d' = \mathrm{transl}_{(a,b)}(d)$. Pairs $(0, i), (|p|_{row} + 1, i), (j, 0), (j, |p|_{col} + 1)$ with $0 \leq i \leq |p|_{col} + 1,\ 0 \leq j \leq |p|_{row} + 1$, are called *external positions* of p, the other are called *internal positions*. Positions in the set $\{(0, 0), (0, |p|_{col} + 1), (|p|_{row} + 1, 0), (|p|_{row} + 1, |p|_{col} + 1)\}$ are called *corner positions*. Given a position (i, j) with $1 \leq i \leq |p|_{row} + 1$ and $1 \leq j \leq |p|_{col} + 1$ its *top-left-* (*tl-* for short) contiguous positions are the positions: $(i, j - 1), (i - 1, j - 1), (i - 1, j)$. Analogously for tr, bl, br where t, b, l, r are used for top, bottom, left and right respectively. For any internal position, its contiguous positions are all the tl-, tr-, br-, and bl-ones. Since each set $P(n, m) = \{0, 1 \ldots, n + 1\} \times \{0, 1 \ldots, m + 1\}$ can be seen as the domain of a bordered picture $\hat{p}$ with p of size (n, m), the elements of $P(n, m)$ are sometimes called positions of $P(n, m)$ as well.

The pixel of the picture p at position (i, j) of $\mathrm{dom}(p)$ is denoted $p(i, j)$. If all pixels of a picture p over Σ belong to an alphabet $\Sigma' \subseteq \Sigma$, p is called *Σ'-homogeneous*, a picture which is $\{a\}$-homogeneous for some $a \in \Sigma$ is called an *a-picture*, or also a homogeneous picture. If $a \in \Sigma$, $a^{h,k}$ stands for the a-picture in $\Sigma^{h,k}$, while $a^{+,+}$ stands for the set of a-pictures in $\Sigma^{+,+}$.

Let p be a picture over Σ and let $d = (x, y; x', y') \subseteq dom(p)$, the *subpicture* $\mathrm{spic}(p, d)$ associated to d is the picture of the same size of d such that, $\forall i \in \{1, \ldots, x' - x + 1\}$ and $\forall j \in \{1, \ldots, y' - y + 1\}$, $\mathrm{spic}(p, d)(i, j) = p(x + i - 1, y + j - 1)$. A subpicture q of p, written $q \trianglelefteq p$, is a subpicture $\mathrm{spic}(p, d)$ associated to some subdomain d of p. If $d = (x, y; x + h - 1, y + k - 1)$, then the subpicture $q = \mathrm{spic}(p, d)$ is also called the subpicture of p of size (h, k) at position (x, y), written $q \trianglelefteq_{(x,y)} p$.The *set of subpictures* of size (h, k) of p is denoted by

$$B_{h,k}(p) = \{q \in \Sigma^{h,k} : q \trianglelefteq p\}.$$

A picture $q \in \Sigma^{m,n}$ is called a *scattered subpicture* [2] of $p \in \Sigma^{+,+}$ if there are strictly monotone functions $f : \{1, 2, \ldots, m\} \to \{n \in N \mid n \geq 1\}$, $g : \{1, 2, \ldots, n\} \to \{n \in N \mid n \geq 1\}$ such that $p(f(i), g(j)) = q(i, j)$ for all $(i, j) \in \{1, 2, \ldots, n\} \times \{1, 2, \ldots, m\}$.

Now we shortly present main picture-combining and transforming operators.

The *column concatenation* $\mathbin{⦶}$, for all pictures p, q such that $|p|_{row} = |q|_{row}$, written $p \mathbin{⦶} q$, is defined as:

[1] Notice that the Cartesian coordinate system is clockwise rotated of 90^o with respect to the standard one.

[2] A scattered subpicture is often called a subpicture, and subpictures in our sense are called blocks.

$$p \oplus q = \begin{array}{ccc} p(1,1) & \dots & p(1,|p|_{col}) \\ \vdots & \ddots & \vdots \\ p(|p|_{row},1) & \dots & p(|p|_{row},|p|_{col}) \end{array} \begin{array}{ccc} q(1,1) & \dots & q(1,|q|_{col}) \\ \vdots & \ddots & \vdots \\ q(|q|_{row},1) & \dots & q(|q|_{row},|q|_{col}) \end{array}$$

The row concatenation $\ominus$ for pictures p, q, written $p \ominus q$, is defined analogously (with p on top). The empty picture λ is the neutral element for both concatenation operations. $p^{k\oplus}$ is the horizontal juxtaposition of k copies of p; $p^{*\oplus}$ is the corresponding closure. ${}^{k\ominus}$, and ${}^{*\ominus}$ are the row analogous.

The *projection by mapping* $\pi : \Sigma \to \Delta$ of a picture $p \in \Sigma^{+,+}$ is a picture $p' \in \Delta^{+,+}$ such that $|p| = |p'|$ and $p'(i,j) = \pi(p(i,j))$ for every position (i,j) of p.

The (clockwise) *rotation* of a picture p, $rot(p)$, is informally described as follows:

$$rot(p) = \begin{array}{ccc} p(|p|_{row},1) & \dots & p(1,1) \\ \vdots & \ddots & \vdots \\ p(|p|_{row},|p|_{col}) & \dots & p(1,|p|_{col}) \end{array}$$

The *pixel-wise Cartesian product* of two pictures $p \in \Sigma_1^{*,*}, q \in \Sigma_2^{*,*}$ with $|p| = |q|$, is a picture $f \in (\Sigma_1 \times \Sigma_2)^{*,*}$ such that $|f| = |p|$, and $f(i,j) = (p(i,j), q(i,j))$ for all $i, j, 1 \leq i \leq |p|_{row}, 1 \leq j \leq |p|_{col}$ [50].

Projection, rotation, row and column concatenation, and pixel-wise Cartesian product can be extended to picture languages as usual. For every language $L \subseteq \Sigma^{*,*}$ we set $L^{0\oplus} = L^{0\ominus} = \lambda$, $L^{i\oplus} = L \oplus L^{(i-1)\oplus}$ and $L^{i\ominus} = L \ominus L^{(i-1)\ominus}$ for every $i \geq 1$. Thus, the row and column closures can be defined as the transitive closures of $\oplus$ and $\ominus$:

$$L^{*\oplus} = \bigcup_{i \geq 0} L^{i\oplus}, \; L^{*\ominus} = \bigcup_{i \geq 0} L^{i\ominus},$$

which can be seen as a sort of 2D Kleene star. In [50] Simplot introduced the closure L^{**}. We omit the detailed definition of Simplot's operator and introduce it quite informally. We say $p \in L^{++}$ iff there exists a partition of $\mathrm{dom}(p)$ where each subpicture associated to a subdomain of the partition is in L. Let L^{**} be the set $L^{++} \cup \{\lambda\}$. For example:

$$\begin{array}{ccc} a & a & b \\ b & e & b \\ b & b & c \end{array} \in \left\{ a\ a\,, \begin{array}{c} b \\ b \end{array}, b\ c\,, \begin{array}{c} d \\ d \end{array}, e \right\}^{**}$$

If all the pictures of L have the same size, then $(L^{*\oplus})^{*\ominus} = (L^{*\ominus})^{*\oplus} = L^{**}$.

A well-known and widely useful concept in 1D languages is substitution, which assigns languages to letters of the alphabet and naturally extends to strings and languages too. In 2D languages, a substitution can be similarly defined. Given two finite alphabets Σ and Δ, a *substitution* from Δ to Σ is a mapping $\sigma : \Delta \to 2^{\Sigma^{+,+}}$. But a difficulty hinders the extension of the mapping to pictures, because of the so-called shearing problem of picture languages: a pixel in a picture cannot be replaced by a larger picture without disrupting the array structure. To overcome the problem in [15] the notion of replacement was introduced. If p, q, q' are pictures such that $q \trianglelefteq_{(i,j)} p$ for some position (i,j) of p, and $|q| = |q'|$, then $p[q'/q]_{(i,j)}$ denotes the picture obtained by replacing the occurrence of q at position (i, j) in p with q', i.e., $p[q'/q]_{(i,j)}(i+x-1, j+y-1) = q'(x, y)$ for all $1 \le x \le |q|_{row}$, $1 \le y \le |q|_{col}$. Then the notion of substitution was modified as follows. Let $\sigma : \Delta \rightarrow 2^{\Sigma^{+,+}}$ be a substitution. Given a picture $p \in \Delta^{+,+}$, a partition $\Pi(\mathrm{dom}(p)) = \{d_1, \ldots, d_n\}$, with $n \ge 1$, of $\mathrm{dom}(p)$ where each subpicture $\mathrm{spic}(p, d_m)$ associated to a subdomain d_m of the partition is a b_m-picture for some $b_m \in \Delta$ is called a *homogeneous partition* of p. Then the *substitution of* $p \in \Delta^{+,+}$ *induced by* $\Pi(\mathrm{dom}(p))$ is the language $\sigma_{\Pi(\mathrm{dom}(p))}(p) = \{p[r_1/\mathrm{spic}(p, d_1)] \ldots [r_n/\mathrm{spic}(p, d_n)] \mid r_m \in \sigma(b_m), 1 \le m \le n\}$. Given $L \subseteq \Sigma^{+,+}$, a set $\Pi = \{(p, \Pi(\mathrm{dom}(p)) \mid p \in L\}$, where each $\Pi(\mathrm{dom}(p))$ is a (homogeneous) partition of $p \in L$, is called a (homogeneous) partition set of L. If $L \subseteq \Delta^{+,+}$ and Π is a homogeneous partition set of L, then the *substitution of* L *induced by the homogeneous partition set* Π is the language $\sigma_\Pi(L) = \{\sigma_{\Pi(\mathrm{dom}(p))}(p) : p \in L\}$.

Roughly speaking a substitution $\sigma : \Delta \rightarrow 2^{\Sigma^{+,+}}$ extends to pictures and to picture languages by replacing a-subpictures p_a, at position (i, j), of p with pictures $q \in \sigma(a)$ of the same size. This definition, however, is not equivalent to the traditional notion of substitution when applied to strings.

Now we are in position of introducing families of 2D languages, but since we are mainly presenting languages based on tiling we remind some notions on Wang tiles.

3 Wang Tiles

A *Wang tile* is a unit square with colored edges. Let T be a finite set of Wang tiles, which are not allowed to rotate. A map $\tau : \mathbb{Z}^2 \rightarrow T$ is called a *valid tiling*, of the Euclidean plane, or equivalently T can tile the Euclidean plane, if common edges of any pair of adjacent tiles have the same color. More formally denote by $N(t)$, $S(t)$, $W(t)$, $E(t)$ the colors of the upper, lower, left and right edges of a tile t respectively, then τ is a valid tiling of the Euclidean plane, if $N(\tau(i, j)) = S(\tau(i, j + 1))$, $S(\tau(i, j)) = N(\tau(i, j - 1))$, $W(\tau(i, j)) = E(\tau(i - 1, j))$, and $E(\tau(i, j)) = W(\tau(i + 1, j))$, for each $(i, j) \in \mathbb{Z}^2$. Analogously, T can tile a rectangle of size $n \times m$ if there is a map $\tau : \{1, \ldots, m\} \times \{1, \ldots, n\} \rightarrow T$ such that adjacent tiles agree on the colors of contiguous edges. In 1961 Wang [53], analyzing the class of the first order formulas in prenex normal form whose prefix is $\forall x \exists y \forall z$, raised the question

Plane tiling problem *given a finite set of Wang tiles establish whether or not it admits a valid tiling.*

The 1D version of this problem admits an easy solution. Namely, to each finite set T of unary segments with colored left and right end points one can associate a direct graph where the set of colors is the set of vertices, and the edges (i, j) are the colors of left and right endpoints of some segment in T. Obviously T admits a valid tiling if and only if there is a bi-infinite path in the associate graph and then if and only if the graph has a loop. Coming back to the 2-dimensional problem, if the given finite set T of Wang tiles has a valid tiling with some vertical periodicity, the plane is covered by the repetition of some horizontal strip. Then, since this strip has only finitely many different vertical cross sections, the tiling has periodicity along two different directions.

A tiling τ is called *periodic* if there are two integers p, q such that $\tau(i, j) = \tau(i + p, j)$, $\tau(i, j) = \tau(i, j + q)$ for all $(i, j) \in \mathbb{Z}^2$. Without loss of generality we can assume

$p = q$. By the above argument it follows that if a finite set of Wang tiles has a tiling with a non zero period along one direction then it admits a periodic tiling.

Wang conjectured that any set of tiles which admits a valid tiling of the plane also admits a periodic tiling and under this assumption he gave an algorithm to solve the plane tiling problem, based on a compactness-like theorem.

Proposition 1. *A finite set of Wang tiles can tile the whole plane iff it can tile arbitrarily large finite areas of the plane.*

In particular a given set of tiles can tile the whole plane if and only if it can tile the first quadrant and so several constraints on the tiling of the first quadrant were posed. These problems were a bit easier to settle than the plane tiling problem and were speedily proved to be undecidable, an overview on these results can be found in [54]. The plane tiling problem on the contrary remained unsolved for years. However, from the above discussion it is clear that if the plane tiling problem is undecidable, then there are finite sets of tiles which admit only non-periodic tilings of the plane.

A finite set of Wang tiles which admits only non-periodic valid tiling is said *aperiodic*. In 1966 Berger [8], proved the following

Theorem 1. *The plane tiling problem is undecidable.*

His proof is based on encoding the halting problem of Turing Machine in the valid tiling of an arbitrary large square portion of the plane. Moreover, he constructed an aperiodic set of 20426 Wang tiles that shortly reduced to 104.

Then several well-known scientists from different areas as discrete mathematics, logic and computer science paid attention to the problem of finding smaller aperiodic sets of tiles and simplified proofs of undecidability of plane tiling problem (see for instance [49]). The smallest aperiodic set of Wang tiles obtained by geometrical arguments is composed by 16 tile (for a survey, see Chapters 10 and 11 of [33]). More recently Kari, [37], proposed a different approach based on sequential machines that multiply Beatty sequences of real numbers by rational constants, and produced an aperiodic set of Wang tiles with 14 tiles. His method was improved by Culik, [20], who built an aperiodic set formed by 13 tiles. This is currently the smallest known aperiodic set of Wang tiles. An expository article describing this approach is [27].

Once proved the existence of aperiodic set of Wang tiles, the following problem naturally arises:

Periodic tiling problem *given a finite set of Wang tiles determine whether or not it can tile the plane periodically.*

The problem was first studied in 1972 by Gurevich and Koriakov, who proved its undecidability [34].

Valid tilings have some quite surprising regularities. Let T be a finite set of Wang tiles, a *pattern* is a partial map $\varphi : P \to T$ from a finite domain P of $\mathbb{Z}^2$ in T. A pattern *appears* in a tiling $\tau : \mathbb{Z}^2 \to T$ if the tiling is the extension of the image of the pattern by a shift.

A valid tiling $\tau : \mathbb{Z}^2 \to T$ is called *quasi-periodic* if for each pattern M appearing in τ there is an integer n such that M appears in all $n \times n$ squares in τ. A valid quasi periodic tiling that is not periodic is called *strictly quasi-periodic*.

In [24] Durand proved the following

Theorem 2. *Each finite set of Wang tiles admitting a valid tiling admits a quasi-periodic valid tiling.*

The *quasi-periodicity* function for a quasi periodic tiling τ is the function that associate to each integer x the minimal size n of the squares in which one can find all the patterns of size x appearing in the tiling.

This function enables to characterize quasi periodic tilings that are periodic.

Proposition 2. *A quasi periodic tiling is periodic if and only if its quasi-periodicity function is bounded by $x \to x + c$, for some constant c .*

Then, using a counting argument on trees suitably associated to valid tilings, Durand obtains the following

Theorem 3. *If a tile set can be used to form a strictly quasi-periodic tiling of the plane, then it can form an uncountable number of different tilings.*

It is important to note that valid tilings could be defined in several different ways. For instance one could arrange all edge colors in complementary pairs and ask for tilings of the plane where common edges of adjacent tiles have complementary colors. This problem is equivalent to the plane tiling problem. If tile rotation is allowed, the tiling problem with matching colors of contiguous edges is trivially solvable while the problem with complementary colors remain undecidable.

A generalized simple way for describing variants of tiling rules is to consider the given finite set T of Wang tiles as a finite alphabet and a set of local rules $L \subseteq T^4$. A tiling τ satisfies the local rules L if and only if all 2×2 patterns appearing in the tiling are in L. In [26] the authors give via this approach a new short proof of the existence of aperiodic tilings.

Besides the strong connections with first order and description logics [25] yet arising from its original motivation, tiling problems have appeared in many branches of physics and mathematics like group theory, topology, quasicrystals, symbolic dynamics. More recently Winfree et al. [56] have demonstrated the feasibility of creating molecular tiles made from DNA that can act as Wang tiles introducing the *tile assembly model*. As pointed out by Brun [13] a tile assembly model is a highly distributed parallel model of computation that may be implemented using molecules, or a large computer network such as the Internet, and this opens several new prospectives.

In a more applicative and less ambitious context, Wang tiles have been proposed as tool for procedural synthesis of textures, and in general they have also proved to be very useful for the creation of large non-periodic textures, point-distributions and complex 2D scenes, see for instance [1,17].

4 Recognizable Picture Languages

The attempt of transferring definitions and properties from string languages to their 2D analogue is quite successful when one considers the first level of Chomsky's hierarchy.

The class of picture languages corresponding to regular one- dimensional languages was intensively studied by several authors with different approaches: finite automata, logical characterizations, regular expressions and so on. An unifying approach to this family of picture language was proposed by Giammarresi and Restivo via local properties and projection. They introduced the so called REC family of picture languages and collected main properties of this family in the nice survey [31]. Here, besides summarizing the results contained in [31], we add some more recent results with the aim of fixing the actual state of art.

4.1 Labeled Wang Tiles and Tiling Systems

First, we remind the definition of REC languages based on tiles endowed with labels in a finite alphabet Σ.

Definition 1. *([21]) A* labeled Wang tile*, shortly* LWT*, is a* 5*-tuple* (c_1, c_2, c_3, c_4, a) *where for all* $i,\ 1 \leq i \leq 4$*,* c_i *belongs to a finite set* C *of "colors" and* a *belongs to a finite set* Σ *of labels.*

A Wang system *(*WS*) is a triple* (C, Σ, T) *where* $T \subseteq C^4 \times \Sigma$ *is a finite set of* LWT*'s.*

Let $B \in C$ *be a special color and let* r *be a picture of size* (n, m) *on the alphabet* T*,* r *is a* tiling *over* T *if*

- $r(1,1) \in \{(B,B,c_3,c_4,a) \mid c_3,c_4 \in C \setminus \{B\}, a \in \Sigma\}$,$r(1,n) \in \{(c_1,B,B,c_4,a) \mid c_1,c_4 \in C \setminus \{B\}, a \in \Sigma\}$, $r(m,n) \in \{(c_1,c_2,B,B,a) \mid c_1,c_2 \in C \setminus \{B\}, a \in \Sigma\}$,$r(m,1) \in \{(B,c_2,c_3,B,a) \mid c_2,c_3 \in C \setminus \{B\}, a \in \Sigma\}$*;*
- *for all* $i,\ 1 < i < n$*,* $r(1,i) \in \{(c_1,B,c_3,c_4,a) \mid c_1,c_3,c_4 \in C \setminus \{B\}, a \in \Sigma\}$, $r(m,i) \in \{(c_1,c_2,c_3,B,a) \mid c_1,c_2,c_3 \in C \setminus \{B\}, a \in \Sigma\}$*;*
- *for all* $i,\ 1 < i < m$*,* $r(i,1) \in \{(B,c_2,c_3,c_4,a) \mid c_2,c_3,c_4 \in C \setminus \{B\}, a \in \Sigma\}$, $r(i,n) \in \{(c_1,c_2,B,c_4,a) \mid c_1,c_2,c_4 \in C \setminus \{B\}, a \in \Sigma\}$*;*
- *for all* $(i,j),\ 1 \leq i \leq m, 1 \leq j \leq n$*,* $r(j,i) \in \{(c_1,c_2,c_3,c_4,a) \mid c_1,c_2,c_3,c_4 \in C \setminus \{B\}, a \in \Sigma\}$*; moreover let* $r(i,j) = (e,n,w,s,a)$*, then if* $i > 1$*,* $r(i-1,j) \in \{(c_1,c_2,c_3,n,a') \mid c_1,c_2,c_3 \in C, a' \in \Sigma\}$*, if* $j > 1$*,* $r(i,j-1) \in \{(c_1,c_2,e,c_4,a') \mid c_1,c_2,c_4 \in C, a' \in \Sigma\}$.

The label $\|r\|$ *of a tiling* r *is a picture over* Σ *of size* $|r|$ *defined by*

$$\|r\|(i,j) = a \Leftrightarrow r(i,j) = (c_1,c_2,c_3,c_4,a)$$

for some $c_1,c_2,c_3,c_4 \in C$*. The set of the labels of all the tilings over* T *is the language* $\mathcal{L}(\mathrm{WS})$ generated by the Wang system WS*. A language* L *generated by a Wang system is called* Wang recognizable.

For each LWT $t = (c_1,c_2,c_3,c_4,a)$ in a Wang system WS, consider the non labeled version $\widetilde{t} = (c_1,c_2,c_3,c_4)$. Roughly speaking the above definition says that the map $\rho : \{1,\ldots,m\} \times \{1,\ldots,n\} \to T$ defined as $\rho(h,k) = r(n+1-h,k)$ is a valid tiling of the region $\{1,\ldots,m\} \times \{1,\ldots,n\}$ by the set $\widetilde{\mathrm{WS}}$ of the non labeled versions of tiles in WS such that the boundary of the tiling r is colored by the special color B that does not occur in inner edges.

The same family of picture languages is also introduced by a formalism based on the local rules introduced in Section 3.

For $p \in \Sigma^{+,+}$ let $[\![p]\!]$ be the set of subpictures of size (2,2) of p.[3] In the sequel the concepts of *tile*, and *local language* are central.

Definition 2. *A* tile *is a square picture of size (2,2). A language* $L \subseteq \Sigma^{*,*}$ *is* local *if there exists a finite set* Θ *of tiles over the alphabet* $\Sigma \cup \{\#\}$ *such that* $L = \{p \in \Sigma^{*,*} \mid [\![\hat{p}]\!] \subseteq \Theta\}$. *We will refer to such language as* $\mathrm{LOC}(\Theta)$.

Notice that $\mathrm{LOC}(\Theta)$ is the set of finite rectangles of Euclidean plane with boundary colored by $\#$ that admit a valid tiling agreeing also with the boundary color. The set of local languages, shortly denoted by LOC, is the natural extension of string local languages and so the following definition extends one of the definitions of regular 1D languages.

Definition 3. *([31]) A* tiling system *(TS) is the 4-tuple* $T = (\Sigma, \Gamma, \Theta, \pi)$, *where:*
Σ and Γ are two finite alphabets,
$\pi : \Gamma \to \Sigma$ is a mapping,
Θ is a finite set of 2×2 *tiles over the alphabet* $\Gamma \cup \{\#\}$.
The language $L(T) = \pi(LOC(\Theta))$ *is the* language defined by the TS T.
The languages over finite alphabets defined by tiling systems constitute the family REC *of* TS-recognizable languages *on* Σ.

The family REC is considered the correct answer to the quest of a natural adaptation of the class of regular word languages for pictures. Namely, like in the 1D case, REC languages can be equivalently characterized by several formalisms. We shortly remind some of them, and we mainly refer to [31] for more information.

First, one can modify the size of tiles. In this way the definition of *domino systems* arises where Θ is a finite set of 1×2 and 2×1 pictures over the alphabet $\Gamma \cup \{\#\}$ and $\mathrm{LOC}(\Theta) = \{p \in \Sigma^{**} \mid B_{1,2}(\hat{p}) \cup B_{2,1}(\hat{p}) \subseteq \Theta\}$. A local language of this type is called hv*-local language*. The family of hv-local languages is properly included in LOC.

Moreover, one can consider the connection between Wang tiles and local rules.

Lastly, a characterization of REC in term of regular string languages can be given using the so called *row-column combination* of two string languages R and C, i.e. the languages $R \oplus C$ of the pictures all whose rows, thought as strings, are in R and whose all columns, seen as string from top to bottom, are in C.

Theorem 4. *([50,21]) Let L be a picture languages. The following are equivalent.*

1. *L is TS-recognizable,*
2. *L is recognizable by some domino system,*
3. *L is Wang recognizable,*
4. *there exist two regular string languages R and C and a projection π such that $L = R \oplus C$.*

Other generalizations of local languages given in 1D case can be extended to picture languages.

[3] In the rest of the paper, we will use this notation instead of $B_{2,2}(p)$ for brevity.

Let h, k be two positive integers. Two pictures $p, r \in \Sigma^{*,*}$ are related in the equivalence relation $\cong_{h,k}$ if and only if their corresponding bordered versions have the same set of subpictures of size (h, k). A picture language is *locally testable* if it is union of $\cong_{h,k}$-equivalence classes for some positive integers $h,\ k$.

Let p be a picture. For h, k, t positive integer and for a picture $q \in (\Sigma \cup \{\perp\})^{*,*}$ of size (h, k) let $occ_p(q)$ the number of subdomains d of $\mathrm{dom}(p)$, such that $spic(p, d)$ is a translation of q and let $occ^t_p(q) = min(t, occ_p(q))$. Let $\cong^t_{h,k}$ be the equivalence relation on $\Sigma^{*,*}$ defined by $p \cong^t_{h,k} r$ if and only if $occ^t_p(q) = occ^t_r(q)$ for all $q \in (\Sigma \cup \{\perp\})^{*,*}$ of size (h, k) .

A picture language is *locally threshold testable* if it is union of $\cong^t_{h,k}$-equivalence classes for some positive integers $h,\ k$ and t.

Above picture languages are proper subclasses of REC.

Proposition 3. *The family LT of locally testable languages is properly included in the family LTT of locally threshold testable languages, which in turn is properly contained in* REC*. Moreover every language in LTT is a projection of a locally testable language.*

The family REC inherits several closure properties of regular string languages. Namely REC is closed under intersection, union, projection, row and column concatenation, closure operations, Cartesian product, and Simplot closure operator **. Moreover REC is closed under substitution of languages in REC induced by homogeneous partition sets, and also under by substitutions of languages in REC induced by the set of all homogeneous partitions of each picture [15].

However, fundamental properties of regular string languages fail in REC.

Proposition 4. REC *is not closed under complement.*
The membership problem for each language L in REC *is NP-complete.*
The emptiness and universe problems for REC *are undecidable.*

It is important to remark that in spite of its NP-completeness, the parsing problem for TS-recognizable languages can be successfully tackled encoding the problem into SAT. Namely, in [45] a recognizer/generator for pictures defined by a tiling system is implemented in a very attractive, unconventional way, by considering for a picture p and each $a \in \Sigma$ the statement $p(i, j) = a$ as a propositional variable of the SAT problem and transforming the tiling problem into a Boolean satisfiability one, then using an efficient off-the-shelf SAT-solver. The prototype is fast enough to experiment on reasonably sized samples, and has the bonus of being able to complete a partial picture, by assigning to unknown pixels some values which satisfy the picture specification.

Another difference between regular string languages and REC arises considering the following modified definition of local testability. Let h, k be two positive integers. Two pictures are related in the equivalence relation $\sim_{h,k}$ if and only if they have the same set of scattered subpictures of size (h, k).

A picture language is *piecewise locally testable* if it is union of $\sim_{h,k}$-equivalence classes for some positive integers $h,\ k$. The language CORNERS of pictures p over $\{a, b\}$ such that whenever $p(i, j) = p(i', j) = p(i, j') = b$ then also $p(i', j') = b$ is piecewise testable, but does not belong to REC.

4.2 Unambiguous and Deterministic Classes of Recognizable Picture Languages

The definition of recognizability in terms of local languages and projections is implicitly nondeterministic, moreover since REC family is not closed under complement, each attempt to overcome its non-determinism gives smaller families of languages, differently of what happens for regular string languages.

We remind the definition of unambiguous REC languages given in [30].

Definition 4. *A quadruple* $(\Sigma, \Gamma, \Theta, \pi)$ *is an* unambiguous tiling system *for a 2D language* $L \subseteq \Sigma^{*,*}$ *if and only if for any picture* $p \in L$ *there exists a unique local picture* $q \in \mathrm{LOC}(\Theta)$ *such that* $p = \pi(q)$*, i.e. the extension of* π *to a map from* $\Gamma^{*,*}$ *to* $\Sigma^{*,*}$ *is injective on* $\mathrm{LOC}(\Theta)$.
$L \in REC$ *is an* unambiguous picture language *if and only if it admits an unambiguous tiling system* $(\Sigma, \Gamma, \Theta, \pi)$.

The family of all unambiguous REC picture languages is denoted by UREC.

The language of pictures with at least two equal columns is in REC, but not in UREC. Hence

Theorem 5. *([5])* UREC *is strictly included in* REC.

The notion of determinism for tiling systems has to be referred to a direction, like in 1D case. The considered direction is one of the four main directions from a corner to another ($c2c$).

Definition 5. *A tiling system* $(\Sigma, \Gamma, \Theta, \pi)$ *is* tl2br-deterministic[4] *if for any* $\gamma_1, \gamma_2, \gamma_3 \in \Gamma \cup \{\#\}$ *and* $\sigma \in \Sigma$ *there exists at most one tile* $t \in \Theta$ *with* $t = \begin{smallmatrix} \gamma_1 & \gamma_2 \\ \gamma_3 & \gamma_4 \end{smallmatrix}$*, and* $\pi(\gamma_4) = \sigma$*. Similarly* d*-deterministic tiling systems for any direction* $d \in c2c$ *are defined.*

$L \in REC$ *is a* deterministic picture language *if and only if it admits a deterministic tiling system for some* $d \in c2c$.

The family of all deterministic REC picture languages is denoted by DREC.

DREC is properly included in UREC and there are some classes of languages that strictly separate DREC from UREC. In [3] the classes of row-UREC and col-UREC are introduced (see also [29]) where four side-to-side scanning directions, namely left-to-right ($l2r$) and vice versa ($r2l$), top-to-bottom ($t2b$) and vice versa ($b2t$), are considered.

Definition 6. *A tiling system* $(\Sigma, \Gamma, \Theta, \pi)$ *is* $l2r$-unambiguous *if for any column* $col \in \Gamma^{m,1} \cup \{\#\}^{m,1}$ *, and picture* $p \in \Sigma^{m,1}$*, there exists at most one local column* $col' \in \Gamma^{m,1}$ *such that* $\pi(col') = p$ *and* $[\![\{\#\}^{1,2} \ominus (col \oplus col') \ominus \{\#\}^{1,2}]\!] \subseteq \Theta$*. Similar properties define* d*-unambiguous tiling systems, for any side-to-side direction* d*.*

A language is column-unambiguous *if it is recognized by a* d*-unambiguous tiling system for some* $d \in \{l2r, r2l\}$ *and it is* row-unambiguous *if it is recognized by a* d*-unambiguous tiling system for some* $d \in \{t2b, b2t\}$*.* Col-UREC *is the class of column-unambiguous languages and* Row-UREC *the class of row-unambiguous languages.*

Proposition 5. *([3])* DREC $\subset$ (Col-UREC $\cap$ Row-UREC) $\subset \subset$ (Col-UREC $\cup$ Row-UREC) $\subset$ UREC.

[4] tl2br means from the top left to the bottom right corner.

More recently, Lonati and Pradella [38] introduced a new kind of determinism for tiles: given $(\Sigma, \Gamma, \Theta, \pi)$, the pre-image of a picture $p \in \Sigma^{*,*}$ is built by scanning p with a boustrophedonic strategy, that is a natural scanning strategy used by many algorithms on pictures and 2D arrays. More precisely, it starts from the top-left corner, scans the first row of p rightwards, then scans the second row leftwards, and so on.

Definition 7. *A tiling system $(\Sigma, \Gamma, \Theta, \pi)$ is snake-deterministic if Γ and Θ can be partitioned as $\Gamma = \Gamma_1 \cup \Gamma_2,\ \Theta = \Theta_1 \cup \Theta_2$, where*

- *$(\Sigma, \Gamma, \Theta_1, \pi)$ is tl2br-deterministic and for each tile $t \in \Theta_1$, $t(i,j) \in \Gamma_{3-i} \cup \{\#\}$,*
- *$(\Sigma, \Gamma, \Theta_2, \pi)$ is tr2bl-deterministic and for each tile $t \in \Theta_2$, $t(i,j) \in \Gamma_i \cup \{\#\}$ and not both $t(1,1), t(1,2)$ are $\#$.*

The closure under rotation of languages recognized by snake deterministic tiling-systems is denoted Snake-DREC.

Proposition 6. *([38])* Snake-DREC $=$ Col-UREC $\cup$ Row-UREC.

UREC is closed under projection, disjoint union, intersection and rotation, and it is not closed under row and column concatenation and under row and column closures. An open problem is whether UREC family is closed under complementation, it is also conjectured that if a REC language is not in UREC then its complement is not in REC. Some recent results in this direction by Anselmo and Madonia are included in this volume. The family DREC is closed under complement but it is not closed under union and intersection. Moreover by Definition 6 it immediately follows that it is decidable whether a given tiling system is d-deterministic for $d \in c2c$. It is also decidable whether a tiling system is column- or row-unambiguous while it is undecidable whether it is unambiguous.

We would like also remark that in [6] a new model of recognizable picture languages without frames surrounding the pictures was introduced, and the changes of properties under the framed vs unframed approaches were considered mainly focusing on determinism and unambiguity. It turns out that the frame surrounding the blocks provides additional memory that, besides enforcing size and content of the recognized pictures, produces unframed ambiguous languages that are unambiguous in REC.

4.3 Models of 2-Dimensional Finite Automata

A tile system $(\Sigma, \Gamma, \Theta, \pi)$ is a natural generalization of non deterministic finite automata to the 2D case. To underlying the analogies, Matz in [42] suggested to consider $\Gamma = \Sigma \times Q$ for some finite set Q, and the projection map π as the map $\pi(a, q) = q$ for each $a \in \Sigma,\ q \in Q$. He calls Q decoration set to point out that element of Q do not correspond to the intuition behind the word "state". Then to see the tile system as an automaton one could imagine to simultaneously "decorate" each pixel of the input picture p and to check the decorated input for local compatibility with the transition relation Θ. Also in [21] some analogies between Wang systems and finite automata were indicated. However neither tile systems nor Wang systems correspond to an effective procedure of recognition, namely when the membership of a picture p to a given REC language has to be checked, no scanning procedure of the picture p is proposed.

Several operational models have been proposed to recognize picture languages. Here we remind only four of them and we refer to [36] for a survey on different models of finite automata recognizing picture languages.

The first model, called *4-way finite automaton*, shortly 4FA, was proposed in 1967 by Blum and Hewitt [10]. It is an extension of 2-way finite automata for strings and allows the finite automaton to move in four directions: t, b, l, r (top, bottom, left, and right).

Definition 8. *([31]) A* 4FA *is a* 7*-tuple* $\mathcal{A} = (\Sigma, Q, \{t, b, l, r\}, q_0, q_a, q_r, \delta)$*, where* Σ *is the input alphabet,* Q *is the set of states,* q_0, q_a, q_r *are three distinguished states, called initial, accepting and rejecting states,* $\delta : (Q \setminus \{q_a, q_r\}) \times \Sigma \to 2^{(Q \times \{t,b,l,r\})}$ *is the transition function.*

$\mathcal{A}$ can be seen as a finite control in Q reading the input picture. If $(q', d) \in \delta(q, a)$ for some $d \in \{t, b, l, r\}$, the automaton goes from the actual state q and the actual position (i, j) with $p(i, j) = a$ to the state q', and moves the reading head by one position according to the direction d. The automaton halts when it reaches either the state q_a or the state q_r. It recognizes a picture $p \in \Sigma^{*,*}$ if starting from the position $(1, 1)$ in the state q_0, it eventually reaches the state q_a, it is not needed that it reads all the pixels of p.

The *2-dimensional on-line tessellation automaton* (2OTA) is a restricted type of 2-dimensional cellular automata, i.e. an array of cells all being in some state at any given time and operating in a sequence of discrete time steps. In 2OTA each cell changes its state depending on the top and left neighbors. This model was introduced by Inoue and Nakamura in 1977 [35]. Here we remind the definition given in [31].

Definition 9. *A* 2OTA *is a* 5*-tuple* $\mathcal{A} = (\Sigma, Q, I, F, \delta)$*, where* Σ *is the input alphabet,* Q *is the set of states,* $I \subseteq Q$*,* $F \subseteq Q$ *are the sets of initial and final states,* $\delta : Q \times Q \times \Sigma \to 2^Q$ *is the transition function.*

A run of $\mathcal{A}$ over a picture $p \in \Sigma^{*,*}$ associates a state to each position of p. At time $t = 0$ a state $q_0 \in I$ is associated to all positions of the first row and column of $\hat{p}$, then moving diagonally across the array, at time $t = k$, states are simultaneously associated to each position (i, j) of the picture with $i + j - 1 = k$, according to δ. The picture p is recognized by $\mathcal{A}$ if there is a run of $\mathcal{A}$ associating a final state to the position $(|p|_{row}, |p|_{col})$.

In 2007 Anselmo and al. [4] proposed *tiling automata* (TA for short) as an effective computational device whose transitions are given by a tiling system with a scanning strategy that uses a next-step function and a data structure to remember some of the local symbols associated to the already scanned positions of the input picture. It is evident that to handle the borders, the next-step function depends also from the size of the input picture.

Definition 10. *Let* $n, m \in \mathbb{N}$ *and* $P(n, m) = \{0, 1, \ldots, n + 1\} \times \{0, 1, \ldots, m + 1\}$.

A next-position function *for pictures is a computable partial function* $f : \mathbb{N}^4 \to \mathbb{N}^2$ *associating to a quadruple* (i, j, n, m)*, with* $(i, j) \in P(n, m)$ *a pair* $(i', j') \in P(n, m)$.

Let $v_1(n, m) = (i_0, j_0) \in P(n, m)$ *and put* $v_h(n, m) = f(v_{h-1}(n, m), n, m)$*, then the sequence* $V_{f,k}(n, m) = \{v_1(n, m), v_2(n, m) \ldots, v_{k-1}(n, m)\}$ *is called the sequence of* visited positions *by* f *at step* k *with starting position* (i_0, j_0).

A scanning strategy *is a next-position function S such that for any* $(n, m) \in \mathbb{N}^2$ *the sequence* $V_{S,(n+2)(m+2)+1}(n, m) = \{v_1(n, m), v_2(n, m) \dots, v_{(n+2)(m+2)}(n, m)\}$ *of visited positions by S at step* $(n + 2)(m + 2) + 1$ *starting from a corner position of* $P(n, m)$ *satisfies:*

1) $V_{S,(n+2)(m+2)+1}(n, m)$ *is a permutation of* $P(n, m)$*.*
2) for any $k = 2, \dots, (n + 2)(m + 2)$*, the tl- (or tr-, or bl-, or br- resp.) contiguous positions of* $v_k(n, m)$ *(when defined) are all in* $V_{S,(n+2)(m+2)+1}(n, m)$*.*

Moreover if S satisfies condition

3) for any $k = 2, \dots, (n+2)(m+2)$*,* $v_k(n, m)$ *is a contiguous position of* $v_{k-1}(n, m)$ *provided that* $v_{k-1}(n, m)$ *is an internal position, otherwise if* $v_{k-1}(n, m)$ *is an external position also* $v_k(n, m)$ *is an external position;*

it is called a continuous *scanning strategy; if S satisfies condition*

4) $v_{(n+2)(m+2)}(n, m)$ *is a corner position,*

it is called a normalized *scanning strategy.*

For each next-position function there is at most one starting corner, verifying conditions 1 and 2 of Definition 10. Moreover property 3 avoids that two non-contiguous regions of a picture are both scanned during a scanning process and together with property 4 forbids the existence of holes in the picture during the scanning process. In [4] several examples of continuous normalized scanning strategies are given, showing the richness of possibilities in 2D case, and they produce, for suitable data structures, different definitions of tiling automata. Here we introduce a formal definition of tiling automata with a scanning strategy that follows a main *tl*2*br*-directed strategy, i.e. a strategy such that for any $(n, m) \in \mathbb{N}^2$ and for any k with $1 \le k \le (n + 2)(m + 2)$ contains the (defined) *tl*-contiguous positions of $v_k(n, m)$ in the set of visited position at step k starting from position $(0, 0)$.

Definition 11. *([4]) A* tiling automaton *of type tl*2*br is a* 4*-tuple* $\mathcal{A} = (\mathcal{T}, S, D_0, \delta)$ *where* $\mathcal{T} = (\Sigma, \Gamma, \Theta, \pi)$ *is a tiling system, S is a tl*2*br-directed scanning strategy,* D_0 *is the initial content of a data structure that supports operations* $state_1(D)$*,* $state_2(D)$*,* $state_3(D)$*,* $update(D, \gamma)$*, for* $\gamma \in \Gamma \cup \{\#\}$*, and* $\delta : (\Gamma \cup \{\#\})^3 \times (\Sigma \cup \{\#\}) \to 2^{(\Gamma \cup \{\#\})}$ *is a relation such that* $\gamma_4 \in \delta(\gamma_1, \gamma_2, \gamma_3, \sigma)$ *if* $\pi(\gamma_4) = \sigma$ *and* $\begin{smallmatrix}\gamma_1 & \gamma_2\\ \gamma_3 & \gamma_4\end{smallmatrix} \in \Theta$*.*

Tiling automata of type d *for each corner to corner (c*2*c) direction* d *are similarly defined.*

The initial configuration of the tiling automaton $\mathcal{A}$ is (p, i, j, D_0), where p is a picture of size (n, m) and $(i, j) = v_1(n, m)$. From a configuration (p, h, k, D), $h, k \in \mathbb{N}$, the automaton moves to the configuration (p, h', k', D) if $S(h, k, n, m)$ is defined, $\gamma_4 \in \delta(state_1(D), state_2(D), state_3(D), p(h, k))$ for some $\gamma_4 \in \Gamma \cup \{\#\}$, $(h', k') = S(h, k, n, m)$ and D' is the content of the data structure after calling $update(D, \gamma_4)$. If $S(h, k, n, m)$ is defined, and there is no $\gamma_4 \in \Gamma \cup \{\#\}$ such that $\gamma_4 \in \delta(state_1(D)$, $state_2(D)$, $state_3(D)$, $p(h, k))$, $\mathcal{A}$ stops without accepting, while if $S(h, k, n, m)$ is not defined, $\mathcal{A}$ stops accepting p.

It is important to remind that this definition [11] refers to a tiling automaton with a given scanning strategy (of type tl2br), another scanning strategy produces a different type of tiling automaton, nevertheless the class of recognized languages is the same.

Another family of automata for dealing with REC family of languages was introduced in 2005 by Bozapalidis and Grammatikopoulou [12]. Their definition is in terms of *doubly ranked monoids*. A doubly ranked semigroup (DR-semigroup for short) is a doubly ranked set $M = (M_{m,n})$ endowed with two associative operations ⓗ : $M_{m,n} \times M_{m,n'} \to M_{m,n+n'}$, and ⓥ : $M_{m,n} \times M_{m',n} \to M_{m+m',n}$, called respectively horizontal and vertical multiplications, that are compatible to each other, i.e. $(a ⓗ a') ⓥ (b ⓗ b') = (a ⓥ b) ⓗ (a' ⓥ b')$, for all a, a', b, b' of suitable ranks. A DR-semigroup M with two sequences $e = (e_m)$ and $f = (f_n)$, with $e_m \in M_{m,0},\ f_n \in M_{0,n}$ such that $e_0 = f_0,\ e_m ⓥ e_n = e_{m+n},\ f_m ⓗ f_n = f_{m+n}$, and $e_m ⓗ a = a ⓗ e_m = a,\ f_n ⓥ b = b ⓥ f_n = B$ for all a, b of suitable rank is called a doubly ranked monoid; $e,\ f$ are called respectively the horizontal and vertical units of M. Given a doubly ranked alphabet X the free DR-monoid generated by X is called $pict(X)$.

Given a non empty set Q a *quadripolic relation* over Q of rank (m, n) is an element of $2^{Q^m \times Q^n \times Q^m \times Q^n}$ and the set of all quadripolic relations over Q of rank (m, n) is denoted by $4Rel_{m,n}(Q)$. The doubly ranked set $4Rel(Q) = (4Rel_{m,n}(Q))$ can be structured as a DR-monoid, by defining the horizontal multiplication as follows: for each $R \in 4Rel_{m,n}(Q)$ and $S \in 4Rel_{m,n'}(Q)$, $R ⓗ S = \{(w_1, w_2, w_3, w_4)| \exists u \in Q^m, v_2, v_4 \in Q^n,\ z_2, z_4 \in Q^{n'} : w_2 = v_2 z_2, w_4 = v_4 z_4,\ (w_1, v_2, u, v_4) \in R, (u, z_2, w_3, z_4) \in S\}$ and in dual way for the vertical multiplication. Let M and M' be two DR-monoids. A *morphism* from M to M' is a family of functions $\varphi_{m,n} : M_{m,n} \to M'_{m,n},\ m, n \in \mathbb{N}$, compatible with horizontal and vertical multiplication and units. Now we are in position of remind the following

Definition 12. *([12]) Let X be a finite doubly ranked set. A quadripolic automaton over X is a 5-tuple $\mathcal{A} = (Q, \delta, F_{West}, F_{Sud}, F_{Est}, F_{North})$ where Q is a finite set of states,$F_{West}, F_{Sud}, F_{Est}, F_{North}$ are subsets of Q, called the four poles of acceptance for $\mathcal{A}$, δ is a family of maps $\delta_{m,n} : X_{m,n} \to 4Rel_{m,n}(Q)$.*

Let $\overline{\delta} : pict(X) \to 4Rel(Q)$ be the morphism of DR-monoids uniquely extending δ and let $F_{m,n} = F^m_{West} \times F^n_{Sud} \times F^m_{Est} \times F^n_{North}$. A picture $p \in pict_{m,n}(X)$ is accepted by $\mathcal{A}$ if and only if $\overline{\delta}_{m,n}(p) \cap F_{m,n} \neq \emptyset$. $L(QA)$ denotes the family of languages recognized by a quadripolic automaton. It is clear that quadripolic automata are related to the description of REC via labeled Wang tiles. This allows an algebraic approach to recognizable languages that is presented in a paper by Bozapalidis and Grammatikopoulou included in the present volume.

The following theorem clarifies the reason behind the name REC given to the family of TS-recognizable languages.

Theorem 6. *([31,4,12]) Let L be a picture language. The following are equivalent:*

1. $L \in$ REC*;*
2. $L \in \mathcal{L}$(2OTA)*;*
3. $L \in \mathcal{L}$(TA)*;*
4. $L \in \mathcal{L}$(QA).

On the other hand, the family of 4-way automata is not enough powerful to define REC.

Proposition 7. *([31]) $\mathcal{L}(\mathrm{4FA})$ is strictly included in* REC. *Moreover $\mathcal{L}(\mathrm{4FA})$ is not closed under row and column concatenation and closure operations, but it is closed under union and intersection.*

Some attempts of increasing the power of 4-way automata by endowing them with a bounded queue or a bounded stack did not produce satisfactory results [7].

The unambiguous versions of on-line tessellation (2-UOTA, for short) and tilings automata (UTA, for short), i.e. 2-dimensional on-line tessellation and tilings automata such that for any picture there is at most one accepting computation, recognize UREC family.

Automata described in Definitions 8, 9, 11 admit also their deterministic counterparts. In the sequel 4DFA, 2DOTA, DTA denote the families of deterministic 4-way, 2-dimensional on-line tessellation and tiling automata. They are less powerful than the corresponding non-deterministic automata. In the deterministic case the family of languages recognized by tiling automata depends on the chosen scanning strategy, so $\mathcal{L}(\mathrm{DTA})$ denotes the set of all languages recognized by a deterministic d-tiling automata for each scanning strategy in any direction $d \in c2c$ and DREC $= \mathcal{L}(\mathrm{DTA})$. Moreover the family $\mathcal{L}(\mathrm{4DFA})$ of languages recognized by a deterministic 4-way automaton and the family $\mathcal{L}(\mathrm{2DOTA})$ recognized by some automaton in 2OTA are not comparable as shown by examples in [35].

4.4 Regular Expressions

One of the main results on regular string languages is Kleene's theorem that characterizes the family of languages recognized by finite automata in term of regular expressions. Such expressions can be analogously defined for picture languages.

Definition 13. *([31]) A* regular expression *on the alphabet Σ is defined recursively as follows:*

1. *$\emptyset$ and each $a \in \Sigma$ are regular expressions;*
2. *if α and β are regular expressions, also $\alpha \cup \beta$, $\alpha \cap \beta$, α^C, $\alpha \oplus \beta$, $\alpha \ominus \beta$, $\alpha^{*\oplus}$, $\alpha^{*\ominus}$ are so.*

Each regular expression over Σ denotes a picture language: $\emptyset$ and $a \in \Sigma$ denote respectively the empty language and the language formed by the unique picture of size $(1,1)$ with $p(1,1) = a$, $\alpha\cup\beta$, $\alpha\cap\beta$, $\alpha\oplus\beta$, $\alpha\ominus\beta$, denote the union, intersection, row and column concatenation of languages α and β; α^C, $\alpha^{*\oplus}$, $\alpha^{*\ominus}$ denote the complement, and Kleene's closures of language α.

A language $L \subseteq \Sigma^{*,*}$ is regular *if it is generated by a regular expression over Σ.*

It is an immediate consequence of the non closure of REC under complement that REC does not coincide with the class $\mathcal{L}(\mathrm{RE})$ of the languages denoted by regular expressions. Then it is quite natural to consider restricted sets of operators to be iteratively applied starting from empty language and languages formed by a single picture of size (1,1).

In [31] the following sets of operators are considered: $\mathcal{R}_1 = \{\cup, \cap, \oplus, \ominus, ^{*\oplus}, ^{*\ominus}\}$, $\mathcal{R}_2 = \{\cup, \cap, ^C, \oplus, \ominus\}$ and in [42] the set $\mathcal{R}_3 = \{\cup, \oplus, \ominus, ^{*\oplus}, ^{*\ominus}\}$ was added.

Regular expressions containing only operators in $\mathcal{R}_1$ are called *complement-free* and $\mathcal{L}(\mathrm{CFRE})$ is the class of languages generated by complement-free regular expressions. Regular expressions using only operators in $\mathcal{R}_2$ are called *star-free* and $\mathcal{L}(\mathrm{SFRE})$ is the class of languages they denote. $\mathcal{L}(\mathrm{CFRE})$ properly contains the family of hv-local languages, hence giving a Kleene-like theorem for picture languages modulo projection.

Theorem 7. *A picture language L is in* REC *if and only if it is the projection of a language in* $\mathcal{L}(\mathrm{CFRE})$.

Also the class $\mathcal{L}(\mathrm{SFRE})$, being closed under complement, does not coincide with REC. In [41] Matz proved that the language CORNERS belongs to $\mathcal{L}(\mathrm{SFRE})$ whereas it is not in REC so showing that $\mathcal{L}(\mathrm{SFRE})$, and more in general the family of languages denoted by regular expressions, and REC are incomparable. This results answers to some open problems in [31], Section 8.4. In [55] it is proved that the language CROSS of all pictures over $\{a, b\}$ containing $\begin{smallmatrix} a & b & a \\ b & b & b \\ a & b & a \end{smallmatrix}$ as subpicture is piecewise testable but does not belongs to $\mathcal{L}(\mathrm{SFRE})$ and obviously $\mathcal{L}(\mathrm{SFRE})$ is not contained in the family PT of piecewise locally testable languages because the inclusion fails for the analog string languages.

The family of languages denoted by a regular expression containing only operators in $\mathcal{R}_3$, but $\cap$, is called REG in [42]. It is a proper subfamily of $\mathcal{L}(\mathrm{CFRE})$ and, in spite of its low expressive power, some arguments (simplicity, polynomial membership problem, polynomial emptiness problem) suggesting that it could be a better analog of regular string languages, are sketched.

In [39] Matz proposed a more powerful type of regular expressions for picture languages, called *regular expressions with operators*. For instance, he considered the column concatenation of a given picture r to the left and to the right like individual objects: $r\obar$ and $\obar r$. He call this kind of objects operators and allows iteration over combinations of operators. If unrestricted, these operators can be combined to generate languages not in REC (e.g $ab((a\obar)(\obar b))^*$ denotes the language $\{a^i b^i | i > 0\}$); but under the natural constraints that an operator working on the left (resp. top) is never juxtaposed, united or intersected with an operator working on the right (resp. bottom), he showed that the power of these expressions does not exceed the family REC and is enough to denote the language of square. It remains an open problem whether regular expressions with operators exhaust REC-family.

More recently Anselmo and al. [2] proposed some new operations on pictures and picture languages with the aim of looking for a homogeneous notion of regular expressions that could extend more naturally the concept of regular expression of 1D languages. They focus on regular expressions on one-letter alphabet but, as they remark, this is a necessary and meaningful case to start since it corresponds to study the "shapes" of pictures: if a picture language is in REC then necessarily the language of its shapes is in REC. First they introduced *diagonal concatenation* of pictures, that starting from two pictures p, q over a one-letter alphabet $\{a\}$, respectively of size (n, m) and (n', m'), produces the picture over $\{a\}$ of size $(n + n', m + m')$, so enabling to express some relationship between the dimensions of the pictures. The regular expressions allowing only union, diagonal concatenation and its closure as operators, and the empty set, empty picture, and empty row and column as atomic expressions denote a family of languages over $\{a\}$, called $\mathcal{L}(\mathrm{D})$. It coincides with the languages of a-pictures whose dimensions belongs to some rational relation or equivalently can be recognized by some 4FA automaton that moves only right and down. $\mathcal{L}(\mathrm{D})$ properly contains the class of languages over one letter alphabet belonging to $\mathcal{L}(\mathrm{CFRE})$ and is closed under intersection and complement. Then they consider the family of languages over one letter alphabet denoted by regular expressions whose operator set contains union, column, row and diagonal concatenations and their closures, getting again a family properly included in REC. So, in the attempt of capture all the shapes allowed by 1D REC languages, they defined new types of iteration operations, called *advanced stars*, that result much more powerful than the classical stars and also seem to constitute a more reasonable approach to the general case because the definitions of advanced stars admit obvious generalizations on larger alphabets.

4.5 Logic Formulas

Let Σ be a finite set and consider the signature $\{S_1, S_2, \{P_a\}_{a\in\Sigma}\}$, where P_a are unary and $S_i,\ i = 1, 2$ binary relation symbols. Monadic second-order (shortly MSO) formulas on this signature using first-order variables $x, y, z, \ldots$ and second order variables $X, Y, Z \ldots$, are inductively built from atomic formulas $x = y$, $S_1(x, y)$, $S_2(x, y)$, $P_a(x), X(x)$ using Boolean connectives and quantifiers applicable to first and second order variables. A MSO formula where no second order variable is quantified is called a first-order (FO) formula. An existential monadic second order (EMSO) is a formula of the form $\exists X_1 \exists X_2 \ldots \exists X_r \phi$ where ϕ is a first-order formula.

A picture p over Σ can be represented by the structure $\underline{p} = (\mathrm{dom}(p),\ S_{p,1}, S_{p,2},$ $\{P_{p,a}\}_{a\in\Sigma})$ where $\mathrm{dom}(p) = \{1, \ldots, |p|_{row}\} \times \{1, \ldots, |p|_{col}\}$, $S_{p,1}, S_{p,2} \subset \mathrm{dom}(p) \times \mathrm{dom}(p)$ are two successor relations defined by $(i, j)S_{p,1}(i + 1, j)$ for $1 \le i < |p|_{row}$, $1 \le i \le |p|_{col}$ and $(i, j)S_{p,2}(i, j + 1)$ for $1 \le i \le |p|_{row}, 1 \le j < |p|_{col}$, $|\Sigma|$ and $P_{p,a} = \{(i, j)|p(i, j) = a\}$, with $a \in \Sigma$ gives the set of positions labeled by a.

Let $\phi(X_1, X_2, ..., X_t)$ be a formula where at most $X_1, X_2, ..., X_t$ are free variables and let $Q_1, Q_2, \ldots, Q_t$ be subsets of $\mathrm{dom}(p)$. Consider the interpretation with domain $\mathrm{dom}(p)$, where first order variables are positions and second order variables are sets of positions in $\mathrm{dom}(p)$, and in particular Q_i is the interpretation of X_i for $1 \le i \le t$, the predicates $S_1(x, y), S_2(x, y), P_a(x), X(x)$ are seen as $(x, y) \in S_{p,1},\ (x, y) \in S_{p,2},\ x \in P_{p,a}, x \in X$. Then

$$(\underline{p}, Q_1, Q_2, \ldots, Q_t) \models \phi(X_1, X_2, ..., X_t)$$

means that p satisfies ϕ in the above interpretation.

A sentence is a formula without free variables. Let ϕ a sentence on the signature $\{S_1, S_2, \{P_a\}_{a\in\Sigma}\}$, the picture language L defined by ϕ is the set of all pictures p such that $\underline{p} \models \phi$. A characterization of REC in term of logic formulas is the following

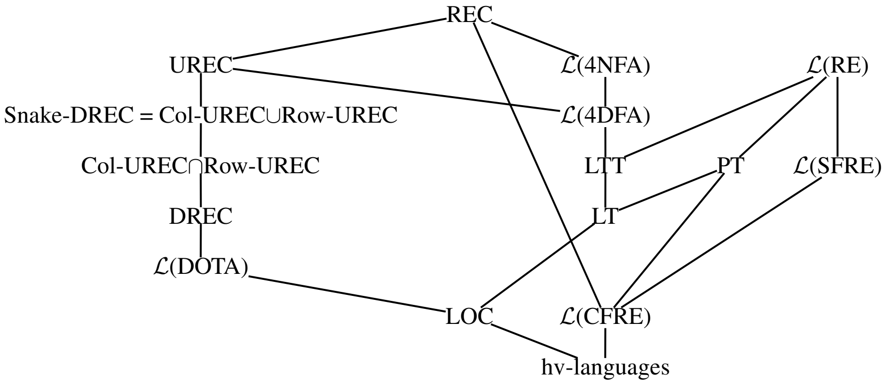
Theorem 8. *A picture language L is in* REC *if and only if it is definable by an* EMSO *formula in the signature* $\{S_1, S_2, \{P_a\}_{a\in\Sigma}\}$.

Matz in [41] enforces the above result showing that every picture language in REC is definable by an EMSO formula of the form $\exists X \phi(X)$ where ϕ is a first order formula.

Also, the families of languages with some kind of local testability admit logical characterization. In fact, a language is locally threshold testable iff it is definable by a first-order formula in the signature $\{S_1, S_2, \{P_a\}_{a\in\Sigma}\}$ ([32]), while is locally testable if and only if it is definable by a first-order formula in the signature $\{S_1, S_2, \{P_a\}_{a\in\Sigma}, left, right, top, bottom\}$, where $left, right, top, bottom$ are unary predicates saying that a position is at the respective border [40].

4.6 Summary

Inclusions of the families introduced in above sections are represented by the following diagram:



4.7 Necessary Conditions for Recognizability

An useful tool to prove whether a language is recognizable in 1D case is pumping lemma for regular languages. An analog of pumping lemma can be stated for languages in REC provided that they contain pictures whose number of columns (rows) is sufficiently larger than the number of rows (columns).

Lemma 1. *(Horizontal iteration lemma, [31]) Let $L \in REC$. Then there is a function $\varphi : \mathbb{N} \to \mathbb{N}$ such that if $p \in L$ and $|p|_{col} > \varphi(|p|_{row})$, there exist some pictures x, y, q with $|x \oplus q|_{col} \leq \varphi(|p|_{row})$ and $|y|_{col} > 1$ so that $p = x \oplus q \oplus y$ and for all $i \geq 0$ $x \oplus q^{i\oplus} \oplus y \in L$. Moreover, $\varphi(n) \leq |\Gamma|^n$ for any local alphabet used to represent L.*

Analogously can be stated a *vertical iteration lemma*.

Another necessary condition for a language being in REC uses the notion of syntactic equivalence modulo a language L. For a language $L \in \Sigma^{*,*}$ two isometric pictures $p,\ q$ are called *syntactically equivalent modulo* L (in symbols, $p \approx_L q$) if for all $x_1, x_2, y_1, y_2 \in \Sigma^{*,*}$ of suitable sizes, $x_1 \oplus (y_1 \ominus p \ominus y_2) \oplus x_2 \in L$ if and only if $x_1 \oplus (y_1 \ominus q \ominus y_2) \oplus x_2 \in L$. The function $f_L(|p|_{row}, |p|_{col})$ gives the number of $\approx_L$-equivalence classes in $\Sigma^{*,*}$ of size $(|p|_{row}, |p|_{col})$.

Lemma 2. *(Syntactic equivalence lemma, [31]) Let $L \in REC$. Then there exists a positive integer c such that $f_L(n, m) \leq c^{n+m}$ for all positive integers n, m.*

Lemma 3. *([40]) Let $L \in REC$ over Σ. For each positive integer n let $\{M_n\}$ be a sequence such that*

1. $M_n \subseteq \Sigma^{n,+} \times \Sigma^{n,+}$;
2. $\forall (p,q) \in M_n, p \oplus q \in L$;
3. $\forall (p,q), (p',q') \in M_n, \{p \oplus q', p' \oplus q\} \nsubseteq L$.

Then $|M_n|$ is $2^{O(n)}$.

The question of the existence of some language not in REC for which the above lemma fails to prove the non recognizability was posed. The language of squares over $\{a,b\}$ with as many a's as b's was proposed as candidate. However, from a result in [49], it follows that the above language is recognizable.

4.8 Recognizable Picture Languages on One-Letter Alphabet

Pictures over a one-letter alphabet, as already remarked in Section 4.4, are a special but meaningful case to consider. Only the shape of the picture is relevant, whence a unary picture is simply identified by a pair of positive integers representing its size. So a picture language over one letter alphabet can be studied looking to the corresponding set of integer pairs, and the definition of recognizability can be extended from languages to functions from $\mathbb{N}$ to $\mathbb{N}$ saying that a function $f : \mathbb{N} \to \mathbb{N}$ is recognizable if its associate language $L_f = \{p \in \{a\}^{*,*} \mid |p_{col}| = f(|p_{row}|)\}$ is recognizable. In [31] it is shown that recognizable functions cannot grow quicker than an exponential function or slower than a logarithmic one.

In 2007 Bertoni and al. [9] presented REC languages over one-letter alphabet via a characterization of strings encoding the pictures of the language. Namely they associate to each picture $p \in \{a\}^{*,*}$ the string $\phi(p) \in \{a,h,v\}^*$ defined as follows:
$\phi(p) = a^{|p|_{row}} h a^{|p|_{col} - |p|_{row} - 1}$, if $|p|_{row} < |p|_{col}$;
$\phi(p) = a^{|p|_{row}}$, if $|p|_{row} = |p|_{col}$;
$\phi(p) = a^{|p|_{col}} v a^{|p|_{row} - |p|_{col} - 1}$, if $|p|_{col} < |p|_{row}$.

Definition of ϕ obviously extends to languages by putting $\phi(L) = \{\phi(p) | \, p \in L\} \subseteq \{a,h,v\}^*$, for $L \subseteq \{a\}^{*,*}$.

Theorem 9. *Let $L \subseteq \{a\}^{*,*}$. L is in* REC *if and only if $\phi(L)$ is a string language that can be recognized by a 1-tape non-deterministic Turing machines working, for any input $x \in \{a,h,v\}^*$, within $|x|$ space and executing at most ${}_a|x|$ head reversals, where ${}_a|x|$ is the length of the longest prefix of x in a^+.*

Languages on one-letter alphabet were considered also for several of the afore-defined subclasses of REC languages.

5 Grammars for Generating Pictures

We did not consider generating grammars for REC family: in literature, 2D grammars are mainly considered as a way to introduce an analog of CF string languages, and several different models of grammars were proposed. There are essentially two main categories of picture grammars: one category imposes the constraint that the left and right parts of a rewriting rule must be isometric arrays, so overcoming the inherent problem of shearing (which pops up while substituting a subpicture in a host picture). The other one relies with several variations on notions of operations among pictures. More recently, to overcome the shearing problem and in general problems arising from the non flexibility of pixels in a picture, a picture deformation theory was introduced by Bozapalidis in [11]. A family of pixels $x^{(r,s)}$ is associated to any pixel x, called the (r,s)-deformed pixels of x, where r, s range over a semiring A. The deformation $p^{(r,s)}$ of a picture p is obtained by replacing all pixels of p by their (r,s)-deformations and is a picture where only the dimensions of p are changed.

In the following section a grammar model specified by a set of rewriting rules is presented with isometric rules. Then some properties of the model that seem to support the claim that the model is a good generalization of CF 1D languages are stated, and some relations with other well-known models of picture grammars are discussed.

5.1 Tile Grammars

Tile grammars were defined in [18] with the name of tile rewriting grammars, then a normal form for those grammars was given in [14]. Here we use the normal form as basic definition because it is simpler to handle.

First we need to introduce the notion of *strong homogeneous partition*. We say that the domain of a picture p admits a strong homogeneous partition if there is a homogeneous partition of $dom(p)$ so that subpictures of p associated to contiguous subdomains have different labels. It is clear that each picture admits at most one strong homogeneous partition.

Definition 14. *A* Tile grammar (TG) *is a 4-tuple* (Σ, N, S, R)*, where* Σ *is the* terminal *alphabet,* N *is a set of* nonterminal *symbols,* $S \in N$ *is the* starting symbol*,* R *is a set of* rules*. Let* $A \in N$*. There are two kinds of rules:*

$$\text{\textit{Fixed size:}} \quad A \to t, \quad \text{\textit{where} } t \in \Sigma; \tag{1}$$

$$\text{\textit{Variable size:}} \quad A \to \omega, \quad \omega \text{ \textit{is a set of tiles over} } N \cup \{\#\}. \tag{2}$$

The nonterminal symbol A in the left part of a variable size rule denotes an A-homogeneous picture. The right part of a variable size rule is a picture of a local language over nonterminal symbols. Thus a variable size rule is a scheme defining a possibly unbounded number of isometric pairs: left picture, right picture. In addition there are rules whose right part is a single terminal.

Notice that tile grammars may be viewed as extending CF grammars from one to two dimensions: the argument that such grammars in one dimension are essentially CF grammars allowing a local regular expression in right parts of rules is in [18].

The derivation process of a picture starts from a S-picture. Picture derivation is a relation between partitioned pictures.

Definition 15. *Consider a grammar* $G = (\Sigma, N, S, R)$*, let* $p, p' \in (\Sigma \cup N)^{h,k}$ *be pictures of identical size. Let* $\pi = \{d_1, \ldots, d_n\}$ *be homogeneous partition of* $\mathrm{dom}(p)$*. We say that* (p', π') derives in one step *from* (p, π)*, written*

$$(p, \pi) \Rightarrow_G (p', \pi')$$

iff, for some $A \in N$ and for some rule $\rho \in R$ with left part A, there exists in π an A-homogeneous subdomain $d_i = (x, y; x', y')$, called application area, *such that:*

- *p' is obtained substituting* $\mathrm{spic}(p, d_i)$ *in p with a picture s, defined as follows:*
  - *if ρ is of type (1), then $s = t$;*
  - *if ρ is of type (2), then $s \in LOC(\omega)$ and admits a strong homogeneous partition $\Pi(s)$*
- *π' is a homogeneous partition of* $\mathrm{dom}(p)$ *into the subdomains*

$$(\pi \setminus \{d_i\}) \cup \mathrm{transl}_{(x-1,y-1)}(\Pi(s))$$

where $\mathrm{transl}_{(x-1,y-1)}(\Pi(s))$ *is the translation by displacement $(x-1, y-1)$ (intuitively, the position of d_i in p) of the subdomains of $\Pi(s)$.*

We say that (q, π') derives from *(p, π) in n steps, written $(p, \pi) \xRightarrow{n}_G (q, \pi')$, iff $p = q$ and $\pi = \pi'$, when $n = 0$, or there are a picture r and a homogeneous partition π'' such that $(p, \pi) \xRightarrow{n-1}_G (r, \pi'')$ and $(r, \pi'') \Rightarrow_G (q, \pi')$. We use the abbreviation $(p, \pi) \xRightarrow{*}_G (q, \pi')$ for a derivation with a finite number of steps.*

Roughly speaking at each step of the derivation, an A-homogeneous subpicture is replaced with an isometric picture of the local language, defined by the right part of a rule $A \to \ldots$ that admits a strong homogeneous partition. The process terminates when all nonterminals have been eliminated from the current picture.

Definition 16. *The* picture language *defined by a grammar G (written $L(G)$) is the set of $p \in \Sigma^{+,+}$ such that*

$$\left(S^{|p|}, \mathrm{dom}(p)\right) \overset{*}{\Rightarrow}_G (p, \mathcal{I}),$$

where $\mathcal{I}$ denotes the partition of $\mathrm{dom}(p)$ *defined by single pixels. For short we also write $S \overset{*}{\Rightarrow}_G p$. $L(TG)$ denote the family of languages generated by some tile grammar.*

Example 1. One row and one column of b's.

The set of pictures such that there is one row and one column (both not at the border) that hold b's, and the remainder of the picture is filled with a's is defined by the tile grammar (we remind the reader that $[\![p]\!]$ stands for the set of all subpictures of size (2,2) of p):

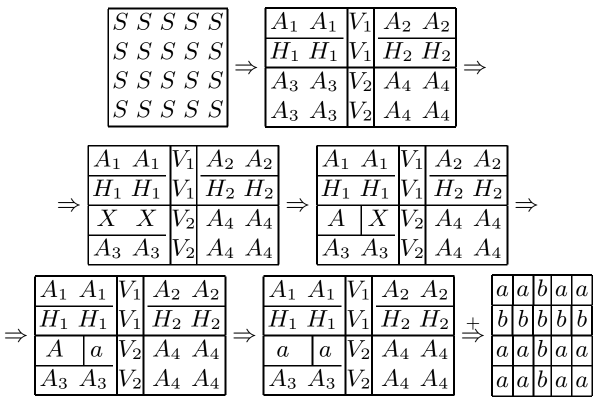
$$S \to \left[\!\!\left[\begin{matrix} \# & \# & \# & \# & \# & \# & \# \\ \# & A_1 & A_1 & V_1 & A_2 & A_2 & \# \\ \# & A_1 & A_1 & V_1 & A_2 & A_2 & \# \\ \# & H_1 & H_1 & V_1 & H_2 & H_2 & \# \\ \# & A_3 & A_3 & V_2 & A_4 & A_4 & \# \\ \# & A_3 & A_3 & V_2 & A_4 & A_4 & \# \\ \# & \# & \# & \# & \# & \# & \# \end{matrix}\right]\!\!\right]$$

$$A_i \to \left[\!\!\left[\begin{matrix} \# & \# & \# & \# \\ \# & X & X & \# \\ \# & A_i & A_i & \# \\ \# & A_i & A_i & \# \\ \# & \# & \# & \# \end{matrix}\right]\!\!\right] \mid \left[\!\!\left[\begin{matrix} \# & \# & \# & \# \\ \# & X & X & \# \\ \# & \# & \# & \# \end{matrix}\right]\!\!\right], \text{ for } 1 \le i \le 4$$

$$X \to \left[\!\left[\begin{array}{ccccc} \# & \# & \# & \# & \# \\ \# & A & X & X & \# \\ \# & \# & \# & \# & \# \end{array}\right]\!\right] \mid a; \;\; H_i \to \left[\!\left[\begin{array}{ccccc} \# & \# & \# & \# & \# \\ \# & B & H_i & H_i & \# \\ \# & \# & \# & \# & \# \end{array}\right]\!\right] \mid b, \text{ for } 1 \leq i \leq 2$$

$$A \to a; \;\; B \to b; \;\; V_i \to \left[\!\left[\begin{array}{ccc} \# & \# & \# \\ \# & B & \# \\ \# & V_i & \# \\ \# & V_i & \# \\ \# & \# & \# \end{array}\right]\!\right] \mid b, \text{ for } 1 \leq i \leq 2.$$

Here is an example of derivation, with partitions outlined for better readability:



The family $\mathcal{L}(\mathrm{TG})$ of TG-languages is closed w.r.t. union, column/row concatenation, column/row closure operations, rotation, alphabetic mapping ([18]).

We remark that this family as well as all families presented in the sequel, which exactly define CF string languages if restricted to one dimension, are not closed w.r.t. intersection and complement. Namely, since they are all closed w.r.t. union, the same arguments as string CF grammars can be used to prove these properties.

5.2 Tile Grammars and Tiling Systems

Proposition 8. *([18])* $\mathrm{REC} \subset \mathcal{L}(\mathrm{TG})$.

In fact, for a tiling system $T = (\Sigma, \Gamma, \Theta, \pi)$, it is quite easy to define a TG T' such that $L(T') = L(T)$. Informally, the idea is to take the tile-set Θ and add two markers, e.g. $\{b, w\}$ in a "chequerboard-like" fashion to build up a tile-set suitable for the right part of the variable size starting rule; other straightforward fixed size rules are used to encode the projection π. We show the construction on a simple example. The interested reader may refer to [18] for details.

Example 2. The following TS defines square pictures of a's.

$$\Theta = \left[\!\left[\begin{array}{cccccc} \# & \# & \# & \# & \# & \# \\ \# & 1 & 0 & 0 & 0 & \# \\ \# & 0 & 1 & 0 & 0 & \# \\ \# & 0 & 0 & 1 & 0 & \# \\ \# & 0 & 0 & 0 & 1 & \# \\ \# & \# & \# & \# & \# & \# \end{array}\right]\!\right], \;\; \pi(0) = a, \;\; \pi(1) = a$$

An equivalent tile grammar is the following:

$$S \to \left[\!\!\left[\begin{array}{cccccc} \# & \# & \# & \# & \# & \# \\ \# & 1_b & 0_w & 0_b & 0_w & \# \\ \# & 0_w & 1_b & 0_w & 0_b & \# \\ \# & 0_b & 0_w & 1_b & 0_w & \# \\ \# & 0_w & 0_b & 0_w & 1_b & \# \\ \# & \# & \# & \# & \# & \# \end{array}\right]\!\!\right] \cup \left[\!\!\left[\begin{array}{cccccc} \# & \# & \# & \# & \# & \# \\ \# & 1_w & 0_b & 0_w & 0_b & \# \\ \# & 0_b & 1_w & 0_b & 0_w & \# \\ \# & 0_w & 0_b & 1_w & 0_b & \# \\ \# & 0_b & 0_w & 0_b & 1_w & \# \\ \# & \# & \# & \# & \# & \# \end{array}\right]\!\!\right]$$

$$1_w \to a,\ \ 1_b \to a,\ \ 0_w \to a,\ \ 0_b \to a.$$

To see that the inclusion is proper, one can restrict to string languages.

From above it immediately follows that the parsing problem for TG-languages is NP-hard, but in [44] it is proved that it is in NP, so

Proposition 9. *The parsing problem for $\mathcal{L}(\mathrm{TG})$ is NP-complete.*

In [15] some restrictions on tile grammars guaranteeing that the generated language is in REC are given. These restrictions are the analog of the restrictions that one dimensional CF grammars have to satisfy in order of defining regular languages.

Let $G = (\Sigma, N, S, R)$ be a tile grammar, a non terminal $A \in N$ is *non recursive* if and only if there is no derivation of the form $(A, \Pi) \Rightarrow^* (q, \Pi')$ with $spic(p, d) \in A^{+,+}$ for some subdomain d of Π'. Two non terminals $A_1, A_2 \in N$ are *mutually recursive* if and only if for each $i = 1, 2$ there are derivations $(A_i, \Pi_i) \Rightarrow^* (q_i, \Pi'_i)$ with $spic(q_i, d_i) \in \{A_{3-i}\}^{*,*}$ for some subdomain d_i of Π'_i. A tile grammar all whose non terminal are non recursive is called *non recursive tile grammar*.

Proposition 10. *([15]) The family of languages generated by non-recursive tile grammars coincides with* REC.

One can define a 2D analogous of a 1D grammar where self-embedding is never allowed.

Definition 17. *A tile grammar $G = (\Sigma, N, S, R)$ is a corner grammar if there exists a partition of N in sets N_1, N_2, N_3, N_4, and $\overline{N}$ such that:*

1. *$\overline{N}$ is the set of non-recursive nonterminals of G;*
2. *for every $i \neq j$, $1 \leq i, j \leq 4$, for each $A \in N_i$, $B \in N_j$, A and B are not mutually recursive;*
3. *for every i, $1 \leq i \leq 4$, for each $A \in N_i$ if $A \Rightarrow^* p$ then p has a subpicture at i-th corner in $N_i^{*,*}$ and the remainder pixels in $\Sigma \cup (N \setminus N_i)$, where the i-th corner is lt for $i = 1$, rt for $i = 2$, rb for $i = 3$, lb for $i = 4$.*

In other words, in every non-corner position of a picture, only terminals or those nonterminals that cannot give rise to recursions are allowed, while disjoint (possibly empty) nonterminal alphabets are considered for the four corners. Clearly, a non-recursive tile grammar is a special case of corner grammar (with $N_i = \emptyset$ for every i, $1 \leq i \leq 4$). A corner grammar is also a generalization of right-linear or left-linear grammars for the 1D case.

Proposition 11. *([15]) The family of languages generated by a corner grammars coincides with* REC.

Notice that checking whether a tile grammar is recursive or if it is a corner grammar is not decidable.

5.3 Regional Tile Grammars

We now introduce the central concepts of *regional language*. The adjective “regional” is a metaphor of geographical political maps, such that different regions are filled with different colors. Of course, regions are rectangles.

Definition 18. *A homogeneous partition is* regional *(HR) iff distinct subdomains have distinct labels. We will call a picture p* regional *if it admits a HR partition.*
A language is regional *if all its pictures are so.*

Definition 19. *([14]) A* regional tile grammar (RTG) *is a tile grammar (see Definition 14), in which every variable size rule $A \to \omega$ is such that* $\mathrm{LOC}(\omega)$ *is a* regional language.

We note that Example 1 is regional, while the picture language presented in Example 2 is not.

For languages generated by regional tile grammars a parsing algorithm generalizing the CKY algorithm is given. A subpicture is conveniently identified by its subdomain as in original algorithm a substring is identified by the positions of its first and last characters.

Theorem 10. *([14]) The parsing problem for* RTG *has polynomial time complexity.*

Analyzing the algorithm, one derives that the complexity of parsing for a picture of size (n, m) is $\mathcal{O}(\mu m^4 n^4)$ where constant μ depends on parameters of the grammar. The property of having polynomial time complexity for picture recognition, together with the remark that pictures with palindromic rules are not in REC immediately give the following results:

Proposition 12. *([14])* $\mathcal{L}(\mathrm{RTG}) \subset \mathcal{L}(\mathrm{TG})$. $\mathcal{L}(\mathrm{RTG})$ *is incomparable with* REC.

Moreover, the polynomial parsing united with the rather simple and intuitively pleasing form of RTG rules, should make them a worth addition to the series of array rewriting grammar models conceived in past years. In the sequel we prove or recall some inclusion relations between grammar models and corresponding language families.

5.4 Průša’s Grammars

The following definitions are taken and adapted from [46,47].

Definition 20. *A 2D CF Průša grammar (*PG*) is a tuple (Σ, N, R, S), where Σ is the finite set of* terminal *symbols, disjoint from the set N of* nonterminal *symbols, $S \in N$ is the* start *symbol, and $R \subseteq N \times (N \cup \Sigma)^{+,+}$ is the set of* rules.

Definition 21. *Let $G = (\Sigma, N, R, S)$ be a PG. We define a picture language $L(G, A)$ over Σ for every $A \in N$. The definition is given by the following recursive descriptions:*

(i) If $A \to w$ is in R, and $w \in \Sigma^{+,+}$, then $w \in L(G, A)$.
(ii) Let $A \to w$ be a production in R, $w = (N \cup \Sigma)^{(m,n)}$, for some $m, n \geq 1$. Let $p_{i,j}$, with $1 \leq i \leq m$, $1 \leq j \leq n$, be pictures such that:

1. *if* $w(i,j) \in \Sigma$, *then* $p_{i,j} = w(i,j)$;
2. *if* $w(i,j) \in N$, *then* $p_{i,j} \in L(G, w(i,j))$;
3. *for* $1 \leq i < m$, $1 \leq j \leq n$, $|p_{i,j}|_{col} = |p_{i+1,j}|_{col}$; *let* $P_k = p_{k,1} \oplus p_{k,2} \oplus \cdots \oplus p_{k,n}$, *and* $P = P_1 \ominus P_2 \ominus \cdots \ominus P_m$.

Then $P \in L(G, A)$.

The set $L(G, A)$ *contains just all the pictures that can be obtained by applying a finite sequence of rules (i) and (ii). The language* $L(G)$ *generated by the grammar* G *is defined as the language* $L(G, S)$.

Informally, rules can either be terminal rules, which are used to generate the pictures which constitute the right parts of rules, or have a picture as right part. In this latter case, the right part is seen as a "grid", where nonterminals can be replaced by other pictures, but maintaining its grid-like structure.

Example 3. The following grammar generates the language of pictures with one row and one column of b's in a background of a's (see Example 1).

$$S \to \begin{matrix} A & V & A \\ H & b & H \\ A & V & A \end{matrix}, \quad A \to AM \mid M, \quad M \to \begin{matrix} a \\ M \end{matrix} \mid a,$$

$$V \to \begin{matrix} b \\ V \end{matrix} \mid b, \quad H \to bH \mid b.$$

It is easy to see that Průša grammars admit a Nonterminal Normal Form:

Definition 22. *A Průša grammar* $G = (\Sigma, N, R, S)$, *is in* Nonterminal Normal Form *iff every rule in* R *has the form either* $A \to t$, *or* $A \to w$, *where* $A \in N$, $w \in N^{+,+}$, *and* $t \in \Sigma$.

To compare Průša's grammars with tile grammars, we must note that the two models are different in their derivations. Tile grammars start from a picture made of S's having a fixed size, and being every derivation step isometric, the resulting picture, if any, has the same size. On the other hand, PG's start from a single S symbol, and then "grow" the picture derivation step by derivation step, obtaining, if any, a usually larger picture.

Proposition 13. *([14])* $\mathcal{L}(\mathrm{PG}) \subset \mathcal{L}(\mathrm{RTG})$.

Remark 1. Essentially, Průša grammars can be seen as RTG's with the additional constraint that tiles used in the right parts of rules must not have one of these forms:

$$\begin{pmatrix} A & B \\ C & C \end{pmatrix}, \begin{pmatrix} A & C \\ B & C \end{pmatrix}, \begin{pmatrix} C & C \\ A & B \end{pmatrix}, \begin{pmatrix} C & A \\ C & B \end{pmatrix}$$

with A, B, C all different.

5.5 Kolam Grammars

Průša introduced his model with the attempt of gaining some generative capacity with respect the class of Kolam grammars. This class of grammars has been introduced by Siromoney et al. [52] under the name “Array grammars”, later renamed “Kolam Array grammars” in order to avoid confusion with Rosenfeld’s homonymous model. Much later Matz reinvented the same model [39] (considering only CF rules). Here the historical name, CF Kolam grammars (CFKG) is kept, the more succinct definition of Matz is used.

Definition 23. *A* sentential form *over an alphabet V is a non-empty well-parenthesized expression using the two concatenation operators, $\ominus$ and $\oplus$, and symbols taken from V. $\mathcal{SF}(V)$ denotes the set of all sentential forms over V. A sentential form ϕ defines either one picture over V denoted by $(\!|\phi|\!)$, or none.*

For example, $\phi_1 = ((a \oplus b) \ominus (b \oplus a)) \in \mathcal{SF}(\{a, b\})$ and $(\!|\phi_1|\!)$ is the picture $\begin{smallmatrix} a & b \\ b & a \end{smallmatrix}$. On the other hand $\phi_2 = ((a \oplus b) \ominus a)$ denotes no picture, since the two arguments of the $\ominus$ operator have different column numbers.

CF Kolam grammars are defined analogously to CF string grammars. Derivation is similar: a sentential form over terminal and nonterminal symbols results from the preceding one by replacing a nonterminal with some corresponding right hand side of a rule. The end of a derivation is reached when the sentential form does not contain any nonterminal symbols. If this resulting form denotes a picture, then that picture is generated by the grammar.

Definition 24. *A* CF Kolam grammar (CFKG) *is a tuple $G = (\Sigma, N, R, S)$, where Σ is the finite set of* terminal *symbols, disjoint from the set N of* nonterminal *symbols; $S \in N$ is the* starting *symbol; and $R \subseteq N \times \mathcal{SF}(N \cup \Sigma)$ is the set of* rules. *A rule $(A, \phi) \in R$ will be written as $A \rightarrow \phi$.*

For a grammar G, we define the *derivation* relation $\Rightarrow_G$ on the sentential forms $\mathcal{SF}(N \cup \Sigma)$ by $\psi_1 \Rightarrow_G \psi_2$ iff there is some rule $A \rightarrow \phi$, such that ψ_2 results from ψ_1 by replacing an occurrence of A by ϕ. As usual, $\overset{*}{\Rightarrow}_G$ denotes the reflexive and transitive closure of $\Rightarrow_G$. Notice that the derivation thus defined rewrites strings, not pictures.

From the derived sentential form, one obtains the denoted picture. The picture language generated by G is the set

$$L(G) = \{(\!|\psi|\!) \mid \psi \in \mathcal{SF}(\Sigma), S \overset{*}{\Rightarrow}_G \psi\}.$$

With a slight abuse of notation, we will often write $A \overset{*}{\Rightarrow}_G p$, with $A \in N, p \in \Sigma^{*,*}$, instead of $\exists \phi : A \overset{*}{\Rightarrow}_G \phi, (\!|\phi|\!) = p$.

CF Kolam grammars admit a normal form with exactly two or zero nonterminals in the right part of a rule [39].

Definition 25. *A grammar $G = (\Sigma, N, R, S)$, is in* Chomsky Normal Form *iff every rule in R has the form either $A \rightarrow t$, or $A \rightarrow B \ominus C$, or $A \rightarrow B \oplus C$, where $A, B, C \in N$, and $t \in \Sigma$.*

We know from [39] that for every CFKG G, if $L(G)$ does not contain the empty picture, there exists a CFKG G' in Chomsky Normal Form, such that $L(G) = L(G')$. Also, the classical algorithm to translate a string grammar into Chomsky Normal Form can be easily adapted to CFKGs.

Example 4. The following Chomsky Normal Form grammar G defines the set of pictures such that each column is a palindrome:
$S \to V \oplus S \mid A_1 \ominus A_2 \mid B_1 \ominus B_2 \mid a \mid b;$
$V \to A_1 \ominus A_2 \mid B_1 \ominus B_2 \mid a \mid b;$
$A_2 \to V \ominus A_1 \mid a;$
$B_2 \to V \ominus B_1 \mid b;$
$A_1 \to a;$
$B_1 \to b.$

Proposition 14. *([14])* $\mathcal{L}(\mathrm{CFKG}) \subset \mathcal{L}(\mathrm{PG})$.

Namely, rules $A \to B \oplus C$ of a CF Kolam grammar G in CNF are equivalent to RTG rules:

$$A \to \left[\!\!\left[\begin{array}{cccccc} \# & \# & \# & \# & \# & \# \\ \# & B & B & C & C & \# \\ \# & B & B & C & C & \# \\ \# & \# & \# & \# & \# & \# \end{array} \right]\!\!\right]$$

and similarly an equivalent form can be stated for rules $A \to B \ominus C$. This is compatible with the constraint of Průša grammars given in Remark 1 and so for each CF Kolam grammar there exists an equivalent Průša's grammar. The inclusion is proper because the language of Example 1 cannot be generated by a CF Kolam grammar.

The time complexity of picture recognition problem for CF Kolam grammars in CNF has been recently proved [19] to be $O(m^2 n^2 (m+n))$. The significant difference with the time complexity of parsing for RTG grammars depends on the fact that in the right part of a rule of a CF Kolam grammars in CNF there are at most two distinct nonterminals. So, checking if a rule is applicable has complexity which is linear with respect to the picture width or height.

5.6 Context-Free Matrix Grammars

The early model of CF Matrix grammars [51] is a very limited kind of CF Kolam grammars. The following definition is taken and adapted from [48].

Definition 26. *Let $M = (G, G')$ where $G = (N, T, P, S)$ is a string grammar, where N is the set of nonterminals, P is a set of productions, S is the starting symbol, $T = \{A_1, A_2, \cdots, A_k\}$, $G' = \{G_1, G_2, \cdots, G_k\}$ where each A_i is the starting symbol of string grammar G_i. The grammars in G' are defined over an alphabet Σ, which is the alphabet of M. A grammar M is said to be a* CF Matrix Grammar *(*CFMG*) iff G and all G_i are CF grammars.*

Let $p \in \Sigma^{+,+}$, $p = c_1 \oplus c_2 \oplus \cdots \oplus c_n$. $p \in L(M)$ iff there exists a string $A_{x_1} A_{x_n} \cdots A_{x_n} \in L(G)$ such that every column c_j, seen as a string, is in $L(G_{x_j}), 1 \leq j \leq n$. The string $A_{x_1} A_{x_n} \cdots A_{x_n}$ is said to be an intermediate *string deriving p.*

If G and G_i for all i, $1 \leq i \leq k$ are regular grammars then M is called a 2D right linear grammar.

Informally, the grammar G is used to generate an horizontal string of starting symbols for the "vertical grammars" $G_j, 1 \leq j \leq k$. Then, the vertical grammars are used to generate the columns of the picture. If every column has the same height, then the generated picture is defined, and is in $L(M)$.

It is trivial to show that the class of CFMG languages is a proper subset of CF Kolam languages. Intuitively, it is possible to consider the string sub-grammars G, and G_j, of a CF Matrix grammar M, all in Chomsky Normal Form. This means that we can define an equivalent M' CF Kolam grammar, in which rules corresponding to those of G use only the $\obar$ operator, while rules corresponding to those of G_j use only the $\ominus$ operator.

Also, it is easy to adapt classical string parsing methods to Matrix grammars, see e.g. [48].

It is also well known that the family of languages generated by 2D right linear grammars is strictly included in the family of languages recognized by deterministic 4-way finite automata.

5.7 Grid Grammars

Grid grammars are an interesting formalism defined by Drewes [22,23]. Grid grammars are based on an extension of quadtrees [28], in which the number of "quadrants" is not limited to four, but can be k^2, with $k \geq 2$ (thus forming a square "grid").

Following the tradition of quadtrees, and differently from the other formalisms presented here, grid grammars generate pictures which are seen as set of points on the "unit square" delimited by the points (0,0), (0,1), (1,0), (1,1) of the Cartesian plane.

To compare such model, in which a picture is in the unit square and mono-chromatic (i.e. black and white), with the ones presented in this work, we introduce a different but basically compatible formalization, in which the generated pictures are square arrays of symbols, and the terminal alphabet is not limited to black and white. Our approach ([44]) is similar to the one used for Kolam grammars.

Definition 27. *A* sentential form *over an alphabet V is either a symbol $a \in V$, or $[t_{1,1}, \ldots, t_{1,k}, \ldots, t_{k,1}, \ldots, t_{k,k}]$, with $k \geq 2$, and every $t_{i,j}$ being a sentential form. $\mathcal{SF}(V)$ denotes the set of all sentential forms over V.*

A sentential form ϕ defines a set of pictures $\llparenthesis\phi\rrparenthesis$:

- *$\llparenthesis a\rrparenthesis$, with $a \in V$, represents the set $\{a\}^{(n,n)}, n \geq 1$ of all a-homogeneous square pictures;*
- *$\llparenthesis[t_{1,1}, \ldots, t_{1,k}, \ldots, t_{k,1}, \ldots, t_{k,k}]\rrparenthesis$, represents the set of all square grid pictures where every $\llparenthesis t_{i,j}\rrparenthesis$ has the same size $n \times n$, for $n \geq 1$, and $\llparenthesis t_{1,1}\rrparenthesis$ is at the bottom-left corner, ..., $\llparenthesis t_{1,k}\rrparenthesis$ is at the bottom right corner, ..., and $\llparenthesis t_{k,k}\rrparenthesis$ is at the top right corner.*

For example, consider the sentential form $\phi = [[a, b, [a, b, b, a], c], a, B, [b, a, a, b]]$, the smallest picture in $\llparenthesis\phi\rrparenthesis$ is

$$\begin{array}{cccccccc}
B & B & B & B & a & a & b & b \\
B & B & B & B & a & a & b & b \\
B & B & B & B & b & b & a & a \\
B & B & B & B & b & b & a & a \\
b & a & c & c & a & a & a & a \\
a & b & c & c & a & a & a & a \\
a & a & b & b & a & a & a & a \\
a & a & b & b & a & a & a & a
\end{array}$$

Definition 28. *A* Grid grammar (GG) *is a tuple $G = (\Sigma, N, R, S)$, where Σ is the finite set of* terminal *symbols, disjoint from the set N of* nonterminal *symbols; $S \in N$ is the* starting *symbol; and $R \subseteq N \times \mathcal{SF}(N \cup \Sigma)$ is the set of* rules. *A rule $(A, \phi) \in R$ will be written as $A \to \phi$.*

For a grammar G, we define the *derivation* relation $\Rightarrow_G$ on the sentential forms $\mathcal{SF}(N \cup \Sigma)$ by $\psi_1 \Rightarrow_G \psi_2$ iff there is some rule $A \to \phi$, such that ψ_2 results from ψ_1 by replacing an occurrence of A by ϕ.

From the derived sentential form, one then obtains the denoted picture. The picture language generated by G is the set

$$L(G) = \{\text{the smallest picture in } (\!|\psi|\!) \mid \psi \in \mathcal{SF}(\Sigma), S \overset{*}{\Rightarrow}_G \psi\}.$$

With a slight abuse of notation, we will often write $A \overset{*}{\Rightarrow}_G p$, with $A \in N, p \in \Sigma^{*,*}$, instead of $\exists \phi : A \overset{*}{\Rightarrow}_G \phi, (\!|\phi|\!) = p$.

In literature, parameter k is fixed for a Grid grammar G, i.e. all the right parts of rules are either terminal or $k \times k$ grids. This constraint could be relaxed, by allowing different k for different rules: the results that are shown next still hold for this generalization.

It is trivial to see that grid grammars admit a Nonterminal Normal Form:

Definition 29. *A grid grammar $G = (\Sigma, N, R, S)$, is in* Nonterminal Normal Form *(NNF) iff every rule in R has the form either $A \to t$, or $A \to [B_{1,1}, \ldots, B_{1,k}, \ldots, B_{k,1}, \ldots, B_{k,k}]$, where $A, B_{i,j} \in N$, and $t \in \Sigma$.*

Example 5. Here is a simple example of a grid grammar in NNF.

$$S \to [S, B, S, B, B, B, S, B, S], \;\; S \to a, \;\; B \to b.$$

The generated language is that of "recursive" crosses of b's in a field of a's.

An example picture:

$$\begin{array}{ccccccccc}
a & b & a & b & b & b & a & a & a \\
b & b & b & b & b & b & a & a & a \\
a & b & a & b & b & b & a & a & a \\
b & b & b & b & b & b & b & b & b \\
b & b & b & b & b & b & b & b & b \\
b & b & b & b & b & b & b & b & b \\
a & b & a & b & b & b & a & a & a \\
b & b & b & b & b & b & a & a & a \\
a & b & a & b & b & b & a & a & a
\end{array}$$

First, we note that this is the only 2D grammatical model presented in this paper which cannot generate string languages, since all the generated pictures, if any, have the same number of rows and columns by definition.

It is easy to see that the class of languages generated by grid grammars are a proper subset of the one of CF Kolam grammars.

Proposition 15. *([44]) $\mathcal{L}(\text{GG}) \subset \mathcal{L}(\text{CFKG})$. $\mathcal{L}(\text{CFMG})$ and $\mathcal{L}(\text{GG})$ are incomparable.*

By definition, grid grammars can generate only square pictures and on the other hand, it is impossible to define CF Matrix grammars generating only square pictures.

5.8 Summary

We finish with a synopsis of the previous language family inclusions.

$$\begin{tikzcd}
 & & \mathcal{L}(\mathrm{TG}) \arrow[dll, dash] \arrow[drr, dash] & & \\
\mathrm{REC} \arrow[ddd, dash] & & & & \mathcal{L}(\mathrm{RTG}) \arrow[d, dash] \\
 & & & & \mathcal{L}(\mathrm{PG}) \arrow[d, dash] \\
 & & & & \mathcal{L}(\mathrm{CFKG}) \arrow[dll, dash] \arrow[d, dash] \\
\mathcal{L}(\mathrm{4DFA}) \arrow[drr, dash] & & \mathcal{L}(\mathrm{GG}) & & \mathcal{L}(\mathrm{CFMG}) \arrow[dll, dash] \\
 & & \mathcal{L}(\mathrm{2RGL}) & &
\end{tikzcd}$$

6 Conclusion

First of all we want to remark that there are several different ways to generate or recognize picture languages that are not considered in this survey, e.g. [16], [43].

Since REC is a robust notion, we believe that it is a necessary starting point for a tutorial on picture languages. If one assumes that REC is the right answer to the quest of a analog for regular string languages then, to maintain hierarchy, TG grammars is the notion corresponding to context-free grammars. This is why we choose to describe this model among the others.

RTG preserves some nice properties of context free string languages and includes several well known models usually introduced as a generalization of context free grammars. So, a question naturally arises: if RTG is the right model for generating context free picture languages, what about the right model for regular string languages? Some criticisms on the fact that REC recognizes a too wide class to be considered the right model in spite of its robustness was posed for instance in [42]. It could be interesting to consider which languages are defined by non recursive RTG grammars in order to verify whether that family can also be proposed as the analog of regular string languages.

Moreover, few attention was paid to study the generalization to two dimensions of push-down automata. For instance how can be defined automata recognizing all the families of languages generated by grammars described in this survey? And finally are there more promising grammatical approaches to "context-free" picture languages?

In conclusion, in our opinion the very idea of defining a Chomsky's hierarchy analogous, moving from one to two dimensions, is probably doomed to partial unsuccess. 2D structures and formalisms, albeit maintaining some similarities with their 1D counterparts, often exhibit very different formal properties and issues which are not present or trivial in string languages.

Acknowledgments. We thank Achille Frigeri for his valuable comments.

References

1. Alstes, A.: Wang tiles for image and texture generation. Technical Report 49432R, Helsinki University of Technology (2004)
2. Anselmo, M., Giammarresi, D., Madonia, M.: New operations and regular expressions for two-dimensional languages over one-letter alphabet. Theor. Comput. Sci. 304, 408–431 (2005)
3. Anselmo, M., Giammarresi, D., Madonia, M.: From determinism to non-determinism in recognizable two-dimensional languages. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 36–47. Springer, Heidelberg (2007)
4. Anselmo, M., Giammarresi, D., Madonia, M.: Tiling automaton: A computational model for recognizable two-dimensional languages. In: Holub, J., Žďárek, J. (eds.) CIAA 2007. LNCS, vol. 4783, pp. 290–302. Springer, Heidelberg (2007)
5. Anselmo, M., Giammarresi, D., Madonia, M., Restivo, A.: Unambiguous recognizable two-dimensional languages. Theoretical Informatics and Applications 40(2), 277–293 (2006)
6. Anselmo, M., Jonoska, N., Madonia, M.: Framed versus unframed two-dimensional languages. In: Nielsen, M., Kucera, A., Miltersen, P.B., Palamidessi, C., Tuma, P., Valencia, F.D. (eds.) SOFSEM 2009. LNCS, vol. 5404, pp. 79–92. Springer, Heidelberg (2009)
7. Anselmo, M., Madonia, M.: Simulating two-dimensional recognizability by pushdown and queue automata. In: Farré, J., Litovsky, I., Schmitz, S. (eds.) CIAA 2005. LNCS, vol. 3845, pp. 43–53. Springer, Heidelberg (2006)
8. Berger, R.: The Undecidability of the Domino Problem. Memoirs of the AMS, vol. (66). AMS, Providence (1966)
9. Bertoni, A., Goldwurm, M., Lonati, V.: On the complexity of unary tiling-recognizable picture languages. Fundamenta Informaticae 91(2), 231–249 (2009)
10. Blum, M., Hewitt, C.: Automata on a 2-dimensional tape. In: FOCS 1967, pp. 155–160. IEEE Computer Society Press, Los Alamitos (1967)
11. Bozapalidis, S.: Picture deformation. Acta Inf. 45(1), 1–31 (2008)
12. Bozapalidis, S., Grammatikopoulou, A.: Recognizable picture series. Journal of Automata, Languages and Combinatorics 10(2-3), 159–183 (2005)
13. Brun, Y.: Solving NP-complete problems in the tile assembly model. Theor. Comput. Sci. 395(1), 31–46 (2008)
14. Cherubini, A., Crespi Reghizzi, S., Pradella, M.: Regional languages and tiling: A unifying approach to picture grammars. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 253–264. Springer, Heidelberg (2008)
15. Cherubini, A., Crespi Reghizzi, S., Pradella, M., San Pietro, P.: Picture languages: Tiling systems versus tile rewriting grammars. Theor. Comput. Sci. 356(1-2), 90–103 (2006)
16. Choffrut, C., Durak, B.: Collage of two-dimensional words. Theor. Comput. Sci. 340(1), 364–380 (2005)
17. Cohen, M.F., Shade, J., Hiller, S., Deussen, O.: Wang tiles for image and texture generation. ACM Trans. Graph. 22(3), 287–294 (2003)
18. Crespi Reghizzi, S., Pradella, M.: Tile rewriting grammars and picture languages. Theor. Comput. Sci. 340(2), 257–272 (2005)
19. Crespi Reghizzi, S., Pradella, M.: A CKY parser for picture grammars. Information Processing Letters 105(6), 213–217 (2008)
20. Culik II., K.: An aperiodic set of 13 Wang tiles. Discrete Mathematics 160, 245–251 (1996)
21. de Prophetis, L., Varricchio, S.: Recognizability of rectangular pictures by Wang systems. Journal of Automata, Languages and Combinatorics 2(4), 269–288 (1997)
22. Drewes, F.: Language theoretic and algorithmic properties of d-dimensional collages and patterns in a grid. J. Comput. Syst. Sci. 53(1), 33–66 (1996)

23. Drewes, F., Ewert, S., Klempien-Hinrichs, R., Kreowski, H.-J.: Computing raster images from grid picture grammars. Journal of Automata, Languages and Combinatorics 8(3), 499–519 (2003)
24. Durand, B.: Tilings and quasiperiodicity. Theor. Comput. Sci. 221, 61–75 (1999)
25. Durand, B.: De la logique aux pavages. Theor. Comput. Sci. 281, 311–324 (2002)
26. Durand, B., Levin, L.A., Shen, A.: Local rules and global order, or aperiodic tilings. Math. Intelligencer 27(1), 64–68 (2005)
27. Eigen, S., Navarro, J., Prasad, V.S.: An aperiodic tiling using a dynamical system and Beatty sequences. In: Hasselblatt, B. (ed.) Dynamics, Ergodic Theory and Geometry. Mathematical Sciences Research Institute Publications, vol. (54), pp. 223–241. Cambridge Univ. Press, Cambridge (2007)
28. Finkel, R.A., Bentley, J.L.: Quad trees: A data structure for retrieval on composite keys. Acta Inf. 4, 1–9 (1974)
29. Giammarresi, D.: Tiling recognizable two-dimensional languages. In: Bozapalidis, S., Rahonis, G. (eds.) CAI 2007. LNCS, vol. 4728, pp. 75–86. Springer, Heidelberg (2007)
30. Giammarresi, D., Restivo, A.: Recognizable picture languages. International Journal of Pattern Recognition and Artificial Intelligence 6(2-3), 241–256 (1992)
31. Giammarresi, D., Restivo, A.: Two-dimensional languages. In: Handbook of formal languages. beyond words, vol. 3, pp. 215–267. Springer, Berlin (1997)
32. Giammarresi, D., Restivo, A., Seibert, S., Thomas, W.: Monadic second-order logic over rectangular pictures and recognizability by tiling systems. Inf. Comput. 125(1), 32–45 (1996)
33. Grünbaum, B., Shephard, G.C.: Tilings and Patterns. W.H. Freeman and Company, New York (1987)
34. Gurevich, Y.S., Koriakov, I.O.: Remarks on Berger's paper on the domino problem. Siberian Mathematical Journal 13(2), 319–321 (1972)
35. Inoue, K., Nakamura, A.: Some properties of two-dimensional on-line tessellation acceptors. Information Sciences 13, 95–121 (1977)
36. Inoue, K., Takanami, I.: A survey of two-dimensional automata theory. Information Sciences 55(1-3), 99–121 (1991)
37. Kari, J.: A small aperiodic set of Wang tiles. Discrete Mathematics 160(1-3), 259–264 (1996)
38. Lonati, V., Pradella, M.: Snake-deterministic tiling systems. MFCS 2009 (to appear, 2009)
39. Matz, O.: Regular expressions and context-free grammars for picture languages. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 283–294. Springer, Heidelberg (1997)
40. Matz, O.: On piecewise testable, starfree, and recognizable picture languages. In: Nivat, M. (ed.) FOSSACS 1998. LNCS, vol. 1378, pp. 203–210. Springer, Heidelberg (1998)
41. Matz, O.: One quantifier will do in existential monadic second-order logic over pictures. In: Brim, L., Gruska, J., Zlatuška, J. (eds.) MFCS 1998. LNCS, vol. 1450, pp. 751–759. Springer, Heidelberg (1998)
42. Matz, O.: Recognizable vs. Regular picture languages. In: Bozapalidis, S., Rahonis, G. (eds.) CAI 2007. LNCS, vol. 4728, pp. 112–121. Springer, Heidelberg (2007)
43. Nivat, M., Saoudi, A., Subramanian, K.G., Siromoney, R., Rajkumar Dare, V.: Puzzle grammars and context-free array grammars. International Journal of Pattern Recognition and Artificial Intelligence 5, 663–676 (1991)
44. Pradella, M., Cherubini, A., Crespi Reghizzi, S.: A unifying approach to context-free picture languages (submitted, 2009)
45. Pradella, M., Crespi Reghizzi, S.: A SAT-based parser and completer for pictures specified by tiling. Pattern Recognition 41(2), 555–566 (2008)
46. Průša, D.: Two-dimensional context-free grammars. In: Andrejková, G., Krajči, S. (eds.) ITAT 2001, pp. 27–40 (2001)

47. Průša, D.: Two-dimensional Languages. Phd thesis, Charles University, Faculty of Mathematics and Physics, Prague (2004)
48. Radhakrishnan, V., Chakaravarthy, V.T., Krithivasan, K.: Pattern matching in matrix grammars. Journal of Automata, Languages and Combinatorics 3(1), 59–76 (1998)
49. Robinson, R.M.: Undecidability and nonperiodicity for tilings of the plane. Inventiones Mathematicae 12, 177–209 (1971)
50. Simplot, D.: A characterization of recognizable picture languages by tilings by finite sets. Theor. Comput. Sci. 218(2), 297–323 (1999)
51. Siromoney, G., Siromoney, R., Krithivasan, K.: Abstract families of matrices and picture languages. Computer Graphics and Image Processing 1(3), 284–307 (1972)
52. Siromoney, G., Siromoney, R., Krithivasan, K.: Picture languages with array rewriting rules. Information and Control 23(5), 447–470 (1973)
53. Wang, H.: Proving theorems by pattern recognition II. Bell Systems Technical Journal 40, 1–42 (1961)
54. Wang, H.: Dominoes and the AEA case of the decision problems. In: Mathematical theory of automata: Symposium on Mathematical Theory of Automata 1963. Brooklyn Polytechnic Institute. Microwave Research Institute. Symposia series, vol. 12, pp. 23–55. Polythechnic Press (1963)
55. Wilke, T.: Star-free picture expressions are strictly weaker than first-order logic. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 347–357. Springer, Heidelberg (1997)
56. Winfree, E., Liu, F., Wenzler, L.A., Seeman, N.C.: Design and self-assembly of two-dimensional DNA crystals. Nature 394, 439–544 (1998)

Process Algebra: An Algebraic Theory of Concurrency

Wan Fokkink

Vrije Universiteit Amsterdam, Department of Theoretical Computer Science,
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
wanf@cs.vu.nl

Abstract. This tutorial provides an overview of the process algebra ACP.

1 Introduction

The term "process algebra" was coined in 1982 by Jan Bergstra and Jan Willem Klop, originally in the sense of universal algebra, to refer to a structure satisfying a particular set of axioms. Nowadays it is used in a more general sense for algebraic approaches to describe and study concurrent processes. In the late 70's, Robin Milner and Tony Hoare largely independently developed the process algebras CSS and CSP, respectively. In the early 80's, Bergstra and Klop developed a third process algebra called ACP.

System behaviour generally consists of processes and data. Processes are the control mechanisms for the manipulation of data. While processes are dynamic and active, data is static and passive. System behaviour tends to be composed of several processes that are executed concurrently, where these processes exchange data in order to influence each other's behaviour. Fundamental to process algebra is a parallel operator, to break down systems into their concurrent components. A set of equations is imposed to derive whether two terms are behaviourally equivalent. In this framework, non-trivial properties of systems can be established in a rigorous and elegant fashion. For example, it may be possible to equate an implementation of a system to the specification of its required input/output relation. A variety of automated tools have been developed to facilitate the derivation of such properties in a process algebraic framework.

Abstract data types (see, e.g., [5]) offer a framework in which also the data can be specified by means of equations. μCRL [10] is a specification language, supported by verification tools, that combines process algebra with equational specification of data types. In this tutorial we will however mainly focus on processes.

Applications of process algebra exist in diverse fields such as safety-critical systems, network protocols, and biology. In the educational vein, process algebra has been recognised to teach skills to deal with complex concurrent systems, by representing and reasoning about such systems in a mathematically clear and precise manner.

Recommended textbooks are [15] for CCS, [17] for CSP, and [8] for ACP. Jos Baeten [2] presented a detailed account on the history of process algebra. Here I will focus on the process algebra ACP; this tutorial is based on [8].

This tutorial is structured as follows. Section 2 explains the general framework. Section 3 introduces the basic process algebra BPA. Section 4 extends the framework with parallelism, communication and encapsulation. Section 5 adds recursion to express infinite behaviour. Section 6 introduces the silent step τ, and abstraction operators to hide internal behaviour. Finally, Section 7 presents a process algebraic specification and verification of the Alternating Bit Protocol.

2 The General Framework

Process graphs As starting point, we assume that system behaviour is represented as a process graph. It basically consists of a set of nodes together with a set of labelled edges between these nodes. A node represents a system state, while a labelled edge represents a transition from one system state to the next. That is, if the process graph contains an edge $s \xrightarrow{a} s'$, then the process graph can evolve from state s into state s' by the execution of action a. One state is selected to be the root state, i.e., the initial state of the process.

Behavioural equivalences. The states in process graphs are distinguished by some behavioural equivalence. For example, such an equivalence may relate two process graphs if and only if their root states can execute exactly the same strings of actions. This tutorial focuses on bisimilarity, which is the finest of all known process equivalences. Bisimilarity requires not only that two process graphs can execute the same strings of actions, but also that they have the same branching structure. Bisimilarity is widely recognised as a well-suited semantic notion when reasoning about concurrent processes.

Process algebra terms. For the purpose of mathematical reasoning it is often convenient to represent process graphs algebraically in the form of terms. Process algebra focuses on the specification and manipulation of process terms as induced by a collection of operator symbols. This symbolic notation facilitates manipulation by a computer. Most process algebras contain basic operators to build finite processes, communication operators to express concurrency, and some notion of recursion to capture infinite behaviour. Moreover, it is convenient to introduce two special constants: the deadlock enables us to force actions into communication, while the silent step allows us to abstract away from internal computations.

Structural operational semantics. Transition rules, which are inductive proof rules, provide each process term with its intended process graph (see, e.g., [1]). We are going to present the process algebra ACP$_\tau$ with recursion in several steps, starting from the basic process algebra BPA. With every extension we need to check that it is conservative, meaning that the transition rules for the new operators do not influence the behaviour of the "old" process algebra terms. This can be checked by inspecting the syntactic form of the transition rules. Moreover, the process algebraic operators should be a congruence with respect to bisimilarity, meaning that if two process terms are bisimilar, then they are also bisimilar under any context. Again this can be checked by inspecting the syntactic form of the transition rules.

Equational logic. The crux of process algebra is that it imposes an equational logic on process terms that is sound and complete. Soundness means that if two process terms can be equated then their process graphs are behaviourally equivalent. Vice versa, completeness means that if two process terms have behaviourally equivalent process graphs, then they can be equated.

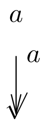
3 Basic Process Algebra

We start with describing a basic process algebra, denoted by BPA.

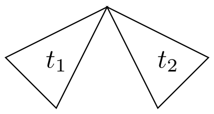
3.1 Syntax of BPA

The core for process algebra consists of the following operators.

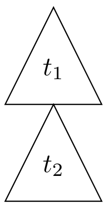
- First of all, we assume a non-empty set A of (atomic) actions, representing indivisible behaviour (such as reading a datum, or sending a datum). Each atomic action a is a constant that can execute itself, after which it terminates successfully:



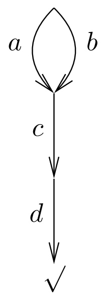
  The predicate $\xrightarrow{a}\ \surd$ represents successful termination after the execution of action a.
- Moreover, we assume a binary operator $+$ called alternative composition. The term $t_1 + t_2$ represents the process that executes the behaviour of either t_1 or t_2. In other words, the process graph of $t_1 + t_2$ is obtained by joining the process graphs of t_1 and t_2 at their root states:



- Finally, we assume a binary operator $\cdot$ called sequential composition. The term $t_1 \cdot t_2$ represents the process that executes first the behaviour of t_1, and then the behaviour of t_2. In other words, the process graph of $t_1 \cdot t_2$ is obtained by replacing each successful termination $s \xrightarrow{a} \surd$ in in the process graph of t_1 by a transition $s \xrightarrow{a} s'$, where s' is the root of the process graph of t_2:



Example 1. Let a, b, c and d be actions. The basic process term $((a+b)\cdot c)\cdot d$ represents the following process graph, with the root state presented at the top:



Each finite process graph can be represented by a process term that is built from the set A of atomic actions, $+$, and $\cdot$. Such terms are called basic process terms, and the collection of all basic process terms is called basic process algebra, abbreviated to BPA.

3.2 Transition Rules of BPA

We have provided a syntax for basic process terms, together with some intuition for the process graph that belongs to such a term. This relationship has to be made formal in order for it to become really meaningful. For this purpose we apply structural operational semantics. This involves giving a collection of transition rules, which define transitions $t \xrightarrow{a} t'$ to express that term t can evolve into term t' by the execution of action a, and predicates $t \xrightarrow{a} \surd$ to express that term t can terminate successfully by the execution of action a.

Table 1 presents the transition rules that constitute the structural operational semantics of BPA. The variables x, x', y and y' in the transition rules range over the collection of basic process terms, while v ranges over the set A of atomic actions.

The transition rules of BPA provide each basic process term with a process graph, according to the intuition that was presented in the previous section:

- the first transition rule says that each atomic action v can terminate successfully by executing itself;
- the next four transition rules express that $t + t'$ behaves as either t or t';

Table 1. Transition rules of BPA

$$\frac{}{v \xrightarrow{v} \surd}$$

$$\frac{x \xrightarrow{v} \surd}{x+y \xrightarrow{v} \surd} \qquad \frac{x \xrightarrow{v} x'}{x+y \xrightarrow{v} x'} \qquad \frac{y \xrightarrow{v} \surd}{x+y \xrightarrow{v} \surd} \qquad \frac{y \xrightarrow{v} y'}{x+y \xrightarrow{v} y'}$$

$$\frac{x \xrightarrow{v} \surd}{x\cdot y \xrightarrow{v} y} \qquad \frac{x \xrightarrow{v} x'}{x\cdot y \xrightarrow{v} x'\cdot y}$$

– the last two transition rules express that $t{\cdot}t'$ executes t until successful termination, after which it proceeds to execute t'.

Example 2. The transition rules in Table 1 provide the basic process term $((a+b){\cdot}c){\cdot}d$ with the following process graph (cf. Example 1):

$$\begin{tikzcd}
((a+b)\cdot c)\cdot d \arrow[d, bend right, "a"'] \arrow[d, bend left, "b"] \\
c\cdot d \arrow[d, "c"'] \\
d \arrow[d, "d"'] \\
\surd
\end{tikzcd}$$

For instance, the transition $((a+b){\cdot}c){\cdot}d \xrightarrow{b} c{\cdot}d$ can be proved from the transition rules in Table 1 as follows:

$$\begin{array}{cl}
b \xrightarrow{b} \surd & \left(\dfrac{}{v \xrightarrow{v} \surd},\quad v := b\right) \\
\cline{1-1}
a + b \xrightarrow{b} \surd & \left(\dfrac{y \xrightarrow{v} \surd}{x + y \xrightarrow{v} \surd},\ v := b,\ x := a,\ y := b\right) \\
\cline{1-1}
(a+b){\cdot}c \xrightarrow{b} c & \left(\dfrac{x \xrightarrow{v} \surd}{x{\cdot}y \xrightarrow{v} y},\quad v := b,\ x := a+b,\ y := c\right) \\
\cline{1-1}
((a+b){\cdot}c){\cdot}d \xrightarrow{b} c{\cdot}d & \left(\dfrac{x \xrightarrow{v} x'}{x{\cdot}y \xrightarrow{v} x'{\cdot}y},\ v := b,\ x := (a+b){\cdot}c,\ x' := c,\ y := d\right)
\end{array}$$

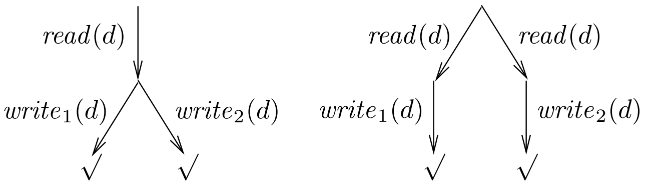
At the right-hand side, the transition rules are displayed that are applied in the consecutive proof steps, together with the substitutions that are applied to them.

From now on, as binding convention we assume that $\cdot$ binds stronger than $+$. For example, $a{\cdot}b + a{\cdot}c$ represents $(a{\cdot}b) + (a{\cdot}c)$. Occurrences of $\cdot$ are often omitted from process terms; that is, st denotes $s{\cdot}t$.

3.3 Bisimulation

In the previous section, each basic process term has been provided with a process graph using structural operational semantics. Processes have been studied since the early 60's, first to settle questions in natural languages, later on to study the semantics of programming languages. These studies originally focused on so-called trace equivalence, in which two processes are said to be equivalent if they can execute exactly the same strings of actions. However, for system behaviour this equivalence is not always satisfactory, which is shown by the following example.

Example 3. Consider the two processes below:



The first process reads datum d, and then decides whether it writes d on disc 1 or on disc 2. The second process makes a choice for disc 1 or disc 2 before it reads datum d. Both processes display the same strings of actions, $read(d)write_1(d)$ and $read(d)write_2(d)$, so they are trace equivalent. Still, there is a crucial distinction between the two processes, which becomes apparent if for instance disc 1 crashes. In this case the first process always saves datum d on disc 2, while the second process may get into a deadlock (i.e., may get stuck).

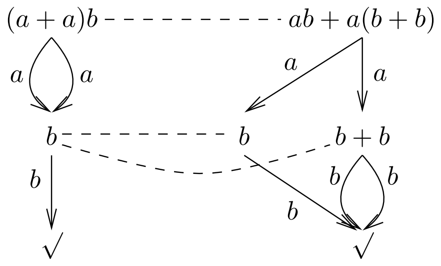
Bisimilarity, defined below, is more discriminative than trace equivalence. Namely, if two processes are bisimilar, then not only they can execute exactly the same strings of actions, but also they have the same branching structure. For example, the two processes in Example 3 are not bisimilar.

A bisimulation relation $\mathcal{B}$ is a binary relation on states in process graphs such that:

1. if $s\,\mathcal{B}\,t$ and $s \xrightarrow{a} s'$, then $t \xrightarrow{a} t'$ with $s'\,\mathcal{B}\,t'$;
2. if $s\,\mathcal{B}\,t$ and $t \xrightarrow{a} t'$, then $s \xrightarrow{a} s'$ with $s'\,\mathcal{B}\,t'$;
3. if $s\,\mathcal{B}\,t$ and $s \xrightarrow{a} \surd$, then $t \xrightarrow{a} \surd$;
4. if $s\,\mathcal{B}\,t$ and $t \xrightarrow{a} \surd$, then $s \xrightarrow{a} \surd$.

Two states s and t are bisimilar, denoted by $s \underline{\leftrightarrow} t$, if there is a bisimulation relation $\mathcal{B}$ such that $s\,\mathcal{B}\,t$.

Example 4. $(a+a)b \underline{\leftrightarrow} ab + a(b+b)$.
A bisimulation relation that relates these two basic process terms is defined by $(a+a)b\,\mathcal{B}\,ab + a(b+b)$, $b\,\mathcal{B}\,b$, and $b\,\mathcal{B}\,b+b$. This bisimulation relation can be depicted as follows:



Bisimilarity is a congruence with respect to BPA. That is, if $s \underline{\leftrightarrow} s'$ and $t \underline{\leftrightarrow} t'$, then $s+t \underline{\leftrightarrow} s'+t'$ and $s{\cdot}t \underline{\leftrightarrow} s'{\cdot}t'$. This follows from the fact that the transition rules in Table 1 are in the so-called path format [21].

3.4 Axioms for BPA

Checking whether the process graphs of two basic process terms are bisimilar requires hard labour. First these process graphs have to be computed, and next a bisimulation relation has to be established between their root states. This section introduces an axiomatisation for BPA, to equate bisimilar basic process terms. This avoids the computation of process graphs and bisimulation relations altogether. The axioms have the additional advantage that they can be used in automated reasoning, so that they facilitate a mechanised derivation that two basic process terms are bisimilar.

We are after an axiomatisation such that the induced equality relation $=$ on basic process terms characterises bisimilarity over BPA in the following sense:

1. the equality relation is sound, meaning that if $s = t$ holds for basic process terms s and t, then $s \underline{\leftrightarrow} t$;
2. the equality relation is complete, meaning that if $s \underline{\leftrightarrow} t$ holds for basic process terms s and t, then $s = t$.

Soundness ensures that if terms can be equated, then they are in the same bisimilarity class, while completeness ensures that bisimilar terms can always be equated.

Table 2. Axioms for BPA

A1	$x + y = y + x$
A2	$(x + y) + z = x + (y + z)$
A3	$x + x = x$
A4	$(x + y){\cdot}z = x{\cdot}z + y{\cdot}z$
A5	$(x{\cdot}y){\cdot}z = x{\cdot}(y{\cdot}z)$

Table 2 presents an axiomatisation for BPA modulo bisimilarity. The equality relation on basic process terms induced by this axiomatisation is obtained by taking the set of substitution instances of A1-5, and closing it under equivalence and contexts.

The axiomatisation A1-5 is sound for BPA modulo bisimilarity. Since bisimilarity is both an equivalence and a congruence for BPA, it suffices to check the soundness of the individual axioms.

Moreover, the axiomatisation is complete for BPA modulo bisimilarity, meaning that $s \underline{\leftrightarrow} t$ implies $s = t$. This can be proved by directing axioms A3-5 from left to right, so that we obtain a term rewriting system (see, e.g., [20]). One can show that bisimilar basic process terms reduce to the same normal form, modulo the axioms A1,2.

From now on, process terms are considered modulo associativity of the $+$, and we often write $t_1 + t_2 + t_3$ instead of $(t_1 + t_2) + t_3$ or $t_1 + (t_2 + t_3)$.

4 Algebra of Communicating Processes

Atomic actions and the operators alternative and sequential composition from the previous section provide relatively primitive tools to construct a process graph. In general, the size of a basic process term is comparable to the size of the related process graph. This section introduces operators to express parallelism and concurrency, which enable us to capture a large process graph by means of a comparatively small process term.

4.1 Parallelism and Communication

In practice, process behaviour is often composed of several processors that are executed in parallel, where these separate entities may influence each other's execution. One could say that the processors are the building blocks that make up the complete system, cemented together by mutual communication actions. In order to model such concurrent systems, we introduce the merge, which is a binary operator that executes the two process terms in its arguments in parallel. That is, $s \| t$ can choose to execute an initial transition of s (i.e., a transition $s \xrightarrow{a} s'$ or $s \xrightarrow{a} \surd$) or an initial transition of t. This is formalised by four transition rules for the merge:

$$\frac{x \xrightarrow{v} \surd}{x\|y \xrightarrow{v} y} \qquad \frac{x \xrightarrow{v} x'}{x\|y \xrightarrow{v} x'\|y} \qquad \frac{y \xrightarrow{v} \surd}{x\|y \xrightarrow{v} x} \qquad \frac{y \xrightarrow{v} y'}{x\|y \xrightarrow{v} x\|y'}$$

Moreover, $s\|t$ can choose to execute a communication between initial transitions of s and t. For this purpose we assume a communication function $\gamma : A \times A \to A$, which produces for each pair of atomic actions a and b their communication $\gamma(a, b)$. This communication function is required to be commutative and associative; that is, for $a, b, c \in A$,

$$\gamma(a,b) \equiv \gamma(b,a)$$
$$\gamma(\gamma(a,b),c) \equiv \gamma(a,\gamma(b,c)).$$

The next four transition rules for the merge express that $s\|t$ can choose to execute a communication of initial transitions of s and t:

$$\frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} \surd}{x\|y \xrightarrow{\gamma(v,w)} \surd} \qquad \frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} y'}{x\|y \xrightarrow{\gamma(v,w)} y'} \qquad \frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} \surd}{x\|y \xrightarrow{\gamma(v,w)} x'} \qquad \frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} y'}{x\|y \xrightarrow{\gamma(v,w)} x'\|y'}$$

Example 5. Let the communication of two atomic actions from $\{a, b, c\}$ always result in c. The process graph of the process term $(ab)\|(ba)$ is depicted below.

This example shows that the merge of two simple process terms produces a relatively large process graph. This partly explains the strength of a theory of communicating processes, as this theory makes it possible to draw conclusions about the full system by studying its separate concurrent components.

$$\begin{tikzcd}
 & & & & & & (ab)\|(ba) \arrow[dlll, "a"'] \arrow[d, "c"] \arrow[drrr, "b"] & & & & & & \\
 & & & b\|(ba) \arrow[dl, "b"'] \arrow[d, "c"] \arrow[dr, "b"] & & & b\|a \arrow[dl, "b"'] \arrow[d, "c"] \arrow[dr, "a"] & & & (ab)\|a \arrow[dl, "a"'] \arrow[d, "c"] \arrow[dr, "a"] & & & \\
 & & b\|a \arrow[dll, "b"'] \arrow[dl, "c"] \arrow[d, "a"] & a \arrow[d, "a"] & ba \arrow[d, "b"] & a \arrow[d, "a"] & \surd & b \arrow[d, "b"'] & ab \arrow[d, "a"'] & b \arrow[d, "b"'] & b\|a \arrow[d, "b"'] \arrow[dr, "c"] \arrow[drr, "a"] & & \\
a \arrow[d, "a"] & \surd & b \arrow[d, "b"] & \surd & a \arrow[d, "a"] & \surd & & \surd & b \arrow[d, "b"'] & \surd & a \arrow[d, "a"'] & \surd & b \arrow[d, "b"'] \\
\surd & & \surd & & \surd & & & & \surd & & \surd & & \surd
\end{tikzcd}$$

4.2 Left Merge and Communication Merge

Moller [16] proved that there does not exist a sound and complete finite axiomatisation for BPA extended with the merge, modulo bisimilarity. This problem is overcome by defining two extra operators, called left merge and communication merge, which both capture part of the behaviour of the merge.

The left merge $s \mathbin{⫗} t$ takes its initial transition from the process term s, and then behaves as the merge $\|$. This is expressed by two transition rules for the left merge, which correspond with the first two transition rules for the merge:

$$\frac{x \xrightarrow{v} \surd}{x \mathbin{⫗} y \xrightarrow{v} y} \qquad \frac{x \xrightarrow{v} x'}{x \mathbin{⫗} y \xrightarrow{v} x'\|y}$$

The communication merge $s|t$ executes as initial transition a communication between initial transitions of the process terms s and t, and then behaves as the merge $\|$. This is expressed by four transition rules for the communication merge, which correspond with the last four transition rules for the merge:

$$\frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} \surd}{x|y \xrightarrow{\gamma(v,w)} \surd} \qquad \frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} y'}{x|y \xrightarrow{\gamma(v,w)} y'} \qquad \frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} \surd}{x|y \xrightarrow{\gamma(v,w)} x'} \qquad \frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} y'}{x|y \xrightarrow{\gamma(v,w)} x'\|y'}$$

As binding convention we assume that $\|$, $⫗$, and $|$ bind stronger than $+$. For example, $a \mathbin{⫗} b + a\|c$ represents $(a \mathbin{⫗} b) + (a\|c)$. We refer to BPA extended with the three parallel operators $\|$, $⫗$, and $|$ as PAP (for process algebra with parallelism).

The left and communication merge together cover the behaviour of the merge, in the sense that $s\|t \;\underline{\leftrightarrow}\; (s \mathbin{⫗} t + t \mathbin{⫗} s) + s|t$. Namely, $s\|t$ can execute either an initial transition of s or t, which is covered by $s \mathbin{⫗} t$ or $t \mathbin{⫗} s$, respectively, or a communication of initial transitions of s and t, which is covered by $s|t$.

The transition rules of PAP constitute a conservative extension of the ones of BPA, meaning that they do not influence the process graphs of basic process terms. That is, an initial transition of a basic process term is derivable from the transition rules of PAP if and only if this transition can be derived from the transition rules of BPA. This follows from the fact that this extension adheres to the syntactic restrictions of the conservative extension format from [11].

Bisimilarity is a congruence with respect to PAP. Again this follows from the fact that the transition rules of PAP are in the path format.

4.3 Axioms for PAP

Table 3 presents the axioms for the three parallel operators modulo bisimilarity. We already noted that the merge can be split into the left merge and the communication merge; this is exploited in axiom M1. Axioms LM2-4 and CM5-10 enable us to eliminate occurrences of the left merge and the communication merge from process terms. The axioms for PAP are added to the ones for BPA.

Table 3. Axioms for merge, left merge, and communication merge

M1	$x \| y = (x \mathbin{\|\!\!\lfloor} y + y \mathbin{\|\!\!\lfloor} x) + x \mid y$
LM2	$v \mathbin{\|\!\!\lfloor} y = v \cdot y$
LM3	$(v \cdot x) \mathbin{\|\!\!\lfloor} y = v \cdot (x \| y)$
LM4	$(x + y) \mathbin{\|\!\!\lfloor} z = x \mathbin{\|\!\!\lfloor} z + y \mathbin{\|\!\!\lfloor} z$
CM5	$v \mid w = \gamma(v, w)$
CM6	$v \mid (w \cdot y) = \gamma(v, w) \cdot y$
CM7	$(v \cdot x) \mid w = \gamma(v, w) \cdot x$
CM8	$(v \cdot x) \mid (w \cdot y) = \gamma(v, w) \cdot (x \| y)$
CM9	$(x + y) \mid z = x \mid z + y \mid z$
CM10	$x \mid (y + z) = x \mid y + x \mid z$

It can be proved that the resulting axiomatisation is sound and complete for PAP modulo bisimilarity. Again, the completeness proof is based on a term rewriting analysis, in which the axioms are directed from left to right.

4.4 Deadlock and Encapsulation

If two atomic actions are able to communicate, then often we only want these actions to occur in communication with each other, and not on their own. For example, let the action $\mathit{send}(d)$ represent sending a datum d into one end of a channel, while $\mathit{read}(d)$ represents receiving this datum at the other end of the channel. Furthermore, let the communication of these two actions result in transferring the datum d through the channel by the action $\mathit{comm}(d)$. For the outside world, the actions $\mathit{send}(d)$ and $\mathit{read}(d)$ never appear on their own, but only in communication in the form $\mathit{comm}(d)$.

In order to enforce communication in such cases, we introduce a special constant δ called deadlock, which does not display any behaviour. The communication function γ

is extended by allowing that the communication of two atomic actions results in δ, i.e., $\gamma : A \times A \to A \cup \{\delta\}$. This extension of γ enables us to express that two actions a and b do not communicate, by defining $\gamma(a,b) \equiv \delta$. Furthermore, we introduce unary encapsulation operators ∂_H for sets H of atomic actions, which rename all actions in H into δ. PAP extended with deadlock and encapsulation operators is called the algebra of communicating processes (ACP).

Since the deadlock does not display any behaviour, there is no transition rule for this constant. Furthermore, since the communication of actions can result in δ, the last four transition rules for the merge and the four transition rules for the communication merge need to be supplied with the requirement $\gamma(v,w) \not\equiv \delta$. Finally, the behaviour of the encapsulation operators is captured by the following transition rules, which express that $\partial_H(t)$ can execute those transitions of t that have a label outside H:

$$\frac{x \xrightarrow{v} \surd}{\partial_H(x) \xrightarrow{v} \surd}\ v \notin H \qquad\qquad \frac{x \xrightarrow{v} x'}{\partial_H(x) \xrightarrow{v} \partial_H(x')}\ v \notin H$$

We give an example of the use of encapsulation operators.

Example 6. Suppose a datum 0 or 1 is sent into a channel, which is expressed by the process term $\mathit{send}(0) + \mathit{send}(1)$. Let this datum be received at the other side of the channel, which is expressed by the process term $\mathit{read}(0) + \mathit{read}(1)$. The communication of $\mathit{send}(d)$ and $\mathit{read}(d)$ results in $\mathit{comm}(d)$ for $d \in \{0,1\}$, while all other communications between actions result in δ. The behaviour of the channel is described by the process term

$$\partial_{\{\mathit{send}(0),\,\mathit{send}(1),\,\mathit{read}(0),\,\mathit{read}(1)\}}((\mathit{send}(0) + \mathit{send}(1)) \| (\mathit{read}(0) + \mathit{read}(1)))$$

The encapsulation operator enforces that the action $\mathit{send}(d)$ can only occur in communication with the action $\mathit{read}(d)$, for $d \in \{0,1\}$.

Beware not to confuse a transition of the form $t \xrightarrow{a} \delta$ with a transition of the form $t \xrightarrow{a} \surd$; intuitively, the first transition expresses that t gets stuck after the execution of a, while the second transition expresses that t terminates successfully after the execution of a. A process term t is said to contain a deadlock if there are transitions $t \xrightarrow{a_1} t_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} t_n$ such that the process term t_n does not have any initial transitions (i.e., $t_n \underline{\leftrightarrow} \delta$). In general it is undesirable that a process contains a deadlock, because it represents that the process gets stuck without producing any output. Experience learns that non-trivial specifications of system behaviour often contain a deadlock. For example, the third sliding window protocol in [19] contained a deadlock; see [14, *Stelling 7*]. It can, however, be very difficult to detect such a deadlock, even if one has a good insight into such a protocol. Automated tools have been developed to help with the detection of deadlocks in a process algebraic framework.

ACP is a conservative extension of PAP, meaning that the transition rules for the encapsulation operators do not influence the process graphs belonging to process terms in PAP. Again this follows from the fact that this extension adheres to the syntactic restrictions of the conservative extension format. Moreover, bisimilarity is a congruence with respect to ACP, because the transition rules of ACP are in the path format.

Table 4. Axioms for deadlock and encapsulation

A6		$x + \delta = x$
A7		$\delta \cdot x = \delta$
D1	$v \notin H$	$\partial_H(v) = v$
D2	$v \in H$	$\partial_H(v) = \delta$
D3		$\partial_H(\delta) = \delta$
D4		$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$
D5		$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$
LM11		$\delta \mathbin{\lfloor\!\lfloor} x = \delta$
CM12		$\delta \mid x = \delta$
CM13		$x \mid \delta = \delta$

Table 4 presents axioms A6,7 for the deadlock, axioms D1-5 for encapsulation, and axioms LM11 and CM12,13 to deal with the interplay of the deadlock with left and communication merge.

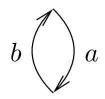
It can be proved that the resulting axiomatisation is sound and complete for ACP modulo bisimilarity. Again, the completeness proof is based on a term rewriting analysis, in which the axioms are directed from left to right.

5 Recursion

Up to now we have focused on finite processes. However, systems can often exhibit infinite traces. In this section it is shown how such infinite behaviour can be specified using recursive equations.

5.1 Guarded Recursive Specifications

Consider the process that alternately executes actions a and b until infinity, with the root node presented at the top:



Since ACP can only specify finite behaviour, there does not exist a process term in ACP with this (or a bisimilar) process graph. The process above can be captured by means of two recursive equations:

$$\begin{aligned} X &= aY \\ Y &= bX. \end{aligned}$$

Here, X and Y are recursion variables, which intuitively represent the two states of the process in which it is going to execute a or b, respectively.

In general, a recursive specification consists of a finite set of recursive equations

$$\begin{aligned} X_1 &= t_1(X_1,\ldots,X_n) \\ &\vdots \\ X_n &= t_n(X_1,\ldots,X_n) \end{aligned}$$

where the left-hand sides X_i are recursion variables, and the $t_i(X_1,\ldots,X_n)$ at the right-hand sides are process terms in ACP with possible occurrences of the recursion variables $X_1,\ldots,X_n$.

Process terms $s_1,\ldots,s_n$ are said to be a solution for a recursive specification $\{X_i = t_i(X_1,\ldots,X_n) \mid i \in \{1,\ldots,n\}\}$ (with respect to bisimilarity) if $s_i \underline{\leftrightarrow} t_i(s_1,\ldots,s_n)$ for all $i \in \{1,\ldots,n\}$.

A recursive specification should represent a unique process graph, so we want its solution to be unique, modulo bisimilarity. That is, if $s_1,\ldots,s_n$ and $s'_1,\ldots,s'_n$ are two solutions for the same recursive specification, then $s_i \underline{\leftrightarrow} s'_i$ for $i \in \{1,\ldots,n\}$. However, there exist recursive specifications that allow more than one solution modulo bisimilarity. We give some examples.

Example 7. Let $a \in A$.

1. All process terms are a solution for the recursive specification $\{X{=}X\}$.
2. All process terms s that can execute an initial transition $s \xrightarrow{a} \surd$ are a solution for the recursive specification $\{X{=}a{+}X\}$.
3. All process terms that cannot terminate successfully are a solution for the recursive specification $\{X{=}Xa\}$.

The following example features recursive specifications that do have a unique solution modulo bisimilarity.

Example 8. Let $a, b \in A$.

1. The only solution for $\{X{=}aY, Y{=}bX\}$, modulo bisimilarity, is $X \underline{\leftrightarrow} abab\cdots$ and $Y \underline{\leftrightarrow} baba\cdots$.
2. The only solution for $\{X{=}Y, Y{=}aX\}$, modulo bisimilarity, is $X \underline{\leftrightarrow} aaa\cdots$ and $Y \underline{\leftrightarrow} aaa\cdots$.
3. The only solution for $\{X{=}(a{+}b) \mathbin{⫰\!\!\_} X\}$, modulo bisimilarity, is $X \underline{\leftrightarrow} (a{+}b)(a{+}b)(a+b)\cdots$.

A recursive specification allows a unique solution modulo bisimilarity if and only if it is guarded. A recursive specification

$$\begin{aligned} X_1 &= t_1(X_1,\ldots,X_n) \\ &\vdots \\ X_n &= t_n(X_1,\ldots,X_n) \end{aligned}$$

is guarded if the right-hand sides of its recursive equations can be adapted to the form

$$a_1{\cdot}s_1(X_1,\ldots,X_n) + \cdots + a_k{\cdot}s_k(X_1,\ldots,X_n) + b_1 + \cdots + b_\ell$$

with $a_1, \ldots, a_k, b_1, \ldots, b_\ell \in A$, by applications of the axioms of ACP and replacing recursion variables by the right-hand sides of their recursive equations. The process term above is allowed to have zero summands (i.e., k and ℓ can both be zero), in which case it represents the deadlock δ.

The recursive specifications in Example 7 are all unguarded; that is, their right-hand sides cannot be brought into the desired form presented above. The recursive specifications in Example 8 are all guarded.

5.2 Transition Rules for Guarded Recursion

If E is a guarded recursive specification, and X a recursion variable in E, then intuitively $\langle X|E\rangle$ denotes the process that has to be substituted for X in the solution for E. For instance, if E is $\{X{=}aY, Y{=}bX\}$, then $\langle X|E\rangle$ represents the process $abab\cdots$, while $\langle Y|E\rangle$ represents the process $baba\cdots$; see the first recursive specification in Example 8. We extend ACP with the constants $\langle X|E\rangle$, for guarded recursive specifications E and recursion variables X in E.

Assume that the guarded recursive specification E is of the form

$$\begin{aligned} X_1 &= t_1(X_1, \ldots, X_n) \\ &\vdots \\ X_n &= t_n(X_1, \ldots, X_n). \end{aligned}$$

Guarded recursion is captured by two transition rules which express that the behaviour of the solutions $\langle X_i|E\rangle$ for the recursion variables X_i in E, for each $i \in \{1, \ldots, n\}$, is exactly the behaviour of its right-hand side $t_i(X_1, \ldots, X_n)$:

$$\frac{t_i(\langle X_1|E\rangle, \ldots, \langle X_n|E\rangle) \xrightarrow{v} \surd}{\langle X_i|E\rangle \xrightarrow{v} \surd} \qquad \frac{t_i(\langle X_1|E\rangle, \ldots, \langle X_n|E\rangle) \xrightarrow{v} y}{\langle X_i|E\rangle \xrightarrow{v} y}$$

Example 9. Let E denote $\{X{=}aY, Y{=}bX\}$. The process graph of $\langle X|E\rangle$ is

$$\begin{tikzcd} \langle X|E\rangle \arrow[d, bend left, "a"] \\ \langle Y|E\rangle \arrow[u, bend left, "b"] \end{tikzcd}$$

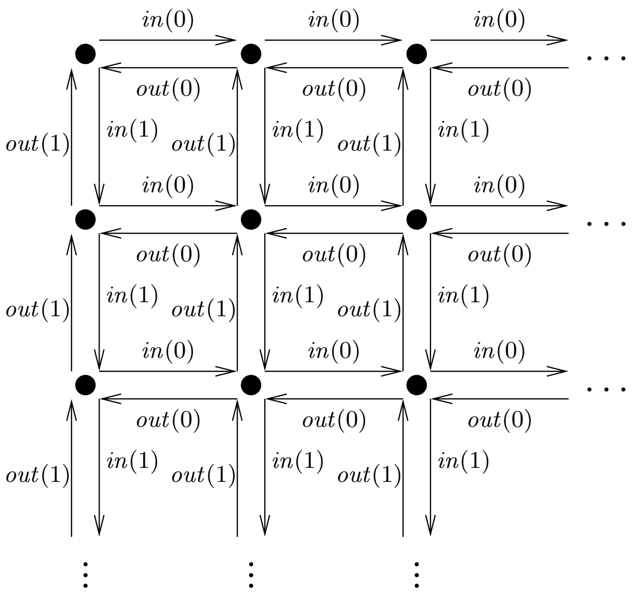
The transition $\langle X|E\rangle \xrightarrow{a} \langle Y|E\rangle$ can be derived from the transition rules as follows:

$$\begin{array}{ll} a \xrightarrow{a} \surd & (\dfrac{}{v \xrightarrow{v} \surd}, \quad v := a) \\ \hline a\langle Y|E\rangle \xrightarrow{a} \langle Y|E\rangle & (\dfrac{x \xrightarrow{v} \surd}{xy \xrightarrow{v} y}, \quad v := a,\ x := a,\ y := \langle Y|E\rangle) \\ \hline \langle X|E\rangle \xrightarrow{a} \langle Y|E\rangle & (\dfrac{a\langle Y|E\rangle \xrightarrow{v} y}{\langle X|E\rangle \xrightarrow{v} y}, \ v := a,\ y := \langle Y|E\rangle) \end{array}$$

ACP with guarded recursion is a conservative extension of ACP, because this extension adheres to the syntactic restrictions of the conservative extension format. Moreover, bisimilarity is a congruence with respect to ACP with guarded recursion, because the transition rules of guarded recursion are in the path format.

As an example of the use of guarded recursion we consider the bag process over the set $\{0,1\}$.

Example 10. We specify a process that can put elements 0 and 1 into a bag, and subsequently collect these elements from the bag in arbitrary order. The actions $in(0)$ and $in(1)$ represent putting a 0 or 1 into the bag, respectively. Similarly, the actions $out(0)$ and $out(1)$ represent collecting a 0 or 1 from the bag, respectively. All communications between actions result in δ. Initially the bag is empty, so that one can only put an element into the bag. The process graph below depicts the behaviour of the bag over $\{0,1\}$, with the root state placed in the leftmost uppermost corner. Note that this bag process consists of infinitely many non-bisimilar states.



The bag over $\{0,1\}$ can be specified by a single recursive equation, using the merge $\|$. Let E denote the guarded recursive specification

$$X \;=\; in(0)\cdot(X\|out(0)) + in(1)\cdot(X\|out(1)).$$

The process graph of $\langle X|E\rangle$ is bisimilar with the behaviour of the bag over $\{0,1\}$ as depicted above. Namely, initially $\langle X|E\rangle$ can only execute an action $in(d)$ for $d \in \{0,1\}$. The subsequent process term $\langle X|E\rangle\|out(d)$ can put elements 0 and 1 in the bag and take them out again (by means of the parallel component $\langle X|E\rangle$), or it can at any time take the initial element d out of the bag (by means of the parallel component $out(d)$).

5.3 Recursive Definition and Specification Principles

As before, we want to fit guarded recursion into an axiomatic framework. Table 5 contains two axioms for guarded recursion, the recursive definition principle (RDP) and the recursive specification principle (RSP). The guarded recursive specification E in the axioms is assumed to be of the form

$$\begin{aligned} X_1 &= t_1(X_1, \ldots, X_n) \\ &\vdots \\ X_n &= t_n(X_1, \ldots, X_n). \end{aligned}$$

Intuitively, RDP expresses that $\langle X_1|E\rangle, \ldots, \langle X_n|E\rangle$ is a solution for E, while RSP expresses that this is the only solution for E modulo bisimilarity.

Table 5. Recursive definition and specification principles

RDP	$\langle X_i|E\rangle \ = \ t_i(\langle X_1|E\rangle, \ldots, \langle X_n|E\rangle)$	$(i \in \{1, \ldots, n\})$
RSP	If $y_i = t_i(y_1, \ldots, y_n)$ for all $i \in \{1, \ldots, n\}$, then	
	$y_i \ = \ \langle X_i|E\rangle \qquad (i \in \{1, \ldots, n\})$	

The resulting axomatisation is sound for ACP with guarded recursion modulo bisimilarity. However, it is not complete. For instance, the following two symmetric guarded recursive specifications of the bag over $\{0, 1\}$ (see Example 10) are bisimilar, but cannot be proved equal by means of the axioms:

$$\begin{aligned} X &= \mathit{in}(0)\cdot(X \| \mathit{out}(0)) + \mathit{in}(1)\cdot(X \| \mathit{out}(1)) \\ Y &= \mathit{in}(0)\cdot(\mathit{out}(0) \| Y) + \mathit{in}(1)\cdot(\mathit{out}(1) \| Y). \end{aligned}$$

(In this particular case, this could be remedied by adding a commutativity axiom for the merge.)

One can prove that the axiomatisation is complete for the subclass of linear recursive specifications. A recursive specification is linear if its recursive equations are of the form

$$X \ = \ a_1X_1 + \cdots + a_kX_k + b_1 + \cdots + b_\ell$$

with $a_1, \ldots, a_k, b_1, \ldots, b_\ell \in A$. (The empty sum represents δ.) Note that a linear recursive specification is by default guarded.

A regular process, which by definition consists of finitely many states and transitions, can always be described by a linear recursive specification. Namely, each state s in the regular process can be represented by a recursion variable X_s. If state s can evolve into state s' by the execution of an action a, then this is expressed by a summand $aX_{s'}$ at the right-hand side of the recursive equation for X_s. Moreover, if state s can terminate successfully by the execution of an action a, then this is expressed by a summand a at the right-hand side of the recursive equation for X_s. The result is a linear recursive specification E, and $\langle X_s|E\rangle \underline{\leftrightarrow}\ s$ for all states s in the regular process. Vice versa, a linear recursive specification always gives rise to a regular process.

6 Abstraction

If a customer asks a programmer to implement a product, ideally this customer is able to provide the external behaviour of the desired program. That is, he or she is able to tell what should be the output of the program for each possible input. The programmer then comes up with an implementation. The question is, does this implementation really display the desired external behaviour? To answer this question, we need to abstract away from the internal computation steps of the program.

6.1 Rooted Branching Bisimulation

In order to abstract away from internal actions, we introduce a special constant τ, called the silent step. Intuitively, a τ-transition represents a sequence of internal actions that can be eliminated from a process graph. As any atomic action, the constant τ can execute itself, after which it terminates successfully. This is expressed by the transition rule

$$\frac{}{\tau \xrightarrow{\tau} \surd}$$

From now on, v and w in the transition rules and the axioms of ACP with guarded recursion range over $A \cup \{\tau\}$. (So the transition rule for atomic actions in Table 1 yields the transition rule for the silent step τ presented above.) The domain of the communication function γ is extended with the silent step, $\gamma : A \cup \{\tau\} \times A \cup \{\tau\} \rightarrow A \cup \{\delta\}$, by defining that each communication involving τ results in δ.

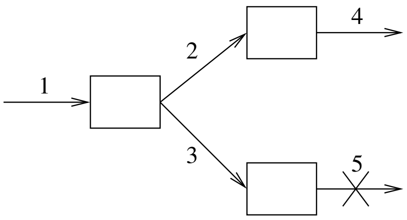
In the presence of the silent step τ, bisimilarity is no longer a satisfactory process equivalence. Namely, if process terms s and t are equivalent, and s can execute an action τ, then it need not be the case that t can simulate this τ-transition of s by the execution of an action τ. The intuition for the silent step, that it represents an internal computation in which we are not really interested, asks for a new process equivalence. The question that we must pose ourselves is:

which τ-transitions are truly silent?

The obvious answer to this question, "all τ-transitions are truly silent", turns out to be incorrect. Namely, this answer would produce an equivalence relation that does not preserve deadlock behaviour.

As an example of an action τ that is not truly silent, consider the process terms $a + \tau\delta$ and a. If the τ in the first term were truly silent, then these two terms would be equivalent. However, the process graph of the first term contains a deadlock, $a+\tau\delta \xrightarrow{\tau} \delta$, while the process graph of the second term does not. Hence, the τ in the first term is not truly silent. In order to describe this case more vividly, we give an example.

Example 11. Consider a protocol that first receives a datum d via channel 1, and then communicates this datum via channel 2 or via channel 3. If the datum is communicated through channel 2, then it is sent into channel 4. If the datum is communicated through channel 3, then it gets stuck, as the subsequent channel 5 is broken. So the system gets into a deadlock if the datum d is transferred via channel 3. This deadlock should not disappear if we abstract away from the internal communication actions via channels 2 and 3, because this would cover up an important problem of the protocol.



The system, which is depicted above, is described by the process term

$$\partial_{\{s_5(d)\}}(r_1(d)\cdot(c_2(d)\cdot s_4(d) + c_3(d)\cdot s_5(d))) \overset{\text{D1,2,4,5}}{=} r_1(d)\cdot(c_2(d)\cdot s_4(d) + c_3(d)\cdot\delta)$$

where $s_i(d)$, $r_i(d)$, and $c_i(d)$ represent a send, read, and communication action of the datum d via channel i, respectively. Abstracting away from the internal actions $c_2(d)$ and $c_3(d)$ in this process term yields $r_1(d)\cdot(\tau\cdot s_4(d)+\tau\cdot\delta)$. The second τ in this process term cannot be deleted, because then the process would no longer be able to get into a deadlock. Hence, this τ is not truly silent.

As a further example of a τ-transition that is not truly silent, consider the process terms $a + \tau b$ and $a + b$. We argued previously that the process terms $\partial_{\{b\}}(a + \tau b) = a + \tau\delta$ and $\partial_{\{b\}}(a+b) = a$ are not equivalent, because the first term contains a deadlock while the second term does not. Hence, $a + \tau b$ and $a + b$ cannot be equivalent, for else the envisioned equivalence relation would not be a congruence.

Problems with deadlock preservation and congruence can be avoided by taking a more restrictive view on abstracting away from silent steps. A correct answer to the question

which τ-transitions are truly silent?

turns out to be

those τ-transitions that do not lose possible behaviours!

For example, the process terms $a + \tau(a + b)$ and $a + b$ are equivalent, because the τ in the first process term is truly silent: after execution of this τ it is still possible to execute a. In general, process terms $s + \tau(s + t)$ and $s + t$ are equivalent for all process terms s and t. By contrast, in a process term such as $a + \tau b$ the τ is not truly silent, since execution of this τ means losing the option to execute a.

The intuition above is formalised in the notion of branching bisimilarity. Let the process terms s and t be branching bisimilar. If $s \xrightarrow{\tau} s'$, then t does not have to simulate this τ-transition if it is truly silent, meaning that s' and t are branching bisimilar. Moreover, a non-silent transition $s \xrightarrow{a} s'$ need not be simulated by t immediately, but only after a number of truly silent τ-transitions: $t \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_0 \xrightarrow{a} t'$, where s and t_0 are branching bisimilar (to ensure that the τ-transitions are truly silent) and s' and t' are branching bisimilar (so that $s \xrightarrow{a} s'$ is simulated by $t_0 \xrightarrow{a} t'$). A special termination predicate $\downarrow$ is needed in order to relate branching bisimilar process terms such as $a\tau$ and a.

Assume a special termination predicate $\downarrow$, and let $\surd$ represent a state with $\surd\downarrow$. A branching bisimulation relation $\mathcal{B}$ is a binary relation on states in process graphs such that:

1. if $s\,\mathcal{B}\,t$ and $s \xrightarrow{a} s'$, then
   - either $a \equiv \tau$ and $s'\,\mathcal{B}\,t$;
   - or there is a sequence of (zero or more) τ-transitions $t \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_0$ such that $s\,\mathcal{B}\,t_0$ and $t_0 \xrightarrow{a} t'$ with $s'\,\mathcal{B}\,t'$;
2. if $s\,\mathcal{B}\,t$ and $t \xrightarrow{a} t'$, then
   - either $a \equiv \tau$ and $s\,\mathcal{B}\,t'$;
   - or there is a sequence of (zero or more) τ-transitions $s \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_0$ such that $s_0\,\mathcal{B}\,t$ and $s_0 \xrightarrow{a} s'$ with $s'\,\mathcal{B}\,t'$.
3. if $s\,\mathcal{B}\,t$ and $s\downarrow$, then there is a sequence of (zero or more) τ-transitions $t \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_0$ such that $s\,\mathcal{B}\,t_0$ and $t_0\downarrow$;
4. if $s\,\mathcal{B}\,t$ and $t\downarrow$, then there is a sequence of (zero or more) τ-transitions $s \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_0$ such that $s_0\,\mathcal{B}\,t$ and $s_0\downarrow$.

Two states s and t are branching bisimilar, denoted by $s \mathrel{\underline{\leftrightarrow}_b} t$, if there is a branching bisimulation relation $\mathcal{B}$ such that $s\,\mathcal{B}\,t$.

Example 12. $a + \tau(a+b) \mathrel{\underline{\leftrightarrow}_b} \tau(a+b) + b$.
A branching bisimulation relation that relates these two process terms is defined by $a+\tau(a+b)\,\mathcal{B}\,\tau(a+b)+b$, $a+b\,\mathcal{B}\,\tau(a+b)+b$, $a+\tau(a+b)\,\mathcal{B}\,a+b$, $a+b\,\mathcal{B}\,a+b$, and $\surd\,\mathcal{B}\,\surd$. This relation can be depicted as follows:

$$\begin{tikzcd}
& a+\tau(a+b) \arrow[dl, "a"'] \arrow[dr, "\tau"] \arrow[rrr, dash, dashed] \arrow[drr, dash, dashed] & & & \tau(a+b)+b \arrow[dl, "\tau"'] \arrow[dr, "b"] \arrow[dll, dash, dashed] & \\
\surd \arrow[rrrrr, dash, dashed, bend right] & & a+b \arrow[ll, shift right, "a"'] \arrow[ll, shift left, "b"] \arrow[r, dash, dashed] & a+b \arrow[rr, shift left, "a"] \arrow[rr, shift right, "b"'] & & \surd
\end{tikzcd}$$

It is left to the reader to verify that this relation satisfies the requirements of a branching bisimulation relation.

Branching bisimilarity satisfies a notion of *fairness*. That is, if an exit from a τ-loop exists, then no infinite execution sequence will remain in this τ-loop forever. The intuition is that there is zero chance that no exit from the τ-loop will ever be chosen. For example, it is not hard to see that $\langle X \mid X = \tau X + a\rangle$ and a are branching bisimilar.

Branching bisimilarity preserves a large class of interesting properties (including deadlock behaviour) [7]. See [13] for an exposition on why branching bisimilarity constitutes a sensible equivalence relation to abstract away from internal computations.

Branching bisimilarity is an equivalence relation; see [4]. However, it is still not a congruence with respect to BPA. For example, b and τb are branching bisimilar, but we already argued that $a + b$ and $a + \tau b$ are not branching bisimilar. This problem can be overcome by adding a rootedness condition: initial τ-transitions are never truly silent. In other words, two states are considered equivalent if they can simulate each other's initial transitions, such that the resulting states are branching bisimilar. This leads to the notion of rooted branching bisimilarity.

A rooted branching bisimulation relation $\mathcal{B}$ is a binary relation on states in process graphs such that:

1. if $s\,\mathcal{B}\,t$ and $s \xrightarrow{a} s'$, then $t \xrightarrow{a} t'$ with $s' \underline{\leftrightarrow}_b t'$;
2. if $s\,\mathcal{B}\,t$ and $t \xrightarrow{a} t'$, then $s \xrightarrow{a} s'$ with $s' \underline{\leftrightarrow}_b t'$;
3. if $s\,\mathcal{B}\,t$ and $s\downarrow$, then $t\downarrow$;
4. if $s\,\mathcal{B}\,t$ and $t\downarrow$, then $s\downarrow$.

Two states s and t are rooted branching bisimilar, denoted by $s \underline{\leftrightarrow}_{rb} t$, if there is a rooted branching bisimulation relation $\mathcal{B}$ such that $s\,\mathcal{B}\,t$.

Since branching bisimilarity is an equivalence relation, it is not hard to see that rooted branching bisimilarity is also an equivalence relation. Branching bisimilarity includes rooted branching bisimilarity, which in turn includes bisimilarity:

$$\underline{\leftrightarrow} \subset \underline{\leftrightarrow}_{rb} \subset \underline{\leftrightarrow}_b \ .$$

In the absence of τ (for example, in ACP), bisimilarity and branching bisimilarity induce exactly the same equivalence classes. In other words, two process terms in ACP are bisimilar if and only if they are branching bisimilar.

6.2 Guarded Linear Recursion Revisited

Assume a recursive specification E that consists of linear recursive equations $X_i = t_i(X_1, \ldots, X_n)$ for $i \in \{1, \ldots, n\}$. Since from now on we consider process terms in the setting of rooted branching bisimilarity, process terms $s_1, \ldots, s_n$ are said to be a solution for E (with respect to rooted branching bisimilarity) if $s_i \underline{\leftrightarrow}_{rb} t_i(s_1, \ldots, s_n)$ for $i \in \{1, \ldots, n\}$.

In the setting with the silent step, the notion of guardedness, which aims to classify those recursive specifications that have a unique solution modulo the process equivalence under consideration, needs to be adapted. For example, all process terms τs are solutions for the recursive specification $X = \tau X$, because $\tau s \underline{\leftrightarrow}_{rb} \tau\tau s$ holds for all process terms s. Hence, we consider such a recursive specification to be unguarded. The notion of guardedness is extended to linear recursive specifications that involve silent steps by requiring the absence of τ-loops.

A recursive specification is linear if its recursive equations are of the form

$$X \ = \ a_1X_1 + \cdots + a_kX_k + b_1 + \cdots + b_\ell$$

with $a_1, \ldots, a_k, b_1, \ldots, b_\ell \in A \cup \{\tau\}$. A linear recursive specification E is guarded if there does not exist an infinite sequence of τ-transitions $\langle X|E\rangle \xrightarrow{\tau} \langle X'|E\rangle \xrightarrow{\tau}$

$\langle X''|E\rangle \xrightarrow{\tau} \cdots$. The guarded linear recursive specifications are exactly the linear recursive specifications that have a unique solution, modulo rooted branching bisimilarity.

ACP with silent step guarded linear recursion constitutes a conservative extension of ACP with linear recursion, because this extension adheres to the syntactic restrictions of the conservative extension format. Moreover, rooted branching bisimilarity is a congruence with respect to ACP silent step and guarded linear recursion. This follows from the fact that the transition rules are in the RBB cool format from [9].

Table 6 presents the axioms B1,2 for the silent step, modulo rooted branching bisimilarity.

Table 6. Axioms for the silent step

B1	$v{\cdot}\tau = v$
B2	$v{\cdot}(\tau{\cdot}(x+y)+x) = v{\cdot}(x+y)$

The resulting axiomatisation is sound for ACP with silent step and guarded linear recursion, modulo rooted branching bisimilarity. Moreover, it can be shown that the axiomatisation is complete, see [12].

6.3 Abstraction Operators

We introduce unary abstraction operators τ_I, for subsets I of A, which rename all atomic actions in I into τ. The abstraction operators enable us to abstract away from the internal computation steps of an implementation. The behaviour of the abstraction operators is captured by the following transition rules, which express that in $\tau_I(t)$ all labels of transitions of t that are in I are renamed into τ:

$$\frac{x \xrightarrow{v} \surd}{\tau_I(x) \xrightarrow{v} \surd}\ v \notin I \qquad \frac{x \xrightarrow{v} x'}{\tau_I(x) \xrightarrow{v} \tau_I(x')}\ v \notin I$$

$$\frac{x \xrightarrow{v} \surd}{\tau_I(x) \xrightarrow{\tau} \surd}\ v \in I \qquad \frac{x \xrightarrow{v} x'}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')}\ v \in I$$

ACP extended with silent step and abstraction operators is denoted by ACP$_\tau$.

ACP$_\tau$ once again constitutes a conservative extension of ACP, because this extension adheres to the syntactic restrictions of the conservative extension format. Moreover, rooted branching bisimilarity is a congruence with respect to ACP$_\tau$ with guarded linear recursion, because the transition rules are in the RBB cool format.

Table 7 presents axioms for the abstraction operators, modulo rooted branching bisimilarity.

The resulting axiomatisation is sound for ACP$_\tau$ with guarded linear recursion modulo rooted branching bisimilarity. However, to obtain a complete axiomatisation, we need one more proof principle.

Table 7. Axioms for abstraction operators

TI1	$v \notin I$	$\tau_I(v) = v$
TI2	$v \in I$	$\tau_I(v) = \tau$
TI3		$\tau_I(\delta) = \delta$
TI4		$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$
TI5		$\tau_I(x{\cdot}y) = \tau_I(x){\cdot}\tau_I(y)$

6.4 Cluster Fair Abstraction Rule

Although τ-loops are prohibited in guarded linear recursive specifications, they can be constructed using an abstraction operator. For example, $\tau_{\{a\}}(\langle X \mid X{=}aX\rangle)$ can only execute τ's until infinity. This observation motivates the following distinction between specifiable and constructible regular processes:

– specifiable regular processes are the process graphs belonging to process terms in ACP with silent step and guarded linear recursion;
– constructible regular processes are the process graphs belonging to process terms in ACP$_\tau$ with guarded linear recursion.

$\tau\tau\tau\cdots$ is the simplest example of a regular process that is constructible, being the process graph of $\tau_{\{a\}}(\langle X \mid X{=}aX\rangle)$, but not specifiable. In general, a constructible regular process is specifiable if and only if it is free of τ-loops. One extra axiom is needed to equate process terms of which the regular process graphs are constructible but not specifiable. For example,

$$\tau_{\{a\}}(\langle X \mid X{=}aX\rangle) \underline{\leftrightarrow}_{rb}\ \tau_{\{a,b\}}(\langle Y \mid Y{=}aZ, Z{=}bY\rangle)$$

because both process terms execute τ's until infinity. However, these process terms cannot be equated by means of the axioms, due to the guardedness restriction on RSP, which is essential for the soundness of this axiom. In order to get rid of τ-loops, we introduce the notion of fair abstraction. For example, let E denote the following guarded linear recursive specification:

$$\begin{aligned} X_1 &= aX_2 + s_1 \\ &\ \vdots \\ X_{n-1} &= aX_n + s_{n-1} \\ X_n &= aX_1 + s_n \end{aligned}$$

for some $a \in A$. The process term $\tau_{\{a\}}(\langle X_1|E\rangle)$ executes τ-transitions that are the result of abstracting away from the occurrences of a in front of the recursion variables X_i, until it exits this τ-loop by executing one of the process terms $\tau_{\{a\}}(s_i)$ for $i \in \{1,\dots,n\}$. Note that the transitions in the τ-loop are all truly silent, because they do not lose possible behaviours; after the execution of such a τ, it is still possible to execute any of the process terms $\tau_{\{a\}}(s_i)$ for $i \in \{1,\dots,n\}$. Fair abstraction says that $\tau_{\{a\}}(\langle X_1|E\rangle)$ does not stay in the τ-loop forever, so that at some time it will start executing a $\tau_{\{a\}}(s_i)$. Hence,

$$\tau_{\{a\}}(\langle X_1|E\rangle) \leftrightarroweq_{rb} \tau_{\{a\}}(s_1 + \tau(s_1 + \cdots + s_n)).$$

Namely, initially $\tau_{\{a\}}(\langle X_1|E\rangle)$ can execute either $\tau_{\{a\}}(s_1)$ or τ. In the latter case, this initial (so non-silent) τ-transition is followed by the execution of a series of truly silent τ's in the τ-loop, until one of the process terms $\tau_{\{a\}}(s_i)$ for $i \in \{1, \ldots, n\}$ is executed.

We now present an axiom to eliminate a cluster of τ-transitions, so that only the exits of such a cluster remain. First, a precise definition is needed of a cluster and its exits.

Let E be a guarded linear recursive specification, and $I \subseteq A$. Two recursion variables X and Y in E are in the same cluster for I if and only if there exist sequences of transitions $\langle X|E\rangle \xrightarrow{b_1} \cdots \xrightarrow{b_m} \langle Y|E\rangle$ and $\langle Y|E\rangle \xrightarrow{c_1} \cdots \xrightarrow{c_n} \langle X|E\rangle$ with $b_1, \ldots, b_m, c_1, \ldots, c_n \in I \cup \{\tau\}$.

a or aX is an exit for the cluster C if and only if:

1. a or aX is a summand at the right-hand side of the recursive equation for a recursion variable in C; and
2. in the case of aX, either $a \notin I \cup \{\tau\}$ or $X \notin C$.

Table 8 presents an axiom called cluster fair abstraction rule (CFAR) for guarded linear recursive specifications. CFAR allows us to abstract away from a cluster of actions that are renamed into τ, after which only the exits of this cluster remain. In Table 8, E is a guarded linear recursive specification. Owing to the presence of the initial action τ at the left- and right-hand side of CFAR, the initial τ-transitions of $\tau_I(\langle X|E\rangle)$ can be truly silent. If the set of exits is empty, then the empty sum at the right-hand side of CFAR represents δ.

Table 8. Cluster fair abstraction rule

CFAR	If in E, X is in a cluster for I with exits $\{v_1Y_1, \ldots, v_mY_m, w_1, \ldots, w_n\}$, then $\tau{\cdot}\tau_I(\langle X|E\rangle) = \tau{\cdot}\tau_I(v_1\langle Y_1|E\rangle + \cdots + v_m\langle Y_m|E\rangle + w_1 + \cdots + w_n)$

The resulting axiomatisation (the axioms for ACP$_\tau$ together with RDP, RSP and CFAR) is sound and complete for ACP$_\tau$ with guarded linear recursion modulo rooted branching bisimilarity, see [8].

7 Alternating Bit Protocol

So far we have presented a standard framework ACP$_\tau$ with guarded linear recursion for the specification and manipulation of concurrent processes. Summarising, it consists of basic operators (A, $+$, $\cdot$) to define finite processes, communication operators ($\|$, $\lfloor\!\lfloor$, $|$)

to express parallelism, deadlock and encapsulation (δ, ∂_H) to force atomic actions into communication, silent step and abstraction (τ, τ_I) to make internal computations invisible, and guarded linear recursion ($\langle X|E\rangle$) to capture regular processes. These constructs form a solid basis for the analysis of a wide range of systems.

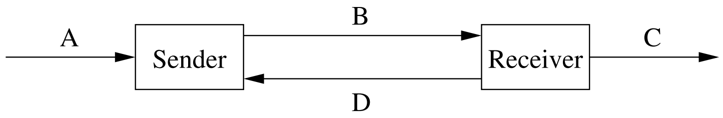
In particular, the framework is suitable for the specification and verification of network protocols. For such a verification, the desired external behaviour of the protocol is represented in the form of a process term that is in general built from the basic operators of BPA together with linear recursion. Moreover, the implementation of the protocol is represented in the form of a process term that involves the basic operators, the three parallel operators, and linear recursion. Next, the internal send and read actions of the implementation are forced into communication using an encapsulation operator, and the internal communication actions are made invisible using an abstraction operator, so that only the input/output relation of the implementation remains. If the two process terms can be equated by the axioms, then this proves that the process graphs belonging to the desired external behaviour and to the input/output relation of the implementation are rooted branching bisimilar.

An alternative to an equational correctness proof is to verify that the states in the process graph above satisfy desirable properties, expressed in some temporal logic (see, e.g., [18]). Such automated techniques to analyse process graphs are called model checking. μCRL [6,10] is a toolset for analysing process algebraic specifications in ACP combined with abstract data types; it supports equational proofs with a theorem prover, as well as generation of process graphs and model checking.

7.1 Specification of the ABP

As an example, we show how the Alternating Bit Protocol (ABP) [3] can be specified in this framework. Suppose two armies have agreed to attack a city at the same time. The two armies reside on different hills, while the city lies in between these two hills. The only way for the armies to communicate with each other is by sending messengers through the hostile city. This communication is inherently unsafe; if a messenger is caught inside the city, then the message does not reach its destination. The paradox is that in such a situation, the two armies are never able to be 100% sure that they have agreed on a time to attack the city. Namely, if one army sends the message that it will attack at say 11am, then the other army has to acknowledge reception of this message, army one has to acknowledge the reception of this acknowledgement, et cetera.

The ABP is a method to ensure successful transmission of data through a corrupted channel (such as messengers through a hostile city). This success is based on the assumption that data can be resent an unlimited number of times, and that eventually each datum will be communicated through the channel successfully. The protocol layout is depicted below.



Data elements $d_1, d_2, d_3, \ldots$ from a finite set Δ are communicated between a Sender and a Receiver. If the Sender reads a datum from channel A, then this datum is communicated through channel B to the Receiver, which sends the datum into channel C. However, channel B is corrupted, so that a message that is communicated through this channel can be turned into an error message $\perp$. Therefore, every time the Receiver receives a message via channel B, it sends an acknowledgement to the Sender via channel D, which is also corrupted.

In the ABP, the Sender attaches a bit 0 to data elements d_{2k-1} and a bit 1 to data elements d_{2k}, when they are sent into channel B. As soon as the Receiver reads a datum, it sends back the attached bit via channel D, to acknowledge reception. If the Receiver receives a corrupted message, then it sends the previous acknowledgement to the Sender once more. The Sender keeps on sending a pair (d_i, b) as long as it receives the acknowledgement $1-b$ or $\perp$. When the Sender receives the acknowledgement b, it starts sending out the next datum d_{i+1} with attached bit $1-b$, until it receives the acknowledgement $1-b$, et cetera. Alternation of the attached bit enables the Receiver to determine whether a received datum is really new, and alternation of the acknowledgement enables the Sender to determine whether it acknowledges reception of a datum or of an error message.

We give a linear recursive specification of the ABP in process algebra. First, we specify the Sender in the state that it is going to send out a datum with the bit b attached to it, represented by the recursion variable S_b for $b \in \{0,1\}$:

$$\begin{aligned} S_b &= \sum_{d\in\Delta} r_{\mathrm{A}}(d)\cdot T_{db} \\ T_{db} &= (s_{\mathrm{B}}(d,b) + s_{\mathrm{B}}(\perp))\cdot U_{db} \\ U_{db} &= r_{\mathrm{D}}(b)\cdot S_{1-b} \ + \ (r_{\mathrm{D}}(1-b) + r_{\mathrm{D}}(\perp))\cdot T_{db} \end{aligned}$$

In state S_b, the Sender reads a datum d from channel A. Then it proceeds to state T_{db}, in which it sends datum d into channel B, with the bit b attached to it. However, the pair (d,b) may be distorted by the channel, so that it becomes the error message $\perp$. Next, the system proceeds to state U_{db}, in which it expects to receive the acknowledgement b through channel D, ensuring that the pair (d,b) has reached the Receiver unscathed. If the correct acknowledgement b is received, then the system proceeds to state S_{1-b}, in which it is going to send out a datum with the bit $1-b$ attached to it. If the acknowledgement is either the wrong bit $1-b$ or the error message $\perp$, then the system proceeds to state T_{db}, to send the pair (d,b) into channel B once more.

Next, we specify the Receiver in the state that it is expecting to receive a datum with the bit b attached to it, represented by the recursion variable R_b for $b \in \{0,1\}$:

$$\begin{aligned} R_b &= \sum_{d'\in\Delta} \{r_{\mathrm{B}}(d',b)\cdot s_{\mathrm{C}}(d')\cdot Q_b + r_{\mathrm{B}}(d',1-b)\cdot Q_{1-b}\} \ + \ r_{\mathrm{B}}(\perp)\cdot Q_{1-b} \\ Q_b &= (s_{\mathrm{D}}(b) + s_{\mathrm{D}}(\perp))\cdot R_{1-b} \end{aligned}$$

In state R_b there are two possibilities.

1. If in R_b the Receiver reads a pair (d', b) from channel B, then this constitutes new information, so the datum d' is sent into channel C. Then the Receiver proceeds to state Q_b, in which it sends acknowledgement b to the Sender via channel D. However, this acknowledgement may be distorted by the channel, so that it becomes

the error message $\perp$. Next, the Receiver proceeds to state R_{1-b}, in which it is expecting to receive a datum with the bit $1-b$ attached to it.

2. If in R_b the Receiver reads a pair $(d', 1-b)$ or an error message $\perp$ from channel B, then this does not constitute new information. So then the Receiver proceeds to state Q_{1-b} straight away, to send acknowledgement $1-b$ to the Sender via channel D. However, this acknowledgement may be distorted by the channel, so that it becomes the error message $\perp$. Next, the Receiver proceeds to state R_b again.

A send and a read action of the same message ((d, b), b, or $\perp$) over the same internal channel (B or D) communicate with each other:

$$\begin{array}{llll}\gamma(s_{\mathrm{B}}(d,b), r_{\mathrm{B}}(d,b)) & \equiv c_{\mathrm{B}}(d,b) & \qquad \gamma(s_{\mathrm{D}}(b), r_{\mathrm{D}}(b)) & \equiv c_{\mathrm{D}}(b) \\ \gamma(s_{\mathrm{B}}(\perp), r_{\mathrm{B}}(\perp)) & \equiv c_{\mathrm{B}}(\perp) & \qquad \gamma(s_{\mathrm{D}}(\perp), r_{\mathrm{D}}(\perp)) & \equiv c_{\mathrm{D}}(\perp)\end{array}$$

for $d \in \Delta$ and $b \in \{0,1\}$. All other communications between actions result in δ.

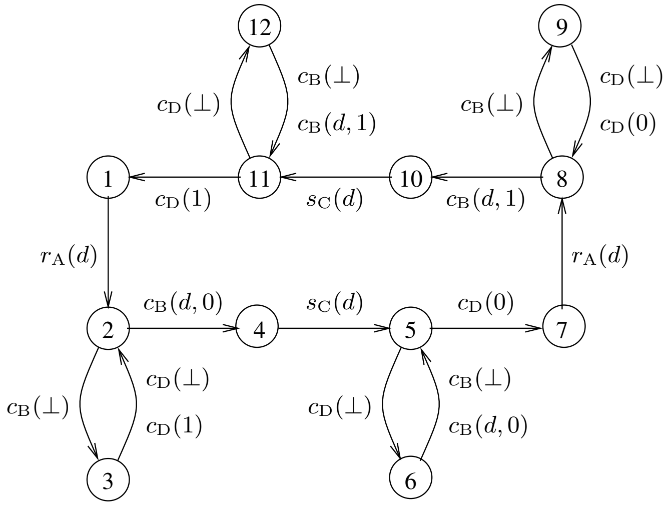
The recursive specification E of the ABP, consisting of the recursive equations for the recursion variables S_b, T_{db}, U_{db}, R_b, and Q_b for $d \in \Delta$ and $b \in \{0,1\}$, can easily be transformed into linear form by introducing extra recursion variables to represent $s_{\mathrm{C}}(d') \cdot Q_b$ for $d' \in \Delta$ and $b \in \{0,1\}$. In the remainder of this section, for notational convenience, process terms $\langle X | E \rangle$ are abbreviated to X. The desired concurrent system is obtained by putting R_0 and S_0 in parallel, encapsulating send and read actions over the internal channels B and D, and abstracting away from communication actions over these channels. That is, the ABP is expressed by the process term

$$\tau_I(\partial_H(R_0 \| S_0))$$

with

$$\begin{array}{ll} H & \equiv \{s_{\mathrm{B}}(d,b), r_{\mathrm{B}}(d,b), s_{\mathrm{D}}(b), r_{\mathrm{D}}(b) \mid d \in \Delta, b \in \{0,1\}\} \\ & \quad \cup \{s_{\mathrm{B}}(\perp), r_{\mathrm{B}}(\perp), s_{\mathrm{D}}(\perp), r_{\mathrm{D}}(\perp)\} \\ I & \equiv \{c_{\mathrm{B}}(d,b), c_{\mathrm{D}}(b) \mid d \in \Delta, b \in \{0,1\}\} \cup \{c_{\mathrm{B}}(\perp), c_{\mathrm{D}}(\perp)\}. \end{array}$$

The process graph of $\partial_H(R_0 \| S_0)$ is depicted below. Initially, in state 1, a datum d is read from channel A, resulting in state 2. Then an error message $\perp$ is communicated through channel B zero or more times, each time invoking an incorrect acknowledgement 1 or $\perp$. Finally, the pair $(d, 0)$ is communicated through channel B, resulting in state 4. Then datum d is sent into channel C, to reach state 5. The corrupted acknowledgement $\perp$ is communicated through channel D zero or more times, each time invoking a renewed attempt to communicate the pair $(d, 0)$ through channel B. Finally, acknowledgement 0 is communicated through channel D, resulting in state 7. There the same process is repeated, with the distinction that the bit 1 attached to the datum that is communicated through channel B. Note that states 2-6 and 8-12 depend on the datum d that is read from channel A.



7.2 Verification of the ABP

This section sketches an equational proof that the process algebra specification of the ABP displays the desired external behaviour; that is, the data elements that are read from channel A by the Sender are sent into channel C by the Receiver in the same order, and no data elements are lost. In other words, the process term is a solution for the guarded recursive specification

$$X \;=\; \sum_{d\in\Delta} r_{\mathrm{A}}(d)\cdot s_{\mathrm{C}}(d)\cdot X$$

where action $r_{\mathrm{A}}(d)$ represents "read datum d from channel A", and action $s_{\mathrm{C}}(d)$ represents "send datum d into channel C".

First, we derive from the axioms the six equations I-VI below, which establish the transitions between states 1-7 in the bottom half of the process graph of $\partial_H(R_0\|S_0)$.

$$\begin{aligned}
\mathrm{I}:\ & \partial_H(R_0\|S_0) && = \textstyle\sum_{d\in\Delta} r_{\mathrm{A}}(d)\cdot\partial_H(T_{d0}\|R_0)\\
\mathrm{II}:\ & \partial_H(T_{d0}\|R_0) && = c_{\mathrm{B}}(d,0)\cdot\partial_H(U_{d0}\|(s_{\mathrm{C}}(d)Q_0)) + c_{\mathrm{B}}(\perp)\cdot\partial_H(U_{d0}\|Q_1)\\
\mathrm{III}:\ & \partial_H(U_{d0}\|Q_1) && = (c_{\mathrm{D}}(1)+c_{\mathrm{D}}(\perp))\cdot\partial_H(T_{d0}\|R_0)\\
\mathrm{IV}:\ & \partial_H(U_{d0}\|(s_{\mathrm{C}}(d)Q_0)) && = s_{\mathrm{C}}(d)\cdot\partial_H(Q_0\|U_{d0})\\
\mathrm{V}:\ & \partial_H(Q_0\|U_{d0}) && = c_{\mathrm{D}}(0)\cdot\partial_H(R_1\|S_1) + c_{\mathrm{D}}(\perp)\cdot\partial_H(R_1\|T_{d0})\\
\mathrm{VI}:\ & \partial_H(R_1\|T_{d0}) && = (c_{\mathrm{B}}(d,0)+c_{\mathrm{B}}(\perp))\cdot\partial_H(Q_0\|U_{d0})
\end{aligned}$$

We start with the derivation of equation I. The process term $R_0\|S_0$ can be expanded as follows. In each step, the subterms that are reduced are underlined.

$$
\begin{aligned}
\underline{R_0\|S_0} \quad &\overset{\text{M1}}{=} \quad \underline{R_0}\mathbin{⌊\!⌊} S_0 + \underline{S_0}\mathbin{⌊\!⌊} R_0 + \underline{R_0}|\underline{S_0}\\
&\overset{\text{RDP}}{=} \quad \underline{(\textstyle\sum_{d'\in\Delta}\{r_{\rm B}(d',0)s_{\rm C}(d')Q_0 + r_{\rm B}(d',1)Q_1\} + r_{\rm B}(\bot)Q_1)\mathbin{⌊\!⌊} S_0}\\
&\qquad + \underline{(\textstyle\sum_{d\in\Delta} r_{\rm A}(d)T_{d0})\mathbin{⌊\!⌊} R_0}\\
&+ \underline{(\textstyle\sum_{d'\in\Delta}\{r_{\rm B}(d',0)s_{\rm C}(d')Q_0 + r_{\rm B}(d',1)Q_1\} + r_{\rm B}(\bot)Q_1)|(\textstyle\sum_{d\in\Delta} r_{\rm A}(d)T_{d0})}\\
&\overset{\text{LM4,CM9,10}}{=} \textstyle\sum_{d'\in\Delta}\{\underline{(r_{\rm B}(d',0)s_{\rm C}(d')Q_0)\mathbin{⌊\!⌊} S_0} + \underline{(r_{\rm B}(d',1)Q_1)\mathbin{⌊\!⌊} S_0}\}\\
&\qquad + \underline{(r_{\rm B}(\bot)Q_1)\mathbin{⌊\!⌊} S_0} + \textstyle\sum_{d\in\Delta} \underline{(r_{\rm A}(d)T_{d0})\mathbin{⌊\!⌊} R_0}\\
&+ \textstyle\sum_{d'\in\Delta}\sum_{d\in\Delta}\{\underline{(r_{\rm B}(d',0)s_{\rm C}(d')Q_0)|(r_{\rm A}(d)T_{d0})} + \underline{(r_{\rm B}(d',1)Q_1)|(r_{\rm A}(d)T_{d0})}\}\\
&\qquad + \textstyle\sum_{d\in\Delta} \underline{(r_{\rm B}(\bot)Q_1)|(r_{\rm A}(d)T_{d0})}\\
&\overset{\text{LM3,CM8}}{=} \textstyle\sum_{d'\in\Delta}\{r_{\rm B}(d',0)((s_{\rm C}(d')Q_0)\|S_0) + r_{\rm B}(d',1)(Q_1\|S_0)\}\\
&\qquad + r_{\rm B}(\bot)(Q_1\|S_0) + \textstyle\sum_{d\in\Delta} r_{\rm A}(d)(T_{d0}\|R_0)\\
&\qquad + \textstyle\sum_{d'\in\Delta}\sum_{d\in\Delta}\{\underline{\delta((s_{\rm C}(d')Q_0)\|T_{d0})} + \underline{\delta(Q_1\|T_{d0})}\}\\
&\qquad + \textstyle\sum_{d\in\Delta} \underline{\delta(Q_1\|T_{d0})}\\
&\overset{\text{A6,7}}{=} \quad \textstyle\sum_{d'\in\Delta}\{r_{\rm B}(d',0)((s_{\rm C}(d')Q_0)\|S_0) + r_{\rm B}(d',1)(Q_1\|S_0)\}\\
&\qquad + r_{\rm B}(\bot)(Q_1\|S_0) + \textstyle\sum_{d\in\Delta} r_{\rm A}(d)(T_{d0}\|R_0).
\end{aligned}
$$

Next, we expand the process term $\partial_H(R_0\|S_0)$.

$$
\begin{aligned}
\partial_H(\underline{R_0\|S_0}) \quad &= \quad \underline{\partial_H(\textstyle\sum_{d'\in\Delta}\{r_{\rm B}(d',0)((s_{\rm C}(d')Q_0)\|S_0) + r_{\rm B}(d',1)(Q_1\|S_0)\}}\\
&\qquad \underline{+ r_{\rm B}(\bot)(Q_1\|S_0) + \textstyle\sum_{d\in\Delta} r_{\rm A}(d)(T_{d0}\|R_0))}\\
&\overset{\text{D4}}{=} \textstyle\sum_{d'\in\Delta}\{\underline{\partial_H(r_{\rm B}(d',0)((s_{\rm C}(d')Q_0)\|S_0))} + \underline{\partial_H(r_{\rm B}(d',1)(Q_1\|S_0))}\}\\
&\qquad + \underline{\partial_H(r_{\rm B}(\bot)(Q_1\|S_0))} + \textstyle\sum_{d\in\Delta} \underline{\partial_H(r_{\rm A}(d)(T_{d0}\|R_0))}\\
&\overset{\text{D1,2,5}}{=} \textstyle\sum_{d'\in\Delta}\{\underline{\delta\partial_H((s_{\rm C}(d')Q_0)\|S_0)} + \underline{\delta\partial_H(Q_1\|S_0)}\} + \underline{\delta\partial_H(Q_1\|S_0)}\\
&\qquad + \textstyle\sum_{d\in\Delta} r_{\rm A}(d)\partial_H(T_{d0}\|R_0)\\
&\overset{\text{A6,7}}{=} \textstyle\sum_{d\in\Delta} r_{\rm A}(d)\partial_H(T_{d0}\|R_0).
\end{aligned}
$$

This completes the proof of equation I. Similar to equation I, we can derive the remaining equations II-VI. These derivations are sketched below.

$$
\begin{aligned}
T_{d0}\|R_0 = {} & (s_{\rm B}(d,0) + s_{\rm B}(\bot))(U_{d0}\|R_0)\\
& + \textstyle\sum_{d'\in\Delta}\{r_{\rm B}(d',0)((s_{\rm C}(d')Q_0)\|T_{d0}) + r_{\rm B}(d',1)(Q_1\|T_{d0})\}\\
& + r_{\rm B}(\bot)(Q_1\|T_{d0}) + c_{\rm B}(d,0)(U_{d0}\|(s_{\rm C}(d)Q_0)) + c_{\rm B}(\bot)(U_{d0}\|Q_1)
\end{aligned}
$$

$$\partial_H(T_{d0}\|R_0) = c_{\rm B}(d,0)\partial_H(U_{d0}\|(s_{\rm C}(d)Q_0)) + c_{\rm B}(\bot)\partial_H(U_{d0}\|Q_1)$$

$$
\begin{aligned}
U_{d0}\|Q_1 = {} & r_{\rm D}(0)(S_1\|Q_1) + (r_{\rm D}(1) + r_{\rm D}(\bot))(T_{d0}\|Q_1)\\
& + (s_{\rm D}(1) + s_{\rm D}(\bot))(R_0\|U_{d0}) + (c_{\rm D}(1) + c_{\rm D}(\bot))(T_{d0}\|R_0)
\end{aligned}
$$

$$\partial_H(U_{d0}\|Q_1) = (c_{\rm D}(1) + c_{\rm D}(\bot))\partial_H(T_{d0}\|R_0)$$

$$\begin{aligned}U_{d0}\|(s_{\mathrm{C}}(d)Q_0) &= r_{\mathrm{D}}(0)(S_1\|(s_{\mathrm{C}}(d)Q_0)) + (r_{\mathrm{D}}(1)+r_{\mathrm{D}}(\perp))(T_{d0}\|(s_{\mathrm{C}}(d)Q_0))\\ &\quad + s_{\mathrm{C}}(d)(Q_0\|U_{d0})\\ \partial_H(U_{d0}\|(s_{\mathrm{C}}(d)Q_0)) &= s_{\mathrm{C}}(d)\partial_H(Q_0\|U_{d0})\end{aligned}$$

$$\begin{aligned}Q_0\|U_{d0} &= (s_{\mathrm{D}}(0)+s_{\mathrm{D}}(\perp))(R_1\|U_{d0}) + r_{\mathrm{D}}(0)(S_1\|Q_0)\\ &\quad + (r_{\mathrm{D}}(1)+r_{\mathrm{D}}(\perp))(T_{d0}\|Q_0) + c_{\mathrm{D}}(0)(R_1\|S_1) + c_{\mathrm{D}}(\perp)(R_1\|T_{d0})\\ \partial_H(Q_0\|U_{d0}) &= c_{\mathrm{D}}(0)\partial_H(R_1\|S_1) + c_{\mathrm{D}}(\perp)\partial_H(R_1\|T_{d0})\end{aligned}$$

$$\begin{aligned}R_1\|T_{d0} &= \sum_{d'\in\Delta}\{r_{\mathrm{B}}(d',1)((s_{\mathrm{C}}(d')Q_1)\|T_{d0}) + r_{\mathrm{B}}(d',0)(Q_0\|T_{d0})\}\\ &\quad + r_{\mathrm{B}}(\perp)(Q_0\|T_{d0}) + (s_{\mathrm{B}}(d,0)+s_{\mathrm{B}}(\perp))(U_{d0}\|R_1)\\ &\quad + (c_{\mathrm{B}}(d,0)+c_{\mathrm{B}}(\perp))(Q_0\|U_{d0})\\ \partial_H(R_1\|T_{d0}) &= (c_{\mathrm{B}}(d,0)+c_{\mathrm{B}}(\perp))\partial_H(Q_0\|U_{d0})\end{aligned}$$

Note that the process term $\partial_H(R_1\|S_1)$ in the right-hand side of equation V is not the left-hand side of an equation I-VI. We proceed to expand $\partial_H(R_1\|S_1)$. That is, similar to equations I-VI, the following six equations VII-XII can be derived, which establish the transitions between states 7-12 and 1 in the top half of the process graph of $\partial_H(R_0\|S_0)$. The derivations of these equations are left to the reader.

$$\begin{aligned}\mathrm{VII}: \partial_H(R_1\|S_1) &= \textstyle\sum_{d\in\Delta} r_{\mathrm{A}}(d)\cdot\partial_H(T_{d1}\|R_1)\\ \mathrm{VIII}: \partial_H(T_{d1}\|R_1) &= c_{\mathrm{B}}(d,1)\cdot\partial_H(U_{d1}\|(s_{\mathrm{C}}(d)Q_1)) + c_{\mathrm{B}}(\perp)\cdot\partial_H(U_{d1}\|Q_0)\\ \mathrm{IX}: \partial_H(U_{d1}\|Q_0) &= (c_{\mathrm{D}}(0)+c_{\mathrm{D}}(\perp))\cdot\partial_H(T_{d1}\|R_1)\\ \mathrm{X}: \partial_H(U_{d1}\|(s_{\mathrm{C}}(d)Q_1)) &= s_{\mathrm{C}}(d)\cdot\partial_H(Q_1\|U_{d1})\\ \mathrm{XI}: \partial_H(Q_1\|U_{d1}) &= c_{\mathrm{D}}(1)\cdot\partial_H(R_0\|S_0) + c_{\mathrm{D}}(\perp)\cdot\partial_H(R_0\|T_{d1})\\ \mathrm{XII}: \partial_H(R_0\|T_{d1}) &= (c_{\mathrm{B}}(d,1)+c_{\mathrm{B}}(\perp))\cdot\partial_H(Q_1\|U_{d1})\end{aligned}$$

Thus, we can derive algebraically the relations depicted in the process graph of $\partial_H(R_0\|S_0)$. Owing to equations I-XII, RSP yields

$$\partial_H(R_0\|S_0) = \langle X_1|E\rangle \qquad (1)$$

where E denotes the linear recursive specification

$$\begin{array}{ll}\{\ X_1 = \sum_{d'\in\Delta} r_{\mathrm{A}}(d')\cdot X_{2d'}, & Y_1 = \sum_{d'\in\Delta} r_{\mathrm{A}}(d')\cdot Y_{2d'},\\ \ \ X_{2d} = c_{\mathrm{B}}(d,0)\cdot X_{4d} + c_{\mathrm{B}}(\perp)\cdot X_{3d}, & Y_{2d} = c_{\mathrm{B}}(d,1)\cdot Y_{4d} + c_{\mathrm{B}}(\perp)\cdot Y_{3d},\\ \ \ X_{3d} = (c_{\mathrm{D}}(1)+c_{\mathrm{D}}(\perp))\cdot X_{2d}, & Y_{3d} = (c_{\mathrm{D}}(0)+c_{\mathrm{D}}(\perp))\cdot Y_{2d},\\ \ \ X_{4d} = s_{\mathrm{C}}(d)\cdot X_{5d}, & Y_{4d} = s_{\mathrm{C}}(d)\cdot Y_{5d},\\ \ \ X_{5d} = c_{\mathrm{D}}(0)\cdot Y_1 + c_{\mathrm{D}}(\perp)\cdot X_{6d}, & Y_{5d} = c_{\mathrm{D}}(1)\cdot X_1 + c_{\mathrm{D}}(\perp)\cdot Y_{6d},\\ \ \ X_{6d} = (c_{\mathrm{B}}(d,0)+c_{\mathrm{B}}(\perp))\cdot X_{5d}, & Y_{6d} = (c_{\mathrm{B}}(d,1)+c_{\mathrm{B}}(\perp))\cdot Y_{5d}\\ \ \ \mid d\in\Delta\ \}. & \end{array}$$

We proceed to prove that the process term $\tau_I(\langle X_1|E\rangle)$ exhibits the desired external behaviour of the ABP. After application of the abstraction operator τ_I to the process term $\langle X_1|E\rangle$, the loops of communication actions in the process graph of $\partial_H(R_0\|S_0)$ (between states 2-3, states 5-6, states 8-9, and states 11-12) become τ-loops. These loops can be removed using CFAR. For example, for $d \in \Delta$ the recursion variables X_{2d} and X_{3d} form a cluster for I with exit $c_{\mathrm{B}}(d,0)\cdot X_{4d}$, so

$$\begin{aligned} r_{\mathrm{A}}(d)\cdot\tau_I(\langle X_{2d}|E\rangle) &\overset{\mathrm{CFAR}}{=} r_{\mathrm{A}}(d)\cdot\tau_I(c_{\mathrm{B}}(d,0)\,\langle X_{4d}|E\rangle) \\ &\overset{\mathrm{TI2,5,B1}}{=} r_{\mathrm{A}}(d)\cdot\tau_I(\langle X_{4d}|E\rangle). \end{aligned} \tag{2}$$

Similarly, CFAR together with TI2,5 and B1 can be applied to eliminate the other three loops of communication actions. Thus, we derive the following equations:

$$s_{\mathrm{C}}(d)\cdot\tau_I(\langle X_{5d}|E\rangle) = s_{\mathrm{C}}(d)\cdot\tau_I(\langle Y_1|E\rangle) \tag{3}$$
$$r_{\mathrm{A}}(d)\cdot\tau_I(\langle Y_{2d}|E\rangle) = r_{\mathrm{A}}(d)\cdot\tau_I(\langle Y_{4d}|E\rangle) \tag{4}$$
$$s_{\mathrm{C}}(d)\cdot\tau_I(\langle Y_{5d}|E\rangle) = s_{\mathrm{C}}(d)\cdot\tau_I(\langle X_1|E\rangle). \tag{5}$$

Applying RDP, TI1,4,5, and equations (2) and (3) we derive

$$\begin{aligned} \tau_I(\langle X_1|E\rangle) &\overset{\mathrm{RDP,TI1,4,5}}{=} \sum_{d\in\Delta} r_{\mathrm{A}}(d)\cdot\tau_I(\langle X_{2d}|E\rangle) \\ &\overset{(2)}{=} \sum_{d\in\Delta} r_{\mathrm{A}}(d)\cdot\tau_I(\langle X_{4d}|E\rangle) \\ &\overset{\mathrm{RDP,TI1,5}}{=} \sum_{d\in\Delta} r_{\mathrm{A}}(d)\cdot s_{\mathrm{C}}(d)\cdot\tau_I(\langle X_{5d}|E\rangle) \\ &\overset{(3)}{=} \sum_{d\in\Delta} r_{\mathrm{A}}(d)\cdot s_{\mathrm{C}}(d)\cdot\tau_I(\langle Y_1|E\rangle). \end{aligned} \tag{6}$$

Likewise, applying RDP, TI1,4,5, and equations (4) and (5) we can derive

$$\tau_I(\langle Y_1|E\rangle) = \sum_{d\in\Delta} r_{\mathrm{A}}(d)\cdot s_{\mathrm{C}}(d)\cdot\tau_I(\langle X_1|E\rangle). \tag{7}$$

Equations (6) and (7) together with RSP enable us to derive the following equation:

$$\tau_I(\langle X_1|E\rangle) = \sum_{d\in\Delta} r_{\mathrm{A}}(d)\cdot s_{\mathrm{C}}(d)\cdot\tau_I(\langle X_1|E\rangle).$$

In combination with equation (1) this yields

$$\tau_I(\partial_H(R_0\|S_0)) = \sum_{d\in\Delta} r_{\mathrm{A}}(d)\cdot s_{\mathrm{C}}(d)\cdot\tau_I(\partial_H(R_0\|S_0)).$$

In other words, the ABP exhibits the desired external behaviour. This finishes the verification of the ABP.

References

1. Aceto, L., Fokkink, W.J., Verhoef, C.: Structural operational semantics. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, pp. 197–292. Elsevier, Amsterdam (2001)
2. Baeten, J.C.M.: A brief history of process algebra. Theoretical Computer Science 335(2-3), 131–146 (2005)
3. Bartlett, K.A., Scantlebury, R.A., Wilkinson, P.T.: A note on reliable full-duplex transmission over half-duplex links. Communications of the ACM 12(5), 260–261 (1969)
4. Basten, T.: Branching bisimilarity is an equivalence indeed! Information Processing Letters 58(3), 141–147 (1996)
5. Bergstra, J., Heering, J., Klint, P.: Module algebra. Journal of the ACM 37(2), 335–372 (1990)
6. Blom, S., Fokkink, W.J., Groote, J.F., van Langevelde, I., Lisser, B., van de Pol, J.: μCRL: A toolset for analysing algebraic specification. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 250–254. Springer, Heidelberg (2001)
7. De Nicola, R., Vaandrager, F.W.: Three logics for branching bisimulation. Journal of the ACM 42(2), 458–487 (1995)
8. Fokkink, W.J.: Introduction to Process Algebra. Springer, Heidelberg (2000)
9. Fokkink, W.J.: Rooted branching bisimulation as a congruence. Journal of Computer and System Sciences 60(1), 13–37 (2000)
10. Fokkink, W.J.: Modelling Distributed Systems. Springer, Heidelberg (2007)
11. Fokkink, W.J., Verhoef, C.: A conservative look at operational semantics with variable binding. Information and Computation 146(1), 24–54 (1998)
12. van Glabbeek, R.J.: A complete axiomatization for branching bisimulation congruence of finite-state behaviours. In: Borzyszkowski, A.M., Sokolowski, S. (eds.) MFCS 1993. LNCS, vol. 711, pp. 473–484. Springer, Heidelberg (1993)
13. van Glabbeek, R.J.: What is branching time and why to use it? In: Nielsen, M. (ed.) The Concurrency Column. Bulletin of the EATCS, vol. 53, pp. 190–198 (1994)
14. Groote, J.F.: Process Algebra and Structured Operational Semantics. PhD thesis, University of Amsterdam (1991)
15. Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)
16. Moller, F.: The importance of the left merge operator in process algebras. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, pp. 752–764. Springer, Heidelberg (1990)
17. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice-Hall, Englewood Cliffs (1998)
18. Stirling, C.: Modal and Temporal Properties of Processes. Springer, Heidelberg (2001)
19. Tanenbaum, A.S.: Computer Networks. Prentice-Hall, Englewood Cliffs (1981)
20. Terese: Term Rewriting Systems. Cambridge University Press, Cambridge (2003)
21. Verhoef, C.: A congruence theorem for structured operational semantics with predicates and negative premises. Nordic Journal of Computing 2(2), 274–302 (1995)

On Several Proofs of the Recognizability Theorem[⋆]

Bruno Courcelle

Université Bordeaux-1, LaBRI, CNRS
Institut Universitaire de France
351, Cours de la Libération
33405, Talence cedex, France
courcell@labri.fr

Abstract. The lecture has presented and compared several proofs of the fundamental Recognizability Theorem that relates the Monadic Second-order definability of a set of finite graphs or relational structures and its Recognizability, this notion being defined in terms of finite congruences and not in terms of automata.

Certain sets of finite and infinite words, terms and graphs can be characterized as the sets of models of logical sentences. How do such definitions relate to other ones, given in terms of finite automata, regular expressions, equation systems or grammars ? For words and terms, Trakhtenbrot, Elgot, Büchi, Doner, Thatcher, Wright, Rabin and others have shown equivalences between *Monadic Second-Order* definability and definability by finite automata. The case of First-Order logic is also well-studied although some questions are still open. The lecture has discussed the case of Monadic Second-order definable sets of finite graphs and the extension to them of the results that are known for finite words, terms and trees (which are not identified with terms) and has been based on the book [1] in preparation.

A subset of an algebra having a finite set of operations (called its *signature*) is *recognizable* if it is saturated by a congruence having finitely many classes. Everybody knows that a language is recognizable with respect to the monoid structure on words if and only if it is accepted by a finite automaton. Recognizability taken in this algebraic sense applies to graphs, provided an algebraic structure is fixed. Several algebraic structures on graphs can be defined. The *VR algebra* is defined as follows.

Graphs are simple, their vertices are labelled by nonnegative integers. The operations of its *countable* signature F^{VR} are the (binary) disjoint union $\oplus$ and the following unary operations : the change of label a into label b, the addition of edges between any vertex labelled a and any vertex labelled b. Basic graphs are (labelled) loops and isolated vertices. This algebra is called the VR algebra because its *equational sets* are the sets defined by the *Vertex Replacement context-free graph grammars*. *Clique-width*, a graph complexity measure with

⋆ Supported by the GRAAL project of "Agence Nationale pour la Recherche".

which FPT algorithms can be parametrized, is defined in terms of these operations. We let F_k^{VR} be the finite subsignature of F^{VR} that uses only the vertex labels from $[k] := \{1, ..., k\}$. A graph has clique-width at most k if (by definition) it is defined (up to isomorphism) by a term over F_k^{VR}. Since the signature F^{VR} is countably infinite, the notion of recognizability must be adapted. The *type* of a graph is the set of labels of its vertices. A congruence witnessing recognizability is required to be *type preserving* (two equivalent graphs must have the same type) and to have finitely many classes of each type (countably many globally).

A labelled graph can be handled as an element of an algebra, but also as a logical structure, with vertex set as domain, a binary adjacency relation and unary predicates for specifying labels. The *Recognizability Theorem* says that every Monadic Second-Order (MS in short) definable set of finite graphs is recognizable with respect to the VR algebra. The converse does not hold because there are uncountably many recognizable sets of finite graphs (up to isomorphism) of type $\{1\}$, i.e., of graphs all vertices of which have label 1. This fact makes impossible to characterize the recognizable sets of graphs in terms of logical formulas or automata.

The lecture has presented two proofs of this theorem and two proofs of its following weak form. The *Weak Recognizability Theorem* says that for each k, the set of graphs of clique-width at most k that satisfy a fixed MS sentence is recognizable with respect to the subalgebra of the VR algebra generated by F_k^{VR}. This theorem does not entail the strong version.

We first recall the structure of the well-known proof (from Doner *et al.*) for finite terms. Consider set variables $X_1, ..., X_n$. The assignments of sets of positions in a term to these variables is encoded by n Booleans attached to each position in the term. For each MS formula with free variables $X_1, ..., X_n$, one constructs a finite deterministic automaton recognizing the terms with the Booleans encoding the assignments that satisfy the formula. This construction is done by induction on formulas and is based on closure properties of automata (Boolean operations, projection, determinization). One can view this construction as a *compilation* of an MS formula into a deterministic automaton, yielding a linear time model-checking algorithm.

Then we consider graphs of bounded clique-width. By *Backwards Translation*, one can translate an MS sentence φ on graphs into one, ψ, that characterizes the terms over F_k^{VR} that define graphs satisfying φ. This gives a short proof of the Weak Recognizability Theorem, but this proof is not at all satisfactory, because ψ is of larger quantifier-height than φ, and large quantifier-height makes the above mentioned compilation infeasible. By constructing directly *small automata* for checking atomic formulas (the one for adjacency over F_k^{VR} uses k^2+k+3 states), one gets the Weak Recognizability Theorem in a more direct and efficient way. However, achieving compilation is still difficult. Experimental results have been reported.

The (full) Recognizability Theorem, can be proved with a Fefermann-Vaught style proof. Its main ingredient is the *Splitting Theorem* saying that the set of n-tuples of sets that satisfy in $S \oplus T$ an MS formula $\varphi(X_1, ..., X_n)$ is a combination of similar sets for S and T relative to (auxiliary) formulas of no larger quantifier-height than φ. It follows that for each h, the equivalence relation on graphs saying that $G \approx H$ if and only if G and H have the same type and satisfy the same sentences of quantifier-height at most h is a type preserving congruence for the VR algebra with finitely many classes of each type. It saturates the set of finite models of any MS sentence of quantifier-height at most h, which proves the Theorem. This proof has useful extensions giving algorithms to compute *counting* and *optimization functions* (like distance) defined by MS formulas. See Makowsky [3] for a survey of such applications.

Finally, a proof of the Recognizability Theorem using Booleans attached to vertices to encode satisfying assignments, thus that generalizes the second proof of the Weak Recognizability Theorem, can be derived from an article by Engelfriet [2].

There is no unique best proof of the Recognizability Theorem. Each proof has some interest.

The Recognizability Theorems presented above are actually two instances of a unique theorem having a unique proof. Other important instances concern the *HR algebra of finite graphs*, from which *tree-width* can be characterized (for MS sentences using edge set quantifications), and an algebra of finite *relational structures*. See [1] where all details are given.

References

1. Courcelle, B.: Graph structure and monadic second-order logic, book in preparation, To be published by Cambridge University Press, `http://www.labri.fr/perso/courcell/Book/CourGGBook.pdf`
2. Engelfriet, J.: A Regular Characterization of Graph Languages Definable in Monadic Second-Order Logic. Theoretical Computer Science 88, 139–150 (1991)
3. Makowsky, J.: Algorithmic uses of the Feferman-Vaught Theorem. Ann. Pure Appl. Logic 126, 159–213 (2004)

Theories of Automatic Structures and Their Complexity

Dietrich Kuske

Institut für Informatik, Universität Leipzig, Germany

Abstract. For automatic structures, several logics have been shown decidable: first-order logic, its extension by the infinity quantifier, by modulo-counting quantifiers, and even by a restricted form of second-order quantification. We review these decidability proofs. As a new result, we determine the data, the expression, and the combined complexity of quantifier-classes for first-order logic. Finally, we also recall that first-order logic becomes elementary decidable for automatic structures of bounded degree.

1 Introduction

The idea of an automatic structure goes back to Büchi and Elgot who used finite automata to decide, e.g., Presburger arithmetic [9]. In essence, a structure is automatic if the elements of the universe can be represented as strings from a regular language (an element can be represented by several strings) and every relation of the structure can be recognized by a finite automaton with several heads that proceed synchronously. Automaton decidable theories [13] and automatic groups [10] are similar concepts. A systematic study was initiated by Khoussainov and Nerode [15] who also coined the name "*automatic structure*". They received increasing interest over the last years [3,4,17,18,1,16,20,2,22]; Rubin's survey [25] gives an excellent overview of the results in this area in particular regarding the structure and decidability issues. One of the main motivations for investigating automatic structures is that their first-order theories are decidable. This decidability holds even if one extends first-order logic by quantifiers "there exist infinitely many", "the number of elements satisfying φ is finite and equals (modulo q) p", and "there exists an infinite relation satisfying φ" (provided φ mentions the infinite relation only negatively).

But there exist automatic structures whose first-order theory is non-elementary (i.e., does not belong to n-EXPSPACE for any $n \in \mathbb{N}$). An inspection of the decidability proof (that we indicate in this article) shows that validity of a formula in Σ_{n+1} (i.e., with at most $n+1$ nested negations) can be decided in n-EXPSPACE. We prove this to be optimal in two very strict senses. First, we construct (for every $n \in \mathbb{N}$) a fixed formula $\varphi_n \in \Sigma_{n+1}$ such that validity in an automatic structure is complete for n-EXPSPACE (the input to this problem is a presentation of the structure by automata). Second, we also construct one automatic structure such that validity of a sentence from Σ_{n+1} is complete for

n-EXPSPACE. In other words, both the data and the expression complexity (and therefore the combined complexity) are complete for n-EXPSPACE.

In the final part, we present a class of automatic structures that allow elementary decision procedures, namely the class of structures of bounded degree [23,21].

2 Preliminaries

Let Γ be a finite alphabet and $w \in \Gamma^*$ be a finite word over Γ. The length of w is denoted by $|w|$.

2.1 Structures

A *signature* is a finite set τ of relational symbols, where every symbol $r \in \tau$ has some fixed arity m_r. Then a τ*-structure* $\mathcal{A}$ consists of a non-empty universe A and, for every $r \in \tau$, an m_r-ary relation $r^{\mathcal{A}} \subseteq A^{m_r}$. Note that we only consider relational structures. Sometimes, we will also use constants, but in our context, a constant c can be always replaced by the unary relation $\{c\}$. Let us fix a τ-structure $\mathcal{A} = (A, (r^{\mathcal{A}})_{r\in\tau})$, where $r^{\mathcal{A}} \subseteq A^{m_r}$. To simplify notation, we will write $a \in \mathcal{A}$ for $a \in A$. For $B \subseteq A$ we define the restriction $\mathcal{A}{\restriction}B = (B, (r^{\mathcal{A}} \cap B^{m_r})_{r\in\tau})$. Given further constants $a_1, \dots, a_n \in \mathcal{A}$, we write $(\mathcal{A}, a_1, \dots, a_k)$ for the structure $(A, (r^{\mathcal{A}})_{r\in\tau}, a_1, \dots, a_k)$. In the rest of the paper, we will often identify a symbol $r \in \tau$ with its interpretation $r^{\mathcal{A}}$.

A *congruence* on the structure $\mathcal{A} = (A, (r)_{r\in\tau})$ is an equivalence relation $\equiv$ on A such that for every $r \in \tau$ and all $a_1, b_1, \dots, a_{m_r}, b_{m_r} \in A$ we have: If $(a_1, \dots, a_{m_r}) \in r$ and $a_1 \equiv b_1, \dots, a_{m_r} \equiv b_{m_r}$, then also $(b_1, \dots, b_{m_r}) \in r$. As usual, the equivalence class of $a \in A$ w.r.t. $\equiv$ is denoted by $[a]_\equiv$ or just $[a]$ and $A/_\equiv$ denotes the set of all equivalence classes. We define the *quotient structure* $\mathcal{A}/_\equiv = (A/_\equiv, (r/_\equiv)_{r\in\tau})$, where $r/_\equiv = \{([a_1], \dots, [a_{m_r}]) \mid (a_1, \dots, a_{m_r}) \in r\}$.

2.2 Automatic Structures

Let us fix $n \in \mathbb{N}$ and a finite alphabet Γ. Let $\# \notin \Gamma$ be an additional padding symbol. For words $w_1, w_2, \dots, w_n \in \Gamma^*$ we define the *convolution* $w_1 \otimes w_2 \otimes \cdots \otimes w_n$, which is a word over the alphabet $(\Gamma \cup \{\#\})^n$, as follows: Let $w_i = a_{i,1}a_{i,2}\cdots a_{i,k_i}$ with $a_{i,j} \in \Gamma$ and $k = \max\{k_1, \dots, k_n\}$. For $k_i < j \le k$ define $a_{i,j} = \#$. Then $w_1 \otimes \cdots \otimes w_n = (a_{1,1}, \dots, a_{n,1}) \cdots (a_{1,k}, \dots, a_{n,k})$. Thus, for instance $aba \otimes bbabb = (a,b)(b,b)(a,a)(\#,b)(\#,b)$. An n-ary relation $R \subseteq (\Gamma^*)^n$ is called *automatic* if the language $\{w_1 \otimes \cdots \otimes w_n \mid (w_1, \dots, w_n) \in R\}$ is a regular language.

An *m-dimensional (synchronous) automaton* over Γ is just a finite automaton A over the alphabet $(\Gamma \cup \{\#\})^m$ such that $L(A) \subseteq \{w_1 \otimes \cdots \otimes w_m \mid w_1, \dots, w_m \in \Gamma^*\}$. Such an automaton defines an m-ary relation

$$R(A) = \{(w_1, \dots, w_m) \mid w_1 \otimes \cdots \otimes w_m \in L(A)\} .$$

An *automatic presentation* is a tuple $P = (\Gamma, A_0, A_=, (A_r)_{r\in\tau})$, where:

– Γ is a finite alphabet.
– τ is a signature (the signature of P), as before m_r is the arity of the symbol $r \in \tau$.
– A_0 is a finite automaton over the alphabet Γ.
– For every $r \in \tau$, A_r is an m_r-dimensional automaton over the alphabet $\Gamma \cup \{\#\}$ such that $R(A_r) \subseteq L(A_0)^{m_r}$.
– $A_=$ is a 2-dimensional automaton over the alphabet $\Gamma \cup \{\#\}$ such that the relation $R(A_=) \subseteq L(A_0) \times L(A_0)$ is a congruence on the structure $(L(A_0), (R(A_r))_{r\in\tau})$.

The structure presented by P is the quotient

$$\mathcal{A}(P) = (L(A_0), (R(A_r))_{r\in\tau})/_{R(A_=)} \ .$$

A structure $\mathcal{A}$ is called *automatic* if there exists an automatic presentation P such that $\mathcal{A} \cong \mathcal{A}(P)$. We will write $[u]$ for the element $[u]_{R(A_=)}$ ($u \in L(A_0)$) of the structure $\mathcal{A}(P)$.

By SA, we denote the set of all automatic presentations (here S indicates that we work with *string*-automata). A presentation $P = (\Gamma, A_0, A_=, (A_r)_{r\in\tau})$ is called *injective* if $R(A_=)$ is the identity relation on $L(A_0)$. In this case, we can omit the automaton $A_=$ and identify P with the tuple $(\Gamma, A_0, (A_r)_{r\in\tau})$. Then iSA denotes the set of injective automatic presentations.

Examples

– All finite structures $\mathcal{A}$ are automatic with alphabet the universe of $\mathcal{A}$. While there are many infinite automatic structures (see below), there are no infinite automatic fields [16].
– The complete binary tree with universe $\{0,1\}^*$, together with the binary relations "first son" S_0, "second son" S_1, "prefix" $\leq$, and "equal length" is automatic.
– Presburger arithmetic $(\mathbb{N}, +)$ is automatic: the alphabet is $\{0,1\}$, the language of A_0 is $\{0,1\}$ where the word $a_0 a_1 \ldots a_n$ represents the number $\sum_{0\leq i\leq n} a_i 2^i$. Differently Skolem arithmetic $(\mathbb{N}, \cdot)$ is not automatic [3]. But there is an extended notion of tree-automaticity based on tree-automata instead of finite automata. Blumensath also showed that Skolem arithmetic is tree-automatic.
– The linear order $(\mathbb{Q}, \leq)$ is automatic: The universe is $\{0,1\}^*$ with $u < v$ iff $(u \wedge v)0$ is a prefix of u or $(u \wedge v)1$ is a prefix of v (where $u \wedge v$ is the longest common prefix of u and v). This presentation is even "automatic-homogeneous": Let $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ be increasing sequences of equal length. Then there is an automatic automorphism f of $(\{0,1\}^*, \leq)$ mapping u_i to v_i [19].
– The rewrite graph $(\Sigma^*, \to)$ of every semi-Thue system and therefore the configuration graph of every Turing machine are automatic.
– The extension of this configuration graph by the binary relation of reachability is in general not automatic. But for pushdown automata, the configuration graph with reachability $(Q\Gamma^*, \to, \to^*)$ is automatic: a configuration is represented the control state followed by the stack content.

– The theory of automatic structures was preceeded by that of automatic groups [10] and semigroups [5]. In terms of automatic structures, a semigroup is automatic (in the original sense) if its Cayley-graph has an injective automatic presentation such that $L(A_0)$ forms a rational cross-section of the (semi-)group. Many natural groups and semigroups were shown to be automatic and therefore to have automatic Cayley-graphs:
  - rational monoids [26],
  - virtually free finitely generated, virtually free Abelian finitely generated, and hyperbolic groups [10],
  - singular Artin monoids of finite type [7], and
  - graph products of such monoids [11].

  In contrast, it seems that not many infinite groups are automatic in the sense of this article. For instance, a finitely generated group is automatic iff it is virtually Abelian [24].
– An automatic structure can be at most countably infinite. Hence, the ordinal ω_1 is certainly not automatic. Delhommé, Goranko, and Knapik proved that an ordinal α is automatic iff $\alpha < \omega^\omega$ [8].
– Let $\mathcal{B}$ denote the Boolean algebra of all finite and co-finite subsets of $\mathbb{N}$. Then an infinite Boolean algebra is automatic iff it is a finite power of $\mathcal{B}$ [16].

3 Model Checking

3.1 The Logic FSO

Fix a signature τ. Then let $V_0 = \{x_i \mid i \in \mathbb{N}\}$ be a countably infinite set of individual variables and, for $k \geq 1$, let $V_k = \{X_i^k \mid i \in \mathbb{N}\}$ be a set of k-ary relation variables. Formulas of FSO are then built according to the following formation rules (where α and β are formulas, $x, y, y_1, \ldots, y_k \in V_0$ are individual variables, R is a k-ary relation symbol, and $X \in V_k$ is a k-ary relation variable):

(L1) $x = y$
(L2) $R(y_1, \ldots, y_k)$
(L3) $X(y_1, \ldots, y_k)$
(L4) $\alpha \vee \beta$ and $\alpha \wedge \beta$
(L5) $\neg\alpha$
(L6) $\exists x : \alpha$
(L7) $\exists^\infty x : \alpha$
(L8) $\exists^{(p,q)} x : \alpha$ for $0 \leq p < q$
(L9) $\exists X$ infinite : α provided $X \in V_k \setminus \mathrm{pos}(\alpha)$

To complete this definition, we have to explain what $\mathrm{pos}(\alpha)$, the set of positively occuring relation variables, is. This is achieved by induction as follows:

(1) $\mathrm{pos}(x = y) = \mathrm{neg}(x = y) = \emptyset$
(2) $\mathrm{pos}(R(y_1, \ldots, y_k)) = \mathrm{neg}(R(y_1, \ldots, y_k)) = \emptyset$
(3) $\mathrm{pos}(X(y_1, \ldots, y_k)) = \{X\}$ and $\mathrm{neg}(X(y_1, \ldots, y_k)) = \emptyset$

(4) $\mathrm{pos}(\alpha \vee \beta) = \mathrm{pos}(\alpha) \cup \mathrm{pos}(\beta)$, $\mathrm{neg}(\alpha \vee \beta) = \mathrm{neg}(\alpha) \cup \mathrm{neg}(\beta)$, and similarly for $\alpha \wedge \beta$
(5) $\mathrm{pos}(\neg\alpha) = \mathrm{neg}(\alpha)$ and $\mathrm{neg}(\neg\alpha) = \mathrm{pos}(\alpha)$
(6) $\mathrm{pos}(\exists x : \alpha) = \mathrm{pos}(\exists x : \alpha)$ and $\mathrm{neg}(\exists x : \alpha) = \mathrm{neg}(\exists x : \alpha)$
(7) $\mathrm{pos}(\exists^\infty x : \alpha) = \mathrm{pos}(\alpha)$ and $\mathrm{neg}(\exists^\infty x : \alpha) = \mathrm{neg}(\alpha)$
(8) $\mathrm{pos}(\exists^{(p,q)} x : \alpha) = \mathrm{pos}(\alpha)$ and $\mathrm{neg}(\exists^{(p,q)} x : \alpha) = \mathrm{neg}(\alpha)$
(9) $\mathrm{pos}(\exists X \text{ infinite} : \alpha) = \mathrm{pos}(\alpha)$ and $\mathrm{neg}(\exists X \text{ infinite} : \alpha) = \mathrm{neg}(\alpha) \setminus \{X\}$

Before we define the semantics, we observe that FSO contains several interesting fragments:

– If we only allow the formation rules (L1,2,4,5,6), we obtain first-order logic FO.
– If, in addition, (L7) is allowed, we obtain $\mathrm{FO}[\exists^\infty]$.
– Similarly, $\mathrm{FO}[\exists^\infty, \exists^{\mathrm{mod}}]$ is obtained by allowing all the rules except (L3) and (L9).

We next define the semantics of these formulas. To this aim, let $\mathcal{A}$ be a τ-structure with universe A. An *interpretation in* $\mathcal{A}$ is a family $f = (f_k)_{k\geq 0}$ of functions with $f_0 : V_0 \to \mathcal{A}$ and $f_k : V_k \to 2^{\mathcal{A}^k}$ for all $k \geq 1$. Given such an interpretation, we set $\mathcal{A} \models_f \varphi$ (read as "φ holds in $\mathcal{A}$ under the interpretation f") iff one of the following hold

(S1) $\varphi = (x = y)$ and $f_0(x) = f_0(y)$.
(S2) $\varphi = (R(y_1, \ldots, y_k))$ and $(f_0(y_1), \ldots, f_0(y_k)) \in R^{\mathcal{A}}$.
(S3) $\varphi = (X(y_1, \ldots, y_k))$ and $(f_0(y_1), \ldots, f_0(y_k)) \in f_k(X)$.
(S4) $\varphi = (\alpha \vee \beta)$ and $\mathcal{A} \models_f \alpha$ or $\mathcal{A} \models_f \beta$, or $\varphi = (\alpha \wedge \beta)$, $\mathcal{A} \models_f \alpha$, and $\mathcal{A} \models_f \beta$.
(S5) $\varphi = \neg\alpha$ and not $\mathcal{A} \models_f \alpha$.
(S6) $\varphi = \exists x : \alpha$ and there exists $a \in \mathcal{A}$ with $\mathcal{A} \models_{f[\frac{a}{x}]} \alpha$ where $f[\frac{a}{x}] = g$ is a family of functions $(g_k)_{k\geq 0}$ with $g_k = f_k$ for all $k \geq 1$, $g_0(x) = a$, and $g_0(y) = f_0(y)$ for all $y \in V_0 \setminus \{x\}$.
(S7) $\varphi = \exists^\infty x : \alpha$ and there exist infinitely many $a \in \mathcal{A}$ with $\mathcal{A} \models_{f[\frac{a}{x}]} \alpha$.
(S8) $\varphi = \exists^{(p,q)} x : \alpha$ and the number of elements $a \in \mathcal{A}$ with $\mathcal{A} \models_{f[\frac{a}{x}]} \alpha$ is finite and congruent p modulo q.
(S9) $\varphi = \exists X \text{ infinite} : \alpha$ and there exists an infinite set $B \subseteq \mathcal{A}^k$ with $\mathcal{A} \models_{f[\frac{B}{X}]} \alpha$ (where we assume $X \in V_k$).

Note that the formulas $\exists^\infty x : \varphi$ and $\neg\exists^{(0,1)} x : \varphi$ are equivalent. Therefore, above, we did not define the fragment $\mathrm{FO}[\exists^{\mathrm{mod}}]$ since it would be equivalent with the more liberal $\mathrm{FO}[\exists^\infty, \exists^{\mathrm{mod}}]$.

It is an easy exercise to show the following: let $\mathcal{A}$ be a τ-structure, φ a formula, and suppose $f(y) = g(y)$ for all $y \in \mathrm{free}(\varphi)$, the set of variables occurring freely in φ. Then $\mathcal{A} \models_f \varphi$ iff $\mathcal{A} \models_g \varphi$. Assuming a fixed tuple of variables $(y_1, \ldots, y_n)$ with $y_i \in \mathrm{free}(\varphi)$ for all $1 \leq i \leq n$, we can therefore simply write $\mathcal{A} \models \varphi(f(y_1), \ldots, f(y_n))$ for $\mathcal{A} \models_f \varphi$. For *sentences* (i.e., formulas without free variables), it makes in particular sense to write $\mathcal{A} \models \varphi$.

3.2 The Model Checking Problem

Let $\mathsf{C} \subseteq \mathsf{SA}$ be a class of automatic presentations and $L \subseteq \mathrm{FSO}$ a set of formulas. Then

$$\{(\varphi, P) \mid \varphi \in L \text{ sentence}, P \in \mathsf{C}, \mathcal{A}(P) \models \varphi\}$$

is the *model checking problem* $\mathrm{MC}(L, \mathsf{C})$ *for* L *and* C, i.e., it is the following problem:

INPUT: a sentence φ from L and an automatic presentation P from C.
OUTPUT: Does $\mathcal{A}(P) \models \varphi$ hold?

To solve the most general model checking problem $\mathrm{MC}(\mathrm{FSO}, \mathsf{SA})$, one proceeds as follows:

Proposition 3.1 (cf. [15,3,17,22]). *Let P be an automatic presentation and $\varphi \in \mathrm{FSO}$ a formula with $\mathrm{free}(\varphi) \subseteq \{y_1, \ldots, y_m\} \subseteq V_0$. Then the relation $R = \{(u_1, \ldots, u_m) \in L(A_0)^m \mid \mathcal{A}(P) \models \varphi([u_1], \ldots, [u_m])\}$ is regular. Even more, an m-dimensional automaton for this relation can be computed.*

First assume P to be injective and therefore $\mathcal{A}(P) = (L(A_0), (R(A_r))_{r\in\tau})$. If φ is a first-order formula, then the automaton is constructed by induction on the structure of the formula φ: disjunction corresponds to the disjoint union of automata, existential quantification to projection, and negation to complementation [15]. The quantifier $\exists^\infty$ can be reduced to the first-order case as follows [3]: Let $\mathcal{B}$ be the extension of $\mathcal{A}(P)$ by the length-lexicographic order $\leq_{\mathrm{ll}}$ on $L(A_0)$. Then $\mathcal{B}$ is still automatic and a formula of the form $\exists^\infty x : \varphi$ is equivalent with $\forall y \exists x (y \leq_{\mathrm{ll}} x \wedge \varphi)$. This allows to apply the result for first-order logic. Such a reduction is not possible for the quantifiers $\exists^{(p,q)}$, but explicite automata-constructions provide the solution [17] (I recommend the presentation in [25, Proof of Thm. 3.19]). The basic idea is that a finite $(k+1)$-dimensional automaton A can be transformed into a k-dimensional that, on input of k words $(u_1, \ldots, u_k)$ determines, modulo q, the number of words u_{k+1} such that $(u_1, \ldots, u_k, u_{k+1})$ is accepted by A.

The idea for handling the remaining quantifier $\exists X$ infinite is as follows (cf. [22] for the details): Let σ be the extension of the signature τ by unary relation symbols L and C_k and $(k+1)$-ary relation symbols el_k for $k \geq 1$. Then let $\mathcal{B}$ be the structure obtained from $\mathcal{A}(P) = (L(A_0), (R(A_r))_{r\in\tau})$ as follows

- $L^{\mathcal{B}} = L(A_0)$ is the universe of $\mathcal{A}(P)$
- $C_k^{\mathcal{B}} \subseteq 2^{L(A_0)^k}$ is the set of all infinite k-ary relations on $L(A_0)$
- the universe of $\mathcal{B}$ consists of the language $L(A_0)$ and all infinite relations, i.e., $B = L(A_0) \cup \bigcup_{k\geq 1} C_k^{\mathcal{B}}$,
- the relations $r^{\mathcal{B}}$ and $r^{\mathcal{A}(P)} = R(A_r)$ coincide for $r \in \tau$, and
- $\mathrm{el}_k^{\mathcal{B}} \subseteq L(A_0)^k \times C_k^{\mathcal{B}}$ is the set of all $(k+1)$-tuples $(u_1, \ldots, u_k, X)$ with $(u_1, \ldots, u_k) \in X$.

Now the formula $\varphi \in \mathrm{FSO}$ can be easily translated into a formula ψ from $\mathrm{FO}[\exists^\infty, \exists^{\bmod}]$ with

$$\mathcal{A}(P) \models \varphi(u_1, \ldots, u_n) \iff \mathcal{B} \models \psi(u_1, \ldots, u_n) \ .$$

The article [18] (see also [25]) provides an encoding of certain infinite sets of words by infinite words: A *word comb* is an infinite set $\{s_0 s_1 \dots s_{i-1} t_i \mid i \in \mathbb{N}\} \subseteq \Sigma^*$ such that $0 < |s_i| < |t_i|$ for all $i \in \mathbb{N}$. A relation $R \subseteq (\Gamma^*)^k$ can be encoded if the set of convolutions $\{w_1 \otimes w_2 \cdots \otimes w_k \mid (w_1, w_2, \dots, w_k) \in R\} \subseteq (\Gamma \cup \{\#\})^*$ is a word comb. Then a compactness argument shows that any infinite k-ary relation on finite words contains an infinite subset that can be encoded in this particular way. In a second step, we restrict the sets $C_k^{\mathcal{B}}$ to these "encodeable" relations and denote the resulting structure by $\mathcal{B}'$. The restriction in formation rule (L9) then ensures

$$\mathcal{B} \models \psi(u_1, \dots, u_n) \iff \mathcal{B}' \models \psi(u_1, \dots, u_n) .$$

The particular coding from [18] ensures that the structure $\mathcal{B}'$ has an injective "ω-automatic presentation" P'. These ω-automatic presentations are defined in the same way as automatic presentations, but using Büchi- instead of finite automata. Techniques similar to the above for $\mathrm{FO}[\exists^\infty, \exists^{\mathrm{mod}}]$ provide a k-dimensional Büchi-automaton for the relation defined by ψ in $\mathcal{B}'$ [20] that can be transformed into a k-dimensional finite automaton for the relation defined by φ in $\mathcal{A}(P)$.

Now let P be non-injective. Then the set of all words from $L(A_0)$ that are length-lexicographically minimal in their equivalence class (as determined by the automaton $A_=$) is effectively regular [15]. This allows to compute an equivalent injective presentation $P' = (B_0, (B_r)_{r \in \tau})$ with $L(B_0) \subseteq L(A_0)$ and $R(B_r) = R(A_r) \cap L(B_0)^k$ for all k-ary relation symbols $r \in \tau$. Then by the above, one can compute an m-dimensional automaton A with $R(A) = \{(v_1, \dots, v_m) \in L(B_0)^m \mid \mathcal{A}(P') \models \varphi(v_1, \dots, v_m)\}$. Hence $P_h = (A_0, A, A_=)$ is an injective automatic presentation and R is the set of tuples $(u_1, \dots, u_m) \in L(A_0)^m$ with

$$\mathcal{A}(P_h) \models \exists v_1, \dots, v_m : (v_1, \dots, v_m) \in R \wedge \bigwedge_{1 \le i \le m} (u_i, v_i) \in R(A_=) .$$

This finishes the proof of Prop. 3.1.

Now we come to a direct consequence of Prop. 3.1: to decide whether the FSO-sentence φ holds, one adds a dummy variable $x \in V_0$ and then computes an automaton A with $L(A) = \{u \in L(A_0) \mid \mathcal{A} \models \varphi([u])\}$. Then $\mathcal{A} \models \varphi$ iff $L(A) \neq \emptyset$ which is decidable.

Theorem 3.2 (cf. [15,3,17,22]). *The model checking problem* $\mathrm{MC}(\mathrm{FSO}, \mathsf{SA})$ *for all automatic presentations is decidable. In particular, the* FSO*-theory* $\{\varphi \in \mathrm{FSO} : \mathcal{A}(P) \models \varphi\}$ *(that corresponds to* $\mathrm{MC}(\mathrm{FSO}, \{P\})$*) of every automatic structure* $\mathcal{A}(P)$ *is decidable.*

Since the first-order theory of the binary tree $(\{0,1\}, S_0, S_1, \le)$ is non-elementary (cf. [6, Example 8.1]), there cannot be an elementary algorithm for deciding the model checking problem $\mathrm{MC}(\mathrm{FSO}, \mathsf{SA})$. The following two sections analyse this situation a bit further for first-order logic.

Since the number of nested negations will be crucial in this analysis, we define the following classes of formulas of FO:

– Σ_0 is the set of quantifier-free formulas, i.e., those build according to the formation rules (L1,2,4,5).
– $B\Sigma_n$ is the set of Boolean combinations of formulas from Σ_n, i.e., we close the set Σ_n with respect to the formation rules (L4,5).
– Σ_{n+1} is the closure of the set $B\Sigma_n$ with respect to the formation rules (L4,6).

By de Morgan's rules, we can eliminate from every Σ_0-formula any nesting of negations. Since $\Sigma_0 = B\Sigma_0$ and since the formation of Σ_1 does not involve additional negations, the same applies to this set. By induction, we can rewrite every $B\Sigma_n$-formula (for $n > 0$) such that it has at most $n+1$ nested negations and the same applies to Σ_{n+1}-formulas. Note that this process does not increase the size of the formula.

So let $\varphi \in \Sigma_{n+1}$ have at most $n+1$ nested negations and consider the proofs of Prop. 3.1 and Theorem 3.2. Since each complementation increases the size of the automaton exponentially, the automaton A from Prop. 3.1 has $(n+1)$-fold exponential size (in the formula φ and the presentation P). Since non-emptiness of the language of a finite automaton is in NL, the decision procedure indicated above requires n-fold exponential space. Thus, we have the following more precise statement of Theorem 3.2 for first-order logic:

Proposition 3.3. *For all $n \geq 0$, the model checking problem* $\mathrm{MC}(\Sigma_{n+1}, \mathsf{SA})$ *belongs to n-$\mathsf{EXPSPACE}$ (where 0-$\mathsf{EXPSPACE} = \mathsf{PSPACE}$).*

From the result of the following sections, it will follow that this upper bound is optimal.

3.3 Data Complexity

In this section, we want to construct, for every $n \in \mathbb{N}$, a first-order sentence $\varphi_n \in \Sigma_{n+1}$ such that the question "Does φ_n hold in the structure $\mathcal{A}(P)$?" is hard for n-fold exponential space. To this aim, we define the following family of functions $F_n : \mathbb{N} \to \mathbb{N}$ by induction:

$$F_0(m) = m \text{ and } F_{n+1}(m) = F_n(m) \cdot 2^{F_n(m)} \ .$$

Note that $F_n(m)$ is an n-fold exponential function. Hence there is a deterministic Turing machine M with an n-$\mathsf{EXPSPACE}$-complete language that runs in space $F_n(|w|) - 2$ for every sufficiently long input w. Let Q be the set of states of M, $q_0 \in Q$ the initial state, $q_f \in Q$ the accepting state, and Γ_{tape} the tape alphabet. For $m \in \mathbb{N}$, an m-configuration of M is a word from $\Gamma_{\text{tape}}^* Q \Gamma_{\text{tape}}^+$ of length $F_n(m) - 1$ ($u\,q\,v$ denotes the configuration with tape content uv, control state q, and head position the first symbol of v). By $\vdash$, we denote the one-step relation of M. An *m-computation* of M is a word $\$c_0\$c_1 \dots \$c_k\$$ over $\Gamma = Q \cup \Gamma_{\text{tape}} \cup \{\$\}$ where $k \in \mathbb{N}$ is arbitrary, $c_0, c_1, \dots, c_k$ are m-configurations of equal length with $c_i \vdash c_{i+1}$ for all $0 \leq i < k$, and $\$$ is an additional delimiter. An m-computation is *successful* if $c_k \in \Gamma_{\text{tape}}^* q_f \Gamma_{\text{tape}}^+$, it is *with input* $w \in \Gamma_{\text{tape}}^*$ if $c_0 \in q_0 w \square^*$ where $\square$ is the blank symbol of the machine M.

Now let w be some input word and let m be its length. We construct an injective automatic presentation P_w such that the acceptance of w by M is equivalent to validity of a formula $\varphi_n \in \Sigma_{n+1}$ that does not depend on the word w. The structure $\mathcal{A}(P_w)$ consists of two parts that both depend on the input word w: the alphabet of the first is Γ, that of the second is $\{0,1\}$. Later, we will present formulas $\lambda_i(s) \in \Sigma_i$ such that $\mathcal{A}(P_w) \models \lambda_i(s)$ for $s \in \{0,1\}^*$ iff $s \in L_i = 0^*10^{F_i(m)-1}10^*$, i.e., iff $s \in \{0,1\}^*$ contains precisely two occurrences of 1 and these two occurrences are $F_i(m)$ apart. But first, we describe the first part of the structure $\mathcal{A}(P_w)$: Its universe is the set Γ^* and its core is the binary relation StepInBlocks. A pair of words (c, c') belongs to StepInBlocks if and only if

- $c = \$c_0\$c_1\$\ldots\$c_k\$$ and $c' = \$c'_0\$c'_1\$\ldots\$c'_k\$$ for some configurations c_i and c'_i,
- $c_0 = c'_0$,
- $|c_i| = |c'_i|$ and $c_i \vdash c'_i$ for all $1 \le i \le k$, and
- $c_k \in \Gamma^*_{\text{tape}} q_f \Gamma^+_{\text{tape}}$ is some accepting configuration.

Then w is accepted by M iff there exists a word $c = \$c_0\$c_1\$\ldots\$c_k\$$ with $c_i \in (\Gamma \setminus \{\$\})^*$ such that

(A1) $c_0 \in q_0 w \,\Box^*$ is of length $F_n(m) - 1$ and
(A2) $(\$c_0\$c_0\$c_1\$\ldots\$c_{k-1}\$, \$c_0\$c_1\$\ldots\$c_k\$) \in$ StepInBlocks.

To express (A1), we use the following two automatic relations:

- W^w is unary and consists of all words from $\$q_0 w \,\Box^*\$$.
- Prefix is binary and consists of all pairs $(u, v) \in \Gamma^* \times \Gamma^*$ where u is a prefix of v.

Let $x \in W^w$ be some prefix of c from W^w (i.e., $x = \$c_0\$$). To express that c_0 is of length $F_n(m) - 1$, we will actually express that the delimiter \$ occurs at positions in x that are $F_n(m)$ apart. To this and later purposes, we will use the following ternary relation:

- EqLet $\subseteq \{0,1\}^* \times [(\Gamma^* \times \Gamma^*) \cup (\{0,1\}^* \times \{0,1\}^*)]$ is the set of triples $(0^{k_0-1}10^{k_1-1}10^{k_2}, x, y)$ such that the letter at position k_0 in x equals the letter at position $k_0 + k_1$ in y.

Recall that $\lambda_n(s)$ is a formula that defines the set $L_n = 0^*10^{F_n(m)-1}10^*$. Then (A1) is equivalent to

$$\begin{aligned} \mathcal{A}(P_w) \models \exists x :\quad & W^w(x) \wedge \text{Prefix}(x, c) \\ & \wedge \exists x' : x' \in 1\{0,1\}^* \wedge \lambda_n(x') \wedge \text{EqLet}(x', x, x) \ . \end{aligned} \tag{1}$$

Here, the second line ensures that the letters number 1 and $F_i(m) + 1$ of x are equal. Hence $x \in W^w = \$q_0 w 0^*\$$ equals $\$q_0 w 0^{F_n(m)-m-2}\$$. Since it is a prefix of $c = \$c_0\$c_1 \ldots c_k\$$, Eq. 1 is equivalent to $c_0 = q_0 w 0^{F_n(m)-m-2}$ and therefore to (A1).

Next we argue that, given (A1) (and therefore in particular $|c| > F_n(m)$), (A2) is equivalent to

$$\begin{aligned}\mathcal{A}(P_w) \models \exists c' : \quad & \mathrm{StepInBlocks}(c', c) \\ & \wedge \forall x' : (\lambda_n(x') \wedge |x'| \leq |c'| \rightarrow \mathrm{EqLet}(x', c, c'))\ .\end{aligned} \tag{2}$$

By StepInBlocks(c', c), the words c and c' have the same length, are sequences of configurations, and c' starts with $\$c_0\$$. The second conjunct expresses that, for all $0 < i \leq |c'| - F_n(m)$, the i^{th} letter of c and the letter number $i + F_n(m) + 1$ of c' coincide, i.e., c' is the prefix of $\$c_0 c$ of length $|c|$. But this is equivalent with $c' = \$c_0\$c_0\$c_1 \ldots \$c_{k-1}\$$ as required by (A2).

It remains to present the formula λ_n that is build by induction and uses not-yet-defined relations on the second part of $\mathcal{A}(P_w)$. Before we present these relations, we need the following auxiliary definitions:

- For $x = x_0 x_1 \ldots x_k$ with $x_i \in \{0, 1\}$, let $\mathrm{val}(x) = \sum_{0 \leq i \leq k} x_i 2^i$, i.e., the word x is considered as binary number written with the least significant bit first.
- For $x = x_1 x_2 \ldots x_k$ with $x_i \in \{0, 1\}^*$ and $1 \leq i \leq j \leq k$, let $x[i, j) = x_i x_{i+1} \ldots x_{j-1}$ be the factor of x from position i to position $j - 1$.

The ternary relation DecBlocks is the core of the second part of the automatic structure $\mathcal{A}(P_w)$, it is very similar to the relation StepInBlocks from the first part:

- Let $x \in \{0, 1\}^*$ such that $0 < k_0 < k_1 \cdots < k_\ell$ are the positions of 1 in x. Then the triple (x, y, z) of words of equal length belongs to DecBlocks iff
  1. $\mathrm{val}(y[k_0, k_1)) = 0$ (i.e., $y[k_0, k_1) = 0^{k_1 - k_0}$),
  2. $\mathrm{val}(y[k_j, k_{j+1})) - 1 = \mathrm{val}(z[k_j, k_{j+1}))$ for all $1 \leq j < \ell$, and
  3. $\mathrm{val}(y[k_{\ell-1}, k_\ell)) = 2^{k_\ell - k_{\ell-1}} - 1$ (i.e., $y[k_{\ell-1}, k_\ell) = 1^{k_\ell - k_{\ell-1}}$).

The idea is that the first word divides the second and third into blocks that are decremented separately. In addition, the first and last blocks of the second word have the minimal and maximal possible value. Note in particular that y has only one block of value 0 (since all the other can be decremented).

Further relations on the second part of $\mathcal{A}(P_w)$ are the following that all can be accepted by automata whose size is polynomial in the size m of the input word w:

- $L^w = 0^* 1 0^{m-1} 1 0^*$.
- $S^w = 0^* (1 0^{m-1})^+ 1 0^*$ is the set of words with at least two occurrences of 1 such that consecutive occurrences are m positions apart.
- S_1 is the set of pairs $(x, 0^{k_0 - 1} 1 0^{k_1 - 1} 1 0^{k_2 - 1})$ of words of equal length such that k_0 and $k_0 + k_1$ are the first and last occurrences of the letter 1 in x.
- T^w is the set of pairs (x, y) of words of equal length such that, for some $1 \leq k_0 \leq |y| - m$, the letter at position k_0 in x is different from the letter at position $k_0 + m$ in y.

– S_2 is the set of pairs $(0^{k_0-1}10^{k_1-1}10^{k_2}, y)$ of words of equal length such that the positions k_0 and k_0+k_1 are either consecutive occurrences of 1 in y, or they both are occurrences of 0 in y.

Next we define formulas $\lambda_i \in \Sigma_i$ that define the sets L_i for $0 \leq i \leq n$: The formula $\lambda_0(s) = (L^w(s))$ is the trivial starting point. Let $\lambda_1(s)$ denote the following formula:

$$\exists x_1, x_2, x_3 : S^w(x_1) \wedge S_1(x_1, s) \wedge \neg T^w(x_2, x_3) \wedge \mathrm{DecBlocks}(x_1, x_2, x_3)$$

The formula $S^w(x_1)$ ensures $x_1 = 0^a(10^{m-1})^\ell 10^b$ for some $a, b, \ell \in \mathbb{N}$, $\ell \geq 1$. Then $S_1(x_1, s)$ expresses $s = 0^a 10^{\ell m-1}10^b$. By $\mathrm{DecBlocks}(x_1, x_2, x_3)$, we know $|x_1| = |x_2| = |x_3|$ and

1. $\mathrm{val}(x_2[a, a+m)) = 0$,
2. $\mathrm{val}(x_2[a+jm, a+(j+1)m)) - 1 = \mathrm{val}(x_3[a+jm, a+(j+1)m))$ for all $1 \leq j < \ell$, and
3. $\mathrm{val}(x_2[a+(\ell-1)m, a+\ell m)) = 2^m - 1$.

Finally, the formula $\neg T^w(x_2, x_3)$ expresses that for all $0 < k \leq |x_2| - m$, the letter at position k in x_2 equals that at position $k+m$ in x_3. Hence we have

$$\begin{aligned}\mathrm{val}(x_2[a+jm, a+(j+1)m)) - 1 &= \mathrm{val}(x_3[a+jm, a+(j+1)m)) \\ &= \mathrm{val}(x_2[a+(j-1)m, a+jm))\end{aligned}$$

for all $1 \leq j < \ell$, i.e., the blocks (as determined by x_1) in x_2 carry consecutive numbers from 0 to $2^m - 1$. Since $\mathrm{DecBlocks}(x_1, x_2, x_3)$ also implies that no other than the first block of x_2 carries 0, we have precisely 2^m blocks of length m each. Hence $1 + (\ell m - 1) = m \cdot 2^m = F_1(m)$ which is equivalent with $s \in L_1$. Thus, indeed, $\lambda_1(s) \in \Sigma_1$ defines the set L_1.

Now we proceed by induction on i and let λ_{i+1} denote the following formula:

$$\begin{aligned}\exists x_1, x_2, x_3 : {} & S_1(x_1, s) \wedge \mathrm{DecBlocks}(x_1, x_2, x_3) \wedge \\ & x_2 \notin 0^* \cup 1^* \wedge \exists x_4 : \lambda_i(x_4) \wedge |x_4| \leq |x_2| \wedge \\ & \forall x_4 : \lambda_i(x_4) \wedge |x_4| \leq |x_2| \rightarrow S_2(x_4, x_2) \wedge \\ & \forall x_4 : \lambda_i(x_4) \wedge |x_4| = |x_3| \rightarrow \mathrm{EqLet}(x_4, x_2, x_3)\end{aligned}$$

By the quantified formula in the second line and the induction hypothesis, $|x_2| > F_i(m) \geq m$. Hence the third line ensures that x_2 is of the form $0^a(10^{F_i(m)-1})^\ell 10^b$ for some $a, b, \ell \in \mathbb{N}$ with $\ell \geq 1$. Then the last line expresses that, for all $0 < k \leq |x_3| - F_i(m)$, the letter at position k in x_2 equals the letter at position $k + F_i(m)$ in x_3. In particular,

$$\begin{aligned}x_2[a+kF_i(m), & a+(k+1)F_i(m)) \\ &= x_3[a+(k+1)F_i(m), a+(k+2)F_i(m))\end{aligned} \tag{3}$$

for all $0 \leq k < \ell$. Now, by the first line, $s = 0^a 10^{\ell F_i(m)-1}10^b$ and

- $\mathrm{val}(x_2[a, a + F_i(m))) = 0$,
- $\mathrm{val}(x_2[a+kF_i(m), a+(k+1)F_i(m))-1 = \mathrm{val}(x_3[a+kF_i(m), a+(k+1)F_i(m))$ for all $0 < k < \ell$, and
- $\mathrm{val}(x_3[a + (\ell - 1)F_i(m), a + \ell F_i(m))) = 2^{F_i(m)} - 1$.

Now, together with Eq. 3, we obtain as above

$$\begin{aligned}\mathrm{val}(x_2[a + kF_i(m), &a + (k + 1)F_i(m))) - 1 \\ &= \mathrm{val}(x_3[a + kF_i(m), a + (k + 1)F_i(m))) \\ &= \mathrm{val}(x_2[a + (k - 1)F_i(m), a + kF_i(m)))\end{aligned}$$

for all $1 \leq k < \ell$. As above, this ensures $1 + (\ell m - 1) = F_i(m) \cdot 2^{F_i(m)} = F_{i+1}(m)$ which is equivalent with $s \in L_{i+1}$.

Assuming by induction $\lambda_i \in \Sigma_i$, we obtain $\lambda_{i+1} \in \Sigma_{i+1}$ as required.

Now we can complete the definition of the automatic structure $\mathcal{A}(P_w)$: Its universe is the set $\Gamma^* \cup \{0, 1\}^*$ and it has the following automatic relations:

- W^w, L^w, S^w, and T^w (these relations depend on the word w).
- StepInBlocks, Prefix, EqLet, DecBlocks, S_1, S_2, $1\{0, 1\}^*$, and $0^* \cup 1^*$ (these relations are independent from the word w).

Summarizing, we have the following: an input word w of length m is accepted by the machine M if and only if $\mathcal{A}(P_w) \models \exists c : C \wedge \alpha_1 \wedge \alpha_2$ where α_1 and α_2 are the Σ_{n+1}-formulas from Eq. 1 and Eq. 2, resp. Note that these formulas are independent from the word w and that the automata from P_w can be computed from w in polynomial time. Since the language of the machine M is complete for n-$\mathsf{EXPSPACE}$, we therefore proved

Proposition 3.4. *For $n \geq 0$, there exists a sentence $\varphi_n \in \Sigma_{n+1}$ such that the model checking problems $\mathrm{MC}(\{\varphi\}, \mathsf{SA})$ and $\mathrm{MC}(\{\varphi\}, \mathsf{iSA})$ are complete for n-$\mathsf{EXPSPACE}$.*

From Propositions 3.3 and 3.4, we obtain immediately

Corollary 3.5. *For all $n \in \mathbb{N}$, the model checking problems $\mathrm{MC}(\Sigma_{n+1}, \mathsf{SA})$ and $\mathrm{MC}(\Sigma_{n+1}, \mathsf{iSA})$ are complete for n-$\mathsf{EXPSPACE}$.*

3.4 Expression Complexity

In the previous section, we constructed a fixed sentence $\varphi_n \in \Sigma_{n+1}$ and, from an input word w, an automatic presentation P_w such that acceptence of w is equivalent to validity of φ_n in $\mathcal{A}(P_w)$. In this section, we proceed complementary: we construct a fixed automatic presentation P and, from an input word w, a sentence $\varphi_n^w \in \Sigma_{n+1}$ such that acceptence of w is equivalent to validity of φ_n^w in $\mathcal{A}(P)$. Consequently, we will prove that the Σ_{n+1}-theory of $\mathcal{A}(P)$ is hard for n-$\mathsf{EXPSPACE}$. Note the subtlety that the presentation P is even independent from n, i.e., the hardness holds for all $n \in \mathbb{N}$. Therefore, we now assume the Turing machine M to be universal. Furthermore, we define the following functions

$G_n : \mathbb{N} \to \mathbb{N}$ that are a slight variation of the functions F_n from the previous section:

$$G_0(m) = m, G_1(m) = 2^m, \text{ and } G_{n+1}(m) = G_n(m) \cdot 2^{G_n(m)}$$

Note that $G_n(m)$ is an n-fold exponential function. Hence, for every $n \in \mathbb{N}$, the following language is complete for n-EXPSPACE:

$$M_n = \{w \in L(M) \mid M \text{ accepts } w \text{ in space } G_n(|w|) - 2\}$$

Now let $w = a_1 a_2 \dots a_m$ be some input word of length m. We construct an injective automatic presentation P (that does not depend on w) such that the acceptance of w by M in space $G_n(|w|)$ is equivalent to validity of a formula $\varphi_n^w \in \Sigma_{n+1}$ of polynomial size. As before, the structure $\mathcal{A}(P)$ consists of two parts: the alphabet of the first is Γ, that of the second is $\{0,1\}$. Later, we will present formulas $\lambda'_i(s) \in \Sigma_i$ such that $\mathcal{A}(P) \models \lambda'_i(s)$ for $s \in \{0,1\}^*$ iff $s \in L'_i = 0^*10^{G_i(m)-1}10^*$, i.e., iff $s \in \{0,1\}^*$ contains precisely two occurrences of 1 and these two occurrences are $G_i(m)$ apart.

But first, we describe the first part of the structure $\mathcal{A}(P)$. Recall that the relation W^w is the only one in the first part of $\mathcal{A}(P_w)$ that depends on the input word w. It is therefore our task to replace it by relations independent from w, and then express membership in W^w by a small and simple formula. To this aim, we use the following relations:

– Succ_a for $a \in \Gamma \cup \{0,1\}$ consists of all pairs (u, ua) with $u \in \Gamma^*$ if $a \in \Gamma$ and $u \in \{0,1\}^*$ if $a \in \{0,1\}$.
– $\mathrm{Succ}_{\square^*\$}$ consists of all pairs (u, uv) with $u \in \Gamma^*$ and $v \in \square^*\$$.

Now consider the following formula:

$$\begin{aligned}\exists x, x_0, \dots, x_m : \quad & \mathrm{Succ}_{q_0}(\varepsilon, x_0) \wedge \bigwedge_{0 \le i < m} \mathrm{Succ}_{a_{i+1}}(x_i, x_{i+1}) \\ & \wedge \mathrm{Succ}_{\square^*\$}(x_m, x) \wedge \mathrm{Prefix}(x, c) \\ & \wedge \exists x' : x' \in 1\{0,1\}^* \wedge \lambda'_n(x') \wedge \mathrm{EqLet}(x', x, x)\end{aligned} \tag{4}$$

Given the relations Succ_x, it is equivalent to the formula from Eq. 1, but this time, it depends on the word w. This completes the changes regarding the first part of the automatic structure.

The second part of the structure $\mathcal{A}(P_w)$ contains the relations L^w, S^w, and T^w that all depend on the word w and therefore have to be replaced. The following formula $\lambda'_0(s)$

$$s \in 0^*10^*10^* \wedge \exists x_0, x_1, \dots, x_m : \left(\begin{array}{c} \mathrm{Succ}_1(x_0, x_1) \wedge \bigwedge_{1 \le i < m} \mathrm{Succ}_0(x_i, x_{i+1}) \\ \wedge \mathrm{Succ}_1(x_{m-1}, x_m) \wedge \mathrm{Prefix}(x_m, s) \end{array} \right)$$

is equivalent with $\lambda_0(s)$, i.e., $\lambda'_0(s)$ holds iff $s \in L'_0 = 0^*10^{m-1}10^*$. But differently from $\lambda_0(s)$, it belongs to Σ_1.

Next we deal with the formula λ'_1 that defines the set $L'_1 = 0^*10^{G_1(m)-1}10^*$. Consider the two automata A_0 and A_1 from Fig. 1 that accept the relations

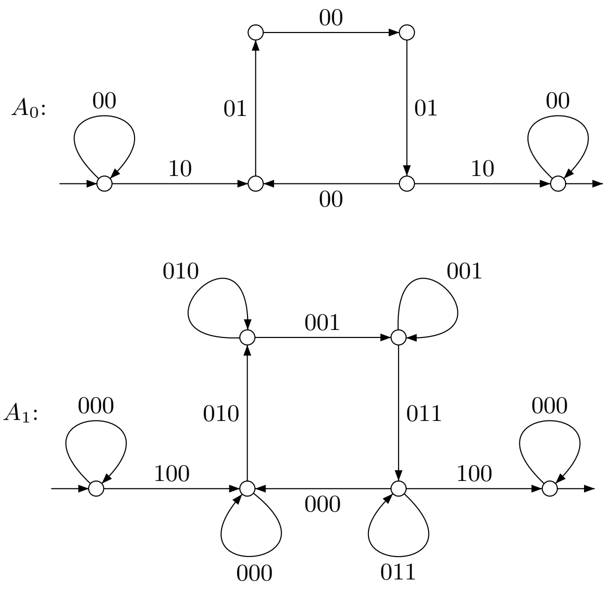


Fig. 1. The automata for R_0 and R_1

$R(A_0) = R_0 \subseteq (\{0,1\}^*)^2$ and $R(A_1) = R_1 \subseteq (\{0,1\}^*)^3$. Now $\lambda_1'(s)$ is the following formula from Σ_1:

$$\exists x_1, \dots, x_m : R_0(s, x_1) \wedge \bigwedge_{1 \leq i < m} R_1(s, x_i, x_{i+1}) \wedge x_m \in 0^*1^*0^*$$

Then $R_0(s, x_1)$ ensures $s = 0^a 1 0^{b-1} 1 0^c$ for some $a, b, c \in \mathbb{N}$ and all the words $x_1, \dots, x_m$ are of the form $0^a 0\{0,1\}^{b-1} 0 0^c$ by $R_0(s, x_1)$ and $R_1(s, x_i, x_{i+1})$, resp. From $R_0(s, x_1)$, we also obtain $x_1 = 0^a (01)^{\frac{1}{2}b} 0 0^c$ (in particular, b is even). Now consider $R_1(s, x_1, x_2)$: it expresses that, at any position between the two occurrences of 1 in s, the digit in the word x_1 drops from 1 to 0 iff the digit in the word x_2 changes. Hence $R_1(s, x_1, x_2)$ ensures $x_2 = 0^a (0011)^{\frac{1}{4}b} 0 0^c$. By induction, we obtain $x_i = 0^a (0^{2^{n-1}} 1^{2^{n-1}})^{\frac{1}{2^n}b} 0 0^c$ and therefore in particular $x_m = 0^a (0^{2^{m-1}} 1^{2^{m-1}})^{\frac{1}{2^m}b} 0 0^c$. Since x_m is from $0^*1^*0^*$, this implies $b = 2^m = G_1(m)$. Conversely, only these words x_i make the above formula true so that, indeed, $\lambda_1'(s)$ expresses $s \in 0^* 1 0^{G_1(m)-1} 1 0^* = L_1'$ as required.

To define the formulas $\lambda_i'(s) \in \Sigma_i$ for $i > 1$, we can proceed as in the previous section.

Now we can complete the definition of the automatic structure $\mathcal{A}(P_w)$: Its universe is the set $\Gamma^* \cup \{0,1\}^*$ and it has the following automatic relations that are all independent from the word w:

– Succ_a, $\mathrm{Succ}_{\Box^*\$}$, R_0, R_1, StepInBlocks, Prefix, EqLet, DecBlocks, S_1, S_2, $1\{0,1\}^*$, $0^* \cup 1^*$, $0^*10^*10^*$, and $0^*1^*0^*$.

Summarizing, we have the following: an input word w of length m is accepted by the machine M in space $G_n(m)$ if and only if $\mathcal{A}(P) \models \exists c : C \wedge \alpha_4 \wedge \alpha_2'$ where α_4 is the Σ_{n+1}-formula from Eq. 4 and α_2' arises from α_2 (cf. Eq. 2) by replacing λ_n by λ_n'. Note that this formula can be computed from the input word w in polynomial time. Since the language M_n is complete for n-$\mathsf{EXPSPACE}$, we therefore proved the following analog of Prop. 3.4 which is, at the same time, a strengthening of Cor. 3.5:

Proposition 3.6. *There exists an injective automatic presentation P such that, for all $n \geq 0$, the model checking problems* $\mathrm{MC}(\Sigma_{n+1}, \{P\})$ *and* $\mathrm{MC}(\Sigma_{n+1}, \{P\})$ *are complete for n-$\mathsf{EXPSPACE}$.*

3.5 Bounded Degree

From Cor. 3.5, it follows immediately that $\mathrm{MC}(\mathrm{FO}, \mathsf{SA})$ cannot be decided in elementary space (although it is decidable by Theorem 3.2). In this section, we present a class of automatic presentations SAb such that $\mathrm{MC}(\mathrm{FO}, \mathsf{SAb})$ becomes elementary.

In the following, let $P = (\Gamma, A_0, A_=, (A_r)_{r\in\tau})$ be an automatic presentation and let $\mathcal{A} = \mathcal{A}(P)$. The decision procedure will be based on the naïve algorithm from Fig. 2 for deciding whether $\mathcal{A} \models \varphi$. The problem with this naïve approach

```
1   check($\varphi(\bar{x}), \bar{u}$) : $\{0,1\}$
          ($\varphi(\bar{x})$ formula of quantifier rank $\leq d$
          with $k = |\bar{u}| = |\bar{x}|$ many free variables,
          $\bar{u}$ tuple of words from $L(A_0)$)
2     case $\varphi = R(\bar{x})$
3       if $\bar{u} \in L(A_R)$ then return(1) else return(0) endif
4     case $\varphi = \alpha \vee \beta$
5       return(max(check($\alpha, \bar{u}$), check($\beta, \bar{u}$)))
6     case $\varphi = \neg\alpha$
7       return(1 − check($\alpha, \bar{u}$))
8     case $\varphi = \exists y : \alpha(\bar{x}, y)$
9       return(max{check($\varphi_1, (\bar{u}, v)$) | $v \in L(A_0)$})
```

Fig. 2. The naïve algorithm

is that the computation in line 9 requires infinitely many recursive calls if the language $L(A_0)$ is infinite. Hence our task is to find properties of the automatic presentation P that allow to effectively reduce the number of recursive calls in line 9. To describe such a condition, we need the following model theoretic definitions.

The *Gaifman-graph* $G(\mathcal{A})$ of the τ-structure $\mathcal{A}$ is the following symmetric graph:

$$G(\mathcal{A}) = (A, \{(a,b) \in A \times A \mid \bigvee_{r\in\tau} \exists (a_1, \ldots, a_{m_r}) \in r\ \exists j,k : a_j = a, a_k = b\})\ .$$

Thus, the set of nodes is the universe of $\mathcal{A}$ and there is an edge between two elements if and only if they are contained in some tuple belonging to one of the relations of $\mathcal{A}$. The structure $\mathcal{A}$ has *bounded degree* if the Gaifman graph $G(\mathcal{A})$ has bounded degree, i.e., there exists a constant δ such that every $a \in A$ is adjacent to at most δ many other nodes in $G(\mathcal{A})$. For convenience, we say that the automatic presentation P has bounded degree if the structure $\mathcal{A}(P)$ has bounded degree.

The Gaifman-graph is also the basis for the definition of spheres: For $a, b \in \mathcal{A}$, let $d(a,b)$ be the distance between a and b in the Gaifman-graph $G(\mathcal{A})$. For $a \in \mathcal{A}$ and $r \in \mathbb{N}$, let $S(r,a)$ denote the set of elements b of $\mathcal{A}$ with $d(a,b) \leq r$. Then, for $\bar{a} = (a_1, \ldots, a_k) \in \mathcal{A}^k$ and $d \geq 0, k > 0$, we denote with $\mathcal{A}[d+k, \bar{a}]$ the induced substructure $\mathcal{A}\restriction \bigcup_{i=1}^{k} S(2^{d+k-i}, a_i)$. Given these notions, the following locality principle can be formulated.

Theorem 3.7 ([14]). *Let $\mathcal{A}$ be a structure, $\bar{a}, \bar{b} \in \mathcal{A}^k$ and $d \geq 0$ such that $(\mathcal{A}[d+k, \bar{a}],\ \bar{a}) \cong (\mathcal{A}[d+k, \bar{b}],\ \bar{b})$. Then, for every formula $\varphi(x_1, \ldots, x_k) \in$ $\mathrm{FO}[\exists^\infty, \exists^{\bmod}]$ of quantifier depth at most d, we have: $\mathcal{A} \models \varphi(\bar{a}) \iff \mathcal{A} \models \varphi(\bar{b})$.*

A short remark on the history of this result: for first-order logic, it was first shown by Gaifman [12] where, in the definition of $\mathcal{A}[d+k, \bar{a}]$, he used the radius 7^d instead of 2^{d+k-i}. His result has been improved in two directions: the radius 7^d was reduced and the logic has been extended. The final result of this line of research was provided by Keisler and Lotfallah in [14] proving the above theorem in an even stronger version: instead of $\mathcal{A}[d+k, \bar{a}]$, it suffices to consider the restriction of $\mathcal{A}$ to $\bigcup_{i=1}^{k} S(2^d, a_i)$ and, secondly, the result holds for even stronger extensions of FO.

This theorem provides the first step in the restriction of the search space in line 9 of the naïve algorithm: If $(\mathcal{A}[d+k, (\bar{u}, v)], (\bar{u}, v)) \cong (\mathcal{A}[d+k, (\bar{u}, v')], (\bar{u}, v'))$, then it suffices to consider the word v. But still, there could be infinitely many isomorphism types of the form $\mathcal{A}[d+k, (\bar{u}, v)]$ – which can be excluded by requiring the structure $\mathcal{A}$ to be of bounded degree. Then pumping arguments show that it suffices to consider words of triply exponential length in line 9 of the naïve algorithm:

Theorem 3.8 ([20]). *Let P be an automatic presentation of bounded degree. Then the model checking problem $\mathrm{MC}(\mathrm{FO}[\exists^\infty, \exists^{\bmod}], \{P\})$ is in 3-EXPSPACE.*

A further improvement of the naïve algorithm is obtained if, instead of the argument $\bar{u}$, one passes the isomorphism type of the finite structure $(\mathcal{A}[d + k, \bar{u}], \bar{u})$. This approach yields the following result.

Theorem 3.9 ([21])

(1) The model checking problem $\mathrm{MC}(\mathrm{FO}, \mathsf{iSAb})$ belongs to 2-EXPSPACE where $\mathsf{iSAb} = \mathsf{iSA} \cap \mathsf{SAb}$ is the set of injective automatic presentations of bounded degree.
(2) The model checking problem $\mathrm{MC}(\mathrm{FO}, \mathsf{SAb})$ belongs to 3-EXPSPACE.
(3) The model checking problem $\mathrm{MC}(\mathrm{FO}, \{P\})$ belongs to 2-EXPSPACE for every automatic presentation P of bounded degree.

Note that (1) and (2) give upper bounds for the combined complexities while (3) bounds the expression complexity of FO-model checking. This latter upper bound is shown to be tight in [21] since we constructed an injective automatic presentation of bounded degree P such that $\mathrm{MC}(\mathrm{FO}, \{P\})$ is hard for 2-EXPSPACE. Hence also the upper bound in (1) is tight. So far, nothing is known about the data complexity, i.e., the complexity of the problems $\mathrm{MC}(\{\varphi\}, \mathsf{iSAb})$ and $\mathrm{MC}(\{\varphi\}, \mathsf{SAb})$ for $\varphi \in \mathrm{FO}$. Furthermore, it is not known whether Theorem 3.8 is optimal nor what the combined or data complexity of $\mathrm{FO}[\exists^\infty, \exists^{\bmod}]$ is.

References

1. Bárány, V.: Invariants of automatic presentations and semi-synchronous transductions. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 289–300. Springer, Heidelberg (2006)
2. Bárány, V., Kaiser, Ł., Rubin, S.: Cardinality and counting quantifiers on omega-automatic structures. In: STACS 2008, pp. 385–396. IFIB Schloss Dagstuhl (2008)
3. Blumensath, A.: Automatic structures. Technical report, RWTH Aachen (1999)
4. Blumensath, A., Grädel, E.: Automatic Structures. In: LICS 2000, pp. 51–62. IEEE Computer Society Press, Los Alamitos (2000)
5. Campbell, C.M., Robertson, E.F., Ruškuc, N., Thomas, R.M.: Automatic semigroups. Theoretical Computer Science 250, 365–391 (2001)
6. Compton, K.J., Henson, C.W.: A uniform method for proving lower bounds on the computational complexity of logical theories. Annals of Pure and Applied Logic 48, 1–79 (1990)
7. Corran, R., Hoffmann, M., Kuske, D., Thomas, R.M.: Singular Artin monoids of finite type are automatic (submitted)
8. Delhommé, Ch., Goranko, V., Knapik, T.: Automatic linear orderings (Manuscript 2003)
9. Elgot, C.C.: Decision problems of finite automata design and related arithmetics. Trans. Am. Math. Soc. 98, 21–51 (1961)
10. Epstein, D.B.A., Cannon, J.W., Holt, D.F., Levy, S.V.F., Paterson, M.S., Thurston, W.P.: Word Processin. Groups. Jones and Bartlett Publishers, Boston (1992)
11. Fohry, E., Kuske, D.: On graph products of automatic and biautomatic monoids. Semigroup forum 72, 337–352 (2006)
12. Gaifman, H.: On local and nonlocal properties. In: Stern, J. (ed.) Logic Colloquium 1981, pp. 105–135. North-Holland, Amsterdam (1982)
13. Hodgson, B.R.: On direct products of automaton decidable theories. Theoretical Computer Science 19, 331–335 (1982)
14. Keisler, H.J., Lotfallah, W.B.: A local normal form theorem for infinitary logic with unary quantifiers. Mathematical Logic Quarterly 51(2), 137–144 (2005)
15. Khoussainov, B., Nerode, A.: Automatic presentations of structures. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 367–392. Springer, Heidelberg (1995)
16. Khoussainov, B., Nies, A., Rubin, S., Stephan, F.: Automatic structures: richness and limitations. Log. Methods in Comput. Sci. 3(2) (2007)
17. Khoussainov, B., Rubin, S., Stephan, F.: Definability and regularity in automatic structures. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 440–451. Springer, Heidelberg (2004)
18. Khoussainov, B., Rubin, S., Stephan, F.: Automatic linear orders and trees. ACM Transactions on Computational Logic 6(4), 675–700 (2005)

19. Kuske, D.: Is Cantor's theorem automatic? In: Y. Vardi, M., Voronkov, A. (eds.) LPAR 2003. LNCS, vol. 2850, pp. 332–345. Springer, Heidelberg (2003)
20. Kuske, D., Lohrey, M.: First-order and counting theories of ω-automatic structures. Journal of Symbolic Logic 73, 129–150 (2008)
21. Kuske, D., Lohrey, M.: Automatic structures of bounded degree revisited. In: CSL 1999. LNCS. Springer, Heidelberg (to appear, 2009)
22. Kuske, D., Lohrey, M.: Some natural problems in automatic graphs. Journal of Symbolic Logic (accepted, 2009)
23. Lohrey, M.: Automatic structures of bounded degree. In: Y. Vardi, M., Voronkov, A. (eds.) LPAR 2003. LNCS, vol. 2850, pp. 344–358. Springer, Heidelberg (2003)
24. Oliver, G.P., Thomas, R.M.: Automatic presentations for finitely generated groups. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 693–704. Springer, Heidelberg (2005)
25. Rubin, S.: Automata presenting structures: A survey of the finite string case. Bulletin of Symbolic Logic 14, 169–209 (2008)
26. Sakarovitch, J.: Easy multiplications. I. The realm of Kleene's Theorem. Information and Computation 74, 173–197 (1987)

The Graph Programming Language GP

Detlef Plump

Department of Computer Science
The University of York, UK

Abstract. GP (for Graph Programs) is a rule-based, nondeterministic programming language for solving graph problems at a high level of abstraction, freeing programmers from handling low-level data structures. The core of GP consists of four constructs: single-step application of a set of conditional graph-transformation rules, sequential composition, branching and iteration. This paper gives an overview on the GP project. We introduce the language by discussing a sequence of small programming case studies, formally explain conditional rule schemata which are the building blocks of programs, and present a semantics for GP in the style of structural operational semantics. A special feature of the semantics is how it uses the notion of *finitely failing* programs to define powerful branching and iteration commands. We also describe GP's prototype implementation.

1 Introduction

This paper gives an overview on GP, an experimental nondeterministic programming language for high-level problem solving in the domain of graphs. The language is based on conditional rule schemata for graph transformation (introduced in [19]) and thereby frees programmers from handling low-level data structures for graphs. The prototype implementation of GP compiles graph programs into bytecode for an abstract machine, and comes with a graphical editor for programs and graphs.

GP has a simple syntax as its core contains only four commands: single-step application of a set of rule schemata, sequential composition, branching and as-long-as-possible iteration. Despite its simplicity, GP is computationally complete in that every computable function on graphs can be programmed [9]. A major goal for the development of GP is to obtain a practical graph-transformation language that comes with a concise formal semantics, to facilitate program verification and other formal reasoning on programs.

There exist a number of graph-transformation languages and tools, such as PROGRES [24], AGG [6], Fujaba [15], GROOVE [21] and GrGen [8]. But to the best of our knowledge, PROGRES has been the only graph-transformation language with a complete formal semantics so far. The semantics given by Schürr in his dissertation [23], however, reflects the complexity of PROGRES and is in our opinion too complicated to be used for formal reasoning.

For GP, we adopt Plotkin's method of structural operational semantics [18] to define the meaning of programs. This approach is well established for imperative programming languages [16] but is novel in the field of graph transformation. In brief, the method consists in devising inference rules which inductively define the effect of commands on program states. Whereas a classic state consists of the values of all program variables at a certain point in time, the analogue for graph transformation is the graph on which the rules of a program operate.

As GP is nondeterministic, our semantics assigns to a program P and an input graph G *all* graphs that can result from executing P on G. A special feature of the semantics is the use of failing computations to define powerful branching and iteration constructs. (Failure occurs when a set of rule schemata to be executed is not applicable to the current graph.) While the conditions of branching commands in traditional programming languages are boolean expressions, GP uses arbitrary programs as conditions. The evaluation of a condition C succeeds if there *exists* an execution of C on the current graph that produces a graph. On the other hand, the evaluation of C is unsuccessful if all executions of C on the current graph result in failure. In this case C *finitely fails* on the current graph.

In logic programming, finite failure (of SLD resolution) is used to define negation [3]. In the case of GP, it allows to "hide" destructive executions of the condition C of a statement `if` C `then` P `else` Q. This is because after evaluating C, the resulting graph is discarded and either P or Q is executed on the graph with which the branching statement was entered. Finite failure also allows to elegantly lift the application of as-long-as-possible iteration from sets of rule schemata (as in [19]) to arbitrary programs: the body of a loop can no longer be applied if it finitely fails on the current graph.

The prototype implementation of GP is faithful to the semantics in that it uses backtracking to compute results for input graphs. Hence, for terminating programs a result will be found whenever one exists. In contrast, most other graph-transformation languages (except PROGRES) lack this completeness because their implementations have no backtracking mechanism. The GP system even provides users with the option to generate all possible results of a terminating program.

The rest of this paper is structured as follows. The next section is a brief summary of the graph-transformation formalism underlying GP, the so-called double-pushout approach with relabelling. Section 3 introduces conditional rule schemata and explains their use by interpreting them as sets of conditional rules. In Section 4, graph programs are gently introduced by discussing seven small case studies of problem solving with GP. The section also defines an abstract syntax for graph programs. Section 5 presents a formal semantics for GP in the style of structural operational semantics and discusses some consequences of the semantics. A brief description of the current implementation of GP is given in Section 6. In Section 7, we conclude and mention some topics for future work. Finally, the Appendix defines the natural pushouts on which the double-pushout approach with relabelling is based.

This overview paper is in parts based on the papers [19,20,13].

2 Graph Transformation

We briefly review the model of graph transformation underlying GP, the double-pushout approach with relabelling [10]. Our presentation is tailored to GP in that we consider graphs over a fixed label alphabet and rules in which only the interface graph may contain unlabelled nodes.

GP programs operate on graphs labelled with sequences of integers and strings (the reason for using sequences will be explained in Section 4). Let $\mathbb{Z}$ be the set of integers and Char be a finite set which represents the characters that can be typed on a keyboard. We fix the label alphabet $\mathcal{L} = (\mathbb{Z} \cup \mathrm{Char}^*)^+$ of all nonempty sequences over integers and character strings.

A *partially labelled graph* (or *graph* for short) is a system $G = (V_G, E_G, s_G, t_G, l_G, m_G)$, where V_G and E_G are finite sets of *nodes* (or *vertices*) and *edges*, $s_G, t_G \colon E_G \to V_G$ are the *source* and *target* functions for edges, $l_G \colon V_G \to \mathcal{L}$ is the partial node labelling function and $m_G \colon E_G \to \mathcal{L}$ is the (total) edge labelling function. Given a node v, we write $l_G(v) = \perp$ if $l_G(v)$ is undefined. Graph G is *totally labelled* if l_G is a total function. The set of all totally labelled graphs is denoted by $\mathcal{G}$.

A *graph morphism* $g \colon G \to H$ between graphs G and H consists of two functions $g_V \colon V_G \to V_H$ and $g_E \colon E_G \to E_H$ that preserve sources, targets and labels. More precisely, we have $s_H \circ g_E = g_V \circ s_G$, $t_H \circ g_E = g_V \circ t_G$, $m_H \circ g_E = m_G$, and $l_H(g(v)) = l_G(v)$ for all v such that $l_G(v) \neq \perp$. Morphism g is an *inclusion* if $g(x) = x$ for all nodes and edges x. It is *injective* (*surjective*) if g_V and g_E are injective (surjective), and an *isomorphism* if it is injective, surjective and satisfies $l_H(g_V(v)) = \perp$ for all nodes v with $l_V(v) = \perp$. In this case G and H are *isomorphic*, denoted by $G \cong H$. The *composition* $h \circ g \colon G \to M$ of two morphisms $g \colon G \to H$ and $h \colon H \to M$ is defined componentwise: $h \circ g = \langle h_V \circ g_V,\, h_E \circ g_E \rangle$.

A *rule* $r = (L \leftarrow K \to R)$ consists of two inclusions $K \to L$ and $K \to R$ where L and R are totally labelled graphs. We call L the *left-hand side*, R the *right-hand side* and K the *interface* of r. Intuitively, an application of r to a graph will remove the items in $L - K$, preserve K, add the items in $R - K$, and relabel the nodes that are unlabelled in K.

Given graphs G, H in $\mathcal{G}$, a rule $r = (L \leftarrow K \to R)$, and an injective graph morphism $g \colon L \to G$, a *direct derivation* from G to H by r and g consists of two natural pushouts as in Figure 1. (See the appendix for the definition of natural pushouts.) We write $G \Rightarrow_{r,g} H$ or just $G \Rightarrow_r H$ if there exists such a direct derivation, and $G \Rightarrow_{\mathcal{R}} H$, where $\mathcal{R}$ is a set of rules, if there is some $r \in \mathcal{R}$ such that $G \Rightarrow_r H$.

$$\begin{tikzcd}
L \arrow[d, "g"'] & K \arrow[l] \arrow[r] \arrow[d] & R \arrow[d] \\
G & D \arrow[l] \arrow[r] & H
\end{tikzcd}$$

Fig. 1. A double-pushout

In [10] it is shown that, given r, G and g as above, there exists a direct derivation as in Figure 1 if and only if g satisfies the *dangling condition*: no node in $g(L) - g(K)$ is incident to an edge in $G - g(L)$. In this case D and H are determined uniquely up to isomorphism and can be constructed from G as follows:

1. Remove all nodes and edges in $g(L)-g(K)$. For each $v \in V_K$ with $l_K(v) = \perp$, define $l_D(g_V(v)) = \perp$. The resulting graph is D.
2. Add disjointly to D all nodes and edges from $R - K$, while keeping their labels. For $e \in E_R - E_K$, $s_H(e)$ is $s_R(e)$ if $s_R(e) \in V_R - V_K$, otherwise $g_V(s_R(e))$. Targets are defined analogously.
3. For each $v \in V_K$ with $l_K(v) = \perp$, define $l_H(g_V(v)) = l_R(v)$. The resulting graph is H.

To define conditional rules, we follow [5] and equip rules with predicates that constrain the morphisms by which rules can be applied. A *conditional rule* $q = (r, P)$ consists of a rule r and a predicate P on graph morphisms. We require that P is invariant under isomorphic codomains: for a morphism $g\colon L \to G$ and an isomorphism $i\colon G \to G'$, we have $P(g)$ if and only if $P(i \circ g)$. Given a direct derivation $G \Rightarrow_{r,g} H$ such that $P(g)$, we write $G \Rightarrow_{q,g} H$ or just $G \Rightarrow_q H$. For a set of conditional rules $\mathcal{R}$, we write $G \Rightarrow_{\mathcal{R}} H$ if there is some $q \in \mathcal{R}$ such that $G \Rightarrow_q H$.

3 Conditional Rule Schemata

Conditional rule schemata are the "building blocks" of GP, as programs are essentially declarations of such schemata together with a command sequence for controlling their application. Rule schemata generalise rules in that labels may be expressions with parameters of type integer or string. In this section, we give an abstract syntax for the textual components of conditional rule schemata and interpret them as sets of conditional rules.

Figure 2 shows an example for the declaration of a conditional rule schema. It consists of the identifier `bridge` followed by the declaration of formal parameters, the left and right graphs of the schema which are labelled with expressions over the parameters, the node identifiers `1`, `2`, `3` determining the interface of the schema, and the keyword `where` followed by the condition.

In the GP programming system [13], conditional rule schemata are constructed with a graphical editor. We give grammars in Extended Backus-Naur Form for the textual components of such schemata. Figure 3 shows the grammar for node and edge labels in the left and right graph of a rule schema (categories LeftLabel and RightLabel), Figure 4 shows the grammar for conditions (category BoolExp).[1]

Labels can be sequences of expressions separated by underscores, as is demonstrated by examples in Section 4. We require that labels in the left graph must

[1] The grammars are ambiguous, we use parentheses to disambiguate expressions where necessary.

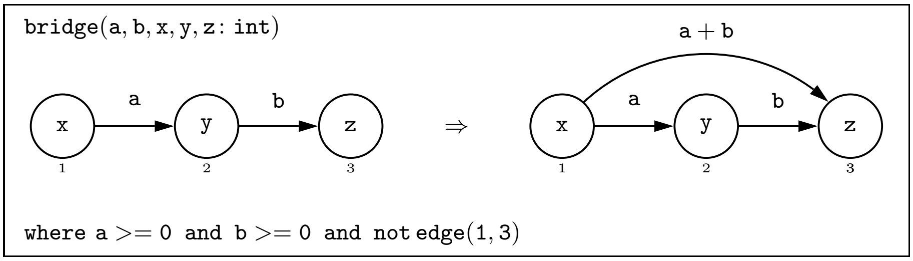

bridge(a, b, x, y, z : int)
a
b
x
y
z
1
2
3
⇒
a + b
where a >= 0 and b >= 0 and not edge(1, 3)

Fig. 2. A conditional rule schema

be simple expressions because their values at execution time are determined by graph matching. All variable identifiers in the right graph must also occur in the left graph. Every expression in category Exp has type `int` or `string`, where the type of a variable identifier is determined by its declaration and arithmetical operators expect arguments of type `int`.

```
LeftLabel  ::= SimpleExp ['_' LeftLabel]
RightLabel ::= Exp ['_' RightLabel]
SimpleExp  ::= ['-'] Num | String | VarId
Exp        ::= SimpleExp | Exp ArithOp Exp
ArithOp    ::= '+' | '-' | '*' | '/'
Num        ::= Digit {Digit}
String     ::= '"' {Char} '"'
```

Fig. 3. Syntax of node and edge labels

```
BoolExp ::= edge '(' Node ',' Node ')' | Exp RelOp Exp
            | not BoolExp | BoolExp BoolOp BoolExp
Node    ::= Digit {Digit}
RelOp   ::= '=' | '\=' | '>' | '<' | '>=' | '<='
BoolOp  ::= and | or
```

Fig. 4. Syntax of conditions

The condition of a rule schema is a Boolean expression built from expressions of category Exp and the special predicate `edge`, where relational operators have arguments of type `int`. Again, all variable identifiers occurring in the condition must also occur in the left graph of the schema. The predicate `edge` demands the (non-)existence of an edge between two nodes in the graph to which the rule schema is applied. For example, the expression `not edge(1, 3)` in the condition of Figure 2 forbids an edge from node 1 to node 3 when the left graph is matched.

We interpret a conditional rule schema as the (possibly infinite) set of conditional rules that is obtained by instantiating variables with any values and evaluating expressions. To define this, consider a declaration D of a conditional rule-schema. Let L and R be the left and right graphs of D, and c the condition. We write $\mathrm{Var}(D)$ for the set of variable identifiers occurring in D. Given x in $\mathrm{Var}(D)$, $\mathrm{type}(x)$ denotes the type associated with x. An *assignment* is a mapping $\alpha\colon \mathrm{Var}(D) \to (\mathbb{Z} \cup \mathrm{Char}^*)$ such that for each x in $\mathrm{Var}(D)$, $\mathrm{type}(x) = \texttt{int}$ implies $\alpha(x) \in \mathbb{Z}$, and $\mathrm{type}(x) = \texttt{string}$ implies $\alpha(x) \in \mathrm{Char}^*$.

Given a label l of category RightLabel occuring in D and an assignment α, the value $l^\alpha \in \mathcal{L}$ is inductively defined. If l is a numeral or a sequence of characters, then l^α is the integer or character string represented by l (which is independent of α). If l is a variable identifier, then $l^\alpha = \alpha(l)$. Otherwise, l^α is obtained from the values of l's components. If l has the form $e_1 \oplus e_2$ with $\oplus$ in ArithOp and e_1, e_2 in Exp, then $l^\alpha = e_1^\alpha \oplus_\mathbb{Z} e_2^\alpha$ where $\oplus_\mathbb{Z}$ is the integer operation represented by $\oplus$.[2] If l has the form e_m with e in Exp and m in RightLabel, then $l^\alpha = e^\alpha m^\alpha$. Note that our definition of l^α covers all labels in D since LeftLabel is a subcategory of RightLabel.

The value of the condition c in D not only depends on an assignment but also on a graph morphism. For, if c contains the predicate `edge`, then we need to consider the structure of the graph to which we want to apply the rule schema. Consider an assignment α and let L^α be obtained from L by replacing each label l with l^α. Let $g\colon L^\alpha \to G$ be a graph morphism with $G \in \mathcal{G}$. Then for each Boolean subexpression b of c, the value $b^{\alpha,g}$ in $\mathbb{B} = \{\texttt{tt}, \texttt{ff}\}$ is inductively defined. If b has the form $e_1 \bowtie e_2$ with $\bowtie$ in RelOp and e_1, e_2 in Exp, then $b^{\alpha,g} = \texttt{tt}$ if and only if $e_1^\alpha \bowtie_\mathbb{Z} e_2^\alpha$ where $\bowtie_\mathbb{Z}$ is the relation on integers represented by $\bowtie$. If b has the form $\texttt{not}\, b_1$ with b_1 in BoolExp, then $b^{\alpha,g} = \texttt{tt}$ if and only if $b_1^{\alpha,g} = \texttt{ff}$. If b has the form $b_1 \oplus b_2$ with $\oplus$ in BoolOp and b_1, b_2 in BoolExp, then $b^{\alpha,g} = b_1^{\alpha,g} \oplus_\mathbb{B} b_2^{\alpha,g}$ where $\oplus_\mathbb{B}$ is the Boolean operation on $\mathbb{B}$ represented by $\oplus$. A special case is given if b has the form $\texttt{edge}(v, w)$ where v, w are identifiers of interface nodes in D. We then have

$$b^{\alpha,g} = \begin{cases} \texttt{tt} \text{ if there is an edge from } g(v) \text{ to } g(w), \\ \texttt{ff} \text{ otherwise.} \end{cases}$$

Let now r be the rule-schema identifier associated with declaration D. For every assignment α, let $r^\alpha = (L^\alpha \leftarrow K \to R^\alpha,\ P^\alpha)$ be the conditional rule given as follows:

– L^α and R^α are obtained from L and R by replacing each label l with l^α.
– K is the discrete subgraph of L and R determined by the node identifiers for the interface, where all nodes are unlabelled.
– P^α is defined by: $P^\alpha(g)$ if and only if g is a graph morphism $L^\alpha \to G$ such that $G \in \mathcal{G}$ and $c^{\alpha,g} = \texttt{tt}$.

Now the *interpretation* of r is the rule set $\mathrm{I}(r) = \{r^\alpha \mid \alpha \text{ is an assignment}\}$. For notational convenience, we sometimes denote the relation $\Rightarrow_{\mathrm{I}(r)}$ by $\Rightarrow_r$.

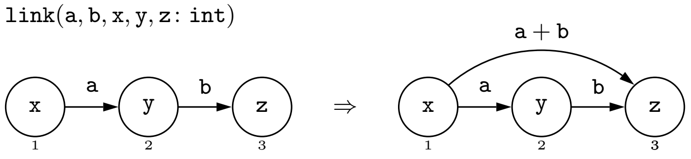
[2] For simplicity, we consider division by zero as an implementation-level issue.

4 Graph Programs

We discuss a number of example programs to familiarize the reader with the features of GP and their use in solving graph problems. At the end of the section, we define the abstract syntax of GP programs.

Example 1 (Transitive closure)
A *transitive closure* of a graph is obtained by inserting an edge between all distinct nodes v and w such that there is a directed path from v to w but no edge. The program `trans_closure` in Figure 5 generates a transitive closure of an integer-labelled input graph by applying the rule schema `link` as long as possible, using the iteration operator '`!`'. In general, arbitrary command sequences can be iterated.

```
main = link!

link(a, b, x, y, z: int)
```



```
where not edge(1, 3)
```

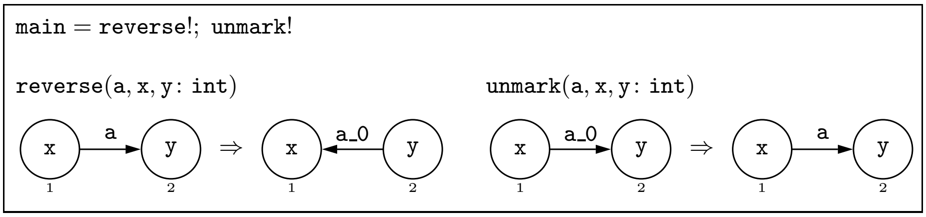
Fig. 5. The program `trans_closure`

The keyword `main` starts the main command sequence of a program to distinguish it from macros (see Example 5). Note that the condition `not edge(1, 3)` of `link` prevents the creation of edges between nodes that are already linked. Without this condition, `trans_closure` could generate parallel edges between nodes 1 and 3 ad infinitum.

By our definition of transitive closure, we can choose any label for the edge created by `link`. Using `a + b` implies that `trans_closure` may produce different results for a given input graph if there are different paths between two nodes. If we want to generate a unique transitive closure, we can replace `a + b` with a constant such as `0`.

Example 2 (Inverse)
The *inverse* of a graph is obtained by reversing the directions of all edges. The program `inverse` in Figure 6 computes the inverse of an integer-labelled input graph in two stages, using the sequential composition of the loops `reverse!` and `unmark!`. The first loop reverses each edge and replaces its label x with the *tagged* label x_`0`, then the second loop removes all tags. In general, arbitrary subprograms can be joined by the semicolon operator.

The underscore operator allows to add a *tag* to a label, used here to mark an edge as having been reversed. In general, a tagged label is a sequence of



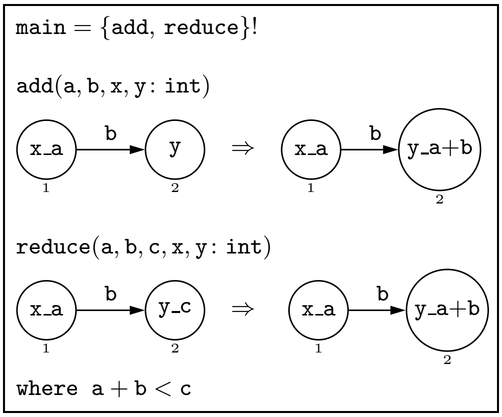
```
main = reverse!; unmark!

reverse(a, x, y: int)                    unmark(a, x, y: int)
```

Fig. 6. The program `inverse`

expressions joined by underscores. Here, we need to mark reversed edges as otherwise the loop `reverse!` would not terminate. Note that the rule schema `reverse` can only be applied to edges with untagged labels.

Example 3 (Shortest distances). Given a graph G whose edge labels are integers, the *distance* of a directed path from a node v to a node w is the sum of the edge labels on that path. If all edge labels in G are nonnegative, then the *shortest distance* from v to w is the minimum of the distances of all paths from v to w.

The program `distances` in Figure 7 expects an integer-labelled input graph where exactly one node v has a tagged label of the form x`_0` and where all edge labels are nonnegative. It adds to each node w that is distinct and reachable from v a tag with the shortest distance from v to w.

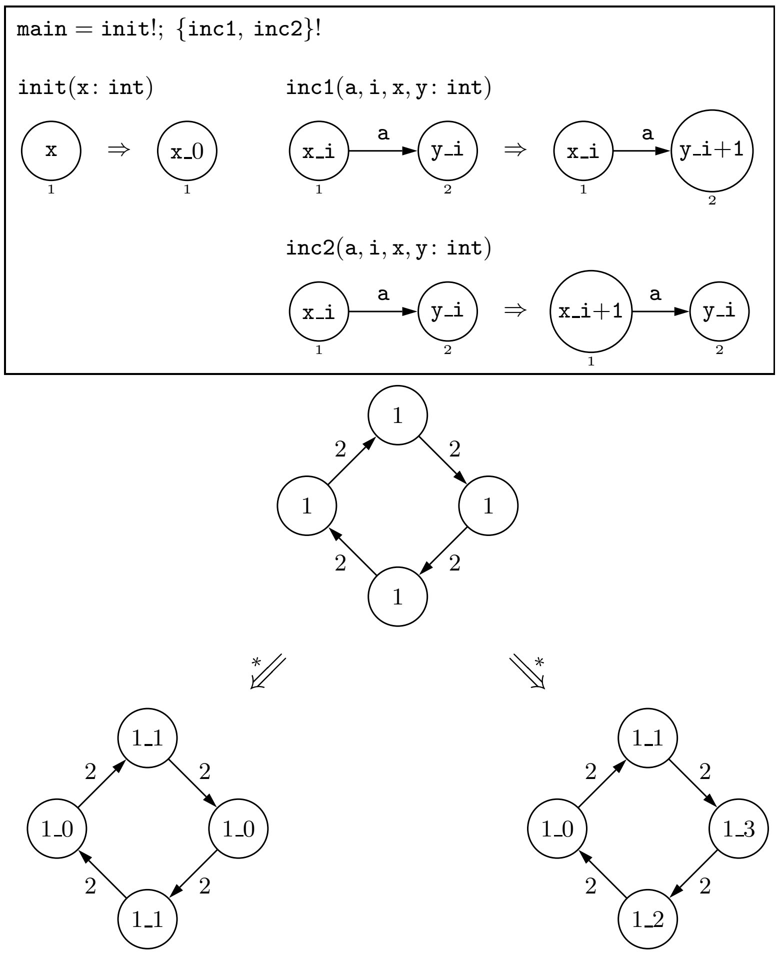


```
main = {add, reduce}!

add(a, b, x, y: int)

reduce(a, b, c, x, y: int)

where a + b < c
```

Fig. 7. The program `distances`

In each iteration of the program's loop, one of the rule schemata `add` and `reduce` is applied to the current graph. If both rule schemata are applicable, one of them is chosen nondeterministically. An equivalent, slightly more deterministic solution is to separate the phases of addition and reduction: `main = add!; reduce!`. A refined version of the program `distances` which implements Dijkstra's shortest-path algorithm can be found in [19].

Example 4 (Colouring). A *colouring* for a graph is an assignment of colours (integers) to nodes such that the source and target of each edge have different colours. The program `colouring` in Figure 8 produces a colouring for every integer-labelled input graph without loops, recording colours as tags. (Checking for loops would be a straightforward extension which we omit for simplicity.)



```
main = init!; {inc1, inc2}!
```

Fig. 8. The program `colouring` and two of its derivations

The program initially colours each node with zero and then repeatedly increments either the source or the target colour of an edge with the same colour at both ends. Note that this process is highly nondeterministic: Figure 8 shows two different colourings produced for the same input graph, where one is optimal in that it uses only two colours while the other uses four colours. (The problem to generate a colouring with a minimal number of colours is NP-complete [7] and requires a more involved program.)

It is easy to see that whenever `colouring` terminates, the resulting graph is a correctly coloured version of the input graph. For, the output cannot contain an edge with the same colour at both nodes as then `inc1` or `inc2` would have been applied at least one more time. It is less obvious though that the program does terminate for every input graph.

To see that `colouring` always terminates, consider graphs whose node labels are of the form n_i, with $n, i \in \mathbb{Z}$. Given a node v, we denote the tag of its label by $\mathrm{tag}(v)$. Now observe that if G is a graph with $\mathrm{tag}(v) = 0$ for each node v, then for every derivation $G \Rightarrow^*_{\{\mathtt{inc1},\mathtt{inc2}\}} H$ there is some $0 \leq k < V_H$ such that $\mathrm{tag}(V_H) = \{0, 1, \ldots, k\}$ (where some tags may occur repeatedly in H). Thus, by assigning to every graph M the integer $\#M = \sum_{v \in V_M} \mathrm{tag}(v)$, we obtain

$$\#H < 1 + 2 + \cdots + |V_H| = 1 + 2 + \cdots + |V_G|.$$

Since $\#H$ equals the number of rule schema applications in $G \Rightarrow^* H$, it follows that every derivation with `inc1` and `inc2` starting from G must eventually terminate. Moreover, as the upper bound for $\#H$ is quadratic in $|V_G|$, `colouring` always performs at most a quadratic number of rule schema applications.

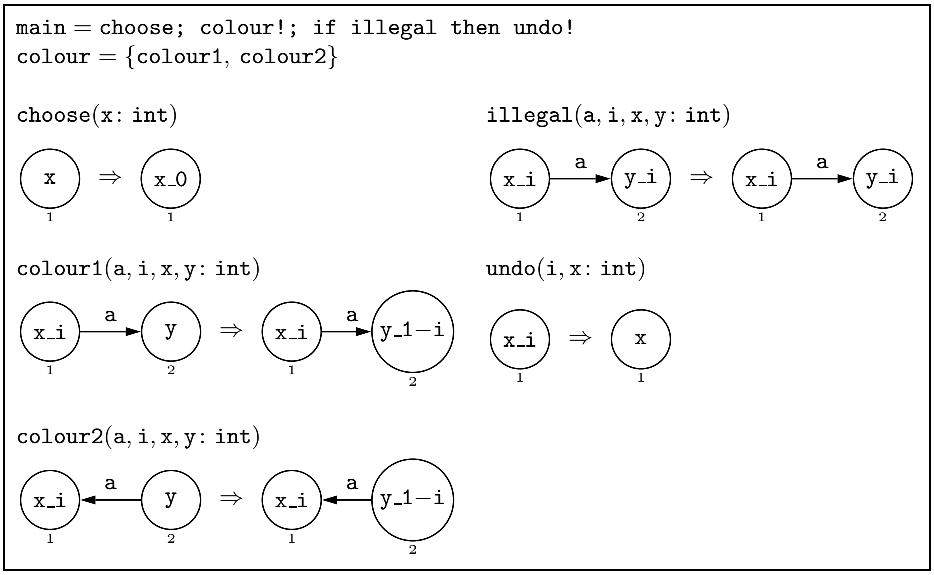
Example 5 (2-Colouring). A graph is *2-colourable* (or *bipartite*) if it possesses a colouring with at most two colours. The program `2-colouring` in Figure 9 generates a 2-colouring for a nonempty and connected input graph if such a colouring exists—otherwise the input graph is returned. The program uses the *macro* `colour` to represent the rule-schema set {`colour1`, `colour2`}.

Given an integer-labelled input graph, first the rule schema `choose` colours an arbitrary node by replacing its label x with x_0. Then the loop `colour!` applies the rule schemata `colour1` and `colour2` as long as possible to colour all remaining nodes. In each iteration of the loop, an uncoloured node adjacent to an already coloured node v gets the colour in $\{0, 1\}$ that is complementary to v's colour. If the input graph is connected, the graph resulting from `colour!` is correctly coloured if and only if the rule schema `illegal` is not applicable. The latter is checked by the if-statement. If `illegal` is applicable, then the input must contain an undirected cycle of odd length and hence is not 2-colourable (see for example [12]). In this case the loop `undo!` removes all tags to return the input graph unmodified. Note that the number of rule-schema applications performed by `2-colouring` is linear in the number of input nodes.

We can extend `2-colouring`'s applicability to graphs that are possibly empty or disconnected by inserting a nested loop:

```
main = (choose; colour!)!; if illegal then undo!.
```

Now if the input graph is empty, `choose` fails which causes the outer loop to terminate and return the current (empty) graph. On the other hand, if the input consists of several connected components, the body of the outer loop is repeatedly called to colour each component.



```
main = choose; colour!; if illegal then undo!
colour = {colour1, colour2}
```

Fig. 9. The program `2-colouring`

Example 6 (Series-parallel graphs)
The class of *series-parallel* graphs is inductively defined as follows. Every graph G consisting of two nodes connected by an edge is series-parallel, where the edge's source and target are called *source* and *target* of G. Given series-parallel graphs G and H, the graphs obtained from the disjoint union $G + H$ by the following two operations are also series-parallel. *Serial composition:* merge the target of G with the source of H; the source of G becomes the new source and the target of H becomes the new target. *Parallel composition:* merge the source of G with the source of H, and the target of G with the target of H; sources and targets are preserved.

It is known [2,4] that a graph is series-parallel if and only if it reduces to a graph consisting of two nodes connected by an edge by repeated application of the following operations: (a) Given a node with one incoming edge i and one outgoing edge o such that $s(i) \neq t(o)$, replace i, o and the node by an edge from $s(i)$ to $t(o)$. (b) Replace a pair of parallel edges by an edge from their source to their target.

Suppose that we want to check whether a connected, integer-labelled graph G is series-parallel and, depending on the result, execute either a program P or a program Q on G. We can do this with the program

```
main = if reduce!; base then P else Q
reduce = {serial, parallel}
```

whose rule schemata `serial`, `parallel` and `base` are shown in Figure 10.

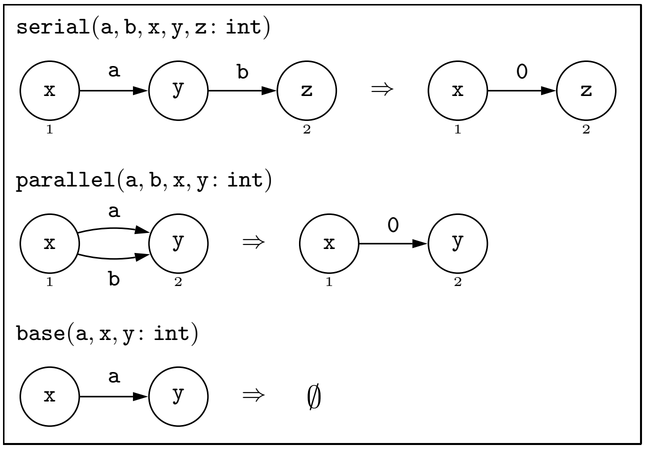


Fig. 10. Rule schemata for recognizing series-parallel graphs

The subprogram `reduce!` applies as long as possible the operations (a) and (b) to the input graph G, then the rule schema `base` checks if the resulting graph consists of two nodes connected by an edge. Graph G is series-parallel if and only if `base` is applicable to the reduced graph. (Note that `reduce!` preserves connectedness and that, by the dangling condition, `base` is applicable only if the images of its left-hand nodes have degree one.) If `base` is applicable, then program P is executed, otherwise program Q. It is important to note that P or Q is executed *on the input graph* G whereas the graph resulting from the test is discarded. The precise semantics of the branching command is given in the next section.

To make the above program usable for possibly disconnected graphs, we can add an if-statement which checks whether the application of `base` has resulted in a nonempty graph:

`main = if (reduce!; base; if nonempty then fail) then` P `else` Q.

Here `nonempty` is a rule schema whose left-hand side is a single interface node, labelled with an integer variable. If `nonempty` is applicable, then the graph resulting from `reduce!` is disconnected and hence the input graph is not series-parallel. In this case `fail` causes the test of the outer if-statement to fail, with the consequence that program Q is executed on the input graph.

Example 7 (Sierpinski triangles). A *Sierpinski triangle* is a self-similar geometric structure which can be recursively defined [17]. Figure 11 shows a Sierpinski triangle of generation three, composed of three second-generation triangles, each of which consists of three triangles of generation one. The triangle and its geometric layout have been generated with the GP programming system [26,13].

The program in Figure 12 expects as input a graph consisting of a single node labelled with the generation number of the Sierpinski triangle to be produced.

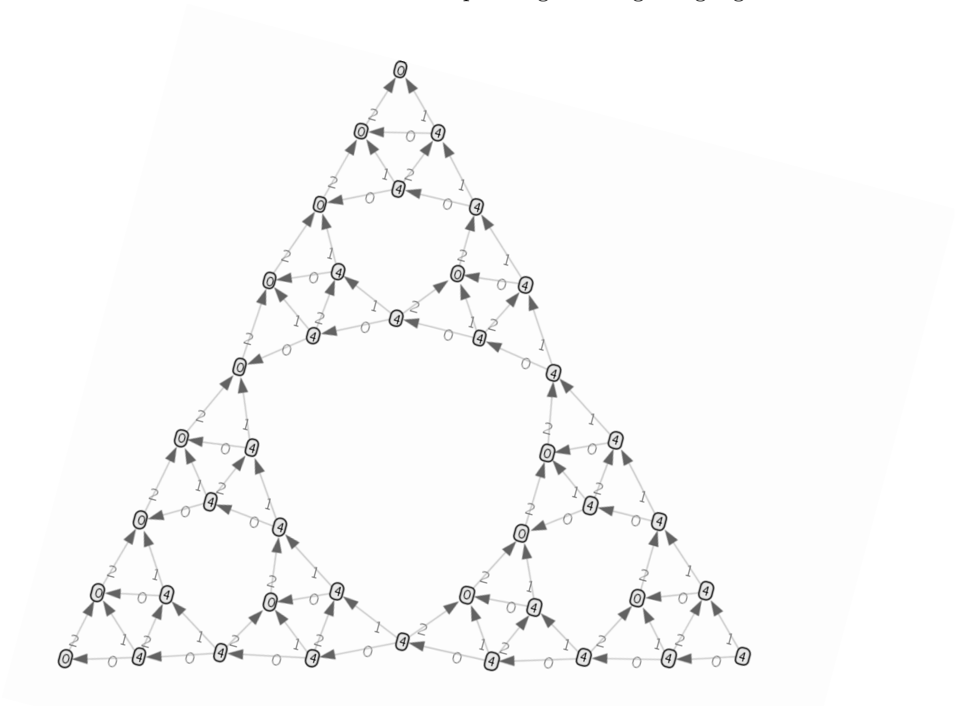


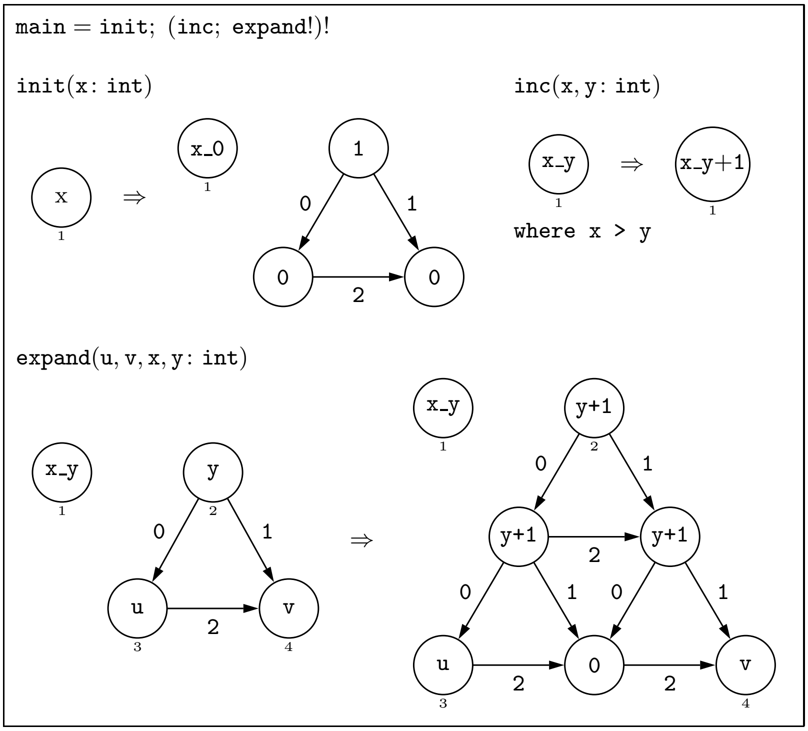
Fig. 11. A Sierpinski triangle (third generation)

The rule schema `init` creates the Sierpinski triangle of generation 0 and turns the input node into a unique "control node" with the tagged label x_0 in order to hold the required generation number x together with the current generation number.

After initialisation, the nested loop (`inc`; `expand`!)! is executed. In each iteration of the outer loop, `inc` increases the current generation number if it is smaller than the required number. The latter is checked by the condition `where x > y`. If the test is successful, the inner loop `expand!` performs a Sierpinski step on each triangle whose top node is labelled with the current generation number: the triangle is replaced by four triangles such that the top nodes of the three outer triangles are labelled with the next higher generation number. The test `x > y` fails when the required generation number has been reached. In this case the application of `inc` fails, causing the outer loop to terminate and return the current graph which is the Sierpinski triangle of the requested generation.

Figure 13 shows the abstract syntax of GP programs.[3] A program consists of a number of declarations of conditional rule schemata and macros, and exactly one declaration of a main command sequence. The rule-schema identifiers (category RuleId) occurring in a call of category RuleSetCall refer to declarations of conditional rule schemata in category RuleDecl (see Section 3). Semantically,

[3] Where necessary we use parentheses to disambiguate programs.



```
main = init; (inc; expand!)!

init(x: int)

inc(x, y: int)
where x > y

expand(u, v, x, y: int)
```

Fig. 12. The program `sierpinski`

Prog	::= Decl {Decl}
Decl	::= RuleDecl | MacroDecl | MainDecl
MacroDecl	::= MacroId '=' ComSeq
MainDecl	::= `main` '=' ComSeq
ComSeq	::= Com {';' Com}
Com	::= RuleSetCall | MacroCall
	| `if` ComSeq `then` ComSeq [`else` ComSeq]
	| ComSeq '!'
	| `skip` | `fail`
RuleSetCall	::= RuleId | '{' [RuleId {',' RuleId}] '}'
MacroCall	::= MacroId

Fig. 13. Abstract syntax of GP

each rule-schema identifier r stands for the set $\mathrm{I}(r)$ of conditional rules induced by that identifier. A call of the form $\{r_1, \ldots, r_n\}$ stands for the union $\bigcup_{i=1}^{n} \mathrm{I}(r_i)$.

Macros are a simple means to structure programs and thereby to make them more readable. Every program can be transformed into an equivalent macro-free program by replacing macro calls with their associated command sequences (recursive macros are not allowed). In the next section we use the terms "program" and "command sequence" synonymously, assuming that all macro calls have been replaced.

The commands `skip` and `fail` can be expressed through the other commands (see next section), hence the core of GP includes only the call of a set of conditional rule schemata (RuleSetCall), sequential composition (';'), the if-then-else statement and as-long-as-possible iteration ('!').

5 Semantics of Graph Programs

This section presents a formal semantics of GP in the style of Plotkin's structural operational semantics [18]. As usual for this approach, inference rules inductively define a small-step transition relation $\rightarrow$ on *configurations*. In our setting, a configuration is either a command sequence together with a graph, just a graph or the special element fail:

$$\rightarrow \ \subseteq \ (\mathrm{ComSeq} \times \mathcal{G}) \times ((\mathrm{ComSeq} \times \mathcal{G}) \cup \mathcal{G} \cup \{\mathrm{fail}\}).$$

Configurations in ComSeq $\times\, \mathcal{G}$ represent unfinished computations, given by a rest program and a state in the form of a graph, while graphs in $\mathcal{G}$ are proper results of computations. In addition, the element fail represents a failure state. A configuration γ is *terminal* if there is no configuration δ such that $\gamma \rightarrow \delta$.

Each inference rule in Figure 14 consists of a premise and a conclusion separated by a horizontal bar. Both parts contain meta-variables for command sequences and graphs, where R stands for a call in category RuleSetCall, C, P, P', Q stand for command sequences in category ComSeq and G, H stand for graphs in $\mathcal{G}$. Given a rule-set call R, let $\mathrm{I}(R) = \bigcup\{\mathrm{I}(r) \mid r \text{ is a rule-schema identifier in } \mathcal{R}\}$ (see Section 3 for the definition of $\mathrm{I}(r)$). The *domain* of $\Rightarrow_{\mathrm{I}(R)}$, denoted by $\mathrm{Dom}(\Rightarrow_{\mathrm{I}(R)})$, is the set of all graphs G in $\mathcal{G}$ such that $G \Rightarrow_{\mathrm{I}(R)} H$ for some graph H. Meta-variables are considered to be universally quantified. For example, the rule $[\mathrm{Call}_1]$ should be read as: "For all R in RuleSetCall and all G, H in $\mathcal{G}$, $G \Rightarrow_{\mathrm{I}(R)} H$ implies $\langle R,\, G\rangle \rightarrow H$."

Figure 14 shows the inference rules for the core constructs of GP. We write $\rightarrow^+$ and $\rightarrow^*$ for the transitive and reflexive-transitive closures of $\rightarrow$. A command sequence C *finitely fails* on a graph $G \in \mathcal{G}$ if (1) there does not exist an infinite sequence $\langle C,\, G\rangle \rightarrow \langle C_1,\, G_1\rangle \rightarrow \ldots$ and (2) for each terminal configuration γ such that $\langle C,\, G\rangle \rightarrow^* \gamma$, $\gamma = \mathrm{fail}$. In other words, C finitely fails on G if all computations starting from $(C,\, G)$ eventually end in the configuration fail.

The concept of finite failure stems from logic programming where it is used to define *negation as failure* [3]. In the case of GP, we use it to define powerful branching and iteration constructs. In particular, our definition of the if-then-else command allows to "hide" destructive tests. This is demonstrated by Example 6 in the previous section, where the test of the if-then-else command reduces input graphs as much as possible by the rule schemata `serial` and `parallel`, followed

$$[\text{Call}_1]\ \frac{G \Rightarrow_{\mathrm{I}(R)} H}{\langle R,\, G\rangle \to H} \qquad [\text{Call}_2]\ \frac{G \notin \text{Dom}(\Rightarrow_{\mathrm{I}(R)})}{\langle R,\, G\rangle \to \text{fail}}$$

$$[\text{Seq}_1]\ \frac{\langle P,\, G\rangle \to \langle P',\, H\rangle}{\langle P; Q,\, G\rangle \to \langle P'; Q,\, H\rangle} \qquad [\text{Seq}_2]\ \frac{\langle P,\, G\rangle \to H}{\langle P; Q,\, G\rangle \to \langle Q,\, H\rangle}$$

$$[\text{Seq}_3]\ \frac{\langle P,\, G\rangle \to \text{fail}}{\langle P; Q,\, G\rangle \to \text{fail}}$$

$$[\text{If}_1]\ \frac{\langle C,\, G\rangle \to^+ H}{\langle \texttt{if}\ C\ \texttt{then}\ P\ \texttt{else}\ Q,\, G\rangle \to \langle P,\, G\rangle} \qquad [\text{If}_2]\ \frac{C \text{ finitely fails on } G}{\langle \texttt{if}\ C\ \texttt{then}\ P\ \texttt{else}\ Q,\, G\rangle \to \langle Q,\, G\rangle}$$

$$[\text{Alap}_1]\ \frac{\langle P,\, G\rangle \to^+ H}{\langle P!,\, G\rangle \to \langle P!,\, H\rangle} \qquad [\text{Alap}_2]\ \frac{P \text{ finitely fails on } G}{\langle P!,\, G\rangle \to G}$$

Fig. 14. Inference rules for core commands

by an application of `base`. By the inference rules [If$_1$] and [If$_2$], the resulting graph is discarded and program P or Q is executed *on the input graph.*

The meaning of the remaining GP commands is defined in terms of the meaning of the core commands, see Figure 15. We refer to these commands as *derived* commands.

$$\begin{array}{ll}
[\text{Skip}] & \langle \texttt{skip},\, G\rangle \to \langle \texttt{r}_\emptyset,\, G\rangle \\
 & \text{where } \texttt{r}_\emptyset \text{ is an identifier for the rule schema } \emptyset \Rightarrow \emptyset \\
[[\text{Fail}] & \langle \texttt{fail},\, G\rangle \to \langle \{\},\, G\rangle \\
[\text{If}_3] & \langle \texttt{if}\ C\ \texttt{then}\ P,\, G\rangle \to \langle \texttt{if}\ C\ \texttt{then}\ P\ \texttt{else skip},\, G\rangle
\end{array}$$

Fig. 15. Inference rules for derived commands

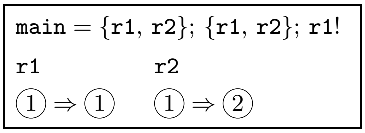
Figure 16 shows a simple example of program evaluation by the transition relation $\to$. It demonstrates that for the same input graph, a program may compute an output graph, reach the failure state or diverge.

We now summarise the meaning of GP programs by a semantic function $[\![\_]\!]$ which assigns to each program P the function $[\![P]\!]$ mapping an input graph G to the set of all possible results of running P on G. The result set may contain, besides proper results in the form of graphs, the special value $\perp$ which indicates a nonterminating or stuck computation. To this end, let the *semantic function* $[\![\_]\!]\colon \text{ComSeq} \to (\mathcal{G} \to 2^{\mathcal{G}\cup\{\perp\}})$ be defined by[4]

$$[\![P]\!]G \;=\; \{H \in \mathcal{G} \mid \langle P,\, G\rangle \xrightarrow{+} H\} \cup \{\perp \mid P \text{ can diverge or get stuck from } G\}$$

where P *can diverge from* G if there is an infinite sequence $\langle P,\, G\rangle \to \langle P_1,\, G_1\rangle \to \langle P_2,\, G_2\rangle \to \ldots$, and P *can get stuck from* G if there is a terminal configuration $\langle Q,\, H\rangle$ such that $\langle P,\, G\rangle \to^* \langle Q,\, H\rangle$.

[4] We write $[\![P]\!]G$ for the application of $[\![P]\!]$ to a graph G.



```
main = {r1, r2}; {r1, r2}; r1!
```

$$\begin{tikzcd}
\langle \texttt{main}, \textcircled{1} \rangle \arrow[r] \arrow[d] & \langle P, \textcircled{2} \rangle \arrow[r] & \mathrm{fail} & \\
\langle P, \textcircled{1} \rangle \arrow[r] \arrow[d] & \langle \texttt{r1!}, \textcircled{1} \rangle \arrow[r] & \langle \texttt{r1!}, \textcircled{1} \rangle \arrow[r] & \ldots \\
\langle \texttt{r1!}, \textcircled{2} \rangle \arrow[d] & & & \\
\textcircled{2} & & &
\end{tikzcd}$$

where $P = \{\texttt{r1}, \texttt{r2}\}; \texttt{r1}!$

Fig. 16. Nondeterminism in program evaluation

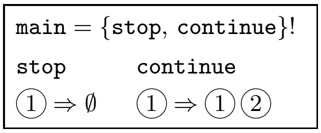
The element fail is not considered as a result of running a program and hence does not occur in result sets. In the current implementation of GP, reaching the failure state triggers backtracking which attempts to find a proper result (see next section). Note that a program P finitely fails on a graph G if and only if $[\![P]\!]G = \emptyset$. In Example 6, for instance, we have $[\![\texttt{reduce}!; \texttt{base}]\!]G = \emptyset$ for every connected graph G containing a cycle. This is because the graph resulting from `reduce`! is still connected and cyclic, so the rule schema `base` is not applicable.

A program can get stuck only in two situations: either it contains a subprogram **`if`** C **`then`** P **`else`** Q where C both can diverge from some graph and cannot produce a proper result from that graph, or it contains a subprogram B! where the loop's body B possesses the said property of C. The evaluation of such subprograms gets stuck because the inference rules for branching resp. iteration are not applicable.

Next we consider programs that produce infinitely many (non-isomorphic) results for some input. A simple example for such a program is given in Figure 17. GP programs showing this behaviour on some input can necessarily diverge from that input. This property is known as *bounded nondeterminism* [22].

Proposition (Bounded nondeterminism). *Let P be a program and G a graph in $\mathcal{G}$. If P cannot diverge from G, then $[\![P]\!]G$ is finite up to isomorphism.*

The reason is that for every configuration γ, the set $\{\delta \mid \gamma \to \delta\}$ is finite up to isomorphism of the graphs in configurations. In particular, the constraints on the syntax of conditional rule schemata ensure that for every rule schema r and every graph G in $\mathcal{G}$, there are up to isomorphism only finitely many graphs H such that $G \Rightarrow_{\mathrm{I}(r)} H$.



$$[\![\texttt{main}]\!]\,\textcircled{1} = \{\bot,\, \emptyset,\, \textcircled{2},\, \textcircled{2}\textcircled{2},\, \textcircled{2}\textcircled{2}\textcircled{2},\, \ldots\}$$

Fig. 17. Infinitely many results for the same input

An important role of a formal semantics is to provide a rigorous notion of program equivalence. We call two programs P and Q *semantically equivalent*, denoted by $P \equiv Q$, if $[\![P]\!] = [\![Q]\!]$. For example, the following equivalences hold for arbitrary programs C, P, P_1, P_2 and Q:

(1) $P;\ \texttt{skip} \equiv P \equiv \texttt{skip};P$
(2) $\texttt{fail};P \equiv \texttt{fail}$
(3) $\texttt{if}\ C\ \texttt{then}\ (P_1;\ Q)\ \texttt{else}\ (P_2;\ Q) \equiv (\texttt{if}\ C\ \texttt{then}\ P_1\ \texttt{else}\ P_2);\ Q$
(4) $P! \equiv \texttt{if}\ P\ \texttt{then}\ (P;\ P!)$

On the other hand, there are programs P such that

$$P;\ \texttt{fail} \not\equiv \texttt{fail}.$$

For, if P can diverge from some graph G, then $[\![P;\ \texttt{fail}]\!]G$ contains $\bot$ whereas $[\![\texttt{fail}]\!]G$ is empty.

6 Implementation

This section briefly describes the current implementation of GP, consisting of a graphical editor for programs and graphs, a compiler, and the York Abstract Machine (YAM). Figure 18 shows how these components interact, where GXL is the Graph Exchange Language [27] and YAMG is an internal graph format.

The graphical editor allows graph and program loading, editing and saving, and program execution on a given graph. Figure 19 shows a screenshot of the graphical editor, where the rule schema `expand` of the program `sierpinski` from Example 7 is being edited. The editor is implemented in Java and uses the prefuse data visualisation library [11], which provides automatic graph layout by a force-directed algorithm. The Sierpinski triangle of Figure 11, for example, was generated by this algorithm.

The York abstract machine (YAM) manages the graph on which a GP program operates, by executing low-level graph operations in bytecode format. The current graph is stored in a complex data structure which is designed to make graph interrogation very quick (at the cost of slightly slower graph updates). Typical query operations are “provide a list of all edges whose target is node n” and “provide a list of all nodes whose (possibly tagged) label has value 0 at position 1”.

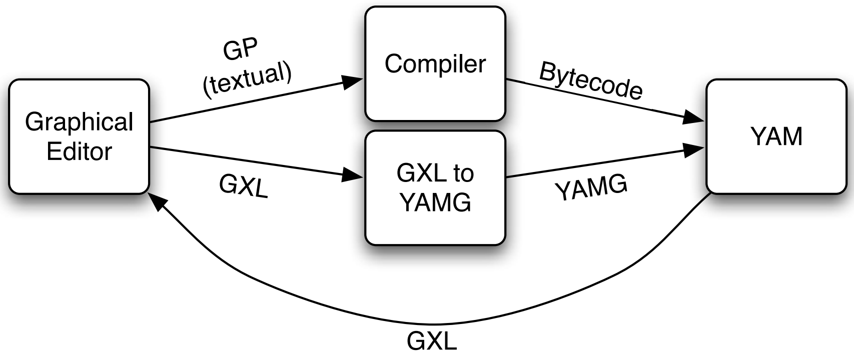
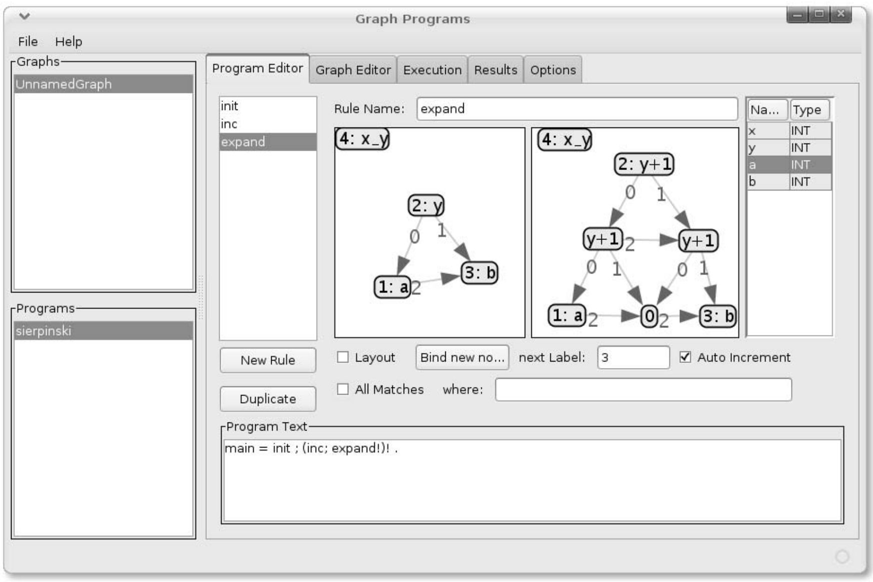


Fig. 18. Components of the GP system



```
main = init ; (inc; expand!)! .
```

Fig. 19. A screenshot of the graphical editor

The YAM is similar to Warren's abstract machine for Prolog [1] in that it handles GP's nondeterminism by backtracking, using a mixed stack of choice points and environment frames. Choice points consist of a record of the number of graph changes at their creation time, a program position to jump to if failure occurs when the choice point is the highest on the stack, and pointers to the previous choice and containing environment. The number of graph changes is recorded so that they can be undone during backtracking: using the stack of graph changes, the graph is recreated as it was at the choice point. Environment frames have a set of registers to store label elements or graph item identities, and an associated function and program position in the bytecode. They also show which environment and program position to return to.

The YAM provides instructions for handling nondeterminism by which the compiler constructs helper functions to implement backtracking. Nondeterministic choice between a set of rule schemata is handled by trying them in textual order until one succeeds. Before each is tried, the failure behaviour is configured to try the next. Nondeterministic choice between graph-item candidates for a match is handled by choosing and saving the first element, and on failure, using the saved previous answer to return and save the next element.

For efficiency reasons, the YAM is implemented in C. See also [14], where a more detailed description of (a slightly older version of) the YAM and its bytecode instructions is given.

The GP compiler is written in Haskell. It converts textually represented GP programs into YAM bytecode by translating each individual rule schema into a sequence of instructions for graph matching and transformation. These sequences are then composed by YAM function calls according to the meaning of GP's control constructs.

The compiler generates code for graph matching by decomposing each rule schema into a searchplan of node lookups, edge lookups (find an edge whose source and target have not been found yet) and extensions (find an edge whose source or target has been found). The choice and order of these search operations is determined by a list of priorities. For example, finding source or target of an edge that has already been found has higher priority (because it is cheaper) than finding an edge between nodes that have already been found. Searchplan generation is a common technique for graph matching and is also used in the implementations of PROGRES [28], Fujaba [15] and GrGen [8].

The semantics of GP assigns to an input graph of a program *all* possible output graphs. This is taken seriously by the implementation in that it provides users with the option to generate all results of a terminating program. (There is no guarantee of completeness for programs that can diverge, because the search for results uses a depth-first strategy.) In contrast, other graph-transformation languages do not fully exploit the nondeterministic nature of graph transformation. For example, AGG [6] makes its nondeterministic choices randomly, with no backtracking. Similarly, Fujaba has no backtracking. PROGRES [24] seems to be the only other graph-transformation language that provides backtracking.

7 Conclusion

We have demonstrated that GP is a rule-based language for high-level problem solving in the domain of graphs, freeing programmers from handling low-level data structures. The hallmark of GP is syntactic and semantic simplicity. Conditional rule schemata for graph transformation allow to express application conditions and computations on labels, in addition to structural changes.

The operational semantics describes the effect of GP's control constructs in a natural way and captures the nondeterminism of the language. In particular, powerful branching and iteration commands have been defined using the concept of finite failure. Destructive tests on the current graph can be hidden in the condition of the branching command, and nested loops can be coded since arbitrary subprograms can be iterated as long as possible.

The prototype implementation of GP is faithful to the semantics and computes a result for a (terminating) program whenever possible. It even provides the option to generate all possible results.

Future extensions of GP may include recursive procedures for writing complicated algorithms (see [25]) and a type concept for restricting the shape of graphs. A major goal is to support formal reasoning on graph programs. We plan to develop static analyses for properties such as termination and confluence (uniqueness of results), and a calculus and tool support for program verification.

Acknowledgements. I am grateful to Sandra Steinert and Greg Manning for their contributions to the design and implementation of GP [19,25,14,13,20].

References

1. Aït-Kaci, H.: Warren's Abstract Machine: A Tutorial Reconstruction. MIT Press, Cambridge (1991)
2. Bang-Jensen, J., Gutin, G.: Digraphs: Theory, Algorithms and Applications. Springer, Heidelberg (2000)
3. Clark, K.L.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 293–322. Plenum Press (1978)
4. Duffin, R.J.: Topology of series-parallel networks. Journal of Mathematical Analysis and Applications 10, 303–318 (1965)
5. Ehrig, H., Habel, A.: Graph grammars with application conditions. In: Rozenberg, G., Salomaa, A. (eds.) The Book of L, pp. 87–100. Springer, Heidelberg (1986)
6. Ermel, C., Rudolf, M., Taentzer, G.: The AGG approach: Language and environment. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) Handbook of Graph Grammars and Computing by Graph Transformation, vol. 2, ch. 14, pp. 551–603. World Scientific, Singapore (1999)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability. W.H. Freeman and Company, New York (1979)
8. Geiß, R., Batz, G.V., Grund, D., Hack, S., Szalkowski, A.: GrGen: A fast SPO-based graph rewriting tool. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 383–397. Springer, Heidelberg (2006)
9. Habel, A., Plump, D.: Computational completeness of programming languages based on graph transformation. In: Honsell, F., Miculan, M. (eds.) FOSSACS 2001. LNCS, vol. 2030, pp. 230–245. Springer, Heidelberg (2001)
10. Habel, A., Plump, D.: Relabelling in graph transformation. In: Corradini, A., Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2002. LNCS, vol. 2505, pp. 135–147. Springer, Heidelberg (2002)
11. Heer, J., Card, S.K., Landay, J.A.: Prefuse: A toolkit for interactive information visualization. In: Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI 2005), pp. 421–430. ACM Press, New York (2005)

12. Kleinberg, J., Tardos, É.: Algorithm Design. Addison Wesley, Reading (2006)
13. Manning, G., Plump, D.: The GP programming system. In: Proc. Graph Transformation and Visual Modelling Techniques (GT-VMT 2008). Electronic Communications of the EASST, vol. 10 (2008)
14. Manning, G., Plump, D.: The York abstract machine. In: Proc. Graph Transformation and Visual Modelling Techniques (GT-VMT 2006). Electronic Notes in Theoretical Computer Science, vol. 211, pp. 231–240. Elsevier, Amsterdam (2008)
15. Nickel, U., Niere, J., Zündorf, A.: The FUJABA environment. In: Proc. International Conference on Software Engineering (ICSE 2000), pp. 742–745. ACM Press, New York (2000)
16. Nielson, H.R., Nielson, F.: Semantics with Applications: An Appetizer. Springer, Heidelberg (2007)
17. Peitgen, H.-O., Jürgens, H., Saupe, D.: Chaos and Fractals, 2nd edn. Springer, Heidelberg (2004)
18. Plotkin, G.D.: A structural approach to operational semantics. Journal of Logic and Algebraic Programming 60–61, 17–139 (2004)
19. Plump, D., Steinert, S.: Towards graph programs for graph algorithms. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 128–143. Springer, Heidelberg (2004)
20. Plump, D., Steinert, S.: The semantics of graph programs (submitted for publication, 2009)
21. Rensink, A.: The GROOVE simulator: A tool for state space generation. In: Pfaltz, J.L., Nagl, M., Böhlen, B. (eds.) AGTIVE 2003. LNCS, vol. 3062, pp. 479–485. Springer, Heidelberg (2004)
22. Reynolds, J.C.: Theories of Programming Languages. Cambridge University Press, Cambridge (1998)
23. Schürr, A.: Operationales Spezifizieren mit programmierten Graphersetzungssystemen. Deutscher Universitäts-Verlag (1991) (in German)
24. Schürr, A., Winter, A., Zündorf, A.: The PROGRES approach: Language and environment. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) Handbook of Graph Grammars and Computing by Graph Transformation, vol. 2, ch. 13, pp. 487–550. World Scientific, Singapore (1999)
25. Steinert, S.: The Graph Programming Language GP. PhD thesis, The University of York (2007)
26. Taentzer, G., Biermann, E., Bisztray, D., Bohnet, B., Boneva, I., Boronat, A., Geiger, L., Geiß, R., Horvath, Á., Kniemeyer, O., Mens, T., Ness, B., Plump, D., Vajk, T.: Generation of Sierpinski triangles: A case study for graph transformation tools. In: Schürr, A., Nagl, M., Zündorf, A. (eds.) AGTIVE 2007. LNCS, vol. 5088, pp. 514–539. Springer, Heidelberg (2008)
27. Winter, A.J., Kullbach, B., Riediger, V.: An overview of the GXL graph exchange language. In: Diehl, S. (ed.) Dagstuhl Seminar 2001. LNCS, vol. 2269, pp. 324–336. Springer, Heidelberg (2002)
28. Zündorf, A.: Graph pattern matching in PROGRES. In: Cuny, J., Engels, G., Ehrig, H., Rozenberg, G. (eds.) Graph Grammars 1994. LNCS, vol. 1073, pp. 454–468. Springer, Heidelberg (1996)

Appendix: Natural Pushouts

This appendix defines the natural pushouts on which direct derivations are based (see Section 2) and characterises them in terms of ordinary pushouts. Further properties can be found in [10].

$$\begin{tikzcd}
K \arrow[r, "b"] \arrow[d, "d"'] & R \arrow[d] \\
D \arrow[r] & H
\end{tikzcd}$$

Fig. 20. A pushout diagram

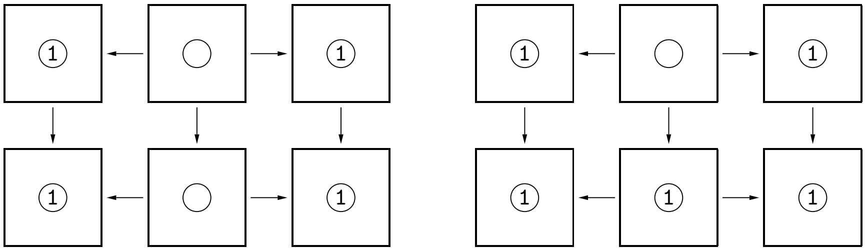


Fig. 21. A natural and a non-natural double-pushout

A diagram of morphisms between partially labelled graphs as in Figure 20 is a *pushout* if the following conditions are satisfied:[5]

- Commutativity: $K \to R \to H = K \to D \to H$.
- Universal property: For every pair of graph morphisms $\langle R \to H', D \to H' \rangle$ such that $K \to R \to H' = K \to D \to H'$, there is a unique morphism $H \to H'$ such that $R \to H' = R \to H \to H'$ and $D \to H' = D \to H \to H'$.

The diagram is a *pullback* if commutativity holds and the following universal property:

- For every pair of graph morphisms $\langle K' \to R, K' \to D \rangle$ such that $K' \to R \to H = K' \to D \to H$, there is a unique morphism $K' \to K$ such that $K' \to R = K' \to K \to R$ and $K' \to D = K' \to K \to D$.

A pushout is *natural* if it is simultaneously a pullback.

[5] Given graph morphisms $f\colon A \to B$ and $g\colon B \to C$, we write $A \to B \to C$ for the composition $g \circ f\colon A \to C$.

Proposition (Characterisation of natural pushouts [10]). *If b is injective, then the pushout in Figure 20 is natural if and only if for all $v \in V_K$,*

$$l_K(v) = \perp \textit{ implies } l_R(b_V(v)) = \perp \textit{ or } l_D(d_V(v)) = \perp.$$

For example, the double-pushout on the left of Figure 21 consists of natural pushouts, the double-pushout on the right consists of non-natural pushouts.

Canonical Reduction Systems in Symbolic Mathematics

Franz Winkler

RISC, Johannes Kepler University, Linz, Austria
Franz.Winkler@risc.jku.at
http://risc.uni-linz.ac.at/people/winkler

Abstract. Many algorithmic methods in mathematics can be seen as constructing canonical reduction systems for deciding membership problems. Important examples are the Gauss elimination method for linear systems, Euclid's algorithm for computing greatest common divisors, Buchberger's algorithm for constructing Gröbner bases, or the Knuth-Bendix procedure for equational theories. We explain the basic concept of a canonical reduction system and investigate the close connections between these algorithms.

1 Introduction

The biological theory of evolution exhibits many instances of similar solutions having been developed for similar problem; examples are the wings of insects, birds, and bats, or the different realizations of light sensitive organs such as eyes. The same phenomenon can be observed in the development of the sciences, and also in particular in mathematics. Many algorithmic methods in different fields of mathematics, e.g. linear algebra, commutative algebra, or logic, can be seen as constructing canonical reduction systems for deciding membership problems. Important examples are the Gauss elimination method for linear systems, Euclid's algorithm for computing greatest common divisors, Buchberger's algorithm for constructing Gröbner bases, or the Knuth-Bendix procedure for equational theories. Here we continue the work started in [4], where we have demonstrated the relations between Buchberger's algorithm for the construction of Gröbner bases and the Knuth-Bendix procedure for the construction of canonical term rewriting systems. We explain the basic concept of canonical reduction systems and investigate the close connections between these algorithms.

1.1 Canonical Reduction Relations and Systems

Canonical reduction systems (see also [5], Chapter 8) are supposed to solve the following kind of problem:

- given a mathematical structure $\mathcal{S}$ and a congruence relation $\cong$ on $\mathcal{S}$ (i.e. $\cong \subseteq \mathcal{S}^2$) defined by a finite set of generators $G = \{(l_i, r_i) \,|\, l_i \cong r_i \text{ for } 1 \leq i \leq n\}$ (i.e. $\cong = \cong_G$),

- we want to construct a new set of generators $\hat{G}$ for this congruence relation, which makes it easy to decide, for any given $s, t \in \mathcal{S}$, whether $s \cong_G t$.

In order to solve such decision problems we introduce a reduction relation $\longrightarrow_G \subseteq \mathcal{S} \times \mathcal{S}$. So, to start with, $\longrightarrow_G$ is simply a binary relation on $\mathcal{S}$. But we will want this relation to have certain properties. Let us first introduce the following notation: for any binary relation $\longrightarrow$ we denote

by $\longleftarrow$ the inverse,
by $\longrightarrow^*$ the reflexive transitive closure, and
by $\longleftrightarrow^*$ the reflexive symmetric transitive closure

of $\longrightarrow$. We will want $\longrightarrow_G$ to have the following properties:

- $\cong_G = \longleftrightarrow_G^*$, i.e. the symmetric reflexive transitive closure of $\longrightarrow_G$ is equal to the congruence generated by G, and
- $\longrightarrow_G$ is **terminating** or **Noetherian**, i.e. every reduction chain $s_0 \longrightarrow_G s_1 \longrightarrow_G \cdots$ is finite.

In addition to being Noetherian, the reduction relation $\longrightarrow_G$ might also be **Church-Rosser**, i.e. $s \longleftrightarrow_G^* t$ implies the existence of a common successor u s.t. $s \longrightarrow_G^* u \longleftarrow_G^* t$. In particular this means that two irreducible elements s, t are congruent if and only if they are syntactically equal.

In case $\longrightarrow_G$ is both Noetherian and Church-Rosser, we call $\longrightarrow_G$ a **canonical reduction relation** and we call G a **canonical reduction system** for the congruence $\cong$.

A canonical reduction system yields the following decision procedure for the underlying congruence $\cong = \longleftrightarrow_G^*$: in order to decide whether $s \cong t$ for $s, t \in \mathcal{S}$,

- reduce s and t to (any) irreducible s' and t' s.t.

$$\begin{aligned} s = s_0 \longrightarrow_G s_1 \longrightarrow_G \cdots \longrightarrow_G s_m = s', \\ t = t_0 \longrightarrow_G t_1 \longrightarrow_G \cdots \longrightarrow_G t_n = t' \end{aligned}$$

  (s' and t' are called **normal forms** of s and t, respectively);
- check whether $s' = t'$; if so then $s \cong_G t$, otherwise not.

1.2 Generating Canonical Reduction Systems

In general a given set of generators G (or its corresponding reduction relation $\longrightarrow_G$) for a congruence $\cong$ will not have the Church-Rosser property. So our goal now becomes to transform G into an equivalent canonical system $\hat{G}$. It turns out that the Church-Rosser property is equivalent to the simpler property of **confluence**, meaning that if s, t have a common predecessor in finitely many steps, $s \uparrow_G^* t$, then they also have a common successor, $s \downarrow_G^* t$. Furthermore, under the assumption of Noetherianity, confluence is equivalent to **local confluence**, meaning that if s, t have a common predecessor in a single step, $s \uparrow_G t$, then they also have a common successor, $s \downarrow_G^* t$. In many interesting cases, such as the algorithms discussed in this paper, the test for local confluence can be reduced to the test of finitely many **critical pairs**. These are pairs (s, t) s.t. $s \uparrow_G t$, and all other such situations can be regarded as specializations of critical pairs. So if we can prove that for all critical pairs there are common successors, then we have a canonical reduction system. Otherwise we take normal forms $s' \neq t'$ of s, t, respectively, and add the pair (s', t') to G. Since obviously $s' \longleftrightarrow_G^* t'$, we also have $s' \cong t'$; so the addition of this new pair to G will not violate the requirement $\cong = \longleftrightarrow_G^*$. Of course we have to ensure that by this modification of the reduction system G the Noetherianity of $\longrightarrow_G$ is preserved. In general this is hard, indeed undecidable in some cases such as term rewriting systems (cf. [1],[2]). In any case, we keep considering critical pairs, adding new pairs to the set of generators G, and in this way creating more critical pairs. So the question is whether this process will ever terminate and deliver a canonical reduction system. Indeed, for the cases of Gauss elimination, Euclid's algorithm, and Gröbner bases, such a canonical system will finally be produced. But in the case of the Knuth-Bendix procedure for term rewriting systems, the completion process might not yield such a canonical system in finitely many steps.

Let us now demonstrate this approach for the cases listed above.

2 Gauss Elimination in Linear Algebra

We consider the following setting:

- the mathematical structure $\mathcal{S}$ is a finite dimensional vector space V over a field K; w.l.o.g. $V = K^n$;
- as the generating elements for the congruence we take a basis B for a subvectorspace $W = \text{span}(B)$;
- now the equivalence relation is $v_1 \cong_W v_2 \iff v_1 - v_2 \in W$; and $\cong_W$ is generated by $b \cong_W 0$ for $b \in B$.

The central problem then is to decide whether, for given $v \in V$,

$$v \cong_W 0 \text{ , i.e. } v \in \text{span}(B) = W \text{ .}$$

Every basis B of W generates this congruence; simply let v be congruent w.r.t. B to w if $v - w$ is a linear combination of B. We write $\cong_B$ for this congruence, and we observe that $\cong_B = \cong_W$ for every basis B of W.

If the basis B (considered as lines of a matrix) is triangular, then this central problem becomes easily decidable. The triangulation or elimination method of Gauss transforms B into such a triangular basis. Let us see that what Gauss elimination does is exactly the construction of a canonical reduction system.

The basis B induces a reduction relation $\longrightarrow_B$ on V as follows:

- for a non-zero vector $b = (0, \ldots, 0, b_i, \ldots, b_n)$ with $b_i \neq 0$ we say $\text{lead}(b) = i$;
- now the reduction relation $\longrightarrow_b$ by a single vector b is

$$c = (c_1, \ldots, c_i \neq 0, \ldots, c_n) \ \longrightarrow_b \ c - \frac{c_i}{b_i} \cdot b$$

and for a finite set B we say

$$c \longrightarrow_B d \quad \Longleftrightarrow \quad \exists b \in B : c \longrightarrow_b d \; .$$

It is not hard to see that for every B the reduction relation $\longrightarrow_B$ has the following properties:

- $\longrightarrow_B$ is terminating
- $v \longleftrightarrow_B^* w$ if and only if $v \cong_B w$.

But $\longrightarrow_B$ in general is **not** confluent. Consider the following example: let

$$B = \{\underbrace{(1,0,0)}_{b_1}, \underbrace{(1,1,1)}_{b_2}\}$$

be a basis for a subvectorspace $W = \text{span}(B)$ of $\mathbb{Q}^3$. Then

$$w_1 = (0,2,2) \longleftarrow_{b_1} (1,2,2) = v \longrightarrow_{b_2} (0,1,1) = w_2$$

and both reduction results are irreducible. So w_1 and w_2 are congruent, $w_1 \cong_B w_2$, but this cannot be determined by reduction w.r.t B.

So what can we do in order to transform $\longrightarrow_B$ into a confluent reduction relation? Well, according to Gauss elimination, we consider the elements of B as lines in a matrix (also denoted by B) and transform the matrix

$$B \;=\; \begin{pmatrix} b_1 \\ \cdots \\ b_m \end{pmatrix}$$

to row echelon form. This means we look at situations, where the component of a vector can be reduced by (at least) two different generators b_j and b_k. Clearly we can simplify this situation to a situation of critical pairs, where a unit vector

$$e_i = (0, \ldots, 0, \underbrace{1}_{i-\text{th position}}, 0, \ldots, 0) \; ,$$

can be reduced by two different generators b_j and b_k. This means that $\text{lead}(b_j) = i = \text{lead}(b_k)$, and

$$e_i - b_j \longleftarrow_{b_j} e_i \longrightarrow_{b_k} e_i - b_k$$

(here we have assumed, w.l.o.g., that the components of both b_j and b_k at their leading positions are 1). These reduction results are congruent w.r.t. $\cong_B$, so their difference $b_{m+1} := b_j - b_k$ is in W. If $b_{m+1} = 0$, then there was no divergence anyway; otherwise we add b_{m+1} to the basis B, thereby enforcing this particular divergence of reduction to converge:

$$\begin{array}{ll} \text{either} & e_i - b_j \longrightarrow_{b_{m+1}} e_i - b_k \\ \text{or} & e_i - b_k \longrightarrow_{b_{m+1}} e_i - b_j \end{array}$$

Observe that this represents exactly a step in the formation of the row echelon form of the matrix (basis) B.

This process terminates and yields a set of generators $\hat{B}$ s.t.

- $\longleftrightarrow^*_B \;=\; \cong_W \;=\; \longleftrightarrow^*_{\hat{B}}$,
- $\longrightarrow_{\hat{B}}$ is both Noetherian and confluent.

So we can decide the membership problem for W by reduction w.r.t. $\hat{B}$.

For our example above this means the following:

$$\begin{array}{rll} B = \{b_1, b_2\} : & b_1 = & (1,0,0) \\ & b_2 = & (1,1,1) \\ & ------- & \\ & b_3 = & (0,1,1) \\ & & \qquad\qquad \to \hat{B} = \{b_1, b_2, b_3\} \end{array}$$

Now $\hat{B}$ spans the same vector space W, and we can use the reduction w.r.t.$\hat{B}$ to decide membership in W:

$$\begin{array}{l} (1,2,2) \longrightarrow_{b_1} (0,2,2) \longrightarrow_{b_3} (0,0,0) \\ (1,2,2) \longrightarrow_{b_2} (0,1,1) \longrightarrow_{b_3} (0,0,0) \end{array}$$

The vector $(1,2,2)$ is indeed in W.

In the end we can clean up the basis by keeping only one element with the same lead; in a confluent system we would never need the others, because in every reduction in which we might want to use one of these basis elements, we might instead use the one we keep. Such an interreduced basis $\hat{B}$ is basically the Hermite matrix associated to B.

3 Euclid's Algorithm for gcds of Univariate Polynomials

We consider the following setting:

- the mathematical structure $\mathcal{S}$ is $K[x]$, the ring of polynomials over a field K;
- as the generating elements for the congruence we take two (or finitely many) non-zero polynomials $F = \{f_1(x), f_2(x)\} \subset K[x]$, generating an ideal $I = \langle F \rangle$ in $K[x]$; F is called a basis for the ideal I;
- now the equivalence relation is $h_1 \equiv_I h_2 \iff h_1 - h_2 \in I$.

The central problem then is to decide whether, for given $h \in K[x]$,

$$h \equiv_I 0 \text{ , i.e. } h \in \langle F \rangle = I \ .$$

If g is the greatest common divisor (gcd) of f_1 and f_2, then $\langle g \rangle = I = \langle f_1, f_2 \rangle$, and the central question can be easily decided as

$$h \equiv_I 0 \iff g | h \ .$$

The Euclidean algorithm computes exactly this gcd, by a sequence of remainders. W.l.o.g. assume that the degree of f_1 is at least as high as the degree of f_2. We let $r_1 := f_1, r_2 := f_2$ be the first two remainders in our sequence; an r_{i+2} is then simply the remainder of r_i on division by r_{i+1}. Throughout the algorithm we always have

$$\gcd(f_1, f_2) \;=\; \gcd(r_i, r_{i+1})\;.$$

It is easy to see that this process of remaindering must terminate with, say, $r_k \neq 0$, but $r_{k+1} = 0$. Then we have

$$\gcd(f_1, f_2) \;=\; \gcd(r_k, 0) \;=\; r_k\;.$$

Throughout the Euclidean algorithm the ideal I remains unchanged, since all these remainders clearly are in I.

An ideal basis F induces a reduction relation $\longrightarrow_F$ on $K[x]$ as follows:

- for a non-zero polynomial $f(x) = f_n x^n + \cdots f_1 x + f_0$ with $f_n \neq 0$ we say $\mathrm{lead}(f) = \deg(f) = n$;
- now the reduction relation $\longrightarrow_f$ by a single polynomial f is

$$p = p_m x^m + \cdots + \underbrace{p_i}_{\neq 0} x^i + \cdots + p_0 \;\longrightarrow_f\; p \;-\; \frac{p_i}{f_n} x^{i-n} f(x), \qquad \text{if } i \geq n$$

and for a finite basis F we say

$$p \longrightarrow_F q \quad\Longleftrightarrow\quad \exists f \in F : p \longrightarrow_f q\;.$$

It is not hard to see that for every F the reduction relation $\longrightarrow_F$ has the following properties:

- $\longrightarrow_F$ is terminating, and
- $p \longleftrightarrow^*_F q$ if and only if $p \equiv_I q$.

But $\longrightarrow_F$ in general is **not** confluent. Consider the following example: let

$$F = \{\underbrace{x^5 + x^4 + x^3 - x^2 - x - 1}_{f_1},\; \underbrace{x^4 + x^2 + 1}_{f_2}\}$$

be a polynomial basis for the ideal $I = \langle f_1, f_2 \rangle$. Then

$$\underbrace{-x^3 + x^2 + x + 2}_{q_1} \longleftarrow_{f_2} -x^4 - x^3 + x + 1 \longleftarrow_{f_1} \underbrace{x^5 - x^2}_{p} \longrightarrow_{f_2} \underbrace{-x^3 - x^2 - x}_{q_2}$$

and both reduction results are irreducible. So q_1 and q_2 are congruent, $q_1 \equiv_I q_2$, but this cannot be determined by reduction w.r.t. F.

So what can we do in order to transform $\longrightarrow_F$ into a confluent reduction relation? Well, we consider terms of least degree which can be reduced by two different polynomials. All other diverging reductions can be seen as derived from such divergences. W.l.o.g. we may assume that all polynomials in our remainder sequence are monic; instead of the actual remainder, we simply take its monic associate. If $d_i = \deg(f_i)$, then x^{d_i} can be reduced both by f_i and f_{i+1}:

$$x^{d_i} - f_i \longleftarrow_F x^{d_i} \longrightarrow_F x^{d_i} - x^{d_i - d_{i-1}} \cdot f_{i+1} \ .$$

If these reduction results are the same, then this divergence of reduction converges, and we are done. Otherwise we add the difference $f_i - x^{d_i - d_{i+1}} f_{i+i}$ to the basis; this obviously leaves the ideal I unchanged. In fact we might as well reduce both sides to normal forms, and then add their difference to the basis. What we have done is simply a step in the division algorithm. In this way we consider all pairs of polynomials in the basis (it can be demonstrated that considering subsequent remainders is sufficient); i.e. we compute a remainder sequence starting with f_1, f_2:

$$\begin{array}{lll} F = \{f_1, f_2\} : f_1 & & \\ \phantom{F = \{f_1, f_2\} : } f_2 & & \\ \phantom{F = \{f_1, f_2\} : } --- & & \\ \phantom{F = \{f_1, f_2\} : } f_3 & := \mathrm{rem}(f_1, f_2) & \\ \phantom{F = \{f_1, f_2\} : } \vdots & & \\ \phantom{F = \{f_1, f_2\} : } f_k & (\neq 0) & \\ \phantom{F = \{f_1, f_2\} : } f_{k+1} & (= 0) & \hat{F} = \{f_1, f_2, \ldots, f_k\} \end{array}$$

This process terminates and yields a set of generators $\hat{F}$ containing $f_k = \gcd(f_1, f_2)$. In fact we have

- $\overset{*}{\longleftrightarrow}_F \ = \ \equiv_I \ = \ \overset{*}{\longleftrightarrow}_{\hat{F}}$, and
- $\longrightarrow_{\hat{F}}$ is both Noetherian and confluent.

So we can decide the membership problem for I by reduction w.r.t. $\hat{F}$:

$$h \in \langle F \rangle \iff f_k | h \iff h \longrightarrow_{\hat{F}} 0 \ .$$

For our example above this means the following:

$$\begin{array}{rlll} F = \{f_1, f_2\} : f_1 = & x^5 + x^4 + x^3 - x^2 - x - 1 & & \\ f_2 = & x^4 + x^2 + 1 & & \\ --------- & & & \\ f_3 = & x^4 - x^2 - 2x - 1 = & & f_1 - x \cdot f_2 \\ f_4 = & x^2 + x + 1 = & & \frac{1}{2}(f_2 - f_3) \\ f_5 = & 0 = & & f_3 - (x^2 - x - 1) f_4 \\ & & & \rightarrow \hat{F} = \{f_1, \ldots, f_4\} \end{array}$$

Now $\hat{F}$ generates the same ideal I, and we can use the reduction w.r.t. $\hat{F}$ to decide membership in I:

$$0 \longleftarrow_{f_3} -x^3 + x^2 + x + 2 \longleftarrow_{f_1, f_2} x^5 - x^2 \longrightarrow_{f_2} -x^3 - x^2 - x \longrightarrow_{f_3} 0 \ .$$

So $x^5 - x^2 \in I$.

In the end we can again interreduce the elements in the confluent reduction system $\hat{F}$. Whenever we might want to use a basis polynomial different from f_k in a reduction, we might as well use f_k. So since our reduction system is now confluent, we don't need the other basis polynomials any more; we simply keep $\hat{F} = \{f_k\}$.

4 Gröbner Bases in Multivariate Polynomial Rings

We consider the following setting:

- the mathematical structure $\mathcal{S}$ is $K[x_1, \ldots, x_n]$, the ring of multivariate polynomials over a field K;
- as the generating elements for the congruence we take finitely many non-zero polynomials $F = \{f_1, \ldots, f_m\} \subset K[x_1, \ldots, x_n]$ generating an ideal $I = \langle F \rangle$ in $K[x_1, \ldots, x_n]$;
- now the equivalence relation is $h_1 \equiv_I h_2 \iff h_1 - h_2 \in I$.

The central problem then is to decide whether, for given $h \in K[x_1, \ldots, x_n]$,

$$h \equiv_I 0 \text{ , i.e. } h \in \langle F \rangle = I \text{ .}$$

As in the case of univariate polynomials we would like to introduce a reduction w.r.t. a basis F, and then add certain polynomials to the basis in order to make the corresponding reduction relation confluent. Such an ideal basis we will then call a **Gröbner basis**. Buchberger's algorithm for the construction of Gröbner bases does exactly that.

For introducing a reduction relation $\longrightarrow_F$, we first have to linearly order the multivariate terms $x_1^{e_1} \cdots x_n^{e_n}$. In the univariate case we did not have any choice; the only reasonable ordering is induced by the degree. But in the multivariate case we have much more freedom. We need to choose an ordering respecting the multiplicative structure of the set of terms, called an **admissible ordering**; i.e.

- $1 = x^{(0,\ldots,0)} \leq s$ for every term s, and
- if $s \leq t$ and u any term, then $s \cdot u \leq t \cdot u$.

There is an abundance of such admissible orderings; e.g. lexicographic orderings, graduated lexicographic orderings, and many others. Admissible orderings are completely classified. Once we have chosen an admissible ordering $<$ of the terms, every non-zero polynomial f has a well-defined **leading term** $\mathrm{lead}(f)$ (the highest term in the ordering appearing with non-zero coefficient in f) and a non-zero **leading coefficient** $\mathrm{lc}(f)$, the coefficient of $\mathrm{lead}(f)$. By $\mathrm{le}(f)$ we denote the exponent (vector) of $\mathrm{lead}(f)$.

Now we are ready for defining the reduction relation $\longrightarrow_F$ on $K[x_1, \ldots, x_n]$ (for the fixed admissible term ordering $<$): for a non-zero polynomial

$$p = p_{\mathrm{le}(p)} x^{\mathrm{le}(p)} + \cdots + p_e x^{e=(e_1,\ldots,e_n)} + \cdots \text{ ,} \qquad \text{with } p_e \neq 0$$

we define $$p \quad \longrightarrow_f \quad p - \frac{p_e}{\mathrm{lc}(f)} x^{e-\mathrm{le}(f)} f(x), \qquad \text{if } e - \mathrm{le}(f) \in \mathbb{N}^n$$

and $$p \longrightarrow_F q \quad \Longleftrightarrow \quad \exists f \in F : p \longrightarrow_f q \,.$$

Again, as in the univariate case, one can prove that $\longrightarrow_F$ has the following properties:

- $\longrightarrow_F$ is terminating, and
- $p \longrightarrow_F^* q$ if and only if $p \equiv_I q$.

But $\longrightarrow_F$ in general is **not** confluent. Consider the following example: let

$$F = \{\underbrace{x^2y^2 + y - 1}_{f_1},\ \underbrace{x^2y + x}_{f_2}\}$$

be a basis for the polynomial ideal $I = \langle f_1, f_2 \rangle$. Then

$$q_1 = -y + 1 \longleftarrow_{f_1} p = x^2y^2 \longrightarrow_{f_2} -xy = q_2$$

and both results are irreducible. So q_1 and q_2 are congruent, $q_1 \equiv_F q_2$, but this cannot be determined by reduction w.r.t. F.

So what do we do in order to transform $\longrightarrow_F$ into a confluent reduction relation? Well, as in the previous cases (Gauss elimination, Euclidean algorithm) we investigate the "smallest" situations in which something can be reduced in essentially two different ways. We look at terms x^e which can be reduced w.r.t. two different generators f_j, f_k. This means that $\mathrm{lead}(f_j)|x^e$ and also $\mathrm{lead}(f_k)|x^e$. The (finitely many) smallest such situations occur when $x^e = \mathrm{lcm}(\mathrm{lead}(f_j), \mathrm{lead}(f_k))$ (least common multiple), and all the other cases are instantiations of such basic situations (see [5] for details). We reduce x^e both modulo f_j and f_k, getting some g_j and g_k, respectively. g_j and g_k may be further reduced modulo $\longrightarrow_F$ to normal forms g_j' and g_k', respectively:

$$g_j' \longleftarrow_F^* g_j \longleftarrow_{f_j} x^e = \mathrm{lcm}(\mathrm{lead}(f_j), \mathrm{lead}(f_k)) \longrightarrow_{f_k} g_k \longrightarrow_F^* g_k'$$

Actually we reduce $g_j - g_k$, the so-called **S-polynomial** of f_j and f_k, to a normal form h. If $h = 0$, then this divergence of reduction converges, and we are done. Otherwise we observe that $h \in I$. So if we add h to the basis F, then this divergence can be resolved, and the ideal remains unchanged.

Of course, now we have a new element in the basis, and there are more S-polynomials to be considered. But this process terminates and yields a set of generators $\hat{F}$ s.t.

- $\longleftrightarrow_F^* \;=\; \equiv_I \;=\; \longleftrightarrow_{\hat{F}}^*$, and
- $\longrightarrow_{\hat{F}}$ is both Noetherian and confluent.

So we have computed a Gröbner basis $\hat{F}$ for the ideal I w.r.t. the term ordering $<$. With the Gröbner basis $\hat{F}$ for I, we can decide the membership problem for I by reduction w.r.t. $\hat{F}$. If in the end we interreduce the elements in $\hat{F}$, we get a **minimal Gröbner basis** for the ideal I.

For our example above this means the following. We choose an admissible term ordering, say graduated lexicographic with $x < y$. Then we consider S-polynomials and reduce them to normal forms. This leads to the following sequence of polynomials being added to the basis:

$$\begin{array}{lll} F: & f_1 = & x^2y^2 + y - 1 \\ & f_2 = & x^2y + x \\ & - - - - - - - - - & \\ & f_3 = & -xy + y - 1 = f_1 - y \cdot f_2 \\ & f_4 = & y - 1 = \qquad\quad f_2 + (x+1)f_3 \\ & f_5 = & -x = \qquad\qquad f_3 + (x-1)f_4 \\ & & \qquad\qquad\qquad \to \hat{F} = \{f_1, \ldots, f_5\} \end{array}$$

Now $\hat{F}$ generates the same ideal I, and we can use the reduction w.r.t. $\hat{F}$ to decide membership in I:

$$\begin{array}{llll} x^2y^2 \longrightarrow_{f_1} & -y+1 & \longrightarrow_{f_4} & 0 \\ x^2y^2 \longrightarrow_{f_2} & -xy & \longrightarrow_{f_5} & 0 \end{array}$$

So $x^2y^2 \in I$. The minimal Gröbner basis for I is $\{x, y-1\}$.

5 The Knuth-Bendix Procedure for Term Rewriting Systems

We consider the following setting:

- a term algebra $\mathcal{T}(\Sigma, V)$ over a signature Σ and variables V;
- $E = \{s_i = t_i \mid i \in I\}$ a set of equations over $\mathcal{T}$ generating an equational theory $=_E$;
- now the equivalence relation is $s \equiv_E t \iff s = t \in =_E$.

The equational theory $=_E$ is the set of all equations which can be derived from E by reflexivity, symmetry, transitivity, substitution, and replacing equals by equals; confer [1], [2].

The central problem then is to decide whether, for given $s, t \in \mathcal{T}(\Sigma, V)$,

$$s =_E t \ .$$

We define a reduction relation on $\mathcal{T}(\Sigma, V)$ by orienting the equations in E

$$e_i : \quad s_i = t_i$$

in one of the ways (according to a reduction ordering)

$$r_i : \quad s_i \longrightarrow t_i \quad \text{or} \quad t_i \longrightarrow s_i$$

(w.l.o.g. assume $r_i : s_i \longrightarrow t_i$). This leads to a so-called **rewrite rule system (RRS)**

$$R = \{r_i \mid i \in I\} \ .$$

The reduction $\longrightarrow_R$ works in the following way: if there is a substitution σ and a position p in the term u, such that σ applied to s_i equals the subterm of u at position p, i.e. $\sigma(s_i) = u_{|p}$, then this subterm of u can be replaced by $\sigma(t_i)$:

$$u \longrightarrow_R v \qquad \Longleftrightarrow \qquad \exists p, i, \sigma : u_{|p} = \sigma(s_i), \text{ and } v = u[p \leftarrow \sigma(t_i)] \ .$$

Here $u[p \leftarrow \sigma(t_i)]$ means that in u we replace the subterm at position p by the term $\sigma(t_i)$.

In general the termination property is undecidabel for rewrite rule systems. But there are several sufficient conditions; e.g. $s_i > t_i$ w.r.t. a reduction ordering. For the following let us assume that the rules can be ordered w.r.t. such a reduction ordering. Then $\longrightarrow_R$ has the following properties:

- $\longrightarrow_R$ is terminating, and
- $\longleftrightarrow^*_R \; = \; =_E$.

But $\longrightarrow_R$ in general is **not** confluent. Consider the example of group theory; i.e. let G consist of the axioms

$$\begin{aligned} G \;=\; \{ & (1) \;\; 1 \cdot x = x, \\ & (2) \;\; x^{-1} \cdot x = 1, \\ & (3) \;\; (x \cdot y) \cdot z = x \cdot (y \cdot z) \;\; \} \ , \end{aligned}$$

which are oriented (lexicographic path ordering with ${}^{-1} > \cdot > 1$) to give the rewrite rule system

$$\begin{aligned} R \;=\; \{ & (1) \;\; 1 \cdot x \longrightarrow x, \\ & (2) \;\; x^{-1} \cdot x \longrightarrow 1, \\ & (3) \;\; (x \cdot y) \cdot z \longrightarrow x \cdot (y \cdot z) \;\; \} \ . \end{aligned}$$

Then

$$x^{-1} \cdot (x \cdot y) \;\longleftarrow_{(3)}\; (x^{-1} \cdot x) \cdot y \;\longrightarrow_{(2)}\; 1 \cdot y \;\longrightarrow_{(1)}\; y$$

Both results are irreducible, they are congruent modulo $=_E$, but they have no common successor.

So again the goal is to transform the RRS R into an equivalent confluent RRS $\hat{R}$,

$$\longleftrightarrow^*_R \;=\; \longleftrightarrow^*_{\hat{R}} \ .$$

As in the previous cases (Gauss elimination, Euclidean algorithm, Gröbner bases) we investigate "smallest" situations in which a term can be reduced in essentially two different ways. Towards this end, we consider (not necessarily different) rules

$$r : s \longrightarrow t \ , \qquad r' : s' \longrightarrow t' \ ,$$

a most general unifier (substitution) σ, and a position p in the term s ($s_{|p}$ not being a variable) s.t.

$$\sigma(s') \;=\; \sigma(s_{|p}) \ .$$

In this case we get the following divergence in reduction

$$v = \sigma(t) \longleftarrow_r \sigma(s) = u \longrightarrow_{r'} \sigma(s[p \leftarrow t']) = v' \ .$$

The pair of terms (v, v') is called a **critical pair** of the RRS R. The components of the critical pair (v, v') are obviously equal modulo $=_E$; so are normal forms w and w' to which v and v' can be reduced, respectively. If $w \neq w'$, then we try to orient them into a new rule $w \longrightarrow w'$ or $w' \longrightarrow w$, which does not violate the termination property of the RRS.

In contrast to the previous cases (Gauss elimination, Euclidean algorithm, Gröbner bases), there is no guarantee that this completion process will terminate. Critical pairs will lead to new rules, which lead to new critical pairs, which will lead to new rules, and so on. Also we might get stuck in a situation where the normal forms of a critical pairs, w and w', cannot be oriented into a rule without violating the termination property. But if this process terminates and yields a RRS $\hat{R}$ then

- $\longleftrightarrow_R^* \ = \ =_E \ = \ \longleftrightarrow_{\hat{R}}^*$, and
- $\longrightarrow_{\hat{R}}$ is both Noetherian and confluent.

So we can decide the equality modulo E by reduction w.r.t. $\hat{R}$. In the end we can interreduce the RRS $\hat{R}$ and so get a minimal RRS for $=_E$.

For the example of group theory this means that because of

$$x^{-1} \cdot (x \cdot y) \ \longleftarrow_{(3)} \ (x^{-1} \cdot x) \cdot y \ \longrightarrow_{(2)} \ 1 \cdot y \ \longrightarrow_{(1)} \ y$$

we add the new rule

$$(4)\ x^{-1} \cdot (x \cdot y) \ \longrightarrow \ y \ .$$

We continue to consider other critical pairs. For the case of group theory this completion process (according to Knuth and Bendix, cf. [3]) actually terminates and yields the following minimal rewrite rule system:

$$\begin{array}{rl}
(1) & 1 \cdot x \longrightarrow x, \\
(2) & x^{-1} \cdot x \longrightarrow 1, \\
(3) & (x \cdot y) \cdot z \longrightarrow x \cdot (y \cdot z), \\
(4) & x^{-1} \cdot (x \cdot y) \longrightarrow y, \\
(5) & x \cdot 1 \longrightarrow x, \\
(6) & 1^{-1} \longrightarrow 1, \\
(7) & (x^{-1})^{-1} \longrightarrow x, \\
(8) & x \cdot x^{-1} \longrightarrow 1, \\
(9) & x \cdot (x^{-1} \cdot y) \longrightarrow y, \\
(10) & (x \cdot y)^{-1} \longrightarrow y^{-1} \cdot x^{-1}.
\end{array}$$

So the equational theory of pure group theory can be decided by reduction modulo this RRS. Also for many other equationally definable algebraic structures there are canonical rewrite rule systems.

6 Conclusion

We have seen that several key algorithms in constructive algebra and logic actually are based on the same idea; namely the formation of critical pairs and the completion of a reduction relation. Recognition of these similarities might lead to a better understanding of algorithms and perhaps to new application areas. And mathematics can be seen as a more unified and interrelated field of knowledge.

Acknowledgement. This work has been supported by the Austrian Science Fund (FWF) under the project DIFFOP, No. P20336-N18.

References

1. Avenhaus, J.: Reduktionssysteme. Springer, Heidelberg (1995)
2. Book, R.V., Otto, R.: String-Rewriting Systems. Springer, Heidelberg (1993)
3. Knuth, D.E., Bendix, P.B.: Simple Word Problems in Universal Algebra. In: Leech, J. (ed.) Computational Problems in Abstract Algebra, pp. 263–297. Pergamon Press, Oxford (1970)
4. Winkler, F.: The Church–Rosser property in computer algebra and special theorem proving: An investigation of critical–pair/completion algorithms, Dissertation Univ. Linz, Austria, VWGÖ Wien (1984)
5. Winkler, F.: Polynomial Algorithms in Computer Algebra. Springer, Wien New York (1996)

Solving Norm Form Equations over Number Fields

Paraskevas Alvanos and Dimitrios Poulakis*

Aristotle University of Thessaloniki,
Department of Mathematics,
Thessaloniki 54124, Greece
{poulakis,paris14}@math.auth.gr

Abstract. Let K be a number field and L a finite extension of K of degree ℓ. Let $\omega_1 = 1, \omega_2, \dots, \omega_\ell$ be K-linearly independent integers of L and k an integer of K. We denote by $N_{L/K}$ the norm from L to K. In this paper we give an algorithm for the computation of algebraic integers, $x_1, \dots, x_\ell \in K$ satisfying the equation $N_{L/K}(\omega_1 x_1 + \dots + x_\ell \omega_\ell) = k$.

1 Introduction

One of the oldest problems studied in number theory is to find the integer solutions of equation

$$x^2 - dy^2 = 1 \tag{1}$$

known as Pell equation, for a given positive nonsquare integer d. If (x_1, y_1) is the least positive solution of (1) ordered by the value of $x_1 + y_1\sqrt{d}$, then all integer solutions of (1) are given by (x_n, y_n) for $n \in \mathbb{Z}$, where

$$x_n + y_n\sqrt{d} = \pm(x_1 + y_1\sqrt{d})^n.$$

Thus, the problem of solving the Pell equation is that of determining x_1 and y_1. Note that finding a solution of (1) comes down to finding a unit > 1 of the ring $\mathbb{Z}[\sqrt{d}]$ of norm 1, and so $x_1 + y_1\sqrt{d}$ is the smallest such unit. A bibliography on the Pell equation and on methods for solving it can be found in [4,15,30].

The regulator of the field $\mathbb{Q}(\sqrt{d})$ is defined to be the quantity $R_d = \log(x_1 + y_1\sqrt{d})$. The inequalities

$$\frac{exp(R_d)}{2} < x_1 < \frac{exp(R_d)}{2} + \frac{1}{4d}, \quad \frac{exp(R_d)}{2\sqrt{d}} - \frac{1}{4d} < y_1 < \frac{exp(R_d)}{2\sqrt{d}},$$

reduce the computation of the unit $x_1 + y_1\sqrt{d}$ to the computation of R_d. On the other hand, since the input size is $\log d$, we see that it is not possible to

* The research of the first author was financial supported by Hellenic State Scholarships Foundation-I.K.Y

have a polynomial-time algorithm for finding (x_1, y_1), because it may take exponentially many bits to represent. The best algorithm, without any assumption, for computing R_d, and hence the solutions of (1), has running time $O(d^{1/4+\epsilon})$, for every $\epsilon > 0$ [7]. Recently, in [12], an algorithm is proposed for unconditionally computing the regulator that runs in expected time $O(d^{1/6+\epsilon})$ assuming the Generalized Riemann Hypothesis. Under this hypothesis subexponential probabilistic algorithms are given in [1,29]. Note that the problem of solution of Pell's equation is in the complexity class SPP [2]. Furthermore, a polynomial time quantum algorithm is described in [13].

The equation (1) has an obvious generalization to the equation

$$x^2 - dy^2 = k, \tag{2}$$

where k is a nonzero integer. The integer solutions of (2) correspond to the elements of $\mathbb{Z}[\sqrt{d}]$ with norm equal to k. Thus, for solving (2) one has to find a maximal set of pairwise non associate elements of $\mathbb{Z}[\sqrt{d}]$ of norm equal to k and solve the corresponding equation (1). Some methods for the solution of (2) can be found in [16,17,18].

Equations (1) and (2) are used for the construction of pseudorandom number generators [14], elliptic curves suitable for cryptographic applications [9, Chapter 24], etc. Furthermore, some cryptographic applications of equation (1) over finite fields or the rings $\mathbb{Z}_N$ are given in [19,8].

Dirichlet [10] was the first who studied equation (1) in a quadratic field. More precisely, he solved (1) over the ring $\mathbb{Z}[i]$. Some results on the solutions of equation (1) and (2) with coefficients in the ring of integers of an arbitrary quadratic field were given in [20,21,11,26,27,28].

Let K be a number field with ring of integers O_K. In case where $d = -1$, Shastri [24] defined an operation on the set of integral solutions $(x, y) \in O_K^2$ of (1) such that it is an abelian group and determined its structure in terms of the number of complex embeddings of K. Next, in [25], he studied the set of solutions $(x, y) \in O_K^2$ of (2) in case where $d = -1$ and $k \in O_K \setminus \{0\}$. Recently, Schmid [23], generalized the results of [24] in case where $d \in O_K \setminus \{0\}$. He defined with the same way an operation on the set of integral solutions of (1) such that it is an abelian group and determined its structure in terms of the number of complex and real embeddings of K, and its number of positive embeddings of d.

Let L be a finite extension of K of degree ℓ and O_K the ring of integers of K. We denote by $N_{L/K}$ the norm from L to K. In this paper, we deal with the following more general computational problem: Given K-linearly independent elements $\omega_1 = 1, \omega_2, \dots, \omega_\ell \in O_L$ and $k \in O_K$, compute the elements $x_1, \dots, x_n \in O_K$ such that

$$N_{L/K}(x_1\omega_1 + \cdots + x_\ell\omega_\ell) = k.$$

In case where $\ell = 2$ and $\omega_2 = \sqrt{d}$, where d is a no square element of O_K, we have the generalized Pell equation (2).

The paper is organized as follows. In section 2 we describe the set of solutions of the above equation with $k = 1$. In sections 3 and 4 we give two algorithms for the computation of its solutions when $k = 1$ and $k \neq 1$, respectively. Finally, in Section 5, we solve some equations using these algorithms.

2 The Set of Solutions of $N_{L/K}(x_1\omega_1 + \cdots + x_\ell\omega_\ell) = 1$

In this section we describe the structure of the set of solutions $(x_1, \ldots, x_\ell) \in O_K^\ell$ of equation

$$N_{L/K}(x_1\omega_1 + \cdots + x_\ell\omega_\ell) = 1. \tag{3}$$

If R is an integral domain, then we denote by R^* its group of invertible elements. The norm of L over K yields the group epimorphism

$$N : O_K[\omega_2, \ldots, \omega_\ell]^* \longrightarrow O_K^*,\ x_1\omega_1 + \cdots + x_\ell\omega_\ell \longmapsto N_{L/K}(x_1\omega_1 + \cdots + x_\ell\omega_\ell).$$

Thus, $(x_1, \ldots, x_\ell) \in O_K^\ell$ is a solution of the above equation if and only if $x_1\omega_1 + \cdots + x_\ell\omega_\ell \in Ker(N)$. By the Dirichlet's Unit Theorem, the torsion subgroup of the unit group O_L^* is cyclic, and hence the torsion subgroup of $Ker(N)$ is also cyclic. Hence, for the determination of solutions of (3) is enough to calculate a basis of the free part of $Ker(N)$ and a generator for the torsion subgroup of $Ker(N)$.

We say that the ℓ-tuples $(x_{j1}, \ldots, x_{j\ell})$ $(j = 1, \ldots, \mu)$ form *a fundamental set of solutions* for (3), if the elements $x_{j1}\omega_1 + \cdots + x_{j\ell}\omega_\ell$ $(j = 1, \ldots, \mu)$ form a basis for the free part of $Ker(N)$. Furthermore, we say that $(\tau_1, \ldots, \tau_\ell)$ is a *torsion solution* for (3) if $\tau_1\omega_1 + \cdots + \tau_\ell\omega_\ell$ is a generator for the torsion subgroup of $Ker(N)$. In the following Proposition we compute the number of elements of a fundamental set of solutions for (3).

Proposition 1. *Let r_1 (respectively s_1) be the number of real embeddings and r_2 (respectively s_2) the number of conjugated pairs of complex embeddings of K (respectively L). Let F be a fundamental set of solutions for the equation (3). Then*

$$|F| = s_1 + s_2 - r_1 - r_2.$$

Proof. Set $[K : \mathbb{Q}] = n$. Let $\{\beta_1, \ldots, \beta_n\}$ be an integral basis of O_K. As the $n\ell$ elements $\beta_i\omega_j$ $(i = 1, \ldots, n,\ j = 1, \ldots, \ell)$ are $\mathbb{Q}$-linearly independent, $O_K[\omega_2, \ldots, \omega_\ell]$ is an order of L. By [23, Lemma 1.1], we have

$$\mathrm{rank}(O_K[\omega_2, \ldots, \omega_\ell]^*) = \mathrm{rank}(O_L^*) = s_1 + s_2 - 1.$$

On the other hand, we have

$$\mathrm{rank}(O_K[\omega_2, \ldots, \omega_\ell]^*) = \mathrm{rank}(O_K^*) + \mathrm{rank}(Ker(N)).$$

Therefore

$$|F| = \mathrm{rank}(Ker(N)) = s_1 + s_2 - r_1 - r_2.$$

3 Solving the Equation $N_{L/K}(x_1\omega_1 + \cdots + x_\ell\omega_\ell) = 1$

In this section we present an algorithm for the computation of integral solutions $(x_1, \ldots, x_\ell)$ of equation (3).

Algorithm 1

Input: K, L and $\omega_1 = 1, \omega_2, \ldots, \omega_\ell$ as above.
Output: A set of fundamental solutions F and a torsion solution t for (3).

1. Compute a basis of the group $O_K[\omega_2, \ldots, \omega_\ell]^*$, $\epsilon_i = \epsilon_{i,1}\omega_1 + \cdots + \epsilon_{i,\ell}\omega_\ell$ $(i = 1, \ldots, r)$, where $\epsilon_{i,j} \in O_K$.
2. Compute the elements $\lambda_i = N(\epsilon_i)$ $(i = 1, \ldots, r)$.
3. Compute a basis $\mathbf{z}_j = (z_{j,1}, \ldots, z_{j,r})$ $(j = 1, \ldots, s)$ for the lattice $\Lambda = \{(x_1, \ldots, x_r) \in \mathbb{Z}^r / \ \lambda_1^{x_1} \cdots \lambda_r^{x_r} = 1\}$.
4. Compute the elements $b_i = \epsilon_1^{z_{i,1}} \cdots \epsilon_r^{z_{i,r}}$ $(i = 1, \ldots, s)$ and write $b_i = b_{i,1}\omega_{i,1} + \cdots + b_{i,\ell}\omega_\ell$.
5. Compute a generator $\tau_1\omega_1 + \cdots + \tau_\ell\omega_\ell$ for the torsion subgroup of $Ker(N)$.
6. Output the set F of $(b_{i,1}, \ldots, b_{i,\ell})$ $(i = 1, \ldots, s)$ and $t = (\tau_1, \ldots, \tau_\ell)$.

Proof of correctness of Algorithm 1. The elements

$$b_j = \epsilon_1^{z_{j,1}} \cdots \epsilon_r^{z_{j,r}}, \quad (j = 1, \ldots, s)$$

generate $Ker(N)$. For if $u = \epsilon_1^{x_1} \cdots \epsilon_r^{x_r} \in Ker(N)$, then $\lambda_1^{x_1} \cdots \lambda_r^{x_r} = 1$. Thus, $(x_1, \ldots, x_r) = c_1\mathbf{z}_1 + \cdots + c_s\mathbf{z}_s$, where $c_1, \ldots, c_s \in \mathbb{Z}$, and so,

$$u = b_1^{c_1} \cdots b_s^{c_s}.$$

Next, suppose there are integers $a_1, \ldots, a_s$ such that

$$b_1^{a_1} \cdots b_s^{a_s} = 1.$$

Since $\epsilon_1, \ldots, \epsilon_r$ is multiplicatively independent, we get

$$a_1 z_{1,i} + \cdots + a_s z_{s,i} = 0 \quad (i = 1, \ldots, r).$$

Thus, we have

$$a_1\mathbf{z}_1 + \cdots + a_s\mathbf{z}_s = 0$$

and, since $\mathbf{z}_1, \ldots, \mathbf{z}_s$ form a basis for Λ, we deduce $a_1 = \cdots = a_s = 0$, and so $b_1, \ldots, b_s$ are multiplicatevely independent. Hence $b_1, \ldots, b_s$ is a basis for the free part of $Ker(N)$.

Step 1 and Step 5 can be achieved by algorithms implemented in the computational algebraic system KASH [31] and MAGMA [32]. A description of these methods can be found in [22]. Furthermore, note that the algorithm of [6] for this task has running time $O(RD^\epsilon)$ (for every $\epsilon > 0$), where R and D is the regulator and the absolute value of the discriminant of the order $O_K[\omega_2, \ldots, \omega_\ell]$, and the O-constant depends only on the degree of L. Step 2 can be achieved by the system KASH [31] or MAGMA [32] and the time needed is polynomial in the degree of L. For the Step 3 is more convenient to use an algorithm implemented in computational algebraic system GAP [33]. A description of this method is contained in [3, Section 6]. An estimate for the size of a basis of Λ is given in [5, Corollary, page 207], once the size of ϵ_i is known. Finally, Step 4 is easily carried out using KASH [31] or MAGMA [32].

4 Solving the Equation $N_{L/K}(x_1\omega_1 + \cdots + x_\ell\omega_\ell) = k$

In this section, we propose an algorithm for the determination of solutions $(x_1, \ldots, x_\ell) \in O_K^\ell$ of equation

$$N_{L/K}(x_1\omega_1 + \cdots + x_\ell\omega_\ell) = k, \tag{4}$$

with $k \neq 1$. Suppose that $w_1, \ldots, w_t$ form a maximal set of pairwise non associate elements of $O_K[\omega_2, \ldots, \omega_\ell]$ sush that $N_{L/K}(w_i) = k$, $(i = 1, \ldots, m)$. If $(x_1, \ldots, x_\ell) \in O_K^\ell$ is a solution of the above equation, then there is $i \in \{1, \ldots, t\}$ and a unit $\eta = \eta_1\omega_1 + \cdots + \eta_\ell\omega_\ell$ of $O_K[\omega_2, \ldots, \omega_\ell]$ such that we have

$$x_1\omega_1 + \cdots + x_\ell\omega_\ell = \eta w_i$$

and $(\eta_1, \ldots, \eta_\ell)$ is a solution of equation (3). Conversely, we see that all the ℓ-tuples $(x_1, \ldots, x_\ell) \in O_K^\ell$ of the previous form are solutions of (4).

Algorithm 2

Input: K, L, k and $\omega_1, \ldots, \omega_\ell$ as above.
Output: "No solution" if (4) does not have any solution in O_K. Else, a set F of fundamental solutions and a torsion solution t for the associated equation (3), and a maximal set M of pairwise non associate elements $w \in O_K[\omega_2, \ldots, \omega_\ell]$ with $N_{L/K}(w) = k$.

1. Determine a maximal set $M \subseteq O_K[\omega_2, \ldots, \omega_\ell]$ such that its elements w are pairwise non associate with $N_{L/K}(w) = k$. If a such set does not exist, then output "No solution". Else, go to the following step.
2. Using Algorithm 1, compute a set F of fundamental solutions and a torsion solution t for the associated equation (3).
3. Output the elements obtained in Steps 1 and 2.

Step 1 of the above algorithm can be achieved by algorithms implemented in the systems KASH and MAGMA.

5 Examples

In this section, using Algorithms 1 and 2, we solve four norm form equations. In order to achieve the different tasks for this purpose we use the computational algebraic systems KASH, MAGMA and GAP, as we have explained in the previous sections.

Example 1. *Let $K = \mathbb{Q}(\omega)$, where ω is a root of the polynomial $T^4 + 3T^2 - 2T - 5$. The solutions of the norm form equation*

$$N_{K/\mathbb{Q}}(x_0 + x_1\omega + x_2\omega^2 + x_3\omega^3) = 16$$

are the quadruples $(x_0, x_1, x_2, x_3) \in \mathbb{Z}^4$ such that

$$x_0 + x_1\omega + x_2\omega^2 + x_3\omega^3 = \pm(\omega + 1)^z(-4\omega^3 - 19\omega^2 + 12\omega + 24)^w \Omega,$$

where $z, w \in \mathbb{Z}$ and $\Omega \in \{2, -\omega^3 - 2\omega^2 + 2\omega + 3, -12\omega^3 + 7\omega^2 + 13\omega - 3\}$.

Proof. The torsion part of the unit group of $\mathbb{Z}[\omega]^*$ is generated by -1 and a basis of the infinite part is given by the elements $\omega + 1$ and $-2\omega^2 + \omega + 2$.

Let $N : \mathbb{Z}[\omega]^* \longrightarrow \{-1, 1\}$ be by the restriction of the norm $N_{K/\mathbb{Q}}$ on $\mathbb{Z}[\omega]^*$. The images of -1, $\omega + 1$ and $-2\omega^2 + \omega + 2$ through the map N are $1, 1, -1$, respectively. It follows that the torsion part of $Ker(N)$ is generated by -1 and a basis of the infinite part is given by the elements $\omega+1$ and $-4\omega^3-19\omega^2+12\omega+24$.

Finally, a maximal set of pairwise non associate algebraic integers of $\mathbb{Z}[\omega]$ with norm equal to 16 is given by the numbers

$$2, \quad -\omega^3 - 2\omega^2 + 2\omega + 3, \quad -12\omega^3 + 7\omega^2 + 13\omega - 3.$$

The result follows.

Example 2. *The solutions of the equation*

$$x^2 - 7y^2 = -17 + 4\sqrt{6}$$

in $\mathbb{Z}[\sqrt{6}]$ are the couples (x, y) such that

$$x + y\sqrt{7} = \pm(8 + 3\sqrt{7})^s(13 + 2\sqrt{6}\sqrt{7})^t(\alpha + \beta\sqrt{7}),$$

where $s, t \in \mathbb{Z}$ and $(\alpha, \beta) = (1 + 2\sqrt{6}, \sqrt{6}), (1 + 2\sqrt{6}, -\sqrt{6})$.

Proof. Put $K = \mathbb{Q}(\sqrt{6})$ and $L = \mathbb{Q}(\sqrt{6}, \sqrt{7})$. Thus the above equation is the norm form equation

$$N_{L/K}(x + y\sqrt{7}) = -17 + 4\sqrt{6}.$$

The ring of integers of K is $O_K = \mathbb{Z}[\sqrt{6}]$. The torsion part of the group $O_K[\sqrt{7}]^*$ is generated by $u_0 = -1$ and a basis for the infinite part of $O_K[\sqrt{7}]^*$ is given by the elements

$$u_1 = \sqrt{6} + \sqrt{7}, \quad u_2 = 5 - 2\sqrt{6}, \quad u_3 = 8 + 3\sqrt{7}.$$

We consider the map

$$N : O_K[\sqrt{7}]^* \longrightarrow O_K^*, \; x + y\sqrt{7} \longmapsto x^2 - 7y^2.$$

Put $\lambda_i = N(u_i)$ $(i = 0, 1, 2, 3)$. We have

$$\lambda_0 = 1, \quad \lambda_1 = -1, \quad \lambda_2 = 49 - 20\sqrt{6}, \quad \lambda_3 = 1.$$

A basis for the lattice

$$\Lambda = \{(x_1, x_2, x_3) \in \mathbb{Z}^3 / \ \lambda_1^{x_1}\lambda_2^{x_2}\lambda_3^{x_3} = 1\}$$

is given by the vectors $(2, 0, 0)$ and $(0, 0, 1)$. Thus, a basis for the infinite part of $Ker(N)$ is formed by the elements

$$b_1 = u_1^2 = 13 + 2\sqrt{6}\sqrt{7}, \quad b_2 = u_3 = 8 + 3\sqrt{7}$$

and so, a set of fundamental solutions for the equation $X^2 - 7Y^2 = 1$ is given by the couples $(13, 2\sqrt{6})$ and $(8, 3)$ and the torsion solution is $(-1, 0)$.

A maximal set of pairwise non associate elements of $O_K[\sqrt{7}]$ with norm equal to $-17 + 4\sqrt{6}$ is given by the elements $1 + 2\sqrt{6} \pm \sqrt{6}\sqrt{7}$. The results follows.

Example 3. *Let a be a root of polynomial $T^5 - 3T^2 + 1$. The solutions of the equation*

$$x^2 - (1 - a^2)y^2 = 2a^2 + a^3 - a^4$$

in integers of $\mathbb{Q}(a)$ are the couples (x, y) such that

$$x + y\sqrt{1 - a^2} = \pm \prod_{i=1}^{3} (\gamma_i + \delta_i\sqrt{1 - a^2})^{z_i} (\pm\kappa + \lambda\sqrt{1 - a^2}),$$

where $z_1, z_2, z_3 \in \mathbb{Z}$ and

$$\begin{aligned} \gamma_1 &= -29 - 62a + 8a^2 + 10a^3 + 22a^4, & \delta_1 &= -46 - 60a + 8a^2 + 16a^3 + 22a^4, \\ \gamma_2 &= 7 - 6a^2 + 2a^3, & \delta_2 &= 2 + 10a - 4a^2 + 2a^3 - 4a^4, \\ \gamma_3 &= -3a + a^4, & \delta_3 &= 3a - a^4, \\ \kappa &= 10 + 18a - 2a^2 - 4a^3 - 6a^4, & \lambda &= -13 + 22a - 3a^2 - 5a^3 - 8a^4. \end{aligned}$$

Proof. Put $K = \mathbb{Q}(a)$ and $L = K(\sqrt{d})$, where $d = 1 - a^2$. Thus the above equation is the norm form equation

$$N_{L/K}(x + y\sqrt{d}) = 2a^2 + a^3 - a^4.$$

The elements $1, a, a^2, a^3, a^4$ form an basis for the ring of integers O_K of K. The torsion part of the group $O_K[\sqrt{d}]^*$ is generated by $u_0 = -1$ and a basis for the infinite part of $O_K[\sqrt{d}]^*$ is given by the elements

$$\begin{aligned} u_1 &= -a, & u_4 &= 1 - 2a + a^2 + a^4, \\ u_2 &= 1 - a, & u_5 &= 1 + (-1 - 2a + a^2 + a^3 + a^4)\sqrt{d}, \\ u_3 &= -3a + a^4 + (3a - a^4)\sqrt{d}, & u_6 &= -2a + a^2 - a^3 + a^4 + (2 - a)\sqrt{d}. \end{aligned}$$

We consider the restriction $N : O_K[\sqrt{d}]^* \longrightarrow O_K^*$ of the norm $N_{L/K}$ on $O_K[\sqrt{d}]^*$. Put $\lambda_i = N(u_i)$ $(i = 0, \ldots, 6)$. We have $\lambda_0 = 1$ and

$$\begin{aligned} \lambda_1 &= a^2, & \lambda_4 &= 3a^4 + a^3 + 3a^2 - 6a + 2, \\ \lambda_2 &= a^2 - 2a + 1, & \lambda_5 &= 2a^4 + 2a^3 + 3a^2 - 2a - 1, \\ \lambda_3 &= 1, & \lambda_6 &= a - 1. \end{aligned}$$

A basis for the lattice

$$\Lambda = \{(x_1,\dots,x_6) \in \mathbb{Z}^6 / \ \lambda_1^{x_1}\cdots\lambda_6^{x_6} = 1\}$$

is given by the vectors

$$(1,0,0,1,-2,-2), \quad (0,1,0,0,0,-2), \quad (0,0,1,0,0,0).$$

We compute

$$\begin{aligned} b_1 &= u_1u_4u_5^{-2}u_6^{-2} = \gamma_1 + \delta_1\sqrt{d}, \\ b_2 &= u_2u_6^{-2} = \gamma_2 + \delta_2\sqrt{d}, \\ b_3 &= u_3 = \gamma_3 + \delta_3\sqrt{d}, \end{aligned}$$

where γ_i, δ_i $(i = 1,2,3)$ are given in the statement above. Hence, a set of fundamental solutions for the equation $X^2 - dY^2 = 1$ is given by the couples (γ_i, δ_i) $(i = 1,2,3)$ and the torsion solution is $(-1,0)$.

Finally, the elements $\pm\kappa + \lambda\sqrt{d}$, where the quantities κ and λ are given in the statement above, form a maximal set of pairwise non associate algebraic integers of $K(\sqrt{d})$ with norm equal to $2a^2 + a^3 - a^4$. The result follows.

Example 4. *Let a be a root of polynomial $T^3 - T + 1$ and $= \mathbb{Q}(a)$. Let b be a root of polynomial $T^3 - (1-a+2a^2)T^2 - a$, $L = K(b)$ and $\omega = -1 + a + (1-a+2a^2)b^2$. Then the solutions $(x_1,x_2,x_3) \in \mathbb{Z}[a]$ of the equation*

$$N_{L/K}(x_1 + x_2\omega + x_3\omega^2) = a^2 - a - 3$$

are given by

$$x_1 + x_2\omega + x_3\omega^2 = \prod_{i=1}^{4}(\mu_i + \nu_i\omega + \xi_i\omega^2)^{z_i}\Omega,$$

where $z_i \in \mathbb{Z}$ $(i = 1,2,3,4)$,

$$\begin{aligned} &\mu_1 = -a^2 + 1, \ \mu_2 = -35a^2 - 7a + 40, \\ &\nu_1 = -a^2 + 1, \ \nu_2 = -91a^2 - 59a + 67, \\ &\xi_1 = 0, \qquad\quad \xi_2 = -83a^2 - 70a + 532, \end{aligned}$$

$$\begin{aligned} &\mu_3 = 169163a^2 - 210187a - 577987, \quad \mu_4 = -1429a^2 + 1778a - 875, \\ &\nu_3 = -192789a^2 - 635482a - 506567, \nu_4 = -1743a^2 + 1865a - 989, \\ &\xi_3 = -254575a^2 - 253327a + 111177, \xi_4 = -71a^2 - 29a + 120, \end{aligned}$$

and

$$\Omega = -1654a^2 + 2178a - 1223 + (-1754a^2 + 2269a - 1280)\omega + (-13a + 21)\omega^2.$$

Proof. Let $\mathcal{O}$ be the order generated over $\mathbb{Z}$ by the elements $a^i\omega^j$, where $i, j \in \{0,1,2\}$. The torsion part of the group $\mathcal{O}^*$ is generated by $u_0 = -1$ and a basis for the infinite part of $\mathcal{O}^*$ is given by the elements

$$\begin{aligned}
u_1 &= -a \\
u_2 &= 1 + (2a - a^2)b^2, \\
u_3 &= -87 + 85a + 115a^2 - (15 - 80a - 67a^2)b + (-108 + 134a + 163a^2)b^2, \\
u_4 &= 217 - 251a - 306a^2 - (366 + 25a - 174a^2)b + (396 + 32a - 199a^2)b^2, \\
u_5 &= 177 - 176a + 165a^2 - (376 - 527a + 257a^2)b + (40 - 125a + 21a^2)b^2.
\end{aligned}$$

Let $N : \mathcal{O}^* \longrightarrow O_K^*$ be the map defined by the restriction of the norm $N_{L/K}$ on $\mathcal{O}^*$. Put $\lambda_i = N(u_i)$ $(i = 0, \dots, 5)$. We have

$$\lambda_0 = \lambda_4 = -1, \quad \lambda_1 = -a + 1, \quad \lambda_2 = \lambda_3 = \lambda_5 = 1.$$

We obtain that a basis for the lattice

$$\Lambda = \{(x_1, \dots, x_5) \in \mathbb{Z}^5 / \ \lambda_1^{x_1} \cdots \lambda_5^{x_5} = 1\}$$

is given by the vectors

$$(0,1,0,0,0), \quad (0,0,1,0,0), \quad (0,0,0,2,0), \quad (0,0,0,0,1).$$

Thus, a basis of the free part of the group $Ker(N)$ is given by the elements $w_1 = u_2, w_2 = u_3, w_3 = u_4^2, w_4 = u_5$ and the torsion part of $Ker(N)$ is trivial. Since

$$\begin{aligned}
b &= a - 1 + (2a^2 - a + 1)\omega^2, \\
b^2 &= -61a^2 + 79a - 46 + (-10a^2 + 14a - 9)\omega + (265a^2 - 353a + 199)\omega^2,
\end{aligned}$$

we obtain $w_i = \mu_i + \nu_i\omega + \xi_i\omega^2$ $(i = 1, 2, 3, 4)$, where the values of μ_i, ν_i, ξ_i are given above.

Finally, we see that every algebraic integer of $\mathcal{O}$ with norm equal to $a^2 - a - 3$ is associated to

$$-1654a^2 + 2178a - 1223 + (-1754a^2 + 2269a - 1280)\omega + (-13a + 21)\omega^2.$$

The result follows.

Acknowledgements. The computations were carried out with the Transnational Access Programme at RISC, Johannes Kepler University Linz, supported by the European Commission Framework 6 Programme for Integrated Infrastructures Initiatives under the project SCIEnce (contract No 026133). The authors are duly greatful to RISC for providing access to this programme and to the referees for theirs helpful suggestions and comments.

References

1. Abel, C.S.: Ein Algorithmus zur Berechnung der Klassenzahl und des Regulators reell quadratischer Ordnungen. Ph.D. Thesis, Universität des Saarlandes, Saarbrücken, Germany (1994)

2. Arvind, V., Kurur, P.P.: On the complexity of computing units in a number field. In: Buell, D.A. (ed.) ANTS 2004. LNCS, vol. 3076, pp. 72–86. Springer, Heidelberg (2004)
3. Assmann, B., Eick, B.: Computing polycyclic presentations for polycyclic rational matrix groups. J. Symbolic Comput. 40(6), 1269–1284 (2005)
4. Barbeau, E.J.: Pell's Equation. Springer, Heidelberg (2003)
5. Bertrand, D.: Dyality on tori and dependence relations. J. Austral. Math. Soc (Series A) 62, 198–216 (1997)
6. Buchmann, J.: On the Computation of Units and Class Numbers by a Generalization of Langrange's Algorithm. J. Number Theory 26, 8–30 (1987)
7. Buchmann, J., Williams, H.C.: On the Infrastructure of the Principal Ideal Class of an Algebraic Number Field of Unit Rank One. Mathematics of Computation 50(182), 569–579 (1988)
8. Chen, C.-Y., Chang, C.-C., Yang, W.-P.: Fast RSA-type Schemes Based on Pell Equations over $\mathbb{Z}_N$. In: Joint Conference of, International Computer Symposium, Kaohsiung, Taiwan, R.O.C, December 19-21 (1996)
9. Cohen, H., Frey, G.: Handbook of Elliptic and Hyperelliptic Cryptography. Chapman and Hall/CRC (2006)
10. Dirichlet, Recherches sur les formes quadratiques à coefficients et à indeterminées complexes, J. Reine Angew. Math. 24, 291-371 = Werke I, 535–618 (1842)
11. Fjellstedt, L.: On a class of Diophantine equations of the second degree in imaginary quadratic fields. Arkiv. Mat. 2, 435–461 (1952-1954)
12. de Haan, R., Jacobson Jr., M.J., Williams, H.C.: A fast, rigorous technique for computing the regulator of a real quadratic field. Math. Comp. 76(260), 2139–2160 (2007)
13. Hallgren, S.: Polynomial-Time Quantum Algorithms for Pell's Equation and the Principal Ideal Problem. Journal of ACM 54(1), Art. 4, 19 (2007) (electronic)
14. Huber, K.: On the Period Length of Generalized Inverse Pseudorandom Generators. AAECC 5, 255–260 (1994)
15. Lenstra Jr., H.W.: Solving the Pell equation. Notices Amer. Math. Soc. 49, 182–192 (2002)
16. Matthews, K.: The Diophantine Equation $x^2 - Dy^2 = N$, $D > 0$. Expos. Math. 18, 323–331 (2000)
17. Mollin, R.A.: Fundamental Number Theory with Applications. CRC Press, Boca Raton (1998)
18. Mollin, R.A.: Simple Continued Fraction Solutions for Diophantine Equations. Expos. Math. 19, 55–73 (2001)
19. Murthy, N.R., Swamy, M.N.S.: Cryptographic Applications of Brahmagupta-Bhãskara Equation. IEEE Transactions on Circuits and Systems 53(7), 1565–1571 (2006)
20. Niven, I.: Quadratic Diophantine Equations in the Rational and Quadratic Fields. Tras. of Amer. Math. Soc. 52(1), 1–11 (1942)
21. Niven, I.: The Pell equation in quadratic fields. Bull. Amer. Math. Soc. 49, 413–416 (1943)
22. Pohst, M.: Computational Algebraic Number Theory. DMV Seminar Band 21. Birkhauser Verlag, Basel (1993)
23. Schmid, W.A.: On the set of integral solutions of the Pell equation in number fields. Aequationes Math 71, 109–114 (2006)
24. Shastri, P.: Integral points on the unit circle. J. Number Theory 91(1), 67–70 (2001)
25. Shastri, P.: Integral points on the circle $X^2 + Y^2 = c$. Currents trends in number theory (Allahabad, 2000), 175–184. Hindustan Book Agency, New Delhi (2002)

26. Skolem, T.: A theorem on the equation $\zeta^2 - \delta\eta^2 = 1$, where δ, ζ, η are integers in an imaginary quadratic field. Avh. Norske Akad. Oslo 1, 1–13 (1945)
27. Skolem, T.: A remark on the equation $\zeta^2 - \delta\eta^2 = 1$, $\delta > 0, \delta', \delta, \dots < 0$ where δ, ζ, η belong to a total real number field. Avh. Norske Vid. Akad. Oslo. I(12), 15 (1945)
28. Val′fiš, A.Z.: Elementary solution of Pell's equation (Russian). Akad. Nauk Gruzin. SSR. Trudy Mat. Inst. Razmadze 18, 116–132 (1951)
29. Vollmer, U.: Asymptotically fast discrete logarithms in quadratic number fields. In: Bosma, W. (ed.) ANTS 2000. LNCS, vol. 1838, pp. 581–594. Springer, Heidelberg (2000)
30. Williams, H.C.: Solving the Pell Equation, Number theory for the millennium, III, Urbana, IL, A K Peters, Natick, MA, pp. 397–435 (2002)
31. http://www.math.tu-berlin.de/~kant/
32. http://magma.maths.usyd.edu.au/magma/
33. http://www.gap-system.org/

A Note on Unambiguity, Finite Ambiguity and Complementation in Recognizable Two-Dimensional Languages*

Marcella Anselmo[1] and Maria Madonia[2]

[1] Dipartimento di Informatica ed Applicazioni,
Università di Salerno I-84084 Fisciano (SA) Italy
anselmo@dia.unisa.it

[2] Dip. Matematica e Informatica, Università di Catania,
Viale Andrea Doria 6/a, 95125 Catania, Italy
madonia@dmi.unict.it

Abstract. The paper deals with some open questions related to unambiguity, finite ambiguity and complementation of two-dimensional recognizable languages. We give partial answers based on the introduction of special classes of languages of "high complexity", in a sense specified in the paper and motivated by some necessary conditions holding for recognizable and unambiguous languages. In the last part of the paper we also show a new necessary condition for recognizable two-dimensional languages on unary alphabet.

Keywords: Automata and Formal Languages, Unambiguity, Complement, Two-dimensional languages.

1 Introduction

The theory of formal languages of strings is well founded and the research in this framework continues from many decades. The increasing interest for pattern recognition and image processing has more recently motivated the research on languages of pictures or two-dimensional languages, and nowadays this is a research field of great interest. Since the sixties, many approaches have been presented in the literature in order to find in two dimensions a counterpart of notions and results of the one-dimensional languages theory: finite automata, grammars, logics and regular expressions. In 1991, an unifying point of view was presented by A. Restivo and D. Giammarresi who defined the family REC of *recognizable picture languages* (see [11] and [12]), as an equivalent of the class of recognizable (or regular) string languages. This definition takes as starting point a characterization of recognizable string languages in terms of local languages and projections (cf. [10]): the pair of a local picture language and a projection

* This work was partially supported by MIUR Project *"Automi e Linguaggi Formali: aspetti matematici e applicativi"* (2005), by ESF Project *"AutoMathA"* (2005-2010), by 60% Projects of University of Catania and Salerno (2007, 2008).

is called *tiling system*. Tiling systems have also analogies with the tiling of the infinite plane.

REC family inherits several properties from the class of regular string languages. A crucial difference lies in the fact that REC family is not closed under complementation: there are languages in REC whose complement is not in REC [12]. It is then important to take into account also the class co-REC of languages whose complement is in REC. The strict inclusion REC$\subset$ (REC$\cup$ co-REC) holds even in the unary case [19] and it fits the fact that the definition of recognizability by tiling systems is intrinsically non-deterministic. The notion of determinism on tiling systems is discussed in [2].

The non-closure of REC under complementation motivated the definition of *unambiguous two-dimensional languages*, whose family is denoted UREC [11]. Informally, a picture language belongs to UREC when it admits an unambiguous tiling system, that is if every picture has a unique pre-image in its corresponding local language. In [3], the proper inclusion of UREC in REC is proved; it holds true in the unary case too (see [6]). In other words there exist in REC inherently ambiguous languages. An open question is whether UREC is closed under complementation or not. Its answer depends on the following open problem.

Question 1. Does $L \in$ REC and $\overline{L} \notin$ REC imply that $L \notin$ UREC?

Question 1 was firstly stated in [22]. The converse is actually an open question too: Does $L \in$ REC $\setminus$ UREC imply that $\overline{L} \notin$ REC? Note that positive answers to both Question 1 and the converse mean that UREC=REC $\cap$ co-REC and that UREC is the largest subset of REC closed under complementation. Also note that such question is related to some difficult problems on complexity classes [7].

All the inherently ambiguous languages known in the literature are indeed *infinitely ambiguous*, in the sense that it is not possible to recognize them by a tiling system, in such a way that each picture has a fixed number of pre-images at most (see Section 2 for more details). The question whether this is always the case or not is open. Let us state it as follows.

Question 2. Does there exist a language $L \in$ REC $\setminus$ UREC such that L is finitely ambiguous?

In this paper we will answer Questions 1 and 2 in some particular cases, where languages involved have "high complexity", as specified in the following. We will introduce a class *HP* $\subseteq$ co-REC $\setminus$ REC of not-recognizable languages and a class *HK* $\subseteq$ REC $\setminus$ UREC of ambiguous languages, whose languages are "hard" with respect to some complexity functions. We will show that:

1. If $L \in$ REC and $\overline{L} \in$ HP then $L \notin$ UREC.
2. If $L \in$ REC and $L \in$ HK then L is infinitely ambiguous.

Let us emphasize that at present it is not known whether the inclusions HP$\subseteq$ co-REC $\setminus$ REC and HK$\subseteq$ REC $\setminus$ UREC are strict or not. No example (nor a candidate) exists showing the inclusions are strict. Hence in the case HP= co-REC $\setminus$ REC and/or HK=REC $\setminus$ UREC, our results would be a positive answer to Questions 1 and/or 2, in their general setting.

The introduction of classes HP and HK is motivated by some necessary conditions for languages in REC and in UREC, respectively, stating that: if $L \in$REC then the size of some permutation matrices associated to L cannot grow so quickly; and if $L \in$UREC then the rank of some matrices associated to L cannot grow so quickly. Indeed here "HP" stands for "High Permutation matrix" and "HK" stands for "High ranK". All the examples of languages that witness the strict inclusions UREC$\subset$ REC$\subset$ (REC $\cup$ co-REC) have been provided applying the necessary conditions we have just mentioned. The main difficulty in this framework is that there are no characterizations of REC and UREC, that could be easily and fruitfully applied, while we do not know whether the mentioned necessary conditions are also sufficient. A main question is thus the following.

Question 3. Find characterizations of meaningful classes of two-dimensional languages.

An intermediate step in view of solving Question 3 is to look for necessary conditions as tight as possible. In the last part of the paper we will introduce a new necessary condition for the belonging of a picture language to REC, in the case when the alphabet is unary. We will then compare in an example the obtained bound with the other ones known in the literature. Remark that the case when the alphabet has a single letter means studying the shapes of pictures before their contents. This is not a simpler subcase: all separation results known for the general case also holds in the unary case. See [1,5,6,7,18] for recent papers on unary two-dimensional languages.

Let us give some more details on the ideas on which the mentioned necessary conditions are based, since our results will be basically related to them.

In 1998 O. Matz [18] isolated a technique for showing that a language is not recognizable. It consists in considering for any recognizable picture language L and integer m the string language $L(m)$ of all pictures in L of fixed height m. Then if $L \in$ REC it is possible to associate to any tiling system recognizing L a family $\{A_m\}$, where each A_m is an automaton accepting $L(m)$ with c^m states at most, for some constant c. Using some known lower bound on the size of an automaton, he proved a necessary condition for the belonging of a picture language to REC (based on the cardinality of a set of pairs of pictures).

In [3] Matz's technique was firstly used together with some lower bound on the size of unambiguous string automata based on the Hankel matrices of the string languages $L(m)$. Recently in [14] the idea of finding necessary conditions for picture languages by studying the Hankel matrices of $L(m)$ has been considered by rephrasing Matz's necessary condition (for belonging to REC) and Cervelle's necessary condition (for belonging to REC$\cup$co-REC) (see [8]) in terms of parameters of the Hankel matrices.

The paper is organized as follows. After giving the basic definitions and results on two-dimensional languages in Section 2, in Section 3 we recall some necessary conditions for two-dimensional languages and introduce the classes HP and HK. Section 4 contains the main results concerning Questions 1 and 2, while the new necessary condition for the unary case is in Section 5.

2 Preliminaries

In this section we recall some definitions about two-dimensional recognizable languages. More details can be mainly found in [12].

A *two-dimensional string* (or a *picture*) over a finite alphabet Σ is a two-dimensional rectangular array of elements of Σ. The set of all pictures over Σ is denoted by Σ^{**} and a *two-dimensional language* over Σ is a subset of Σ^{**}.

Given a picture $p \in \Sigma^{**}$, let $p_{(i,j)}$ denote the symbol in p with coordinates (i,j), $\ell_1(p) = m$, the number of rows and $\ell_2(p) = n$ the number of columns; the pair (m,n) is the *size* of p. Note that when a one-letter alphabet is concerned, a picture p is totally defined by its size (m,n), and we will write $p = (m,n)$. Remark that the set Σ^{**} includes also all the empty pictures, i.e. all pictures of size $(m,0)$ or $(0,n)$ for all $m,n \geq 0$. It will be needed to identify the symbols on the boundary of a given picture: for any picture p of size (m,n), we consider the *bordered picture* $\widehat{p}$ of size $(m+2,n+2)$ obtained by surrounding p with a special *boundary symbol* $\# \notin \Sigma$.

A *tile* is a picture of size $(2,2)$ and $B_{2,2}(p)$ is the set of all sub-blocks of size $(2,2)$ of a picture p. Given an alphabet Γ, a two-dimensional language $L \subseteq \Gamma^{**}$ is *local* if there exists a finite set Θ of tiles over $\Gamma \cup \{\#\}$ such that $L = \{p \in \Gamma^{**} | B_{2,2}(\widehat{p}) \subseteq \Theta\}$ and we will write $L = L(\Theta)$.

A *tiling system* is a quadruple $(\Sigma, \Gamma, \Theta, \pi)$ where Σ and Γ are finite alphabets, Θ is a finite set of tiles over $\Gamma \cup \{\#\}$ and $\pi : \Gamma \to \Sigma$ is a projection. A two-dimensional language $L \subseteq \Sigma^{**}$ is *tiling recognizable* if there exists a tiling system $(\Sigma, \Gamma, \Theta, \pi)$ such that $L = \pi(L(\Theta))$ (extending π in the usual way). For any $p \in L$, a local picture $p' \in L(\Theta)$, such that $p = \pi(p')$, is called a *pre-image* of p. We denote by *REC* the family of all *tiling recognizable* picture languages.

The family REC is closed with respect to different types of operations. The *column concatenation* of p and q (denoted by $p \obar q$) and the *row concatenation* of p and q (denoted by $p \ominus q$) are partial operations, defined only if $\ell_1(p) = \ell_1(q)$ and if $\ell_2(p) = \ell_2(q)$, respectively and are given by:

$$p \obar q = \begin{array}{|c|c|}\hline \quad p \quad & \quad q \quad \\ \hline\end{array} \qquad\qquad p \ominus q = \begin{array}{|c|}\hline \quad p \quad \\ \hline q \\ \hline\end{array}.$$

REC family is closed under row and column concatenation and their closures, under union, intersection and under rotation (see [12] for all the proofs).

Let us give some examples to which we will refer later.

Example 1. Let $L_{fc=lc}$ be the language of pictures over $\Sigma = \{a,b\}$, with more than one column, whose first column is equal to the last one. Language $L_{fc=lc} \in$ REC. Informally we can define a local language where information about first column symbols of a picture p is brought along horizontal direction, by means of subscripts, to match the last column of p. Tiles are defined to have always same subscripts within a row while, in left and right border tiles, subscripts and main symbols should match. Below it is an example of a picture $p \in L_{fc=lc}$ together with a pre-image p' of p.

$$p = \begin{array}{|c|c|c|c|c|}\hline b & b & a & b & b \\ \hline a & a & b & a & a \\ \hline b & a & a & a & b \\ \hline a & b & b & b & a \\ \hline \end{array} \qquad p' = \begin{array}{|c|c|c|c|c|}\hline b_b & b_b & a_b & b_b & b_b \\ \hline a_a & a_a & b_a & a_a & a_a \\ \hline b_b & a_b & a_b & a_b & b_b \\ \hline a_a & b_a & b_a & b_a & a_a \\ \hline \end{array}.$$

Let $L_{fc=c'}$ be the language of pictures such that the first column is equal to some i-th column, $i \neq 1$. Note that $L_{fc=c'} = L_{fc=lc} \obar \Sigma^{**}$ and thus $L_{fc=c'} \in$ REC. Similarly we can show that the languages $L_{c'=lc} = \Sigma^{**} \obar L_{fc=lc}$, and $L_{c=c'} = \Sigma^{**} \obar L_{fc=lc} \obar \Sigma^{**}$ are in REC.

Example 2. Consider the language $CORNERS$ of all pictures p over $\Sigma = \{a, b\}$ such that whenever $p_{(i,j)} = p_{(i',j)} = p_{(i,j')} = b$ then also $p_{(i',j')} = b$. Intuitively, whenever three corners of a rectangle carry a b, then also the fourth one does. In [18], it is shown that $CORNERS \notin REC$. Consider now, the language $L = \overline{CORNERS}$. We have $L \in REC$; indeed, we can set

$L_1 = \Sigma^{**} \ominus (\Sigma^{**} \obar (\boxed{b} \ominus \Sigma^{**} \ominus \boxed{b}) \obar \Sigma^{**} \obar (\boxed{b} \ominus \Sigma^{**} \ominus \boxed{a}) \obar \Sigma^{**}) \ominus \Sigma^{**}$, and then L is equal to the union of L_1 with the languages obtained by its 90°, 180° and 270° rotations.

A recognizable two-dimensional language $L \subseteq \Sigma^{**}$ is *unambiguous* if and only if it admits an unambiguous tiling system $\mathcal{T}$; a tiling system $\mathcal{T} = (\Sigma, \Gamma, \Theta, \pi)$ is *unambiguous* for L if and only if any picture $p \in L$ has an unique pre-image in the local language $L(\Theta)$ (see [11]). The family of all unambiguous recognizable two-dimensional languages is denoted by *UREC*. In [3] it is proved that the inclusion of UREC in REC is strict and in [6] that this strict inclusion holds even if the alphabet is unary. Therefore in REC there exist languages that are inherently ambiguous.

Let us now recall the definitions of k-ambiguity, finite and infinite ambiguity given in [4] for languages in REC. Note that a similar definition of k-ambiguity is contained in [21]. A tiling system $\mathcal{T} = (\Sigma, \Gamma, \Theta, \pi)$ recognizing L is said to be k*-ambiguous* if every picture $p \in L$ has at most k pre-images. A recognizable language L is said k*-ambiguous* if $k = \min\{s \mid \mathcal{T}$ is s-ambiguous tiling system and $\mathcal{T}$ recognizes $L\}$. A language L is *finitely ambiguous* if it is k-ambiguous for some k whereas a language L is *infinitely-ambiguous* if it is not finitely ambiguous.

3 Classes HP and HK

In this section we introduce the definitions of the classes HP and HK of picture languages motivated by some necessary conditions we recall as well.

Let $L \subseteq \Sigma^{**}$ be a picture language. For any $m \geq 1$, we can consider the subset $L(m) \subseteq L$ containing all pictures in L with exactly m rows. Note that the language $L(m)$ can be viewed as a string language over the alphabet of the columns of height m. If L is in REC then it is possible to associate to any tiling system recognizing L a family $\{A_m\}$, where each A_m is an automaton accepting $L(m)$ with a number of states that is at most c^m for some constant c (see [18]).

Moreover, for any string language L, one can define the infinite boolean Hankel matrix $M_L = \|a_{\alpha\beta}\|_{\alpha\in\Sigma^*,\beta\in\Sigma^*}$ where $a_{\alpha\beta} = 1$ if and only if $\alpha\beta \in L$ (see [15]). Observe that, when L is a regular language, the number of different rows of M_L is finite (Myhill-Nerode Theorem). A sub-matrix $M_{(U,V)}$ of an Hankel matrix M_L is a matrix specified by a pair of languages (U, V), with $U, V \subseteq \Sigma^{**}$, that is obtained by intersecting all rows and all columns of M_L that are indexed by the strings in U and V, respectively. Moreover, given a matrix M, we denote by $Rank_Q(M)$, the rank of M over the field of rational numbers Q. A *permutation matrix* is a boolean matrix that has exactly one 1 in each row and in each column.

Definition 1. *[13] Let L be a picture language.*

i) The row complexity function $R_L(m)$ gives the number of distinct rows of the matrix $M_{L(m)}$
ii) The permutation complexity function $P_L(m)$ gives the size of the maximal permutation matrix that is a sub-matrix of $M_{L(m)}$
iii) The rank complexity function $K_L(m)$ gives the rank of the matrix $M_{L(m)}$.

The following theorem collects some necessary conditions for picture languages.

Theorem 1. *Let $L \subseteq \Sigma^{**}$.*

1. *If $L \in REC \cup$ co-REC then there is a $c \in I\!N$ such that, for all $m \geq 1$, $R_L(m) \leq 2^{c^m}$.*
2. *If $L \in REC$ then there is a $c \in I\!N$ such that, for all $m \geq 1$, $P_L(m) \leq c^m$.*
3. *If $L \in UREC$ then there is a $c \in I\!N$ such that, for all $m \geq 1$, $K_L(m) \leq c^m$.*
4. *If $L \in REC \setminus UREC$ and L is k-ambiguous then there is a $c \in I\!N$ such that, for all $m \geq 1$, $K_L(m) \leq c^m$.*

Proof. Item 1 is essentially due to J. Cervelle [8] and item 2 to O. Matz [18]; both of them are rephrased in the matrix framework as in [14]. Item 3 is proved in [3]. Item 4 can be found in [21], for a bit different definition of k-ambiguity, but it holds even for the definition presented in this paper. Indeed if L is k-ambiguous then there exists a constant $c \in I\!N$ such that, for any $m \geq 1$, there is a k-ambiguous automaton A_m that recognizes language $L(m)$ and has c^m states at most: $|A_m| \leq c^m$. Then we can apply a lower bound on the number of states of k-ambiguous automata in [15] that guarantees that $|A_m| \geq Rank_Q(M_L)^{1/k} - 1$. Therefore $K_L(m)^{1/k} \leq d^m$, for some constant $d \in I\!N$, and finally $K_L(m) \leq (d^k)^m$. □

Note that in [2], some subclasses of UREC have been introduced and other necessary conditions founded on $R_L(m)$ have been proved.

Definition 2. *HP is the class of all picture languages $L \in$ co-REC for which there does not exist a constant c such that $P_L(m) \leq c^m$, for all $m \geq 1$.*

HK is the class of all picture languages $L \in REC$ for which there does not exist a constant c such that $K_L(m) \leq c^m$, for all $m \geq 1$.

From previous results, if $L \in$ HP then $L \notin$ REC and if $L \in$ HK then $L \notin$ UREC. Let us now show some examples of languages in HP and in HK. For this, we will use a result, proved in the following Lemma, concerning the rank of some special boolean matrices. In the following, for any matrix $A = \|a_{ij}\|$ with $i = 1, \cdots, m$, $j = 1, \cdots, n$, A_{ij} will denote the (i, j) minor of A.

Lemma 1. *Let $A = \|a_{ij}\|$ be a boolean square matrix of size k such that, for any $1 \leq i, j \leq k$, $a_{ij} = 0$ if and only if $i + j = k + 1$. Then $Rank_Q(A) = k$.*

Proof. It suffices to prove that $det(A) \neq 0$. Remark that A is a square matrix with 0 in all counter-diagonal positions and 1 elsewhere. Let us evaluate $det(A)$ along its first row: $det(A) = \sum_{i=1}^{k}(-1)^{1+i} a_{1i} det(A_{1i}) = det(A_{11}) + (-1)det(A_{12}) + \ldots + (-1)^k det(A_{1n-1}) + 0 \; det(A_{1n})$.

Since, for any $i = 2, \ldots, k-1$, the matrix A_{1i} can be obtained from the matrix A_{1i-1} by swapping its $(k-i+1)$-th row with its $(k-i)$-th one, we can say that $det(A) = (k-1)det(A_{11})$. Therefore, in order to prove that $det(A) \neq 0$, it suffices to show that $det(A)_{11} \neq 0$. Note that A_{11} is a square matrix, of size $k-1$, that has 0 in all the positions immediately above the counter-diagonal and 1 elsewhere. Let us denote by B^h the square matrix of size h of this form and let us show that, for any h, B^h has a non-null determinant. The proof is by induction on h. The basis, $h = 2$, is obvious. Suppose that it is true for B^{h-1} and consider the matrix $B^h = \|b_{ij}\|$. If we evaluate $det(B^h)$ along its first column, we have $det(B^h) = \sum_{i=1}^{h}(-1)^{1+i} b_{i1} det(B^h_{i1})$. Remark that the first $(h-2)$ terms of the sum are equal to 0 (every matrix B^h_{i1} has two identical rows, the last one and the second-last one, and therefore it has a null determinant) and the $(h-1)$-th term is equal to 0 too (note that $b_{(h-1)1} = 0$). So we have $det(B^h) = (-1)^{h+1} det(B^{h-1})$ and, therefore, by inductive hypothesis, $det(B^h) \neq 0$. □

Now, let us fix some notation: we denote by ε the empty string and, for $\Sigma = \{a\}$ and $n \in N$, by a^n the string over Σ^* of length n. Moreover, for $n_1, n_2, \ldots, n_m \in N$, we denote by $lcm(n_1, n_2, \ldots, n_m)$ the lowest common multiple of $n_1, n_2, \ldots, n_m$.

Example 3. Consider, for any $m \geq 0$, the function $f(m) = lcm(2^m+1, \ldots, 2^{m+1})$ and the language L_M over the unary alphabet $\Sigma = \{a\}$, $L_M = \{(m, n) \mid n$ is not a multiple of $f(m)\}$. It was shown that $L_M \in$ REC (see [19,20]).

Now, we will show that $L_M \in$ HK. Indeed, for any $m > 1$, consider languages $L_M(m)$ as defined above and the corresponding boolean matrix $M = M_{L_M(m)}$. Let us denote by c the picture over the alphabet Σ with m rows and one column and consider the set S of $f(m)$ rows of M indexed by c, c^2, ..., $c^{f(m)}$. They are all distinct (the rows indexed by c^i and c^{i+1} differ in the position corresponding to the column indexed by $c^{f(m)-i}$) and, moreover, any other row in M is equal to one of the rows in S. So $R_L(m) = f(m)$. Consider now, in M, the finite sub-matrix M_c composed by the $f(m)$ rows indexed by c, c^2, ..., $c^{f(m)}$, in this order, and the $f(m)$ columns indexed by ε, c, c^2, ..., $c^{f(m)-1}$, in this order, as in the following figure.

$$\begin{array}{c|c|c|c|c|c|c|c|}
 & \varepsilon & c & c^2 & \cdots & c^{f(m)-2} & c^{f(m)-1} & c^{f(m)} \\ \hline
c & 1 & 1 & 1 & \cdots & 1 & 1 & 0 \\ \hline
c^2 & 1 & 1 & 1 & \cdots & 1 & 0 & 1 \\ \hline
c^3 & 1 & 1 & 1 & \cdots & 0 & 1 & 1 \\ \hline
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \hline
c^{f(m)-3} & 1 & 1 & 0 & \cdots & 1 & 1 & 1 \\ \hline
c^{f(m)-2} & 1 & 0 & 1 & \cdots & 1 & 1 & 1 \\ \hline
c^{f(m)-1} & 0 & 1 & 1 & \cdots & 1 & 1 & 1 \\ \hline
\end{array} .$$

Then, the i-th row of M_c will have symbol 1 in all its entries except the $f(m)+1-i$ position that will have symbol 0. For this, matrix M_c satisfies the hypothesis of Lemma 1, and, therefore, $Rank_Q(M_c) = f(m)$. But $f(m) = Rank_Q(M_c) \leq K_L(m) \leq R_L(m) = f(m)$, so we have $K_L(m) = f(m)$. Since $f(m) = 2^{\theta(2^m)}$ (see [18,19]), then $K_{L_M}(m)$ cannot be bounded by k^m where k is a constant, and therefore $L_M \in$ HK.

At last, it is easy to see that, for any $m > 1$, $P_L(m) = 2$.

Consider now the language $\overline{L}_M$ and, for any $m > 1$, languages $\overline{L}_M(m)$. The finite sub-matrix of $M_{\overline{L}_M(m)}$, with same rows and columns indexes as M_c, is a square matrix of size $f(m)$ with 1 in all counter-diagonal positions and 0 elsewhere. It is easy to show that, for any m, $P_{\overline{L}_M}(m) = R_{\overline{L}_M}(m) = f(m)$ and, therefore, $\overline{L}_M \in$ HP. Furthermore we have $K_{\overline{L}_M}(m) = f(m)$ too.

Example 4. Let CORNERS be the language defined in Example 2. We are going to show that CORNERS$\in$ HP, following the proof that CORNERS$\notin$ REC in [18].

Consider for any $n \geq 1$ a partition $\mathcal{P}$ of $\{1, 2, \cdots, 2n\}$ into two-element sets and fix a bijection $\alpha_{\mathcal{P}} : \mathcal{P} \to \{1, 2, \cdots, n\}$. Then define picture $P_{\mathcal{P}}$ over $\{a, b\}$ as the picture of size $(2n, n)$ such that the position (i, j) in $P_{\mathcal{P}}$ carries a b if and only if $j = \alpha_{\mathcal{P}}(\{i, i'\})$ and $\{i, i'\} \in \mathcal{P}$. As an example let $n = 3$, $\mathcal{P}= \{(1, 2), (3, 4), (5, 6)\}$ and $\mathcal{P}'= \{(1, 3), (2, 4), (5, 6)\}$; then fix $\alpha_{\mathcal{P}}((1, 2)) = 1$, $\alpha_{\mathcal{P}}((3, 4)) = 2$, and $\alpha_{\mathcal{P}}((5, 6)) = 3$; $\alpha_{\mathcal{P}'}((1, 3)) = 1$, $\alpha_{\mathcal{P}'}((2, 4)) = 2$, and $\alpha_{\mathcal{P}'}((5, 6)) = 3$. Pictures $P_{\mathcal{P}}$ and $P_{\mathcal{P}'}$ are as follows:

$$P_{\mathcal{P}} = \begin{array}{|c|c|c|}\hline b & a & a \\ \hline b & a & a \\ \hline a & b & a \\ \hline a & b & a \\ \hline a & a & b \\ \hline a & a & b \\ \hline \end{array} \qquad P_{\mathcal{P}'} = \begin{array}{|c|c|c|}\hline b & a & a \\ \hline a & b & a \\ \hline b & a & a \\ \hline a & b & a \\ \hline a & a & b \\ \hline a & a & b \\ \hline \end{array}$$

Let $M_{L(2n)}$ be the Hankel matrix of the language $L(2n)$ of pictures in CORNERS of fixed height $2n$, and $M_{(U,V)}$ its sub-matrix specified by the pair of languages (U, V) with $U = V = \{P_{\mathcal{P}} \mid \mathcal{P}$ is a partition of $\{1, 2, \cdots, 2n\}$ into two-element sets$\}$. We have that $M_{(U,V)}$ is a permutation matrix. Indeed the entry $(P_{\mathcal{P}}, P_{\mathcal{P}'})$ of $M_{(U,V)}$ is 1 iff $\mathcal{P} = \mathcal{P}'$.

Furthermore the size of matrix $M_{(U,V)}$ is equal to the number A_n of partitions of $\{1, 2, \cdots, 2n\}$ into two-element sets. And it can be shown that $A_n \geq n!$ and then there does not exist $c \in I\!N$ such that $A_n \leq c^n$.

Let us mention that another language in HK is $L_{c=c'}$ as introduced in Example 1 (see [2]), while its complement is in HP (see [13]).

4 Some Results on Classes HP and HK

In this section we give partial answers to Questions 1 and 2 as stated in the Introduction, in the case the involved languages belong to classes HP and HK introduced in Section 3. Firstly let us compare the values of the complexity functions $R_L(m)$, $P_L(m)$ and $K_L(m)$ introduced in Section 3, for a language L and its complement, in the case L is in REC $\cup$ co-REC (and therefore functions $R_L(m)$, $P_L(m)$ and $K_L(m)$ have finite values.

Proposition 1. *Let $L \in$ REC $\cup$ co-REC.*

1. $R_{\overline{L}}(m) = R_L(m)$.
2. $P_L(m) + P_{\overline{L}}(m) \leq R_L(m) + 2$ *and the bound is tight.*
3. $K_L(m) + K_{\overline{L}}(m) \leq 2R_L(m)$ *and the bound is tight.*
4. $K_{\overline{L}}(m) \geq P_L(m)$ *and the bound is tight.*

Proof

1. The Hankel matrices for $\overline{L}$ can be obtained by exchanging entries 0 with entries 1 in the Hankel matrices for L.
2. Let $m \geq 1$, $M_{(U,V)}$ be a permutation matrix of maximal size that is a sub-matrix of the Hankel matrix $M_{L(m)}$ for $L(m)$, and let $\overline{M}_{(U',V')}$ be a permutation matrix of maximal size that is a sub-matrix of the Hankel matrix $M_{\overline{L}(m)}$ for the complement of $L(m)$. We claim that $|U \cap U'|$, $|V \cap V'| \leq 2$. In other words the sub-matrices of $M_{L(m)}$ specified by (U, V) and (U', V'), respectively, cannot overlap more than on a square matrix of size 2. Consider indeed a column of $M_{L(m)}$ indexed by a string in $|V \cap V'|$. The set of entries on such column indexed by strings in U are all 0's except for one 1 and then they cannot share more than two elements (a 0 and a 1) with the set of entries indexed by strings in U' (that are all 1's except for one 0).

   The bound is tight for language L_M in Example 3: $P_{L_M}(m) = 2$ and $P_{\overline{L}_M}(m) = R_{L_M}(m) = f(m)$ and, therefore, $P_{L_M}(m) + P_{\overline{L}_M}(m) = R_{L_M}(m) + 2$.
3. The inequality follows from Item 1 and from the remark $K_L(m) \leq R_L(m)$. The bound is tight for language L_M in Example 3: $K_{L_M}(m) = K_{\overline{L}_M}(m) = R_{L_M}(m) = f(m)$ and, therefore, $K_{L_M}(m) + K_{\overline{L}_M}(m) = 2R_{L_M}(m)$.
4. Let P be a maximal permutation matrix of $H_L(m)$. Clearly, P is a boolean square matrix of size $P_L(m)$ and we can assume, without loss of generality, that P has 1 in all counter-diagonal positions and 0 elsewhere. Now, consider $H_{\overline{L}}(m)$ and its submatrix of size $P_L(m)$, say $\overline{P}$, that corresponds to the permutation matrix P of $H_L(m)$. Remark that $\overline{P}$ is a square matrix of size

$P_L(m)$ with 0 in all counter-diagonal positions and 1 elsewhere. Therefore, the matrix $\overline{P}$ satisfies the hypothesis of Lemma 1 and we have $Rank_Q(\overline{P}) = P_L(m)$ that implies $K_{\overline{L}}(m) \geq P_L(m)$. The bound is tight for language $\overline{L}_M$, that is the complement of language L_M in Example 3. $\square$

Corollary 1. *If $L \in HP$ then $\overline{L} \in HK$.*

The following proposition is a positive answer to Question 1 (see Section 1) in the case where $\overline{L} \notin$ REC since $\overline{L} \in$ HP. Recall that if a language is in HP then it is necessarily not in REC; and that we do not know at present whether this is also a sufficient condition. Note that if this condition were also sufficient then HP=co-REC\REC. Vice versa, if HP = co-REC\REC then the condition would be also sufficient for languages in co-REC.

Proposition 2. *If $L \in REC$ and $\overline{L} \in HP$ then $L \notin UREC$.*

Proof. If $\overline{L} \in$ HP then, from Corollary 1, $L \in$ HK and therefore, from the definition of HK and Item 3 of Theorem 1, $L \notin$ UREC. $\square$

As an application, consider the following example.

Example 5. In Example 4 we showed that $CORNERS \in$ HP. Applying Corollary 1 we have that $\overline{CORNERS} \in$ HK and finally $\overline{CORNERS} \notin$ UREC.

The following proposition is a negative answer to Question 2 (see Section 1) in the case where $L \notin$ UREC since $L \in$ HK. Recall that if a language is in HK then it is necessarily inherently ambiguous; and that we do not know at present whether this is also a sufficient condition. If this condition were also sufficient then HK=REC\UREC. Vice versa, if HK = REC\UREC then the condition would be also sufficient for languages in REC.

Proposition 3. *Any language $L \in REC \setminus UREC$ such that $L \in HK$ is infinitely ambiguous.*

Proof. The proof follows from item 4 in Theorem 1. $\square$

Example 6. In Example 5 we showed that $\overline{CORNERS} \in$ HK; hence from Proposition 3 we have that $\overline{CORNERS}$ is infinitely ambiguous.

5 Necessary Conditions in the Unary Case

In this section we will introduce a new necessary condition for the belonging of a picture language to REC, in the case when the alphabet is unary. We will then compare the obtained bound with the other ones known in the literature (see Theorem 1). All along this section $|\Sigma| = 1$.

Let us recall some classical results on unary regular string languages (see [9,10]). A unary language is regular if and only if it is ultimately periodic. The size of a deterministic finite automaton (dfa) for an unary language is (λ, μ), where λ is the number of states in the cycle and μ is the number of states not in the cycle. A language is said properly ultimately λ-cyclic when it is accepted by a dfa of size (λ, μ) and by no dfa of size (λ', μ') with $\lambda' < \lambda$; λ is said the period of the language. Obviously any regular unary language is properly ultimately λ-cyclic for some $\lambda \in I\!N$.

Suppose $\lambda \in I\!N$ factorizes in prime powers as $\lambda = p_1^{k_1} p_2^{k_2} \cdots p_s^{k_s}$; we will denote by $spp(\lambda)$ the sum of its prime powers, that is $spp(\lambda) = p_1^{k_1} + p_2^{k_2} + \cdots + p_s^{k_s}$. The following lower bound was proved for unary non-deterministic finite automata (nfa) in [16]: Each nfa accepting a properly ultimately λ-cyclic language has at least $spp(\lambda)$ states in its cycles. We will apply this lower bound to obtain a new necessary condition for unary picture languages in REC. Recall the definition of $L(m)$ in Section 3.

Proposition 4. *Let L be a recognizable language over a unary alphabet and for any $m \geq 1$, λ_m be the period of $L(m)$. Then there exists a constant $c \in I\!N$ such that for any $m \geq 1$, $spp(\lambda_m) \leq c^m$.*

Proof. If $L \in$ REC then there exist nfa's A_m that recognize languages $L(m)$ and have c^m states at most, for some constant $c \in I\!N$ (see Section 3): $|A_m| \leq c^m$ for any $m \geq 1$. In the case $|\Sigma| = 1$, we can apply the above mentioned lower bound of [16] to automata A_m's, and obtain that for any $m \geq 1$: $spp(\lambda_m) \leq |A_m| \leq c^m$. $\square$

When the alphabet is unary, Proposition 4, with items 1 and 2 of Theorem 1, provides three different conditions for the belonging of a language to REC. Let us summarize the three conditions.

Proposition 5. *Let L be a recognizable language over a unary alphabet and for any $m \geq 1$, λ_m be the period of $L(m)$.*

Then there exists a constant $c \in I\!N$ such that for any $m \geq 1$:

1. $\log R_L(m) \leq c^m$
2. $P_L(m) \leq c^m$
3. $spp(\lambda_m) \leq c^m$.

In the following example we compare the three bounds of Proposition 5.

Example 7. Consider language L_M as defined in Example 3: $L_M = \{(m, n) \mid n$ is not a multiple of $f(m)\}$ where $f(m) = lcm(2^m + 1, \ldots, 2^{m+1})$, for all $m \geq 0$. Recall that $L_M \in$ REC. The Hankel matrices associated to languages $L(m)$ are described in Example 3. We show that $P_{L_M}(m) \leq \log R_{L_M}(m) \leq spp(\lambda_m)$.

We have $\lambda_m = R_{L_M}(m) = f(m)$ and $P_{L_M}(m) = 2$. Suppose that for any $m \geq 0$, λ_m factorizes in prime powers as $\lambda_m = \prod p_{m,i}^{k_{m,i}}$, then $spp(\lambda_m) = \sum p_{m,i}^{k_{m,i}}$. Hence $2 = P_L(m) < \log R_{L_M}(m) = \log(\prod p_{m,i}^{k_{m,i}}) = \sum \log(p_{m,i}^{k_{m,i}}) < \sum p_{m,i}^{k_{m,i}} = spp(\lambda_m)$.

6 Conclusions and Open Questions

In the paper we afforded some open problems on unambiguity, finite ambiguity and complementation (Questions 1, 2 and 3 in the Introduction) and gave some partial answers. A complete answer to Question 1 seems far to be found, also due to its interpretation in the computational complexity framework.

With Question 2, we considered the possibility that in REC there exist finitely ambiguous languages, and showed that this is not true for a class of languages in REC. Note that in a bit different framework (see [4]), when the recognition is accomplished without border symbols (tiles with # are not allowed), it is shown that there is an infinite hierarchy of finitely ambiguous languages. Therefore the border symbols have to play a major role, in order to show that in REC there do not exist finitely ambiguous languages. We figure that when a language is recognized by a tiling system with finite ambiguity, then it is possible to obtain an unambiguous tiling system for the language by paying special attention to border tiles.

Finally the problem of finding characterizations of meaningful classes of recognizable languages (Question 3) deserves some more investigation.

References

1. Anselmo, M., Giammarresi, D., Madonia, M.: New Operators and Regular Expressions for two-dimensional languages over one-letter alphabet. Theoretical Computer Science 340(2), 408–431 (2005)
2. Anselmo, M., Giammarresi, D., Madonia, M.: From determinism to non-determinism in recognizable two-dimensional languages. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 36–47. Springer, Heidelberg (2007)
3. Anselmo, M., Giammarresi, D., Madonia, M., Restivo, A.: Unambiguous Recognizable Two-dimensional Languages. RAIRO: Theoretical Informatics and Applications 40(2), 227–294 (2006)
4. Anselmo, M., Jonoska, N., Madonia, M.: Framed Versus Unframed Two-dimensional Languages. In: Nielsen, M., Kucera, A., Miltersen, P.B., Palamidessi, C., Tuma, P., Valencia, F.D. (eds.) SOFSEM 2009. LNCS, vol. 5404, pp. 79–92. Springer, Heidelberg (2009)
5. Anselmo, M., Madonia, M.: Deterministic two-dimensional languages over one-letter alphabet. In: Bozapalidis, S., Rahonis, G. (eds.) CAI 2007. LNCS, vol. 4728, pp. 147–159. Springer, Heidelberg (2007)
6. Anselmo, M., Madonia, M.: Deterministic and unambiguous two-dimensional languages over one-letter alphabet. Theoretical Computer Science 410, 1477–1485 (2009)
7. Bertoni, A., Goldwurm, M., Lonati, V.: On the complexity of unary tiling-recognizable picture languages. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 381–392. Springer, Heidelberg (2007)
8. Cervelle, J.: Langages de figures. Rapport de stage, ENS de Lyon (1997)
9. Chrobak, M.: Finite automata and unary languages. Theoret. Comput. Sci. 47, 149–158 (1986)
10. Eilenberg, S.: Automata, Languages and Machines, vol. A. Academic Press, London (1974)
11. Giammarresi, D., Restivo, A.: Recognizable picture languages. Int. Journal Pattern Recognition and Artificial Intelligence 6(2&3), 241–256 (1992)
12. Giammarresi, D., Restivo, A.: Two-dimensional languages. In: Rozenberg, G., et al. (eds.) Handbook of Formal Languages, vol. III, pp. 215–268. Springer, Heidelberg (1997)

13. Giammarresi, D., Restivo, A.: Matrix based complexity functions and recognizable picture languages. In: Grader, E., Flum, J., Wilke, T. (eds.) Logic and Automata: History and Perspectives. Texts in Logic and Games 2, pp. 315–337. Amsterdam University Press (2007)
14. Giammarresi, D., Restivo, A.: Ambiguity and complementation in recognizable two-dimensional languages. In: Ausiello, G., Karhumäki, J., Mauri, G., Ong, L. (eds.) Procs. Intern. Conf. on Theoret. Compu. Sci. IFIP, vol. 273, pp. 5–20. Springer, Boston (2008)
15. Hromkovic, J., Karumäki, J., Klauck, H., Schnitger, G., Seibert, S.: Communication Complexity Method for Measuring Nondeterminism in Finite Automata. Information and Computation 172, 202–217 (2002)
16. Jiang, T., McDowell, E., Ravikumar, B.: structure and complexity of minimal nfa's over a unary alphabet. Intern. J. Found. Comput. Sci. 2, 163–182 (1991)
17. Lindgren, K., Moore, C., Nordahl, M.: Complexity of two-dimensional patterns. Journal of Statistical Physics 91(5-6), 909–951 (1998)
18. Matz, O.: On piecewise testable, starfree, and recognizable picture languages. In: Nivat, M. (ed.) FOSSACS 1998. LNCS, vol. 1378, p. 203. Springer, Heidelberg (1998)
19. Matz, O.: Dot-depth and monadic quantifier alternation over pictures. Ph.D. thesis Technical Report 99-08, RWTH Aachen (1999)
20. Matz, O.: Dot-depth, monadic quantifier alternation, and first-order closure over grids and pictures. Theoretical Computer Science 270(1-2), 1–70 (2002)
21. Mäurer, I.: Characterizations of Recognizable Picture Series, Ph.D Thesis Universität Leipzig, Institut für Informatik, Abteilung Automaten und Sprachen (2007)
22. Potthoff, A., Seibert, S., Thomas, W.: Nondeterminism versus determinism of finite automata over directed acyclic graphs. Bull. Belgian Math. Soc. 1, 285–298 (1994)

Context-Free Categorical Grammars

Michel Bauderon, Rui Chen, and Olivier Ly

Université de Bordeaux - LaBRI CNRS,
351 cours de la Libération Talence 33405 France
Zhongnan University of Economics and Law,
1 Nanhunan Dadao, Hongshan Wuhan 430073 China
{bauderon,ly}@labri.fr,
rui.chen.bordeaux1@gmail.com

Abstract. We define generic categorical notions of rewriting and grammar, using two basic operations, pullback and pushout, and show that these categorical grammars are intrinsically context-free in the sense of Courcelle. We then specialise to various settings, including classical word grammars, hyperedge replacement grammars or node-replacement grammars. We show that some languages which are classical counter-example to context-freeness become context-free within this new framework[1].

Keywords: Category, rewriting system, grammar, context-freeness.

1 Introduction

This works stems from research in the more specific area of graph rewriting where two main directions have been explored, which correspond to two distinct (somehow dual) approaches to the structure of a graph, either as nodes linked by arrows (vertex rewriting) or as arrows glued by nodes (edge or hyperedge rewriting).

In both directions, four levels of description have been explored : set theoretic, algebraic (namely using universal algebra), logical or categorical (using category theory as a basic tool). In this last setting - using category theory - the main effort has been devoted to edge (and hyperedge) replacement - using pushout as a basic operation to generalize the usual substitution, leading to the development of a large theoretical body, via the double and single pushout approach to graph rewriting (the so-called algebraic approach) and their extensions (the reader may refer to [9] for an extensive descriptions of formalisms and results).

In earlier works (such as [2,3,4,5]), we have shown how a dual approach - using pullback in place of pushout as the basic rewriting operation - could provide a sound categorical approach to node rewriting in graphs (and hypergraphs [4]). We have shown in [3] that pullback graph grammars are context-free.

In this paper, we generalize this approach by defining a generic categorical treatment of substitution, rewriting and grammars, abstracting as much as possible and lifting main notions and results to their proper level of abstraction. We

[1] This work was been completed while the first author was on a CNRS leave at LIAMA, Chinese Academy of Sciences, Institute of Automation, Beijing.

thus reach a necessary but sufficient level of abstraction to ensure the context-freeness property (in the sense of [6]) and it is our main result that categorical grammars[2] are intrinsically context-free. The main ideas of the proof remain those which have been presented in [3,5], although some useless conditions have been removed here.

This allows us to describe in this new setting several standard examples (such as words and graphs), showing that the mere reversing of arrows leads to very different situations and that the projective grammars are much more powerful than the inductive ones.

We show for instance in section 3, that inductive word grammars are exactly classical context free word grammars, while projective word grammars can generate some context-sensitive languages such as the well known $a^n b^n c^n$, which then becomes context-free in the categorical setting.

This work relies on elementary category theory whose basic definitions will be taken for granted (but the reader may refer e.g. to [1] available on line). Due to space limitations, proofs have been omitted.

2 Rewriting in a Category

There are (at least) two possible ways to rewrite objects in a category, by using the two simplest standard binary operations available in this framework, namely pullback and pushout (product and coproduct are much simpler, but much less flexible). Although pushout directly generalizes classical substitution, we follow the traditional approach where products, pullbacks and projective limits are put first, pushout and inductive limits being left to a duality argument ([1]).

In this paper, we shall not consider double-pushout or double-pullback rewritings, which introduce some pattern matching in the computation process. Our rules will always be directly applicable.

2.1 Basic Definitions

As usual, we let $\#S$ denote the cardinality of a set S and $\mathbb{N}$ be the set of non negative integers. In a category $\mathcal{C}$, a *span* (resp. a *cospan*) in $\mathcal{C}$ is a pair of arrows with same codomain (resp. domain). A family of arrows F has domain G (resp. codomain G) if the domain (resp. codomain) of each arrow of F is G. Let $G \xrightarrow{u} X \xleftarrow{p} H$ be a span. If it exists, let us denote by $G[p/u]$ the pullback object of this span and let $G \xleftarrow{a} G[p/u] \xrightarrow{b} H$ be the associated co-span. Let us note that it is symmetric: $G[p/u] = H[u/p]$. For any arrow f with domain G, we define a new arrow $f[p/u] = f \circ a$ with domain $G[p/u]$ and by extension, for any family F of arrows with domain G, we define $F[p/u] = \{f[p/u] \mid f \in F\}$.

Source. From now on, we shall consider a distinguished subset $\mathcal{N}$ of objects in $\mathcal{C}$ whose elements will be called *non-terminals*.

[2] We are aware that, by its simplicity, the expressions "categorical grammars" has been extensively used, e.g. in the area of computational linguistics, but we did not find any more accurate expression to designate our grammars.

Definition 1. *An $\mathcal{N}$-*source *(or simply a* source*) is a triple $\overline{G} = (G, U_G, P_G)$ consisting of an object G in $\mathcal{C}$ and two families U_G and P_G of arrows which share the same domain G, whose codomains are non-terminal objects and such that for any non-terminal X, there is at most one arrow in P_G with codomain X. Elements of U are called the* unknowns *occuring in $\overline{G}$. An element $p : G \to X$ of P_G is called the* replacement scheme *for the non-terminal X in $\overline{G}$. A source is said to be* terminal *if it has no unknown, i.e., if $\#U_G = 0$.*

Substitution. Substitutions operate on sources: substituting an unknown in a source by an other source will rise to a new source, in the following way:

Definition 2. *Let $\overline{G} = (G, U_G, P_G)$ and $\overline{H} = (H, U_H, P_H)$ be two sources. Let $u : G \to X$ be an unknown of $\overline{G}$. If $\overline{H}$ has a replacement scheme $p : H \to X$ for the non-terminal X, the* substitution *(or* replacement*) of u by $\overline{H}$ in $\overline{G}$ is the source, denoted by $\overline{G}[\overline{H}/u]$, and defined by the triple:*

$$(G[p/u],\ (U_G\backslash\{u\})[p/u] \cup U_H[u/p],\ P_G[p/u])$$

Let us note that the notation $\overline{G}[\overline{H}/u]$ is no longer symmetric.

Rewriting. We can now define a rewriting rule (together with the rewriting mechanism):

Definition 3. *A* rewriting rule *is a pair denoted by $X \to \overline{H}$ where X is a non-terminal object and $\overline{H}$ is a source provided with a replacement scheme for X. Applying a rule r to a source $\overline{G}$ consists in selecting an unknown $u : G \to X$ of $\overline{G}$ (if any) and substituting $\overline{H}$ to u, giving rise to $\overline{G}[\overline{H}/u]$. This will be denoted by $\overline{G} \overset{r,u}{\Longrightarrow} \overline{G}[\overline{H}/u]$, or simply $\overline{G} \overset{r}{\Longrightarrow} \overline{G}[\overline{H}/u]$.*

As usual, one-step derivation $\overset{r}{\Rightarrow}$ defines a binary relation on sources whose reflexive and transitive closure is denoted by $\overset{*}{\Rightarrow}$.

2.2 Dual Approach

As already mentionned, similar definitions can of course be given in a dual way, by defining a *sink* (or co-source) with families of morphisms of the form $((f_i : X_i \to G)_{i\in I})$. G is the codomain of the sink and its domains are in $\mathcal{N}$. The definition of substitution will be dual, making use of pushout instead of pullback as a basic operation to produce a sink out of two sinks.

In the sequel, we shall try to simplify by using the expressions *production rules* (resp. *occurrence*) to designate both sources or sinks (resp. both sources or sinks with $\#P_G = 0$).

2.3 Rewriting Structures

Definition 4. *A* categorical rewriting system *in a category $\mathcal{C}$, over a family of objects $\mathcal{N}$ called non-terminals is given by a family of productions. The system is* admissible *when all productions may be applied at any stage. A* categorical grammar *will be given by a categorical rewriting system and a specific occurrence called the* axiom.

We shall in the sequel use the prefix *projective* (resp. *inductive*) to denote systems relying on pullback (resp. pushout) as their rewriting mechanism.

If the category is complete (projective systems) or co-complete (inductive systems), any categorical rewriting system is admissible. If not, we shall have to give conditions for such a system to be admissible.

The *language* generated by the grammar is the family of terminal occurences derived through the relation $\overset{*}{\Rightarrow}$. The *extended language* is the family of not necessarily terminal occurrences derived through the relation $\overset{*}{\Rightarrow}$.

2.4 Context-Freeness

We shall follow Courcelle [6], and use his axiomatic definition of the notion of context-freeness as the conjunction of three properties: preservation, confluence and associativity. Let us first describe categorical (co)rewriting system as a *substitution system* in the sense of [6].

The *alphabet* is simply $\mathcal{N}$. To completely fall within the framework described by [6], we need an indexing of the non-terminal objects as $\mathcal{N} = \{X_i / i \in [1, n]\}$.

The *objects* are the sources over $\mathcal{N}$, whose set is denoted by $\mathcal{C}_{\mathcal{N}}$. If $\overline{G} = (G, U_G, P_G)$ is such an object, the *arity function* α sends it to the ordered list $\alpha(\overline{G}) = (X_1 X_2 \dots X_m)$ consisting of the elements of the (co)domain of $U_G = \{(u_j : G \to X_j)_{j \in [1,m]}\}$.

According to definition 2, the *substitution* operator [ ] defines a partial mapping from $\mathcal{C}_{\mathcal{N}} \times \mathcal{N} \times \mathcal{C}_{\mathcal{N}}$ to $\mathcal{C}_{\mathcal{N}}$ by $(\overline{G}, X, \overline{R}) \to \overline{G}[\overline{R}/X]$ whenever this makes sense. Using the previous indexing, it may also be described as a partial mapping $\mathcal{C}_{\mathcal{N}} \times \mathbb{N} \times \mathcal{C}_{\mathcal{N}}$ to $\mathcal{C}_{\mathcal{N}}$ defined by $(\overline{G}, i, \overline{R}) \to \overline{G}[\overline{R}/X_i]$.

Preservation. The first condition for context-freeness is to satisfy the *preservation axiom*: for all $(\overline{G}, i, \overline{R})$ in the domain of [ ], one must have

$$\alpha(\overline{G}[\overline{R}/X_i]) = X_1 X_2 \dots X_{i-1} \alpha(\overline{R}) X_{i+1} \dots X_n$$

where $X_1 X_2 \dots X_n = \alpha(\overline{G})$, for $X_1, \dots, X_n \in X$. It follows from definition 2 that:

Proposition 1. *The substitution mapping satisfies the preservation axiom.*

Hence, $\mathbf{G} = \langle \mathcal{C}_{\mathcal{N}}, \mathcal{N}, \alpha, [\,] \rangle$ is a *substitution system* in the sense of [6]. We may now study its properties.

Associativity. The first property expresses the fact that two consecutive steps of rewriting can be condensed into one.

Proposition 2. $\overline{G}[\overline{R}/X][\overline{S}/(Y[\overline{R}/X])]$ *and* $\overline{G}[\overline{R}[\overline{S}/Y]/(X[\overline{S}/X])]$ *are isomorphic (whenever they can both be computed), hence categorical rewriting systems are* associative.

Confluence. This second property expresses the commutativity of the substitution operation, or the fact that rewriting steps can be applied in any order, giving the same result.

Proposition 3. *$\overline{G}[\overline{R}/X][\overline{S}/Y]$ and $\overline{G}[\overline{S}/Y][\overline{R}/X]$ are isomorphic whenever they can both be computed hence categorical rewriting systems are* confluent.

These results show that categorical rewriting systems satify Courcelle's conditions for context-freeness ([6]) hence that :

Theorem 1. *Categorical rewriting systems are context-free.*

In the sequel, we shall use the expression *context-free* (abbreviated as CF) to denote the notion of context-freeness in the sense of [6] that we have briefly recalled in this section. When comparing with other definitions, we shall use a prefix such as w-CF to denote the standard definition of context-freeness in the classical theory of word languages.

3 Word Grammars

To describe word (or tree) languages in this setting, we need to put them in a categorical framework which is as close as possible to the usual definitions where for instance words are mappings from $[1, n] \subset \mathbb{N}$ to an alphabet A.

We shall consider as a base category the category **Pos** of partially ordered sets (with order-preserving mappings) which is well known to be both complete and cocomplete (see for instance [1]) and more precisely, categories $\mathbf{Pos}_A$ of posets labeled over an alphabet $A = \{a, b, c, \ldots\}$.

To model words or trees, we have to consider subcategories of **Pos** which are neither complete nor cocomplete, meaning that the pushout and pullback objects (which always exists in the enclosing category) may fail to be elements of the category of interest, hence that the rewriting systems mail fail to be admissible. Ensuring that they are will put stringent conditions of the type of rewriting systems that we can build. In each case we must identify conditions under which a pushout or a pullback object belongs to this subcategory as well as coherence conditions to be fulfilled by the labelings. Due to space limitations, we shall deal here only with words.

3.1 Words

To model words, we shall consider the subcategory $A^* = \mathbf{Tos}_A$ of totally ordered finite sets labeled over A.

Let us first give a few definitions. If $m \in A^*$, with $\#m = n$, we shall write $m = m^1m^2m^3 \ldots m^n$ (subscripts will be used to denote elements in a family of words). If its order relation is denoted by $\leq$, we shall say that two elements of $m^1 \leq m^2$ in m are *adjacent* if there is no m^3 such that $m^1 \leq m^3 \leq m^2$; m^2 is then the *successor* of m^1, while m^1 is the *predecessor* of m^2.

Totally Ordered Sets. The following two lemmata give a characterization of those pushouts and pullbacks which build total orders out of total orders.

Starting with the more intuitive case, let us consider the following pushout diagram in **Pos**, where x, u and v are total orders:

$$\begin{tikzcd} x \arrow[r, "xv"] \arrow[d, "xu"] & v \arrow[d, "vm"] \\ u \arrow[r, "um"] & m \end{tikzcd}$$

The pushout object m is easily constructed: its carrier is the pushout object in **Set** (the category of sets) while the order relation is induced by those on u and v : $m^1 \leq m^2$ if and only if $v^1 \leq v^2$ or $u^1 \leq u^2$, where $m^i = um(u^i)$ and/or $m^i = vm(v^i)$, $i = 1, 2$.

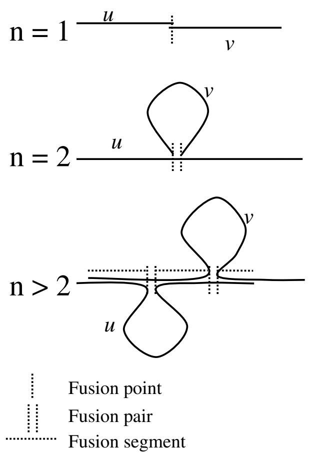


Fig. 1. Pushouts of total orders

Intuitively, if we represent u and v by linear segments, m is obtained by gluing these two segments at some fusion points (or along fusion segments) which are images of elements of x in u and v. The main possibilities to build total orders out of total orders are described in figure 1.

Lemma 1. *Let $n = \#x$.*

1. *if $n = 1$, m is a total order if and only if $xv(x)$ is the maximal point of v and $xu(x)$ is the minimal point of u (or conversely).*
2. *if $n = 2$, m is a total order if and only if $xv(x^1)$ and $xv(x^2)$ are adjacent in v, while $xu(x^1)$ and $xu(x^2)$ are the extremal points of u (or conversely)*
3. *if $n > 2$, m is a total order if and only if the three following conditions hold*
   (a) either $xv(x^1)$ is the minimum of v or $xu(x^1)$ is the minimum of u,
   (b) either $xv(x^n)$ is the maximum of v or $xu(x^n)$ is the maximum of u,
   (c) if x^i and x^j are adjacent in x, then at least one of the pairs $xv(x^i)$ and $xv(x^j)$ or $xu(x^i)$ and $xu(x^j)$ is adjacent (in v or u).

Remark 1. The first two cases are easily interpreted. The case where $n = 1$ clearly corresponds to the concatenation vu (or conversely uv) and could therefore be used to model *regular word grammars* (which we will not do), while the case $n = 2$ can be seen as the substitution of x by u in v (or conversely x by v in u) and will be used later to model *context-free word grammars.* The third case identifies and mixes parts of the two words according to the definition of the mappings o_x and r_x.

Let us now consider the pullback case, by considering the following pullback diagram in **Pos**:

$$\begin{tikzcd}
m \arrow[r, "mv"] \arrow[d, "mu"] & v \arrow[d, "vx"] \\
u \arrow[r, "ux"] & x
\end{tikzcd}$$

The carrier of the pullback object m is computed as the pullback object in **Set**, ie the set of those elements (u_1, v_1) in the cartesian product $u \times v$ such that $ux(u_1) = vx(v_1)$. The order on m is the order induced by the product order, namely $(u_1, v_1) \leq (u_2, v_2)$ if and only if $u_1 \leq u_2$ and $v_1 \leq v_2$.

Lemma 2. *The pullback object is a total order if and only if for each x_i in x, either $\#ux^{-1} \leq 1$ or $\#vx^{-1} \leq 1$.*

Words. After describing conditions for pushout and pullback of total orders to be total orders, we must describe the action of these operations on the labeling which we need in order to turn total orders into words.

Let $A = \{a, b, c, \ldots\} \cup \{\odot, \top, \bot\}$ be a set of terminal letters with three special symbols $\odot$, $\top$ and $\bot$. We shall let the alphabet A be partially ordered by $\bot, \top \leq a \leq \odot, \forall a \in A$ (letters in A are not comparable, $\bot$ and $\top$ need not be either).

In $\mathbf{Tos}_A$, an object m of length n (a word in A^*) is a mapping $[1, n] \overset{m}{\to} A$, that we may write in the form $\{m_1 \leq m_2 \leq \ldots \leq m_n\}$ where $m_i \in A$.

Since the set A of labels (terminal letters) is ordered, we can set :

Definition 5. *A morphims of words $f : m \to m'$ is an order-preserving mapping (i.e. an arrow in **Tos**) such that $m_i \leq m'_{f(i)}$.*

One may check that the composition of two such morphism is still a morphism hence that $A^* = \mathbf{Tos}_A$ is a category, and that if a diagram is a pushout (resp. a pullback) in **Tos**, then it is a pushout (resp. a pullback) in $\mathbf{Tos}_A$. It is enough for that to set that whenever an element m_i in m has preimages (resp. is image of) both u_i and v_i in u and v, the label of m_i will be the maximum (resp. the minimum) of the labels of u_i and v_i. It will be shown in the two following sections that whenever the result of the computation is in **Tos**, then these labels are comparable.

3.2 Inductive Grammar

It follows from lemma 1, that the usual context-free substitution of a letter by a word can be modelled in the category $A^* = \mathbf{Tos}_A$.

A non terminal letter x will be modeled by a 2-elements total order $x = \{x^1 \leq x^2\}$ (we shall use the same symbol to denote the letter and the order), which we shall label $\bot\top$. An occurence of x in a word w is an arrow $x \overset{o_x}{\to} w$, such that $o_x(x^1)$ and $o_x(x^2)$ are adjacent. This means that w is of the form $w^1 \dots w^k o_x(x^1) o_x(x^2) w^{k+2} \dots w^p$. We shall write as usual $w = txu$, with $t = w^1 \dots w^k$ and $u = w^{k+2} \dots w^p$.

Let us be given a context-free word rewriting rule of the form $x \to m$, where $m \in A^*$ is a word of length n and some other non-terminals y_i occuring in m.

This rule may be modelled by a sink $((y_i \overset{o_{y_i}}{\to} m), x \overset{r_x}{\to} m)$ where $r_x(x^1) = m^1, r_x(x^2) = m^n, \#m = n$, and for each i, $y_i \overset{o_{y_i}}{\to} m$ is an occurence[3] of y_i in m.

The substitution is given by computing the pushout corresponding to the following diagram (which is well defined in $\mathbf{Pos}_A$):

$$\begin{tikzcd} x \arrow[r, "r_x"] \arrow[d, "o_x"] & m \arrow[d, "\alpha"] \\ txu \arrow[r, "\beta"] & tmu \end{tikzcd}$$

From Lemma 1, it follows that tmu is a total order. The labelling on tmu is uniquely defined everywhere except on the images of x^1 and x^2 where there are two possibilities, one coming from x in txu, the other from m. Since x is labelled by $\bot\top$, the labelling on m is well defined in each case as the maximum of the two possible labels.

Each arrow $(y_i \overset{r_{y_i}}{\to} m)$ defines by composition with α an occurence $(y_i \overset{\alpha\circ r_{y_i}}{\to} tmu)$ and the computation can be continued by further application of rules of the form $(y_i \to m_i)$.

This construction shows that inductive rewriting models the substitution mechanism used in the standard theory of words languages.

The details of the encoding are omitted due to the lack of space.

Conversely, it may be shown that any pushout rule can be interpreted as a classical context-free rewriting rule, hence:

Lemma 3. *Context-free word languages are exactly inductive word languages, which we summarize as w-CF = po.*

3.3 Projective Grammar

We shall now make use of the symbol $\odot$ which we added to A to label a single element order which will thus be turned into a terminal object in the category A^* and a neutral element for the categorical product. For the sake of clarity in the following diagrams defining morphisms, we shall also use the letters $x, y, z, \dots$ to denote the same neutral element $\odot$.

[3] Of course m must be of the form $m = a_1 y_1 \dots a_p y_p a_{p+1}$ for some (possibly empty) words $a_1, \dots a_{p+1}$, meaning that the images $o_{y_i}(y_i)$ in m must not overlap. This corresponds to the fact that there can not be two distinct letters at the same place in a word and is necessary to ensure that all computed pushout objects are actually words.

Let us first note that the following diagram, where the definition of the arrows r_x and o_x should be clear from the drawing (modulo the use of the letter x to denote an occurence of $\odot$), models in a projective way the application of the rule $x \to m$ to the word axb (the relabelling of the pullback object is being defined as earlier):

$$\begin{tikzcd} amb \arrow[r] \arrow[d] & axb \arrow[d, "r_x"] \\ \odot m \odot \arrow[r, "o_x"] & \odot x \odot \end{tikzcd}$$

Hence the following lemma:

Lemma 4. *Every word context-free rule can be encoded as a projective rule, hence: w-CF* $= po \subset pb =$ *CF*

Let us consider a slightly more complex rule, by considering the production defined by the pair of arrows

$$(\odot ax \odot by \odot cz \overset{o_y}{\to} \odot x \odot y \odot z, \odot ax \odot by \odot cz \overset{r_x}{\to} \odot x \odot y \odot z)$$

where $r_x(\odot) = \odot$, $r_x(ax) = x$ and so on (o_y being defined in the same way). Let o_x be the occurence $axbycz \to \odot x \odot y \odot z$.

Then the following pullback diagram :

$$\begin{tikzcd} aaxbbyccz \arrow[r, "\alpha"] \arrow[d] & \odot ax \odot by \odot cz \arrow[d, "r_x"] \\ axbycz \arrow[r, "o_x"] & \odot x \odot y \odot z \end{tikzcd}$$

computes a new word $aaxbbyccz$ and generates a new occurence $aaxbbyccz \xrightarrow{o_y \circ \alpha} \odot x \odot y \odot z$ where the production rule may be applied once more.

Lemma 5. *This rewriting system (together with an unknown erasing rule) generates* in a context free way [4] *the language* $a^n b^n c^n$.

This shows that:

Theorem 2. *w-CF*$= po \subsetneq CF = pb \subsetneq CS$

The last inequality is quite clear: projective grammars can not generate all context-sensitive grammars (CS): the rewriting mechanism does not provide any sort of pattern matching as needed for the most general CS grammars, since all new occurences are built by composition of functions and can not appear through mere juxtaposition of letters.

The question remains of the exact expressive power of projective word grammars, although a careful examination of the possible codomains for productions suggests that they can not generate anything really more complex than $a^n b^n c^n$.

[4] It is not new that the language $a^n b^n c^n$ can be generated in a context-free way, but this needs an encoding of words as string graphs (see [8]) hence needs going out of word-rewriting to use techniques from hyperedge replacement grammars within the category of hypergraphs.

4 Graphs and Hypergraphs

It is well known that categories of graphs or hypergraphs are both complete and cocomplete. Pushout and pullback can always be computed, and we therefore simply need to interpret the nature of inductive and projective graph grammars.

4.1 Inductive Hypergraph Grammars

Let $\mathcal{H}$ be the category of hypergraphs and $\mathcal{X}$ be a set of non-terminal hypergraphs. Inductive hypergraph grammars can be defined with no restrictions along the lines of section 2.3.

Theorem 3. *Inductive hypergraph grammars are exactly hyperedge replacement grammars in the sense of [8].*

4.2 Projective Graph Grammars

Projective graph grammars have not so far been studied in general.

We shall recall here the basic definitions of *pullback graph grammars*, which, while being a very restricted case, are sufficient to describe e.g. node replacement systems and to provide already intersting results (details and proofs may be found in [2,3,4,5]).

Definition 6. *A graph G is a 4-tuple $G = \langle V_G, E_G, s_G, t_G \rangle$ where the sets V_G of vertices and E_G of edges are two finite disjoint sets and s_G and t_G are mappings from E_G to V_G. For every element $e \in E_G$, $s_G(e)$ and $t_G(e)$ are called source vertex and target vertex of the edge e respectively.*

A vertex $v \in V_G$ is *reflexive* if there exists an edge $e \in E_G$ such that $s_G(e) = t_G(e) = v$. A graph G is reflexive if all its vertices are reflexive, it is said to be simple if for any pair x, y of vertices of G, there is at most one edge from x to y.

Non Terminals. Non-terminals graphs have a quite specific form which allows them to distinguish between nodes to be transformed, nodes to be identically reproduced and an intermediate zone.

Definition 7. *A* non-terminal graph *X is a graph made out of two components: a complete reflexive graph K_{m+1}, and a reflexive subgraph U linked to only m of the vertices of K_{m+1}.*

Occurrences and Productions. For the sake of simplicity (and to keep some coherence with e.g. [3]), we shall simply describe the shapes of the morphisms involved in the definition of a source $\overline{G} = (G, U_G, P_G)$, calling occurence any morphism appearing in U_G and production the morphism involved as a replacement scheme in P_G. They both will have a very special structure.

Definition 8. *Let G be a directed simple graph and X a non terminal graph, an* occurrence *x on G is a graph morphism from G to X such that the pre-image $x^{-1}(U)$ is non empty.*

Definition 9. *A* production *r is a morphism $r : R \to X$ which is isomorphic on the inverse image of the subgraph of X generated by K_{m+1}.*

Theorem 4. *Pullback graph grammar are projective graph grammars hence are context-free. For every inductive hypergraph grammar, there is an equivalent pullback graph grammar.*

The converse of the second assertion is false, since hypergraph replacement grammars cannot generate square grids ([8]), which can be generated by a pullback graph grammar with only one rule (as shown in [3]).

In [6], Courcelle shows that vertex replacement grammars (VR-grammars) are context-free. The result from [3] shows that although it satisfies the same definition of context-freeness, pullback rewriting provides us with a strictly more powerful context-free mechanism.

The real expressive power of general projective graph grammars remain to be investigated.

5 Conclusion

In this paper, we have set a generic categorical framework for rewriting, with two distinct possibilities, inductive rewriting based on the pushout operation and projective rewriting based on pullback. We have then shown that both types of categorical rewriting, either inductive or projective, are intrinsically context-free (after the well accepted definition of [6]).

This general framework has then been instantiated to two specific cases. First of all, we have shown that inductive rewriting on words mimics the classical theory of context-free word languages (generating exactly the same languages), while projective rewriting gives a more powerful notion of context-freeness, where languages such that $a^n b^n c^n$ become context-free.

In a similar way, inductive rewriting of hypergraphs describes hyperedge replacement grammars (well known to be context-free), while projective graph rewriting yields a much more powerful context-free mechanism, in which graphs languages such as the language of all complete graphs or that of square grids become context-free (as already noticed, we actually used only a very specific case of projective graph rewriting by putting strong conditions on the rules and occurences).

While we have shown that inductive rewriting describes (at least in two cases) "classical" context-free rewriting, the exact power of projective rewriting, both in the case of words or graphs remains an open question. It is also open whether projective rewriting is *always* strictly more powerfull than inductive rewriting, as is the case for words or graphs.

References

1. Adamek, J., Herrlich, H., Strecker, G.E.: Abstract and Concrete Categories, `http://katmat.math.uni-bremen.de/acc/acc.pdf`
2. Bauderon, M.: A uniform approach to graph rewriting: the pullback approach. In: Nagl, M. (ed.) WG 1995. LNCS, vol. 1017, pp. 101–115. Springer, Heidelberg (1995)

3. Bauderon, M., Chen, R., Ly, O.: Pullback Grammars Are Context-Free. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008. LNCS, vol. 5214, pp. 366–378. Springer, Heidelberg (2008)
4. Bauderon, M., Jacquet, H.: Node rewriting in graphs and hypergraphs: a categorical framework. Theoretical Computer Science 266(1-2), 463–487 (2001)
5. Chen, R.: Graph Transformation and Graph Grammar Based on Pullback Operation, PhD thesis, Université Bordeaux 1 (2007)
6. Courcelle, B.: An axiomatic approach to context-free rewriting and its application to NLC graph grammars. Theoretical Computer Science 55, 141–181 (1987)
7. Engelfriet, J., Rozenberg, G.: Node Replacement Graph Grammars. In: [9], pp. 1–94
8. Habel, A.: Hyperedge Replacement: Grammars and Languages. LNCS, vol. 643. Springer, Heidelberg (1992)
9. Rozenberg, G.: Handbook of Graph Grammars and Computing by Graph Transformation. World Scientific Publishing, Singapore (1997)

An Eilenberg Theorem for Pictures

Symeon Bozapalidis and Archontia Grammatikopoulou

Department of Mathematics
Aristotle University of Thessaloniki
54124 Thessaloniki,Greece
bozapali@math.auth.gr, arxontia@gmail.com

Abstract. Picture language recognizability by 2-monoids is shown to be equivalent to recognizability by frame action. Then we establish a bijection between the pseudovarieties of finite 2-monoids and varieties of frame recognizable picture languages.

Dedicated to Werner Kuich for his retirement.

1 Introduction

A picture of rank $(\alpha, \beta) \in \mathbb{R}_+^2$ is a rectangular array of dimensions α, β constructed by elementary rectangular pieces called *pixels*.

$Pict_{\alpha,\beta}(X)$ denotes the set of all such pictures over the pixel alphabet X.

On the set $Pict(X) = (Pict_{\alpha,\beta}(X))_{\alpha,\beta \in \mathbb{R}_+}$ of all pictures over X two natural operations are defined: the horizontal and the vertical concatenations. The first one is carried out over pictures with the same width and the other over pictures of the same length.

In order to achieve grammatical generation, we have introduced in [3] the operation of picture deformation.

It consists of associating to every $(r, s) \in (\mathbb{R}_+ - \{0\})^2$ and every pixel $x \in X$ a new pixel $x^{(r,s)}$ which results from x by multiplying its dimensions by r, s respectively.

The (r, s)-deformation of a picture p is then constructed by replacing all pixels occurring in p by their (r, s)-deformed pixels.

The above structure is the typical instance of the algebraic structure of what we call a *deformation monoid*. This is a family of sets $M = (M_{\alpha,\beta})_{\alpha,\beta \in \mathbb{R}_+}$ equipped with families of horizontal and vertical multiplications

$$ⓗ : M_{\alpha_1,\beta_1} \times M_{\alpha_1,\beta_2} \to M_{\alpha_1,\beta_1+\beta_2} \quad , \quad ⓥ : M_{\alpha_1,\beta_1} \times M_{\alpha_2,\beta_1} \to M_{\alpha_1+\alpha_2,\beta_1}$$

$(\alpha_1, \alpha_2, \beta_1, \beta_2 \in \mathbb{R}_+)$ as well as with a family of deformation operators

$$(def_M^{(r,s)})_{\alpha,\beta} : M_{\alpha,\beta} \to M_{r\alpha,s\beta}$$

which simulate the preceding structure on pictures. Actually, $Pict(X)$ is the *free deformation monoid*.

A 2-monoid is a set endowed with two monoid structures compatible to each other. Clearly any 2-monoid can be viewed as a deformation monoid: its deformation operations are the identity functions.

In the present paper we discuss two recognition devices: through frame action and through 2-monoids.

Let ξ be a pixel not in X. A frame over X is a picture $f \in \mathit{Pict}(X \cup \xi)$ with just one occurrence of ξ. The set of all such frames is denoted by $\mathit{Frame}(X)$. Substitution at ξ is independent of deformation and thus the set $\mathfrak{Frame}(X)$ of all deformation classes of frames becomes a monoid which canonically acts on the set $\mathfrak{Pict}(X)$ of all deformation classes of pictures.

A deformation closed picture language $L \subseteq \mathfrak{Pict}(X)$ is said to be *frame recognizable* whenever there exist a finite action $M \times Q \to Q$ (M monoid) and a morphism of monoid actions

$$\begin{tikzcd}
\mathfrak{Frame}(X) \times \mathfrak{Pict}(X) \arrow[r] \arrow[d, "(\varphi, f)"'] & \mathfrak{Pict}(X) \arrow[d, "f"] \\
M \times Q \arrow[r] & Q
\end{tikzcd}$$

(φ monoid morphism) such that $f^{-1}(P) = L$, for some $P \subseteq Q$.

On the other hand, if there exist a finite 2-monoid $\mathcal{M}$ and a deformation morphism $H : \mathit{Pict}(X) \to M(N)$ so that $L = H^{-1}(R)$, $R \subseteq N$ then we say that L is recognized through N.

A main result in this paper states that the two modes of recognition just stated are equivalent.

Then we use this characterization in order to establish Eilenberg's Variety Theorem in the setup of pictures. Precisely, we show that there is a bijection between pseudovarieties of finite 2-monoids and varieties of frame recognizable picture languages.

2 Recognizability through Monoid Action

Our first recognition mode is referred to monoid action.

A monoid is a set M equipped with an associative multiplication $M \times M \to M$, $(m_1, m_2) \mapsto m_1 m_2$ which admits a unit element 1.

Let us fix a monoid $(M, \cdot, 1)$. Any set Q equipped with a function $M \times Q \to Q$, $(m, q) \mapsto m \cdot q$ such that

$$m_1(m_2 q) = (m_1 m_2)q \text{ and } 1 \cdot q = q \text{ for all } q \in Q \ , \ m_1, m_2 \in M$$

is called an *M-set.*

Given M-sets Q and Q', any function $h : Q \to Q'$ such that

$$h(m \cdot q) = m \cdot h(q) \qquad \text{for all } m \in M \ , \ q \in Q$$

is called an *M-function.*

The *left derivative* of a subset L of an M-set Q at the point $q \in Q$ is

$$q^{-1}L = \{m | m \in M \ , \ mq \in L\}.$$

The set of all left derivatives of L

$$Q_L = \{q^{-1}L \mid q \in Q\}$$

with the (well defined) action

$$m(q^{-1}L) = (mq)^{-1}L \qquad (m \in M, q \in Q)$$

is structured into an M-set called the *syntactic M-set* of L.

Proposition 1. *Let L be a subset of an M-set Q. If $h : Q \to Q'$ is a surjective M-function such that $h^{-1}(h(L)) = L$, then there results a unique M-function $h' : Q' \to Q_L$ making commutative the triangle*

$$\begin{tikzcd} Q \arrow[dr, "h"] \arrow[dd, "h_L"'] & \\ & Q' \arrow[dl, "h'"] \\ Q_L & \end{tikzcd}$$

where $h_L : Q \to Q_L$ is given by $h_L(q) = q^{-1}L$.

Proof. For any $q' \in Q'$, let $q \in Q$ be such that $h(q) = q'$. Then we set $h'(q') = q^{-1}L$. The reader will verify that h' is well defined and has all the announced properties.

The *right derivative* of $L \subseteq Q$ at $m \in M$ is

$$Lm^{-1} = \{q \mid q \in Q \ , \ mq \in L\}.$$

Proposition 2. *If L has finitely many left derivatives, then it has finitely many right derivatives and vice versa.*

Proof. Assume that $q_1^{-1}L$, ... , $q_k^{-1}L$ are all the distinct left derivatives of L. This means that for all $q \in Q$ there is an index i $(1 \leqslant i \leqslant k)$ such that $q^{-1}L = q_i^{-1}L$.

Now, we define a function

$$\varphi : \{Lm^{-1} \mid m \in M\} \to \{0,1\}^k$$

by setting

$$\varphi(Lm^{-1}) = (\varepsilon_1, ..., \varepsilon_k) \text{ with } \varepsilon_i = 1 \Leftrightarrow mq_i \in L.$$

Observe first that φ is well defined i.e. $Lm^{-1} = Lm'^{-1}$ implies $(\varepsilon_1, ..., \varepsilon_k) = (\varepsilon'_1, ..., \varepsilon'_k)$.

It suffices to show that $\varepsilon_i = 1$ iff $\varepsilon'_i = 1$ $(1 \leqslant i \leqslant k)$. Indeed

$$\varepsilon_i = 1 \text{ iff } mq_i \in L \text{ iff } q \in Lm^{-1} = Lm'^{-1} \text{ iff } m'q_i \in L \text{ iff } \varepsilon'_i = 1.$$

The fact that φ is injective comes as follows: suppose that $\varepsilon_i = \varepsilon'_i$ for $i = 1, ..., k$ i.e. $mq_i \in L \Leftrightarrow m'q_i \in L$. Then

$$q \in Lm^{-1} \text{ iff } mq \in L \text{ iff } m \in q^{-1}L = q_i^{-1}L \text{ iff } mq_i \in L \text{ iff } m'q_i \in L \text{ iff }$$
$$m' \in q_i^{-1}L = q^{-1}L \text{ iff } m'q \in L \text{ iff } q \in Lm'^{-1}.$$

Thus φ is injective and so

$$card\{Lm^{-1} \mid m \in M\} \leqslant 2^k.$$

By interchanging the roles of left and right derivatives in the above argument we get the converse assertion.

A subset L of an M-set Q is said to be *recognizable* if there is a finite M-set Q' and an M-function $h : Q \to Q'$ so that $L = h^{-1}(P)$, for some $P \subseteq Q'$.

Theorem 1. *Next conditions are equivalent for a subset L of the M-set Q:*

i. *L is recognizable*
ii. $card\{q^{-1}L \mid q \in Q\} < \infty$
iii. $card\{Lm^{-1} \mid m \in M\} < \infty$

Proof. It is a combination of Proposition 1 and Proposition 2.

3 Deformation Monoids

A *picture semigroup* is a family of sets $M = (M_{\alpha,\beta})_{\alpha,\beta\in\mathbb{R}_+}$ equipped with two (families of) operations

$$ⓗ\colon M_{\alpha,\beta_1} \times M_{\alpha,\beta_2} \to M_{\alpha,\beta_1+\beta_2} \text{ (horizontal multiplication)}$$

$$ⓥ\colon M_{\alpha_1,\beta} \times M_{\alpha_2,\beta} \to M_{\alpha_1+\alpha_2,\beta} \text{ (vertical multiplication)}$$

$(\alpha, \alpha_1, \alpha_2, \beta, \beta_1, \beta_2 \in \mathbb{R}_+)$ which are associative in the obvious sense and moreover compatible with each other, i.e.

$$(aⓗa')ⓥ(bⓗb') = (aⓥb)ⓗ(a'ⓥb')$$

whenever both sides are defined $(a, a', b, b' \in M)$.

Since we deal with two-dimensional objects, we need two kinds of units with respect to the kind of multiplication we use. In a picture semigroup $M = (M_{\alpha,\beta})_{\alpha,\beta\in\mathbb{R}_+}$ we say that the family (e_α) , $e_\alpha \in M_{\alpha,0}$ $(\alpha \in \mathbb{R}_+)$ is a *horizontal unit* whenever for all $a \in M_{\alpha,\beta}$ it holds that

$$e_\alpha ⓗ a = a = a ⓗ e_\alpha \text{ and } e_\alpha ⓥ e_\beta = e_{\alpha+\beta} \quad (\alpha, \beta \in \mathbb{R}_+).$$

The family of *vertical units* (f_β) are symmetrically defined. Of course whenever a horizontal (resp. vertical) unit exists, it is unique. A picture semigroup with both horizontal and vertical units is called a *picture monoid*.

Picture monoids were introduced in [1],[2] in order to study picture codes and picture automata, respectively.

Assume that two picture monoids $M = (M_{\alpha,\beta})_{\alpha,\beta\in\mathbb{R}_+}$ and $M' = (M'_{\alpha,\beta})_{\alpha,\beta\in\mathbb{R}_+}$ are given. A *morphism of rank* (r,s), with $r,s \in \mathbf{R}_+ - \{0\}$, from M to M' is an $(\mathbf{R}_+ - \{0\})^2$-ranked family of functions

$$H = (H_{\alpha,\beta} : M_{\alpha,\beta} \to M'_{r\alpha,s\beta})_{\alpha,\beta\in\mathbf{R}_+}$$

preserving horizontal and vertical multiplications and units

$$\begin{aligned} H_{\alpha,\beta_1+\beta_2}(a \circledh a') &= H_{\alpha,\beta_1}(a) \circledh H_{\alpha,\beta_2}(a') \\ H_{\alpha_1+\alpha_2,\beta}(b \circledv b') &= H_{\alpha_1,\beta}(b) \circledv H_{\alpha_2,\beta}(b') \\ H_{\alpha,0}(e_\alpha) = e'_\alpha \qquad & H_{0,\beta}(f_\beta) = f'_\beta \end{aligned}$$

where $a \in M_{\alpha,\beta_1}, a' \in M_{\alpha,\beta_2}, b \in M_{\alpha_1,\beta}, b' \in M_{\alpha_2,\beta}$ $(\alpha,\alpha_1,\alpha_2,\beta,\beta_1,\beta_2 \in \mathbf{R}_+)$ and (e_α) , (f_β) (resp. (e'_α) , (f'_β)) are the horizontal and vertical units of M (resp. M').

Clearly, the composition of two morphisms of ranks (r,s) and (r',s') respectively is a morphism of rank (rr',ss'). The morphisms of rank $(1,1)$ are simply referred to as *morphisms of picture monoids*.

An instance of deformation monoid that will be used later on is that of a 2-monoid which is a structure $\mathbb{M} = (M, \circledh, \circledv, e, f)$ where $\circledh, \circledv$: $M^2 \to M$ are two associative operations admitting e,f respectively as unit elements.

Moreover, we demand that $\circledh, \circledv$ satisfy the coherence condition

$$(m_1 \circledh m_2) \circledv (m'_1 \circledh m'_2) = (m_1 \circledv m'_1) \circledh (m_2 \circledv m'_2)$$

for all $m_i, m'_i \in M$, $i = 1,2$.

Furthermore, let $M = (M_{\alpha,\beta})_{\alpha,\beta\in\mathbb{R}_+}$ be a picture monoid and $\sim = (\sim_{\alpha,\beta})_{\alpha,\beta\in\mathbb{R}_+}$ be an equivalence relation on M compatible with horizontal and vertical multiplications

$$\begin{array}{lll} a \sim_{\alpha,\beta_1} a' \text{ and } b \sim_{\alpha,\beta_2} b' & \text{implies} & a \circledh b \sim_{\alpha,\beta_1+\beta_2} a' \circledh b' \\ a \sim_{\alpha_1,\beta} a' \text{ and } b \sim_{\alpha_2,\beta} b' & \text{implies} & a \circledv b \sim_{\alpha_1+\alpha_2,\beta} a' \circledv b' \end{array}$$

for all $a, a', b, b' \in M$ of suitable rank. Then we say that $\sim$ is a *congruence* on M.

The quotient $^{M}/_{\sim}$ can be organized into a picture monoid in the obvious way:

$$\overline{a} \circledh \overline{b} = \overline{a \circledh b} \quad , \quad \overline{a'} \circledv \overline{b'} = \overline{a' \circledv b'}$$

where $\overline{a}$ stands for the $\sim$-class of a and a, a', b, b' are elements of M with appropriate rank. It is called the *quotient picture monoid* of M by $\sim$.

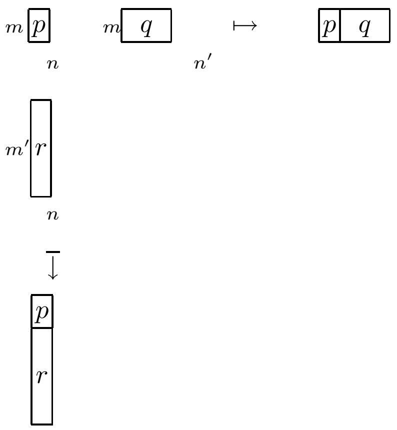
Let X be a finite (pixel) alphabet. A *picture of rank* (m,n) over X is just an $m \times n$ matrix with entries in X:

$$p = \begin{matrix} x_{11} & \cdots & x_{1n} \\ \vdots & & \vdots \\ x_{m1} & \cdots & x_{mn} \end{matrix} \qquad x_{ij} \in X.$$

We denote by $pict_{m,n}(X)$ the set of all such pictures.

Pictures can be composed in two ways: horizontally and vertically. More precisely, the *horizontal concatenation* of an (m, n)-picture p with an (m, n')-picture q is the $(m, n + n')$-picture pq obtained by writing q on the right of p.

The *vertical concatenation* of an (m, n)-picture p with an (m', n)-picture r is the $(m + m', n)$-picture $\begin{pmatrix} p \\ r \end{pmatrix}$ obtained by writing r on the bottom of p.



Now we are going to introduce our basic algebraic structure.

A *deformation monoid* (DM) is a pair $\mathcal{M} = (M, def_M)$ consisting of a picture monoid M and a family of morphisms of rank (r, s)

$$def_M^{(r,s)} : M \to M \qquad , \ r, s \in \mathbb{R}_+ - \{0\}$$

called the (r, s)-*deformation operator*, verifying the equalities:

$$def_M^{(r,s)} \circ def_M^{(r',s')} = def_M^{(rr',ss')} \qquad , \qquad def_M^{(1,1)} = id_M$$

where id_M stands for the identity function on M.

Apparently a 2-monoid $\mathbb{M}$ can be viewed as a deformation monoid $N(\mathbb{M})$ by setting $N(\mathbb{M})_{\alpha,\beta} = M$ for all $\alpha, \beta \in \mathbb{R}_+$ whereas its deformation operators coincide with the identity function on M.

Given two deformation monoids $\mathcal{M} = (M, def_M)$ and $\mathcal{M}' = (M', def_{M'})$, any morphism of picture monoids $H : M \to M'$ commuting with deformation i.e.

$$def_{M'}^{(r,s)} \circ H_{\alpha,\beta} = H_{r\alpha,s\beta} \circ def_M^{(r,s)} \qquad \alpha, \beta \in \mathbb{R}_+ \ , \ r, s \in \mathbb{R}_+ - \{0\}$$

is termed a *morphism of deformation monoids* (DM morphism).

Next we are going to construct the free deformation monoid generated by a pixel alphabet X.

Denote by $P(X) = (P_{\alpha,\beta}(X))_{\alpha,\beta\in\mathbb{R}_+}$ the least $\mathbb{R}_+ \times \mathbb{R}_+$- indexed family of sets formally constructed by the following items:

i. $X \subseteq P_{1,1}(X)$
ii. if $p_1 \in P_{\alpha,\beta_1}(X)$ and $p_2 \in P_{\alpha,\beta_2}(X)$, then their horizontal concatenation $p_1p_2 \in P_{\alpha,\beta_1+\beta_2}(X)$, $\alpha, \beta_1, \beta_2 \in \mathbb{R}_+$
iii. if $p_1 \in P_{\alpha_1,\beta}(X)$ and $p_2 \in P_{\alpha_2,\beta}(X)$ then their vertical concatenation $\begin{pmatrix} p_1 \\ p_2 \end{pmatrix} \in P_{\alpha_1+\alpha_2,\beta}(X)$, $\alpha_1, \alpha_2, \beta \in \mathbb{R}_+$
iv. the horizontal and vertical *empty pictures* of rank $\alpha \in \mathbb{R}_+$

$$\varepsilon_\alpha \in P_{\alpha,0}(X) \text{ and } \zeta_\alpha \in P_{0,\alpha}(X)$$

play the role of units for the above two concatenations($\varepsilon_0 = \zeta_0$)
v. if $p \in P_{\alpha,\beta}(X)$, then $p^{(r,s)} \in P_{r\alpha,s\beta}(X)$, for all $r, s \in \mathbb{R}_+ - \{0\}$, $\alpha, \beta \in \mathbb{R}_+$.

The items *i.* – *iv.* ensure that $P(X)$ is a picture monoid containing X whose operations are horizontal and vertical concatenations.

Consider the congruence $\sim$ on $P(X)$ generated by the relations

d_1. $p^{(1,1)} \sim p \quad , \quad (p^{(r,s)})^{(r',s')} \sim p^{(rr',ss')}$
d_2. $(p_1p_2)^{(r,s)} \sim p_1^{(r,s)}p_2^{(r,s)}$
d_3. $\begin{pmatrix} p_1 \\ p_2 \end{pmatrix}^{(r,s)} \sim \begin{pmatrix} p_1^{(r,s)} \\ p_2^{(r,s)} \end{pmatrix}$

for all $r, s, r', s' \in \mathbb{R}_+ - \{0\}$ and all $p, p_1, p_2 \in P(X)$ of suitable rank. Then the quotient $Pict(X) = {}^{P(X)}/_{\sim}$ is a deformation monoid: the deformation operation associated with $(r, s) \in (\mathbb{R}_+ - \{0\})^2$ is given by the mapping

$$\overline{p} \;\mapsto\; \overline{p^{(r,s)}}$$

where $\overline{p}$ denotes the $\sim$-class of p.

Clearly any element p of $Pict_{\alpha,\beta}(X)$ can be represented by a picture of rank (α, β) constructed by the deformed pixels $x^{(r,s)}$ ($x \in X$, $r, s \in \mathbb{R}_+ - \{0\}$) whereas $p^{(r,s)}$ is the picture obtained by substituting any deformed pixel $x^{(\gamma,\delta)}$ in p by $x^{(r\gamma,s\delta)}$, $\alpha, \beta, \gamma, \delta \in \mathbb{R}_+$.

A *picture language of rank* (α, β) over X is a subset of $Pict_{\alpha,\beta}(X)$ ($\alpha, \beta \in \mathbb{R}_+$). $Pict(X)$ is the free deformation monoid generated by X, as next theorem confirms.

Theorem 2. *The function $j : X \to Pict_{1,1}(X)$, $j(x) = x$ has the following universal property: for any deformation picture monoid $\mathcal{M} = (M, def_M)$ and any function $f : X \to M_{1,1}$ there is a unique morphism of deformation monoids $\hat{f} : Pict(X) \to \mathcal{M}$ rendering commutative the diagram*

$$\begin{tikzcd}
X \arrow[dr, "j"] \arrow[dd, "f"'] & \\
 & Pict(X). \arrow[dl, "\hat{f}"] \\
\mathcal{M} &
\end{tikzcd}$$

The morphism $\hat{f}$ *is defined by the clauses:*

- $\hat{f}(x) = f(x) \quad , x \in X$
- $\hat{f}(p_1p_2) = \hat{f}(p_1)\textcircled{h}\hat{f}(p_2)$
- $\hat{f}\left(\begin{pmatrix} p_1 \\ p_2 \end{pmatrix}\right) = \hat{f}(p_1)\textcircled{v}\hat{f}(p_2)$
- $\hat{f}(p^{(r,s)}) = def_M^{(r,s)}(\hat{f}(p))$

for all $p, p_1, p_2 \in P(X)$ *and* $(r,s) \in \mathbb{R}_+ - \{0\}$.

4 Frame Recognizability

Now we shall apply the previous data to the setup of pictures. Consider a pixel alphabet X and an auxiliary pixel $\xi \notin X$.

The *deformation equivalence* $\sim_{def}$ is defined on the set

$$\bigcup_{\alpha,\beta \in \mathbb{R}_+} Pict_{\alpha,\beta}(X)$$

as follows:

$$p \sim_{def} q \text{ iff } q = p^{(r,s)} \text{ for some } r,s \in \mathbb{R}_+ - \{0\}.$$

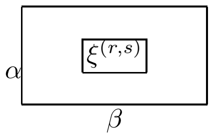
The quotient set

$$\mathfrak{Pict}(X) = (\bigcup_{\alpha,\beta \in \mathbb{R}_+} Pict_{\alpha,\beta}(X))/\sim_{def}$$

has as elements the $\sim_{def}$-classes of the pictures over X, i.e. the elements of $\mathfrak{Pict}(X)$ are of the form

$$\overline{p} = \{p^{(r,s)} \mid r,s \in \mathbb{R}_+ - \{0\}\}.$$

Next we introduce the monoid of frames. The set of *frames* with *exterior rank* (α,β) and *interior rank* (r,s) is the subset $Frame^{r,s}_{\alpha,\beta}(X)$ of $Pict_{\alpha,\beta}(X \cup \xi)$ consisting of all pictures with just one occurrence of a deformation of ξ, namely $\xi^{(r,s)}$

$f =$ 

$r \leqslant \alpha \; , \; s \leqslant \beta.$

Given frames

$$f \in Frame^{r,s}_{\alpha,\beta}(X) \qquad \text{and} \qquad f' \in Frame^{\gamma,\delta}_{r,s}(X)$$

their composition $f \circ f'$ is the frame obtained by substituting f' at $\xi^{(r,s)}$ in f.

In general, if $f_i \in Frame^{r_i,s_i}_{\alpha_i,\beta_i}(X)$ we define the product

$$f_1 \cdot f_2 = f_1 \circ f_2^{(\frac{r_1}{\alpha_2},\frac{s_1}{\beta_2})}$$

For $f'_i \sim_{def} f_i$ $(i = 1,2)$, it holds that $f_1 \cdot f_2 \sim_{def} f'_1 \cdot f'_2$. It turns out that the quotient set

$$\mathfrak{Frame}(X) = \left(\bigcup_{\alpha,\beta \in \mathbb{R}_+ - \{0\}} Frame^{r,s}_{\alpha,\beta}(X) \right) / \sim_{def}$$

with multiplication $\overline{f_1} \cdot \overline{f_2} = \overline{f_1 \cdot f_2}$ becomes a monoid which canonically acts on $\mathfrak{Pict}(X)$: $\overline{f} \cdot \overline{p} = \overline{f \cdot p}$. In other words $\mathfrak{Pict}(X)$ is a $\mathfrak{Frame}(X)$-set and thus we can speak of recognizable subsets $\mathcal{L}$ of $\mathfrak{Pict}(X)$.

The left and right derivatives of $\mathcal{L} \subseteq \mathfrak{Pict}(X)$ are

$$\overline{p}^{-1}\mathcal{L} = \{\overline{f} \mid \overline{f} \in \mathfrak{Frame}(X)\ ,\ \overline{f} \cdot \overline{p} = \overline{f \cdot p} \in \mathcal{L}\}$$

$$\mathcal{L}\overline{f}^{-1} = \{\overline{p} \mid \overline{p} \in \mathfrak{Pict}(X)\ ,\ \overline{f} \cdot \overline{p} = \overline{f \cdot p} \in \mathcal{L}\}.$$

By applying Theorem 1 in the present setup we obtain the following result.

Proposition 3. *Next conditions are equivalent for a subset $\mathcal{L}$ of $\mathfrak{Pict}(X)$:*

i. there is a finite $\mathfrak{Frame}(X)$-set Q and a $\mathfrak{Frame}(X)$-function $h : \mathfrak{Pict}(X) \to Q$ so that $\mathcal{L} = h^{-1}(P)$, for some $P \subseteq Q$,
ii. $card\{\overline{p}^{-1}\mathcal{L} \mid \overline{p} \in \mathfrak{Pict}(X)\} < \infty$
iii. $card\{\mathcal{L}\overline{f}^{-1} \mid \overline{f} \in \mathfrak{Frame}(X)\} < \infty$.

We call $\mathcal{L} \subseteq \mathfrak{Pict}(X)$ *frame recognizable* whenever it satisfies one (and thus all) of the above conditions *i-iii*.

An immediate consequence of Proposition 3 concerns closure properties.

Proposition 4. *The frame recognizable subsets of $\mathfrak{Pict}(X)$ are closed under the boolean operations.*

A picture language $L \subseteq \mathfrak{Pict}(X)$ is said to be *deformation closed* whenever

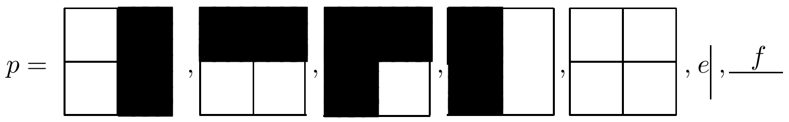
$$p \in L \quad \text{and} \quad p' \sim_{def} p \qquad \text{implies} \qquad p' \in L.$$

For a deformation closed picture language $L \subseteq Pict(X)$ its right and left derivatives are defined to be the corresponding derivatives of the associated set $\overline{L}$:

$$Lp^{-1} = \overline{L}\overline{p}^{-1}\ ,\ f^{-1}L = \overline{f}^{-1}L \quad \text{for all} \quad p \in Pict(X)\ ,\ f \in Frame(X)$$

A deformation closed picture language $L \subseteq Pict(X)$ is called *frame recognizable* whenever $\overline{L} = \{\overline{p} \mid p \in L\} \subseteq \mathfrak{Pict}(X)$ is frame recognizable.

Example 1. Let $X = \{\square, \blacksquare\}$ and consider the language $L \subseteq Pict(X)$ consisting of all pictures having black pixels along their north-western faces. L is obviously deformation-closed and has seven distinct left derivatives $p^{-1}L$ with:

$p =$ 

where e, f are the empty horizontal and vertical pixels respectively.
Thus L is frame recognizable.

5 Recognizability through Picture Monoids

In [5] Matz raised the question whether the word language recognizability through monoids can be transferred into the framework of pictures. In the present section we deal with this problem.

A deformation monoid $M = (M_{\alpha,\beta})_{\alpha,\beta \in \mathbb{R}_+}$ is said to be

- *locally finite* whenever the set $M_{\alpha,\beta}$ is finite for all indices $\alpha, \beta \in \mathbb{R}_+$
- *finite* whenever the set
$$\bigcup_{\alpha,\beta \in \mathbb{R}_+} M_{\alpha,\beta}$$
is finite
- *reachable* if there is a finite pixel alphabet X and a deformation morphism $H : Pict(X) \to M$ which is locally surjective, i.e. all the functions $H_{\alpha,\beta} : Pict_{\alpha,\beta}(X) \to M_{\alpha,\beta}$ are surjective. This implies that any element $m \in M_{\alpha,\beta}$ $(\alpha, \beta \in \mathbb{R}_+)$ can be obtained from a list of elements $def_M^{(r_1,s_1)}(m_1)$, ... ,$def_M^{(r_k,s_k)}(m_k)$ (with $m_1, ..., m_k \in M_{1,1}$) by applying the operations of horizontal and vertical multiplication.

Since $\mathbb{R}_+ - \{0\}$ is a multiplicative group, the deformation operator $def_M^{(r,s)} : M_{\alpha,\beta} \to M_{r\alpha,s\beta}$ is bijective and its inverse is $def_M^{(\frac{1}{r},\frac{1}{s})} : M_{r\alpha,s\beta} \to M_{\alpha,\beta}$.

Now, a picture language $L \subseteq Pict(X)$ is *recognizable* if there is a locally finite deformation monoid M and a deformation morphism $H : Pict(X) \to M$ so that $L = h^{-1}(R)$, with $R \subseteq M$ (i.e. $R_{\alpha,\beta} \subseteq M_{\alpha,\beta}$ for all $\alpha, \beta \in \mathbb{R}_+$).

We have next nice result.

Theorem 3. *A deformation closed language $L \subseteq Pict(X)$ is frame recognizable if and only if there is a finite deformation monoid $M = (M_{\alpha,\beta})$ whose deformation operator $def_M^{(r,s)} : M_{1,1} \to M_{r,s}$ $(r, s \in \mathbb{R}_+ - \{0\})$ satisfies the condition*

$$(c) \qquad def_M^{(r,s)}(m) = m \qquad , \quad \text{for all } m \in M_{1,1}$$

and a deformation morphism $H : Pict(X) \to M$ such that $L = H^{-1}(R)$ for some $R \subseteq M$.

Proof. Assume that L is frame recognizable. This means that there exist a finite $\mathfrak{Frame}(X)$-set Q and a $\mathfrak{Frame}(X)$-function $\theta : \mathfrak{Pict}(X) \to Q$ so that $\overline{L} = \theta^{-1}(Q')$ for some $Q' \subseteq Q$.

For all $\alpha, \beta \in \mathbb{R}_+$ we set

$$M_{\alpha,\beta} = \{\theta(\overline{p}) \mid p \in Pict_{\alpha,\beta}(X)\}$$

and we define the horizontal multiplication

$$\textcircled{h} : M_{\alpha,\beta_1} \times M_{\alpha,\beta_2} \to M_{\alpha,\beta_1+\beta_2}$$

as follows: if $m_i = \theta(\overline{p}_i), p_i \in Pict_{\alpha,\beta_i}(X)$ $(i = 1, 2)$ then

$$m_1 \textcircled{h} m_2 = \theta(\overline{p_1 p_2}). \tag{1}$$

This formula is consistent. Indeed, let $p_i' \in Pict_{\alpha,\beta_i}(X)$, $i = 1, 2$ with

$$\theta(\overline{p_i'}) = \theta(\overline{p_i}) \ , \ i = 1, 2 \tag{2}$$

and consider the frames

$$f = \begin{array}{|c|c|}\hline p_1 & \xi^{(\alpha,\beta_2)} \\ \hline \end{array} \ ,$$

$$f' = \begin{array}{|c|c|}\hline \xi^{(\alpha,\beta_1)} & p_2' \\ \hline \end{array} \ .$$

Then

$$\begin{aligned}\theta(\overline{p_1 p_2}) &= \theta(\overline{f} \cdot \overline{p_2}) = \overline{f} \cdot \theta(\overline{p_2}) \overset{(2)}{=} \overline{f} \cdot \theta(\overline{p_2'}) = \theta(\overline{f} \cdot \overline{p_2'}) = \theta(\overline{f \cdot p_2'}) \\ &= \theta(\overline{p_1 p_2'}) = \theta(\overline{f'} \cdot \overline{p_2}) = \overline{f'} \cdot \theta(\overline{p_1}) = \overline{f'} \cdot \theta(\overline{p_2}) = \theta(\overline{p_1' p_2'}).\end{aligned}$$

This operation is associative since for all m_i $(i = 1, 2, 3)$ of suitable rank we have $m_i = \theta(\overline{p_i})$ $(i = 1, 2, 3)$ and so

$$\begin{aligned}m_1 \textcircled{h} (m_2 \textcircled{h} m_3) &= \theta(\overline{p_1}) \textcircled{h} (\theta(\overline{p_2}) \textcircled{h} \theta(\overline{p_3})) = \theta(\overline{p_1}) \textcircled{h} \theta(\overline{p_2 p_3}) = \theta(\overline{p_1(p_2 p_3)}) \\ &= \theta(\overline{(p_1 p_2) p_3}) = (m_1 \textcircled{h} m_2) \textcircled{h} m_3.\end{aligned}$$

The vertical multiplication is obtained in a similar way:

$$m_1 \textcircled{v} m_2 = \theta\left(\overline{\begin{pmatrix} p_1 \\ p_2 \end{pmatrix}}\right).$$

Finally, for all $r, s \in \mathbb{R}_+ - \{0\}$ the formula

$$def_M^{(r,s)}(m) = \theta(\overline{p^{(r,s)}}) \tag{3}$$

defines a deformation operator on $M = (M_{\alpha,\beta})$. Thus M is a finite deformation monoid satisfying the condition (c).

The mapping $H_{\alpha,\beta} : Pict_{\alpha,\beta}(X) \to M_{\alpha,\beta}$, $p \mapsto \theta(\overline{p})$ is by construction a deformation morphism.

Now by taking $P_{\alpha,\beta} = Q' \cap M_{\alpha,\beta}$ $(\alpha, \beta \in \mathbb{R}_+)$ we get for every $p \in Pict_{\alpha,\beta}(X)$

$$\begin{aligned} &p \in H^{-1}_{\alpha,\beta}(P_{\alpha,\beta}) \quad \text{iff} \quad H_{\alpha,\beta}(p) \in P_{\alpha,\beta} \quad \text{iff} \quad \theta(\overline{p}) \in P_{\alpha,\beta} \\ &\text{iff} \quad \theta(\overline{p}) \in Q' \quad \text{iff} \quad \overline{p} \in \overline{L} \quad \text{iff} \quad p \in L \end{aligned}$$

as wanted.

In order to establish the converse let $M = (M_{\alpha,\beta})$ be a finite deformation monoid satisfying (c) and $H : Pict(X) \to M$ a deformation morphism such that $L = H^{-1}(P)$, for some $P \subseteq M$ (i.e. $P_{\alpha,\beta} \subseteq M_{\alpha,\beta}$ for all $\alpha, \beta \in \mathbb{R}_+$).

Without any loss of generality we may assume that H is locally surjective.

For all $r, s \in \mathbb{R}_+ - \{0\}$ the function $def_M^{(r,s)} : M_{1,1} \to M_{r,s}$ is a bijection and so by virtue of (c), $M_{1,1} = M_{r,s}$ and $def_M^{(r,s)}$ is the identity function.

Therefore, from the next commutative triangle

$$\begin{tikzcd} M_{1,1} \arrow[dr, "def_M^{(r,s)}"] \arrow[dd, "def_M^{(r',s')}"'] & \\ & M_{r,s} \arrow[dl, "def_M^{(\frac{r'}{r},\frac{s'}{s})}"] \\ M_{r',s'} & \end{tikzcd}$$

we obtain that $def_M^{(\frac{r'}{r},\frac{s'}{s})}$ is also the identity function.

Fact. Let $p \in Pict_{r,s}(X)$, $p' \in Pict_{r',s'}(X)$ and $p \sim_{def} p'$. Then

$$H_{r,s}(p) = H_{r',s'}(p').$$

The monoid $\mathfrak{Frame}(X)$ acts on the set $Q = M_{1,1} \cup \{e, f\}$ as follows: for $f \in Frame^{r,s}_{\alpha,\beta}(X)$ and $m \in M_{1,1}$ we set $f \cdot m = H_{\alpha,\beta}(f \cdot p)$, where $p \in Pict_{r,s}(X)$ is such that $H_{r,s}(p) = m$.

If $p' \in Pict_{r,s}(X)$ and $H_{r,s}(p') = m$ the fact that H is a morphism guarantees that $H_{\alpha,\beta}(f \cdot p) = H_{\alpha,\beta}(f \cdot p')$.

Now, for $f' \sim_{def} f$, $f' \in Frame^{r',s'}_{\alpha',\beta'}$ then $f' \cdot p \sim_{def} f \cdot p$ and so by the previous fact we get $H_{\alpha,\beta}(f' \cdot p) = H_{\alpha,\beta}(f \cdot p)$.

Hence, formula (3) is legitimate.

Furthermore, we define the function $\theta : \mathfrak{Pict}(X) \to Q$ by setting

$$\theta(\overline{p}) = H_{\alpha,\beta}(p) \ \ , \ p \in Pict_{\alpha,\beta}(X). \tag{4}$$

Formula (4) is also consistent because if $\overline{p'} = \overline{p}$, i.e. $p' \sim_{def} p$, then by virtue of our fact, $H_{\alpha,\beta}(p) = H_{\alpha',\beta'}(p')$.

The equality $h(\overline{f} \cdot \overline{p}) = \overline{f} \cdot h(\overline{p})$ comes directly from the above considerations.

We conclude that $\overline{L} = \theta^{-1}(p)$, that is L is recognizable.

Given a 2-monoid $\mathbb{M}=(M,ⓗ,ⓥ,e,f)$ a *morphism* from $Pict(X)$ to $\mathbb{M}$ is a family of functions

$$H_{\alpha,\beta} : Pict_{\alpha,\beta}(X) \to M \quad , \ \alpha, \beta \in \mathbb{R}_+$$

which are compatible with horizontal, vertical concatenations and units

$$\begin{aligned} H_{\alpha,\beta_1+\beta_2}(p_1 p_2) &= H_{\alpha,\beta_1}(p_1) ⓗ H_{\alpha,\beta_2}(p_2) \\ H_{\alpha_1+\alpha_2,\beta}\left(\begin{pmatrix} q_1 \\ q_2 \end{pmatrix}\right) &= H_{\alpha_1,\beta}(q_1) ⓥ H_{\alpha_2,\beta}(q_2) \\ H_{1,0}(\varepsilon) &= e \\ H_{0,1}(\varphi) &= f \end{aligned}$$

and respect deformation

$$p \sim_{def} p' \quad \text{implies} \quad H_{\alpha,\beta}(p) = H_{\alpha',\beta'}(p').$$

Then we can state

Theorem 4. *A deformation closed language $L \subseteq Pict(X)$ is frame recognizable if and only if there exist a finite 2-monoid $\mathbb{M}=(M,ⓗ,ⓥ,e,f)$ and a morphism $H : Pict(X) \to \mathbb{M}$ so that*

$$L = H^{-1}(P) \quad , \text{for some } P \subseteq M.$$

Proof. The one direction results by the argument of Theorem 3 and the opposite direction is straightforward.

6 Syntactic 2-Monoids

In the present section we investigate the minimization problem relative to recognizability through 2-monoids. Since $\mathbb{R}_+ - \{0\}$ is a multiplicative group, all deformation operators of $Pict(\Sigma)$ are bijective and so if $H : Pict(\Sigma) \to M$ is a morphism of deformation monoids (M is a 2-monoid) then for all $r, s \in \mathbb{R}_+ - \{0\}$ the language $H^{-1}_{r,s}(Q)$, $Q \subseteq M$, is the (r,s)-deformation of the language $H^{-1}_{1,1}(Q)$. In other words $H^{-1}(Q)$ is completely determined by $H^{-1}_{1,1}(Q)$.

Next, given a deformation-closed picture language $L \subseteq Pict(\Sigma)$, the set of its right derivatives

$$M_L = \{Lp^{-1} \mid p \in Pict(\Sigma)\}$$

can be canonically converted into a 2-monoid by defining the horizontal and vertical multiplication via the formulas

$$(Lp_1^{-1}) ⓗ (Lp_2^{-1}) = L(p_1 p_2)^{-1} \quad , \quad (Lq_1^{-1}) ⓥ (Lq_2^{-1}) = L \begin{pmatrix} q_1 \\ q_2 \end{pmatrix}^{-1}$$

where p_1, p_2 and q_1, q_2 above can be chosen to have ranks (α_1, β_1), (α_1, β_2) and (α_1, β_1), (α_2, β_1) respectively.

The canonical deformation morphism $H_L : Pict(\Sigma) \to M_L$, $H_L(p) = Lp^{-1}$ is clearly surjective and verifies the equation $H^{-1}(H(L)) = L$.

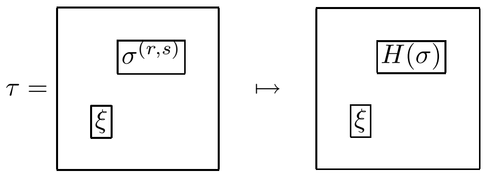
Actually H_L is universal with the above property in the following sense.

Theorem 5. *Let $H : Pict(\Sigma) \to M$ be a deformation epimorphism (M a 2-monoid) such that $H^{-1}(H(L)) = L$. Then there exists a unique epimorpism of 2-monoids $H' : M \to M_L$ making commutative the triangle*

$$\begin{tikzcd}
Pict(\Sigma) \arrow[dr, "H"] \arrow[dd, "H_L"'] & \\
& M \arrow[dl, "H'"] \\
M_L &
\end{tikzcd}$$

Proof. In order to achieve the proof we need some preliminary matter.

First, let us observe that the deformation morphism $H : Pict(\Sigma) \to M$ induces a monoid morphism $\mathfrak{Frame}(H) : \mathfrak{Frame}(\Sigma) \to \mathfrak{Frame}(M)$ which sends every frame τ over Σ into the frame over M obtained by replacing every deformed pixel $\sigma^{(r,s)}$ by $H(\sigma)$



The monoid $\mathfrak{Frame}(M)$ acts on the set M as follows: for any frame π over M and any element $m \in M$, $\pi \cdot m$ is the element of M obtained by replacing ξ by m in π and then taking the valuation of the resulting picture of $Pict(M)$

$$\pi \cdot m = val_M(\pi[m/\xi]).$$

Now, $\mathfrak{Frame}(\Sigma)$ acts also on M by setting

$$\tau \cdot m = \mathfrak{Frame}(\tau) \cdot m \quad \text{for all } \tau \in \mathfrak{Frame}(\Sigma)\ ,\ m \in M.$$

By construction we have for every $\tau \in \mathfrak{Frame}(\Sigma)$

$$H(\tau \cdot p) = \tau \cdot H(p) \qquad p \in Pict(\Sigma). \tag{5}$$

From the equality $L = H^{-1}(H(L))$ we get

$$p \in L \quad \text{iff} \quad H(p) \in H(L). \tag{6}$$

Since H is surjective, each element $m \in M$ is written as $m = H(p)$, for some $p \in Pict(\Sigma)$.

We set $H'(m) = Lp^{-1}$. We are going to show that H' is a well defined function, i.e.

$$H(p_1) = H(p_2) \quad \text{implies} \quad Lp_1^{-1} = Lp_2^{-1}.$$

Indeed, for all frames $\tau \in \mathfrak{Frame}(\Sigma)$ it holds

$$\tau \in Lp_1^{-1} \Leftrightarrow \tau \cdot p_1 \in L \overset{(6)}{\Leftrightarrow} H(\tau \cdot p_1) \in H(L) \overset{(5)}{\Leftrightarrow} \tau \cdot H(p_1) \in H(L) \Leftrightarrow \tau \cdot H(p_2) \in H(L)$$
$$\Leftrightarrow H(\tau \cdot p_2) \in H(L) \overset{(6)}{\Leftrightarrow} \tau \cdot p_2 \in L \Leftrightarrow \tau \in Lp_2^{-1}$$

as wanted. Clearly H' preserves the horizontal and vertical multiplications and units and it is surjective.

Its uniqueness is immediate.

Given 2-monoids M, M' we write $M < M'$ whenever M is a surjective image of a 2-submonoid of M'.

With this notation, we have

Proposition 5. *Let $L_1, L_2, L \subseteq Pict(\Sigma)$ be deformation closed. Then*

$$M_{L_1 \cup L_2} < M_{L_1} \times M_{L_2} \text{ , } M_{L_1 \cap L_2} < M_{L_1} \times M_{L_2} \text{ , } M_{L^c} = M_L \text{ , } M_{\tau^{-1}L} < M_L$$

$\tau \in \mathfrak{Frame}(\Sigma)$, *where L^c is the set theoretic complement of L.*

Moreover, if $F : Pict(\Delta) \to Pict(\Sigma)$ is a homomorphism of deformation monoids, then $M_{F^{-1}(L)} < M_L$.

Proof. A routine application of Theorem 5.

Remark 1. The theory of syntactic 2-monoids could be linked with the general theory of syntactic algebras, and especially with the many-sorted version presented in [6].

7 The Variety Theorem

In the framework of picture languages an analogue of the nice Eilenberg variety theorem can be established. More precisely, we show that there is a bijection between the varieties of frame recognizable picture languages and the class of pseudovarieties of finite 2-monoids.

To establish thee above result actually we adapt the arguments of Eilenberg (cf. [4]).

A class of finite 2-monoids closed under isomorphism is called a *pseudovariety* of 2-monoids whenever next axioms are fulfilled:

*pv*1) If $M_1, ..., M_k \in \mathbb{V}$, then $M_1 \times ... \times M_k \in \mathbb{V}$.
*pv*2) If $g : M' \to M$ is a monomorphism of 2-monoids and $M \in \mathbb{V}$, then $M' \in \mathbb{V}$.
*pv*3) If $h : M \to M''$ is an epimorphism of 2-monoids and $M \in \mathbb{V}$, then $M'' \in \mathbb{V}$.

Clearly *pv*1) + *pv*2) can be replaced by the single axiom

pv) If $M' < M$ and $M \in \mathbb{V}$ then $M' \in \mathbb{V}$

where the symbol $<$ was introduced in the previous section.

The intersection of any family of pseudovarieties of 2-monoids is again a pseudovariety of 2-monoids so we can speak of the pseudovariety generated by a class V of finite 2-monoids. It is denoted by $< V >$.Clearly

Proposition 6. *It holds*

$$M \in < V > \quad \textit{iff} \quad M < M_1 \times ... \times M_k \ \textit{ with } M_i \in V \ (i = 1, ..., k).$$

Proposition 7. *Each pseudovariety* $\mathbb{V}$ *is generated by the syntactic 2-monoid it contains.*

Proof. Let $M \in \mathbb{V}$ and denote by $\Sigma(M)$ a pixel alphabet in bijection with M, $b : \Sigma(M) \tilde{\to} M$. By Theorem 2 b is uniquely extended into a morphism of deformation monoids

$$\overline{b} : Pict(\Sigma(M)) \to M.$$

We set $L_m = \overline{b}^{-1}(m)$, $m \in M$. Since $H^{-1}(H(L_m)) = L_m$ there results an epimorphism $b_m : M \to M_{L_m}$, $m \in M$ (Theorem 5). The induced morphism

$$B : M \to \prod_{m \in M} M_{L_m} \quad , \quad B(a) = (b_m(a))_{m \in M}$$

is obviously a monomorphism. Since $M_{L_m} \in \mathbb{V}$ for all $m \in M$ the cartesian product $\prod_{m \in M} M_{L_m}$ is also in $\mathbb{V}$ and so M belongs to the variety generated by the syntactic 2-monoids belonging in $\mathbb{V}$. The result follows.

Now, assume that for every finite pixel alphabet Σ, a family $\mathcal{P}(\Sigma)$ of frame recognizable picture languages is given, so that

*lv*1) $\mathcal{P}(\Sigma)$ is closed under boolean operations(union, intersection, complement)
*lv*2) $\mathcal{P}(\Sigma)$ is closed under right derivatives

$$\tau \in \mathfrak{Frame}(\Sigma) \ , \ L \in \mathcal{P}(\Sigma) \ \text{ implies } \ \tau^{-1}L \in \mathcal{P}(\Sigma)$$

*lv*3) for any deformation homomorphism $F : Pict(\Delta) \to Pict(\Sigma)$ we have

$$L \in \mathcal{P}(\Sigma) \ \text{ implies } \ F^{-1}(L) \in \mathcal{P}(\Delta).$$

Then we say that the family $(\mathcal{P}(\Sigma))_\Sigma$ is a *variety of frame recognizable languages.*

Proposition 8. *Let* $\mathcal{P} = (\mathcal{P}(\Sigma))_\Sigma$ *be a variety of frame recognizable languages. If* $L \in \mathcal{P}(\Sigma)$ *and* $p \in Pict(\Sigma)$ *then the equivalence class of* p, $[p] = H_L^{-1}(H_L(p))$ *is also in* $\mathcal{P}(\Sigma)$.

Proof. We only have to observe that

$$[p] = \bigcap_{\tau p \in L} \tau^{-1}L - \bigcup_{\tau p \notin L} \tau^{-1}L$$

and take into account *lv*1) and *lv*2).

Now we are ready to define the main correspondences. To each variety $\mathcal{P} = (\mathcal{P}(\Sigma))_\Sigma$ of frame recognizable languages we associate the pseudovariety of 2-monoids $\mathbb{V}_\mathcal{P}$ which is generated by the syntactic 2-monoid of the languages of $\mathcal{P}$.

In the opposite direction, to each pseudovariety $\mathbb{V}$ of finite 2-monoids we associate the class $\mathcal{P}_\mathbb{V} = (\mathcal{P}_\mathbb{V}(\Sigma))_\Sigma$ such that $L \in \mathcal{P}_\mathbb{V}(\Sigma)$ iff $M_L \in \mathbb{V}$. By virtue of Proposition 5, $\mathcal{P}_\mathbb{V}$ is a variety of frame recognizable picture languages.

Theorem 6. *The assignments*

$$\mathcal{P} \mapsto \mathbb{V}_{\mathcal{P}} \ \ \textit{and} \ \ \mathbb{V} \mapsto \mathcal{P}_{\mathbb{V}}$$

are mutually inverse to each other.

Proof. It follows the classical argument of Eilenberg's Theorem.

Acknowledgements. We thank the referees for their fruitful suggestions and comments.

References

1. Bozapalidis, S., Grammatikopoulou, A.: Picture Codes. RAIRO: Theoretical Informatics and Applications 40, 537–550 (2006)
2. Bozapalidis, S., Grammatikopoulou, A.: Recognizable Picture Series. Journal of Automata, Languages and Combinatorics 10(2/3), 159–183 (2005)
3. Bozapalidis, S.: Picture Deformation. Acta Informatica 45, 1–31 (2008)
4. Eilenberg, S.: Automata, Languages and Machines, vol. B. Academic Press, New York (1977)
5. Matz, O.: Regular Expressions and Context-free Grammars for Picture Languages. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 283–294. Springer, Heidelberg (1997)
6. Salehi, S., Steinby, M.: Varieties of many-sorted recognizable sets. PU.M.A 18, 319–343 (2007)

On the Complexity of the Syntax of Tree Languages

Dedicated to Prof. Werner Kuich on the occasion of his retirement.

Symeon Bozapalidis[1] and Antonios Kalampakas[2,3]

[1] Aristotle University of Thessaloniki, Department of Mathematics, 54124, Thessaloniki, Greece
bozapali@math.auth.gr

[2] Democritus University of Thrace, Department of Production Engineering and Management, 67100, Xanti, Greece

[3] Technical Institute of Kavala, Department of Exact Sciences, 65404, Kavala, Greece
akalamp@math.auth.gr

Abstract. The syntactic complexity of a tree language is defined according to the number of the distinct syntactic classes of all trees with a fixed yield length. This leads to a syntactic classification of tree languages and it turns out that the class of recognizable tree languages is properly contained in that of languages with bounded complexity. A refined syntactic complexity notion is also presented, appropriate exclusively for the class of recognizable tree languages. A tree language is recognizable if and only if it has finitely many refined syntactic classes. The constructive complexity of a tree automaton is also investigated and we prove that for any reachable tree automaton it is equal with the refined syntactic complexity of its behavior.

1 Introduction

The notion of graph language recognizability by virtue of magmoids was investigated in [2]. An advantage of this approach is that it is possible to determine the syntactic complexity of graph languages.

We say that two graphs of the same type are equivalent modulo the *syntactic congruence* $\sim_L$, of a graph language L, whenever they have the same set of contexts with respect to L. A graph language L is recognizable if and only if there are finitely many syntactic classes at every type.

The *syntactic complexity* of a recognizable graph language L is then measured by a function mapping any type (m, n) to the number of syntactic classes at this type. This leads to a classification of graph languages according to their syntax. For instance the syntactic complexity of the set $Con(\Sigma)$ of connected graphs is bellian and also graph languages with constant, polynomial and exponential complexity are displayed (cf. [2]). In [6] the language of Eulerian graphs is shown to be syntactically more complicated than that of connected graphs. On the other hand the notion of syntactic complexity in the setup of pictures is discussed in [1].

In the present paper we develop a similar descriptive complexity theory in order to investigate and classify tree languages according to their syntactic structure. Let us denote by T_Γ the set of all trees over the ranked alphabet Γ and by P_Γ the monoid of all trees with just one occurrence of the variable x in their yield. P_Γ acts on T_Γ via substitution at x

$$P_\Gamma \times T_\Gamma \to T_\Gamma, \quad (\tau, t) \mapsto \tau \cdot t = \tau[t/x].$$

Two notions of derivative, with respect to a tree language $L \subseteq T_\Gamma$, arise: for $\tau \in P_\Gamma$ and $t \in T_\Gamma$,

$$\tau^{-1}L = \{t \mid t \in T_\Gamma, \tau \cdot t \in L\}, \quad Lt^{-1} = \{\tau \mid \tau \in P_\Gamma, \tau \cdot t \in L\}.$$

The syntactic congruence associated with L is then

$$t \sim_L t' \text{ if and only if } Lt^{-1} = Lt'^{-1}$$

and the syntactic complexity of L is the function $SC_L : \mathbb{N} \to \mathbb{N}$ which sends every natural number n to the number of distinct $\sim_L$-classes of trees with yield length n. It is well known that every recognizable language L (i.e., behavior of a finite tree automaton) has finitely many right derivatives and so its SC_L is bounded. Thus the growth rate of this syntactic measure can only be used for a classification of non-recognizable tree languages (Section 3).

A refined notion of syntactic complexity is introduced in Section 4 in order to define a syntactic hierarchy within the class of recognizable tree languages. Denote by $P_\Gamma^{(n)}$ the set formed by all trees where $x_1, \ldots, x_n$ occur in the yield of the tree (in this order from left to right) exactly once. For $t_1, \ldots, t_n \in T_\Gamma$, we write $\tau[t_1, \ldots, t_n]$ for the tree obtained by substituting in τ the trees $t_1, \ldots, t_n$ at $x_1, \ldots, x_n$ respectively. There results a function

$$P_\Gamma^{(n)} \times T_\Gamma^n \to T_\Gamma, \quad (\tau, t_1, \ldots, t_n) \mapsto \tau[t_1, \ldots, t_n]$$

according to which two dual notions of derivatives, with respect to a language $L \subseteq T_\Gamma$, arise: for $\tau \in P_\Gamma^{(n)}$ and $t_1, \ldots, t_n \in T_\Gamma$,

$$\tau^{-1}L = \{(t_1, \ldots, t_n) \mid t_1, \ldots, t_n \in T_\Gamma, \tau[t_1, \ldots, t_n] \in L\},$$

$$L(t_1, \ldots, t_n)^{-1} = \{\tau \mid \tau \in P_\Gamma^{(n)}, \tau[t_1, \ldots, t_n] \in L\}.$$

A main result of this paper states that the following conditions are equivalent

i) a language $L \subseteq T_\Gamma$ is recognizable;
ii) for all n, $card\{\tau^{-1}L \mid \tau \in P_\Gamma^{(n)}\} < \infty$;
iii) for all n, $card\{L(t_1, \ldots, t_n)^{-1} \mid t_1, \ldots, t_n \in T_\Gamma\} < \infty$.

The refined syntactic complexity of a recognizable language $L \subseteq T_\Gamma$ is the function $RSC_L : \mathbb{N} \to \mathbb{N}$ sending every natural number n to the number of distinct left derivatives $\tau^{-1}L$, where τ ranges over the set $P_\Gamma^{(n)}$. Two interesting languages with linear and exponential refined syntactic complexity are displayed. The first one is generated by the regular tree grammar

$$G: \quad x \to f(a, x, b), \quad x \to c$$

and $RSC_L(n) = 2(n+1)$, for all n. The second is generated by the regular tree grammar

$$G: \quad x_1 \to f(x_1, x_2), \quad x_1 \to a, \quad x_2 \to g(x_1, x_2), \quad x_2 \to b$$

and $RSC_L(n) = 2^n + 1$, for all n.

In the last section we present a way to measure how complicated the structure of a tree automaton is. Let $\mathcal{M} = (Q, \mu, F)$ be a (deterministic bottom up) tree automaton, over the input alphabet Γ, where Q is the state set, $F \subseteq Q$ the final state set and $\mu = (\mu_f : Q^k \to Q)$, $f \in \Gamma_k$, $k \geq 0$, is the table of moves of $\mathcal{M}$. For $\tau \in P_\Gamma^{(n)}$ and $q_1, \ldots, q_n \in Q$ we denote by $\tau[q_1, \ldots, q_n]$ the state obtained by substituting q_i at x_i inside τ, $1 \leq i \leq n$. The constructive complexity of $\mathcal{M}$ is the function $CC_{\mathcal{M}} : \mathbb{N} \to \mathbb{N}$ defined by the fornmula

$$CC_{\mathcal{M}}(n) = card\{\tau^{-1}F \mid \tau \in P_\Gamma^{(n)}\}, \text{ for all } n,$$

with $\tau^{-1}F = \{(q_1, \ldots, q_n) \mid q_1, \ldots, q_n \in Q, \tau[q_1, \ldots, q_n] \in F\}$. For reachable tree automata $\mathcal{M}, \mathcal{M}'$ we demonstrate that if $\mathcal{M}$ simulates $\mathcal{M}'$, then $CC_{\mathcal{M}} = CC_{\mathcal{M}'}$.

Consequently, the constructive complexity of any reachable automaton $\mathcal{M}$, with behavior L, coincides with the constructive complexity of the minimal tree automaton $\mathcal{M}_L$ associated with L: $CC_{\mathcal{M}} = CC_{\mathcal{M}_L}$.

As $CC_{\mathcal{M}_L} = RSC_L$ we get that the constructive complexity of any reachable automaton is equal with the refined syntactic complexity of its behavior. As a byproduct we get a bound for the function RSC_L, namely

$$RSC_L(n) \leq 2^{(cardQ_L)^n}, \text{ for all } n,$$

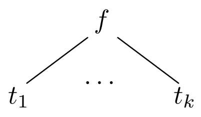
where Q_L is the state set of the minimal automaton $\mathcal{M}_L$.

2 Basic Facts

To construct trees we need a (finite) ranked alphabet $\Gamma = \bigcup\limits_{k \geq 0} \Gamma_k$ and a set $X = \{x_1, x_2, \ldots\}$ of variables. Let $X_n = \{x_1, x_2, \ldots, x_n\}$, $X_0 = \emptyset$. The set of trees over Γ and X is the smallest set of $T_\Gamma(X)$ inductively defined by the items

- $\Gamma_0 \cup X \subseteq T_\Gamma(X)$
- $t_1, \ldots, t_k \in T_\Gamma(X)$ and $f \in \Gamma_k$ implies $f(t_1, \ldots, t_k) \in T_\Gamma(X)$.

Often $f(t_1, \ldots, t_k)$ is depicted as



hence the denomination tree. We write T_Γ instead of $T_\Gamma(\emptyset)$. The height of a tree $t \in T_\Gamma(X)$ is the length of its longest branch. Formally the function *height* : $T_\Gamma(X) \to \mathbb{N}$ is inductively defined by

– $height(\alpha) = 0$, for $a \in \Gamma_0 \cup X$;
– $height(f(t_1, \ldots, t_k)) = 1 + max\{height(t_1), \ldots, height(t_k)\}$, $f \in \Gamma_k$ and $t_1, \ldots, t_k \in T_\Gamma(X)$.

Subsets of $T_\Gamma(X)$ are refereed to as *tree languages.*

The basic operation on trees is substitution. Given $t, t_1, \ldots, t_n \in T_\Gamma(X_n)$, we denote by $t[t_1, \ldots, t_n]$ the result of substituting t_i at every occurrence of x_i, inside t, $1 \le i \le n$. Denote by P_Γ the subset of $T_\Gamma(x)$ consisting of all trees with exactly one occurrence of the variable x. P_Γ becomes a monoid with operation the substitution at x: for $\tau, \pi \in P_\Gamma$, $\tau \cdot \pi = \tau[\pi/x]$. This monoid is *free* over the set of trees of the form

$$f(t_1, \ldots, t_{i-1}, x, t_{i+1}, \ldots, t_k), f \in \Gamma_k, k \ge 1, t_j \in T_\Gamma \ (j \ne i)$$

and acts, again by substitution at x, on the set T_Γ:

$$P_\Gamma \times T_\Gamma \to T_\Gamma, \quad (\tau, t) \mapsto \tau \cdot t = \tau[t/x].$$

The classical machine model consuming trees is the deterministic bottom up Γ-tree automaton. Such a system is a structure $\mathcal{M} = (Q, \mu, F)$ where Q is the finite set of states, $F \subseteq Q$ is the final state set and $\mu = (\mu_f : Q^k \to Q)_{f \in \Gamma_k, k \ge 0}$ is the table of moves of $\mathcal{M}$. The reachability map $\mu_\mathcal{M} : T_\Gamma \to Q$ is inductively defined by

$$\mu_\mathcal{M}(f(t_1, \ldots, t_k)) = \mu_f(\mu_\mathcal{M}(t_1), \ldots, \mu_\mathcal{M}(t_k)), \quad f \in \Gamma_k, t_i \in T_\Gamma, k \ge 0$$

and the behavior of $\mathcal{M}$ is the tree language

$$|\mathcal{M}| = \{t \mid t \in T_\Gamma, \mu_\mathcal{M}(t) \in F\} = \mu_\mathcal{M}^{-1}(F).$$

Tree languages obtained in this way are called *recognizable.* The automaton $\mathcal{M}$ is said to be *reachable* whenever $\mu_\mathcal{M}$ is a surjective function. Given a tree automaton $\mathcal{M} = (Q, \mu, F)$ the monoid P_Γ acts on each state set Q

$$P_\Gamma \times Q \to Q, \quad (\tau, q) \mapsto \tau \cdot q$$

as follows

– $x \cdot q = q$, $q \in Q$
– if τ is of the form $f(t_1, \ldots, t_{i-1}, x, t_{i+1}, \ldots, t_k)$ then

$$\tau \cdot q = \mu_f(\mu_\mathcal{M}(t_1), \ldots, \mu_\mathcal{M}(t_{i-1}), q, \mu_\mathcal{M}(t_{i+1}), \ldots, \mu_\mathcal{M}(t_k))$$

– if $\tau = \tau_1 \cdot \tau_2$, with $\tau_1 \ne x \ne \tau_2$, then

$$\tau \cdot q = \tau_1 \cdot (\tau_2 \cdot q), \quad q \in Q.$$

The reachability map respects the above action.

Proposition 1. *It holds that*

$$\mu_{\mathcal{M}}(\tau \cdot t) = \tau \cdot \mu_{\mathcal{M}}(t) \text{ for every } \tau \in P_\Gamma \text{ and } t \in T_\Gamma.$$

We are going to characterize recognizability in algebraic terms. The *right* and *left derivatives* of a tree language $L \subseteq T_\Gamma$ at $t \in T_\Gamma$ and $\tau \in P_\Gamma$ are given by

$$Lt^{-1} = \{\tau \mid \tau \in P_\Gamma, \tau \cdot t \in L\}, \quad \tau^{-1}L = \{t \mid t \in T_\Gamma, \tau \cdot t \in L\}$$

respectively. The equivalence relation $\sim_L$ on T_Γ

$$t \sim_L t' \text{ if } Lt^{-1} = Lt'^{-1}$$

is well known to be a congruence, i.e.,

$$t_1 \sim_L t'_1, \ldots, t_k \sim_L t'_k \text{ and } f \in \Gamma_k \quad \text{imply} \quad f(t_1, \ldots, t_k) \sim_L f(t'_1, \ldots, t'_k).$$

The next result is folklore.

Proposition 2. *The folowing conditions are equivalent for a language $L \subseteq T_\Gamma$*

i) *L is recognizable*
ii) *$card\{Lt^{-1} \mid t \in T_\Gamma\} < \infty$*
iii) *$card\{\tau^{-1}L \mid \tau \in P_\Gamma\} < \infty$*
iv) *The syntactic congruence $\sim_L$ has finite index (i.e., a finite number of classes).*

A device which is equipowerful to tree automata is the *regular tree grammar*. Such a grammar is a triple $G = (\Gamma, X_n, \mathcal{R})$ where Γ, X_n are the input ranked alphabet and the set of variables respectively, whereas $\mathcal{R}$ is a finite set of rules $x_i \to t$, $t \in T_\Gamma(X_n)$. For $s, s' \in T_\Gamma(X_n)$, we write $s \underset{G}{\Rightarrow} s'$ if there exist $\tau \in P_\Gamma$ and a rule $x_i \to t \in \mathcal{R}$ such that $s = \tau \cdot x_i$ and $s' = \tau \cdot t$. We set

$$L(G, x_i) = \{t \mid t \in T_\Gamma, x_i \overset{*}{\underset{G}{\Rightarrow}} t\}$$

where $\overset{*}{\underset{G}{\Rightarrow}}$ denotes as usual the reflexive and transitive closure of $\underset{G}{\Rightarrow}$.

Proposition 3 (cf. [3,4,5]). *A language $L \subseteq T_\Gamma$ is recognizable if and only if it is generated by a regular tree grammar G: $L = L(G, x_1)$.*

3 Syntactic Complexity of Tree Languages

Syntactic complexity is a tool to study the syntax of a tree language. It counts the number of distinct syntactic classes of trees with a fixed yield length. Formally the *syntactic complexity* of a tree language $L \subseteq T_\Gamma$ is the function

$$SC_L : \mathbb{N} \to \mathbb{N}, \quad SC_L(n) = card\{\bar{t} \mid t \in T_\Gamma, |y(t)| = n\}, \;\; n \in \mathbb{N}$$

where $\bar{t}$ stands for the $\sim_L$-class of t and the function yield, $y : T_\Gamma \to \Gamma_0^*$, is inductively defined by

$$y(c) = c, \;\; (c \in \Gamma_0), \quad y(f(t_1, \ldots, t_k)) = y(t_1) \cdots y(t_k), \;\; (f \in \Gamma_k, \, t_i \in T_\Gamma).$$

Alternatively we have

$$SC_L(n) = card\{Lt^{-1} \mid t \in T_\Gamma, |y(t)| = n\}, \ \ n \in \mathbb{N}.$$

We say that a language $L \subseteq T_\Gamma$ has bounded, polynomial or exponential syntactic complexity if the explicit formula defining the function SC_L is upper bounded by a constant, polynomial or exponential function respectively.

First let us point out that augmenting the basis alphabet Γ the syntactic complexity remains unchanged. Indeed, if $\Gamma \subseteq \Gamma'$ and $L \subseteq T_\Gamma \subseteq T_{\Gamma'}$ then the syntactic complexity of L computed with respect to Γ and Γ' differ at most by 1 since, for all $t, t' \in T_{\Gamma'} \setminus T_\Gamma$, we have that $t \sim_L t'$. Thus SC_L does not depend on Γ.

According to Proposition 2 every recognizable tree language has bounded syntactic complexity $SC_L(n) \leq k$ for a fixed k and all $n \in \mathbb{N}$. However this fact does not characterize tree language recognizability as is confirmed by the next example.

Example 1. Take the alphabet $\Gamma = \{f, \alpha\}$ with $rank(f) = 2$, $rank(\alpha) = 0$ and consider the tree languages L_{bal} of all balanced trees and L_{fib} of all Fibonacci trees

$$L_{bal} = \{t_k \mid t_0 = \alpha, \, t_{k+1} = f(t_k, t_k), \, k \geq 1\},$$
$$L_{fib} = \{s_k \mid s_0 = s_1 = a, \, s_{k+2} = f(s_{k+1}, s_k), \, k \geq 0\},$$

respectively. Observe that $|y(t_k)| = 2^k$ while $|y(s_k)| = f_k$, the k-th Fibonacci number. The trees

$$\tau_k = f(t_k, x), \quad \pi_k = f(s_{k+1}, x),$$

have the properties

$$\tau_k \cdot t_k \in L_{bal}, \text{ but } \tau_k \cdot t \notin L_{bal} \text{ for } t \neq t_k,$$

and

$$\pi_k \cdot s_k \in L_{fib}, \text{ but } \pi_k \cdot s \notin L_{fib} \text{ for } s \neq s_k,$$

respectively. Therefore the derivatives $L_{bal}t_k^{-1}$ are pairwise distinct and so are the derivatives $L_{fib}s_k^{-1}$ respectively. It turns out that

$$card\{L_{bal}t^{-1} \mid t \in T_\Gamma\} = \infty = card\{L_{fib}s^{-1} \mid s \in T_\Gamma\}$$

and so both the languages L_{bal} and L_{fib} are not recognizable. Moreover, it holds

$$\begin{aligned} SC_{L_{bal}}(n) &= 2, \text{ if } n = 2^k \\ &= 1, \text{ otherwise} \end{aligned}$$

and similarly,

$$\begin{aligned} SC_{L_{fib}}(n) &= 2, \text{ if } n = f_k \\ &= 1, \text{ otherwise.} \end{aligned}$$

Thus, although L_{bal}, L_{fib} are not recognizable, they have bounded syntactic complexity.

Consequently,

Proposition 4. *The class BSC of tree languages with bounded syntactic complexity properly contains the class REC of recognizable tree languages.*

Our notion of complexity permits to classify the non recognizable tree languages in a non trivial way as it is presented below.

Proposition 5. *Given the ranked alphabet $\Gamma = \{f_1, \ldots, f_k, \alpha\}$, $rank(f_i) = 2$, $1 \leq i \leq k$, $rank(\alpha) = 0$, the Dyck tree language of order k*

$$D_k = \{t \mid t \in T_\Gamma,\ |t|_{f_1} = \cdots = |t|_{f_k}\}$$

has polynomial syntactic complexity of degree $k-1$, namely

$$SC_{D_k}(n) = \frac{1}{(k-1)!}\, n(n+1)\cdots(n+k-2).$$

Proof. For $t, t' \in T_\Gamma$ we have

$$D_k t^{-1} = D_k t'^{-1} \text{ if and only if } |t|_{f_i} = |t'|_{f_i} \text{ for } i = 1, \ldots, k.$$

On the other hand the number of binary symbols occurring in a tree $t \in T_\Gamma$ with yield length n is just $n-1$. Therefore the different ways to share the symbols $f_1, \ldots, f_k$ in the nodes of t is equal with the number of k-tuples of natural numbers $(x_1, \ldots, x_k)$ verifying the equation

$$x_1 + \cdots + x_k = n - 1$$

which, as it is well known from Combinatorics, is equal with

$$\binom{n-1+k-1}{k-1} = \binom{n+k-2}{k-1} = \frac{1}{(k-1)!}\, n(n+1)\cdots(n+k-2).$$

Hence the proposed formula.

A tree language $L \subseteq T_\Gamma$ such that for every n

$$card\{Lt^{-1} \mid t \in T_\Gamma, |y(t)| = n\} = card\{t \mid t \in T_\Gamma, |y(t)| = n\}$$

will be called *syntactically hard.* Of course such a language L has the highest possible syntactic complexity, i.e.,

$$SC_L(n) = card\{t \mid t \in T_\Gamma, |y(t)| = n\}.$$

In the case that $\Gamma = \{f, a\}$, with $rank(f) = 2$, $rank(\alpha) = 0$ the above number is well known from Combinatorics and is the $n-1$-th Catalan number C_{n-1}, where

$$C_n = \frac{1}{n+1}\binom{2n}{n} \simeq \frac{4^n}{n^{3/2}\sqrt{\pi}}.$$

Proposition 6. *The diagonal language*

$$L_d = \{f(t,t) \mid t \in T_\Gamma\}, \quad \Gamma = \{f, \alpha\},$$

is syntactically hard

$$SC_{L_d}(n+1) = \frac{1}{n+1}\binom{2n}{n}.$$

Proof. Actually, we shall show that the right derivatives $L_d t^{-1}$, $t \in T_\Gamma$, are pairwise distinct. First observe that

$$L_d t^{-1} = \{f(s,\tau) \mid s \in T_\Gamma,\ \tau \in P_\Gamma,\ s = \tau\cdot t\} \cup \{f(\tau,s) \mid s \in T_\Gamma,\ \tau \in P_\Gamma,\ s = \tau\cdot t\}.$$

Now, if $L_d t^{-1} \cap L_d t'^{-1} \neq \emptyset$, then

$$f(s,\tau) = f(s',\tau'), \quad s = \tau \cdot t,\ s' = \tau' \cdot t',$$

or

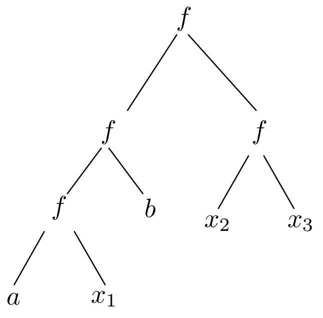
$$f(\tau,s) = f(\tau',s'), \quad s = \tau \cdot t,\ s' = \tau' \cdot t'.$$

Hence $s = s'$, $\tau = \tau'$ and $t = t'$.

4 Refined Syntactic Complexity

As we have seen the growth rate of the function SC_L introduced in the previous section gives no information that allows us to compare recognizable tree languages with respect to their complexity. Our intention in the present section is to provide an efficient complexity measure for recognizable tree languages. Let us denote by $P_\Gamma^{(n)}$ the subset of $T_\Gamma(X_n)$ formed by all trees where $x_1, \dots, x_n$ occur in the yield of the tree (in this order from left to right) exactly once. For instance the tree

$\tau =$



$\in P_\Gamma^{(3)}$.

For every $n \geq 1$ there is a junction function

$$P_\Gamma^{(n)} \times T_\Gamma^n \to T_\Gamma, \quad (\tau, t_1, \dots, t_n) \mapsto \tau[t_1, \dots, t_n].$$

With respect to $L \subseteq T_\Gamma$, two dual notions of derivatives can be defined:

$$\tau^{-1}L = \{(t_1, \dots, t_n) \mid \tau[t_1, \dots, t_n] \in L\},$$

$$L(t_1, \dots, t_n)^{-1} = \{\tau \mid \tau \in P_\Gamma^{(n)},\ \tau[t_1, \dots, t_n] \in L\},$$

for all $\tau \in P_\Gamma^{(n)}$ and $t_1, \dots, t_n \in T_\Gamma$.

Theorem 1. *For $L \subseteq T_\Gamma$, the following conditions are equivalent*

i) *L is recognizable*
ii) *for every $n \geq 1$, $card\{\tau^{-1}L \mid \tau \in P_\Gamma^{(n)}\} < \infty$*
iii) *for every $n \geq 1$, $card\{L(t_1, \dots, t_n)^{-1} \mid t_1, \dots, t_n \in T_\Gamma\} < \infty$.*

Proof. iii) $\Rightarrow$ ii). Assume that $L(t_{11}, \dots, t_{1n})^{-1}, \dots, L(t_{k1}, \dots, t_{kn})^{-1}$ are the distinct right derivatives of L. Then the function

$$\phi : \{\tau^{-1}L \mid \tau \in P_\Gamma^{(n)}\} \to \{0,1\}^k, \quad \phi(\tau^{-1}L) = (\varepsilon_1, \dots, \varepsilon_k)$$

with $\varepsilon_i = 1$ iff $\tau[t_{i1}, \dots, t_{in}] \in L$ is well defined and moreover it is injective since

$$\phi(\tau^{-1}L) = \phi(\tau'^{-1}L) \text{ implies } \tau^{-1}L = \tau'^{-1}L.$$

The hypothesis $\phi(\tau^{-1}L) = \phi(\tau'^{-1}L)$ is equivalent to

$$\tau[t_{i1}, \dots, t_{in}] \in L \text{ iff } \tau'[t_{i1}, \dots, t_{in}] \in L \tag{1}$$

for all $i = 1, 2, \dots, k$. We have

$$\begin{aligned}
(s_1, \dots, s_n) \in \tau^{-1}L &\Leftrightarrow \tau[s_1, \dots, s_n] \in L \\
&\Leftrightarrow \tau \in L(s_1, \dots, s_n)^{-1} = L(t_{i1}, \dots, t_{in})^{-1}, \quad \text{for some } i \\
&\Leftrightarrow \tau[t_{i1}, \dots, t_{in}] \in L \quad \text{(by 1 above)} \\
&\Leftrightarrow \tau'[t_{i1}, \dots, t_{in}] \in L \\
&\Leftrightarrow \tau' \in L(t_{i1}, \dots, t_{in})^{-1} \\
&\Leftrightarrow (s_1, \dots, s_n) \in \tau'^{-1}L
\end{aligned}$$

that is $\tau^{-1}L = \tau'^{-1}L$ as wanted. From the injectivity of ϕ we get

$$card\{\tau^{-1}L \mid \tau \in P_\Gamma^{(n)}\} < \infty.$$

The implication *ii) $\Rightarrow$ iii)* can be proved in a similar way.

The fact that *ii) $\Rightarrow$ i)* follows from Proposition 2 since $P_\Gamma = P_\Gamma^{(1)}$.

i) $\Rightarrow$ ii). Consider a tree automaton $\mathcal{M} = (Q, \mu, F)$ with behavior L and let $\mu_\mathcal{M} : T_\Gamma \to Q$ be its reachability map. We shall demonstrate that for every $t_1, \dots, t_n \in T_\Gamma$ there exist $\bar{t}_1, \dots, \bar{t}_n \in T_\Gamma$ with *height* less or equal to *cardQ* such that

$$\mu_\mathcal{M}(\tau[t_1, \dots, t_n]) = \mu_\mathcal{M}(\tau[\bar{t}_1, \dots, \bar{t}_n]), \quad \text{for all } \tau \in P_\Gamma^{(n)}.$$

Indeed let us choose $\bar{t}_i$ with the property

$$\mu_\mathcal{M}(\bar{t}_i) = \mu_\mathcal{M}(t_i), \quad height(\bar{t}_i) \leq cardQ$$

for all $i = 1, \dots, n$. Then we get

$$\begin{aligned}
\mu_\mathcal{M}(\tau[t_1, \dots, t_n]) &= \mu_\mathcal{M}(\tau[x, t_2, \dots, t_n] \cdot t_1) && \text{(by Prop. 1)} \\
&= \tau[x, t_2, \dots, t_n] \cdot \mu_\mathcal{M}(t_1) \\
&= \tau[x, t_2, \dots, t_n] \cdot \mu_\mathcal{M}(\bar{t}_1) && \text{(by Prop. 1)} \\
&= \mu_\mathcal{M}(\tau[x, t_2, \dots, t_n] \cdot \bar{t}_1) \\
&= \mu_\mathcal{M}(\tau[\bar{t}_1, \dots, t_n]) = \cdots = \mu_\mathcal{M}(\tau[\bar{t}_1, \dots, \bar{t}_n]).
\end{aligned}$$

Now it holds that

$$L(t_1,\dots,t_n)^{-1} = L(\bar{t}_1,\dots,\bar{t}_n)^{-1}.$$

In fact

$$\begin{aligned}
\tau \in L(t_1,\dots,t_n)^{-1} &\Leftrightarrow \tau[t_1,\dots,t_n] \in L = \mu_{\mathcal{M}}^{-1}(F) \\
&\Leftrightarrow \mu_{\mathcal{M}}(\tau[t_1,\dots,t_n]) \in F \\
&\Leftrightarrow \mu_{\mathcal{M}}(\tau[\bar{t}_1,\dots,\bar{t}_n]) \in F \\
&\Leftrightarrow \tau[\bar{t}_1,\dots,\bar{t}_n] \in \mu_{\mathcal{M}}^{-1}(F) = L \\
&\Leftrightarrow \tau \in L(\bar{t}_1,\dots,\bar{t}_n)^{-1}
\end{aligned}$$

as wanted. It follows that

$$card\{L(t_1,\dots,t_n)^{-1} \mid t_1,\dots t_n \in T_\Gamma\} \le (cardQ)^n < \infty$$

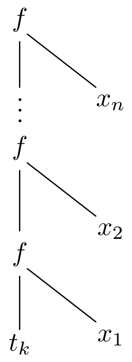
and the proof is completed.

The *refined syntactic complexity* of a recognizable tree language $L \subseteq T_\Gamma$ is the function $RSC_L : \mathbb{N} \to \mathbb{N}$ sending every natural number n to the number of the distinct left derivatives $\tau^{-1}L$ when τ ranges over $P_\Gamma^{(n)}$, i.e.,

$$RSC_L(n) = card\{\tau^{-1}L \mid \tau \in P_\Gamma^{(n)}\}.$$

Example 2. Return to the non-recognizable language L_{bal} and let us choose the trees

$\tau_{k,n} =$



$\in P_\Gamma^{(n)}$.

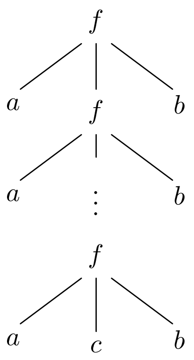
For $s_0,\dots,s_{n-1} \in T_\Gamma$, we have $\tau_{k,n}[s_0,\dots,s_{n-1}] \in L_{bal}$ iff $s_i = t_{k+i}$ for $0 \le i < n$. In other words, there are infinitely many distinct left derivatives $\tau^{-1}L_{bal}$, $\tau \in P_\Gamma^{(n)}$, i.e., $RSC_{L_{bal}}(n) = \infty$ for all n.

Similar observations can be made for L_{fib}.

In the sequel we display two recognizable languages having linear and exponential syntactic complexity respectively.

Example 3. Consider the recognizable tree language L consisting of all trees of the form

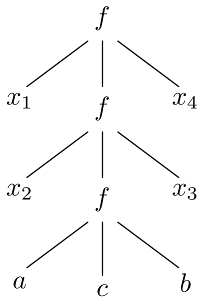
$t_m =$



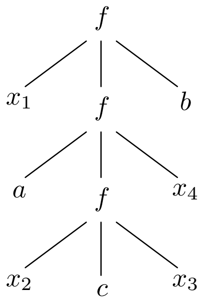
(m occurrences of f, $m \geq 1$).

For $0 \leq k \leq n \leq 2m$ let us denote by d_k a strictly increasing function from $\{1, 2, \ldots, n\}$ to $\{1, 2, \ldots, 2m\}$ such that $d_k(i) \leq m$ for all $i = 1, 2, \ldots, k$ and $d_k(i) \geq m + 1$ for $i = k + 1, k + 2, \ldots, n$. We introduce the tree $\tau(d_k)$ obtained from t_m above by distributing via d_k the variables $x_1, \ldots, x_k$ on the left nodes labelled by a and the remaining variables $x_{k+1}, \ldots x_n$ on the right nodes labelled by b. For instance

$\tau(d_2) =$



and $\tau(d_2') =$



with $d_2, d_2' : \{1, 2, 3, 4\} \to \{1, 2, 3, 4, 5, 6\}$ given by $d_2(1) = 1$, $d_2(2) = 2$, $d_2(3) = 5$, $d_2(4) = 6$ and $d_2'(1) = 1$, $d_2'(2) = 3$, $d_2'(3) = 4$, $d_2'(4) = 5$ respectively. It is not hard to see that, if d_k, d_k' are two distributions as defined previously, then we have

$$\tau(d_k)^{-1}L = \tau(d_k')^{-1}L$$

therefore there are exactly $n + 1$ distinct left derivatives of the above form, namely,

$$\tau(d_0)^{-1}L, \tau(d_1)^{-1}L, \ldots, \tau(d_n)^{-1}L.$$

Next for $1 \leq k \leq n \leq 2m+1$ let us denote by δ_k a strictly increasing function from $\{1, 2, \ldots, n\}$ to $\{1, 2, \ldots, 2m + 1\}$ with the property $\delta_k(k) = m + 1$ and $\delta_k(i) \leq m$ $(i \leq k - 1)$, $\delta_k(i) \geq m + 2$ $(i \geq k + 1)$. Also we denote by $\tau(\delta_k)$ the tree obtained from t_m above by distributing via δ_k the variables $x_1, \ldots x_{k-1}$ on the left nodes labelled by a, the variables $x_{k+1}, \ldots x_n$ on the right nodes labelled by b whereas the variable x_k replaces the node labelled by c. As above we can verify that there are exactly n distinct left derivatives

$$\tau(\delta_1)^{-1}L, \tau(\delta_2)^{-1}L, \ldots, \tau(\delta_n)^{-1}L.$$

Of course if $\tau \in P_\Gamma^{(n)}$ is neither of the form $\tau(d)$ nor $\tau(\delta)$ then $\tau^{-1}L = \emptyset$. We conclude that there are in total $n+1+n+1 = 2(n+1)$ distinct left derivatives of L at level n, i.e., $RSC_L(n) = 2(n+1)$ and the language L has linear syntactic complexity.

Example 4. Consider the regular tree grammar

$$G: \quad y_1 \to f(y_1, y_2), \quad y_2 \to g(y_1, y_2), \quad y_1 \to a, \quad y_2 \to b,$$

and the tree language $L(G, y_1)$ generated by G starting from the variable y_1. A tree t belongs to $L(G, y_1)$ if and only if the left (resp. right) child of any node of t is labelled either by f or a (resp. g or b). From any $t \in L(G, y_1)$ and any strictly increasing function $d : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, |y(t)|\}$ we derive the tree $\tau(t, d) \in P_\Gamma^{(n)}$ by replacing the letter located at the $d(i)$-th position of $y(t)$ (from left to right) with the variable x_i $(1 \le i \le n)$. It is not hard to see that two such trees $\tau(t, d)$ and $\tau(t', d')$ define the same left derivative,

$$\tau(t, d)^{-1}L(G, y_1) = \tau(t', d')^{-1}L(G, y_1)$$

if and only if for every $i \in \{1, 2, \ldots, n\}$ the $d(i)$-th and $d'(i)$-th letters in $y(t)$, $y(t')$, respectively, are equal. It turns out that the distinct left derivatives of $L(G, y_1)$ correspond to the possible ways of substituting x_1 by a or b, x_2 by a or b, etc. Taking into account the empty left derivative we finally obtain

$$RSC_{L(G,y_1)}(n) = 2^n + 1$$

that is $L(G, y_1)$ has exponential syntactic complexity.

5 Constructive Complexity of a Tree Automaton

Here we display a way to measure how complicated the structure of a tree automaton is. First we need some additional notation. Given a tree automaton $\mathcal{M} = (Q, \mu, F)$, for every $t \in T_\Gamma(X_n)$ and every $q_1, \ldots, q_n \in Q$, the element $t[q_1, \ldots, q_n] \in Q$ is inductively defined as follows

- for $t = x_i$, $x_i[q_1, \ldots, q_n] = q_i$, $1 \le i \le n$;
- for $t = c \in \Gamma_0$, $c[q_1, \ldots, q_n] = \mu_c$;
- for $t = f(t_1, \ldots, t_k)$, $f \in \Gamma_k$, $t_i \in T_\Gamma(X_n)$

$$f(t_1, \ldots, t_k)[q_1, \ldots, q_n] = \mu_f(t_1[q_1, \ldots, q_n], \ldots, t_k[q_1, \ldots, q_n]).$$

The *constructive complexity* of the automaton $\mathcal{M} = (Q, \mu, F)$ is the function $CC_\mathcal{M} : \mathbb{N} \to \mathbb{N}$ defined by the formula

$$CC_\mathcal{M}(n) = card\{\tau^{-1}F \mid \tau \in P_\Gamma^{(n)}\}$$

where

$$\tau^{-1}F = \{(q_1, \ldots, q_n) \mid \tau[q_1, \ldots, q_n] \in F\}.$$

Since for all n we have $\tau^{-1}F \subseteq Q^n$, we get that

$$CC_{\mathcal{M}}(n) \leq 2^{(cardQ)^n}$$

and so $CC_{\mathcal{M}}$ is everywhere defined.

Example 5. Let Γ be a finite ranked alphabet and consider the automaton $\mathcal{M} = (\mathbb{Z}_m, \mu, F = \{0\})$ where $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$ is the additive group of integers $mod\, m$. The moves $\mu_f : \mathbb{Z}_m^k \to \mathbb{Z}_m$ are given by

$$\mu_c = 1, \ \ (c \in \Gamma_0), \qquad \mu_f(\alpha_1, \ldots, \alpha_k) = 1 + \alpha_1 + \cdots + \alpha_k, \ \ (f \in \Gamma_k, k \geq 1)$$

where at the right hand side the designated addition is the $mod\, m$ addition. The reachability map $\mu_{\mathcal{M}} : T_\Gamma \to \mathbb{Z}_m$ sends every tree t to its $mod\, m$ size, i.e.,

$$\mu_{\mathcal{M}}(t) = |t| (mod\, m)$$

and the behavior of $\mathcal{M}$ consists of all trees whose size is divisible by m. For $\tau, \tau' \in P_\Gamma^{(n)}$, we have

$$\tau^{-1}F = \tau'^{-1}F \text{ if and only if } |\tau| \equiv |\tau'| (mod\, m).$$

Consequently, there are exactly m distinct classes $\tau^{-1}F$, that is $CC_{\mathcal{M}}(n) = m$ for all n and thus $\mathcal{M}$ has constant constructive complexity.

A naturally arising question concerns the comparison of the complexities $CC_{\mathcal{M}}$ and $RSC_{|\mathcal{M}|}$. Recall that a *simulation* of $\mathcal{M} = (Q, \mu, F)$ to $\mathcal{M}' = (Q', \mu', F')$ is a surjective function $h : Q \to Q'$ respecting the moves

$$h(\mu_f(q_1, \ldots, q_n)) = \mu'_f(h(q_1), \ldots, h(q_k)), \qquad f \in \Gamma_k, \ q_i \in Q,$$

and moreover $h^{-1}(F') = F$. An induction argument on the complexity of the tree $\tau \in P_\Gamma^{(n)}$ shows that

$$h(\tau[q_1, \ldots, q_n]) = \tau[h(q_1), \ldots, h(q_n)], \qquad q_1, \ldots, q_n \in Q. \tag{2}$$

Proposition 7. *Let $\mathcal{M}, \mathcal{M}'$ be reachable tree automata. If there is a simulation $h : \mathcal{M} \to \mathcal{M}'$ then both $\mathcal{M}$ and $\mathcal{M}'$ have the same constructive complexity*

$$CC_{\mathcal{M}} = CC_{\mathcal{M}'}.$$

Proof. We have to show that

$$CC_{\mathcal{M}}(n) = CC_{\mathcal{M}'}(n), \quad \text{for all } n$$

or that

$$card\{\tau^{-1}F \mid \tau \in P_\Gamma^{(n)}\} = card\{\tau^{-1}F' \mid \tau \in P_\Gamma^{(n)}\}, \quad \text{for all } n.$$

The last fact will follow if we show that the assignment $\tau^{-1}F \mapsto \tau^{-1}F'$ is a well defined injection, which is expressed by the following logical equivalence

$$\tau^{-1}F = \tau'^{-1}F \Leftrightarrow \tau^{-1}F' = \tau'^{-1}F'.$$

Assume that $\tau^{-1}F = \tau'^{-1}F$, then

$$\begin{aligned}(q_1', \dots, q_n') \in \tau^{-1}F' &\Leftrightarrow (h(q_1), \dots, h(q_n)) \in \tau^{-1}F', \qquad q_i' = h(q_i),\ 1 \leq i \leq n \\ &\Leftrightarrow \tau[h(q_1), \dots, h(q_n)] \in F' \qquad \text{(by Eq. (2))} \\ &\Leftrightarrow h(\tau[q_1, \dots, q_n]) \in F' \qquad (\text{by } F = h^{-1}(F')) \\ &\Leftrightarrow \tau[q_1, \dots, q_n] \in F \\ &\Leftrightarrow (q_1, \dots, q_n) \in \tau^{-1}F = \tau'^{-1}F \\ &\Leftrightarrow \tau'[q_1, \dots, q_n] \in F \\ &\Leftrightarrow h(\tau'[q_1, \dots, q_n]) \in F' \\ &\Leftrightarrow (q_1', \dots, q_n') \in \tau'^{-1}F'\end{aligned}$$

and thus $\tau^{-1}F' = \tau'^{-1}F'$. The implication

$$\tau^{-1}F' = \tau'^{-1}F' \Rightarrow \tau^{-1}F = \tau'^{-1}F$$

is proved analogously.

The minimal automaton associated with a tree language $L \subseteq T_\Gamma$ is

$$\mathcal{M}_L = (Q_L, \mu_L, F_L)$$

where

- $Q_L = \{Lt^{-1} \mid t \in T_\Gamma\}, \quad F_L = \{Lt^{-1} \mid t \in L\}$;
- $(\mu_L)_f : Q_L^k \to Q_L, \quad (\mu_L)_f(Lt_1^{-1}, \dots, Lt_k^{-1}) = Lf(t_1, \dots, t_k)^{-1}, \quad f \in \Gamma_k$.

Clearly $\mathcal{M}_L$ is a reachable automaton with behavior L and for every reachable automaton $\mathcal{M} = (Q, \mu, F)$ with behavior L, there is a (unique) simulation $h : \mathcal{M} \to \mathcal{M}_L$ defined by

$$h(q) = Lt^{-1}, \quad \mu_{\mathcal{M}}(t) = q, \quad q \in Q$$

therefore, by virtue of Proposition 7, we get

$$CC_{\mathcal{M}} = CC_{\mathcal{M}_L}.$$

On the other hand

Proposition 8. *If $\mathcal{M}$ is a reachable automaton with behavior L, then*

$$CC_{\mathcal{M}} = RSC_L.$$

Proof. Analogous to that of Proposition 7.

Taking into account the previous discussion we conclude that

Proposition 9. *For every recognizable tree language $L \subseteq T_\Gamma$ it holds*

$$RSC_L(n) \leq 2^{(CardQ_L)^n}, \quad \textit{for all } n.$$

Acknowledgement

We thank the referees for their fruitful suggestions and remarks.

References

1. Bozapalidis, S.: Picture languages: Recognizability, Syntax and Context Freeness (submitted)
2. Bozapalidis, S., Kalampakas, A.: Recognizability of graph and pattern languages. Acta Informatica 42, 553–581 (2006)
3. Engelfriet, J.: Tree Automata and Tree Grammars, DAIMNI FN-10, University of Aarhus (1975)
4. Gécseg, F., Steinby, M.: Tree Automata. Akademiai Kiado, Budapest (1984)
5. Gécseg, F., Steinby, M.: Tree Languages. Handbook of Formal Languages 3, 1–68 (1997)
6. Kalampakas, A.: The syntactic complexity of eulerian graphs. In: Bozapalidis, S., Rahonis, G. (eds.) CAI 2007. LNCS, vol. 4728, pp. 208–217. Springer, Heidelberg (2007)

On the Reversibility of Parallel Insertion, and Its Relation to Comma Codes[⋆]

Bo Cui, Lila Kari, and Shinnosuke Seki

Department of Computer Science, University of Western Ontario,
London, Ontario, Canada, N6A 5B7
{bcui2,lila,sseki}@csd.uwo.ca

Abstract. This paper studies conditions under which the operation of parallel insertion can be reversed by parallel deletion, i.e., when does the equality $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$ hold for languages L_1 and L_2. We obtain a complete characterization of the solutions in the special case when both languages involved are singleton words. We also define *comma codes*, a family of codes with the property that, if L_2 is a comma code, then the above equation holds for any language $L_1 \subseteq \Sigma^*$. Lastly, we generalize the notion of comma codes to that of comma intercodes of index m. Besides several properties, we prove that the families of comma intercodes of index m form an infinite proper inclusion hierarchy, the first element which is a subset of the family of infix codes, and the last element of which is a subset of the family of bifix codes.

1 Introduction

In combinatorics on words and formal language theory, operations play an essential role in understanding the mechanisms of generating words and languages. Several generalizations of catenation and quotient, such as shuffle, shuffle on trajectories, [14], sequential and parallel insertion and deletion, [5], distributed catenation, [10], mix operation, [11], deletion on trajectories, [2], and hairpin completion and reduction, [13], have been studied in the literature. Follow-up studies investigated properties of languages produced by sequential and parallel insertion and deletion in [3,6,7,8,9]. One particular topic of interest was the reversibility of some of these operations, originally motivated by cryptography applications: If one uses the insertion of a key as one component of a cryptosystem to encrypt a plain-text message, and one step of decryption is accomplished by the deletion of the key, what are the language properties that would ensure that the plain-text can be uniquely deciphered? Motivated by this potential application, the determinism and inversibility of insertion and deletion operations on words were studied in, e.g., [6].

The question can be asked in a more general framework wherein the operations involved are the parallel insertion and deletion. This paper represents a first step

⋆ This research was supported by Discovery Grant of the Natural Science and Engineering Research Council of Canada, and Canada Research Chair Award to L.K.

towards an answer. More precisely, similar to sequential insertion and deletion, if we parallel-delete a word v from the language obtained by parallel-inserting v into u, we will not always obtain u. Thus, the question we ask is "Under what conditions, after parallel-inserting v into u, followed by the parallel deletion of v from the result, do we obtain exactly u?".

In Sect. 3, after the investigation of various properties of parallel insertion and deletion, we give a complete answer to this question for the singleton case, and furthermore we generalize the question to languages. We show that, if L_2 is a comma code (formally introduced in Sect. 4), any language L_1 can be recovered after first parallel-inserting L_2 into L_1 and then parallel-deleting L_2 from the result.

The notion of codes was defined for applications in communication systems. That is, if a message is encoded by using words from a code, then any arbitrary catenation of words should be uniquely decodable into code-words. Various codes with specific algebraic properties, such as prefix codes, infix codes, and comma-free codes [1,16,17], have been defined and used for various purposes. In Sect. 4, we define a family of codes, named *comma codes*, and show that this family is a proper subfamily of that of infix codes. Also, we give a characterization of comma codes, obtain some closure and algebraic properties, as well as compare the comma code family with other families, such as that of comma-free codes and that of solid codes.

Based on the similarity between the definition of comma codes and that of comma-free codes, in Sect. 5, we generalize comma codes and introduce the notion of *comma intercodes*. Similar to the notion of intercodes [16,17,18], the families of comma intercodes of index m form a proper inclusion hierarchy within the family of bifix codes. However, we show that any two families of intercodes and comma intercodes are incomparable.

2 Preliminaries

An alphabet Σ is a nonempty finite set of letters. A word over Σ is a sequence of letters in Σ. The length of a word w, denoted by $|w|$, is the number of letters in this word. The empty word, denoted by λ, is the word of length 0, while a unary word is a word of the form a^j, $j \geq 1$, $a \in \Sigma$. The set of all words over Σ is denoted by Σ^*, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$ is the set of all nonempty words. A language is a subset of Σ^*. A language with exactly one word is called a *singleton*. In this paper, for a word $w \in \Sigma^*$, we often denote a singleton $\{w\}$ as w. A catenation of two languages $L_1, L_2 \subseteq \Sigma^*$, denoted by L_1L_2, is defined as $L_1L_2 = \{uv \mid u \in L_1, v \in L_2\}$. As mentioned, if an operand is a singleton, say $L_1 = \{u\}$ or $L_2 = \{v\}$, then we write uL_2 or L_1v instead of $\{u\}L_2$ or $L_1\{v\}$.

A word $x \in \Sigma^*$ is called an infix (prefix, suffix) of a word $u \in \Sigma^+$ if $u = zxy$ ($u = xy$, $u = zx$) for some words $y, z \in \Sigma^*$. In this definition, if z and y are nonempty, then such an x is called a *proper* infix, prefix, or suffix of u. For a word $u \in \Sigma^*$, the set of its infixes (prefixes, suffixes) is denoted by $\mathrm{F}(u)$ (resp. $\mathrm{Pref}(u)$, $\mathrm{Suff}(u)$). For a word $u \in \Sigma^*$, we denote the prefix (suffix) of length

$n \geq 0$ by $\mathrm{pref}_n(u)$ (resp. $\mathrm{suff}_n(u)$). These notations can be naturally extended to languages, e.g., $\mathrm{Pref}(L)$ is the set of prefixes of the words in L.

A nonempty word $u \in \Sigma^+$ is said to be *primitive* if $u = v^n$ implies $n = 1$ and $u = v$ for any $v \in \Sigma^+$. Any non-primitive word can be written as a power of a unique primitive word [16], which is called the *primitive root* of the word.

It is well known that [16], if nonempty words $x, y, z \in \Sigma^+$ satisfy $xy = yz$, then there exist $\alpha, \beta \in \Sigma^*$ such that $\alpha\beta$ is primitive, $x = (\alpha\beta)^i$, $y = (\alpha\beta)^j\alpha$, and $z = (\beta\alpha)^i$ for some $i \geq 1$ and $j \geq 0$.

A nonempty word $u \in \Sigma^+$ is called *bordered* if there exists a nonempty word which is both proper prefix and proper suffix of u. A *bordered primitive word* is a primitive word which is bordered, and it can be written as xyx for some $x, y \in \Sigma^+$ [16].

Parallel insertion and deletion on words and languages are variants of well-known (sequential) insertion and deletion, introduced in [5]. For two words $u, v \in \Sigma^*$, the *parallel insertion of* v *into* u results in a word $va_1va_2 \cdots a_nv$, where $u = a_1a_2 \cdots a_n$ for letters $a_1, \ldots, a_n \in \Sigma$. We denote this resulting word by $u \Leftarrow v$. This operation can be generalized to languages as follows: for two languages $L_1, L_2 \subseteq \Sigma^*$, the *parallel insertion of* L_2 *into* L_1 generates a language

$$L_1 \Leftarrow L_2 = \bigcup_{n \geq 1,\, a_1, \ldots, a_n \in \Sigma \text{ s.t. } a_1a_2\cdots a_n \in L_1} L_2a_1L_2a_2 \cdots L_2a_nL_2.$$

Example 1. For $L_1 = \{cd\}$ and $L_2 = \{a, b\}$,

$$\begin{aligned} L_1 \Leftarrow L_2 &= L_2cL_2dL_2 \\ &= \{acada, acadb, acbda, acbdb, bcada, bcadb, bcbda, bcbdb\}. \end{aligned}$$

In contrast, the parallel deletion of a language L_2 from another language L_1 results in a set of words which can be obtained by deleting elements of L_2 from an element of L_1 in a "maximal parallel manner". We denote the resulting set by $L_1 \Rightarrow L_2$. For $u \in L_1$, let

$$\begin{aligned} u \Rightarrow L_2 = \big\{ & u_1u_2 \cdots u_ku_{k+1} \mid u_1, \ldots, u_{k+1} \in \Sigma^*,\, k \geq 1,\, u \in u_1L_2u_2L_2 \cdots L_2u_{k+1} \\ & \text{and } \mathrm{F}(u_i) \cap (L_2 \setminus \{\lambda\}) = \emptyset \text{ for all } 1 \leq i \leq k+1 \big\}. \end{aligned}$$

By this definition, it is clear that if u does not contain any word in L_2 as its infix, then $u \Rightarrow L_2 = \emptyset$. Then we define $L_1 \Rightarrow L_2 = \bigcup_{u \in L_1}(u \Rightarrow L_2)$.

Example 2. Let $L_1 = \{abababa, aababa, abaabaaba\}$ and $L_2 = \{aba\}$. Then

$$\begin{aligned} L_1 \Rightarrow L_2 &= (\{abababa\} \Rightarrow L_2) \cup (\{aababa\} \Rightarrow L_2) \cup (\{abaabaaba\} \Rightarrow L_2) \\ &= \{b, abba\} \cup \{aba, aab\} \cup \{\lambda\} = \{b, abba, aba, aab, \lambda\}. \end{aligned}$$

3 When Does $(L_1 \Leftarrow L_2) \Rightarrow L_2$ Equal L_1?

By definitions, parallel insertion and deletion are not inverse operations in the sense that L_1 may not equal to $(L_1 \Leftarrow L_2) \Rightarrow L_2$. Thus, a question of interest is to find under what conditions does the equality $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$ hold. We start by providing some properties of parallel insertions and deletions relevant to this question.

The simplest case is when the operation is the parallel insertion and both operands are singleton words. The next theorem will show that, unless w and u are unary words over the same letter, $w \Leftarrow u$ is primitive.

Lemma 1. *Let $u \in \Sigma^+$ and $u_s \in \mathrm{Suff}(u)$. If $u_s a u \in \mathrm{Pref}(u^2)$ for some $a \in \Sigma$, then u is a power of a.*

Proof. Due to the assumption, $u = u_s a u'_p = u'_p u_s a$ for some $u'_p \in \Sigma^*$. It well known that, for two words $u, v \in \Sigma^+$, if $uv = vu$, then they share their primitive roots. Therefore, the primitive root of u is same as that of $u_s a$. Hence, if u_s is empty, it is clear that $u \in a^+$. Even, otherwise, since $u_s \in \mathrm{Suff}(u'_p u_s a)$, u_s is a power of a. Thus, this lemma holds. □

Theorem 1. *Let $u, w \in \Sigma^+$. Then $w \Leftarrow u$ is not primitive if and only if w and u are unary words over the same letter $a \in \Sigma$.*

Proof. The if-direction is trivial. So we consider here the only-if direction under the assumption that $w \Leftarrow u$ is non-primitive. Then $w \Leftarrow u$ overlaps with its square in a nontrivial way. Let $w = a_1 a_2 \ldots a_n$ for some $n \geq 1$ and $a_1, \ldots, a_n \in \Sigma$. Also let $w \Leftarrow u = v^k$ for some $v \in \Sigma^+$ and $k \geq 2$. In the following, we prove that in all possible cases v is a unary word, which trivially implies what we want.

Firstly we consider the case when there is an integer ℓ such that $ua_1 \cdots ua_\ell = v^i$ for some $i \geq 1$, which further implies that $ua_1 \cdots ua_\ell = a_{n-\ell+1}u \cdots a_n u$. In this case, we can always find such ℓ in the range $\lceil n/2 \rceil \leq \ell$. For such ℓ, this equation implies that all of $a_1, \ldots, a_n$ are the same, say a, and v is a power of a. If $|u| = 1$, this is always the case so that all we have to consider is the case $|u| \geq 2$ under the assumption that such ℓ cannot be found. Note that then we cannot find an integer $\ell' \geq 0$ such that $ua_1 \cdots a_{\ell'} u$ is a power of v, either.

Under the assumption, one of the occurrences of u in $w \Leftarrow u$ overlaps with the factor u^2 of $(w \Leftarrow u)^2$ nontrivially ($x \neq \lambda$ and $y \neq \lambda$ in Fig. 1.) As shown there, we have $u_s a_m u \in \mathrm{Pref}(u^2)$ for some $1 \leq m \leq n$. Lemma 1 implies that u is a unary word over a_m longer than 1. Note that the overlap between $w \Leftarrow u$ and its square implies that for all $1 \leq i \leq n$, $a_i = a_n$ because these characters in $w \Leftarrow u$ must be contained within some u in $(w \Leftarrow u)^2$. □

As mentioned before, $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$ is not always the case. Even if we limit L_1 and L_2 to be singletons $\{w\}$ and $\{u\}$, $(w \Leftarrow u) \Rightarrow u$ can contain

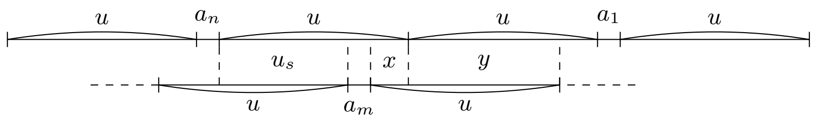


Fig. 1. How $ua_m u$ overlaps with $ua_n u^2$

words except w. Since parallel insertion of a word into another word certainly generates a singleton, it is the parallel deletion that creates such words. We initiate our investigation on this problem with a more general question: under what conditions, parallel deletion results in a singleton.

Note that $w \Rightarrow u = \emptyset$ if and only if w does not contain u as its infix. In the following, we only consider cases where w contains u as its infix. Also, note that two occurrences of u in w have to overlap in a nontrivial manner for $w \Rightarrow u$ not to be a singleton. If u is unbordered, two occurrences of u never overlap non-trivially regardless of what w is. Thus we have the following proposition.

Proposition 1. *If $u \in \Sigma^*$ is unbordered, then $w \Rightarrow u$ is a singleton for any word $w \in \Sigma^*$ that contains u as its infix.*

This also suggests that, even for a bordered word u, $w \Rightarrow u$ is at most a singleton as long as the form of w guarantees that nontrivial overlaps between u's do not occur in it. We will give a necessary and sufficient condition for $w \Rightarrow u$ to be a singleton in the case when w and u share the same primitive root.

Proposition 2. *For $a \in \Sigma$, let $w = a^j$ and $u = a^k$ for some $j \geq k \geq 1$. Then $w \Rightarrow u$ is a singleton if and only if either $k = 1$, $k \leq j \leq 2k-1$, or $j = 3k-1$.*

Proof. We consider the if-direction first. If $k = 1$, then this operation results in a singleton of the empty word. If $j < k$, then we cannot delete any u from w so that $w \Rightarrow u = \{w\}$. If $k \leq j \leq 2k-1$, then by the definition of parallel deletion, the operation deletes exactly one u from w, and hence $w \Rightarrow u = \{a^{j-k}\}$. In the case when $j = 3k-1$, we let $w = a^{i_1}a^k a^{i_2}$ for some $0 \leq i_1 < k$. Then $k \leq i_2 \leq 2k-1$. We know that $a^{i_2} \Rightarrow u = \{a^{i_2-k}\}$. Hence $w \Rightarrow u$ is a singleton.

On the other hand, we show that if k and j do not satisfy these conditions, then $w \Rightarrow u$ contains at least two elements. If $2k \leq j \leq 3k-2$, then it is clear that we can delete two u's from w. In addition, we can write w as $a^{k-1}a^k a^{j-2k-1}$. Since $j-2k-1 < k$, $a^{k-1}a^{j-2k-1}$ is also included in $w \Rightarrow u$. In the case $3k \leq j$, note that $(a^{2k} \Rightarrow u)(a^{j-2k} \Rightarrow u) \subseteq w \Rightarrow u$. We know that $(a^{2k} \Rightarrow u)$ is not a singleton, and hence $w \Rightarrow u$ cannot be a singleton. □

Since a primitive word cannot be a proper infix of its square [17], this proposition has the following corollary.

Corollary 1. *Let $w = g^j$ and $u = g^k$ for some primitive word g and $j \geq k \geq 1$. Then $w \Rightarrow u$ is a singleton if and only if either $k = 1$, $k \leq j \leq 2k$, or $j = 3k-1$.*

Next we consider the more general case when w and u may have distinct primitive roots. If the primitive root of u is unbordered, then we can give a condition similar to the one given in Proposition 2. The proof for this proposition works to prove the next proposition.

Proposition 3. *Let $w \in \Sigma^*$ and $u = g^k$ for some unbordered primitive word g and $k \geq 1$. If the following condition holds, then $w \Rightarrow u$ is a singleton.*
(Condition) whenever $w = w_p g^j w_s$ for some $w_p, w_s \in \Sigma^*$ with $g \notin \mathrm{Suff}(w_p)$ and $g \notin \mathrm{Pref}(w_s)$, and $j \geq 1$, either $k = 1$, $k \leq j \leq 2k-1$, or $j = 3k-1$.

Now we consider the main problem of finding conditions for $(L_1 \Leftarrow L_2) \Rightarrow L_2$ to be equal to L_1. We start our investigation of this problem with the special case when $L_1 = \{w\}$ and $L_2 = \{u\}$. Hence our first aim is to clarify when $(w \Leftarrow u) \Rightarrow u$ does not contain any word other than w. If either w or u is the empty word, then $(w \Leftarrow u) \Rightarrow u$ is always $\{w\}$. Therefore in the remainder of this paper we will assume, without loss of generality, that u and w are nonempty. Let $w = a_1a_2 \cdots a_n$ for some $n \geq 1$ and $a_1, \ldots, a_n \in \Sigma$. In order for the parallel deletion to create another word besides w, there must exist at least two different ways to parallel-delete the occurrences of u from $w \Leftarrow u$. In other words, we have to delete some occurrences of u that have not been parallel-inserted into w. Formally speaking, u has to be a proper infix of ua_iu for some $1 \leq i \leq n$. Based on this idea, we define the set:

$$X = \big\{u \in \Sigma^+ \mid \mathrm{pref}_x(u) \neq \mathrm{suff}_x(u) \text{ or } \mathrm{pref}_y(u) \neq \mathrm{suff}_y(u) \\ \text{for any } (x, y) \in N^2 \text{ with } x + y + 1 = |u|\big\}.$$

Informally, X contains words u which cannot be proper infixes of ubu for any letter $b \in \Sigma$. For such words $u \in X$, there cannot exist two different ways to parallel-delete the occurrences of u from $w \Leftarrow u$, and hence we have the following result.

Proposition 4. *If $u \in X$, then $(w \Leftarrow u) \Rightarrow u = \{w\}$ for any $w \in \Sigma^*$.*

In the following, we give a characterization of X. First of all, no unary word can be in X. By the informal definition of X, the set of all unbordered words of length at least 2, denoted by $U^{>1}$, is a subset of X. Let $N_{(>1)}$ denote the set of all non-primitive words whose primitive root is of length at least 2. The next result shows that no word u in $N_{(>1)}$ can be a proper infix of ubu, for any $b \in \Sigma$.

Lemma 2. $N_{(>1)} \subseteq X$.

Proof. Suppose that there were $u \in N_{(>1)}$ such that $u \notin X$. Let $u = g^i$ for some primitive word g of length at least 2 and $i > 1$. Also we can let $u = u_s a u_p$ for some $u_s \in \mathrm{Suff}(u)$, $a \in \Sigma$, and $u_p \in \mathrm{Pref}(u)$. The equation $g^i = u_s a u_p$ implies that this a is inside one and only one of these g's. Since g^2 cannot overlap with g in any nontrivial way, either u_s or u_p is a power of g. We only consider the case when $u_s = g^j$ for some $j \geq 1$; the other can be proved in a similar way. Then $au_p = g^{i-j}$. Since $u_p \in \mathrm{Pref}(g^i)$, this means g is a power of a, a contradiction with the primitivity of g. □

Let Q_B be the set of all bordered primitive words. Any word in Q_B can be written as $w = (\alpha\beta)^k\alpha$ for some primitive word $\alpha\beta$, and $k \geq 1$. We partition Q_B into two sets. The first one, $Q_B^{(=1)}$, denotes the set of all bordered primitive words w that can be written as $(\alpha\beta)^k\alpha$ with $|\beta| = 1$. The second one is simply the complement, $Q_B^{(>1)} = Q_B \setminus Q_B^{(=1)}$. For example, $aaabaa, abbabba \in Q_B^{(>1)}$ while $aabaabaa \in Q_B^{(=1)}$. This is because even though we can regard $aabaabaa$ as $\alpha\beta\alpha$ with $\alpha = a$ and $\beta = abaaba$, we can also consider it as $(\alpha'\beta')^2\alpha'$, where $\alpha' = aa$ and $\beta' = b$.

The next result shows that every bordered primitive word w that can only be written as $(\alpha\beta)^k\alpha$ such that $\alpha\beta$ is primitive, $k \geq 1$, and $|\beta|$ cannot be 1, cannot be a proper infix of waw for any $a \in \Sigma$. Formally, we have

Lemma 3. $Q_B^{(>1)} \subseteq X$.

Proof. Suppose that there exists $u \in Q_B^{(>1)}$ but $u \notin X$. This means that $u = u_s a u_p$ for some $u_s \in \mathrm{Suff}(u)$ and $u_p \in \mathrm{Pref}(u)$ and $a, b \in \Sigma$ such that $u = u_p b u_s$. The Parikh vector of a word contains the occurrences of each letter in Σ. Since the Parikh vectors of u_p and u_s together contain the same number of occurrences of each letter in $u_s a u_p$ and $u_p b u_s$, we can obtain $a = b$ and hence $u = u_p a u_s$. Due to a well known result mentioned in Sect. 2, there exist $\alpha, \beta \in \Sigma^*$ such that $u_s a = (\alpha\beta)^i$ and $u_p = \alpha(\beta\alpha)^j$ for some $i \geq 1$ and $j \geq 0$ and $\beta\alpha$ is primitive. Then $ua = u_p a u_s a = u_p a(\alpha\beta)^i = \alpha(\beta\alpha)^{i+j}a$, and hence the suffix of length $|\alpha\beta| + 1$ of ua is $b\alpha\beta = \beta\alpha a$. Again, based on the Parikh vector of this suffix, $b = a$, i.e., $a\alpha\beta = \beta\alpha a$. Note that $|\beta| \geq 2$ because $u \in Q_B^{(>1)}$ and hence a is a proper suffix of β. Therefore, this equation means that $\beta\alpha$ overlaps with its square in a nontrivial way, a contradiction with its primitivity. □

The next result states that any word w that is either a unary word or a bordered primitive word that can be written as $(\alpha\beta)^k\alpha$ with $\alpha\beta$ being primitive, $k \geq 1$, and $|\beta| = 1$, can be a proper infix of waw for some $a \in \Sigma$.

Lemma 4. $\left(Q_B^{(=1)} \cup \{a^i \mid a \in \Sigma, i \geq 1\}\right) \cap X = \emptyset$.

Proof. As mentioned above, any unary word cannot be in X. Let $w \in Q_B^{(=1)}$. By definition, there exist $\alpha \in \Sigma^+$ and $b \in \Sigma$ such that αb is primitive and $w = (\alpha b)^k\alpha$ for some $k \geq 1$. Then w is a proper infix of wbw, and hence $w \notin X$. □

The next proposition characterizes the set of all words u that cannot be a proper infix of uau for any $a \in \Sigma$, as being either unbordered words of length greater than 1, or bordered primitive words of the form $(\alpha\beta)^k\alpha$ such that $\alpha\beta$ is primitive, $k \geq 1$, and $|\beta|$ cannot be 1, or non-primitive words whose primitive root has length longer than 1.

Proposition 5. $X = U^{>1} \cup Q_B^{(>1)} \cup N_{(>1)}$.

Proof. Note that $\Sigma^+ = U^{>1} \cup Q_B \cup N_{(>1)} \cup \{a^i \mid a \in \Sigma, i \geq 1\}$. Combining Lemmas 2, 3, and 4 together, we can reach this proposition. □

As mentioned in Proposition 4, u being an element of X is sufficient for it to satisfy $(w \Leftarrow u) \Rightarrow u = \{w\}$ for any word w. In the following, we give necessary and sufficient conditions for the equality to be true in the case when $u \notin X$, that is, either u is unary or $u \in Q_B^{(=1)}$.

Proposition 6. *Let $w \in \Sigma^*$ and $u = a^k$ for some $a \in \Sigma$ and $k \geq 1$. Then $(w \Leftarrow u) \Rightarrow u = \{w\}$ if and only if*

1. *if $k = 2$, then $aa \notin F(w)$;*
2. *otherwise, $w \in (\Sigma \setminus \{a\})^*$.*

Proof. If w contains aa as its infix, then $a^{3k+2} \in \mathrm{F}(w \Leftarrow u)$. Proposition 3 implies that $(w \Leftarrow u) \Rightarrow u$ is not a singleton. Next we consider the case when w contains no aa but a as its infix, and $k = 2$. Then $a^5 \in \mathrm{F}(w \Leftarrow u)$. Since $5 = 3k - 1$, $(w \Leftarrow u) \Rightarrow u$ is a singleton due to the proposition. It is clear that for $w \in (\Sigma \setminus \{a\})^*$, $(w \Leftarrow u) \Rightarrow u = \{w\}$. $\square$

Having considered the case of u being unary, now the only one remaining case is when u is an element of $Q_B^{(=1)}$. For such a word u, there exist $\alpha \in \Sigma^+$, $b \in \Sigma$, and $k \geq 1$ such that $u = (\alpha b)^k \alpha$. We define $M_u = \{a \in \Sigma \mid u \in \mathrm{Suff}(u) a \mathrm{Pref}(u)\}$. By definition, $M_u \neq \emptyset$ if and only if $u \notin X$.

Lemma 5. *For a bordered primitive word u, if $b \in M_u$, then there exists a nonempty word $\alpha \in \Sigma^+$ such that $u = \alpha(b\alpha)^k$ for some $k \geq 1$ and αb is primitive.*

Proof. Since $b \in M_u$, $u = u_p b u_s = u_s b u_p$ for some $u_p, u_s \in \Sigma^*$. Then $u_s b = (\alpha\beta)^i$ and $u_p = \alpha(\beta\alpha)^j$ for some $i \geq 1$, $j \geq 0$, and $\alpha, \beta \in \Sigma^*$ such that $\alpha\beta$ is primitive. Suppose that α were empty. Then $u = \beta^{i+j}$. On one hand, $i + j$ has to be 1 because u is primitive; on the other hand, $i + j \geq 2$ because u_p cannot be empty, otherwise, u is a unary word over b longer than 2. Thus, α is nonempty. So $ub = u_p b u_s b = \alpha(\beta\alpha)^{i+j} b$, and hence $b(\alpha\beta)^i = (\beta\alpha)^i b$. Since $\alpha\beta$ is primitive, β has to be of length 1, and hence $\beta = b$. $\square$

Lemma 6. *For $u \in Q_B^{(=1)}$, $|M_u| = 1$.*

Proof. Suppose $|M_u| > 1$, say two distinct characters b, d are in M_u. Then Lemma 5 implies that $u = \alpha(b\alpha)^i = \gamma(d\gamma)^j$ for some $i, j > 0$ and $\alpha, \gamma \in \Sigma^*$ such that both αb and γd are primitive. Without loss of generality, we assume $|\alpha b| > |\gamma d|$. Then by Fine-and-Wilf's theorem [12], $i = 1$. Hence $u = \alpha b \alpha = \gamma(d\gamma)^j$. If j is odd, then clearly $b = d$, a contradiction. Otherwise, $\alpha = (\gamma d)^{j/2}\gamma_p = \gamma_s(d\gamma)^{j/2}$ and $\gamma = \gamma_p b \gamma_s$ for some $\gamma_p, \gamma_s \in \Sigma^*$ of same length. Then we have $(\gamma d)^{j/2-1}\gamma d \gamma_p = \gamma_s (d\gamma)^{j/2-1} d\gamma_p b \gamma_s$, and hence $b = d$, the same contradiction. $\square$

Proposition 7. *Let $u \in Q_B^{(=1)}$. Then $(w \Leftarrow u) \Rightarrow u = \{w\}$ for $w \in \Sigma^+$ if and only if $w \in (\Sigma \setminus M_u)^+$.*

Proof. If w does not contain any letter in M_u, then it is clear that $(w \Leftarrow u) \Rightarrow u = \{w\}$.

We prove the converse implication. Due to Lemmas 5 and 6, $M_u = \{b\}$ and there exists $\alpha \in \Sigma^+$ such that $u = \alpha(b\alpha)^k$ for some $k \geq 1$ and αb is primitive. Let $w = a_1 \cdots a_n$ for some $n \geq 1$ and $a_i \in \Sigma$ for all $1 \leq i \leq n$, and assume that w contains b. Then we can find an integer $1 \leq m \leq n$ such that $a_{m-1} \neq b$ (if any), $a_m = \cdots = a_{m+j-2} = b$, and $a_{m+j-1} \neq b$ (if any) for some $j \geq 2$. Now

$$w \Leftarrow u = ua_1 \cdots ua_{m-1}[\alpha(b\alpha)^k b\alpha(b\alpha)^k b \cdots b\alpha(b\alpha)^k]a_{m+j-1}u \cdots a_n u.$$

We can parallel-delete u's from the bracketed infix in two ways: one is to delete j u's that were actually inserted by the preceding insertion; the other is to leave the first $\alpha\beta$ and delete u from every $(k+1)|\alpha\beta|$ position. Note that in the latter way, we delete exactly $j-1$ u's. If in both cases, we parallel-delete the inserted u's from the prefix and suffix, then we obtain two distinct words $w, a_1 \cdots a_{m-1}\alpha b b^{j-2}(b\alpha)^k a_{m+j-1} \cdots a_n$. We still need to check that the latter parallel deletion is valid. For that, it is enough to check that neither $a_{m-1}\alpha b$ or $(b\alpha)^k a_{m+j-1}$ contain u. Their lengths are at most $|u|$ so that if one of them contains u, then it is u itself. However, this is not the case because of the primitivity of αb and $\alpha \neq \lambda$. □

Since

$$u \in \Sigma^+ = \underbrace{N_{(>1)} \cup \{aa^+ \mid a \in \Sigma\}}_{\text{non-primitive}} \cup \underbrace{\Sigma \cup U^{>1} \cup Q_B^{(=1)} \cup Q_B^{(>1)}}_{\text{primitive}},$$

Propositions 4, 5, 6, 7 completely characterize the solutions to the equation $(w \Leftarrow u) \Rightarrow u = \{w\}$.

Hence now we are ready to consider the more general equation $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$. When L_2 is a singleton, say $L_2 = \{u\}$, the set X plays an important role.

Proposition 8. *If $u \in X$, then $(L \Leftarrow u) \Rightarrow u = L$ for any language $L \subseteq \Sigma^*$.*

Proof. By definition, $(L \Leftarrow u) \Rightarrow u = \bigcup_{w \in L}(w \Leftarrow u) \Rightarrow u$. Then this result is immediate from Proposition 4. □

4 Comma Codes

In the previous section, we saw that if $u \in X$, then $(L \Leftarrow u) \Rightarrow u = L$ for any language $L \subseteq \Sigma^*$. The aim of this section is to introduce a new language family with the property that if a language L_2 belongs to this family, then $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$ holds for any language $L_1 \subseteq \Sigma^*$.

Definition 1. *A set $L \subseteq \Sigma^+$ is called a* comma code *if $L\Sigma L \cap \Sigma^+ L \Sigma^+ = \emptyset$.*

Intuitively, a comma code is a set L with the property that none of its words can be a proper infix of $u_1 a u_2$ where u_1 and u_2 are words in L, and $a \in \Sigma$ is a "comma". As it turns out (Corollary 2) a comma code is indeed a code.

As examples, $L = \{ab^k a \mid k > 1\}$ is a comma code, while any language that contains unary words or words in $Q_B^{(=1)}$ is not a comma code.

Theorem 2. *If the language $L_2 \subseteq \Sigma^+$ is a comma code, the equation $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$ holds for any language $L_1 \subseteq \Sigma^*$.*

The definition of comma codes reminds us of that of comma-free codes. A nonempty set $L \subseteq \Sigma^+$ is a *comma-free code* if $L^2 \cap \Sigma^+ L \Sigma^+ = \emptyset$. Recall that a nonempty set $L \subseteq \Sigma^+$ is an *infix code* if $L \cap (\Sigma^* L \Sigma^+ \cup \Sigma^+ L \Sigma^*) = \emptyset$, and that a comma-free code is an infix code [17]. We establish a relationship among these three codes, which leads us to the fact that comma codes are actually codes.

Lemma 7. *For a language $A \subseteq \Sigma^*$, A is a comma code if and only if $A\Sigma$ is a comma-free code.*

Proof

(If) We assume that $A\Sigma$ is a comma-free code, and suppose that A were not a comma code. Then there exist $w_1, w_2, w_3 \in A$, $a \in \Sigma$, and $x, y \in \Sigma^+$ such that $w_1 a w_2 = x w_3 y$. By putting some $b \in \Sigma$ at the ends of both sides, we can reach a contradiction with $A\Sigma$ being a comma-free code.

(Only-if) Suppose that $A\Sigma$ were not a comma-free code. Then we have $u_1 a_1 u_2 a_2 = x' u_3 a_3 y'$ for some $u_1, u_2, u_3 \in A$, $a_1, a_2, a_3 \in \Sigma$, and $x', y' \in \Sigma^+$. Since y' is nonempty, we can cut the rightmost letters of both sides from this equation, and reaches the contradiction. □

Lemma 8. *For a language $A \subseteq \Sigma^*$, A is an infix code if and only if $A\Sigma$ is an infix code.*

Proof. The only-if direction is trivial because the family of infix codes is closed under concatenation. As of the if direction, under the assumption that $A\Sigma$ is an infix code, suppose that A were not. Then there exist $u \in A$ and $x, y \in \Sigma^*$ such that $xuy \in A$ and $xy \neq \lambda$. Then for any $b \in \Sigma$, $xuyb \in A\Sigma$, which contains $uc \in A\Sigma$ as its factor, where c is a first letter of yb. Since $uc \neq xuyb$, this is a contradiction. □

Corollary 2. *A comma code is an infix code, and hence a code.*

Actually, the family of comma codes is a proper subset of the family of infix codes. For example, $L = \{ab, ba\}$ is an infix code, but not a comma code. Hence we give a characterization of infix codes which are comma codes. For this purpose, we define the following terms:

$$\begin{aligned}
L_{\overline{p}} &= \{x \in \Sigma^+ \mid xy, yz \in L \text{ for some } y, z \in \Sigma^+\},\\
L_i &= \{y \in \Sigma^+ \mid xy, yz \in L \text{ for some } x, z \in \Sigma^+\},\\
L_{\overline{s}} &= \{z \in \Sigma^+ \mid xy, yz \in L \text{ for some } x, y \in \Sigma^+\},\\
L_{\overline{\overline{p}}} &= \{x \in \Sigma^+ \mid xa \in L_{\overline{p}} \text{ for some } a \in \Sigma\},\\
L_{\overline{\overline{s}}} &= \{x \in \Sigma^+ \mid ax \in L_{\overline{s}} \text{ for some } a \in \Sigma\}.
\end{aligned}$$

Proposition 9 ([16]). *Let $L \subseteq \Sigma^+$. If L is an infix code, then the following four conditions are equivalent and make L a comma-free code: (1) $L_{\overline{s}} \cap L_i = \emptyset$, (2) $L_{\overline{p}} \cap L_i = \emptyset$, (3) $L \cap L_{\overline{s}} L_{\overline{s}} = \emptyset$, and (4) $L \cap L_{\overline{p}} L_{\overline{p}} = \emptyset$. Conversely, if L is a comma-free code, then L is an infix code with these properties.*

Proposition 10. *Let $L \subseteq \Sigma^+$. If L is an infix code such that $L \cap \Sigma = \emptyset$ and $(L_{\overline{s}} \cup L_{\overline{p}}) \cap \Sigma = \emptyset$, then the following four conditions are equivalent and make L a comma code: (1) $L_{\overline{\overline{s}}} \cap L_i = \emptyset$, (2) $L_{\overline{\overline{p}}} \cap L_i = \emptyset$, (3) $L \cap L_{\overline{\overline{s}}} L_{\overline{s}} = \emptyset$, and (4) $L \cap L_{\overline{p}} L_{\overline{\overline{p}}} = \emptyset$. Conversely, if L is a comma code, then L is an infix code with these properties.*

Proof. Note that the emptiness of $L \cap \Sigma$ and $(L_{\overline{s}} \cup L_{\overline{p}}) \cap \Sigma$ is the minimal requirement for L to be a comma code.

(Only-if) Lemma 7 implies that $L\Sigma$ and ΣL are comma-free codes. Using Proposition 9, we have the four properties: (a) $(L\Sigma)_{\overline{s}} \cap (L\Sigma)_i = \emptyset$, (b) $(\Sigma L)_{\overline{p}} \cap (\Sigma L)_i = \emptyset$, (c) $L\Sigma \cap (L\Sigma)_{\overline{s}}(L\Sigma)_{\overline{s}} = \emptyset$, and (d) $\Sigma L \cap (\Sigma L)_{\overline{p}}(\Sigma L)_{\overline{p}} = \emptyset$. Suppose that there were $u \in L_{\overline{\overline{s}}} \cap L_i$. Then there exist $x, y, z, w \in \Sigma^+$ and $a \in \Sigma$ such that $xy, yau, zu, uw \in L$. Let $w = bw'$ for some $w' \in \Sigma^*$. Then $xya, yaub \in L\Sigma$ and hence $ub \in (L\Sigma)_{\overline{s}}$. Moreover, $zub, ubw'c \in L\Sigma$ for any $c \in \Sigma$, and hence $ub \in (L\Sigma)_i$. These two results cause a contradiction with the property (a). The 2nd one derives from the property (b) in the same manner. Next we prove the 3rd property from (c). Suppose that $L \cap L_{\overline{\overline{s}}}L_{\overline{s}} \neq \emptyset$. Then there exist $x, y, z, w, u, v \in \Sigma^+$ and $a \in \Sigma$ such that $xy, yau, zw, wv \in L$ and $uv \in L$. Let $v = bv'$ for some $v' \in \Sigma^*$. Then $xya, yaub, zwb, wbv'c \in L$ for any $c \in \Sigma$. Thus, $ub, v'c \in (L\Sigma)_{\overline{s}}$ and $ubv'c \in L\Sigma$, a contradiction. The 4th derives from the property (d) in this way.

(If) Suppose L were not a comma code. Then there exist $u, v, w \in L$, $x, y \in \Sigma^+$, and $a \in \Sigma$ such that $uav = xwy$. Since $L \cap \Sigma = \emptyset$, $(L_{\overline{s}} \cup L_{\overline{p}}) \cap \Sigma = \emptyset$, and L is an infix code, $u = x\alpha$, $v = \beta y$, and $w = \alpha a \beta$ for some $\alpha, \beta \in \Sigma^+$. Therefore, $\beta \in L_{\overline{\overline{s}}} \cap L_i$, $\alpha \in L_{\overline{\overline{p}}} \cap L_i$, $\beta y \in L \cap \in L_{\overline{\overline{s}}}L_{\overline{s}}$, and $x\alpha \in L \cap \in L_{\overline{p}L_{\overline{\overline{p}}}}$. These contradict the properties 1-4. $\square$

Example 3. Let $L_1 = \{aba, abba\}$. While this is a comma-free code, $abababa \in L\Sigma L \cap \Sigma^+ L \Sigma^+$ and hence L_1 is not a comma code. On the other hand, let us consider $L_2 = \{aaab, abab\}$. This is a comma code but not a comma-free code because any element of comma-free codes has to be primitive [17]. Moreover, there is a language which is both a comma and comma-free code. An example is $L_3 = \{abba, abbba\}$.

This example is enough to verify the following result.

Proposition 11. *The family of comma codes and the family of comma-free codes are incomparable, but not disjoint.*

Another important subfamily of infix codes is the family of solid codes. A nonempty set $L \subseteq \Sigma^+$ is called a *solid code* if L is an infix code and $\mathrm{Pref}(L) \cap \mathrm{Suff}(L) \cap \Sigma^+ = \emptyset$. This is a strict requirement. In fact, if L is a solid code, then all of L_i, $L_{\overline{s}}$, $L_{\overline{p}}$, $L_{\overline{\overline{s}}}$, and $L_{\overline{\overline{p}}}$ are empty. Thus, the following is a corollary of Proposition 10.

Corollary 3. *Let L be a solid code. If $L \cap \Sigma = \emptyset$, then L is a comma code.*

Since there exists a solid code all of whose elements are of length at least 2, this corollary clarifies that the family of solid codes and that of comma codes are not disjoint. However, these two families are incomparable as shown in the next example.

Example 4. Let $L_1 = \{ab, c\}$. This is a solid code, but not a comma code because it contains a word of length 1. On the other hand, L_2 in Example 3 provides an example of a comma code which is not a solid code.

Proposition 12. *The family of comma codes and the family of solid codes are incomparable.*

Next we consider the closure properties of comma codes under certain operations. For alphabets Σ_1, Σ_2, let $f : \Sigma_1^* \to \Sigma_2^*$ be a homomorphism. Then the inverse homomorphism $f^{-1} : \Sigma_2^* \to 2^{\Sigma_1^*}$ is defined as: for $u \in \Sigma_2^*$, $f^{-1}(u) = \{v \in \Sigma_1^* \mid f(v) = u\}$.

Proposition 13. *The family of comma codes is not closed under union, catenation, +, complement, non-erasing homomorphism, and inverse non-erasing homomorphism. On the contrary, it is closed under reversal and intersection with an arbitrary set.*

Proof. The union of comma codes $\{ab\}$ and $\{ba\}$ is not a comma code. The catenation AB of comma codes $A = \{aaba\}$ and $B = \{abaa\}$ is not so because $(aaba)(abaa)b(aaba)(abaa)$ contains $(aaba)(abaa)$ as a proper infix. For a comma code $L = \{abab\}$, $ababababaabab \in L^+\Sigma L^+ \cap \Sigma^+ L^+ \Sigma^+$. Thus L^+ is not a comma code. The complement of a comma code $\{ab\}$ contains a word of length 1 and hence not a comma code. Consider alphabets $\Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{a\}$, and let $f : \Sigma_1^* \to \Sigma_2^*$ be a non-erasing homomorphism defined as $f(a) = f(b) = a$. Then f maps a comma code $\{aaab, abab\}$ onto $\{aaaa\}$, which is not a comma code. Consider alphabets $\Sigma_3 = \{a\}$ and $\Sigma_4 = \{a, b\}$, and let $g : \Sigma_3^* \to \Sigma_4^*$ be a homomorphism defined as $g(a) = ab$. Since $L = \{abab\}$ is a comma code but $g^{-1}(L) = \{aa\}$ is not, the class of comma codes is not closed under inverse non-erasing homomorphisms.

By definition, it is clear that the family of comma codes is closed under reversal or intersection with an arbitrary set. □

Proposition 13 says that the catenation of two comma codes is not always a comma code. So we investigate a condition under which a catenation of two languages A and B becomes a comma code under the assumption that $A \cup B$ is an infix code. Under this assumption, an element of AB can be a proper infix of an element of $AB\Sigma AB$ only in two ways as shown in Fig. 2. The following results offer additional conditions on A and B, which make AB a comma code by preventing both cases in Fig. 2 from occurring.

Proposition 14. *Let $A, B \subseteq \Sigma^*$ such that $A \cup B \neq \emptyset$. If $A \cup B$ is either a comma code or a comma-free code, then AB is a comma code.*

Proof. Suppose that AB were not a comma code. Then there exist $u_1, u_2, u_3 \in A$, $v_1, v_2, v_3 \in B$, and $a \in \Sigma$ such that $u_1v_1au_2v_2 = ru_3v_3s$ for some $r, s \in \Sigma^+$. Since comma-free codes and comma codes are infix codes, then $A \cup B$ is an infix code. Thus, we have the two cases shown in Fig. 2. Nevertheless, they cause a contradiction with $A \cup B$ being a comma or comma-free code. □

Proposition 15. *Let $A, B \subseteq \Sigma^*$ such that $A \cap B = \emptyset$ and $A \cup B$ is an infix code. If $A_{\overline{s}} \cap B_{\overline{p}} = \emptyset$, then AB is a comma code.*

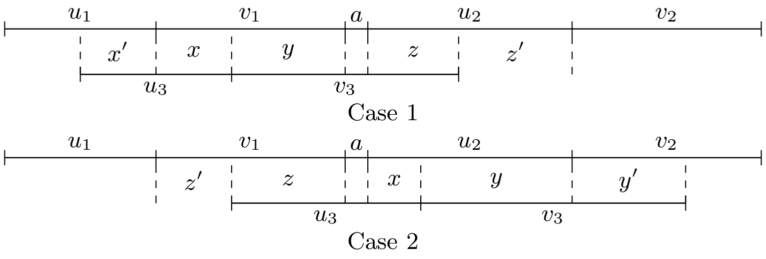


Fig. 2. For $u_1, u_2, u_3 \in A$ and $v_1, v_2, v_3 \in B$, if $A \cup B$ is an infix code, u_3v_3 can be a proper infix of $u_1v_1au_2v_2$ only in these two ways. Note that x' and y in Case 1 can be empty at the same time, and x and y' in Case 2 can be empty at the same time.

Proof. Suppose that AB were not a comma code. Then there exist $u_1, u_2, u_3 \in A$, $v_1, v_2, v_3 \in B$, and $a \in \Sigma$ such that $u_1v_1au_2v_2 = ru_3v_3s$ for some $r, s \in \Sigma^+$. Since $A \cup B$ is an infix code and $A \cap B = \emptyset$, we have only two cases: (1) $u_3 = x'x$, $v_1 = xy$, $v_3 = yaz$, and $u_2 = zz'$, or (2) $v_1 = z'z$, $u_3 = zax$, $u_2 = xy$, and $v_3 = yy'$ for some $x', x, y, z \in \Sigma^+$ and $a \in \Sigma$. Then x in case (1) or y in case (2) is in $A_{\overline{s}} \cap B_{\overline{p}}$, a contradiction. □

Note that the condition in the above proposition is also the condition for AB to be a comma-free code [16]. Therefore, if A and B are two disjoint languages such that $A \cup B$ is an infix code and $A_{\overline{s}} \cap B_{\overline{p}} = \emptyset$, then AB is in the intersection of the family of comma codes and that of comma-free codes.

5 Comma Intercodes

In coding theory, the notion of comma-free code was extended to the more general one of intercode. For $m \geq 1$, a nonempty set $L \subseteq \Sigma^+$ is called an *intercode of index* m if $L^{m+1} \cap \Sigma^+ L^m \Sigma^+ = \emptyset$. An intercode of index 1 is a comma-free code. Based on the similarity between the definition of comma code and that of comma-free code, we introduce the comma intercode as a generalization of comma code.

For $m \geq 1$, a nonempty set $L \subseteq \Sigma^+$ is called a *comma intercode of index* m if $(L\Sigma)^m L \cap \Sigma^+ (L\Sigma)^{m-1} L\Sigma^+ = \emptyset$. It is immediate that a comma intercode of index 1 is a comma code. A language L is called a *comma intercode* if there exists an integer $m \geq 1$ such that L is a comma intercode of index m. First of all, we have to prove that a comma intercode is actually a code. A nonempty set $L \subseteq \Sigma^+$ is a *bifix code* if $L \cap L\Sigma^+ = \emptyset$ (prefix code) and $L \cap \Sigma^+ L = \emptyset$ (suffix code).

Proposition 16. *A comma intercode is a bifix code.*

Proof. Let L be a comma intercode of index m for some $m \geq 1$. Suppose that L were not a prefix code. Then we have $u, w \in L$ such that $w = uv$ for some $v \in \Sigma^+$. This implies that for some $a_1, \ldots, a_m \in \Sigma$, $wa_1wa_2 \cdots a_m w = wa_1(wa_2 \cdots a_m u)v \in \Sigma^+ (L\Sigma)^{m-1} L\Sigma^+$, which contradicts that L is a comma intercode. In the same way, we can prove that L must be a suffix code. Thus, L is a bifix code. □

Like comma codes, a comma intercode consists of only non-unary words of length at least 2. From now, we introduce several properties of comma intercodes.

Proposition 17. *Let L be a regular language. Then for a given integer $m \geq 1$, it is decidable whether or not L is a comma intercode of index m.*

Proof. Since the family of regular languages is closed under catenation and intersection, $(L\Sigma)^m L \cap \Sigma^+(L\Sigma)^{m-1}L\Sigma^+$ is regular. Hence it is decidable whether this language is empty. □

Proposition 18. *Let L be a comma intercode of index m for some $m \geq 1$. Then $L \subseteq X$.*

Proof. Suppose that there were $w \in L$ but $w \notin X$. Then $w = w_s a w_p$ for some $w_s \in \mathrm{Suff}(w)$, $a \in \Sigma$, and $w_p \in \mathrm{Pref}(w)$. This implies that $w = w_p a w_s$. Then $(wa)^m w = w_p a (w_s a w_p a)^{m-1} w_s a w_p a w_s \in \Sigma^+(L\Sigma)^{m-1}L\Sigma^+$, a contradiction. □

Proposition 19. *For any $m \geq 1$, every comma intercode of index m is a comma intercode of index $m+1$.*

Proof. Let L be a comma intercode of index m. By definition, we have $(L\Sigma)^m L \cap \Sigma^+(L\Sigma)^{m-1}L\Sigma^+ = \emptyset$. Suppose that L were not a comma code of index $m+1$. Then $(L\Sigma)^{m+1}L \cap \Sigma^+(L\Sigma)^m L\Sigma^+ \neq \emptyset$. That is, there exist $x_1, \ldots, x_{m+2} \in L$, $y_1, \ldots, y_{m+1} \in L$, $a_1, \ldots, a_{m+1}, b_1, \ldots, b_m \in \Sigma$, and $u, v \in \Sigma^+$ such that $x_1 a_1 \cdots a_{m+1} x_{m+2} = u y_1 b \cdots b_m y_{m+1} v$. Because of L being a comma intercode of index m, $|u| < |x_1|$ and $|v| < |x_{m+2}|$ must hold. However, even so, $y_1 b_1 \cdots b_m y_{m+1}$ is in $\Sigma^+ x_2 a_2 \cdots a_m x_{m+1} \Sigma^+$, and hence $(L\Sigma)^m L \cap \Sigma^+(L\Sigma)^{m-1} L\Sigma^+ \neq \emptyset$. This is a contradiction. □

For any $m \geq 1$, we denote the family of comma intercodes of index m by I_m. Proposition 19 implies that $I_m \subseteq I_{m+1}$ for any $m \geq 1$. This inclusion is actually proper. Let $\{a, b\} \subseteq \Sigma$ and $u_i = ab^i a$ for some $i \geq 1$. Then, for some $a_1, \ldots, a_{m+1} \in \Sigma$, $L = \{u_1 a_1 \cdots u_{m+1} a_{m+1} u_{m+2}, u_2, u_3, \ldots, u_m, u_{m+1}\}$ satisfies the condition $(L\Sigma)^{m+1} L \cap \Sigma^+(L\Sigma)^m L\Sigma^+ = \emptyset$, and hence $L \in I_{m+1}$. On the other hand, $L \notin I_m$. This is because a word $u_1 a_1 \cdots u_{m+1} a_{m+1} u_{m+2} \in \Sigma^+ u_2 a_2 \cdots u_{m+1} \Sigma^+$.

Moreover, let C_b denote the family of bifix codes. Then $\{aba, abba\}$ is in C_b but not in I_m for any $m \geq 1$. Combining Proposition 19 with this example, we have the following hierarchy, where $\subset$ denotes proper inclusion.

Theorem 3. *$I_1 \subset I_2 \subset \cdots \subset I_m \subset \cdots \subset C_b$ holds.*

Let I'_m denote the family of intercodes of index m for any $m \geq 1$. It is known that $I'_1 \subset I'_2 \subset \cdots \subset I'_m \subset \cdots \subset C_b$ holds [16]. Due to these results and Proposition 11, we obtain the following corollary.

Corollary 4. *For any $m, n \geq 1$, the family of intercodes of index m and the family of comma intercode of index n are incomparable.*

Furthermore, we know that the family of comma-free codes and that of comma codes are proper subsets of the family of infix codes. Thus, we can draw the proper inclusion hierarchy of the families of bifix codes, intercodes, comma intercodes, and infix codes as follows.

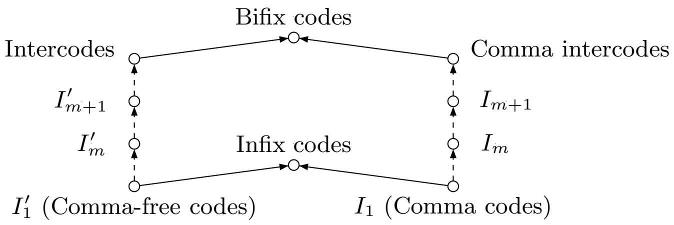


Fig. 3. The inclusion hierarchy of bifix codes, intercodes, comma intercodes, and infix codes, where arrows indicate proper inclusion

Although the definition and some properties of comma intercodes are similar with those of intercodes, we show in the following that these two codes are not similar in terms of synchronous decoding delay. A code L is *synchronously decipherable* if there is a non-negative integer n such that for all $u, v \in \Sigma^*$ and $x \in L^n$, $uxv \in L^*$ implies $u, v \in L^*$. If a code L is synchronously decipherable, then the smallest such n is called the *synchronous decoding delay* of L. It is known that, for a code $L \subseteq \Sigma^+$, L is an intercode of index n if and only if L is synchronously decipherable with delay less than or equal to n [17]. In contrast, comma intercodes do not have such a property.

Proposition 20. *Let $L \subseteq \Sigma^+$ be a comma intercode of index n. Then L is not necessarily synchronously decipherable with delay less than or equal to n.*

Proof. Consider $L = \{abab, aaab\}$, which is a comma intercode of index 1, and hence a comma code of any index. For $m \geq 1$, $aaab(abab)^m = aa(abab)^m ab \in L^{m+1}$ and $(abab)^m \in L^m$ but $aa, ab \notin L$. Therefore, L is not with delay m. □

6 Conclusion

In this paper, we obtained some properties of parallel insertion and deletion, and investigated conditions for the equation $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$ to hold. We obtained a complete characterization of solutions in the special case when L_1 and L_2 are singleton languages. For the general case, we introduced the definition of comma codes and proved that, if L_2 is a comma code, then the equation holds for any language $L_1 \subseteq \Sigma^*$. We also obtained a characterization, some closure properties, and algebraic properties of comma codes, and compared this family of codes with the families of comma-free codes and solid codes. Lastly, we generalized the notion of comma codes to that of comma intercodes of index m. As it turns out, the families of comma intercodes of index m form an infinite proper inclusion hierarchy within the family of bifix codes. The first element of this hierarchy, the family of comma codes, is a subset of the family of infix codes, while the last element of which is a subset of the family of bifix codes. This hierarchy parallels, but is different from, the one that starts with comma-free codes (which are infix codes), and continues with intercodes of index m (which are bifix codes).

Acknowledgement

The authors acknowledge the anonymous referees for their useful comments.

References

1. Berstel, J., Perrin, D.: Theory of Codes. Academic Press. Inc., Orlando (1985)
2. Domaratzki, M.: Deletion along trajectories. Theoretical Computer Science 320, 293–313 (2004)
3. Ito, M., Kari, L., Thierrin, G.: Insertion and deletion closure of languages. Theoretical Computer Science 183, 3–19 (1997)
4. Jürgensen, H., Konstantinidis, S.: The hierarchy of codes. In: Ésik, Z. (ed.) FCT 1993. LNCS, vol. 710, pp. 50–68. Springer, Heidelberg (1993)
5. Kari, L.: On Insertion and Deletion in Formal Languages. Ph.D. Thesis, University of Turku (1991)
6. Kari, L.: Insertion and deletion of words: determinism and reversibility. In: Havel, I.M., Koubek, V. (eds.) MFCS 1992. LNCS, vol. 629, pp. 315–326. Springer, Heidelberg (1992)
7. Kari, L., Thierrin, G.: Words insertions and primitivity. Utilitas Mathematica 53, 49–61 (1998)
8. Kari, L., Mateescu, A., Paun, G., Salomaa, A.: Deletion sets. Fundamenta Informatica 18(1), 355–370 (1993)
9. Kari, L., Mateescu, A., Paun, G., Salomaa, A.: On parallel deletions applied to a word. RAIRO. Theoret. Inform. Appl. 29, 129–144 (1995)
10. Kudlek, M., Mateescu, A.: On distributed catenation. Theoretical Computer Science 180, 341–352 (1997)
11. Kudlek, M., Mateescu, A.: On mix operation. New Trends in Formal Languages. In: Păun, G., Salomaa, A. (eds.) New Trends in Formal Languages. LNCS, vol. 1218, pp. 35–44. Springer, Heidelberg (1997)
12. Lothaire, M.: Algebraic Combinatorics on Words. Cambridge University Press, Cambridge (2002)
13. Manea, F., Mitrana, V., Yokomori, T.: Two complementary operations inspired by the DNA hairpin formation: Completion and reduction. Theoretical Computer Science 410, 417–425 (2009)
14. Mateescu, A., Rozenberg, G., Salomaa, A.: Shuffle on trajectories: syntactic constraints. Theoretical Computer Science 197, 1–56 (1998)
15. Parikh, R.J.: On context-free languages. Journal of the Association for Computing Machinery 13, 570–581 (1966)
16. Shyr, H.J.: Free Monoids and Languages. Lecture Notes, Institute of Applied Mathematics, National Chung-Hsing University, Taichung, Taiwan (2001)
17. Yu, S.S.: Languages and Codes. Lecture Notes, Department of Computer Science, National Chung-Hsing University, Taichung, Taiwan 402 (2005)
18. Yu, S.S.: A characterization of intercodes. Intern. J. Computer Math. 36, 39–48 (1990)

Computation of Pell Numbers of the Form pX^2

Konstantinos A. Draziotis

Technological and Educational Institute of Kavala
Department of Exact Sciences
65 404 Kavala, Greece
drazioti@gmail.com

Abstract. We give an algorithm for the computation of Pell numbers of the form $P_n = px^2$, where p is prime and $x \in \mathbb{Z}$.

1 Introduction

The Pell sequence is defined by the linear recurrence relation :

$$P_0 = 0,\ P_1 = 1,\ P_n = 2P_{n-1} + P_{n-2},\quad n \geq 2.$$

This sequence has many combinatorial meaning. For instance, the number of 132−avoiding two-stack sortable permutations involves Pell and Fibonacci numbers, [3]. Also we have the Pell primality test : "If N is odd prime, then $P_n - \left(\frac{2}{n}\right)$ is divisible by N," where $\left(\frac{2}{n}\right)$ is the Kronecker symbol. For other information and applications of Pell numbers see [14].

The arithmetic properties of Pell numbers of special form have held the interest of many mathematicians. In [7], Ljunggren showed that a Pell number is a positive square only if $n = 1$ or 7. Pethö in [9], proved that these are the only perfect powers of the Pell sequence. For the numbers of the form $P_n = a^m \pm t$, where $m = 2, 3$ and $t \in \{1, 2, 5, 6, 14\}$, some results are obtained in [11]. In [8], it was shown that if k is an integer all of whose prime factors are congruent to 3 modulo 4, then neither term of sequence P_n is of the form kx^2.

Furthermore, Robbins [12], in order to compute Pell numbers of the form $P_n = px^2$, computed the number $z^*(p) = \min\{k :\ p|P_k\}$ and checked if $P_{z^*(p)}$ is equal to px_1^2. Note that, there is not known upper bound for the size of $z^*(p)$, as p is increasing and so is not clear if p is a factor of P_k for some k. So this may cause an endless searching in order to compute $z^*(p)$. For a comparison see example 4 in section 4. This method is very fast in practice, for small primes. Our purpose is to give an algorithm for the computation of Pell numbers of the form $P_n = px^2$, for p fixed prime number. The main idea is a reduction of the problem to the study of the integer points (x, y) to the elliptic curve $Y^2 = X^3 - 32p^2X$, with x even. The determination of these integral points is achieved using the *multiplication by 2 Chabauty method*, [10], [1]. The simplicity and the usefulness of the method is illustrated by examples.

2 The Reduction

Suppose that n is odd and $P_{2n-1} = pr^2$. Since P_n is odd, then necessarily p is an odd prime. A straightforward calculation with the general term of Pell sequence,

$$P_n = \frac{\sqrt{2}}{4}(\lambda_+^n - \lambda_-^n),\ \lambda_\pm = 1 \pm \sqrt{2},$$

gives

$$P_{2n-1}^2 + P_{2n+1}^2 + 4 = 6P_{2n-1}P_{2n+1}. \tag{1}$$

Since $P_{2n-1} = pr^2$ and setting $P_{2n+1} = t$, we get

$$p^2r^4 + t^2 + 4 = 6pr^2t.$$

This equation defines an elliptic curve over $\mathbb{Q}$. Using the map

$$(r,t) \to (X, 3pX^2 + Y),$$

we get the curve $Y^2 = 8p^2X^4 - 4$. Note that if (r,t) is an integer point, then also (X,Y) is integer point. Setting $Y = 2Y''$ we get $Y''^2 = 2p^2X^4 - 1$, thus $Y''^2 \equiv -1/p$. So $(-1/p) = 1$, when $p \equiv 1/4$. Multiplying both parts with $16p^2$, we get $(2pY'')^2 = 2(2pX)^4 - 16p^2$, and this implies $Y'^2 = 2X'^4 - 16p^2$. Finally, setting $x = 2X'^2$ and $y = 2X'Y'$ we get the elliptic curve $y^2 = x^3 - 32p^2x$. So we have to determine its integer points under the condition x to be of the form $2X'^2$. The relation between x, r is $x = 8p^2r^2$ and also $P_{2n-1} = x/8p$. We note that x cannot be a square.

Now if n is even, Theorem 2 of [12] give us the following.

Lemma 1. *If p is an odd and $P_{2m} = px^2$, then $p = 3$ and $m = 2$.*

So the problem of computation of Pell numbers of the form $P_n = pr^2$ splits to two cases. Firstly, if n is odd, then $p \equiv 1/4$ and using the equation (1), we reduce the problem to the study of the elliptic curve $y^2 = x^3 - 32p^2x$. Secondly, if n is even then $n = 4,\ p = 3$. If $p = 2$, then from [12, Theorem 1] we get $n = 2$. Thus in the next section we shall determine the integer points of the elliptic curve $Y^2 = X^3 - 32p^2X$, with X even.

3 Integer Points to $Y^2 = X^3 - 32p^2X$

Let $E : Y^2 = X^3 + AX$. We set $P = (a,b) \in E(\mathbb{Z})$ and let $R = (s,t)$ be a point of E over the algebraic closure $\overline{\mathbb{Q}} \subset \mathbb{C}$ of $\mathbb{Q}$, such that $2R = P$. By [13, chapter 3, p.59], we get

$$a = \frac{s^4 - 2As^2 + A^2}{4(s^3 + As)} \tag{2}$$

and so s is a root of the polynomial

$$\Theta_a(T) = T^4 - 4aT^3 - 2AT^2 - 4AaT + A^2. \tag{3}$$

If $A = -32p^2$, then we get

$$\Theta_a(T) = T^4 - 4aT^3 + 64p^2T^2 + 128p^2aT + 1024p^4.$$

We have

$$0 = \frac{\Theta_a(s)}{s^2} = \Big(s - \frac{32p^2}{s}\Big)^2 - 4a\Big(s - \frac{32p^2}{s}\Big) + 128p^2,$$

whence

$$s = a \pm \sqrt{a^2 - 32p^2} \pm \sqrt{2a^2 \pm 2a\sqrt{a^2 - 32p^2}},$$

where the first $\pm$ coincide with the third. Thus,

$$L = \mathbb{Q}(s) = \mathbb{Q}\big(\sqrt{2a^2 \pm 2a\sqrt{a^2 - 32p^2}}\big). \tag{4}$$

Since a is not a square, then also $a^2 - 32p^2$ is not a square and so L can not be neither a quadratic extension of $\mathbb{Q}$ nor equal to $\mathbb{Q}$. Necessarily L is a quartic extension of $\mathbb{Q}$. Since a is of the form $2r_1^2$, we get that $a^2 - 32p^2 = 2r_2^2$, for some $r_2 \in \mathbb{Z}$. Thus from (4) we get $L = \mathbb{Q}\big(\sqrt{2a^2 \pm 2ar_2\sqrt{2}}\big)$. Note also that $K = \mathbb{Q}(\sqrt{2}) \subset L$. From the form of the number field L we conclude that its Galois group is either the Dihedral group or the Cyclic group of 4 elements or the Klein group. The relation between a, r_1, r_2 allow us to prove the following.

Lemma 2. *The extension $L/\mathbb{Q}$ is a cyclic extension of $\mathbb{Q}$.*

Proof. Since $L/\mathbb{Q}$ is quartic, $\Theta_a(T)$ is an irreducible polynomial. We shall use the result of [6]. The cubic resolvent of $\Theta_a(T)$ is

$$r(T) = (T + 64p^2)(T^2 - 128Tp^2 - 512a^2p^2 + 4096p^4)$$

and the auxiliary polynomial is

$$g(x) = (x^2 + 64p^2x + 1024p^4)(x^2 - 4ax + 128p^2) = (x + 32p^2)^2(x^2 - 4ax + 128p^2).$$

The second factor of $g(x)$ has discriminant $2a \pm 2\sqrt{a^2 - 32p^2} = 2a \pm 2r_2\sqrt{2}$. Remarking that the splitting field of $r(x)$ is $K = \mathbb{Q}(\sqrt{2})$, we conclude that $g(x)$ splits in K, so the Galois group is $\mathbb{Z}_4$.

From [4] or [5] we get that L can be written in a unique way as

$$L = \mathbb{Q}\Big(\sqrt{A(D + B\sqrt{D})}\Big),$$

with $B \geq 1$, A square free and odd, $D \geq 2$ square free, $D - B^2$ is square and $\gcd(A, D) = 1$. Further, using again [4] or [5], the discriminant Δ_L is equal to $2^cA^2D^3$, where $c \in \{0, 4, 6, 8\}$. Since $D = 2$ we get $B = 1$ and $\Delta_L = 2^{c+3}A^2$. From [13, Proposition 1.5, p.193], we get that the number field $K(s)$ is unramified outside the primes dividing the discriminant of E, so L is unramified outside

$\{2, p\}$. Using that $\Delta_L = 2^{c+3}A^2$ and since A is odd, we get $A \in \{\pm 1, \pm p\}$. So the possible number fields are

$$L_{p,+} = \mathbb{Q}\big(\sqrt{2p \pm p\sqrt{2}}\big),\ L_{p,-} = \mathbb{Q}\big(\sqrt{-2p \pm p\sqrt{2}}\big),$$

$$L_{2,+} = \mathbb{Q}\big(\sqrt{2 \pm \sqrt{2}}\big),\ L_{2,-} = \mathbb{Q}\big(\sqrt{-2 \pm \sqrt{2}}\big).$$

The minimal polynomials are

$$f_{\pm}(T) = T^4 \mp 4pT^2 + 2p^2$$

for $L_{p,\pm}$ and

$$g_{\pm}(T) = T^4 \mp 4T^2 + 2$$

for $L_{2,\pm}$. The first two are ramified at $\{2, p\}$ and the other two only at $\{2\}$.

From the set up of our problem we have $a = 4pz$. So $s = 4pr$. Then r is a root of the polynomial

$$\theta_z(T) = T^4 - 4zT^3 + 4T^2 + 8zT + 4.$$

The element

$$r_{\pm} = \frac{r \pm \sqrt{2}}{2}$$

is a root of the polynomial with integer coefficients:

$$\begin{aligned}\lambda(S) &= (1/256)res_W(\theta_z(2T \mp W), W^2 - 2)\\ &= T^8 - 4aT^7 + \cdots + 1,\end{aligned}$$

where $res_W(\cdot,\cdot)$ denotes the resultant of two polynomials with respect to W. Since the constant term is 1, the norm of $r_{\pm}$ shall divide 1. Thus $r_{\pm}$ is a unit in L. So

$$u = \frac{r + \sqrt{2}}{2} \ \text{ and } v = \frac{\sqrt{2} - r}{2}$$

satisfy the unit equation $u + v = \sqrt{2}$ in L. Also from [2, Chapter 9, Proposition 9.4.1, p.461] we get that the polynomial $\theta_z(T)$ defines a totally real quartic extension, thus $\mathbb{Q}(s) = \mathbb{Q}(r)$, is totally real. We conclude therefore that the possible number fields are either

$$L_1 = \mathbb{Q}\big(\sqrt{2 + \sqrt{2}}\big) \ \text{ or } L_2 = \mathbb{Q}\Big(\sqrt{p(2 + \sqrt{2})}\Big).$$

The algorithm of Wildanger [15] which is implemented in the computer algebra system Kant 2.5[1] provide us with the solutions of this unit equation in L. Since $s = 4pr$, then the relation

$$a = \frac{(s^2 + 32p^2)^2}{4s(s^2 - 32p^2)},$$

[1] http://www.math.tu-berlin.de/˜kant

transforms to

$$a = p\frac{(r^2+2)^2}{r(r^2-2)},$$

and from $r = 2u - \sqrt{2}$ we get

$$a = \frac{p((2u-\sqrt{2})^2+2)^2}{(2u-\sqrt{2})((2u-\sqrt{2})^2-2)}.$$

In the case we work in L_1, the solutions of the unit equation are listed in table 1, where we have put $[a_1\ a_2\ a_3\ a_4] = a_0 + a_1\omega_1 + a_2\omega_2 + a_3\omega_3$, and $\{\omega_0 = 1, \omega_1, \omega_2, \omega_3\}$ is an integral basis of the number field L_1. We found that $a = \pm 1352p$ or $\pm 8p$. If we substitute these values to equation $y^2 = x^3 - 32p^2x$, and since p is odd, does not provide us with an integer value for y,

Table 1. The solutions (u, v) of the unit equation $u + v = \sqrt{2}$ in $\mathbb{Q}\big(\sqrt{2+\sqrt{2}}\big)$

[-1,0,0,0] [-1,0,1,0]	[1,0,0,0] [-3 0,1,0]	[-1,-1,0,0] [-1,-1,1,0]
[-1,1,0,0] [-1,-1,1,0]	[-1,-1,1,0] [-1,1,0,0]	[-3,0,1,0] [1,0,0,0]
[407,533,-119,-156] [-409,-533,120,156]	[-1,1,1,0] [-1,-1,0,0]	[-1,0,1,0] [-1,0,0,0]
[-409,533,120,-156] [407,-533,-119,156]	[5,7,-1,-2] [-7,-7,2,2]	[1,4,0,-1] [-3,-4,1,1]
[-71,39,120,-65] [69,-39,-119,65]	[-1,-1,-1,1] [-1,1,2,-1]	[1,2,-3,-2] [-3,-2,4,2]
[69,39,-119,-65] [-71,-39,120,65]	[-7,7,2,-2] [5,-7,-1,2]	[-3,2,4,-2] [1,-2,-3,2]
[-71,-39,120,65] [69,39,-119,-65]	[-1,2,0,-1] [-1,-2,1,1]	[1,3,0,-1] [-3,-3,1,1]
[11,14,-3,-4] [-13,-14,4,4]	[-1,2,1,-1] [-1,-2,0,1]	[-3,3,1,-1] [1,-3,0,1]
[-1,1,-1,-1] [-1,-1,2,1]	[-1,1,2,-1] [-1,-1,-1,1]	[-3,-4,1,1] [1,4,0,-1]
[11,-14,-3,4] [-13,14,4,-4]	[1,-3,0,1] [-3,3,1,-1]	[-1,-2,0,1] [-1,2,1,-1]
[-13,14,4,-4] [11,-14,-3,4]	[-3,-3,1,1] [1,3,0,-1]	[-1,-2,1,1] [-1,2,0,-1]
[-409,-533,120,156] [407,533,-119,-156]	[1,-2,-3,2] [-3,2,4,-2]	[5,-7,-1,2] [-7,7,2,-2]
[69,-39,-119,65] [-71,39,120,-65]	[-1,-1,2,1] [-1,1,-1,-1]	[1,-4,0,1] [-3,4,1,-1]
[-13,-14,4,4] [11,14,-3,-4]	[-3,-2,4,2] [1,2,-3,-2]	[-3,4,1,-1] [1,-4,0,1]
[407,-533,-119,156] [-409,533,120,-156]	[-7,-7,2,2] [5,7,-1,-2]	

If we work with the number field L_2, we have the dependence from p and so the set of solutions of the unit equation varies. So we have the following algorithm.

Input. p odd prime.

Output. The integer solutions of the equation

$$P_n = px^2. \tag{5}$$

1. If $p \equiv 3/4$, then the only solutions of (5) are given by the triple $(p, n, P_n) = (3, 4, 12)$.
2. if $p \equiv 1/4$, then solve the unit equation $u + v = \sqrt{2}$ in $\mathbb{Q}\Big(\sqrt{p(2+\sqrt{2})}\Big)$.
3. Check if $r = 2u - \sqrt{2}$ gives integer value to the expression $c = (r^2+1)^2/(r(r^2-2))$.

4. If $c \notin \mathbb{Z}$, then the equation (5) does not have any solution.
5. If $c \in \mathbb{Z}$, then find the integer n such that $P_n = c/(8p)$. (The values of n from that step, give all the solutions of (5)).

Remark

(**i**) If the rank of the elliptic curve $y^2 = x^3 - 32p^2x$ is 0, then does not have any integer non trivial solution and so the same holds for equation (5).

(**ii**) As we saw in section 2 the solutions to the equation $P_n = px^2$, with p odd prime, is reduced to the study of integral points to the elliptic curve $Y^2 = X^3 - 32p^2X$. If instead of the prime p we consider a square free integer k, then again considering n odd, we get the elliptic curve $Y^2 = X^3 - 32k^2X$. If the rank of this curve is zero then necessarily the equation $P_n = kx^2$ does not have any solution for n odd.

4 Examples

All the computations are implemented with Kash 2.5. We assume that our hardware and mainly the software was working properly.

1. $p = 5$. We are interested in the equation $P_n = 5r^2$. Since $p \equiv 1/4$, we have to solve the unit equation $u + v = \sqrt{2}$ in the field $\mathbb{Q}\left(\sqrt{5(2+\sqrt{2})}\right)$. From Kash 2.5 we get the following solutions :

$$[[11, 7, -3, -2], [-13, -7, 4, 2]], [[-13, 7, 4, -2], [11, -7, -3, 2]],$$
$$[[1, 1, -3, -1], [-3, -1, 4, 1]],$$
$$[[-3, 1, 4, -1], [1, -1, -3, 1]], [-1, [-1, 0, 1, 0]], [1, [-3, 0, 1, 0]],$$
$$[[-3, 0, 1, 0], 1], [[-1, 0, 1, 0], -1],$$
$$[[1, -1, -3, 1], [-3, 1, 4, -1]], [[-3, -1, 4, 1], [1, 1, -3, -1]],$$
$$[[11, -7, -3, 2], [-13, 7, 4, -2]], [[-13, -7, 4, 2], [11, 7, -3, -2]]$$

   From these solutions we get the integer solution $(200, \pm 2800)$ on the elliptic curve
$$y^2 = x^2 - 800x.$$
   So $P_n = x/8 = 200/40 = 5$. This gives $n = 3$. Thus, $(n, r) = (3, 1)$.

2. $p = 29$. We are interested in the equation $P_n = 29r^2$. Since $p \equiv 1/4$, we have to solve the unit equation $u + v = \sqrt{2}$ in the field $\mathbb{Q}\left(\sqrt{29(2+\sqrt{2})}\right)$. From Kash 2.5 we get the following solutions:

$$[[71, 99, -21, -29], [-69, -99, 20, 29]], [[-69, 99, 20, -29], [71, -99, -21, 29]],$$
$$[[13, -1, -21, 0], [-11, 1, 20, 0]], [[13, 1, -21, 0], [-11, -1, 20, 0]], [[1, 0, -1, 0], 1],$$

$$[[3,0,-1,0],-1],[-1,[3,0,-1,0]],[1,[1,0,-1,0]],$$
$$[[-11,-1,20,0],[13,1,-21,0]],[[-11,1,20,0],[13,-1,-21,0]],$$
$$[[71,-99,-21,29],[-69,99,20,-29]],[[-69,-99,20,29],[71,99,-21,-29]].$$

These, provide us with the integer point $(6728, \pm 551696)$, on the curve $y^2 = x^3 - 26912x$, which give us $P_n = 29$, so $(n,r) = (5,1)$.

3. For all primes $p \equiv 1/4$, $1000 < p < 2000$ we did not get any solution for $P_n = px^2$.
4. For $p = 95317$ it took less than 10 minutes in Kant, in order to compute the solution of the unit equation, and we found that there is not any solution to $P_n = px^2$. Further in Maple, we computed that $z^*(p) > 21000$, and we did not continue the computations further, since only for the lower bound took many hours.

References

1. Bugeaud, Y.: On the size of integer solutions of elliptic equations. Bull. Austral. Math. Soc. 57(2), 199–206 (1998)
2. Cohen, H.: Advanced topics in computational number theory. Graduate Texts in Mathematics, vol. 193., pp. xvi+578. Springer, New York (2000)
3. Egge, E.S., Mansour, T.: 132-avoiding two-stack sortable permutations, Fibonacci numbers, and Pell numbers. Discrete Appl. Math. 143(1-3), 72–83 (2004)
4. Hardy, K., Hudson, R.H., Richman, D., Williams, K.S.: Determination of all imaginary cyclic quartic fields with class number 2. Trans. Am. Math. Soc. 311(1), 1–55 (1989)
5. Huard, J.G., Spearman, B.K., Williams, K.S.: Integral bases for quartic fields with quadratic subfields. J. Number Theory 51(1), 87–102 (1995)
6. Kappe, L.-C., Warren, B.: An elementary test for the Galois group of a quartic polynomial. Amer. Math. Monthly 96(2), 133–137 (1989)
7. Ljunggren, W.: Zur Theorie der Gleichung $\mathrm{x}^2 + 1 = \mathrm{Dy}^4$. Avh. Norsk. Vid. Akad. Oslo, 1–27 (1942)
8. McDaniel, W.L.: On Fibonacci and Pell numbers of the form kx^2 (Almost every term has a 4r + 1 prime factor). Fibonacci Q. 40(1), 41–42 (2002)
9. Pethö, A.: Full cubes in the Fibonacci sequence. Publ. Math. Debrecen 30, 117–127 (1983)
10. Poulakis, D.: Integer points on algebraic curves with exceptional units. J. Austral. Math. Soc. Ser. A 63(2), 145–164 (1997)
11. Ribenboim, P.: Pell numbers, squares and cubes. Publ. Math. Debrecen 54(1-2), 131–152 (1999)
12. Robbins, N.: On Pell numbers of the form px^2, where p is prime. Fibonacci Quart 22(4), 340–348 (1984)
13. Silverman, J.H.: The Aritmetic of Elliptic Curves. Springer, Heidelberg (1986)
14. Sloane, N.J.A.: An On-Line Version of the Encyclopedia of Integer Sequences, http://www.research.att.com/~njas/sequences/A000129
15. Wildanger, K.: Über das Lösen von Einheiten- und Indexformgleichungen in algebraischen Zahlkörpern (German) [Solving unit and index form equations in algebraic number fields]. J. Number Theory 82(2), 188–224 (2000)

Iteration Grove Theories with Applications

Dedicated to Prof. Werner Kuich on the Occasion of His Retirement

Z. Ésik* and T. Hajgató

Dept. of Computer Science, University of Szeged, Hungary

Abstract. Iteration grove theories are iteration theories equipped with an additive structure satisfying certain one-sided distributivity laws. In any iteration grove theory, the fixed point operation determines and is determined by a generalized star operation that takes familiar form in many applications. We relate properties of the dagger operation to properties of the generalized star operation and present some applications to continuous functions over complete lattices, continuous monoids, and to tree languages.

1 Introduction

Fixed point operations occur in just about all areas of theoretical computer science including automata and languages, semantics of programming languages, logical theories of computational systems, etc. Fixed point, or dagger operations are usually defined in Lawvere theories of functions over a set equipped with structure (such as a partial order or a complete metric), or more generally, over abstract Lawvere theories, or cartesian or co-cartesian categories, cf. [2,8,16,21].

In a Lawvere theory, the fixed point operation takes a morphism $f : n \to n+p$ to a morphism $f^\dagger : n \to p$ which provides a solution to the fixed point equation $\xi = f \cdot \langle \xi, \mathbf{1}_p \rangle$. (See the beginning of Section 2 for notation.) If a theory is equipped with an additional structure, such as an additive structure, then the dagger operation is usually related to some "Kleenean operations".

For example, the theory of matrices over a semiring has an additive structure. Under a natural condition, cf. [2], any dagger operation over a matrix theory determines and is determined by a star operation mapping an $n \times n$ square matrix A (i.e., a morphism $A : n \to n$) to an $n \times n$ square matrix A^*. Properties of the dagger operation are then reflected by corresponding properties of the star operation.

For another example, consider a semiring-semimodule pair (S, V) and the theory of augmented matrices over (S, V), cf. [2,7]. In this theory, a morphism $n \to m$ is a pair (A, v) consisting of an $n \times m$ matrix A over the semiring S and an n-dimensional column vector v over the S-semimodule V. This theory also has a natural additive structure and under a natural condition, any dagger operation determines and is determined by two operations, a star operation over the underlying matrix theory of matrices over S as above, and an omega operation mapping

* Partially supported by grant no. K 75249 from the National Foundation of Hungary for Scientific Research.

an $n \times n$ matrix A over S to an n-dimensional column vector A^ω over V. These two operations in turn determine a generalized star operation by

$$(A, v) \quad \mapsto \quad (A, v)^\otimes = (A^*, A^\omega + A^* v),$$

or more generally,

$$(A, B, v) \quad \mapsto \quad (A, B, v)^\otimes = (A^*, A^* B, A^\omega + A^* v),$$

where A is an $n \times n$ matrix and B is an $n \times p$ matrix over S and v is an n-dimensional column vector over V, so that (A, v) and $(A, v)^\otimes$ are morphisms $n \to n$ and (A, B, v) and $(A, B, v)^\otimes$ are morphisms $n \to n + p$.

More generally, one can define generalized star operations in all grove theories, which are theories equipped with an additive structure such that composition distributes over finite sums on the right. (Composition is written in the diagrammatic order.) Such generalized star operations in grove theories were first studied in [2]. Natural sources of the generalized star operation include theories of continuous (or monotone) functions on complete lattices where the additive structure is given by the binary supremum operation, theories of tree languages where the additive structure is given by set union, theories of formal tree series over semirings with pointwise addition, and theories of synchronization trees and theories of synchronization trees with respect to various behavioral equivalences.

In this paper we give a systematic treatment of the generalized star operation in grove theories. After an introductory section on grove theories, in Section 3 we provide axiomatizations of Conway grove theories and iteration grove theories in terms of the generalized star operation. Section 4 is devoted to ordered grove theories. Here we show how the fixed point induction rule can be expressed in terms of the generalized star operation. Last, in Section 5, we apply our results to grove theories of continuous functions on complete lattices, theories of continuous functions over continuous monoids, theories of (regular) tree languages. Due to space limitations, we left out applications to synchronization trees (processes) and their behavioral equivalences, as well as applications to formal tree series.

2 Grove Theories

In this section, we review some concepts from [2]. In any category, we will write the composite of morphisms $f : a \to b$ and $g : b \to c$ in diagrammatic order as $f \cdot g$. The identity morphism corresponding to an object c will be written $\mathbf{1}_c$. When n is a nonnegative integer, we will denote the set $\{1, \ldots, n\}$ by $[n]$.

We start by recalling that a *theory* [16,2] T is a small category whose objects are the nonnegative integers such that each object n is the n-fold coproduct of object 1 with itself. Moreover, we assume that each theory T comes with distinguished coproduct injections $i_n : 1 \to n$, $i \in [n]$, $n \geq 1$, called *distinguished morphisms*. By the coproduct property, for any sequence of morphisms $f_1, \ldots, f_n : 1 \to p$ in T there is a unique morphism $f : n \to p$, usually denoted $\langle f_1, \ldots, f_n \rangle$ such that $i_n \cdot f = f_i$ for all i. The operation implicitly defined by this condition is called *tupling*. In particular, when $n = 0$, we obtain that for each p there is a unique morphism $0 \to p$ that we denote 0_p. Note that

$$\mathbf{1}_n = \langle 1_n, \dots, n_n \rangle$$

holds for all n. In addition, we require that $1_1 = \mathbf{1}_1$ which in turn implies that $\langle f \rangle = f$ for all $f : 1 \to p$. We call a *base morphism* any morphism which is a tupling of distinguished morphisms. For example, 0_n and $\mathbf{1}_n$ are base morphisms. When ρ is a function $[n] \to [p]$, there is an associated base morphism $n \to p$ which is the tupling of the distinguished morphisms $(1\rho)_p, \dots, (n\rho)_p$. A *base permutation* is a base morphism associated with a bijective function. Injective and surjective base morphisms are defined in the same way. Note that a base permutation corresponding to a bijection ρ is an invertible morphism whose inverse is the base permutation corresponding to the inverse of ρ.

When $f : n \to p$ and $g : m \to p$ in a theory T, we define $\langle f, g \rangle$ as the morphism $h : n+m \to p$ with $i_{n+m} \cdot h = i_n \cdot f$ and $(n+j)_{n+m} \cdot h = j_m \cdot g$ for all $i \in [n]$ and $j \in [m]$. And when $f : n \to p$ and $g : m \to q$ then we define $f \oplus g = \langle f \cdot \kappa, g \cdot \lambda \rangle$ where κ is the base morphism corresponding to the inclusion $[p] \to [p+q]$ and λ is the base morphism corresponding to the translated inclusion $[q] \to [p+q]$, $j \mapsto p+j$. Note that the *pairing* operation $\langle f, g \rangle$ and the *separated sum* operation $f \oplus g$ are associative.

A *morphism* $\varphi : T \to T'$ between theories is a functor preserving objects and distinguished morphisms. It follows that any theory morphism preserves the pairing and separated sum operations.

Example 1. A fundamental example of a theory is the theory $\mathbf{Fun}_A$ of functions over a set A. In this theory, a morphism $n \to p$ is a function $f : A^p \to A^n$. Note the reversal of the arrow. Composition is function composition and the distinguished morphisms are the projection functions. It is known, see e.g. [2] that any theory can be embedded in some theory $\mathbf{Fun}_A$.

Below we will consider theories enriched with constant morphisms $+ : 1 \to 2$ and $\# : 1 \to 0$. In any such theory, we define the *sum* of two morphisms $f, g : 1 \to p$, $p \geq 0$ as the morphism $+ \cdot \langle f, g \rangle : 1 \to p$. Moreover, we define $0_{1,p} = \# \cdot 0_p$, for all p. More generally, we define

$$\begin{aligned} f + g &= \langle f_1 + g_1, \dots, f_n + g_n \rangle : n \to p \\ 0_{n,p} &= \langle 0_{1,p}, \dots, 0_{1,p} \rangle : n \to p \end{aligned}$$

for any $f = \langle f_1, \dots, f_n \rangle$, $g = \langle g_1, \dots, g_n \rangle$ and $n, p \geq 0$. Note that $0_{0,p} = 0_p$, for each p. Thus, the set $T(n,p)$ of morphisms $n \to p$ is equipped with a binary sum operation and the constant $0_{n,p}$ which satisfy the following conditions:

$$\begin{aligned} (f+g) \cdot h &= (f \cdot h) + (g \cdot h), \quad f, g : n \to p,\ h : p \to q \\ 0_{n,p} \cdot h &= 0_{n,q}, \quad h : p \to q. \end{aligned}$$

Definition 1. *A* grove theory *[2] is a theory T with additional constants $+ : 1 \to 2$ and $\# : 1 \to 0$ subject to the following conditions:*

$$\begin{aligned}(1_3 + 2_3) + 3_3 &= 1_3 + (2_3 + 3_3)\\ 1_2 + 2_2 &= 2_2 + 1_2\\ 1_1 + 0_{11} &= 1_1.\end{aligned}$$

These conditions imply that each set $T(n, p)$ of morphisms $n \to p$ is a commutative monoid $(T(n, p), +, 0_{n,p})$.

Given grove theories T, T', a *grove theory morphism* is a theory morphism $\varphi : T \to T'$ preserving $+$ and $\#$. It follows that any morphism of grove theories preserves the sum operation on each hom-set, and the constants $0_{n,p}$.

Remark 1. In a grove theory, for every base morphism $\rho : m \to n$ and every pair of morphisms $f, g : n \to p$ it holds that $\rho \cdot (f + g) = (\rho \cdot f) + (\rho \cdot g)$, moreover $\rho \cdot 0_{n,p} = 0_{m,p}$, for all $n, m, p \geqslant 0$.

Example 2. Suppose that L is a complete lattice with least element $\perp$. Thus, each direct power L^n of L is also a complete lattice. Recall that a function $L^p \to L^n$ is *continuous* if it preserves the suprema of (nonempty) directed sets. Let $\mathbf{Cont}_L$ denote the theory of all continuous functions over L. Thus, $\mathbf{Cont}_L$ is the "subtheory" of $\mathbf{Fun}_L$ determined by the continuous functions.

Let $+$ denote the function $L^2 \to L$, $(x, y) \mapsto x \vee y$, the supremum of the set $\{x, y\}$. It follows that for any $f, g : 1 \to p$, $f + g$ is the function $L^p \to L$ mapping $x \in L^p$ to $f(x) \vee g(x)$. Moreover, let $\#$ denote the least element $\perp$ considered as a function $L^0 \to L$. Then $\mathbf{Cont}_L$ is a grove theory. Note that for each n, p, the morphism $0_{n,p}$ is the function $L^p \to L^n$ which maps each $z \in L^p$ to $\perp_n$, the least element of L^n.

Example 3. A *ranked alphabet Σ* is a family of pairwise disjoint sets $(\Sigma_n)_n$ where n ranges over the nonnegative integers. We assume that the reader is familiar with the notion of (finite) *Σ-trees* over a set $X_p = \{x_1, \ldots, x_p\}$ of variables as defined e.g. in [2]. Below we will denote the collection of such trees by $T_\Sigma(X_p)$. Σ-trees form a theory where a morphism $n \to p$ is an n-tuple of trees in $T_\Sigma(X_p)$. Composition is defined by substitution and for each $i \in [n]$, $n \geq 0$ the ith distinguished morphism $1 \to p$ is the tree x_i. See [2] for details.

We build another theory $\mathbf{Lang}_\Sigma$, whose morphisms $1 \to p$ are Σ-tree languages $L \subseteq T_\Sigma(X_p)$, and whose morphisms $n \to p$ are the n-tuples of morphisms $1 \to p$. Let $L : 1 \to p$ and $L' = (L'_1, \ldots, L'_p) : p \to q$. Then $L \cdot L'$ is the collection of all trees in $T_\Sigma(X_q)$ that can be obtained by OI-substitution [6], i.e., the set of those trees t such that there is a tree $s \in L$ such that t can be constructed from s by replacing each leaf labeled x_i for $i \in [p]$ by some tree in L'_i so that different occurrences of x_i may be replaced by different trees. The distinguished morphism i_n is the set $\{x_i\}$, and the morphisms $+$ and $\#$ are the sets $\{x_1, x_2\}$ and $\varnothing$, respectively. It then follows that addition is (component-wise) set union, and each component of any $0_{n,p}$ is $\varnothing$.

3 Grove Theories Equipped with a Dagger or a Generalized Star Operation

In this section, we consider grove theories equipped with a dagger operation and grove theories equipped with a generalized star operation, and under some natural assumptions we establish a correspondence between them in terms of a categorical isomorphism. We then use this isomorphism to relate equational properties of the dagger operation to equational properties of the generalized star operation.

Definition 2. *Suppose that T is a theory. We say that T is a* dagger theory[1] *if T is equipped with a* dagger operation

$$^\dagger : T(n, n+p) \to T(n,p), \quad n,p \geq 0$$

which need not satisfy any particular properties. Moreover, we say that T is a generalized star theory *if T is* grove theory *equipped with a* (generalized) star operation

$$^\otimes : T(n, n+p) \to T(n, n+p), \quad n,p \geq 0$$

which also need not satisfy any particular properties. Morphisms of dagger and generalized star theories also preserve the dagger and generalized star operation.

Notation. In any grove theory T, for any morphism $f : n \to n+p$ let f^τ denote the following morphism: $f^\tau = f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p) : n \to n+n+p$.[2]

Thus, when $T = \mathbf{Fun}_A$, then f^τ is the function $A^{n+n+p} \to A^n$, $(x,y,z) \mapsto f(x,z) + y$, for all $x, y \in A^n$, $z \in A^p$.

Suppose that T is a grove theory which is dagger theory. Then we define a generalized star operation by

$$f^\otimes = (f^\tau)^\dagger : n \to n+p, \tag{1}$$

for all $f : n \to n+p$. We denote by $T_\otimes$ the resulting generalized star theory. Conversely, suppose now that S is a generalized star theory. Then we define a dagger operation on S by

$$f^\dagger = f^\otimes \cdot \langle 0_{n,p}, \mathbf{1}_p \rangle : n \to p, \tag{2}$$

for all $f : n \to n+p$. Let $S_\dagger$ denote the resulting dagger theory which is also a grove theory.

Proposition 1. *The category of grove theories which are dagger theories and satisfy the equation*

$$f^\dagger = (f^\tau)^\dagger \cdot \langle 0_{n,p}, \mathbf{1}_p \rangle, \quad f : n \to n+p \tag{3}$$

[1] In [2], a dagger theory is called a preiteration theory.

[2] Here and from now on we assume that composition has higher precedence than sum.

is isomorphic to the category of those grove theories which are generalized star theories and satisfy

$$f^{\otimes} = (f^{\tau})^{\otimes} \cdot \langle 0_{n,n+p}, \mathbf{1}_{n+p} \rangle, \tag{4}$$

for all $f : n \to n + p$.

Proof. The equations (3) and (4) yield

$$(T_{\otimes})_{\dagger} = T \quad \text{and} \quad (S_{\dagger})_{\otimes} = S$$

for any grove theories T, S such that T is a dagger theory and S is a generalized star theory. Moreover, for any grove theories T, T' which are dagger theories, it holds that any grove theory morphism $T \to T'$ preserving dagger preserves the generalized star operation and is thus a morphism $T_{\otimes} \to T'_{\otimes}$. Conversely, if S, S' are grove theories which are generalized star theories, and if φ is a grove theory morphism $S \to S'$ preserving the generalized star operation, then φ is also a morphism $S_{\dagger} \to S'_{\dagger}$.

By the above proposition, when a grove theory is both a dagger theory and a generalized star theory and the two operations are related by (1) and (2), then properties of the dagger operation are reflected by certain properties of the generalized star operation and vice versa.

The following identities were used in the characterization of the equational properties of the least fixed point operation on continuous (or monotone) functions over cpo's or complete lattices, cf. [2,9]. For the origins of these identities, the reader is referred to [1,5,17,18,20,22].

- *Fixed point identity*: $f^{\dagger} = f \cdot \langle f^{\dagger}, \mathbf{1}_p \rangle, \quad f : n \to n + p$.
- *Parameter identity*: $(f \cdot (\mathbf{1}_n \oplus g))^{\dagger} = f^{\dagger} \cdot g$, where $f : n \to n+p$ and $g : p \to q$.
- *Left zero identity*: $(0_n \oplus f)^{\dagger} = f, \quad f : n \to p$.
- *Right zero identity*: $(f \oplus 0_q)^{\dagger} = f^{\dagger} \oplus 0_q, \quad f : n \to n + p$.
- *Double dagger identity*: $f^{\dagger\dagger} = (f \cdot (\langle \mathbf{1}_n, \mathbf{1}_n \rangle \oplus \mathbf{1}_p))^{\dagger}, \quad f : n \to n + n + p$.
- *Composition identity*: $(f \cdot \langle g, 0_n \oplus \mathbf{1}_p \rangle)^{\dagger} = f \cdot \langle (g \cdot \langle f, 0_m \oplus \mathbf{1}_p \rangle)^{\dagger}, \mathbf{1}_p \rangle$, where $f : n \to m + p$ and $g : m \to n + p$.
- *Permutation identity*: $(\pi \cdot f \cdot (\pi \oplus \mathbf{1}_p))^{\dagger} = \pi \cdot f^{\dagger} \cdot (\pi^{-1} \oplus \mathbf{1}_p)$, where $f : n \to n+p$ and $\pi : n \to n$ is any base permutation with inverse π^{-1}.
- *Pairing identity*: $\langle f, g \rangle^{\dagger} = \langle f^{\dagger} \cdot \langle h^{\dagger}, \mathbf{1}_p \rangle, h^{\dagger} \rangle$ where $f : n \to n + m + p$, $g : m \to n + m + p$ and $h = g \cdot \langle f^{\dagger}, \mathbf{1}_{m+p} \rangle$.

Note that when the fixed point identity holds, then for all $f : n \to n + p$, $f^{\dagger}$ is a solution of the fixed point equation $\xi = f \cdot \langle \xi, \mathbf{1}_p \rangle$ associated with f, where ξ ranges over the morphisms $n \to p$. Below we will provide equivalent forms of the above identities in grove theories that use the generalized star operation instead of dagger, provided that the two operations are related by (1) and (2). By Proposition 1, such a translation is always possible, but we might get rather complicated equations as the result of a direct application of Proposition 1. Some of the equivalences proved below assume the parameter identity. This is no problem for the applications, since any well-behaved dagger operation does satisfy this identity. See also Proposition 3.

Proposition 2. *Suppose that T is a grove theory which is both a dagger theory and a generalized star theory. Suppose that the dagger and generalized star operations are related by (1) and (2). Then the following equivalences hold in T:*

(a) *the fixed point identity holds iff the* generalized star fixed point identity *holds:*
$f^\otimes = f \cdot \langle f^\otimes, 0_n \oplus \mathbf{1}_p \rangle + (\mathbf{1}_n \oplus 0_p)$*, for every morphism* $f : n \to n + p$,

(b) *the parameter identity holds iff the* generalized star parameter identity *holds:*
$f^\otimes \cdot (\mathbf{1}_n \oplus g) = (f \cdot (\mathbf{1}_n \oplus g))^\otimes$*, where* $f : n \to n + p$ *and* $g : p \to q$,

(c) *the left zero identity holds iff the* generalized star left zero identity *holds:*
$(0_n \oplus f)^\otimes = (0_n \oplus f) + (\mathbf{1}_n \oplus 0_p)$*, where* $f : n \to p$,

(d) *the right zero identity holds iff the* generalized star right zero identity *holds:*
$(f \oplus 0_q)^\otimes = f^\otimes \oplus 0_q$*, where* $f : n \to n + p$.

Moreover if the parameter identity *holds in T, then the following equivalences are valid:*

(e) *the double dagger identity holds iff the* generalized double star identity *holds:*

$$(f^\otimes \cdot (\pi \oplus \mathbf{1}_p))^\otimes \cdot \langle 0_{n,n+p}, \mathbf{1}_{n+p} \rangle = (f \cdot (\langle \mathbf{1}_n, \mathbf{1}_n \rangle \oplus \mathbf{1}_p))^\otimes,$$

for all $f : n \to n + n + p$*, where* $\pi = \langle 0_n \oplus \mathbf{1}_n, \mathbf{1}_n \oplus 0_n \rangle$,

(f) *the composition identity holds iff the* generalized star composition identity *holds:*

$$(f \cdot \langle g, 0_n \oplus \mathbf{1}_p \rangle)^\otimes = f^\tau \cdot \langle (g \cdot \langle f^\tau, 0_{m+n} \oplus \mathbf{1}_p \rangle)^\otimes, 0_m \oplus \mathbf{1}_{p'} \rangle \cdot \langle 0_{m,p'}, \mathbf{1}_{p'} \rangle$$

for all $f : n \to m + p$ *and* $g : m \to n + p$*, where* $p' = n + p$*, and* $f^\tau = f \cdot (\mathbf{1}_m \oplus 0_n \oplus \mathbf{1}_p) + (0_m \oplus \mathbf{1}_n \oplus 0_p)$,

(g) *the permutation identity holds iff the* generalized star permutation identity *holds:*

$$(\pi \cdot f \cdot (\pi^{-1} \oplus \mathbf{1}_p))^\otimes = \pi \cdot f^\otimes \cdot (\pi^{-1} \oplus \mathbf{1}_p),$$

for all $f : n \to n + p$ *and base permutation* $\pi : n \to n$ *with inverse* π^{-1},

(h) *the pairing identity holds iff the* generalized star pairing identity *holds:*

$$\langle f, g \rangle^\otimes = \langle f^\otimes \cdot \langle \mathbf{1}_n \oplus 0_{m+p}, k^\otimes \cdot (\pi^{-1} \oplus \mathbf{1}_p), 0_{n+m} \oplus \mathbf{1}_p \rangle, k^\otimes \cdot (\pi^{-1} \oplus \mathbf{1}_p) \rangle.$$

for all $f : n \to n+m+p$ *and* $g : m \to n+m+p$*, where* $\pi = \langle 0_m \oplus \mathbf{1}_n, \mathbf{1}_m \oplus 0_n \rangle : n+m \to m+n$ *with inverse* $\pi^{-1} = \langle 0_n \oplus \mathbf{1}_m, \mathbf{1}_n \oplus 0_m \rangle : m+n \to n+m$ *and*

$$k = g \cdot \langle f^\otimes \cdot (\pi \oplus \mathbf{1}_p), \mathbf{1}_m \oplus 0_n \oplus \mathbf{1}_p \rangle : m \to m + n + p.$$

Proof. **(a)** Suppose that the fixed point identity holds. Then

$$\begin{aligned}
f^\otimes &= (f^\tau)^\dagger \\
&= f^\tau \cdot \langle (f^\tau)^\dagger, \mathbf{1}_{n+p} \rangle \\
&= (f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p)) \cdot \langle f^\otimes, \mathbf{1}_{n+p} \rangle \\
&= f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) \cdot \langle f^\otimes, \mathbf{1}_{n+p} \rangle + (0_n \oplus \mathbf{1}_n \oplus 0_p) \cdot \langle f^\otimes, \mathbf{1}_{n+p} \rangle \\
&= f \cdot \langle f^\otimes, 0_n \oplus \mathbf{1}_p \rangle + (\mathbf{1}_n \oplus 0_p),
\end{aligned}$$

for all $f : n \to n+p$. Suppose now that the generalized star fixed point identity holds. Then

$$\begin{aligned}
f^\dagger &= f^\otimes \cdot \langle 0_{n,p}, \mathbf{1}_p\rangle \\
&= (f \cdot \langle f^\otimes, 0_n \oplus \mathbf{1}_p\rangle + (\mathbf{1}_n \oplus 0_p)) \cdot \langle 0_{n,p}, \mathbf{1}_p\rangle \\
&= f \cdot \langle f^\otimes \cdot \langle 0_{n,p}, \mathbf{1}_p\rangle, (0_n \oplus \mathbf{1}_p) \cdot \langle 0_{n,p}, \mathbf{1}_p\rangle\rangle + (\mathbf{1}_n \oplus 0_p) \cdot \langle 0_{n,p}, \mathbf{1}_p\rangle \\
&= f \cdot \langle f^\dagger, \mathbf{1}_p\rangle + 0_{n,p} \\
&= f \cdot \langle f^\dagger, \mathbf{1}_p\rangle
\end{aligned}$$

for all $f : n \to n+p$.
(b) Let $f : n \to n+p$ and $g : p \to q$. Suppose that the parameter identity holds. Then

$$\begin{aligned}
f^\otimes \cdot (\mathbf{1}_n \oplus g) &= (f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p))^\dagger \cdot (\mathbf{1}_n \oplus g) \\
&= ((f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p)) \cdot (\mathbf{1}_{2n} \oplus g))^\dagger \\
&= (f \cdot (\mathbf{1}_n \oplus 0_n \oplus g) + (0_n \oplus \mathbf{1}_n \oplus 0_q))^\dagger \\
&= (f \cdot (\mathbf{1}_n \oplus g) \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_q) + (0_n \oplus \mathbf{1}_n \oplus 0_q))^\dagger \\
&= (f \cdot (\mathbf{1}_n \oplus g))^\otimes.
\end{aligned}$$

Suppose now that the generalized star parameter identity holds. Then

$$\begin{aligned}
f^\dagger \cdot g &= f^\otimes \cdot \langle 0_{n,p}, \mathbf{1}_p\rangle \cdot g \\
&= f^\otimes \cdot \langle 0_{n,q}, g\rangle \\
&= f^\otimes \cdot (\mathbf{1}_n \oplus g) \cdot \langle 0_{n,q}, \mathbf{1}_q\rangle \\
&= (f \cdot (\mathbf{1}_n \oplus g))^\otimes \cdot \langle 0_{n,q}, \mathbf{1}_q\rangle \\
&= (f \cdot (\mathbf{1}_n \oplus g))^\dagger.
\end{aligned}$$

(c) Suppose that the left zero identity holds. Then

$$\begin{aligned}
(0_n \oplus f)^\otimes &= ((0_n \oplus f) \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p))^\dagger \\
&= ((0_{2n} \oplus f) + (0_n \oplus \mathbf{1}_n \oplus 0_p))^\dagger \\
&= (0_n \oplus ((0_n \oplus f) + (\mathbf{1}_n \oplus 0_p)))^\dagger \\
&= (0_n \oplus f) + (\mathbf{1}_n \oplus 0_p),
\end{aligned}$$

for all $f : n \to p$. Suppose now that the generalized star left zero identity holds. Then

$$\begin{aligned}
(0_n \oplus f)^\dagger &= (0_n \oplus f)^\otimes \cdot \langle 0_{n,p}, \mathbf{1}_p\rangle \\
&= ((0_n \oplus f) + (\mathbf{1}_n \oplus 0_p)) \cdot \langle 0_{n,p}, \mathbf{1}_p\rangle \\
&= f + 0_{n,p} \\
&= f,
\end{aligned}$$

for all $f : n \to p$.

(d) Suppose that the right zero identity holds. Then

$$\begin{aligned}(f \oplus 0_q)^{\otimes} &= ((f \oplus 0_q) \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_{p+q}) + (0_n \oplus \mathbf{1}_n \oplus 0_{p+q}))^{\dagger} \\ &= (((f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p)) \oplus 0_q) + (0_n \oplus \mathbf{1}_n \oplus 0_{p+q}))^{\dagger} \\ &= ((f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p)) \oplus 0_q)^{\dagger} \\ &= (f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p))^{\dagger} \oplus 0_q \\ &= f^{\otimes} \oplus 0_q\end{aligned}$$

for all $f : n \to n + p$. The other direction also holds: assuming the generalized star right zero identity, we have

$$\begin{aligned}(f \oplus 0_q)^{\dagger} &= (f \oplus 0_q)^{\otimes} \cdot \langle 0_{n,p+q}, \mathbf{1}_{p+q} \rangle \\ &= f^{\otimes} \cdot (\mathbf{1}_n \oplus \mathbf{1}_p \oplus 0_q) \cdot \langle 0_{n,p+q}, \mathbf{1}_{p+q} \rangle \\ &= f^{\otimes} \cdot \langle 0_{n,p+q}, \mathbf{1}_p \oplus 0_q \rangle \\ &= (f^{\otimes} \cdot \langle 0_{n,p}, \mathbf{1}_p \rangle) \oplus 0_q \\ &= f^{\dagger} \oplus 0_q,\end{aligned}$$

for all $f : n \to n + p$.

(e) Let $f : n \to n + n + p$ and let $\pi = \langle 0_n \oplus \mathbf{1}_n, \mathbf{1}_n \oplus 0_n \rangle$. Suppose that the parameter and the double dagger identities hold.

$$\begin{aligned}&(f^{\otimes} \cdot (\pi \oplus \mathbf{1}_p))^{\otimes} \cdot \langle 0_{n,n+p}, \mathbf{1}_{n+p} \rangle = \\ &= ((f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_{n+p}) + (0_n \oplus \mathbf{1}_n \oplus 0_{n+p}))^{\dagger} \cdot (\pi \oplus \mathbf{1}_p))^{\dagger} \\ &= ((f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_{n+p}) + (0_n \oplus \mathbf{1}_n \oplus 0_{n+p})) \cdot (\mathbf{1}_n \oplus \pi \oplus \mathbf{1}_p))^{\dagger\dagger} \\ &= (f \cdot (\mathbf{1}_{2n} \oplus 0_n \oplus \mathbf{1}_p) + (0_{2n} \oplus \mathbf{1}_n \oplus 0_p))^{\dagger\dagger} \\ &= (f \cdot (\langle \mathbf{1}_n, \mathbf{1}_n \rangle \oplus \mathbf{1}_p) \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p))^{\dagger} \\ &= (f \cdot (\langle \mathbf{1}_n, \mathbf{1}_n \rangle \oplus \mathbf{1}_p))^{\otimes}.\end{aligned}$$

Suppose now that the parameter and generalized double star identities hold. Then:

$$\begin{aligned}(f \cdot (\langle \mathbf{1}_n, \mathbf{1}_n \rangle \oplus \mathbf{1}_p))^{\dagger} &= (f \cdot (\langle \mathbf{1}_n, \mathbf{1}_n \rangle \oplus \mathbf{1}_p))^{\otimes} \cdot \langle 0_{n,p}, \mathbf{1}_p \rangle \\ &= (f^{\otimes} \cdot (\pi \oplus \mathbf{1}_p))^{\otimes} \cdot \langle 0_{n,n+p} \oplus \mathbf{1}_{n+p} \rangle \cdot \langle 0_{n,p}, \mathbf{1}_p \rangle \\ &= (f^{\otimes} \cdot (\pi \oplus \mathbf{1}_p))^{\dagger} \cdot \langle 0_{n,p}, \mathbf{1}_p \rangle \\ &= (f^{\otimes} \cdot (\pi \oplus \mathbf{1}_p) \cdot (\mathbf{1}_n \oplus \langle 0_{n,p}, \mathbf{1}_p \rangle))^{\dagger} \\ &= (f^{\otimes} \cdot \langle 0_{n,n+p}, \mathbf{1}_{n+p} \rangle)^{\dagger} \\ &= f^{\dagger\dagger}.\end{aligned}$$

(f) We now show that the composition identity holds iff the generalized star composition identity holds. Suppose first that the composition identity holds. Then

$$\begin{aligned}&(f \cdot \langle g, 0_n \oplus \mathbf{1}_p \rangle)^{\otimes} = \\ &= (f \cdot \langle g, 0_n \oplus \mathbf{1}_p \rangle \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p))^{\dagger} \\ &= (f \cdot \langle g \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p), 0_{2n} \oplus \mathbf{1}_p \rangle + (0_n \oplus \mathbf{1}_n \oplus 0_p))^{\dagger} \\ &= (h \cdot \langle t, 0_n \oplus \mathbf{1}_{p'} \rangle)^{\dagger},\end{aligned}$$

where $h = f^\tau = f \cdot (\mathbf{1}_m \oplus 0_n \oplus \mathbf{1}_p) + (0_m \oplus \mathbf{1}_n \oplus 0_p)$, and $t = g \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p)$.

Then by the composition identity we have

$$\begin{aligned}
&(h \cdot \langle t, 0_n \oplus \mathbf{1}_{p'} \rangle)^\dagger = \\
&= h \cdot \langle (t \cdot \langle h, 0_m \oplus \mathbf{1}_{p'} \rangle)^\dagger, \mathbf{1}_{p'} \rangle \\
&= f^\tau \cdot \langle (g \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) \cdot \langle f^\tau, 0_m \oplus \mathbf{1}_{p'} \rangle)^\dagger, \mathbf{1}_{p'} \rangle \\
&= f^\tau \cdot \langle (g \cdot \langle f^\tau, 0_{m+n} \oplus \mathbf{1}_p \rangle)^\dagger, \mathbf{1}_{p'} \rangle \\
&= f^\tau \cdot \langle (g \cdot \langle f^\tau, 0_{m+n} \oplus \mathbf{1}_p \rangle)^\otimes, 0_m \oplus \mathbf{1}_{p'} \rangle \cdot \langle 0_{m,p'}, \mathbf{1}_{p'} \rangle .
\end{aligned}$$

Suppose now that the generalized star composition identity holds. Then

$$(f \cdot \langle g, 0_n \oplus \mathbf{1}_p \rangle)^\otimes \cdot \langle 0_{n,p}, \mathbf{1}_p \rangle = (f \cdot \langle g, 0_n \oplus \mathbf{1}_p \rangle)^\dagger ,$$

and also

$$\begin{aligned}
&f^\tau \cdot \langle (g \cdot \langle f^\tau, 0_{m+n} \oplus \mathbf{1}_p \rangle)^\dagger, \mathbf{1}_{p'} \rangle \cdot \langle 0_{n,p}, \mathbf{1}_p \rangle = \\
&= f^\tau \cdot \langle (g \cdot \langle f^\tau, 0_{m+n} \oplus \mathbf{1}_p \rangle \cdot (\mathbf{1}_m \oplus \langle 0_{n,p}, \mathbf{1}_p \rangle))^\dagger, 0_{n,p}, \mathbf{1}_p \rangle \\
&= f \cdot (\mathbf{1}_m \oplus 0_n \oplus \mathbf{1}_p) \cdot \langle (g \cdot \langle f^\tau, 0_{m+n} \oplus \mathbf{1}_p \rangle \cdot (\mathbf{1}_m \oplus \langle 0_{n,p}, \mathbf{1}_p \rangle))^\dagger, 0_{n,p}, \mathbf{1}_p \rangle \\
&= f \cdot \langle (g \cdot \langle f^\tau, 0_{m+n} \oplus \mathbf{1}_p \rangle \cdot (\mathbf{1}_m \oplus \langle 0_{n,p}, \mathbf{1}_p \rangle))^\dagger, \mathbf{1}_p \rangle \\
&= f \cdot \langle (g \cdot \langle f^\tau \cdot (\mathbf{1}_m \oplus \langle 0_{n,p}, \mathbf{1}_p \rangle), 0_m \oplus \mathbf{1}_p \rangle)^\dagger, \mathbf{1}_p \rangle \\
&= f \cdot \langle (g \cdot \langle f, 0_m \oplus \mathbf{1}_p \rangle)^\dagger, \mathbf{1}_p \rangle ,
\end{aligned}$$

since

$$\begin{aligned}
&f^\tau \cdot (\mathbf{1}_m \oplus \langle 0_{n,p}, \mathbf{1}_p \rangle) = \\
&= (f \cdot (\mathbf{1}_m \oplus 0_n \oplus \mathbf{1}_p) + (0_m \oplus \mathbf{1}_n \oplus 0_p)) \cdot (\mathbf{1}_m \oplus \langle 0_{n,p}, \mathbf{1}_p \rangle) \\
&= f + 0_{n,n+p} \\
&= f .
\end{aligned}$$

(g) Let $f : n \to n + p$ and assume that $\pi : n \to n$ is a base permutation with inverse π^{-1}. Suppose that the parameter and the permutation identities hold.

$$\begin{aligned}
&(\pi \cdot f \cdot (\pi^{-1} \oplus \mathbf{1}_p))^\otimes = \\
&= (\pi \cdot f \cdot (\pi^{-1} \oplus \mathbf{1}_p) \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p))^\dagger \\
&= (\pi \cdot f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) \cdot (\pi^{-1} \oplus \mathbf{1}_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p) \cdot (\pi^{-1} \oplus \mathbf{1}_n \oplus \mathbf{1}_p))^\dagger \\
&= ((\pi \cdot f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p)) \cdot (\pi^{-1} \oplus \mathbf{1}_n \oplus \mathbf{1}_p))^\dagger \\
&= ((\pi \cdot f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + \pi \cdot (0_n \oplus \pi^{-1} \oplus 0_p)) \cdot (\pi^{-1} \oplus \mathbf{1}_n \oplus \mathbf{1}_p))^\dagger \\
&= (\pi \cdot (f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \pi^{-1} \oplus 0_p)) \cdot (\pi^{-1} \oplus \mathbf{1}_{n+p}))^\dagger \\
&= \pi \cdot (f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \pi^{-1} \oplus 0_p))^\dagger \\
&= \pi \cdot ((f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p)) \cdot (\mathbf{1}_n \oplus \pi^{-1} \oplus \mathbf{1}_p))^\dagger \\
&= \pi \cdot (f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p))^\dagger \cdot (\pi^{-1} \oplus \mathbf{1}_p) \\
&= \pi \cdot f^\otimes \cdot (\pi^{-1} \oplus \mathbf{1}_p) .
\end{aligned}$$

Supposing that the generalized star permutation identity holds, we have

$$\begin{aligned}(\pi \cdot f \cdot (\pi^{-1} \oplus \mathbf{1}_p))^\dagger &= (\pi \cdot f \cdot (\pi^{-1} \oplus \mathbf{1}_p))^\otimes \cdot \langle 0_{n,p}, \mathbf{1}_p\rangle \\ &= \pi \cdot f^\otimes \cdot (\pi^{-1} \oplus \mathbf{1}_p) \cdot \langle 0_{n,p}, \mathbf{1}_p\rangle \\ &= \pi \cdot f^\dagger.\end{aligned}$$

(h) Assume that $f : n \to n+m+p$ and $g : m \to n+m+p$. Let π denote the base permutation $\langle 0_m \oplus \mathbf{1}_n, \mathbf{1}_m \oplus 0_n\rangle : n+m \to m+n$ with inverse $\pi^{-1} = \langle 0_n \oplus \mathbf{1}_m, \mathbf{1}_n \oplus 0_m\rangle : m+n \to n+m$. Define

$$k = g \cdot \langle f^\otimes \cdot (\pi \oplus \mathbf{1}_p), \mathbf{1}_m \oplus 0_n \oplus \mathbf{1}_p\rangle : m \to m+n+p.$$

We show that when the parameter identity holds, then the pairing identity holds iff the following generalized star pairing identity holds:

$$\langle f, g\rangle^\otimes = \langle f^\otimes \cdot \langle \mathbf{1}_n \oplus 0_{m+p}, k^\otimes \cdot (\pi^{-1} \oplus \mathbf{1}_p), 0_{n+m} \oplus \mathbf{1}_p\rangle, k^\otimes \cdot (\pi^{-1} \oplus \mathbf{1}_p)\rangle.$$

Assume first that the parameter identity and the pairing identity hold. Then

$$\begin{aligned}\langle f, g\rangle^\otimes &= (\langle f, g\rangle^\tau)^\dagger \\ &= (\langle f, g\rangle \cdot (\mathbf{1}_{n+m} \oplus 0_{n+m} \oplus \mathbf{1}_p) + (0_{n+m} \oplus \mathbf{1}_{n+m} \oplus 0_p))^\dagger \\ &= \langle f \cdot (\mathbf{1}_{n+m} \oplus 0_{n+m} \oplus \mathbf{1}_p) + (0_{n+m} \oplus \mathbf{1}_n \oplus 0_{m+p}), \\ &\quad\; g \cdot (\mathbf{1}_{n+m} \oplus 0_{n+m} \oplus \mathbf{1}_p) + (0_{n+m+n} \oplus \mathbf{1}_m \oplus 0_p)\rangle^\dagger \\ &= \langle \overline{f}, \overline{g}\rangle^\dagger.\end{aligned}$$

Thus, by the pairing identity,

$$\langle f, g\rangle^\otimes = \langle \overline{f}^\dagger \cdot \langle h^\dagger, \mathbf{1}_{n+m+p}\rangle, h^\dagger\rangle,$$

where

$$h = \overline{g} \cdot \langle \overline{f}^\dagger, \mathbf{1}_{m+n+m+p}\rangle.$$

Now

$$\begin{aligned}\overline{f}^\dagger &= (f \cdot (\mathbf{1}_{n+m} \oplus 0_{n+m} \oplus \mathbf{1}_p) + (0_{n+m} \oplus \mathbf{1}_n \oplus 0_{m+p}))^\dagger \\ &= ((f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_{m+p}) + (0_n \oplus \mathbf{1}_n \oplus 0_{m+p})) \cdot (\mathbf{1}_n \oplus \pi \oplus 0_m \oplus \mathbf{1}_p))^\dagger \\ &= (f^\tau \cdot (\mathbf{1}_n \oplus \pi \oplus 0_m \oplus \mathbf{1}_p))^\dagger \\ &= f^\otimes \cdot (\pi \oplus 0_m \oplus \mathbf{1}_p).\end{aligned}$$

Thus,

$$\begin{aligned}h &= \overline{g} \cdot \langle f^\otimes \cdot (\pi \oplus 0_m \oplus \mathbf{1}_p), \mathbf{1}_{m+n+m+p}\rangle \\ &= (g \cdot (\mathbf{1}_{n+m} \oplus 0_{n+m} \oplus \mathbf{1}_p) + (0_{n+m+n} \oplus \mathbf{1}_m \oplus 0_p)) \\ &\quad\; \cdot \langle f^\otimes \cdot (\pi \oplus 0_m \oplus \mathbf{1}_p), \mathbf{1}_{m+n+m+p}\rangle \\ &= g \cdot \langle f^\otimes \cdot (\pi \oplus 0_m \oplus \mathbf{1}_p), \mathbf{1}_m \oplus 0_{n+m} \oplus \mathbf{1}_p\rangle + (0_{m+n} \oplus \mathbf{1}_m \oplus 0_p) \\ &= (g \cdot \langle f^\otimes \cdot (\pi \oplus \mathbf{1}_p), \mathbf{1}_m \oplus 0_n \oplus \mathbf{1}_p\rangle \cdot (\mathbf{1}_m \oplus 0_m \oplus \mathbf{1}_{n+p}) + (0_m \oplus \mathbf{1}_m \oplus 0_{n+p})) \\ &\quad\; \cdot (\mathbf{1}_m \oplus \pi^{-1} \oplus \mathbf{1}_p) \\ &= (g \cdot \langle f^\otimes \cdot (\pi \oplus \mathbf{1}_p), \mathbf{1}_m \oplus 0_n \oplus \mathbf{1}_p\rangle)^\tau \cdot (\mathbf{1}_m \oplus \pi^{-1} \oplus \mathbf{1}_p) \\ &= k^\tau \cdot (\mathbf{1}_m \oplus \pi^{-1} \oplus \mathbf{1}_p),\end{aligned}$$

where

$$k = g \cdot \langle f^{\otimes} \cdot (\pi \oplus \mathbf{1}_p), \mathbf{1}_m \oplus 0_n \oplus \mathbf{1}_p \rangle.$$

Thus,

$$h^{\dagger} = k^{\otimes} \cdot (\pi^{-1} \oplus \mathbf{1}_p).$$

Using this,

$$\begin{aligned}\langle f, g\rangle^{\otimes} &= \langle f^{\otimes} \cdot (\pi \oplus 0_m \oplus \mathbf{1}_p) \cdot \langle k^{\otimes} \cdot (\pi^{-1} \oplus \mathbf{1}_p), \mathbf{1}_{n+m+p}\rangle, k^{\otimes} \cdot (\pi^{-1} \oplus \mathbf{1}_p)\rangle \\ &= \langle f^{\otimes} \cdot \langle \mathbf{1}_n \oplus 0_{m+p}, k^{\otimes} \cdot (\pi^{-1} \oplus \mathbf{1}_p), 0_{n+m} \oplus \mathbf{1}_p\rangle, k^{\otimes} \cdot (\pi^{-1} \oplus \mathbf{1}_p)\rangle.\end{aligned}$$

Suppose now that the parameter identity and the generalized star pairing identity hold. Let f, g be as above. We want to show that

$$\langle f, g\rangle^{\dagger} = \langle f^{\dagger} \cdot \langle h^{\dagger}, \mathbf{1}_p\rangle, \mathbf{1}_p\rangle,$$

where $h = g \cdot \langle f^{\dagger}, \mathbf{1}_{m+p}\rangle$. We have

$$\begin{aligned}\langle f, g\rangle^{\dagger} &= \langle f, g\rangle^{\otimes} \cdot \langle 0_{n+m,p}, \mathbf{1}_p\rangle \\ &= \langle f^{\otimes} \cdot \langle \mathbf{1}_n \oplus 0_{m+p}, k^{\otimes} \cdot (\pi^{-1} \oplus \mathbf{1}_p), 0_{n+m} \oplus \mathbf{1}_p\rangle, k^{\otimes} \cdot (\pi^{-1} \oplus \mathbf{1}_p)\rangle \\ &\quad \cdot \langle 0_{n+m,p}, \mathbf{1}_p\rangle,\end{aligned}$$

where k was defined above. First we show that

$$k^{\otimes} \cdot (\pi^{-1} \oplus \mathbf{1}_p) \cdot \langle 0_{n+m,p}, \mathbf{1}_p\rangle = h^{\dagger}.$$

Indeed,

$$\begin{aligned}&k^{\otimes} \cdot (\pi^{-1} \oplus \mathbf{1}_p) \cdot \langle 0_{n+m,p}, \mathbf{1}_p\rangle = \\ &= k^{\otimes} \cdot \langle 0_{m+n,p}, \mathbf{1}_p\rangle \\ &= k^{\otimes} \cdot \langle 0_{m,n+p}, \mathbf{1}_{n+p}\rangle \cdot \langle 0_{n,p}, \mathbf{1}_p\rangle \\ &= k^{\dagger} \cdot \langle 0_{n,p}, \mathbf{1}_p\rangle \\ &= (k \cdot (\mathbf{1}_m \oplus \langle 0_{n,p}, \mathbf{1}_p\rangle))^{\dagger} \\ &= (g \cdot \langle f^{\otimes} \cdot (\pi \oplus \mathbf{1}_p), \mathbf{1}_m \oplus 0_n \oplus \mathbf{1}_p\rangle \cdot (\mathbf{1}_m \oplus \langle 0_{n,p}, \mathbf{1}_p\rangle))^{\dagger} \\ &= (g \cdot \langle f^{\otimes} \cdot \langle 0_{n,m+p}, \mathbf{1}_{m+p}\rangle, \mathbf{1}_{m+p}\rangle)^{\dagger} \\ &= (g \cdot \langle f^{\dagger}, \mathbf{1}_{m+p}\rangle)^{\dagger} \\ &= h^{\dagger}.\end{aligned}$$

Using this,

$$\begin{aligned}f^{\otimes} \cdot \langle \mathbf{1}_n \oplus 0_{m+p}, k^{\otimes} \cdot (\pi^{-1} \oplus \mathbf{1}_p), 0_{n+m} \oplus \mathbf{1}_p\rangle \cdot \langle 0_{n+m,p}, \mathbf{1}_p\rangle = \\ = f^{\otimes} \cdot \langle 0_{n,p}, h^{\dagger}, \mathbf{1}_p\rangle \\ = f^{\otimes} \cdot \langle 0_{n,m+p}, \mathbf{1}_{m+p}\rangle \cdot \langle h^{\dagger}, \mathbf{1}_p\rangle \\ = f^{\dagger} \cdot \langle h^{\dagger}, \mathbf{1}_p\rangle.\end{aligned}$$

completing the proof.

Thus, when the generalized star fixed point identity holds, then for each $f : n \to n+p$, $f^{\otimes}$ solves the fixed point equation $\xi = f \cdot \langle \xi, 0_n \oplus \mathbf{1}_p \rangle + (\mathbf{1}_n \oplus 0_p)$ in the variable $\xi : n \to n+p$. When $p = 0$ this becomes $\xi = f \cdot \xi + \mathbf{1}_n$. Note also that if the pairing identity holds, then the dagger operation is completely determined by its restriction to the scalar morphisms $1 \to 1+p$, $p \geq 0$. Similarly, if the generalized star pairing identity holds, then the generalized star operation is completely determined by its restriction to the scalar morphisms $1 \to 1+p$. The parameter identity and the generalized star parameter identity are of special importance due to the following fact.

Proposition 3. *Suppose that T is a grove theory. If T is a dagger theory satisfying the parameter identity, then (3) holds. If T is a generalized star theory satisfying the generalized star parameter identity, then (4) holds.*

Proof. We only prove the first claim, since the proof of the second is similar. Assume that T is a dagger theory satisfying the parameter identity. Then,

$$\begin{aligned}(f^{\tau})^{\dagger} \cdot \langle 0_{n,p}, \mathbf{1}_p \rangle &= (f^{\tau} \cdot (\mathbf{1}_n \oplus \langle 0_{n,p}, \mathbf{1}_p \rangle))^{\dagger} \\ &= ((f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p)) \cdot (\mathbf{1}_n \oplus \langle 0_{n,p}, \mathbf{1}_p \rangle))^{\dagger} \\ &= (f + 0_{n,n+p})^{\dagger} \\ &= f^{\dagger},\end{aligned}$$

for all $f : n \to n+p$.

The above identities are not complete for the equational theory of the fixed point operation in theories of continuous functions over complete lattices, see Section 5. To achieve completeness, the semigroup identities and the group identities were introduced in [9,11] as simplifications of the *commutative identities* of [2].

Let $S = ([n], \circ)$ be a finite semigroup. For every theory T we define the following base morphisms $\rho_i^S : n \to n$, for $i \in [n]$: $\rho_i^S = \langle (i \circ 1)_n, \ldots, (i \circ n)_n \rangle$, where $\circ$ is the semigroup operation of S. Then for an arbitrary $f : n \to n+p$ the morphism f_S can be defined as follows: if $f = \langle f_1, \ldots, f_n \rangle$ then let $f_S = \langle f_1 \cdot (\rho_1^S \oplus \mathbf{1}_p), \ldots, f_n \cdot (\rho_n^S \oplus \mathbf{1}_p) \rangle$, and for every $g : 1 \to n+p$ let $g_S = (\tau_n \cdot g)_S = \langle g \cdot (\rho_1^S \oplus \mathbf{1}_p), \ldots, g \cdot (\rho_n^S \oplus \mathbf{1}_p) \rangle : n \to n+p$, where τ_n is the unique base morphism $n \to 1$.

Definition 3. *The* semigroup identity $C(S)$ *associated with a semigroup* $S = ([n], \circ)$ *is:*

$$g_S^{\dagger} = \tau_n \cdot (g \cdot (\tau_n \oplus \mathbf{1}_p))^{\dagger}, \text{ where } g : 1 \to n+p.$$

When S is a group, $C(S)$ is called a group identity.

Let us observe that an equivalent form of the semigroup identity $C(S)$ is:

$$\langle g \cdot (\rho_1^S \oplus \mathbf{1}_p), \ldots, g \cdot (\rho_n^S \oplus \mathbf{1}_p) \rangle^{\dagger} = \langle (g \cdot (\tau_n \oplus \mathbf{1}_p))^{\dagger}, \ldots, (g \cdot (\tau_n \oplus \mathbf{1}_p))^{\dagger} \rangle,$$

where obviously $\rho_i^S \cdot \tau_n = \tau_n$ for each $i \in [n]$.

Proposition 4. *Let T be a grove theory equipped with a dagger and a generalized star operation, which are related by (1) and (2). Suppose that the parameter identity holds in T. Then for a semigroup S the identity $C(S)$ holds iff the following identity $C_\otimes(S)$ holds:*

$$g_S^{\otimes} \cdot (\tau_n \oplus \mathbf{1}_p) = \tau_n \cdot (g \cdot (\tau_n \oplus \mathbf{1}_p))^{\otimes}, \tag{5}$$

where $g : 1 \to n + p$.

Proof. Since

$$g_S^{\otimes} \cdot (\tau_n \oplus \mathbf{1}_p) \cdot \langle 0_{1,p}, \mathbf{1}_p \rangle = g_S^{\otimes} \cdot \langle 0_{n,p}, \mathbf{1}_p \rangle = g_S^{\dagger}$$

and

$$\tau_n \cdot (g \cdot (\tau_n \oplus \mathbf{1}_p))^{\otimes} \cdot \langle 0_{1,p}, \mathbf{1}_p \rangle = \tau_n \cdot (g \cdot (\tau_n \oplus \mathbf{1}_p))^{\dagger},$$

if $C_\otimes(S)$ holds, then so does $C(S)$.

In the computations below, in order to save space, we will only indicate the generic ith component of a tuple. We will make use of the following equation:

$$\begin{aligned}
&\langle \ldots, g \cdot (\rho_i^S \oplus \mathbf{1}_p) \cdot (\mathbf{1}_n \oplus 0_1 \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_1 \oplus 0_p), \ldots \rangle = \\
&= \langle \ldots, g \cdot (\rho_i^S \oplus \mathbf{1}_p) \cdot (\mathbf{1}_n \oplus 0_1 \oplus \mathbf{1}_p), \ldots \rangle + (0_n \oplus \tau_n \oplus 0_p).
\end{aligned}$$

Indeed, since

$$\begin{aligned}
&i_n \cdot \langle \ldots, g \cdot (\rho_i^S \oplus \mathbf{1}_p) \cdot (\mathbf{1}_n \oplus 0_1 \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_1 \oplus 0_p), \ldots \rangle = \\
&= g \cdot (\rho_i^S \oplus \mathbf{1}_p) \cdot (\mathbf{1}_n \oplus 0_1 \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_1 \oplus 0_p) \\
&= i_n \cdot \langle \ldots, g \cdot (\rho_i^S \oplus \mathbf{1}_p) \cdot (\mathbf{1}_n \oplus 0_1 \oplus \mathbf{1}_p), \ldots \rangle + i_n \cdot (0_n \oplus \tau_n \oplus 0_p) \\
&= i_n \cdot (\langle \ldots, g \cdot (\rho_i^S \oplus \mathbf{1}_p) \cdot (\mathbf{1}_n \oplus 0_1 \oplus \mathbf{1}_p), \ldots \rangle + (0_n \oplus \tau_n \oplus 0_p)),
\end{aligned}$$

for all $i \in [n]$.

Suppose now that $C(S)$ holds, then

$$\begin{aligned}
&\tau_n \cdot (g \cdot (\tau_n \oplus \mathbf{1}_p))^{\otimes} = \\
&= \tau_n \cdot (g \cdot (\tau_n \oplus \mathbf{1}_p) \cdot (\mathbf{1}_1 \oplus 0_1 \oplus \mathbf{1}_p) + (0_1 \oplus \mathbf{1}_1 \oplus 0_p))^{\dagger} \\
&= \tau_n \cdot ((g \cdot (\mathbf{1}_n \oplus 0_1 \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_1 \oplus 0_p)) \cdot (\tau_n \oplus \mathbf{1}_1 \oplus \mathbf{1}_p))^{\dagger} \\
&= (g \cdot (\mathbf{1}_n \oplus 0_1 \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_1 \oplus 0_p))_S^{\dagger} \\
&= \langle \ldots, (g \cdot (\mathbf{1}_n \oplus 0_1 \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_1 \oplus 0_p)) \cdot (\rho_i^S \oplus \mathbf{1}_{1+p}), \ldots \rangle^{\dagger} \\
&= (\langle \ldots, g \cdot (\rho_i^S \oplus \mathbf{1}_p) \cdot (\mathbf{1}_n \oplus 0_1 \oplus \mathbf{1}_p), \ldots \rangle + (0_n \oplus \tau_n \oplus 0_p))^{\dagger} \\
&= (g_S^{\tau} \cdot (\mathbf{1}_n \oplus \tau_n \oplus \mathbf{1}_p))^{\dagger} \\
&= g_S^{\otimes} \cdot (\tau_n \oplus \mathbf{1}_p).
\end{aligned}$$

Note that we have used the parameter identity in the last equation.

A *Conway theory* [2] is a dagger theory which satisfies the left and right zero identities, the pairing identity and the permutation identity. There are several alternative axiomatizations: the parameter, double dagger and composition identities; and the *scalar versions* of the parameter, double dagger, composition and pairing identities are two other sets of axioms. Here, the scalar versions of the parameter and double dagger identities are obtained by taking $n = 1$ in the corresponding identity, while the scalar version of the composition identity is the composition identity restricted to $n = m = 1$. Finally, the scalar pairing identity is the pairing identity with $m = 1$ (or $n = 1$). Since the fixed point identity is a special case of the composition identity, it follows that except for the identities associated with finite (semi)groups, all identities involving the dagger operation defined above hold in all Conway theories. A morphism of Conway theories is a dagger theory morphism. A *Conway grove theory* is a Conway theory which is a grove theory. Morphisms of Conway grove theories are grove theory morphisms which are dagger theory morphisms.

Definition 4. *A* Conway star theory *is a generalized star theory satisfying the generalized star left zero, right zero, pairing and permutation identities. A morphism of Conway star theories is a generalized star theory morphism.*

Corollary 1. *A generalized star theory is a Conway star theory iff it satisfies the generalized star parameter, double star and star composition identities; or the scalar versions of the generalized star parameter, double star, star composition and star pairing identities.*

The scalar versions are defined in the same way as for the dagger identities. By Propositions 1, 2 and 3 we have:

Corollary 2. *The category of Conway grove theories is isomorphic to the category of Conway star theories.*

An *iteration theory* [2,11] is a Conway theory satisfying all group identities. A morphism of iteration theories is a Conway theory morphism. It is known that all semigroup identities hold in all iteration theories. An *iteration grove theory* is an iteration theory which is a grove theory. A morphism of iteration grove theories is a Conway grove theory morphism.

Definition 5. *An* iteration star theory *is a Conway star theory satisfying the identities $C_{\otimes}(G)$ for all finite groups G. A morphism of iteration star theories is a Conway star theory morphism.*

Corollary 3. *All identities $C_{\otimes}(S)$ hold in all iteration star theories, where S is any finite semigroup.*

Corollary 4. *The category of iteration grove theories is isomorphic to the category of iteration star theories.*

In Conway theories, the semigroup identities $C(S)$ are implied by a simple implication. Let T be a dagger theory and C a subset of the morphisms of T.

Following [2], we say that T satisfies the *functorial dagger implication* for C if for all $f : n \to n+p$, $g : m \to m+p$ in T and for all $h : n \to m$ in C,

$$f \cdot (h \oplus \mathbf{1}_p) = h \cdot g \quad \Rightarrow \quad f^\dagger = h \cdot g^\dagger.$$

It is known that any Conway theory satisfies the functorial implication for injective base morphisms, and that a Conway theory satisfies the functorial dagger implication for all base morphisms iff it satisfies the functorial dagger implication for the set of all base morphisms $n \to 1$, $n \geq 2$.

Definition 6. *Let T be a generalized star theory and C a subset of the set of morphisms of T. We define two versions of the* generalized functorial star implication *for C. We say that T satisfies the first version if for all $f : n \to n+p$, $g : m \to m+p$ in T and $h : n \to m$ in C,*

$$f^\tau \cdot (h \oplus \langle 0_{n,p}, \mathbf{1}_p\rangle) = h \cdot g \quad \Rightarrow \quad f^\otimes \cdot \langle 0_{n,p}, \mathbf{1}_p\rangle = h \cdot g^\otimes \cdot \langle 0_{m,p}, \mathbf{1}_p\rangle. \qquad (6)$$

Moreover, we say that T satisfies the second version if for all f, g, h as above,

$$f^\tau \cdot (h \oplus h \oplus \mathbf{1}_p) = h \cdot g^\tau \quad \Rightarrow \quad f^\otimes \cdot (h \oplus \mathbf{1}_p) = h \cdot g^\otimes. \qquad (7)$$

Proposition 5. *Let T be a grove theory which is both a dagger and a generalized star theory in which the dagger and generalized star operations are related by (1) and (2). Moreover, suppose that T satisfies the* parameter *identity.*

Then for an arbitrary set C of T-morphisms the functorial dagger implication holds for C iff the first version of the generalized functorial star implication holds. Also, the functorial dagger implication holds for some set C of base morphisms iff the second version of the generalized functorial star implication holds.

Proof. To prove the first claim, let C be an arbitrary set of morphisms. Then by $f^\tau \cdot (h \oplus \langle 0_{n,p}, \mathbf{1}_p\rangle) = f \cdot (h \oplus \mathbf{1}_p)$ and $f^\otimes \cdot \langle 0_{n,p}, \mathbf{1}_p\rangle = f^\dagger$, $h \cdot g^\otimes \cdot \langle 0_{m,p}, \mathbf{1}_p\rangle = h \cdot g^\dagger$, the functorial dagger implication holds for C iff (6) holds.

Now let C be a set of base morphisms and assume that (7) holds. Let $f : n \to n+p$, $g : m \to m+p$ and $h : n \to m$ with $h \in C$. Assume that $f \cdot (h \oplus \mathbf{1}_p) = h \cdot g$. Then, using Remark 1,

$$\begin{aligned}
f^\tau \cdot (h \oplus h \oplus \mathbf{1}_p) &= f \cdot (h \oplus 0_m \oplus \mathbf{1}_p) + (0_m \oplus h \oplus 0_p)\\
&= f \cdot (h \oplus \mathbf{1}_p) \cdot (\mathbf{1}_m \oplus 0_m \oplus \mathbf{1}_p) + (0_m \oplus h \oplus 0_p)\\
&= h \cdot g \cdot (\mathbf{1}_m \oplus 0_m \oplus \mathbf{1}_p) + (0_m \oplus h \oplus 0_p)\\
&= h \cdot g^\tau.
\end{aligned}$$

Thus, by (7),

$$f^\otimes \cdot (h \oplus \mathbf{1}_p) = h \cdot g^\otimes,$$

so that

$$\begin{aligned}
f^\dagger &= f^\otimes \cdot \langle 0_{m,p}, \mathbf{1}_p\rangle\\
&= f^\otimes \cdot (h \oplus \mathbf{1}_p) \cdot \langle 0_{m,p}, \mathbf{1}_p\rangle\\
&= h \cdot g^\otimes \cdot \langle 0_{m,p}, \mathbf{1}_p\rangle\\
&= h \cdot g^\dagger.
\end{aligned}$$

We have thus proved that the functorial dagger implication holds.

Suppose now that the functorial dagger implication holds for C. We show that (7) also holds for C. For this reason, let f, g and h be as above, and assume that $f^\tau \cdot (h \oplus h \oplus \mathbf{1}_p) = h \cdot g^\tau$. Let

$$\overline{f} = f^\tau \cdot (\mathbf{1}_n \oplus h \oplus \mathbf{1}_p) = f \cdot (\mathbf{1}_n \oplus 0_m \oplus \mathbf{1}_p) + (0_n \oplus h \oplus 0_p) : n \to n+m+p.$$

Then,

$$\begin{aligned} \overline{f} \cdot (h \oplus \mathbf{1}_{m+p}) &= f \cdot (h \oplus 0_m \oplus \mathbf{1}_p) + (0_m \oplus h \oplus 0_p) \\ &= f^\tau \cdot (h \oplus h \oplus \mathbf{1}_p) \\ &= h \cdot g^\tau . \end{aligned}$$

Thus, by the functorial dagger implication, also

$$\overline{f}^\dagger = h \cdot (g^\tau)^\dagger .$$

Using this fact and the parameter identity,

$$\begin{aligned} f^\otimes \cdot (h \oplus \mathbf{1}_p) &= (f^\tau)^\dagger \cdot (h \oplus \mathbf{1}_p) \\ &= (f^\tau \cdot (\mathbf{1}_n \oplus h \oplus \mathbf{1}_p))^\dagger \\ &= \overline{f}^\dagger \\ &= h \cdot (g^\tau)^\dagger \\ &= h \cdot g^\otimes . \end{aligned}$$

Corollary 5. *Let T be a Conway star theory. Then the generalized functorial star implications hold for the set of injective base morphisms. Moreover, the generalized functorial star implications hold for the set of all base morphisms iff they hold for the base morphisms $n \to 1$ for all $n \geq 2$.*

Any Conway star theory satisfying one of the two versions of the functorial star implication for base morphisms is an iteration star theory.

4 Ordered Iteration Grove Theories

An *ordered theory* is a theory T equipped with a partial order $\leq$ on each hom-set $T(n,p)$ which is preserved by the composition and tupling operations. A morphism of ordered theories is a theory morphism which preserves the partial order.

Proposition 6. *Suppose that T is both a grove theory and an ordered theory. Then the sum operation is monotone:*

$$f \leq f' \ \& \ g \leq g' \quad \Rightarrow \quad f+g \leq f'+g', \quad f,f',g,g' : n \to p.$$

Definition 7. *An* ordered grove theory *is a grove theory T which is an ordered theory such that for each p, $0_{1,p}$ is the least morphism $1 \to p$. A morphism of ordered grove theories is a grove theory morphism which is an ordered theory morphism.*

It then follows that for any pair n, p of nonnegative integers, $0_{n,p}$ is least in $T(n,p)$.

Example 4. Suppose that T is a grove theory such that $+ \cdot \langle \mathbf{1}_1, \mathbf{1}_1 \rangle = \mathbf{1}_1$. It then follows that the sum operation is idempotent: $f + f = f$ for all $f : n \to p$, and we call T an *idempotent grove theory.*

When T is idempotent, there is a unique partial order $\leq$ turning T into an ordered grove theory: We have $f \leq g$ for $f, g : n \to p$ iff $f + g = g$ iff there is some $h : n \to p$ with $f + h = g$. It follows that any grove theory morphism between idempotent grove theories preserves this order and is thus an ordered grove theory morphism.

Definition 8. *An* ordered dagger theory *is an ordered theory which is a dagger theory such that the dagger operation is monotone and for each n, p, $\perp_{n,p} = (\mathbf{1}_n \oplus 0_p)^\dagger$ is the least morphism $n \to p$. An* ordered iteration theory *is an ordered dagger theory which is an iteration theory. An* ordered generalized star theory *is an ordered grove theory which is a generalized star theory such that the generalized star operation is monotone. An* ordered iteration star theory *is an ordered generalized star theory which is an iteration star theory. Morphisms of these structures also preserve the order.*

Note that by Definitions 7 and 8, $\perp_{n,p} = 0_{n,p}$ holds for all $n, p \geq 0$ in an ordered iteration star theory.

The theories $\mathbf{Cont}_L$ defined in Example 2 satisfy the following *fixed point induction rule*, cf. [19,9]:

$$f \cdot \langle g, \mathbf{1}_p \rangle \leq g \quad \Rightarrow \quad f^\dagger \leq g$$

for all $f : n \to n + p$ and $g : n \to p$.

Proposition 7. *Suppose that T is an ordered grove theory which is both a dagger theory and a generalized star theory. Suppose that the dagger and generalized star operations are related by (1) and (2). If T satisfies the fixed point induction rule and the parameter identity, then T satisfies the following* generalized star fixed point induction rule*:*

$$f \cdot \langle g, 0_n \oplus \mathbf{1}_p \rangle + h \leq g \quad \Rightarrow \quad f^\otimes \cdot \langle h, 0_n \oplus \mathbf{1}_p \rangle \leq g,$$

for all $f, g, h : n \to n + p$. Moreover, if T satisfies the generalized star parameter identity and the generalized star fixed point induction rule then T satisfies the fixed point induction rule.

Proof. To prove the first claim, suppose that T satisfies the parameter identity and the fixed point induction rule. Assume that $f, g, h : n \to n + p$ with $f \cdot \langle g, 0_n \oplus \mathbf{1}_p \rangle + h \leq g$. Then

$$\begin{aligned}
&f^\tau \cdot (\mathbf{1}_n \oplus \langle h, 0_n \oplus \mathbf{1}_p \rangle) \cdot \langle g, \mathbf{1}_{n+p} \rangle = \\
&= f^\tau \cdot \langle g, h, 0_n \oplus \mathbf{1}_p \rangle \\
&= (f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_n)) \cdot \langle g, h, 0_n \oplus \mathbf{1}_p \rangle \\
&= f \cdot \langle g, 0_n \oplus \mathbf{1}_p \rangle + h \\
&\leq g.
\end{aligned}$$

Thus, by the fixed point induction rule and the parameter identity,

$$\begin{aligned} f^{\otimes} \cdot \langle h, 0_n \oplus \mathbf{1}_p \rangle &= (f^{\tau})^{\dagger} \cdot \langle h, 0_n \oplus \mathbf{1}_p \rangle \\ &= (f^{\tau} \cdot (\mathbf{1}_n \oplus \langle h, 0_n \oplus \mathbf{1}_p \rangle))^{\dagger} \\ &\leq g. \end{aligned}$$

Suppose now that the generalized star fixed point induction rule holds. Let $f : n \to n + p$ and $g : n \to p$ with $f \cdot \langle g, \mathbf{1}_p \rangle \leq g$. Then

$$f \cdot \langle 0_n \oplus g, 0_n \oplus \mathbf{1}_p \rangle + 0_{n,n+p} \leq 0_n \oplus g,$$

so that

$$f^{\otimes} \cdot \langle 0_{n,n+p}, 0_n \oplus \mathbf{1}_p \rangle \leq 0_n \oplus g.$$

Composing both sides with $\langle 0_{n,p}, \mathbf{1}_p \rangle$ on the right, this gives

$$f^{\dagger} = f^{\otimes} \cdot \langle 0_{n,p}, \mathbf{1}_p \rangle \leq g.$$

5 Applications

In this section, we present some applications of the results of the previous sections. Some other applications, not treated here because of space limitations, include formal tree series, theories of synchronization trees and theories of synchronization trees with respect to various behavioral equivalences.

Below, by a *dagger term* we will mean any term built in the usual way from symbols representing morphisms in dagger grove theories and the distinguished morphisms by composition, the cartesian operations, sum and dagger. *Star terms* are defined in the amalogous way. Note that each dagger or star term has a source n and a target p, and under each evaluation of the morphism variables, the term evaluates to a morphism $n \to p$ in any dagger theory or generalized star theory. An equation $t = t'$, or inequation $t \leq t'$ between dagger or star terms is a formal (in)equality between terms $t, t' : n \to p$. The validity or satisfaction of an (in)equation in a dagger grove theory or a generalized star theory is defined as usual.

Example 5. Let L be a complete lattice. We define a dagger and a generalized star operation on $\mathbf{Cont}_L$. Let $f : L^{n+p} \to L^n$ be a continuous function. By the Knaster-Tarski fixed point theorem, for each $z \in L^p$, the endofunction $L^n \to L^n$, $x \mapsto f(x, z)$ has a least (pre-)fixed point. We define $f^{\dagger}(z)$ as this least pre-fixed point. An easy argument shows that $f^{\dagger}$ is also continuous.

Note that when $f : L^{n+p} \to L^n$ is continuous, then so is the function $f^{\tau} : L^{n+n+p} \to L^n$, defined by $(x, y, z) \mapsto f(x, z) \vee y$. We define $f^{\otimes} := (f^{\tau})^{\dagger}$.

By definition, (1) holds. Since $f^{\tau}(x, \perp_n, z) = f(x, z)$, it follows that (2) also holds. Moreover, since equipped with the dagger operation, $\mathbf{Cont}_L$ is a iteration grove theory, cf. [2,12], it is also an iteration star theory, in fact an idempotent iteration grove theory and an idempotent iteration star theory.

Proposition 8. *Suppose that $f : L^{n+p} \to L^n$ in $\mathbf{Cont}_L$. Then for each $y \in L^n$ and $z \in L^p$, $f^{\otimes}(y, z)$ is the least pre-fixed point of the endofunction $f_z : L^n \to L^n$, $x \mapsto f(x, z)$ which is greater than or equal to y.*

Proof. Since $\mathbf{Cont}_L$ is an iteration star theory, the generalized star fixed point identity holds. Thus, for any y and z, $f^{\otimes}(y, z) = f(f^{\otimes}(y, z), z) \vee y$ so that $f(f^{\otimes}(y, z), z) \leq f^{\otimes}(y, z)$ and $y \leq f^{\otimes}(y, z)$. Suppose now that $x \in L^n$ satisfies $f(x, z) \leq x$ and $y \leq x$. Then $f^{\tau}(x, y, z) = f(x, z) \vee y \leq x$ and thus $f^{\otimes}(y, z) = (f^{\tau})^{\dagger}(y, z) \leq x$ by the definition of dagger.

The following result was proved in [12]. (For dagger terms without + see also [2].)

Theorem 1. *An (in)equation between dagger terms holds in all theories $\mathbf{Cont}_L$, where L is any complete lattice iff it holds in all iteration grove theories satisfying $+^{\dagger} = \mathbf{1}_1$.*

The last equation can also be written as $(1_2 + 2_2)^{\dagger} = \mathbf{1}_1$. As a corollary of this result we obtain:

Corollary 6. *An equation between star terms holds in all theories $\mathbf{Cont}_L$, where L is any complete lattice iff it holds in all iteration grove theories satisfying $\mathbf{1}_1{}^{\otimes} = \mathbf{1}_1$.*

The following result is a reformulation of a result of [12].

Theorem 2. *An equation between dagger terms holds in all theories $\mathbf{Cont}_L$ iff it holds in all idempotent grove theories which are dagger theories satisfying the (scalar versions) of the fixed point equation, the parameter identity, and the fixed point induction rule.*

Corollary 7. *An equation between star terms holds in all theories $\mathbf{Cont}_L$ iff it holds in all idempotent generalized star theories satisfying the (scalar versions of the) generalized star fixed point equation, the generalized star parameter identity, and the generalized star fixed point induction rule.*

The last two results also hold for the broader class of monotone functions.

The free theories in the corresponding equational class can be described as theories of regular synchronization trees modulo simulation equivalence. See [13].

Example 6. Suppose that S is a *continuous monoid*, i.e., a commutative monoid $S = (S, +, 0)$ equipped with a partial order $\leq$ such that $(S, \leq)$ is a cpo with least element 0, so that the supremum of each nonempty directed set exists, and the sum operation preserves such suprema (and is thus monotone).

Let $\mathbf{Cont}_S$ denote the theory of continuous functions over S. It is an iteration grove theory in the same way as the theory $\mathbf{Cont}_L$, where L is a complete lattice. But unless the monoid S is idempotent, $\mathbf{Cont}_S$ is not necessarily idempotent. Note that unlike in [4] or [15], we do not require here any linearity conditions for the functions themselves.

The following results were proved in [13].

Theorem 3. *An (in)equation between dagger terms holds in all theories* $\mathbf{Cont}_S$*, where* S *is any continuous monoid iff it holds in all ordered iteration grove theories satisfying* $(1_3 + 2_3 + 3_3)^{\dagger\dagger} = (1_2 + 2_2)^{\dagger}$*.*

Theorem 4. *An (in)equation between dagger terms holds in all theories* $\mathbf{Cont}_S$ *iff it holds in all ordered dagger grove theories satisfying the (scalar versions) of the fixed point equation, the parameter identity and the fixed point induction rule, together with the equation* $(1_3 + 2_3 + 3_3)^{\dagger\dagger} = (1_2 + 2_2)^{\dagger}$*.*

Corollary 8. *An (in)equation between star terms holds in all theories* $\mathbf{Cont}_S$*, where* S *is any continuous monoid iff it holds in all ordered iteration star theories satisfying* $\mathbf{1}_1{}^{\otimes\otimes} = \mathbf{1}_1{}^{\otimes}$*, or when it holds in all ordered generalized star theories satisfying the star forms the (scalar versions) of fixed point equation, the parameter identity, the fixed point induction rule, together with the equation* $\mathbf{1}_1{}^{\otimes\otimes} = \mathbf{1}_1{}^{\otimes}$*.*

The free theories in the corresponding equational class can be described as theories of regular synchronization trees modulo injective simulation equivalence. See [13].

In our last example, we consider tree languages. Recall Example 3.

Example 7. Let Σ be a ranked set and consider the idempotent (and thus ordered) theory $\mathbf{Lang}_\Sigma$. Each hom-set is a complete lattice and the theory and sum operations are continuous. It follows that for each morphism $L = (L_1, \ldots, L_n) : n \to n + p$ the fixed point equation

$$X = L \cdot \langle X, \mathbf{1}_p \rangle$$

has a least solution in the variable $X = (X_1, \ldots, X_n)$ over $\mathbf{Lang}_\Sigma(n, p)$, denoted $L^\dagger$. Given the dagger operation, we can also define a generalized star operation in the usual way. For each L as above, $L^\otimes$ provides a least solution to the equation

$$Y = L \cdot \langle Y, 0_n \oplus \mathbf{1}_p \rangle + (\mathbf{1}_n \oplus 0_p)$$

in the variable $Y = (Y_1, \ldots, Y_n)$ ranging over the set of morphisms $n \to p$. (When $p = 0$, the above equation reads as $Y = L \cdot Y + \mathbf{1}_n$.) Equipped with the dagger operation, $\mathbf{Lang}_\Sigma$ is an iteration grove theory, and equipped with the generalized star operation, $\mathbf{Lang}_\Sigma$ is an iteration star theory, containing the theory of *regular* trees cf. [14,10] as a sub iteration grove (resp, star) theory denoted $\mathbf{Reg}_\Sigma$. Note also that on morphisms $L : 1 \to 1 + p$ (i.e., tree languages $L \subseteq T_\Sigma(X_{1+p})$), $L^\otimes$ is the familiar x_1*-iterate* of L, cf. [14].

We give a reformulation of the main result of [10]. The original theorem used the language of μ-terms. We may identify each letter in Σ with a singleton set containing the corresponding atomic tree.

Theorem 5. $\mathbf{Reg}_\Sigma$*, equipped with dagger, has the following universal property. Given any idempotent grove theory* T *which is a dagger theory satisfying the* *(scalar versions of) the parameter identity, the fixed point identity and the fixed point induction rule, and given any (rank preserving) function $h : \Sigma \to T$ such that (8) and (9) hold, there is a unique dagger grove theory morphism $h^\sharp : \mathbf{Reg}_\Sigma \to T$ extending h.*

$$\sigma \cdot \langle f_1 + g_1, \ldots, f_n + g_n \rangle = \sum_{h_i \in \{f_i, g_i\}} \sigma \cdot \langle h_1, \ldots, h_n \rangle \tag{8}$$

$$\sigma \cdot \langle f_1, \ldots, 0_p, \ldots, f_n \rangle = 0_p \tag{9}$$

where $\sigma \in \Sigma_n$, $f_i, g_i : 1 \to p$, $i \in [n]$.

It is now a routine matter to formulate the same result in terms of generalized star grove theories.

References

1. Bekić, H.: Definable operations in genaral algebras and the theory of automata and flowcharts. LNCS, vol. 177, pp. 30–55. Springer, Heidelberg (1984)
2. Bloom, S.L., Ésik, Z.: Iteration Theories. Springer, Heidelberg (1993)
3. Bloom, S.L., Ésik, Z.: An extension theorem with an application to formal tree series. J. of Automata, Languages and Combinatorics 8, 145–185 (2003)
4. Bozapalidis, S.: Equational elements in additive algebras. Theory of Comput. Sys. 32, 1–33 (1999)
5. de Bakker, J.W., Scott, D.: A theory of programs. IBM Seminar, Vienna (1969)
6. Engelfriet, J., Schmidt, E.M.: IO and OI. I. and II. J. Comput. System Sci. 15, 328–353 (1977); 16, 67–99 (1978)
7. Elgot, C.C.: Matricial theories. J. Algebra 42, 391–421 (1976)
8. Elgot, C.C.: Monadic computation and iterative algebraic theories. In: Logic Colloquium 1973, Bristol. Studies in Logic and the Foundations of Mathematics, vol. 80, pp. 175–230. North-Holland, Amsterdam (1975)
9. Ésik, Z.: Completeness of Park induction. Theoret. Comput. Sci. 177, 217–283 (1997)
10. Ésik, Z.: Axiomatizing the equational theory of regular tree languages (extended abstract). In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 455–465. Springer, Heidelberg (1998)
11. Ésik, Z.: Group axioms for iteration. Information and Computation 148, 131–180 (1999)
12. Ésik, Z.: Axiomatizing the least fixed point operation and binary supremum. In: Clote, P.G., Schwichtenberg, H. (eds.) CSL 2000. LNCS, vol. 1862, pp. 302–316. Springer, Heidelberg (2000)
13. Ésik, Z.: Continuous additive algebras and injective simulations of synchronization trees. J. Logic and Comutation 12, 271–300 (2002)
14. Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó, Budapest (1984)
15. Kuich, W.: Linear systems of equations and automata on distributive multioperator monoids. In: Contributions to general algebra 12, Vienna, 1999, Heyn, Klagenfurt, pp. 247–256 (2000)
16. Lawvere, F.W.: Functorial semantics of algebraic theories. Proc. Nat. Acad. Sci. U.S.A. 50, 869–872 (1963)

17. Niwinski, D.: Equational μ-calculus. In: Skowron, A. (ed.) SCT 1984. LNCS, vol. 208, pp. 169–176. Springer, Heidelberg (1985)
18. Niwinski, D.: On fixed-point clones (extended abstract). In: Kott, L. (ed.) ICALP 1986. LNCS, vol. 226, pp. 464–473. Springer, Heidelberg (1986)
19. Park, D.M.R.: Fixed point induction and proofs of program properties. In: Michie, D., Meltzer, B. (eds.) Machine Intelligence 5, pp. 59–78. Edinburgh Univ. Press (1970)
20. Plotkin, G.: Domains. Univeristy of Edinburgh (1983)
21. Simpson, A., Plotkin, G.: Complete axioms for categorical fixed-point operators. In: 15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, CA, pp. 30–41. IEEE Comput. Soc. Press, Los Alamitos (2000)
22. Wright, J.B., Thatcher, J.W., Wagner, E.G., Goguen, J.A.: Rational algebraic theories and fixed-point solutions. In: 17th Annual Symposium on Foundations of Computer Science, Houston, Tex, pp. 147–158. IEEE Comput. Soc, Long Beach (1976)

Combinatorics of Finite Words and Suffix Automata*

Gabriele Fici

Dipartimento di Informatica e Applicazioni
Università di Salerno, Italy
fici@dia.unisa.it

Abstract. The suffix automaton of a finite word is the minimal deterministic automaton accepting the language of its suffixes. The states of the suffix automaton are the classes of an equivalence relation defined on the set of factors. We explore the relationship between the combinatorial properties of a finite word and the structural properties of its suffix automaton. We give formulas for expressing the total number of states and the total number of edges of the suffix automaton in terms of special factors of the word.

1 Introduction

The suffix automaton, also called DAWG (Directed Acyclic Word Graph), is a data structure largely used in many text processing problems, like pattern matching and data compression. It is an indexing structure on a text which solves the following problem: given a text (string) X, preprocess it in order to search all the occurrences of a pattern (substring) P in linear time with respect to the size of the pattern. From an algorithmic point of view, the efficiency of the suffix automaton is a consequence of the fact that it can be constructed (on-line) in linear time and space with respect to the size of the input text X [1,2].

We introduce the suffix automaton from an algebraic point of view. The states of the suffix automaton of w are in fact the classes of an equivalence relation defined on the set of factors of w by means of their occurrences in w. The terminal states of the suffix automaton are then the classes containing a suffix of w. The classes of this equivalence have several well known remarkable properties. Indeed, any two classes are either in an inclusion relation or disjoint. Moreover, the total number of classes, i.e. the number of states of the suffix automaton, is smaller than twice the length of w.

Our investigation deals with this question: what are the relations between the structure of the suffix automaton and the combinatorics of the word w?

We show that each class contains as longest element either a prefix of w or a left special factor which is not a prefix. A left special factor is a factor u such that au and bu are both factors of w, for a and b distinct letters.

* Partially supported by the **MIUR** Project *"Aspetti matematici e applicazioni emergenti degli automi e dei linguaggi formali"* (2007).

As a consequence, we can characterize the words having suffix automaton of minimal size as the words such that every left special factor is a prefix.

Then we focus on binary words. In this case we can express the total number of classes in terms of two parameters, H_w and P_w. The first is the length of the shortest prefix of w that has no repetitions in w, while the second is the length of the longest prefix of w which is left special. We then derive a new combinatorial characterization of standard sturmian words in terms of these two parameters.

Finally, we give a formula that expresses the total number of edges of the suffix automaton of a binary word w in terms of its special factors.

The paper is organized as follows. In Section 2 we fix the notation and recall some definitions and basic facts about words. In Section 3 we introduce the suffix automaton and we recall some of its properties. In Section 4 we deal with the size of the suffix automaton and the number of its terminal states. In Section 5 we focus on binary words and we derive the size of the suffix automaton in terms of two combinatorial parameters, H_w and P_w. Finally, we give a formula for the computation of the number of edges of the suffix automaton of a binary word.

2 Notation and Background

An *alphabet*, denoted by A, is a finite set of symbols. The size of A is denoted by $|A|$. A *word* over A is a sequence of symbols from A. The *length* of a word w is denoted by $|w|$. The set of all words over A is denoted by A^*. The empty word has length zero and is denoted by ε. The set of all words over A having length $n \geq 0$ is denoted by A^n. A *language* over A is a subset of A^*. For a finite language L we denote by $|L|$ the number of its elements.

Let $w = a_1a_2 \dots a_n$, $n > 0$, be a nonempty word over the alphabet A. Any i such that $1 \leq i \leq n$ is called a *position* of w, and the letter $a_i \in A$ is called *the letter in position i*.

The *reversal* of the word $w = a_1a_2 \dots a_n$ is the word $\widetilde{w} = a_na_{n-1} \dots a_1$.

A *prefix* of w is any word v such that $v = \varepsilon$ or v is of the form $v = a_1a_2 \dots a_i$, with $1 \leq i \leq n$. A *suffix* of w is any word v such that $v = \varepsilon$ or v is of the form $v = a_ia_{i+1} \dots a_n$, with $1 \leq i \leq n$. A *factor* (or *substring*) of w is a prefix of a suffix of w (or, equivalently, a suffix of a prefix of w). Therefore, a factor of w is any word v such that $v = \varepsilon$ or v is of the form $v = a_ia_{i+1} \dots a_j$, with $1 \leq i \leq j \leq n$.

We denote by $\mathit{Pref}(w)$, $\mathit{Suff}(w)$, $\mathit{Fact}(w)$ respectively the set of prefixes, suffixes, factors of the word w.

The *factor complexity* of a word w is defined as $p_n(w) = |\mathit{Fact}(w) \cap A^n|$, for every $n \geq 0$. The *maximal factor complexity* of w is defined as $p(w) = \max_{n \geq 0}\{p_n(w)\}$, which represents the maximum number of distinct factors of w having the same length. Note that $p_1(w)$ is the number of distinct letters occurring in w. A *binary word* is a word w such that $p_1(w) = 2$.

A factor u of w is said *left special* if there exist $a, b \in A$, $a \neq b$, such that $au, bu \in \mathit{Fact}(w)$. A factor u of w is said *right special* if there exist $a, b \in A$,

$a \neq b$, such that $ua, ub \in \mathit{Fact}(w)$. A factor u of w is said *bispecial* if it is both left special and right special. We denote by $LS(w)$ (resp. $RS(w)$, $BS(w)$) the set of left special (resp. right special, bispecial) factors of the word w. We note $S^l_n(w)$ (resp. $S^r_n(w)$) the number of left (resp. right) special factors of w which have length n. We note $S^l(w)$ (resp. $S^r(w)$) the total number of left (resp. right) special factors of w, i.e. $S^l(w) = \sum_{n \geq 0} S^l_n(w) = |LS(w)|$ (resp. $S^r(w) = \sum_{n \geq 0} S^r_n(w) = |RS(w)|$). Since a word w has exactly one factor of length zero (the empty word ε), one has $S^l_0(w) = S^r_0(w) = 1$ if and only if $p_1(w) > 1$.

More about combinatorics on words can be found in [6].

3 The Suffix Automaton

Let $w = a_1a_2 \ldots a_n$, $n > 0$, be a nonempty word over the alphabet A. For any $v \in \mathit{Fact}(w)$ we can define the *set of ending positions* of v in w. It is the set $\mathit{Endset}_w(v)$ of the positions of w in which an occurrence of v ends. We assume that $\mathit{Endset}_w(\varepsilon) = \{0, 1, \ldots, n\}$.

Example 1. Let $w = aabaab$. Then one has $\mathit{Endset}_w(ba) = \{4\}$, $\mathit{Endset}_w(aab) = \mathit{Endset}_w(ab) = \{3, 6\}$.

In the next proposition we recall some properties of the sets of ending positions (see [3]):

Proposition 1. *[3] Let $u, v \in \mathit{Fact}(w)$. Then one of the three following conditions holds:*

1. $\mathit{Endset}_w(v) \subseteq \mathit{Endset}_w(u)$
2. $\mathit{Endset}_w(u) \subseteq \mathit{Endset}_w(v)$
3. $\mathit{Endset}_w(v) \cap \mathit{Endset}_w(u) = \emptyset$

Moreover, if $u \in \mathit{Suff}(v)$ then $\mathit{Endset}_w(v) \subseteq \mathit{Endset}_w(u)$. If $\mathit{Endset}_w(v) = \mathit{Endset}_w(u)$ then $v \in \mathit{Suff}(u)$ or $u \in \mathit{Suff}(v)$.

On the set $\mathit{Fact}(w)$ we can thus define the following equivalence:

$$u \equiv_w v \quad \Leftrightarrow \quad \mathit{Endset}_w(u) = \mathit{Endset}_w(v).$$

The set $\mathit{Fact}(w)$ is then partitioned into a finite number of classes with respect to this equivalence. These classes are called *right-equivalence classes.*

We note $[u]_w$ (or simply $[u]$, if the context does not make it ambiguous) the right-equivalence class of u in w. So $[u]_w = \{v \in \mathit{Fact}(w) : \mathit{Endset}_w(v) = \mathit{Endset}_w(u)\}$.

In the following proposition we gather some useful facts about the right-equivalence classes, that we will use in next sections.

Proposition 2. *Let $[u]$ be a right-equivalence class of factors of the word w. Then:*

1. *Two distinct elements in $[u]$ cannot have the same length. If v is the longest element in $[u]$, then any other element in $[u]$ is a proper suffix of v.*
2. *The class $[u]$ contains at most one prefix of w; this prefix is the longest element in $[u]$ and we call $[u]$ a* prefix class.
3. *If $v \in [u]$ is a suffix of w, then all the elements in $[u]$ are suffixes of w. In this case we call $[u]$ a* suffix class.

We now recall the definition and the basic properties of the suffix automaton (for more details see, for instance, [3]).

Definition 1 ([1,2]). *The* suffix automaton *(or* Direct Acyclic Word Graph*) of a word $w \in A^*$ is the minimal deterministic automaton accepting the language Suff(w). It is denoted by $\mathcal{A}(w)$.*

The states of $\mathcal{A}(w)$ are in fact the right-equivalence classes of factors of the word w. For each state q of the suffix automaton, the elements of the class $[u]_q$ associated to q are the labeled paths starting at the initial state and ending in q. Hence one can associate to each state q of $\mathcal{A}(w)$ the set of ending positions of factors in $[u]_q$.

There is an edge from the class q to the class q' labeled by the letter $a \in A$ if q' is the class of ua for any u in the class q.

An example of suffix automaton is displayed in Figure 1.

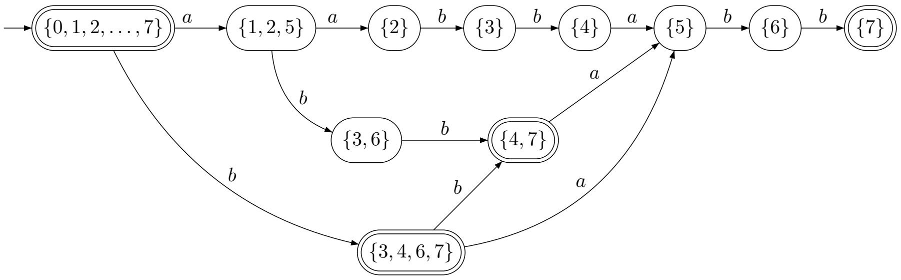


Fig. 1. The suffix automaton of the word $w = aabbabb$. Terminal states are double circled.

The *size* of $\mathcal{A}(w)$, denoted by $|Q_w|$, is the number of its states. Therefore, $|Q_w|$ is the number of right-equivalence classes of factors of w.

The bounds on $|Q_w|$ are well known. The following proposition can be found in [3].

Proposition 3. *[3] Let w be a word over A. If $|w| = 0$ then $|Q_w| = 1$; if $|w| = 1$ then $|Q_w| = 2$. If $|w| \geq 2$ then $1 + |w| \leq |Q_w| \leq 2|w| - 1$, and the upper bound is reached when w has the form $ab^{|w|-1}$, for a and b distinct letters.*

The set of edges of the suffix automaton of the word w is denoted by $\mathcal{E}_w$. In [3] we can find the following bounds.

Proposition 4. *[3] Let w be a word over A. If $|w| = 0$ then $|\mathcal{E}_w| = 0$; if $|w| = 1$ then $|\mathcal{E}_w| = 1$; if $|w| = 2$ then $2 \leq |\mathcal{E}_w| \leq 3$. If $|w| \geq 3$ then $|w| \leq |\mathcal{E}_w| \leq 3|w| - 4$, and the upper bound is reached when w has the form $ab^{|w|-2}c$, for a, b and c pairwise distinct letters.*

4 The Size of the Suffix Automaton

In this section we give some formulas for the computation of the number of states of the suffix automaton of a word w.

Definition 2. *Let w be a word. We denote by $D(w)$ the set of factors u of w such that u is not a prefix of w and u is left special.*

Proposition 5. *Let w be a word. Any $u \in D(w)$ is the longest element in its class $[u]$. In particular, then, the elements of $D(w)$ are each in a distinct class.*

Proof. By contradiction, suppose that there exists a factor v of w such that $v \in [u]$ and $|v| > |u|$. By Proposition 2, u is a proper suffix of v. Let us write $v = zau$, with $z \in A^*$, $a \in A$. Since u and v are in the same class, this implies that every occurrence of u in w is an occurrence of zau, and so u appears in w always preceded by the letter a, against the hypothesis that u is left special.

Since the longest element of a class is unique (by Proposition 2), each $u \in D(w)$ is in a distinct class. $\square$

Proposition 6. *Let w be a word over the alphabet A such that $|w| > 2$. Then the suffix automaton of w has size:*

$$|Q_w| = 1 + |w| + |D(w)|.$$

Proof. From Proposition 2 we know that each right-equivalence class contains at most one prefix of w, so the prefixes of w are each in a distinct class. Since a word w has exactly $|w| + 1$ prefixes the suffix automaton of w has $|w| + 1$ prefix classes. It rests to prove that the number of classes which are not prefix classes is $|D(w)|$.

Let $[u]$ be a class which is not a prefix class, and let u be its (unique) longest element. Thus, by Proposition 2, u is not a prefix of w (and in particular this implies that $|u| > 0$). So there exists a letter $a \in A$ such that au is factor of w. From Proposition 1 we have $Endset_w(au) \subseteq Endset(u)$. Since we supposed that u is the longest element in its class, au cannot belong to $[u]$, and so $Endset(au) \subset Endset(u)$. Hence there exists a position i such that $i \in Endset_w(u)$ but $i \notin Endset_w(au)$. Since u is not a prefix of w this implies that there exists a letter $b \in A$, $b \neq a$, such that $bu \in Fact(w)$, and so u is left special. Thus $u \in D(w)$.

The statement then follows from Proposition 5. $\square$

An interesting consequence of Proposition 6 is the following.

Corollary 1. *Let w be a word over the alphabet A such that $|w| > 2$. Then the suffix automaton of w has minimal number of states $|Q_w| = |w| + 1$ if and only if every left special factor of w is a prefix of w.*

Let us say that a (finite or infinite) word w over A has property LSP if every left special factor of w is a prefix of w.

Sciortino and Zamboni showed in [11] that finite words over a binary alphabet having property LSP are exactly the prefixes of standard sturmian words. Recall that a right infinite binary word w is a *sturmian word* if, for every $n \geq 0$, w has exactly $n+1$ distinct factors of length n. A *standard sturmian word* is a sturmian word having property LSP.

To the best of our knowledge property LSP does not characterize known sets of finite words in the case of larger alphabets.

A right infinite word is an *episturmian word* if it has at most one left special factor (or equivalently right special factor) for each length and the set of its factors is closed under reversal. A *standard episturmian word* is an episturmian word having property LSP.

More generally, if in the definition of episturmian word we substitute the reversal operator with any involutory antimorphism ϑ (i.e. a map $\vartheta : A^* \mapsto A^*$ such that $\vartheta(uv) = \vartheta(v)\vartheta(u)$ and $\vartheta \circ \vartheta = id$), we obtain a *$\vartheta$-episturmian word*. Once again, a *standard ϑ-episturmian word* is a ϑ-episturmian word having property LSP (see [8]).

The word $w = abcaaba$ has property LSP but has two right special factors of the same length (a and b). This implies that w cannot be a factor of a ϑ-episturmian word for any involutory antimorphism ϑ [7]; in particular, it cannot be a factor of an episturmian word.

5 Binary Words

In this section we show that for binary words the number of states of the suffix automaton can be expressed in terms of two combinatorial parameters related to the structure of the word.

We introduce the two parameters H_w and P_w.

Definition 3. *Let w be a word over A.*
We note H_w the minimal length of a prefix of w which occurs only once in w.
We note P_w the maximal length of a prefix of w which is left special.

We now show a property of binary words that underlines the relationship between H_w and the total number $S^l(w)$ of left special factors of w.

The next proposition shows that there is a close relation between the number of left special factors and the factor complexity of w.

Lemma 1. *Let w be a binary word such that $|w| > 2$. Then $S^l_n(w) = p_{n+1}(w) - p_n(w)$ if $0 \leq n < H_w$ and $S^l_n(w) = p_{n+1}(w) - p_n(w) + 1$ if $H_w \leq n \leq |w| - 1$.*

Proof. Let $0 \leq n < H_w$. Among the $p_n(w)$ factors of w of length n there are $S^l_n(w)$ factors that can be extended to the left with two letters, and $p_n(w) - S^l_n(w)$ factors that can be extended to the left with only one letter. If $H_w \leq n \leq |w|-1$, there is one factor (the prefix of w of length n) that cannot be extended to the left by any letter, since it appears in w only as a prefix.

Thus, the number of factors of w having length $n+1$, that is $p_{n+1}(w)$, is $2S^l_n(w)+p_n(w)-S^l_n(w)$ when $1 \leq n < H_w$, and it is $2S^l_n(w)+p_n(w)-S^l_n(w)-1$ when $H_w \leq n \leq |w|-1$. □

The following lemma gives us a simple formula for the computation of the total number of left special factors of a binary word.

Lemma 2. *Let w be a binary word such that $|w| > 2$. Then the total number of left special factors of w is $S^l(w) = |w| - H_w$.*

Proof. In view of Lemma 1 we have:

$$\begin{aligned}
S^l(w) &= \sum_{i=0}^{|w|-1} S^l_i(w) \\
&= \sum_{i=0}^{H_w-1} S^l_i(w) + \sum_{i=H_w}^{|w|-1} S^l_i(w) \\
&= \sum_{i=0}^{H_w-1} (p_{i+1}(w) - p_i(w)) + \sum_{i=H_w}^{|w|-1} (p_{i+1}(w) - p_i(w) + 1) \\
&= \sum_{i=0}^{|w|-1} (p_{i+1}(w) - p_i(w)) + (|w| - 1 - H_w + 1) \\
&= p_{|w|}(w) - p_0(w) + |w| - H_w \\
&= |w| - H_w
\end{aligned}$$

□

Analogous results hold for right special factors. If we denote by K_w the minimal length of a suffix of w which occurs only once in w, we get the following result:

Lemma 3. *Let w be a binary word such that $|w| > 2$. Then $S^r_n(w) = p_{n+1}(w) - p_n(w)$ if $0 \leq n < K_w$ and $S^r_n(w) = p_{n+1}(w) - p_n(w) + 1$ if $K_w \leq n \leq |w| - 1$.*

The total number of right special factors of w is $S^r(w) = |w| - K_w$.

Proof. The proof is very similar to that of Proposition 1 and Lemma 2. In fact, one reaches the result by using a symmetric argument in which "right" is replaced by "left", and K_w by H_w. □

The previous technical results (Lemmas 1,2 and 3) are rather easy observations on the factor complexity of finite binary words. For a deep study on the combinatorics of finite words over alphabets of arbitrary size see [4].

For binary words we can then express the number of states of the suffix automaton, $|Q_w|$, in terms of H_w and P_w.

Theorem 1. *Let w be a binary word. Then the number of states of the suffix automaton of w is*

$$|Q_w| = 2|w| - H_w - P_w$$

Proof. Let first $|w| = 2$. Since w is a binary word then either $w = ab$ or $w = ba$ for a and b distinct letters. In both cases we have $H_w = 1$ and $P_w = 0$. Moreover, in both cases $|Q_w| = 3$, so the claim holds.

Let now $|w| > 2$. From Proposition 6 we have $|Q_w| = 1 + |w| + |D(w)|$, where $D(w)$ is the set of left special factors of w that are not prefixes of w.

The total number of left special factors of w is given by the formula $S^l(w) = |w| - H_w$ by Lemma 2. In order to obtain the number of left special factors which are not prefixes, i.e. $|D(w)|$, we have to subtract the number of left special prefixes of w from $S^l(w)$.

If u is the longest prefix of w which is left special, then all the other left special prefixes of w are the prefixes of u. Since, by definition, $|u| = P_w$, we have that the number of left special factors of w which are not prefixes is $|D(w)| = S^l(w) - (|u| + 1) = |w| - H_w - (P_w + 1)$. Hence, the total number of distinct right-equivalence classes of the suffix automaton of w is given by:

$$\begin{aligned} |Q_w| &= 1 + |w| + |D(w)| \\ &= 1 + |w| + |w| - H_w - (P_w + 1) \\ &= 2|w| - H_w - P_w \end{aligned}$$

□

The previous result does not hold for words over an arbitrary alphabet. As an example, consider the word $w = abbccb$. The set of left special factors of w which are not prefixes of w is $D(w) = \{b, c\}$. Hence, by Proposition 6, one has $|Q_w| = 9$. Nevertheless, $H_w = 2$ and $P_w = 0$.

In fact, for words over alphabets larger than two, one has $S^l(w) \leq |w| - H_w$ (see [4]), and so Lemma 2 does not hold in general.

Another formula, involving left special factors, can be derived from the previous theorem and Lemma 2.

Corollary 2. *Let w be a binary word. Then the number of states of the suffix automaton of w is*

$$|Q_w| = |w| + S^l(w) - P_w$$

As a consequence of Theorem 1, we get another characterization of the prefixes of standard sturmian words.

Proposition 7. *Let w be a binary word. Then w is a prefix of a standard sturmian word if and only if $|w| = H_w + P_w + 1$.*

Remark. The previous proposition does not apply to words of the form $w = a^n$, $n \geq 0$. Indeed, such a word belongs to the set of prefixes of standard sturmian words, but $H_w + P_w + 1 = |w| + 1$.

We now deal with the number of edges of the suffix automaton of binary words. The following proposition gives a formula for the computation of the number of edges $|\mathcal{E}_w|$ of the suffix automaton of a binary word.

Proposition 8. *Let w be a binary word. Let $G(w) = (Pref(w) \cap RS(w)) \cup BS(w)$. Then:*

$$|\mathcal{E}_w| = |Q_w| + |G(w)| - 1$$

Proof. Let q be a state of the suffix automaton. If q is the state corresponding to the class of w itself then the outgoing degree of q is 0. Else the outgoing degree of q is either 1 or 2. If it is 2 we call the class corresponding to q a right special class. It is worth noting that all the factors in a right special class are right special factors.

Hence the total number of edges of the suffix automaton of w is $|\mathcal{E}_w| = |Q_w| - 1 + |G'(w)|$, where $G'(w)$ is the set of right special classes. So we have proved the claim once we prove that the sets $G(w)$ and $G'(w)$ are in bijection.

The set of right special classes $G'(w)$ is the union of the set of right special classes which are prefix classes and the set of ones which are not prefix classes.

We know that the longest element of a prefix class is unique and it is a prefix of w.

By Proposition 5 a class which is not a prefix class contains as longest element a left special factor which is not a prefix. So a right special class which is not a prefix class contains as longest element a bispecial factor of w.

Moreover, two different bispecial factors cannot share the same class, since two different left special factors cannot do it (by Proposition 5).

Thus, each right special class contains as longest element an element of $G(w)$ and the elements of $G(w)$ are each in a different class. □

6 Conclusions and Open Problems

This work is an attempt to investigate the combinatorics of a finite word by looking at the structure of its suffix automaton and vice versa.

The characterization of the set of prefixes of standard sturmian words in terms of their suffix automaton given by Sciortino and Zamboni ([11]) does not seem to easy generalize to larger alphabets. A more general question is to characterize the set of words having property LSP, both in the finite and in the infinite case.

Our formulas on the number of states and edges of the suffix automaton can also be used in the study of the average size of the suffix automaton, at least for binary words (this subject is treated for example in [10]). Indeed, both the parameters H_w and P_w are smaller than or equal to the repetition index of w (recall that, for a finite word w, the *repetition index* $r(w)$ is the length of the longest factor of w that has at least two occurrences in w). And it is known that for a word w randomly generated by a memoryless source with identical symbol probabilities, $r(w)$ is logarithmic in the length of w [5,9].

Another direction of research may consist in considering other data structures in place of the suffix automaton (e.g. factor oracles, suffix tries, suffix arrays, etc.). For example, a characterization of the class of words having factor automaton with minimal number of states is still lacking.

References

1. Blumer, A., Blumer, J., Haussler, D., Ehrenfeucht, A., Chen, M.T., Seiferas, J.I.: The smallest automaton recognizing the subwords of a text. Theor. Comput. Sci. 40, 31–55 (1985)
2. Crochemore, M.: Transducers and repetitions. Theor. Comput. Sci. 45(1), 63–86 (1986)
3. Crochemore, M., Hancart, C.: Automata for matching patterns. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Linear Modeling: Background and Application, vol. 2, ch. 9, pp. 399–462. Springer, Berlin (1997)
4. de Luca, A.: On the combinatorics of finite words. Theor. Comput. Sci. 218, 13–39 (1999)
5. Fici, G., Mignosi, F., Restivo, A., Sciortino, M.: Word assembly through minimal forbidden words. Theor. Comput. Sci. 359(1), 214–230 (2006)
6. Lothaire, M.: Algebraic Combinatorics on Words. Encyclopedia of Mathematics and its Applications. Cambridge Univ. Press, New York (2002)
7. De Luca, A.: Private communication (2009)
8. De Luca, A., Bucci, M., de Luca, A., Zamboni, L.Q.: On different generalizations of episturmian words. Theor. Comput. Sci. 393, 23–36 (2008)
9. Mignosi, F., Restivo, A., Sciortino, M.: Forbidden Factors and Fragment Assembly. RAIRO Theoret. Inform. Appl. 35(6), 565–577 (2001)
10. Raffinot, M.: Asymptotic estimation of the average number of terminal states in dawgs. Discrete Appl. Math. 92(2-3), 193–203 (1999)
11. Sciortino, M., Zamboni, L.Q.: Suffix automata and standard sturmian words. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 382–398. Springer, Heidelberg (2007)

Polynomial Operators on Classes of Regular Languages

Ondřej Klíma and Libor Polák[*]

Department of Mathematics and Statistics, Masaryk University
Kotlářská 2, 611 37 Brno, Czech Republic

Abstract. We assign to each positive variety $\mathcal{V}$ and each natural number k the class of all (positive) Boolean combinations of the restricted polynomials, i.e. the languages of the form $L_0a_1L_1a_2\dots a_\ell L_\ell$, where $\ell \le k$, $a_1,\dots,a_\ell$ are letters and $L_0,\dots,L_\ell$ are languages from the variety $\mathcal{V}$. For this polynomial operator we give a certain algebraic counterpart which works with identities satisfied by syntactic (ordered) monoids of languages considered. We also characterize the property that a variety of languages is generated by a finite number of languages. We apply our constructions to particular examples of varieties of languages which are crucial for a certain famous open problem concerning concatenation hierarchies.

Keywords: positive varieties of languages, polynomial operators.

2000 Classification: 68Q45 Formal languages and automata.

1 Introduction

The polynomial operator assigns to each positive variety of languages $\mathcal{V}$ the class of all (positive) Boolean combinations of the languages of the form

$$L_0a_1L_1a_2\dots a_\ell L_\ell\ , \qquad (*)$$

where A is an alphabet, $a_1,\dots,a_\ell \in A$, $L_0,\dots,L_\ell \in \mathcal{V}(A)$ (i.e. they are over A). Such operators on classes of languages lead to several concatenation hierarchies. Well-known cases are the Straubing-Thérien and the group hierarchies. Concatenation hierarchies has been intensively studied by many authors – see Section 8 of the Pin's Chapter [8]. The main open problem concerning concatenation hierarchies, which is in fact one of the most interesting open problem in the theory of regular languages, is the membership problem for the level 2 in the Straubing-Thérien hierarchy, i.e. the decision problem whether a given regular language can be written as a Boolean combination of polynomials over languages

[*] Both authors were supported by the Ministry of Education of the Czech Republic under the project MSM 0021622409 and by the Grants no. 201/06/0936 and 201/09/1313 of the Grant Agency of the Czech Republic.

from level 1 in that hierarchy. It is known that a language is of this type if and only if it is a Boolean combination of polynomials with languages $L_i = B_i^*$ where each $B_i \subseteq A$ $(i = 0, \dots, \ell)$. So this instance of polynomial operator is the most important case to study.

In the restricted case we fix a natural number k and we allow only $\ell \leq k$ in $(*)$. This operator was considered mainly in the case that $\mathcal{V}$ is the trivial variety by Simon in [10], in a series of papers by Blanchet-Sadri, see for instance [4], and in a recent paper by the authors [6].

The basic question both for general and restricted polynomial operator is to translate the construction on languages to the corresponding pseudovarieties of (ordered) monoids. A crucial tool is the Schützenberger product of (ordered) monoids (see Pin [9]). Other important questions for varieties resulting by the polynomial operator concern the existence of finite basis of (pseudo)identities for the corresponding pseudovarieties of (ordered) monoids and the possibility to generate such pseudovariety by a single monoid (see Volkov [11]).

In the present paper we continue our research from [6]. We concentrate here on identity problems for corresponding pseudovarieties and on the question whether they are generated by a single (ordered) monoid. In our basic examples the class $\mathcal{V}(A)$ equals to $\{\emptyset, A^*\}$ or to finite unions of $B^*,\ B \subseteq A$ or to finite unions of $\overline{B},\ B \subseteq A$ where $\overline{B}$ is the set of all words over A containing exactly the letters from B.

In the next section we recall the necessary background and we introduce there four examples which we will follow thorough the whole paper. We show in Section 3 that the locally finite positive varieties of languages (i.e. such that each $\mathcal{V}(A)$ is finite) correspond to the so-called finite characteristics which are certain relations on $\{x_1, x_2, \dots\}^*$. Section 4 contains the main result which effectively translates the polynomial operation on languages to an operator on finite characteristics. The last section studies the varieties of languages which are generated by a finite number of languages. In fact, this is equivalent to the property that corresponding pseudovariety of (ordered) monoids is generated by a single monoid. We transfer this property to finite characteristics. We conclude here to by investigating this "finiteness condition" on our basic examples.

2 Preliminaries

For a relation ρ on a set S we define its *dual* relation $\rho^{\mathsf{d}} = \{\, (v, u) \in S \times S \mid (u, v) \in \rho \,\}$. A *quasiorder* ρ on a set S is a reflexive and transitive relation. Let $\widehat{\rho} = \rho \cap \rho^{\mathsf{d}}$ be the corresponding equivalence relation.

An *ordered monoid* is a structure $(M, \cdot, \leq)$ where $(M, \cdot)$ is a monoid and $\leq$ is a *compatible* order on $(M, \cdot)$, i.e. $a \leq b$ implies both $a \cdot c \leq b \cdot c,\ c \cdot a \leq c \cdot b$, for all $a, b, c \in M$. *Morphisms* of ordered monoids are isotone monoid morphisms.

Let $(M, \cdot, \leq)$ be an ordered monoid and let $\preceq$ be a compatible quasiorder on $(M, \cdot)$ satisfying $\leq\ \subseteq\ \preceq$. Then the relation $\leq_{\preceq}$ defined by

$$a\widehat{\preceq}\ \leq_{\preceq}\ b\widehat{\preceq} \quad \text{if and only if} \quad a \preceq b,\ \text{for all } a, b \in M$$

is a compatible order on $(M/\widehat{\preceq}, \cdot)$ and the mapping $a \mapsto a\widehat{\preceq}$ is a morphism of $(M, \cdot, \leq)$ onto $(M/\widehat{\preceq}, \cdot, \leq_{\preceq})$.

Let Y^* be the set of all words over an alphabet Y including the empty one, denoted by λ. For a word $u \in Y^*$, let

$$\mathsf{cont}(u) = \{\, y \in Y \mid u = u'yu'' \text{ for some } u', u'' \in Y^* \,\} .$$

For a set $Z \subseteq Y$, let $\overline{Z} = \{\, u \in Y^* \mid \mathsf{cont}(u) = Z \,\}$. Let $|u|_y$ be the number of occurrences of a letter $y \in Y$ in $u \in Y^*$.

An *ideal* I of an ordered set $(M, \leq)$ is a subset of M satisfying $b \leq a \in I$ implies $b \in I$, for all $a, b \in M$. For $a \in M$, we write $(a] = \{\, b \in M \mid b \leq a \,\}$. A language L over an alphabet A is *recognized* by a finite ordered monoid $(M, \cdot, \leq)$ if there exist a morphism $\phi : A^* \to M$ and an ideal I of $(M, \leq)$ such that $L = \phi^{-1}(I)$.

We recall now some basic facts about Eilenberg-type theorems. The Boolean case was invented by Eilenberg [5] and the positive case was introduced by Pin [7].

A *Boolean variety of languages* $\mathcal{V}$ associates to every finite alphabet A a class $\mathcal{V}(A)$ of regular languages over A in such a way that

- $\mathcal{V}(A)$ is closed under finite unions, finite intersections and complements (in particular $\emptyset, A^* \in \mathcal{V}(A)$),
- $\mathcal{V}(A)$ is closed under derivatives, i.e. $L \in \mathcal{V}(A),\ u, v \in A^*$ implies $u^{-1}Lv^{-1} = \{\, w \in A^* \mid uwv \in L \,\} \in \mathcal{V}(A)$,
- $\mathcal{V}$ is closed under inverse morphisms, i.e. $f : B^* \to A^*,\ L \in \mathcal{V}(A)$ implies $f^{-1}(L) = \{\, v \in B^* \mid f(v) \in L \,\} \in \mathcal{V}(B)$.

To get the notion of a *positive* variety of languages, we use in the first item only intersections and unions (not complements). In fact in this paper we consider mainly positive varieties and the Boolean ones are treated as special cases.

The meaning of $\mathcal{V} \subseteq \mathcal{W}$ is that $\mathcal{V}(A) \subseteq \mathcal{W}(A)$, for each finite alphabet A. Similarly, $\bigcup_{i \in I} \mathcal{V}_i$ means that $(\bigcup_{i \in I} \mathcal{V}_i)(A) = \bigcup_{i \in I} \mathcal{V}_i(A)$, for each finite A and arbitrary set I.

A *pseudovariety* of finite monoids is a class of finite monoids closed under taking submonoids, morphic images and products of finite families. Similarly for ordered monoids (see [8]). When defining a *variety* of (ordered) monoids we use arbitrary products.

For a regular language $L \subseteq A^*$, we define the relations $\sim_L$ and $\preceq_L$ on A^* as follows: for $u, v \in A^*$ we have

$$u \sim_L v \text{ if and only if } (\ \forall\, p, q \in A^*\ )\ (\ puq \in L \iff pvq \in L\ ),$$

$$u \preceq_L v \text{ if and only if } (\ \forall\, p, q \in A^*\ )\ (\ pvq \in L \implies puq \in L\ ).$$

The relation $\sim_L$ is the *syntactic congruence* of L on A^*. It is of finite index (i.e. there are only finitely many classes) and the quotient structure $\mathsf{M}(L) = A^*/\sim_L$ is called the *syntactic monoid* of L.

The relation $\preceq_L$ is the *syntactic quasiorder* of L and we have $\widehat{\preceq_L} = \sim_L$. Hence $\preceq_L$ induces an order on $\mathsf{M}(L) = A^*/\sim_L$, namely: $u\sim_L \leq v\sim_L$ if and only if $u \preceq_L v$. Then we speak about the *syntactic ordered monoid* of L and we denote the structure by $\mathsf{O}(L)$.

Result 1 (Eilenberg[5], Pin[7].) *Boolean varieties (positive varieties) of languages correspond to pseudovarieties of finite monoids (ordered monoids). The correspondence, written $\mathcal{V} \longleftrightarrow \mathbf{V}$ ($\mathcal{P} \longleftrightarrow \mathbf{P}$), is given by the following relationship: for $L \subseteq A^*$ we have*

$$L \in \mathcal{V}(A) \textit{ if and only if } \mathsf{M}(L) \in \mathbf{V} \quad (L \in \mathcal{P}(A) \textit{ if and only if } \mathsf{O}(L) \in \mathbf{P}) .$$

The pseudovarieties of ordered monoids can be characterized by pseudoidentities (see e.g. [1]). The pseudovarieties we consider here are *equational* – they are given by identities. For the set $X = \{x_1, x_2, \dots\}$, an *identity* is a pair $u = v$ ($u \leq v$) of words over X, i.e. $u, v \in X^*$. An identity $u = v$ ($u \leq v$, respectively) is *satisfied* in a monoid M (ordered monoid $(M, \leq)$) if for each morphism $\phi : X^* \to M$ we have $\phi(u) = \phi(v)$ ($\phi(u) \leq \phi(v)$). In such a case we write $M \models u = v$ ($M \models u \leq v$), and for a set of identities Π, we define $\mathsf{Mod}\,\Pi = \{ M \mid (\forall \pi \in \Pi)\ M \models \pi \}$. For a class $\mathbf{M}$ of ordered monoids, the meaning of $\mathbf{M} \models \Pi$ is that, for each $M \in \mathbf{M}$, we have $M \models \Pi$. Let $\mathsf{Id}\,\mathbf{V}$ be the set of all identities which are satisfied in a variety of ordered monoids $\mathbf{V}$. Let $\mathsf{Fin}\,\mathbf{V}$ denote the class of all finite members of a class $\mathbf{V}$.

For a fixed A and $L \subseteq A^*$, let $L^{\mathsf{c}} = A^* \setminus L$ be the complement of L. For a class $\mathcal{V}$ of languages, we define $\mathcal{V}^{\mathsf{c}}$ by $\mathcal{V}^{\mathsf{c}}(A) = \{ L^{\mathsf{c}} \mid L \in \mathcal{V}(A) \}$. The following is obvious.

Lemma 1. *For a positive variety $\mathcal{V}$ the following holds.*

(i) $\mathcal{V}^{\mathsf{c}}$ is a positive variety.

(ii) Let $\mathcal{V} \vee \mathcal{V}^{\mathsf{c}}$ be the smallest positive variety containing both $\mathcal{V}$ and $\mathcal{V}^{\mathsf{c}}$. Then $(\mathcal{V} \vee \mathcal{V}^{\mathsf{c}})(A)$ consists of all positive Boolean combinations of the languages from $\mathcal{V}(A) \cup \mathcal{V}^{\mathsf{c}}(A)$.

(iii) The class $\mathcal{V} \vee \mathcal{V}^{\mathsf{c}}$ is a Boolean variety.

Next we define the positive varieties of languages $\mathcal{T}$, $\mathcal{S}^+$, $\mathcal{S}$, $\mathcal{A}_m$. We will return to them several times in our paper again.

Example 1. 1. Let $\mathcal{T}(A) = \{\emptyset, A^*\}$ for each finite set A.

2. Let $\mathcal{S}^+(A)$ be the set of all finite unions of the languages of the form B^*, where $B \subseteq A$, for each finite set A.

3. Let $\mathcal{S}(A)$ be the set of all finite unions of the languages of the form $\overline{B}$, where $B \subseteq A$, for each finite set A.

4. Let m be a fixed natural number and let $\mathcal{A}_m(A)$ be the set of all Boolean combinations of the languages of the form $L(a, r) = \{ u \in A^* \mid |u|_a \equiv r \pmod{m} \}$, where $a \in A$ and $0 \leq r < m$, for each finite set A.

Notice that the classes $\mathcal{T}$, $\mathcal{S}$, $\mathcal{A}_m$ are Boolean varieties. Moreover, for the corresponding pseudovarieties of (ordered) monoids consist of finite members of the following varieties:

$$\mathbf{T} = \mathsf{Mod}(\, x = y\,),\ \mathbf{S}^+ = \mathsf{Mod}(\, x^2 = x,\, xy = yx, 1 \leq x\,),$$

$$\mathbf{S} = \mathsf{Mod}(\, x^2 = x,\, xy = yx\,),\ \mathbf{A}_m = \mathsf{Mod}(\, xy = yx,\, x^m = 1\,)\ .$$

The names for the (ordered) monoids of the varieties $\mathbf{T}$, $\mathbf{S}^+$, $\mathbf{S}$, $\mathbf{A}_m$ are *trivial monoids*, *semilattices with the smallest element 1*, *semilattices* and *Abelian groups of index m*, respectively – see Pin [8].

3 Locally Finite Varieties of Languages

In this paper we concentrate on positive varieties of languages which correspond to locally finite pseudovarieties of ordered monoids. Each such pseudovariety is formed by the finite members of locally finite variety of ordered monoids (i.e. finitely generated ordered monoids are finite), and consequently such a variety of languages can be described by a fully invariant compatible quasiorder on the monoid X^* which has locally finite index; more precisely:

Definition 1. *A relation γ on X^* is* **a finite characteristic** *if it satisfies the following conditions:*
(i) γ is a quasiorder on X^*;
(ii) γ is compatible with the multiplication, i.e. for each $u, v, w \in X^*$ we have

$$u \;\gamma\; v \quad \textit{implies} \quad uw \;\gamma\; vw, \quad wu \;\gamma\; wv \;;$$

(iii) γ is fully invariant, i.e. for each morphism $\varphi : X^* \to X^*$ and each $u, v \in X^*$ we have

$$u \;\gamma\; v \quad \textit{implies} \quad \varphi(u) \;\gamma\; \varphi(v) \;;$$

(iv) for each finite subset Y of the set X, the set Y^* intersects only finitely many classes of $X^*/\,\widehat{\gamma}$.

For each finite alphabet A, we define the *natural adaptation* γ_A of a finite characteristic γ in the following way. For $u, v \in A^*$, we have

$$u \;\gamma_A\; v \quad \text{if and only if} \quad (\; \forall\, \varphi : A^* \to X^* \;)\; \varphi(u) \;\gamma\; \varphi(v)\;. \tag{$\dagger$}$$

It follows from the property (iii) in Definition 1 that in (†) we can use just one morphism given by a fixed injective mapping $\phi : A \to X$. In particular, if A is a finite subset of X then γ_A is a restriction of γ on A^*. The condition (iv) from Definition 1 means that γ_A (more precisely $\widehat{\gamma_A}$) has a finite index (i.e. the quotient set $A^*/\,\widehat{\gamma_A}$ is finite).

A relation γ on X^* satisfying the conditions (i) – (iii) is called a *fully invariant compatible quasiorder*. It determines a variety $\mathbf{V}_\gamma$ of ordered monoids; namely γ can be considered as a set of identities and $\mathbf{V}_\gamma = \mathsf{Mod}\,\gamma$. Basics of universal algebra, see [3] and [2], give that Id and Mod are mutually inverse bijections between varieties of ordered monoids and fully invariant compatible quasiorders on X^*. Moreover, for each $Y \subseteq X$, the ordered monoid Y^*/γ_Y is a free ordered monoid in $\mathbf{V}_\gamma$ over Y. The condition (iv) says that the finitely generated free ordered monoids in $\mathbf{V}_\gamma$ are finite. In this case the variety $\mathbf{V}_\gamma$ is *locally finite*, which means that all finitely generated ordered monoids are finite.

The pseudovariety $\mathsf{Fin}\,\mathbf{V}_\gamma$ of all finite members from $\mathbf{V}_\gamma$ corresponds to the positive variety $\mathcal{V}_\gamma$ of languages by

$$L \in \mathcal{V}_\gamma(A) \text{ if and only if } \mathsf{O}(L) \in \mathsf{Fin}\,\mathbf{V}_\gamma, \text{ for all finite } A .$$

We say that γ is a *finite characteristic of a class of languages* $\mathcal{V}$ if γ is a finite characteristic and for every finite alphabet A we have

$$L \in \mathcal{V}(A) \quad \text{if and only if} \quad \gamma_A \subseteq \preceq_L \ .$$

The following lemma explains the universal algebra point of view.

Lemma 2. *Let $\mathcal{V}$ be a class of languages and γ be a finite characteristic of $\mathcal{V}$. Then*

(i) $\mathcal{V}$ equals to the positive variety of languages $\mathcal{V}_\gamma = \mathsf{Fin}\,\mathsf{Mod}\,\gamma$;

(ii) if $\mathbf{V}$ is the pseudovariety of finite ordered monoids corresponding to $\mathcal{V}$ then $\gamma = \mathsf{Id}\,\mathbf{V}$;

(iii) γ^{d} is a finite characteristic of the positive variety $\mathcal{V}^{\mathsf{c}}$;

(iv) $\widehat{\gamma}$ is a finite characteristic of the Boolean variety $\mathcal{V} \vee \mathcal{V}^{\mathsf{c}}$.

Proof. "(i)" Let A be a finite alphabet. We have to show that $L \in \mathcal{V}(A)$ is equivalent to $L \in \mathcal{V}_\gamma(A)$. The statement on the left hand side is equivalent to $\gamma_A \subseteq \preceq_L$ which is equivalent to the fact that $\mathsf{O}(L)$ is a morphic image of A^*/γ_A. The last is equivalent to $\mathsf{O}(L) \in \mathsf{Fin}\,V_\gamma$, which means $L \in \mathcal{V}_\gamma(A)$.

"(ii)" Notice that $\mathbf{V}_\gamma$ is generated by its finitely generated free ordered monoids which are in $\mathsf{Fin}\,\mathbf{V}_\gamma$.

"(iii)" The statement follows from the fact that $\preceq_{L^{\mathsf{c}}} = (\preceq_L)^{\mathsf{d}}$.

"(iv)" It follows from Lemma 1. □

We present the finite characteristics for our four basic examples.

Example 2. (A continuation of Example 1.)

1. $\mathsf{Id}\,\mathbf{T} = X^* \times X^*$.
2. $\mathsf{Id}\,\mathbf{S}^+ = \{\, (u,v) \in X^* \times X^* \mid \mathsf{cont}(u) \subseteq \mathsf{cont}(v) \,\}$.
3. $\mathsf{Id}\,\mathbf{S} = \{\, (u,v) \in X^* \times X^* \mid \mathsf{cont}(u) = \mathsf{cont}(v) \,\}$.
4. $\mathsf{Id}\,\mathbf{A}_m = \{\, (u,v) \in X^* \times X^* \mid (\forall\, x \in X\,)\ |u|_x \equiv |v|_x \pmod{m} \,\}$.

Proposition 1. *Let $\mathcal{V}$ be a positive variety of languages and $\mathbf{V}$ be the corresponding pseudovariety of ordered monoids. Then the following conditions are equivalent.*

(i) For each finite alphabet A, the set $\mathcal{V}(A)$ is finite.

(ii) The pseudovariety of ordered monoids $\mathbf{V}$ is locally finite, i.e. each finitely generated submonoid of an arbitrary product of ordered monoids from $\mathbf{V}$ is finite.

(iii) There exists a finite characteristic of $\mathcal{V}$.

Proof. "(i) $\implies$ (ii)" Let $(M_i)_{i\in I}$ be an arbitrary family of ordered monoids from the class $\mathbf{V}$. Let A be a finite set, let $\phi : A^* \to M' = \prod_{i\in I} M_i$ be a morphism, and let $\pi_i : M' \to M_i$ be the i-th projection ($i \in I$). We want to show that $M = \phi(A^*)$ is finite.

For each $m \in M$, we have $\phi^{-1}((m]) = \bigcap_{i \in I} L_i$ where $L_i = (\pi_i \phi)^{-1}((\pi_i(m)])$. We have $L_i \in \mathcal{V}(A)$ as L_i is recognized by M_i. Since we have only finitely many languages in $\mathcal{V}(A)$ we intersect only finitely many languages. Consequently $\phi^{-1}((m]) \in \mathcal{V}(A)$. For different $m, n \in M$, the languages $\phi^{-1}((m])$ and $\phi^{-1}((n])$ are different. Now the finiteness of $\mathcal{V}(A)$ gives that M is finite.

"(ii) $\Longrightarrow$ (iii)" Let $\mathbf{W} = \langle \mathbf{V} \rangle = \mathsf{HSP}\,\mathbf{V}$ be the variety of ordered monoids generated by the pseudovariety $\mathbf{V}$. We claim that the variety $\mathbf{W}$ is locally finite. Indeed, let M be an ordered submonoid of $\prod_{i \in I} M_i$ where each $M_i \in \mathbf{V}$, and let ϕ be a surjective morphism of M onto an ordered monoid N with a finite generating set G. We need to show that N is finite. Let $F \subseteq M$ be a finite set such that $\phi(F) = G$. By assumption (ii), the set F generates in M a finite ordered monoid and N is its image.

It follows that $\gamma = \mathsf{Id}\,\mathbf{W}$ is a finite characteristic for $\mathcal{V}$.

"(iii) $\Longrightarrow$ (i)" Let γ be a finite characteristic for $\mathcal{V}$. Then $L \in \mathcal{V}(A)$ implies that L is a union of classes of A^*/γ_A. Since the set A^*/γ_A is finite there are only finitely many possibilities for L. $\square$

A positive variety $\mathcal{V}$ is called *locally finite* if it satisfies (i) of Proposition 1.

4 Polynomial Operators of Bounded Length

Let $\mathcal{V}$ be a positive variety of languages and let k be a natural number. We define the class $\mathsf{PPol}_k\mathcal{V}$ of *positive polynomials* of length at most k of languages from the class $\mathcal{V}$. Namely, for a finite alphabet A, $\mathsf{PPol}_k\mathcal{V}(A)$ consists of finite unions of finite intersections of the languages of the form

$$L_0 a_1 L_1 a_2 \ldots a_\ell L_\ell, \quad \text{where} \quad \ell \leq k,\ a_1, \ldots, a_\ell \in A,\ L_0, \ldots, L_\ell \in \mathcal{V}(A)\ . \quad (*)$$

Similarly, we define the classes $\mathsf{BPol}_k\mathcal{V}$ of *Boolean polynomials* using all finite Boolean combinations of languages of the form $(*)$. Clearly, it holds that $\mathsf{PPol}_k\mathcal{V} \subseteq \mathsf{PPol}_{k'}\mathcal{V}$ for $k \leq k'$ and the same for BPol's. We denote the union of all $\mathsf{PPol}_k\mathcal{V}$'s by $\mathsf{PPol}\,\mathcal{V}$. Similarly for $\mathsf{BPol}_k\mathcal{V}$'s.

Example 3. (A continuation of Examples 1 and 2.)

1. The case $\mathcal{V} = \mathcal{T}$ was studied in [6]. Notice only that $\mathsf{PPol}\,\mathcal{T}$ is the 1/2-level of the Straubing-Thérien hierarchy and $\mathsf{BPol}\,\mathcal{T}$ is the first level, i.e. the class of all piecewise testable languages.

2. and 3. One can show that $\mathsf{PPol}\,\mathcal{S}^+ = \mathsf{PPol}\,\mathcal{S}$ is the 3/2-level and $\mathsf{BPol}\,\mathcal{S}^+ = \mathsf{BPol}\,\mathcal{S}$ is the second level – see Theorem 8.8 in [8].

Lemma 3. *If $\mathcal{V}$ is a positive variety of languages then $\mathsf{PPol}_k\mathcal{V}$ is a positive variety of languages and $\mathsf{BPol}_k\mathcal{V}$ is a Boolean variety of languages.*

Proof. One can prove the statements directly. For locally finite varieties it also immediately follows from Theorem 1. $\square$

Let k be a fixed natural number and α be a finite characteristic. Let A be a fixed set; in particular, A can be a finite alphabet or the set X.

For a word $u \in A^*$, we say that

$$f = (u_0, a_1, \dots, a_\ell, u_\ell)$$

is *a factorization* of u of length ℓ if $u_0, u_1, \dots, u_\ell \in A^*$, $a_1, a_2, \dots, a_\ell \in A$ and $u_0 a_1 u_1 \dots a_\ell u_\ell = u$. The set of all factorizations of length at most k of the word u is denoted by $\mathsf{Fact}_k(u)$. For a factorization $f = (u_0, a_1, \dots, a_\ell, u_\ell)$ of a word $u \in A^*$ and a factorization $g = (v_0, b_1, v_1, \dots b_m, v_m)$ of a word $v \in A^*$, we write

$$f \leq_\alpha g$$

if $\ell = m$, $a_i = b_i$ for every $i \in \{1, \dots, \ell\}$ and $u_i \;\alpha_A\; v_i$ for every $i \in \{0, 1, \dots, \ell\}$. We define the relation $(\mathsf{p}_k(\alpha))_A$ on the set A^* as follows: for $u, v \in A^*$, we have

$$u \;(\mathsf{p}_k(\alpha))_A\; v \quad \text{if and only if} \quad (\forall\, g \in \mathsf{Fact}_k(v)\,)\;(\exists\, f \in \mathsf{Fact}_k(u)\,)\; f \leq_\alpha g\ .$$

We show in Theorem 1 that the relation $(\mathsf{p}_k(\alpha))_X$ is a finite characteristic and therefore the relation $(\mathsf{p}_k(\alpha))_A$ is equal to $((\mathsf{p}_k(\alpha))_X)_A$ as defined after Definition 1. We write $\mathsf{p}_k(\alpha)$ instead of $(\mathsf{p}_k(\alpha))_X$. Further we denote $\mathsf{b}_k(\alpha) = \widehat{\mathsf{p}_k(\alpha)}$.

Theorem 1. *Let $\mathcal{V}$ be a locally finite positive variety of languages and α be the finite characteristic of $\mathcal{V}$. Then $\mathsf{PPol}_k\mathcal{V}$ is a locally finite positive variety of languages with the finite characteristic $\mathsf{p}_k(\alpha)$ and $\mathsf{BPol}_k\mathcal{V}$ is a locally finite Boolean variety of languages with the finite characteristic $\mathsf{b}_k(\alpha)$.*

Proof. We prove that $\mathsf{p}_k(\alpha)$ is a finite characteristic of $\mathsf{PPol}_k\mathcal{V}$. The rest follows from Lemma 1 and Lemma 2.

We have to check the properties (i) – (iv) from Definition 1 and also the property

(v) $\qquad L \in \mathsf{PPol}_k\mathcal{V}(A) \quad \text{if and only if} \quad (\mathsf{p}_k(\alpha))_A \subseteq \preceq_L .$

"(i)" The reflexivity of the relation $\mathsf{p}_k(\alpha)$ is trivial. The transitivity follows from the transitivity of the relation $\leq_\alpha$.

"(ii)" Let $u, v, w \in X^*$ be such that $(u, v) \in \mathsf{p}_k(\alpha)$. We want to show that $(uw, vw) \in \mathsf{p}_k(\alpha)$. Let $g \in \mathsf{Fact}_k(vw)$ be an arbitrary factorization of length at most k of the word vw, i.e. $g = (v_0, a_1, v_1, \dots, a_\ell, v_\ell)$, where $\ell \leq k$, $a_1, \dots, a_\ell \in X$, $v_0, \dots, v_\ell \in X^*$ and there exist $0 \leq i \leq \ell$ and $v_i', v_i'' \in X^*$ such that $v_i' v_i'' = v_i$ and

$$v = v_0 a_1 v_1 \dots a_i v_i'\ , \quad w = v_i'' a_{i+1} \dots a_\ell v_\ell\ .$$

From the assumption $(u, v) \in \mathsf{p}_k(\alpha)$ we know that there is a factorization f of the word u such that $f \leq_\alpha (v_0, a_1, v_1, \dots, a_i, v_i')$, i.e. $f = (u_0, a_1, u_1, \dots, a_i, u_i')$ such that $u_0 \;\alpha\; v_0$, $\dots$, $u_i' \;\alpha\; v_i'$. Since α is a compatible quasiorder we have $u_i' v_i'' \;\alpha\; v_i' v_i''$. Hence

$$h = (u_0, a_1, u_1, \dots, a_i, u_i' v_i'', a_{i+1}, \dots, a_\ell, v_\ell)$$

is a factorization of uw such that $h \leq_\alpha g$. This implies $(uw, vw) \in \mathsf{p}_k(\alpha)$.

The proof of the implication "$(u,v) \in \mathsf{p}_k(\alpha) \implies (wu, wv) \in \mathsf{p}_k(\alpha)$" is similar.

"(iii)" Let $u, v \in X^*$ be such that $(u,v) \in \mathsf{p}_k(\alpha)$ and $\varphi : X^* \to X^*$ be an arbitrary morphism. We want to show that $(\varphi(u), \varphi(v)) \in \mathsf{p}_k(\alpha)$. So, let

$$g' = (v_0, a_1, v_1 \ldots, a_\ell, v_\ell) \in \mathsf{Fact}_k(\varphi(v))$$

where $\ell \leq k$, $v_i \in X^*, a_i \in X$ and $v_0 a_1 v_1 \ldots a_\ell v_\ell = \varphi(v)$. We consider a factorization $g = (w_0, b_1, w_1, \ldots, b_m, w_m)$ of v where the occurrences of the letters $b_1, \ldots, b_m$ are such that the corresponding occurrences of $\varphi(b_1), \ldots, \varphi(b_m)$ in $\varphi(v)$ contain all a_i's in the factorization g'. Note that $m \leq \ell$ as $\varphi(b_j)$ can contain more than one a_i. Now $(u,v) \in \mathsf{p}_k(\alpha)$ and there exists a factorization f of u such that $f \leq_\alpha g$, i.e. $f = (t_0, b_1, t_1 \ldots, b_m, t_m)$ where $t_i \;\alpha\; w_i$ for $i \in \{0, \ldots, m\}$. Since α is a finite characteristic we have $\varphi(t_i) \;\alpha\; \varphi(w_i)$. Hence $\varphi(u) = \varphi(t_0)\varphi(b_1)\varphi(t_1) \ldots \varphi(b_m)\varphi(t_m)$ has a factorization f' such that $f' \leq_\alpha g'$. We can conclude that $(\varphi(u), \varphi(v)) \in \mathsf{p}_k(\alpha)$.

"(iv)" Let Y be a finite subset of X. Since $\widehat{\alpha_Y}$ has a finite index, there are only finitely many factorizations of length at most k over Y when identifying the $\widehat{\leq_\alpha}$-related ones. Hence there are only finitely many sets of the form $\mathsf{Fact}_k(u)$ up to $\widehat{\leq_\alpha}$, where $u \in Y^*$. So, $\widehat{\mathsf{p}_k(\alpha)|_Y}$ has a finite index too.

"(v)" For simplicity denote the relation $(\mathsf{p}_k(\alpha))_A$ by β.

" $\implies$ " We prove that for every language

$$L = L_0 a_1 L_1 \ldots a_\ell L_\ell, \quad \text{where} \quad \ell \leq k,\ a_1, \ldots, a_\ell \in A,\ L_0, \ldots, L_\ell \in \mathcal{V}(A)\ ,$$

we have $\beta \subseteq \preceq_L$. This is enough because $\beta \subseteq \preceq_L$ and $\beta \subseteq \preceq_K$ imply $\beta \subseteq \preceq_{L\cap K}$ and $\beta \subseteq \preceq_{L\cup K}$, for each $L, K \subseteq A^*$.

Let L be such a language and let $u, v \in A^*$ satisfy $u \;\beta\; v$. We want to show that $u \preceq_L v$. So, let $p, q \in A^*$ be such that $pvq \in L$. Hence $pvq = v_0 a_1 v_1 \ldots a_\ell v_\ell$, where $v_i \in L_i$ for every $i \in \{0, \ldots, \ell\}$. Then there exist $0 \leq i < j \leq \ell$ and $v_i', v_i'', v_j', v_j'' \in A^*$, such that $v_i' v_i'' = v_i$, $v_j' v_j'' = v_j$ and

$$p = v_0 a_1 \ldots v_i', \quad v = v_i'' a_{i+1} \ldots a_j v_j' \quad \text{ and } q = v_j'' a_{j+1} \ldots a_\ell v_\ell$$

or there exist $0 \leq i \leq \ell$ and $v_i', v_i'', v_i''' \in A^*$ such that $v_i' v_i'' v_i''' = v_i$ and

$$p = v_0 a_1 \ldots v_i', \quad v = v_i'' \quad \text{ and } q = v_i''' a_{i+1} \ldots a_\ell v_\ell\ .$$

In the first case we have $g = (v_i'', a_{i+1}, \ldots, a_j, v_j')$ a factorization of v. We assumed that $u \;\beta\; v$, so there is a factorization $f = (u_i'', a_{i+1}, \ldots, u_j')$ of u such that $(u_i'', v_i''), (u_{i+1}, v_{i+1}), \ldots, (u_j', v_j') \in \alpha_A$. Since α_A is a compatible quasiorder we have $(v_i' u_i'', v_i' v_i'') \in \alpha_A$ and hence $v_i' u_i'' \preceq_{L_i} v_i' v_i'' = v_i$, so we have $v_i' u_i'' \in L_i$. Similarly $u_{i+1} \in L_{i+1}$, $\ldots$, $u_{j-1} \in L_{j-1}$ and $u_j' v_j'' \in L_j$. Consequently $puq \in L$. The second case is similar and we see that $u \;\beta\; v$ really implies $u \preceq_L v$.

"$\Longleftarrow$" Let $\beta \subseteq \preceq_L$. This means that L is a finite union of languages of the form

$$\beta v = \{\, u \in A^* \mid u \;\beta\; v \,\}, \quad \text{where} \;\; v \in A^* \;.$$

It is enough to prove that each βv belongs to $\mathsf{PPol}_k\mathcal{V}(A)$. Consider all possible factorizations of the word v of length at most k, i.e. all elements of the set $\mathsf{Fact}_k(v)$. So, we have

$$g_1 = (v_{10}, a_{11}, \ldots, a_{1\ell_1}, v_{1\ell_1}) \;,$$

$$g_2 = (v_{20}, a_{21}, \ldots, a_{2\ell_2}, v_{2\ell_2}) \;,$$

$$\vdots$$

$$g_m = (v_{m0}, a_{m1}, \ldots, a_{m\ell_m}, v_{m\ell_m}) \;,$$

where for each $i \in \{1, \ldots, m\}$ we have $\ell_i \leq k$ and $a_{ij} \in A$ are letters and $v_{ij} \in A^*$ are words and $\{g_1, g_2, \ldots, g_m\} = \mathsf{Fact}_k(v)$. For each $i \in \{1, \ldots, m\}$ we consider the following language L_i corresponding to the factorization $g_i = (v_{i0}, a_{i1}, \ldots, a_{i\ell_i}, v_{i\ell_i})$:

$$L_i = L_{i0} \; a_{i1} \; L_{i1} \; \ldots \; a_{i\ell_i} \; L_{i\ell_i} \;,$$

where $\;L_{ij} = \alpha_A v_{ij} = \{\, u \in A^* \mid u \;\alpha_A\; v_{ij} \,\} \in \mathcal{V}(A)\;$ for each $\;j \in \{0, \ldots, \ell_i\}$. Then the language

$$K = \bigcap_{i=1}^{m} L_i$$

belongs to $\mathsf{PPol}_k\mathcal{V}(A)$ and we prove that $K = \beta v$.

"$\subseteq$" If $u \in K$ then $u \in L_i$ for each $i \in \{1, \ldots, m\}$. This means that for each $i \in \{1, \ldots, m\}$ we have

$$u = u_{i0} a_{i1} \ldots a_{i\ell_i} u_{i\ell_i} \;,$$

where $(u_{i0}, v_{i0}), \ldots, (u_{i\ell_i}, v_{i\ell_i}) \in \alpha_A$. Therefore, there is a factorization f_i of u such that $f_i \leq_\alpha g_i$. Consequently $(u, v) \in (\mathsf{p}_k(\alpha))_A = \beta$, i.e. $u \in \beta v$.

"$\supseteq$" If $u \in \beta v$. Then for each $i \in \{1, \ldots, m\}$, we have some factorization f_i of u such that $f_i \leq_\alpha g_i$. This implies that $u \in L_i$ for each $i \in \{1, \ldots, m\}$, and hence $u \in K$. □

The following lemmas concern the preservation of aperiodicity (i.e. monoids have only trivial subgroups).

Lemma 4. *Let α be a finite characteristic and let k, n be arbitrary natural numbers. Put $m = (k+1)(n+1)$.*

(i) If $(x^n, x^{n+1}) \in \alpha$ then $(x^{m-1}, x^m) \in \mathsf{p}_k(\alpha)$.

(ii) If $(x^n, x^{n+1}) \in \widehat{\alpha}$ then $(x^{m-1}, x^m) \in \widehat{\mathsf{p}_k(\alpha)}$.

Proof. "(i)" Let g be a factorization of x^m of length $\ell \le k$, i.e.

$$g = (x^{i_0}, x, x^{i_1}, x, \dots, x, x^{i_\ell})$$

where $i_0+i_1+\cdots+i_\ell+\ell = m$ and $i_0, \dots, i_\ell$ are non-negative integers. Assume that for every $j \in \{0, \dots, \ell\}$ we have $i_j \le n$, then $i_0+i_1+\cdots+i_\ell+\ell \le (\ell+1)n+\ell \le (k+1)n+k < (k+1)(n+1) = m$ a contradiction. Thus, there is $j \in \{0, \dots, \ell\}$ such that $i_j \ge n+1$, hence $x^{i_j-1} \ \alpha \ x^{i_j}$ and consequently there exists a factorization f of x^{m-1} such that $f \le_\alpha g$. This proves $(x^{m-1}, x^m) \in \mathsf{p}_k(\alpha)$.

"(ii)" With respect to the part (i) it is enough to prove the implication $(x^{n+1}, x^n) \in \alpha \implies (x^m, x^{m-1}) \in \mathsf{p}_k(\alpha)$. This is not a direct consequence of statement (i) since $(\mathsf{p}_k(\alpha))^{\mathsf{d}} \ne \mathsf{p}_k(\alpha^{\mathsf{d}})$ but the implication can be proved in a similar way as part (i). □

Lemma 5. *Let $\mathcal{V}$ be a positive variety with the finite characteristic α, such that the corresponding pseudovariety of ordered monoids contains only aperiodic monoids. Then, for each natural number k, the pseudovariety of ordered monoids corresponding to the positive variety of languages $\mathsf{PPol}_k\mathcal{V}$ contains only aperiodic monoids too.*

Proof. Let $A = \{a\}$ be an alphabet. Then A^*/α_A belongs to the corresponding pseudovariety of monoids, i.e. A^*/α_A is a finite aperiodic monoid. This implies that $(a^n, a^{n+1}) \in \widehat{\alpha_A}$ for some natural number n and $(x^n, x^{n+1}) \in \widehat{\alpha}$ follows. By Lemma 4, we have $(x^{m-1}, x^m) \in \widehat{\mathsf{p}_k(\alpha)}$ for a certain m. Hence for every alphabet B, the monoid B^*/α_B is aperiodic, and consequently the pseudovariety of monoids corresponding to the positive variety of languages $\mathsf{PPol}_k\mathcal{V}$ contains only aperiodic monoids because each of them is a morphic images of the monoid B^*/α_B for some B. □

5 Generating Pseudovarieties by a Single Monoid

It is known (see Volkov [11] or the authors [6]) that the pseudovarieties of ordered monoids corresponding to $\mathsf{PPol}_k\mathcal{T}$, k a natural number, are generated by a single ordered monoid. We show such result also for the positive varieties $\mathsf{PPol}_k\mathcal{S}^+$ and we prove that this is not true for the positive varieties $\mathsf{PPol}_k\mathcal{S}$. At first we define a "finiteness-like" condition concerning finite characteristics.

Definition 2. *Let α be a finite characteristic. We say that α is* **finitely determined** *if there is a finite alphabet A such that for every finite alphabet B and all $u, v \in B^*$ we have:*

$$(\ (\forall\ \varphi : B \to A\ )\ \varphi(u)\ \alpha_A\ \varphi(v)\ ) \quad \textit{implies} \quad u\ \alpha_B\ v\ .$$

The extension of a mapping $\varphi : B \to A$ to a morphism from B^* to A^* is denoted by the same symbol. Clearly, the opposite implication is always true due to Definition 1.

Example 4. The finite characteristic of the positive variety $\mathcal{S}^+$ was described in Example 2.2. It is finitely determined since one can show that the condition from the previous definition is satisfied for $A = \{a, a'\}$, $a \neq a'$. Indeed, for an arbitrary finite alphabet B and $u, v \in B^*$ such that $\mathsf{cont}(u) \not\subseteq \mathsf{cont}(v)$ we can consider a letter $b \in \mathsf{cont}(u) \setminus \mathsf{cont}(v)$. Then we take a mapping $\varphi : B \to A$ sending b to a and (possible) other elements of B to a'. For this φ we have $a \in \mathsf{cont}(\varphi(u)) \setminus \mathsf{cont}(\varphi(v))$.

The same considerations for two element set A are true for $\mathcal{S}$ and for $\mathcal{A}_m$.

Proposition 2. *The following properties for a positive variety $\mathcal{V}$ and the corresponding pseudovariety of ordered monoids $\mathbf{V}$ are equivalent.*

(i) The positive variety $\mathcal{V}$ is generated by a finite number of languages.

(ii) The pseudovariety $\mathbf{V}$ is generated by a single ordered monoid.

(iii) There exists a finite characteristic of $\mathcal{V}$ which is finitely determined.

Proof. "$(i) \implies (ii)$" If $\mathcal{V}$ is generated by a finite number of languages then we can take their syntactic ordered monoids and consider the product of all of them. The resulting ordered monoid generates the pseudovariety of ordered monoids $\mathbf{V}$.

"$(ii) \implies (iii)$" Let the pseudovariety $\mathbf{V}$ be generated by a single finite ordered monoid M. We consider the variety $\mathbf{W} = \langle \mathbf{V} \rangle = \langle M \rangle$ generated by the monoid M. If we take the free ordered monoid F over X in the variety $\mathbf{W}$ and denote α the kernel of the projection from X^* onto F, then this α is a finite characteristic of $\mathcal{V}$. Moreover, for a finite alphabet C, the (finite) structure C^*/α_C is a free ordered monoid over C in $\mathbf{W}$.

Now we put $A = M$ and we prove the property from Definition 2 for this set A. At first, there is a natural morphism $\theta : A^* \to M$ which maps the word $a_1 a_2 \ldots a_m \in A^*$ to the product of elements $a_1, a_2, \ldots, a_m \in A = M$ in M, i.e. $\theta(a_1 a_2 \ldots a_m) = a_1 \cdot a_2 \cdot \ldots \cdot a_m$. Note that M is a morphic image of the free ordered monoid A^*/α_A, in other words, α_A is a subset of the kernel of θ.

Let B be a finite alphabet and $u, v \in B^*$ be such that for each $\varphi : B \to A$ we have $\varphi(u)\ \alpha_A\ \varphi(v)$. Each mapping $\varphi : B \to A = M$ determines a morphism $\overline{\varphi} = \theta \circ \varphi : B^* \to M$.

Recall that a free monoid over B in $\mathbf{W}$ can be constructed in the following way. There are only finitely many mappings $\varphi : B \to M$; denote Σ the set of all of them. Then we consider the finite product $\prod_{\varphi \in \Sigma} M = M^\Sigma$ and the corresponding morphism $\psi : B^* \to M^\Sigma$ given by $\psi(w) = (\overline{\varphi}(w))_{\varphi \in \Sigma}$. The image of ψ is a free monoid over B in $\mathbf{W}$ and α_B is a kernel of ψ. Now for each $\varphi : B \to A = M$ we have $\varphi(u)\ \alpha_A\ \varphi(v)$. Thus $\varphi(u) \leq \varphi(v)$ in A^*/α_A and $\overline{\varphi}(u) \leq \overline{\varphi}(v)$ in M follows. Hence $\psi(u) \leq \psi(v)$ in the free ordered monoid B^*/α_B and consequently $u\ \alpha_B\ v$.

"$(iii) \implies (i)$" Let α be a finite characteristic of $\mathcal{V}$ which is finitely determined. Let A be the corresponding finite alphabet. Since α_A has a finite index, there are only finitely many languages of the form $\alpha_A v = \{\, u \in A^* \mid u\ \alpha_A\ v \,\}$ where $v \in A^*$. We show that these languages generate $\mathcal{V}$.

Let B be an arbitrary finite alphabet and let $L \in \mathcal{V}(B)$. Since α is a finite characteristic of $\mathcal{V}$ we have $\alpha_B \subseteq \preceq_L$. Hence L is a finite union of languages of the form $\alpha_B w = \{\, t \in B^* \mid t \ \alpha_B \ w \,\}$, where $w \in B^*$.

There are only finitely many mappings from B to A; denote them $\varphi_1, \dots, \varphi_m$, where $m = |A|^{|B|}$. Now for every $u, v \in B^*$ we have

$$u \ \alpha_B \ v \ \text{ if and only if } \ (\ \forall\, i \in \{1, \dots, m\}\ )\ \ \varphi_i(u) \ \alpha_A \ \varphi_i(v)\ .$$

We show that

$$\alpha_B w = \bigcap_{i=1}^{m} \varphi_i^{-1}(\alpha_A w_i), \quad \text{ where } \quad w_i = \varphi_i(w) \quad \text{for} \quad i \in \{1, \dots, m\}\ . \qquad (\ddagger)$$

Indeed, for $t \in B^*$, it holds $t \in \alpha_B w$ if and only if for each $i \in \{1, \dots, m\}$ we have $\varphi_i(t) \ \alpha_A \ \varphi_i(w) = w_i$, and this is equivalent to: for each $i \in \{1, \dots, m\}$ we have $t \in \varphi_i^{-1}(\alpha_A w_i)$.

Equation ($\ddagger$) means that we can obtain each language of the form $\alpha_B w$ from the languages $\alpha_A v$, for $v \in A^*$, when we use inverse morphisms and the operation of intersection. □

Example 5. In paper [6] the authors proved that $\mathsf{PPol}_k\mathcal{T}$ is generated by a language $A^*a_1A^*a_2 \dots a_kA^*$ where $a_1, a_2, \dots, a_k$ are pairwise different letters and $A = \{a_1, \dots, a_k\}$. We show that the corresponding finite characteristic $\alpha = \mathsf{p}_k(X^* \times X^*)$ is finitely determined.

Indeed, let $\mathsf{Sub}_k(w)$ denote the set of all subwords of $w \in X^*$ of length at most k. Then $u \ \alpha \ v$ if and only if $\mathsf{Sub}_k(v) \subseteq \mathsf{Sub}_k(u)$. Let $A' = \{a_1, \dots, a_{k+1}\}$ be of cardinality $k+1$, let B be a finite set, and let $u, v \in B^*$ satisfy $\varphi(u) \ \alpha_{A'} \ \varphi(v)$ for each $\varphi : B \rightarrow A'$. Suppose that $u \ \alpha_B \ v$ does not hold. Then there exists $w \in B^*$ of length at most k such that $w \in \mathsf{Sub}_k\,(v) \setminus \mathsf{Sub}_k\,(u)$. Let $C = \mathsf{cont}(w)$. Take an injective mapping $\varphi : C \rightarrow \{a_1, \dots, a_k\}$ and put $\varphi(b) = a_{k+1}$ for $b \notin C$. Thus $\varphi : B \rightarrow A'$ and $\varphi(w) \in \mathsf{Sub}_k(\varphi(v)) \setminus \mathsf{Sub}_k(\varphi(u))$ – a contradiction.

Proposition 3. *The positive variety* $\mathsf{PPol}_k\mathcal{S}^+$ *is generated by a finite number of languages.*

Proof. Although a direct proof would be possible we apply Proposition 2. Recall that the finite characteristic α for $\mathcal{S}^+$ is given as follows: for each $u, v \in X^*$, we have $u \ \alpha \ v$ if and only if $\mathsf{cont}(u) \subseteq \mathsf{cont}(v)$. We show that the finite characteristic $\beta = \mathsf{p}_k(\alpha)$ of $\mathsf{PPol}_k\mathcal{S}^+$ is finitely determined.

Let A be an alphabet containing 2^{2k+1} letters:

$$A = \{\, a_r \mid r \in \{0,1\}^{2k+1} \,\}\ .$$

We prove the property from Definition 2. Let B be a finite alphabet and assume that $u, v \in B^*$ satisfy

$$(\ \forall\, \varphi : B \rightarrow A\ )\ \ \varphi(u) \ \beta_A \ \varphi(v)\ .$$

We want to prove $u\ \beta_B\ v$. So, let $g = (v_0, b_1, v_1, \ldots, b_\ell, v_\ell) \in \mathsf{Fact}_k(v)$ be an arbitrary factorization of length at most k of the word v. For each letter $c \in B$ we consider the letter $a_r \in A$ where the sequence r has 1 at j-th position if and only if c is at the j-th position in the factorization g. More precisely, $r_{2i+1} = 1$ iff $c \in \mathsf{cont}(v_i)$ and $r_{2i} = 1$ iff $c = b_i$. So, we have defined a mapping $\varphi : B \to A$. Note that if a letter c does not occur in v then $\varphi(c) = a_{(0,0,\ldots,0)}$ by this definition. Now $\varphi(u)\ \beta_A\ \varphi(v)$ and there exists a factorization f' of $\varphi(u)$ such that $f' \leq_\alpha g' = (\varphi(v_0), \varphi(b_1), \varphi(v_1), \ldots, \varphi(b_\ell), \varphi(v_\ell))$. If $\varphi(b_i) = a_r$ then $r_{2i} = 1$ and for this r there is a unique letter $c \in B$, namely b_i, with the property $\varphi(c) = a_r$. Hence we have a factorization $f = (u_0, b_1, u_1, \ldots, b_\ell, u_\ell)$ of u such that $\varphi(u_i)\ \alpha_A\ \varphi(v_i)$ for each $i \in \{0, \ldots, \ell\}$. We show that this implies $u_i\ \alpha_B\ v_i$. Let $d \in \mathsf{cont}(u_i)$ be an arbitrary letter from the alphabet B. Then $\varphi(d) \in \mathsf{cont}(\varphi(u_i)) \subseteq \mathsf{cont}(\varphi(v_i))$. Let $\varphi(d) = a_r$. Then $a_r \in \mathsf{cont}(\varphi(v_i))$ implies that $r_{2i+1} = 1$. If $d \notin \mathsf{cont}(v_i)$ then $r_{2i+1} = 0$ by the definition of the mapping φ. Hence $d \in \mathsf{cont}(v_i)$, and thus, we have $u_i\ \alpha_B\ v_i$ for each $i = 0, \ldots, \ell$. For a given $g \in \mathsf{Fact}_k(v)$, we found $f \in \mathsf{Fact}_k(u)$ such that $f \leq_\alpha g$. This means that we proved $u\ \beta_B\ v$. □

Proposition 4. *The positive variety* $\mathsf{PPol}_1\mathcal{S}$ *is generated by a finite number of languages.*

Proof. Recall that the finite characteristic α for $\mathcal{S}$ is given as follows: for each $u, v \in X^*$, we have $u\ \alpha\ v$ if and only if $\mathsf{cont}(u) = \mathsf{cont}(v)$.

We show that finite characteristic $\beta = \mathsf{p}_1(\alpha)$ of the variety $\mathsf{PPol}_1\mathcal{S}$ is finitely determined on a six-element alphabet $A = \{a_0, a_1, a_2, a_3, a_4, a_5\}$. First we formulate some basic consequences of the assumption $s\ \beta_A\ t$ for a pair of words $s, t \in A^*$. We have $\mathsf{cont}(s) = \mathsf{cont}(t)$ since we can consider (unique) factorizations of s and t of length 0. Further, if we assume that the first occurrence of a letter $a \in A$ in s is before the first occurrence of a letter $a' \in A$ in s then there is a factorization (s_0, a', s_1) of the word s such that $a \in \mathsf{cont}(s_0)$, $s_0, s_1 \in A^*$ but there is no factorization (s_0, a, s_1) of s such that $a' \in \mathsf{cont}(s_0)$, $s_0, s_1 \in A^*$. Thus from $s\ \beta_A\ t$ we can conclude that the sequences of the first occurrences of all letters in s and in t coincide. Equivalently this can be expressed by the equality $\{\, \mathsf{cont}(s') \mid s' \text{ prefix of } s \,\} = \{\, \mathsf{cont}(t') \mid t' \text{ prefix of } t \,\}$. The similar observations can be done for the last occurrences of letters in s and t.

Let B be a finite alphabet containing at least seven letters[1] and assume that for a given pair of words $u, v \in B^*$ we have

$$(\ \forall\ \varphi : B \to A\ )\ \ \varphi(u)\ \beta_A\ \varphi(v)\ .$$

Let $g = (g_0, b, g_1) \in \mathsf{Fact}_1(v)$ be a factorization of v. We need to show that there exists a factorization $f = (f_0, b, f_1) \in \mathsf{Fact}_1(u)$ such that $\mathsf{cont}(f_0) = \mathsf{cont}(g_0)$ and $\mathsf{cont}(f_1) = \mathsf{cont}(g_1)$.

First of all, we take an arbitrary pair of different letters $b_1, b_2 \in B$ and consider the mapping $\varphi_{b_1,b_2} : B \to A$ given by the rules $\varphi_{b_1,b_2}(b_1) = a_1, \varphi_{b_1,b_2}(b_2) = a_2$

[1] For alphabets with at most six letters the statement is trivial.

and $\varphi_{b_1,b_2}(c) = a_0$ for all $c \in B \setminus \{b_1, b_2\}$. Since $(\varphi_{b_1,b_2}(u), \varphi_{b_1,b_2}(v)) \in \beta_A$ we can apply our basic observations concerning β_A and we see that $\mathsf{cont}(\varphi_{b_1,b_2}(u)) = \mathsf{cont}(\varphi_{b_1,b_2}(v))$ from which we observe $b_1 \in \mathsf{cont}(u) \iff b_1 \in \mathsf{cont}(v)$. This is true for each b_1 and thus $\mathsf{cont}(u) = \mathsf{cont}(v)$ follows. Further, the sequence of the first occurrences of letters in $\varphi_{b_1,b_2}(u)$ and $\varphi_{b_1,b_2}(v)$ coincide. Hence the first occurrence of b_1 in the word u is before the first occurrence of b_2 in u if and only if the first occurrence of b_1 in v is before the first occurrence of b_2 in v. This is true for every pair of letters b_1 and b_2 and we can summarize that $\{\, \mathsf{cont}(u') \mid u' \text{ prefix of } u \,\} = \{\, \mathsf{cont}(v') \mid v' \text{ prefix of } v \,\}$. When we consider the same idea from the right we obtain the same observations concerning the last occurrences of letters and finally we obtain the equality $\{\, \mathsf{cont}(u') \mid u' \text{ suffix of } u \,\} = \{\, \mathsf{cont}(v') \mid v' \text{ suffix of } v \,\}$.

There is a prefix u' of the word u such that $\mathsf{cont}(u') = \mathsf{cont}(g_0)$. Let u_1 be the shortest prefix of u with this property and u_2 be the longest prefix of u with this property. Note that u_1 can be the empty word (when $\mathsf{cont}(g_0) = \emptyset$, i.e. in the case $g_0 = \lambda$) and u_2 can be equal to u (when $\mathsf{cont}(g_0) = \mathsf{cont}(v)$). If u_1 is not the empty word then $u_1 = u_1' b_1$ where $b_1 \in B$ and $b_1 \in \mathsf{cont}(u_1) = \mathsf{cont}(g_0), b_1 \notin \mathsf{cont}(u_1')$. A useful consequence is that this b_1 is the first occurrence of b_1 in u. Similarly, if $u_2 \neq u$ then $u = u_2 b_2 u_2'$ where $b_2 \in B$, $u_2' \in B^*$ and $b_2 \notin \mathsf{cont}(u_2) = \mathsf{cont}(g_0)$. Once again this b_2 is the first occurrence of b_2 in u. Note that if b_1 and b_2 are defined then they are different because $b_2 \notin \mathsf{cont}(g_0)$, but one of them can be equal to the letter b. These definitions can be also consider dually from the right. I.e. we can consider the shortest suffix u_3 of u and the longest suffix u_4 of u with the properties $\mathsf{cont}(u_3) = \mathsf{cont}(u_4) = \mathsf{cont}(g_1)$. If $u_3 \neq \lambda$ then we denote its first letter b_3, i.e $u_3 = b_3 u_3'$ and we have $b_3 \in \mathsf{cont}(u_3) = \mathsf{cont}(g_1)$, $b_3 \notin \mathsf{cont}(u_3')$. If $u_4 \neq u$ then we denote $u = u_4' b_4 u_4$ where $b_4 \in B$, $u_4' \in B^*$, $b_4 \notin \mathsf{cont}(u_4) = \mathsf{cont}(g_1)$.

Now we have the subset $B' = \{b, b_1, b_2, b_3, b_4\}$ of the alphabet B which has at most five elements. Note that some of the letters can be equal, some of them can not be defined. We consider some mapping $\varphi : B \to A$ such that $\varphi(c) = a_5$ for every $c \notin B'$, $\varphi(B') \subseteq A \setminus \{a_5\}$, $\varphi(b) = a_0$ and which is injective on B'. Then $(\varphi(g_0), a_0, \varphi(g_1))$ is a factorization of $\varphi(v)$ and there is a factorization $f = (f_0, d, f_1)$ of u such that $(\varphi(f_0), \varphi(d), \varphi(f_1)) \leq_\alpha (\varphi(g_0), \varphi(b), \varphi(g_1))$ where $\varphi(d) = \varphi(b)$, i.e. $d = b$, $\varphi(f_0)\ \alpha_A\ \varphi(g_0)$ and $\varphi(f_0)\ \alpha_A\ \varphi(g_0)$. We show that $\mathsf{cont}(f_0) = \mathsf{cont}(g_0)$ and $\mathsf{cont}(f_1) = \mathsf{cont}(g_1)$.

"$\mathsf{cont}(g_0) \subseteq \mathsf{cont}(f_0)$" If $\mathsf{cont}(g_0) = \emptyset$ then it is clear. If $\mathsf{cont}(g_0) \neq \emptyset$ then $b_1 \in \mathsf{cont}(g_0)$ is defined. Hence $\varphi(b_1) \in \mathsf{cont}(\varphi(g_0)) = \mathsf{cont}(\varphi(f_0))$ and since φ is injective on B' we have $b_1 \in \mathsf{cont}(f_0)$. By the definition of b_1 we can conclude that u_1 is a prefix of f_0, so, $\mathsf{cont}(g_0) = \mathsf{cont}(u_1) \subseteq \mathsf{cont}(f_0)$.

"$\mathsf{cont}(f_0) \subseteq \mathsf{cont}(g_0)$" If $\mathsf{cont}(g_0) = \mathsf{cont}(v) = \mathsf{cont}(u)$ then it is clear. If $\mathsf{cont}(g_0) \neq \mathsf{cont}(v)$ then b_2 is defined. We have $b_2 \notin \mathsf{cont}(g_0)$. Hence $\varphi(b_2) \notin \mathsf{cont}(\varphi(g_0)) = \mathsf{cont}(\varphi(f_0))$ and this implies $b_2 \notin \mathsf{cont}(f_0)$. By the definition of b_2 we can conclude that f_0 is a prefix of u_2, so, $\mathsf{cont}(f_0) \subseteq \mathsf{cont}(u_2) = \mathsf{cont}(g_0)$.

One can prove the equality $\mathsf{cont}(f_1) = \mathsf{cont}(g_1)$ in the same way using the letters b_3 and b_4. $\square$

Proposition 5. *The positive variety* $\mathsf{PPol}_2\mathcal{S}$ *is not generated by a finite number of languages.*

Proof. For the finite characteristic α for $\mathcal{S}$ we have, for each $u, v \in X^*$, it holds $u\ \alpha\ v$ if and only if $\mathsf{cont}(u) = \mathsf{cont}(v)$

Assume that the finite characteristic $\beta = \mathsf{p}_2(\alpha)$ of the positive variety $\mathsf{PPol}_2\mathcal{S}$ is finitely determined. Let $A = \{c_1, \dots, c_m\}$ be an alphabet for which the property from Definition 2 is satisfied. Let $B = A \cup \{d\}$, $d \notin A$. Assume that $s_1, \dots, s_n$ are all words of length at most $m + 1$ over the alphabet A such that $\mathsf{cont}(s_j) \neq A$ for $j \in \{1, \dots, n\}$. Further $t_{j_0 j_1 j_2} = dc_{j_0} ds_{j_1} dc_{j_2} d$ for all $j_1 \in \{1, \dots, n\}$, $j_0, j_2 \in \{1, \dots, m\}$ and t be a product of all words $t_{j_0 j_1 j_2}$ in a fixed order. Finally, we denote $s = c_1 \dots c_m$ and we define a pair of words over the alphabet B:

$$u = sstt\ ttss \quad \text{and} \quad v = sstt\ dsd\ ttss\ .$$

We show that this pair of words contradicts the assumption, namely we show
(i) $(u, v) \notin \beta_B$ and
(ii) for each $\varphi : B \to A$ we have $\varphi(u)\ \beta_A\ \varphi(v)$.

To prove the first claim we can consider the factorization

$$g = (sstt,\ d,\ s,\ d,\ ttss)$$

of the word v. For this g there is no factorization f of the word u such that $f \leq_\alpha g$ because there are no two consecutive occurrences of d in u such that the word between them has a content equal to the set A.

The second claim is more complicated. Let $\varphi : B \to A$ be a mapping. We consider two cases.

I) First assume that there is a letter $c_i \in A$ such that $\varphi(c_i) = \varphi(d)$. Then we consider the mapping $\varphi' : B \to A$ such that $\varphi'|_A$ is the identity mapping and $\varphi'(d) = c_i$ and the mapping $\varphi'' : A \to A$ such that $\varphi''(c) = \varphi(c)$ for each $c \in A$. Then $\varphi = \varphi'' \circ \varphi'$ and it is enough to show that $\varphi'(u)\ \beta_A\ \varphi'(v)$, since the rest is a consequence of the fact that β is fully invariant. Let g be an arbitrary factorization of

$$\varphi'(v) = ss\ \varphi'(t)\varphi'(t)\ c_i s c_i\ \varphi'(t)\varphi'(t)\ ss$$

where $g = (g_0, a, g_1, b, g_2)$ with $a, b \in A$, $g_0, g_1, g_2 \in A^*$. We want to show the existence of a factorization $f = (f_0, a, f_1, b, f_2)$ of $\varphi'(u)$ such that $\mathsf{cont}(f_0) = \mathsf{cont}(g_0)$, $\mathsf{cont}(f_1) = \mathsf{cont}(g_1)$, $\mathsf{cont}(f_2) = \mathsf{cont}(g_2)$ and $f_0 a f_1 b f_2 = \varphi'(u)$. We distinguish several cases:

1a) "$\mathsf{cont}(g_0) \neq A$, $\mathsf{cont}(g_1) \neq A$"
Then $g_0 a g_1 b$ is a prefix of the prefix ss of the word $\varphi'(v)$, i.e. $ss = g_0 a g_1 b h$ for some $h \in A^*$. Hence $\mathsf{cont}(g_2) = A$, and we can put $f_0 = g_0$, $f_1 = g_1$, $f_2 = h\varphi'(t)\varphi'(t)c_i s c_i \varphi'(t)\varphi'(t)ss$.

1b) "$\mathsf{cont}(g_0) \neq A$, $\mathsf{cont}(g_1) = A$, $\mathsf{cont}(g_2) \neq A$"
Then g_0 is a prefix of the first s in $\varphi'(v)$ and g_2 is a suffix in the last s in $\varphi'(v)$. We can put $f_0 = g_0$, $f_2 = g_2$ and f_1 is an appropriate word.

1c) "$\mathsf{cont}(g_0) \neq A$, $\mathsf{cont}(g_1) = A$, $\mathsf{cont}(g_2) = A$"
Then g_0 is a prefix of the first s in $\varphi'(v)$, i.e. we put $f_0 = g_0$ and we can choose b from the last but one s from $\varphi'(u)$ and define f_1 and f_2 adequately.
Altogether we finished the case of $\mathsf{cont}(g_0) \neq A$.

2) Dually we can solve the cases of $\mathsf{cont}(g_2) \neq A$.

3) Assume $\mathsf{cont}(g_0) = \mathsf{cont}(g_2) = A$. And in addition we assume:

3a) "$\mathsf{cont}(g_1) = A$"
Then we can choose a from the second s in $\varphi'(u)$ and b from the last but one s in $\varphi'(u)$ and define f_0, f_1, f_2 in the expected way.

3b) "$\mathsf{cont}(g_1) \neq A$ and $\mathsf{cont}(ag_1b) \neq A$"
Then there is a word f_1 of length at most $m-1$ such that $\mathsf{cont}(f_1) = \mathsf{cont}(g_1)$ and the word af_1b is equal to some s_j. Hence we can find the word af_1b as a factor of the first occurrence $\varphi'(t)$ in $\varphi'(u)$ and then define f_0 and f_2.

3c) "$\mathsf{cont}(g_1) \neq A$ and $\mathsf{cont}(ag_1b) = A$, $c_i \in \mathsf{cont}(g_1)$"
Then we can find some s_j such that $w = c_i s_j c_i$ has the property $\mathsf{cont}(w) = \mathsf{cont}(g_1)$. Further ads_jdb is a factor of t, hence we can put $f_1 = w$ and af_1b is a factor of the first occurrence of $\varphi'(t)$ in $\varphi'(u)$. As usually, we denote f_0 and f_2 as needed.

3d) "$\mathsf{cont}(g_1) \neq A$ and $\mathsf{cont}(ag_1b) = A$ and $c_i \notin \mathsf{cont}(g_1)$"
Then $a = c_i$ or $b = c_i$.
If $a = b = c_i$ then we can find s_j such that $\mathsf{cont}(s_j) = \mathsf{cont}(g_1)$ and $c_i s_j c_i$ is a factor of the first occurrence of $\varphi'(t)$ in $\varphi'(u)$. Thus we consider the factorization f of u where f_1 is equal to this occurrence of s_j.
If $a = c_i$, $b \neq c_i$ then we can find f_1 such that f_1b is one of s_j with $\mathsf{cont}(f_1b) = \mathsf{cont}(g_1b)$ because $c_i \notin \mathsf{cont}(g_1b)$, i.e. $\mathsf{cont}(s_j) \neq A$. The case $a \neq c_i$, $b = c_i$ is dual.

II) Now assume that there is no such a letter. This means that $\varphi(c_i) = \varphi(c_{i'})$ for some different $i, i' \in \{1, \ldots, m\}$. Considerations are analogous to that of Case I). □

Remark. 1. If a positive variety of languages is locally finite we can generate the corresponding pseudovariety of ordered monoids by finitely generated free monoids. We are able to present effectively the free ordered monoids in pseudovarieties corresponding to $\mathsf{PPol}_k\mathcal{V}$ and $\mathsf{BPol}_k\mathcal{V}$ for $\mathcal{V}$ being any of $\mathcal{T}$, $\mathcal{S}^+$, $\mathcal{S}$, $\mathcal{A}_m$. It would be desirable to put a closer look into their structures.

2. For each positive variety of languages $\mathcal{V}$ the pseudovariety of ordered monoids corresponding to $\mathsf{PPol}_k\mathcal{V}$ is generated by the Schützenberger products of the form $\diamondsuit_{k+1}(M_0, \ldots, M_k)$ where $M_0, \ldots, M_k \in \mathbf{V}$ (see [9]). Notice that our Proposition 3 follows from results from [9].

References

1. Almeida, J.: Finite Semigroups and Universal Algebra. World Scientific, Singapore (1994)
2. Bloom, S.L.: Varieties of ordered algebras. J. Comput. Syst. Sci. 13(2), 200–212 (1976)

3. Burris, S., Sankappanavar, H.P.: A Course in Universal Algebra. Springer, Berlin (1981)
4. Blanchet-Sadri, F.: Equations and monoids varieties of dot-depth one and two. Theoret. Comput. Sci. 123, 239–258 (1994)
5. Eilenberg, S.: Automata, Languages and Machines, vol. B. Academic Press, New York (1976)
6. Klíma, O., Polák, L.: Hierarchies of piecewise testable languages. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 479–490. Springer, Heidelberg (2008)
7. Pin, J.-E.: A variety theorem without complementation. Russian Mathem (Iz. VUZ) 39, 74–83 (1995)
8. Pin, J.-E.: Syntactic semigroups. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, ch. 10. Springer, Heidelberg (1997)
9. Pin, J.-E.: Algebraic tools for the concatenation product. Theoretical Computer Science 292, 317–342 (2003)
10. Simon, I.: Piecewise testable events. In: ICALP 1975. LNCS, vol. 33, pp. 214–222. Springer, Heidelberg (1975)
11. Volkov, M.V.: Reflexive relations, extensive transformations and piecewise testable languages of a given height. Int. J. Algebra and Computation 14, 817–827 (2004)

Self-dual Codes over Small Prime Fields from Combinatorial Designs

Christos Koukouvinos and Dimitris E. Simos

Department of Mathematics, National Technical University of Athens, Zografou 15773, Athens, Greece
ckoukouv@math.ntua.gr, dsimos@math.ntua.gr

Abstract. In this paper, we give some new extremal ternary self-dual codes which are constructed by skew-Hadamard matrices. This has been achieved with the aid of a recently presented modification of a known construction method. In addition, we survey the known results for self-dual codes over $GF(5)$ constructed via combinatorial designs, i.e. Hadamard and skew-Hadamard matrices, and we give a new self-dual code of length 72 and dimension 36 whose minimum weight is 16 over $GF(5)$ for the first time. Furthermore, we give some properties of the generated self-dual codes interpreted in terms of algebraic coding theory, such as the orders of their automorphism groups and the corresponding weight enumerators.

Keywords: Self-dual codes, combinatorial designs, construction.

1 Introduction

A linear $[n, k]$ code C over $GF(p)$ is a k-dimensional vector subspace of $GF(p)^n$, where $GF(p)$ is the Galois field with p elements. In this paper, we consider the case where p is a prime. The elements of C are called codewords and the (Hamming) weight $wt(x)$ of a codeword x is the number of non-zero coordinates in x. The minimum weight of C is defined as $\min\{wt(x) \,|\, 0 \neq x \in C\}$. An $[n, k, d]$ code is an $[n, k]$ code with minimum weight d. A matrix whose rows generate the code C is called a generator matrix of C. The dual code $C^\perp$ of C is defined as $C^\perp = \{x \in \mathrm{GF}(p)^n |\ x \cdot y = 0 \text{ for all } y \in C\}$. C is *self-dual* if $C = C^\perp$. For $p \equiv 1 \pmod 4$, a self-dual $[n, n/2]$ code over $GF(p)$ exists if and only if n is even, and for $p \equiv 3 \pmod 4$, a self-dual $[n, n/2]$ code over $GF(p)$ exists if and only if $n \equiv 0 \pmod 4$ [29]. We say that self-dual codes with the largest minimum weight among self-dual codes of that length are *optimal*. Bounds on the minimum distance of linear codes can be found in [4] and [12].

One reason for the interest in self-dual codes is that they include some of the nicest and best-known error-correcting codes, and there are strong connections with other areas of combinatorics, group theory and lattices, while some of their applications can be found in communications, number and design theory [31]. By the Gleason-Pierce theorem [34], there are divisible self-dual codes over $GF(p)$ for $p = 2, 3$ and 4. Hence much work has been done concerning self-dual codes over these fields. For example, self-dual codes of small lengths over $GF(2)$,

$GF(3)$ and $GF(4)$ have been classified (cf. [33, Sections 11.3 – 11.6]), in order to determine which codes exist and which weight enumerators are possible. In addition, much is known about the largest minimum weights for self-dual codes over these fields (cf. [33, Tables X, XII, XIII and XIV]). Moreover t-designs are formed from extremal self-dual codes over $GF(2)$, $GF(3)$, or $GF(4)$ [32] using the Assmus-Mattson theorem [2]. Conversely, self-dual codes over larger fields have not been widely studied [33].

Now we consider the weight enumerators of self-dual codes over $\mathrm{GF}(p)$.

Theorem 1 (MacWilliams, Mallows and Sloane [30]). *The weight enumerator of a self-dual code over* $\mathrm{GF}(p)$ *is an element of*

$$\mathbb{C}[(x + (\sqrt{p} - 1)y)^2, y(x - y)].$$

Hence we have a trivial upper bound $d \le n/2 + 1$ which coincides with the Singleton bound for an $[n, n/2, d]$ code. However, the weight enumerator $W_p(n)$ of a self-dual $[n, n/2, n/2 + 1]$ code over $\mathrm{GF}(p)$ is uniquely determined.

In Section 2, we give some new extremal ternary self-dual codes which are constructed by skew-Hadamard matrices. Moreover, in Section 3, we survey the known results for self-dual codes over $GF(5)$ constructed via combinatorial designs, i.e. Hadamard and skew-Hadamard matrices, and we give a new self-dual code of length 72 and dimension 36 whose minimum weight is 16 over $GF(5)$ for the first time.

2 Ternary Self-dual Codes from Skew-Hadamard Matrices

A *Hadamard matrix* of order n is an $n \times n$ matrix with entries from $\{1, -1\}$ that satisfy $HH^T = nI_n$. It is well known that if n is the order of a Hadamard matrix then n is necessarily $1, 2$ or a multiple of 4. A Hadamard matrix is *normalized* if all entries in its first row and column are equal to 1. Two Hadamard matrices are *equivalent* if one can be transformed into the other by a series of row or column permutations and negations. A matrix H with entries from $\{1, -1\}$, for which

$$H = C + I_n \tag{1}$$

is said to be *skew-Hadamard matrix* of order n if $CC^T = (n-1)I_n$ and $C^T = -C$.

More details on the construction of Hadamard and skew-Hadamard matrices can be found in [10]. The comprehensive survey article [27] discusses the existence and the equivalence of skew-Hadamard matrices.

2.1 A Construction Method for Ternary Self-dual Codes

The following Theorem provides a general method for constructing self dual codes over $GF(p)$ taking into account the beautiful combinatorial structures that skew-Hadamard matrices possess and was given in [9]. Some other constructions for ternary self-dual codes using combinatorial designs are given in [1],[13] and [18].

Theorem 2 (Georgiou, Koukouvinos and Lappas [9]). *Let H be a skew-Hadamard matrix of order n and suppose that there exist three elements $a \neq 0, b, c$ from $GF(p)$ such that $a^2 + b^2 + (n-1)c^2 \equiv 0 \pmod{p}$. Then the matrix $G = [aI_n \; cC + bI_n]$ generates a self-dual code of length $2n$ and dimension n.*

We restate here, in the following Remark, a slight modification to the previous construction method given in [26], since it will be used throughout the paper. More details, regarding the following construction can be found in [26].

Remark 1 (Koukouvinos and Simos [26]). Let H be a skew-Hadamard matrix of order n and suppose that there exist elements $a, b, c \neq 0$ from $GF(p)$ such that $a^2 + (b-c)^2 + (n-1)c^2 \equiv 0 \pmod{p}$. Then the matrix $G = [aI_n \; cH - bI_n]$ generates a self-dual code of length $2n$ and dimension n.

A ternary self-dual code C which is *optimal* is called *extremal*, i.e. if it has the largest possible minimum weight. The known bounds of d for $p = 3$ are given in [33] and [35]. In particular the following theorem is known.

Theorem 3 (Tonchev [35]). *The minimum distance d of a ternary self-dual $[2n, n]$ code C satisfies*

$$d \leq 3\left[\frac{n}{6}\right] + 3.$$

where by $[x]$ we denote the nearest integer function of x.

2.2 Ternary Extremal Self-dual Codes

In this Section, we computed the minimum weight of the self-dual codes derived by Remark 1 for each possible solution of the diophantine equation,

$$a^2 + (b-c)^2 + (n-1)c^2 \equiv 0 \pmod{3}$$

when $a, b, c \neq 0$ over $GF(3)$. The diophantine equation has solutions for $n = 8, 12, 20, 24$. We present in the following Sections, extremal self-dual codes of lengths $2n = 16, 24, 40, 48$ derived from inequivalent skew-Hadamard matrices of orders $n = 8, 12, 20, 24$. We were motivated to perform a complete study for these orders of skew-Hadamard matrices since their respective number of inequivalent classes is completely determined for orders up to 28, and moreover since in [26] only self-dual codes over $GF(5)$ were given. Let N_n denote the number of inequivalent skew-Hadamard matrices for a given order n. We summarize the known results for N_n in the table below, taken from [27].

Table 1. Inequivalent skew-Hadamard matrices for orders 4 to 28

n	4	8	12	16	20	24	28
N_n	1	1	1	2	1	16	54

In Table 1., we denote by n the order of the skew-Hadamard matrix and by N_n the number of known inequivalent skew-Hadamard matrices of order n.

In the results that follow, we give for each inequivalent skew-Hadamard matrix only the inequivalent self-dual codes produced, the order of the automorphism group of these codes and their respective weight enumerators. We remind that, two linear codes C_1 and C_2 over $GF(p)$ are monomially equivalent if there is a monomial matrix M over $GF(p)$ such that $C_2 = C_1 M = \{cM \mid c \in C_1\}$. A monomial matrix over $GF(p)$ which maps C to itself is called an automorphism of C. The set of all automorphisms of C is called the automorphism group $Aut(C)$ of C. For a self-dual code derived from the i-th inequivalent skew-Hadamard matrix of order n we shall use the notation $C_{n,i}$.

[16, 8] Ternary Self-dual Codes. In this Section, we study self-dual codes over $GF(3)$, which arise from the unique skew-Hadamard matrix of order 8. The unique skew-Hadamard matrix (up to equivalence) of order 8 is

$$H_8 = C + I_8 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 \\ -1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 \end{pmatrix}$$

The results obtained by using Remark 1 are presented in the following Table.

Table 2. [16,8] self-dual code from the skew-Hadamard matrix of order 8

C	a	b	c	d	$\lvert Aut(C)\rvert$	$W(x,y)$
$C_{8,1}$	1	2	1	6	$43008 = 2^{11} \cdot 3 \cdot 7$	$x^{16} + 224x^{10}y^6 + 2720x^7y^9 + 3360x^4y^{12} + 256xy^{15}$

The code $C_{8,1}$ is extremal since the bound for $n = 8$ from Theorem 3 is 6.

[24, 12] Ternary Self-dual Codes. In this Section, we study self-dual codes over $GF(3)$, which arise from the unique skew-Hadamard matrix of order 12. The unique skew-Hadamard matrix (up to equivalence) of order 12 is

$$H_{12} = C + I_{12} = \begin{pmatrix} 1 & 1 & -1 & 1 & 1 & 1 & 1 & -1 & 1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 & -1 \\ 1 & -1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 & 1 & -1 & 1 \\ -1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & -1 & -1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & 1 \end{pmatrix}$$

The results obtained by using Remark 1 are presented in the following Table.

The code $C_{12,1}$ is extremal since the bound for $n = 12$ from Theorem 3 is 9.

Table 3. [24,12] self-dual code from the skew-Hadamard matrix of order 12

C	a	b	c	d	$\|Aut(C)\|$	$W(x,y)$
$C_{12,1}$	1	1	1	9	$5280 = 2^5 \cdot 3 \cdot 5 \cdot 11$	$x^{24} + 4048x^{15}y^9 + 61824x^{12}y^{12} + 242880x^9y^{15} +198352x^6y^{18} + 24288x^3y^{21} + 48y^{24}$

$[40, 20]$ Ternary Self-dual Codes. In this Section, we study self-dual codes over $GF(3)$, which arise from the unique skew-Hadamard matrix of order 20. The skew-Hadamard matrix we used is

$$H_{20} = C + I_{20} = \left(\begin{array}{cccccccccccccccccccc}
+&-&+&-&+&+&-&-&+&+&-&-&-&-&+&-&-&-&-&+\\
+&+&-&+&-&-&-&+&+&+&-&-&-&+&-&-&-&-&+&-\\
-&+&+&-&+&-&+&+&+&-&-&-&+&-&-&-&-&+&-&-\\
+&-&+&+&-&+&+&+&-&-&-&+&-&-&-&-&+&-&-&-\\
-&+&-&+&+&+&+&-&-&+&+&-&-&-&-&+&-&-&-&-\\
-&+&+&-&-&+&-&+&-&+&-&-&-&-&+&+&+&+&+&-\\
+&+&-&-&-&+&+&-&+&-&-&-&-&+&-&+&+&+&-&+\\
+&-&-&-&+&-&+&+&-&+&-&-&+&-&-&+&+&-&+&+\\
-&-&-&+&+&+&-&+&+&-&-&+&-&-&-&+&-&+&+&+\\
-&-&+&+&-&-&+&-&+&+&+&-&-&-&-&-&+&+&+&+\\
+&+&+&+&-&+&+&+&+&-&+&-&+&-&+&+&-&-&+&+\\
+&+&+&-&+&+&+&+&-&+&+&+&-&+&-&-&-&+&+&+\\
+&+&-&+&+&+&+&-&+&+&-&+&+&-&+&-&+&+&+&-\\
+&-&+&+&+&+&-&+&+&+&+&-&+&+&-&+&+&+&-&-\\
-&+&+&+&+&-&+&+&+&+&-&+&-&+&+&+&+&-&-&+\\
+&+&+&+&-&-&-&-&-&+&-&+&+&-&-&+&-&+&-&+\\
+&+&+&-&+&-&-&-&+&-&+&+&-&-&-&+&+&-&+&-\\
+&+&-&+&+&-&-&+&-&-&+&-&-&-&+&-&+&+&-&+\\
+&-&+&+&+&-&+&-&-&-&-&-&-&+&+&+&-&+&+&-\\
-&+&+&+&+&+&-&-&-&-&-&-&+&+&-&-&+&-&+&+
\end{array}\right)$$

The results obtained by using Remark 1 are presented in the following Table.

Table 4. [40,20] self-dual code from the skew-Hadamard matrix of order 12

C	a	b	c	d	$\|Aut(C)\|$	$W(x,y)$
$C_{20,1}$	1	2	1	12	$13680 = 2^4 \cdot 3^2 \cdot 5 \cdot 19$	$x^{40} + 19760x^{28}y^{12} + 1138176x^{25}y^{15}+ 25549680x^{22}y^{18} + 236945280x^{19}y^{21} + 907161840x^{16}y^{24}+ 1389711680x^{13}y^{27} + 783017664x^{10}y^{30} + 137826000x^7y^{33}+ 5394480x^4y^{36} + 19840xy^{39}$

The code $C_{20,1}$ is extremal since the bound for $n = 20$ from Theorem 3 is 12.

$[48, 24]$ Ternary Self-dual Codes. In this Section, we study self-dual codes over $GF(3)$, which arise from the sixteen inequivalent skew-Hadamard matrices of order 24. The results obtained by using Remark 1 are presented in the following Table. The skew-Hadamard matrices we have used can be retrieved from [25]. We note that in this case, we list only the order of the automorphism groups of the derived self-dual codes, and not the respective weight enumerators due to a computational complexity limit.

The code $C_{24,14}$ is extremal since the bound for $n = 24$ from Theorem 3 is 15.

Table 5. [48,24] self-dual codes from the skew-Hadamard matrices of order 24

C	a	b	c	d	$\|Aut(C)\|$	C	a	b	c	d	$\|Aut(C)\|$
$C_{24,1}$	1	1	1	12	$48 = 2^4 \cdot 3$	$C_{24,9}$	1	1	1	12	$48 = 2^4 \cdot 3$
$C_{24,2}$	1	1	1	12	$24 = 2^3 \cdot 3$	$C_{24,10}$	1	1	1	12	$96 = 2^5 \cdot 3$
$C_{24,3}$	1	1	1	12	$48 = 2^4 \cdot 3$	$C_{24,11}$	1	1	1	12	$96 = 2^5 \cdot 3$
$C_{24,4}$	1	1	1	12	$80 = 2^4 \cdot 5$	$C_{24,12}$	1	1	1	12	$10560 = 2^6 \cdot 3 \cdot 5 \cdot 11$
$C_{24,5}$	1	1	1	12	$32 = 2^5$	$C_{24,13}$	1	1	1	12	$10560 = 2^6 \cdot 3 \cdot 5 \cdot 11$
$C_{24,6}$	1	1	1	12	$32 = 2^5$	$C_{24,14}$	1	1	1	15	$48576 = 2^6 \cdot 3 \cdot 11 \cdot 23$
$C_{24,7}$	1	1	1	12	$32 = 2^5$	$C_{24,15}$	1	1	1	12	$80 = 2^4 \cdot 5$
$C_{24,8}$	1	1	1	12	$48 = 2^4 \cdot 3$	$C_{24,16}$	1	1	1	12	$24 = 2^3 \cdot 3$

3 Self-dual Codes over $GF(5)$ from Combinatorial Designs

For $GF(5)$, only self-dual codes up to length 12, and lengths 14 and 16 have been classified respectively, in [28] and [20]. The largest minimum weights of self-dual codes over $GF(5)$ up to length 24 have been determined in [5]. Tables with the highest minimum distance known for self-dual codes over $GF(5)$ for lengths up to 64 and 70, are given in (cf. [6, Table V]) and (cf. [8, Tables 9, 10]), respectively. For online Tables with constructions and the highest minimum distance known for self-dual codes over $GF(5)$ for lengths up to 70, see [7]. Constructions of self-dual codes over $GF(5)$, can be found in [1],[6],[8],[14],[15],[21],[24],[28].

Recently, some authors (for instance [21] and [24]) have improved the lower and upper bounds of the minimum distance of self-dual codes over $GF(5)$ for lengths from 26 up to 40 and 34, respectively. A method for constructing self-dual codes over $GF(5)$ from skew-Hadamard designs for lengths from 20 up to 60 has appeared in [22]. In [15], codes over $GF(5)$ with parameters $[36, 18, 12]$, $[48, 24, 15]$, $[60, 30, 18]$, $[64, 32, 18]$ and $[76, 38, 21]$ which improve the previously known bounds on the minimum weight for linear codes over $GF(5)$ were constructed from conference matrices. In the same paper, the authors noted that it seems infeasible to determine the minimum weight for the next case of their method, i.e. length 84. In a recent paper [26] the authors gave a slight modification to a general method for constructing self-dual codes over $GF(5)$ using skew-Hadamard matrices. This modification gave optimal self-dual codes for lengths up to 56. In particular, new inequivalent $[48, 24]$ and $[56, 28]$ self-dual codes over $GF(5)$ whose minimum weights are 14 and 16, respectively, were constructed by using skew-Hadamard matrices of order 24 and 28. These results, improved the only known quadratic double circulant self-dual codes of lengths 48 and 56. Moreover, they constructed $[80, 40]$ and $[88, 44]$ self-dual codes whose minimum weights are 17 and 19 over $GF(5)$. These codes were derived from skew-Hadamard matrices of order 40 and 44, respectively.

3.1 A $[72, 36]$ Self-dual Code over $GF(5)$

We used one of the eighteen inequivalent skew-Hadamard matrices of order 36 given in [25] to form a generator matrix G of the form $G = [I_{36}\ 3H - I_{36}]$ for $a = b = 1$ and $c = 3$ and $p = 5$ in Remark 1. The matrix G generates a self-dual code of length 72 and dimension 36. We give below the rows of the submatrix $3H - I_{36}$ of the generator matrix G of a $[72, 36]$ self-dual code over $GF(5)$.

```
222232333333233322223222223232223233
322223233332333223232222232322232332
332222323323332233322222322222323323
333222232233322333222223223223233232
233322223333223332222232232232332322
323332222332233323222322322323323222
232333222322333233223223222233232223
223233322223332333232232222332322232
222323332233323332322322222323222323
222322233222232333323222323333332332
223222332322223233323222323333323323
232223322332222323322223233233233233
322233222333222232222323323332332333
222332223233322223223233232323323333
223322232323332222232332322233233333
233222322232333222323323222332333332
332223222223233322233232223323333323
322232223222323332332322232233333233
332333332232333232222323332333323332
323333323323332322322223233333233322
233333233233323223332222323332333223
333332332333232232333222232323332233
333323323332322323233322223233322333
333233233323223233323332222333223332
332332333232232333232333222332233323
323323333322323332223233322322333233
233233333223233323222323332223332333
323332322222223223322232223222232333
233323223222232232222322233322223233
333232232222322322223222332332222323
332322323223223222232223322332222232
323223233232232222322233222233322223
232223233322322222222233222332333222
322323332223222223223322232232333222
223233323232222232233222322223233322
232333232322222322332223222222323332
```

Computation of minimum weight. We have used Magma, a computer algebra system for symbolic computation developed at the University of Sydney, to compute the minimum weight of the previously constructed $[72, 36]$ self-dual code [3],[11]. We give below the details of the last phase of our computation for length 72.

```
Linear Code over GF(5) of length 72 with 36 generators.
Enumerating using 8 generators at a time:
Completed Matrix  1:
lower = 16, upper = 16.
Computation complete
72574065912 vectors enumerated
in total (0.000000% of 72 36 code)
Final Results: lower = 16, upper = 16
IsSelfDual: True
```

Theorem 4. *There exists a* $[72, 36, 16]$ *self-dual code over* $GF(5)$*.*

3.2 Optimal Minimum Distances of Self-dual Codes over $GF(5)$

In this Section, we give an updated Table with the best up-to-date optimal minimum distances of self-dual codes over $GF(5)$ which summarizes the survey

Table 6. Optimal minimum distances of self-dual codes over $GF(5)$

Length	d	N	Reference	Length	d	N	Reference
2	2	1	[28]	28	$10-11$	≥ 20	[8],[16],[21]
4	2	1	[28]	30	$10-12$	≥ 204	[8],[16]
6	4	1	[28]	32	$11-12$	≥ 1	[8],[16],[21]
8	4	1	[28]	34	$11-12$	≥ 11	[8],[16],[21]
10	4	3	[28]	36	$12-13$	≥ 1	[8],[21]
12	6	1	[28]	38	$12-14$	≥ 1	[8],[21]
14	6	3	[20],[28]	40	$13-15$	≥ 1	[8],[21],[26]
16	7	1	[20],[28]	48	$14-20$	≥ 2	[8],[26]
18	7	9	[20],[21]	56	$16-23$	≥ 2	[8],[26]
20	8	≥ 8	[21],[28]	72	$16-?$	≥ 1	Section 3.1
22	8	≥ 59	[21]	80	$17-?$	≥ 1	[26]
24	9	≥ 2	[14],[17],[19],[26],[28]	88	$19-?$	≥ 1	[26]
26	$9-10$	≥ 1	[8],[16],[21]				

and the results given previously. The first and fifth columns give code lengths, the second and sixth columns give the optimal minimum distances for self-dual codes over $GF(5)$, and the third and seventh columns give the number of inequivalent optimal self-dual codes.

Acknowledgments. This research was financially supported from the General Secretariat of Research and Technology by a grant PENED 03ED740. The research of the second author was financially supported by a scholarship awarded by the Secretariat of the Research Committee of National Technical University of Athens.

References

1. Arasu, K.T., Gulliver, T.A.: Self-dual codes over $\mathbb{F}_p$ and weighing matrices. IEEE Trans. Inform. Theory 47, 2051–2055 (2001)
2. Assmus Jr., E.F., Mattson Jr., H.F.: New 5-designs. J. Combin. Theory Ser. A 6, 122–151 (1969)
3. Bosma, W., Cannon, J.: Handbook of Magma Functions. Version 2.9, Sydney (2002)
4. Brouwer, A.E.: Bounds on linear codes. In: Pless, V., Huffman, W.C. (eds.) Handbook of Coding Theory, pp. 295–461. Elsevier, Amsterdam (1998)
5. Dougherty, S.T., Gulliver, T.A., Harada, M.: Optimal formally self-dual codes over $\mathbb{F}_5$ and $\mathbb{F}_7$. Appl. Algebra Engrg. Comm. Comput. 10, 227–236 (2000)
6. Gaborit, P.: Quadratic double circulant codes over fields. J. Combin. Theory Ser. A 97, 85–107 (2002)
7. Gaborit, P.: Tables of self-dual codes, `http://www.unilim.fr/pages_perso/philippe.gaborit/SD`
8. Gaborit, P., Otmani, A.: Experimental constructions of self-dual codes. Finite Fields Appl. 9, 372–394 (2003)

9. Georgiou, S., Koukouvinos, C., Lappas, E.: Self-dual codes over some prime fields constructed from skew-Hadamard matrices. J. Discrete Math. Sci. Cryptogr. 10, 255–266 (2007)
10. Geramita, A.V., Seberry, J.: Orthogonal Designs. Quadratic Forms and Hadamard Matrices. Lecture Notes in Pure and Applied Mathematics, 45. Marcel Dekker, Inc., New York (1979)
11. Grassl, M.: Searching for linear codes with large minimum distance. In: Bosma, W., Cannon, J. (eds.) Discovering Mathematics with Magma. Springer, Heidelberg (2006)
12. Grassl, M.: Bounds on the minimum distance of linear codes, `http://www.codetables.de`
13. Gulliver, T.A., Harada, M.: New optimal self-dual codes over $GF(7)$. Graphs Combin. 15, 175–186 (1999)
14. Gulliver, T.A., Harada, M.: Double circulant self-dual codes over $GF(5)$. Ars Comb. 56, 3–13 (2000)
15. Gulliver, T.A., Harada, M.: On the minimum weight of codes over $\mathbb{F}_5$ constructed from certain conference matrices. Des. Codes Cryptogr. 31, 139–145 (2004)
16. Gulliver, T.A., Harada, M., Miyabayashi, H.: Double circulant self-dual codes over $\mathbb{F}_5$ and $\mathbb{F}_7$. Adv. Math. Commun. 1, 223–238 (2007)
17. Han, S., Kim, J.-L., Lee, H., Lee, Y.: Cubic self-dual codes based on building-up constructions (preprint)
18. Harada, M.: New extremal ternary self–dual codes. Austral. J. Combin. 17, 133–145 (1998)
19. Harada, M., Munemasa, A.: There exists no self-dual [24,12,10] code over $\mathbb{F}_5$. Des. Codes Cryptogr. 52, 125–127 (2009)
20. Harada, M., Östergård, P.R.J.: On the classification of self-dual codes over $\mathbb{F}_5$. Graphs Combin 19, 203–214 (2003)
21. Kim, J.-L., Han, S.: On self-dual codes over $\mathbb{F}_5$. Des. Codes Cryptogr. 48, 43–58 (2008)
22. Kim, J.-L., Sole, P.: Skew Hadamard designs and their codes. Des. Codes Cryptogr. 49, 135–145 (2008)
23. Kotsireas, I.S., Koukouvinos, C.: New skew-Hadamard matrices via computational algebra. Australas. J. Combin. 41, 235–248 (2008)
24. Kotsireas, I.S., Koukouvinos, C., Simos, D.E.: MDS and near-MDS self-dual codes over large prime fields (submitted for publication)
25. Koukouvinos, C.: Hadamard matrices of order $4t$, t odd positive integer, `http://www.math.ntua.gr/~ckoukouv`
26. Koukouvinos, C., Simos, D.E.: Construction of new self-dual codes over GF(5) using skew-Hadamard matrices (to appear in Adv. Math. Commun.)
27. Koukouvinos, C., Stylianou, S.: On skew-Hadamard matrices. Discrete Math. 308, 2723–2731 (2008)
28. Leon, J.S., Pless, V., Sloane, N.J.A.: Self-dual codes over $GF(5)$. J. Combin. Theory Ser. A 32, 178–194 (1982)
29. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error-Correcting Codes. North-Holland, Amsterdam (1977)
30. MacWilliams, F.J., Mallows, C.L., Sloane, N.J.A.: Generalizations of Gleason's theorem on weight enumerators of self-dual codes. IEEE Trans. Inform. Theory 18, 794–805 (1972)
31. Nebe, G., Rains, E.M., Sloane, N.J.A.: Self-dual codes and invariant theory. Springer, Heidelberg (2006)

32. Pless, V.S., Huffman, W.C., Brualdi, R.A.: An introduction to algebraic codes. In: Pless, V., Huffman, W.C. (eds.) Handbook of Coding Theory, pp. 295–461. Elsevier, Amsterdam (1998)
33. Rains, E.M., Sloane, N.J.A.: Self-dual codes. In: Pless, V., Huffman, W.C. (eds.) Handbook of Coding Theory, pp. 177–294. Elsevier, Amsterdam (1998)
34. Sloane, N.J.A.: Self-dual codes and lattices, Relations between combinatorics and other parts of mathematics. In: Proceedings of Symposium on pure mathematics, pp. 273–308. American Mathematics Society, Providence (1979)
35. Tonchev, V.D.: Codes. In: Colbourn, C.J., Dinitz, J.H. (eds.) The CRC Handbook of Combinatorial Designs, pp. 517–543. CRC Press, Boca Raton (1996)

A Backward and a Forward Simulation for Weighted Tree Automata[⋆]

Andreas Maletti

Universitat Rovira i Virgili
Departament de Filologies Romàniques
Avinguda de Catalunya 35, 43002 Tarragona, Spain
andreas.maletti@urv.cat

Abstract. Two types of simulations for weighted tree automata (wta) are considered. Wta process trees and assign a weight to each of them. The weights are taken from a semiring. The two types of simulations work for wta over additively idempotent, commutative semirings and can be used to reduce the size of wta while preserving their semantics. Such reductions are an important tool in automata toolkits.

1 Introduction

Automata minimization is an important and well-studied subject. Here we consider (finite-state) tree automata and weighted tree automata, which are used in applications such as model checking [1] and natural language processing [2]. Deterministic (bottom-up) tree automata can be minimized efficiently using, for example, an algorithm inspired by Hopcroft [3,4]. However, minimizing nondeterministic tree automata is PSPACE-complete [5] and cannot be approximated well [6,7,8] unless P = PSPACE. Consequently, alternative (efficient) methods to reduce the size of tree automata were explored [4,9,10,11]. An efficient minimization procedure for deterministic (bottom-up) weighted tree automata is presented in [12] and efficient reductions of nondeterministic weighted tree automata with the help of bisimulation relations are considered in [13].

Here we consider the simulation approach of [10] for weighted tree automata over additively idempotent, commutative semirings. A weighted tree automaton essentially is a tree automaton in which each transition carries a weight (an element of a semiring). Instead of accepting a certain set of trees, a weighted tree automaton assigns a weight to each tree. First, the automaton assigns a weight to each run, which is the same as a run of the corresponding unweighted automaton. The weight of the run is obtained by multiplying (in the semiring) the participating transition weights (each transition weight as often as it occurs in the run) and eventually the final weight associated to the state reached at the root. Should there be several runs on the same input tree, then the weights of those runs are summed up to obtain the weight assigned to this input tree.

⋆ This work was financially supported by the *Ministerio de Educación y Ciencia* (MEC) grant JDCI-2007-760.

In [10] two types of simulation relations, called downward and upward simulations, are examined for tree automata. Roughly speaking, we generalize these notions to the setting of weighted tree automata. While there are several potential generalizations, our approach requires us to consider ordered semirings. Here we choose to work with additively idempotent (i.e., $a + a = a$ for all semiring elements a) semirings and their natural order. We define two types of simulation relations: backward and forward simulation. Intuitively, these notions correspond to backward and forward bisimulation of [13], but are unfortunately not generalizations of those concepts. Backward simulation generalizes downward simulation of [10] and our forward simulation generalizes upward simulation with respect to the identity as downward simulation [10]. We choose not to generalize upward simulations [10] with respect to arbitrary downward simulations since we believe that two completely separate notions are easier to handle and understand.

A simulation is a quasi-order (i.e., a reflexive and transitive relation) on the states of an input automaton M. A backward simulation is such that larger states dominate the smaller states; i.e., if the smaller state accepts a tree with weight a, then the larger state accepts the same tree with a weight that is larger than a (see Lemma 3). We take the equivalence induced by this quasi-order (i.e., two states are equivalent if they simulate each other) and reduce M with it (see Definition 6). This construction is simple for tree automata, however our reductions need to address the weights. This yields separate constructions for the backward (see Definition 6) and forward (see Definition 13) case. We show in Theorem 7 that the weighted tree automaton obtained with the help of a backward simulation, which never has more states than M, is equivalent to M.

In a forward simulation we do not consider the trees that a state can accept, but rather the contexts (i.e., trees over the input ranked alphabet with a unique occurrence of the extra symbol $\square$) that can be processed starting from that state. For those contexts a similar domination property as in the backward case must hold (see Lemma 12). Again, we use the induced equivalence to reduce the automaton. Theorem 15 shows that we obtain an equivalent weighted tree automaton.

Both types of simulations admit a greatest simulation that can be used for greatest gain in reduction (see Theorems 2 and 11). For deterministic weighted tree automata, we show that backward simulation is ineffective and forward simulation is only as effective as forward bisimulation [13]. This essentially means that our new tools do not surpass the existing tools in the deterministic case, but they can yield much greater reductions in the nondeterministic case. In summary, we add two more tools to the toolbox, which can be used to reduce nondeterministic weighted tree automata.

2 Preliminaries

We denote the nonnegative integers, which include 0, by $\mathbb{N}$. For every $l, u \in \mathbb{N}$, the subset $\{n \in \mathbb{N} \mid l \leqslant n \leqslant u\}$ is simply written as $[l, u]$. An alphabet is a nonempty and finite set. Its elements are called symbols. A ranked alphabet (Σ, rk) consists of an alphabet Σ and a mapping $\mathrm{rk}\colon \Sigma \to \mathbb{N}$, which associates to each symbol a rank. The set $\Sigma_k = \{\sigma \in \Sigma \mid \mathrm{rk}(\sigma) = k\}$ contains the symbols of rank k. Henceforth, we will denote such a ranked alphabet by Σ alone and assume that the mapping rk is implicit. For a ranked alphabet Σ and a set T, we write $\Sigma(T)$ for $\{\sigma(t_1, \ldots, t_k) \mid \sigma \in \Sigma_k, t_1, \ldots, t_k \in T\}$. We generally write α instead of $\alpha()$ for $\alpha \in \Sigma_0$. The set $T_\Sigma(V)$ of Σ-trees indexed by a set V is the smallest set such that $V \subseteq T_\Sigma(V)$ and $\Sigma(T_\Sigma(V)) \subseteq T_\Sigma(V)$. We just write T_Σ for $T_\Sigma(\emptyset)$.

A relation ϱ on a set S is a subset of $S \times S$. The inverse ϱ^{-1} is the relation $\{(s', s) \mid s \mathrel{\varrho} s'\}$ and the composition of two relations ϱ_1 and ϱ_2 on S is

$$\varrho_1 \mathbin{;} \varrho_2 = \{(s, s'') \mid \exists s' \in S \colon s \mathrel{\varrho_1} s' \mathrel{\varrho_2} s''\} \enspace .$$

A quasi-order $\preceq$ on S is a reflexive, transitive relation on S. An up-set $A \subseteq S$ (with respect to $\preceq$) is such that for every $s \preceq s'$ with $s \in A$ also $s' \in A$. The smallest up-set containing $A \subseteq S$ is denoted by $\uparrow(A)$. If $A = \{s\}$, then we simply write $\uparrow(s)$. The quasi-order $\preceq$ is an equivalence relation if it is symmetric, and it is a partial order if it is anti-symmetric. A partial order $\leqslant$ on S is total if $s \leqslant s'$ or $s' \leqslant s$ for every $s, s' \in S$. Let $\equiv$ be an equivalence on S. We write $[s]_\equiv$ for the equivalence class of $s \in S$ and $(S/{\equiv})$ for the partition $\{[s]_\equiv \mid s \in S\}$. Whenever possible without confusion, we drop $\equiv$ from $[s]_\equiv$. Note that if $\preceq$ is a quasi-order on S, then $\simeq \; = \; \preceq \cap \preceq^{-1}$ is an equivalence relation on S and $\preceq$ induces a partial order on $S/{\simeq}$.

A commutative semiring is an algebraic structure $\mathcal{A} = (A, +, \cdot, 0, 1)$ comprising two commutative monoids $(A, +, 0)$ and $(A, \cdot, 1)$ such that $\cdot$ distributes over $+$ and 0 is absorbing for $\cdot$ (i.e., $0 \cdot a = 0$ for every $a \in A$). It is (additively) idempotent if $1 + 1 = 1$. Moreover, let $\leqslant$ be a partial order on A. It partially orders $\mathcal{A}$ if $a_1 + b_1 \leqslant a_2 + b_2$ and $a_1 \cdot b_1 \leqslant a_2 \cdot b_2$ for every $a_1 \leqslant a_2$ and $b_1 \leqslant b_2$. Let $\sqsubseteq$ be the quasi-order on A such that $a \sqsubseteq b$ if there exists $c \in A$ with $a + c = b$. Whenever $\sqsubseteq$ is anti-symmetric, it is called the natural order. Note that for an idempotent semiring, the relation $\sqsubseteq$ is always a partial order. Morever, the natural order always (independent of idempotency) partially orders $\mathcal{A}$.

A tree series (over Σ and A) is a mapping $\varphi \colon T_\Sigma \to A$. The set of all such tree series is $A\langle\langle T_\Sigma \rangle\rangle$. We write (ψ, t) instead of $\psi(t)$ for every $t \in T_\Sigma$. A weighted tree automaton (wta) [14,15,16] is a tuple $M = (Q, \Sigma, \mathcal{A}, \mu, F)$ such that

- Q is a finite set of states,
- Σ is a ranked alphabet of input symbols,
- $\mathcal{A} = (A, +, \cdot, 0, 1)$ is a semiring,
- $\mu = (\mu_k)_{k \in \mathbb{N}}$ is such that $\mu_k \colon \Sigma_k \to A^{Q \times Q^k}$, and
- $F \colon Q \to A$ is a final weight assignment.

The wta is deterministic if for every $\sigma \in \Sigma_k$ and $q_1, \ldots, q_k \in Q$ there exists at most one $q \in Q$ such that $\mu_k(\sigma)_{q, q_1, \ldots, q_k} \neq 0$. A wta computes a tree series as follows. Let $h_\mu \colon T_\Sigma(Q) \to A^Q$ be the mapping such that

- for every $p, q \in Q$

$$h_\mu(p)_q = \begin{cases} 1 & \text{if } p = q \\ 0 & \text{otherwise} \end{cases}$$

– for every $\sigma \in \Sigma_k$, $t_1, \ldots, t_k \in T_\Sigma$, and $q \in Q$

$$h_\mu(\sigma(t_1, \ldots, t_k))_q = \sum_{q_1,\ldots,q_k \in Q} \mu_k(\sigma)_{q,q_1,\ldots,q_k} \cdot \prod_{i=1}^{k} h_\mu(t_i)_{q_i} \enspace .$$

The wta M recognizes the tree series $\varphi_M \in A\langle\langle T_\Sigma\rangle\rangle$, which is defined for every $t \in T_\Sigma$ by $(\varphi_M, t) = \sum_{q \in Q} F(q) \cdot h_\mu(t)_q$. Two wta are equivalent if they recognize the same tree series.

3 A Backward Simulation

In this section, we investigate backward simulation for wta [14,15,16]. Such simulations for unweighted tree automata were already considered in [10] and backward bisimulations, which are a related concept, for wta were considered in [13]. To avoid a very detailed discussion, we restrict ourselves to idempotent and commutative semirings and their natural order. With minor modifications, our arguments also work for other idempotent (even non-commutative) semirings that are partially ordered. In the following, we fix an idempotent semiring $\mathcal{A} = (A, +, \cdot, 0, 1)$ and its natural order $\sqsubseteq$. In addition, let $M = (Q, \Sigma, \mathcal{A}, \mu, F)$ be a wta, and without loss of generality, suppose that Q is totally ordered. We will use $\min(P)$ with $P \subseteq Q$ for the minimal state of P with respect to that total order.

Let us start with the definition of a backward simulation. Note that our definition yields the definition of [10] when considered in the unweighted case. In that case, if a state q simulates a state p and there exists a transition $\sigma(p_1, \ldots, p_k) \to p$, then there also exists a transition $\sigma(q_1, \ldots, q_k) \to q$ such that the q_i simulate the corresponding p_i. Now, let us consider the weighted setting. In essence, for a state q to simulate a state p, written $p \preceq q$, we demand that for every transition weight $\mu_k(\sigma)_{p,p_1,\ldots,p_k}$ there exists a larger (with respect to the natural order $\sqsubseteq$) transition weight $\mu_k(\sigma)_{q,q_1,\ldots,q_k}$ such that, for every $i \in [1, k]$, the state q_i simulates p_i. Note that there is no condition on the final weights.

Definition 1 (cf. [10, Section 2]). *A quasi-order $\preceq$ on Q is a* backward simulation *for M if for every $p \preceq q$, $\sigma \in \Sigma_k$, and $p_1, \ldots, p_k \in Q$ there exist $q_1, \ldots, q_k \in Q$ such that $\mu_k(\sigma)_{p,p_1,\ldots,p_k} \sqsubseteq \mu_k(\sigma)_{q,q_1,\ldots,q_k}$ and $p_i \preceq q_i$ for every $i \in [1, k]$.*

Let us discuss the definition. We already remarked that it coincides with the definition of a backward simulation [10] in the unweighted case [i.e., the case where $\mathcal{A} = (\{\bot, \top\}, \vee, \wedge, \bot, \top)$ is the Boolean semiring]. However, the definition does not generalize the notion of backward bisimulation for wta of [13]. Next, let us establish some central properties of backward simulations. First, there is a greatest backward simulation for M. We prove this along the lines of [13, Theorem 22].

Theorem 2. *There exists a greatest (with respect to $\subseteq$) backward simulation for M.*

Proof. Let $\preceq$ and $\preceq'$ be backward simulations for M. We claim that $(\preceq \cup \preceq')^*$, the reflexive and transitive closure of $\preceq \cup \preceq'$, is again a backward simulation. Clearly, $(\preceq \cup \preceq')^*$ is a quasi-order. Now, let $(p,q) \in (\preceq \cup \preceq')^*$, $\sigma \in \Sigma$, and $p_1, \ldots, p_k \in Q$. Consequently, there exist $r_1, \ldots, r_n \in Q$ such that

$$p = r_0 \preceq r_1 \preceq' r_2 \preceq r_3 \preceq' \cdots \preceq' r_n = q \enspace .$$

By this chain of inequalities, there also exist $q_1, \ldots, q_k \in Q$ such that

$$\mu_k(\sigma)_{p,p_1,\ldots,p_k} \sqsubseteq \mu_k(\sigma)_{q,q_1,\ldots,q_k} \quad \text{and} \quad p_i \; (\preceq ; \preceq' ; \preceq ; \preceq' ; \cdots ; \preceq') \; q_i$$

for every $i \in [1,k]$, which proves that $(\preceq \cup \preceq')^*$ is a backward simulation. □

The main property of a state q that simulates a state p is that the state q accepts every input tree with a weight that is larger than the weight with which the same tree is accepted by p. In the unweighted case, this corresponds to the statement that the tree language accepted by p is a subset of the tree language accepted by q (see [10, Section 6.1]). In general, this immediately yields that any two states equivalent in $\preceq \cap \preceq^{-1}$, which is always an equivalence relation since $\preceq$ is a quasi-order, accept the same tree series.

Lemma 3. *Let $\preceq$ be a backward simulation for M. Then $h_\mu(t)_p \sqsubseteq h_\mu(t)_q$ for every $t \in T_\Sigma$ and $p \preceq q$.*

Proof. Let $t = \sigma(t_1, \ldots, t_k)$ for some $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$. We compute as follows:

$$h_\mu(\sigma(t_1, \ldots, t_k))_p = \sum_{p_1,\ldots,p_k \in Q} \mu_k(\sigma)_{p,p_1,\ldots,p_k} \cdot \prod_{i=1}^{k} h_\mu(t_i)_{p_i} \enspace .$$

For all $p_1, \ldots, p_k \in Q$ there exist $(q_1, \ldots, q_k) \in Q^k$ such that $p_i \preceq q_i$ for every $i \in [1,k]$ and $\mu_k(\sigma)_{p,p_1,\ldots,p_k} \sqsubseteq \mu_k(\sigma)_{q,q_1,\ldots,q_k}$ because $p \preceq q$. Denote $(q_1, \ldots, q_k)$ by $f(p_1, \ldots, p_k)$ and q_i by $f(p_1, \ldots, p_k)_i$ for every $i \in [1,k]$. Then we continue with

$$h_\mu(\sigma(t_1, \ldots, t_k))_p \sqsubseteq \sum_{p_1,\ldots,p_k \in Q} \mu_k(\sigma)_{q,f(p_1,\ldots,p_k)} \cdot \prod_{i=1}^{k} h_\mu(t_i)_{f(p_1,\ldots,p_k)_i}$$

by induction hypothesis and the fact that $\sqsubseteq$ partially orders $\mathcal{A}$ and

$$\begin{aligned} h_\mu(\sigma(t_1, \ldots, t_k))_p &\sqsubseteq \sum_{(q_1,\ldots,q_k) \in f(Q^k)} \mu_k(\sigma)_{q,q_1,\ldots,q_k} \cdot \prod_{i=1}^{k} h_\mu(t_i)_{q_i} \\ &\sqsubseteq \sum_{q_1,\ldots,q_k \in Q} \mu_k(\sigma)_{q,q_1,\ldots,q_k} \cdot \prod_{i=1}^{k} h_\mu(t_i)_{q_i} \\ &= h_\mu(\sigma(t_1, \ldots, t_k))_q \end{aligned}$$

by idempotency of $\mathcal{A}$ and the definition of the natural order. □

We already remarked that this yields that states $p, q \in Q$ such that $p \preceq q$ and $q \preceq p$, which we call equivalent, recognize the same tree series. Let us note another property of such states.

Note 4. Let $p \preceq q \preceq p$. For every $\sigma \in \Sigma_k$ and $p_1, \ldots, p_k \in Q$ there exist $q_1, \ldots, q_k \in Q$ and $r_1, \ldots, r_k \in Q$ such that $p_i \preceq q_i \preceq r_i \preceq q_i$ for every $i \in [1, k]$ and

$$\mu_k(\sigma)_{p,p_1,\ldots,p_k} \sqsubseteq \mu_k(\sigma)_{q,q_1,\ldots,q_k} = \mu_k(\sigma)_{p,r_1,\ldots,r_k} \enspace .$$

So equivalent states enforce equally weighted transitions, but not necessarily within the same blocks (because, in general, we might have $q_i \not\preceq p_i$ for some $i \in [1, k]$ in Note 4). However, the property hints at an essential property of equivalent states p and q. If $p \neq q$, then there must be at least two transitions, one to p and one to q, with the same weight. Otherwise p and q cannot be equivalent.

Corollary 5 (of Lemma 3). *Let $\preceq$ be a backward simulation for M and $\simeq = (\preceq \cap \preceq^{-1})$. Then $h_\mu(t)_p = h_\mu(t)_q$ for every $p \simeq q$.*

This completes our investigation of the principal properties of backward simulation. Next, let us show how to reduce the size of a wta using a backward simulation. The main idea is, of course, to collapse equivalent states into just a single state. Recall, that we assumed a total order on Q and that $\min(P)$ with $P \subseteq Q$ denotes the smallest element in P with respect to that order. We use this order in our construction to obtain a unique wta. In contrast to [13, Definition 18], we thus need not discuss why the constructed wta is well-defined.

Definition 6. *Let $\preceq$ be a backward simulation for M and $\simeq = (\preceq \cap \preceq^{-1})$. The collapsed wta $(M/{\simeq}) = (Q', \Sigma, \mathcal{A}, \mu', F')$ is given by*

- *$Q' = (Q/{\simeq})$,*
- *$F'(P) = \sum_{q \in P} F(q)$ for every $P \in Q'$, and*
- *for every $\sigma \in \Sigma_k$, states $P, P_1, \ldots, P_k \in Q'$*

$$\mu'_k(\sigma)_{P,P_1,\ldots,P_k} = \sum_{q_1 \in P_1, \ldots, q_k \in P_k} \mu_k(\sigma)_{\min(P),q_1,\ldots,q_k} \enspace .$$

Clearly, $(M/{\simeq})$ never has strictly more states than M. Naturally, the best reduction is achieved by the greatest backward simulation. Next, let us show that $M/{\simeq}$ is equivalent to M, which proves that our construction preserves the semantics. Note that we make no assumptions on the total order on Q, so that the theorem will hold for any total order on Q.

Theorem 7 (cf. [10, Theorem 7]). *Let $\preceq$ be a backward simulation for M and $\simeq = (\preceq \cap \preceq^{-1})$. Then M and $M/{\simeq}$ are equivalent.*

Proof. Let $(M/{\simeq}) = (Q', \Sigma, \mathcal{A}, \mu', F')$ be the collapsed wta. We first prove that $h_{\mu'}(t)_P = h_\mu(t)_q$ for every $t \in T_\Sigma$, $P \in Q'$, and $q \in P$. Suppose that $t = \sigma(t_1, \ldots, t_k)$ for some $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$. Then

$$h_{\mu'}(\sigma(t_1, \ldots, t_k))_P$$

$$\begin{aligned}
&= \sum_{P_1,\dots,P_k \in Q'} \mu'_k(\sigma)_{P,P_1,\dots,P_k} \cdot \prod_{i=1}^{k} h_{\mu'}(t_i)_{P_i} \\
&= \sum_{P_1,\dots,P_k \in Q'} \Big(\sum_{q_1 \in P_1,\dots,q_k \in P_k} \mu_k(\sigma)_{\min(P),q_1,\dots,q_k} \Big) \cdot \prod_{i=1}^{k} h_{\mu'}(t_i)_{P_i} \\
&\overset{\dagger}{=} \sum_{q_1,\dots,q_k \in Q} \mu_k(\sigma)_{\min(P),q_1,\dots,q_k} \cdot \prod_{i=1}^{k} h_{\mu}(t_i)_{q_i} \\
&= h_\mu(\sigma(t_1,\dots,t_k))_{\min(P)} \\
&= h_\mu(\sigma(t_1,\dots,t_k))_q
\end{aligned}$$

using the induction hypothesis at † and Corollary 5 in the last step where $q \simeq \min(P)$. With this auxiliary result

$$\begin{aligned}
(\varphi_{(M/\simeq)}, t) &= \sum_{P \in Q'} F'(P) \cdot h_{\mu'}(t)_P = \sum_{P \in Q'} \Big(\sum_{q \in P} F(q) \Big) \cdot h_{\mu'}(t)_P \\
&= \sum_{q \in Q} F(q) \cdot h_\mu(t)_q = (\varphi_M, t) \enspace . \qquad \square
\end{aligned}$$

An other negative property of our notion of backward simulation is that the result obtained by collapsing M with the greatest backward simulation is not minimal with respect to backward simulation. We call M *backward-simulation minimal* if every backward simulation for it is a partial order. If a backward simulation $\preceq$ is a partial order, then $\preceq \cap \preceq^{-1}$ is the identity, which yields no reduction in the number of states if used to collapse M. Thus, a backward-simulation minimal wta cannot be reduced any further using backward simulation, which justifies the name. Let us illustrate the definitions and the principal disadvantages of backward simulation on a very simplistic example. Similar examples can easily be constructed for more commonly used idempotent semirings such as the tropical semiring $(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$.

Example 8. Let $S = \{1, 2\}$. We consider the semiring $\mathcal{P}(S) = (\mathcal{P}(S), \cup, \cap, \emptyset, S)$ where $\mathcal{P}(S)$ is the powerset of S. Moreover, consider the wta

$$M = ([1,6], \Sigma, \mathcal{P}(S), \mu, F)$$

where

– $\Sigma = \{\alpha, \gamma\}$ contains the nullary symbol α and the unary symbol γ,
– $F(i) = \{1, 2\}$ for every $i \in [1, 6]$, and
– the following transitions

$$\begin{array}{lll}
\mu_0(\alpha)_{1,\varepsilon} = \{1,2\} & \mu_0(\alpha)_{2,\varepsilon} = \{1,2\} & \mu_0(\alpha)_{3,\varepsilon} = \{1,2\} \\
\mu_1(\gamma)_{5,1} = \{1\} & \mu_1(\gamma)_{4,2} = \{1\} & \\
\mu_1(\gamma)_{5,2} = \{2\} & \mu_1(\gamma)_{4,1} = \{2\} & \\
\mu_1(\gamma)_{6,3} = \{1,2\} \enspace . & &
\end{array}$$

Let $\preceq$ be the greatest backward simulation on M, and let $\simeq = (\preceq \cap \preceq^{-1})$. Then $1 \simeq 2 \simeq 3$ and $4 \simeq 5 \preceq 6$, but $6 \not\preceq 5$ and $6 \not\preceq 4$. Then the collapsed wta is $M' = (Q', \Sigma, \mathcal{P}(S), \mu', F')$ with $Q' = \{\{1,2,3\}, \{4,5\}, \{6\}\}$, $F'(P) = \{1,2\}$ for every $P \in Q'$, and

$$\mu'_0(\alpha)_{\{1,2,3\},\varepsilon} = \{1,2\} \quad \mu'_1(\gamma)_{\{4,5\},\{1,2,3\}} = \{1,2\} \quad \mu'_1(\gamma)_{\{6\},\{1,2,3\}} = \{1,2\} \ .$$

The wta M and M' are displayed in Figure 1. Now the states $\{4,5\}$ and $\{6\}$ are equivalent; i.e., $\{4,5\}$ simulates $\{6\}$ and vice versa. This demonstrates that the collapsed wta with respect to the greatest backward simulation need not be backward-simulation minimal. □

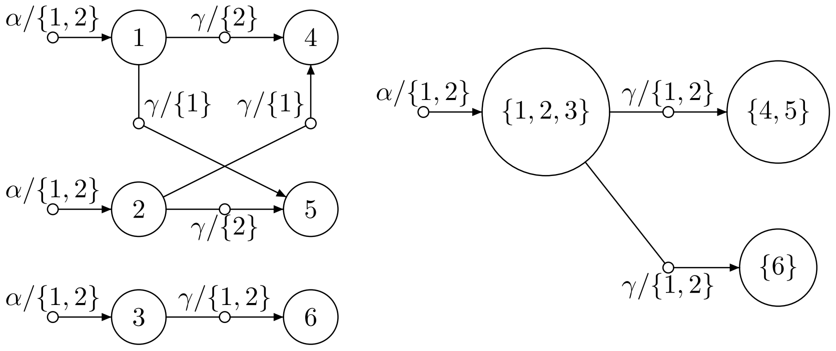


Fig. 1. The wta of Example 8 (without final weights)

Let us quickly consider deterministic wta. For every input tree $t \in T_\Sigma$, the vector $h_\mu(t)$ contains at most one nonzero entry if M is deterministic [17, Observation 4.1.6]. Thus, if M is deterministic and has no useless states (a state $q \in Q$ is useless if $h_\mu(t)_q = 0$ for every $t \in T_\Sigma$), then it is automatically backward-simulation minimal by Corollary 5. In other words, we cannot reduce a deterministic wta with the help of a backward simulation.

At the end of this section, let us develop a very simple algorithm to compute the greatest backward simulation. Our algorithm (Algorithm 1) starts with the optimistic assumption that all states simulate each other and the refines the relation as it finds evidence to the contrary (see [10,13]). To speed up the algorithm, we could also use the property mentioned in Note 4, but we present the simple, non-optimized version of the algorithm here for clarity.

Theorem 9. *Algorithm 1 returns the greatest backward simulation for M.*

Proof. Let $\preceq$ be the greatest backward simulation for M. First, we prove that $\preceq \subseteq R_i$ for every i that is encountered during the run of the algorithm. Let us proceed by induction on i. Trivially $\preceq \subseteq R_0$ because $R_0 = Q \times Q$. Now, let us assume that $p \preceq q$. Consequently, $(p,q) \in R_i$ by the induction hypothesis. Let

Algorithm 1. Computing the greatest backward simulation for M.

```
$R_0 \leftarrow Q \times Q$
$i \leftarrow 0$
**repeat**
  $j \leftarrow i$
  **for all** $\sigma \in \Sigma_k$ and $p_1, \dots, p_k \in Q$ **do**
    $R_{i+1} \leftarrow \{(p, q) \in R_i \mid$
              $\exists (p_1, q_1), \dots, (p_k, q_k) \in R_i \colon \mu_k(\sigma)_{p,p_1,\dots,p_k} \sqsubseteq \mu_k(\sigma)_{q,q_1,\dots,q_k}\}$
    $i \leftarrow i + 1$
**until** $R_i = R_j$
```

$\sigma \in \Sigma_k$ and $p_1, \dots, p_k \in Q$. Then by Definition 1 there exist $q_1, \dots, q_k \in Q$ such that $\mu_k(\sigma)_{p,p_1,\dots,p_k} \sqsubseteq \mu_k(\sigma)_{q,q_1,\dots,q_k}$ and $p_i \preceq q_i$ for every $i \in [1, k]$. By induction hypothesis, we also have $(p_i, q_i) \in R_i$ for every $i \in [1, k]$. Consequently, we obtain $(p, q) \in R_{i+1}$. This proves $\preceq \subseteq R_{i+1}$. At termination, R_i is a backward simulation (see Definition 1) and since $\preceq \subseteq R_i$ and $\preceq$ is the greatest backward simulation for M, we can conclude that $R_i = \preceq$. □

4 A Forward Simulation

Next, we consider a forward version of the simulation of Section 3. Similar simulations (called composed simulations) for the unweighted case are considered in [10] and forward bisimulation for wta is considered in [13]. Let us follow the structure of the previous section and start with the definition of a forward simulation. Note that we will use the same symbols here as in Section 3, but it should be clear that we exclusively speak about forward simulations here unless otherwise mentioned.

For state $q \in Q$ to (forward) simulate another state p, written $p \preceq q$, we demand that for every transition weight $\mu_k(\sigma)_{p',q_1,\dots,q_{i-1},p,q_{i+1},\dots,q_k}$, there exists a larger transition weight $\mu_k(\sigma)_{q',q_1,\dots,q_{i-1},q,q_{i+1},\dots,q_k}$ with the additional restriction that the state q' simulates p'. In addition, the final weight of q should be larger than the one of p. In the unweighted case, this coincides with the definition of an upward simulation [10, Section 2] with respect to the identity as a backward simulation. There it is demanded that q should be a final state if p is. Moreover, for every transition $\sigma(q_1, \dots, q_{i-1}, p, q_{i+1}, \dots, q_k) \to p'$ there should exist a transition $\sigma(q_1, \dots, q_{i-1}, q, q_{i+1}, \dots, q_k) \to q'$ such that q' simulates p'.

Definition 10. *A quasi-order $\preceq \subseteq Q \times Q$ is a* forward simulation *for M if for every $p \preceq q$ the following two conditions are satisfied:*

- *$F(p) \sqsubseteq F(q)$ and*
- *for every $\sigma \in \Sigma_k$, $i \in [1, k]$, and $p', q_1, \dots, q_k \in Q$ there exist $q' \in Q$ such that $p' \preceq q'$ and*

$$\mu_k(\sigma)_{p',q_1,\dots,q_{i-1},p,q_{i+1},\dots,q_k} \sqsubseteq \mu_k(\sigma)_{q',q_1,\dots,q_{i-1},q,q_{i+1},\dots,q_k} \enspace .$$

A forward simulation is also only a quasi-order and not an equivalence relation like every forward bisimulation. We do not consider upward simulations [10] here since we believe that two independent simulations are easier to understand and we can always first reduce with the help of a backward simulation and then with a forward simulation to achieve roughly the same as with an upward simulation of [10]. Let us proceed with the principal properties of forward simulations. As in the backward case, there exists a greatest forward simulation for M.

Theorem 11 (see [13, Theorem 7]). *There exists a greatest forward simulation for M.*

Proof. Let $\preceq$ and $\preceq'$ be forward simulations for M. Again, we claim that $(\preceq \cup \preceq')^*$ is a forward simulation. Clearly, $(\preceq \cup \preceq')^*$ is a quasi-order. Let $(p, q) \in (\preceq \cup \preceq')^*$, $\sigma \in \Sigma_k$, $i \in [1, k]$, and $p', q_1, \ldots, q_k \in Q$. Consequently, there exist $r_1, \ldots, r_n \in Q$ such that

$$p = r_0 \preceq r_1 \preceq' r_2 \preceq r_3 \preceq' \cdots \preceq' r_n = q \enspace .$$

By this chain of inequalities, there also exists $q' \in Q$ such that

$$\mu_k(\sigma)_{p', q_1, \ldots, q_{i-1}, p, q_{i+1}, \ldots, q_k} \sqsubseteq \mu_k(\sigma)_{q', q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_k}$$

and $p' \; (\preceq ; \preceq' ; \preceq ; \preceq' ; \cdots ; \preceq') \; q'$, which proves that $(\preceq \cup \preceq')^*$ is a forward simulation. ☐

To state the main property of similar states, we need some additional notions. A context is a tree of $T_\Sigma(\{\square\})$, where $\square$ is a distinguished (fixed) symbol, such that $\square$ occurs exactly once. The set of all contexts is denoted by C_Σ. The tree $c[t]$ is obtained by replacing the symbol $\square$ in the context $c \in C_\Sigma$ by the tree $t \in T_\Sigma$.

Lemma 12. *Let $\preceq$ be a forward simulation for M. Moreover, let $c \in C_\Sigma$ and $p \preceq q$. Then $\sum_{r \in B} h_\mu(c[p])_r \sqsubseteq \sum_{r \in B} h_\mu(c[q])_r$ for every up-set $B \subseteq Q$.*

Proof. We prove the statement by induction on $c \in C_\Sigma$. In the base case, let $c = \square$. Then

$$\sum_{r \in B} h_\mu(p)_r = \begin{cases} 1 & \text{if } p \in B \\ 0 & \text{otherwise.} \end{cases}$$

Since B is an up-set, $p \in B$ implies $q \in B$ and thus by $0 \sqsubseteq 1$

$$\begin{aligned} \sum_{r \in B} h_\mu(p)_r &\sqsubseteq \begin{cases} 1 & \text{if } q \in B \\ 0 & \text{otherwise} \end{cases} \\ &= \sum_{r \in B} h_\mu(q)_r \enspace . \end{aligned}$$

In the induction step, let $c = \sigma(t_1, \ldots, t_{j-1}, c', t_{j+1}, \ldots, t_k)$ for some $\sigma \in \Sigma_k$, $j \in [1, k]$, $c' \in C_\Sigma$, and $t_1, \ldots, t_k \in T_\Sigma$. Then

$$\sum_{r \in B} h_\mu(\sigma(t_1, \ldots, t_{j-1}, c'[p], t_{j+1}, \ldots, t_k))_r$$

$$= \sum_{\substack{r \in B \\ p_1, \dots, p_k \in Q}} \mu_k(\sigma)_{r, p_1, \dots, p_k} \cdot h_\mu(c'[p])_{p_j} \cdot \prod_{i \in [1,k] \setminus \{j\}} h_\mu(t_i)_{p_i} \enspace .$$

Then $h_\mu(c'[p])_{p_j} \sqsubseteq \sum_{p' \in \uparrow(p_j)} h_\mu(c'[q])_{p'}$ by the induction hypothesis, and thus

$$\sqsubseteq \sum_{\substack{r \in B \\ p_1, \dots, p_k \in Q}} \mu_k(\sigma)_{r, p_1, \dots, p_k} \cdot \Bigl(\sum_{p' \in \uparrow(p_j)} h_\mu(c'[q])_{p'} \Bigr) \cdot \prod_{i \in [1,k] \setminus \{j\}} h_\mu(t_i)_{p_i}$$

$$\sqsubseteq \sum_{\substack{r \in B \\ p_1, \dots, p_k \in Q \\ p' \in \uparrow(p_j)}} \Bigl(\sum_{r' \in \uparrow(r)} \mu_k(\sigma)_{r', p_1, \dots, p_{j-1}, p', p_{j+1}, \dots, p_k} \Bigr) \cdot h_\mu(c'[q])_{p'} \cdot \prod_{i \in [1,k] \setminus \{j\}} h_\mu(t_i)_{p_i}$$

because $p_j \preceq p'$ and thus for every $r \in B$ there exists $r' \in Q$ such that $r \preceq r'$ and $\mu_k(\sigma)_{r, p_1, \dots, p_k} \sqsubseteq \mu_k(\sigma)_{r', p_1, \dots, p_{j-1}, p', p_{j+1}, \dots, p_k}$. Since B is an up-set and $\mathcal{A}$ idempotent, we continue with

$$= \sum_{\substack{r \in B \\ p_1, \dots, p_k \in Q}} \mu_k(\sigma)_{r, p_1, \dots, p_k} \cdot h_\mu(c'[q])_{p_j} \cdot \prod_{i \in [1,k] \setminus \{j\}} h_\mu(t_i)_{p_i}$$

$$= \sum_{r \in B} h_\mu(\sigma(t_1, \dots, t_{j-1}, c'[q], t_{j+1}, \dots, t_k))_r \enspace . \qquad \square$$

Next, let us show how to reduce the size of a wta using a forward simulation. We again make use of the total order on Q to simplify the construction. In particular, the minimum operation in the construction refers to this total order and not to the forward simulation for M.

Definition 13 (cf. [13, Definition 3]). *Let $\preceq$ be a forward simulation for M and $\simeq = (\preceq \cap \preceq^{-1})$. The collapsed wta $(M/\simeq) = (Q', \Sigma, \mathcal{A}, \mu', F')$ is given by*

- *$Q' = (Q/\simeq)$,*
- *$F'(P) = F(\min(P))$ for every $P \in Q'$, and*
- *for every $\sigma \in \Sigma_k$, states $P, P_1, \dots, P_k \in Q'$*

$$\mu'_k(\sigma)_{P, P_1, \dots, P_k} = \sum_{q \in P} \mu_k(\sigma)_{q, \min(P_1), \dots, \min(P_k)}$$

As before, the collapsed wta $M/\simeq$ never has more states than M itself and the best reduction is achieved by the greatest forward simulation. However, we first need to show that $M/\simeq$ is equivalent to M. Beforehand, let us note an important property of equivalent states (i.e., states that simulate each other) that follows immediately from Definition 10.

Note 14. Let $\preceq$ be a forward simulation for M, and let $\simeq = (\preceq \cap \preceq^{-1})$. Then for every $p \simeq q$, $\sigma \in \Sigma_k$, $i \in [1, k]$, and $p', q_1, \dots, q_k \in Q$, there exist $q', r' \in Q$ such that $p' \preceq q' \simeq r'$ and

$$\begin{aligned} \mu_k(\sigma)_{p', q_1, \dots, q_{i-1}, p, q_{i+1}, \dots, q_k} &\sqsubseteq \mu_k(\sigma)_{q', q_1, \dots, q_{i-1}, p, q_{i+1}, \dots, q_k} \\ &= \mu_k(\sigma)_{r', q_1, \dots, q_{i-1}, q, q_{i+1}, \dots, q_k} \enspace . \end{aligned}$$

We will use this property in the proof of the next theorem, which will prove the correctness of our construction.

Theorem 15 (cf. [10, Theorem 7]). *Let $\preceq$ be a forward simulation for M and $\simeq = (\preceq \cap \preceq^{-1})$. Then M and $M/\simeq$ are equivalent.*

Proof. Let $(M/\simeq) = (Q', \Sigma, \mathcal{A}, \mu', F')$ be the collapsed wta. We first prove that $h_{\mu'}(t)_P = \sum_{q \in \uparrow(P)} h_\mu(t)_q$ for every $t \in T_\Sigma$, and $P \in Q'$. Suppose that $t = \sigma(t_1, \ldots, t_k)$ for some $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$. Then we compute as follows where the equality marked † is explained below.

$$\begin{aligned}
& h_{\mu'}(\sigma(t_1, \ldots, t_k))_P \\
&= \sum_{P_1, \ldots, P_k \in Q'} \mu'_k(\sigma)_{P, P_1, \ldots, P_k} \cdot \prod_{i=1}^{k} h_{\mu'}(t_i)_{P_i} \\
&= \sum_{P_1, \ldots, P_k \in Q'} \mu'_k(\sigma)_{P, P_1, \ldots, P_k} \cdot \prod_{i=1}^{k} \Big(\sum_{q_i \in \uparrow(P_i)} h_\mu(t_i)_{q_i} \Big) \\
&= \sum_{\substack{P_1, \ldots, P_k \in Q' \\ q_1 \in \uparrow(P_1), \ldots, q_k \in \uparrow(P_k)}} \Big(\sum_{q \in \uparrow(P)} \mu_k(\sigma)_{q, \min(P_1), \ldots, \min(P_k)} \Big) \cdot \prod_{i=1}^{k} h_\mu(t_i)_{q_i} \\
&= \sum_{q \in \uparrow(P)} \sum_{\substack{P_1, \ldots, P_k \in Q' \\ q_1 \in \uparrow(P_1), \ldots, q_k \in \uparrow(P_k)}} \mu_k(\sigma)_{q, \min(P_1), \ldots, \min(P_k)} \cdot \prod_{i=1}^{k} h_\mu(t_i)_{q_i} \\
&\overset{\dagger}{=} \sum_{q \in \uparrow(P)} \sum_{\substack{P_1, \ldots, P_k \in Q' \\ q_1 \in \uparrow(P_1), \ldots, q_k \in \uparrow(P_k)}} \mu_k(\sigma)_{q, q_1, \ldots, q_k} \cdot \prod_{i=1}^{k} h_\mu(t_i)_{q_i} \\
&= \sum_{q \in \uparrow(P)} h_\mu(\sigma(t_1, \ldots, t_k))_q
\end{aligned}$$

Let us take a closer look at the equation marked †. We can show this equality by showing both inequalities. Let us consider the inequality $\sqsubseteq$ first. Clearly, it is sufficient to show that for each summand of the left-hand side there exists a larger summand in the right-hand side. For this we consider a summand $\mu_k(\sigma)_{p, \min(P_1), \ldots, \min(P_k)} \cdot \prod_{i=1}^{k} h_\mu(t_i)_{q_i}$ of the left-hand side of † for some $P_1, \ldots, P_k \in Q'$, $p \in \uparrow(P)$, and $q_1, \ldots, q_k \in Q$ such that $q_i \in \uparrow(P_i)$ for every $i \in [1, k]$. Since $\min(P_i) \preceq q_i$ for every $i \in [1, k]$, there exists $q \in Q$ such that $\mu_k(\sigma)_{p, \min(P_1), \ldots, \min(P_k)} \sqsubseteq \mu_k(\sigma)_{q, q_1 \cdots q_k}$ by Definition 10. Consequently,

$$\mu_k(\sigma)_{p, \min(P_1), \ldots, \min(P_k)} \cdot \prod_{i=1}^{k} h_\mu(t_i)_{q_i} \sqsubseteq \mu_k(\sigma)_{q, q_1, \ldots, q_k} \cdot \prod_{i=1}^{k} h_\mu(t_i)_{q_i}$$

and the latter is a summand on the right-hand side of †. For the converse inequality, let us consider a summand $\mu_k(\sigma)_{q, q_1, \ldots, q_k} \cdot \prod_{i=1}^{k} h_\mu(t_i)_{q_i}$ in the right-hand side where $q \in \uparrow(P)$ and $q_1, \ldots, q_k \in Q$. For every $i \in [1,k]$ let $P_i = [q_i]$. Then $\min(P_i) \simeq q_i$ for every $i \in [1,k]$. Then by Definition 10 and the property remarked in Note 14, there exist $p, q' \in Q$ such that $q \preceq q' \simeq p$ and

$$\mu_k(\sigma)_{q,q_1,\ldots,q_k} \sqsubseteq \mu_k(\sigma)_{q',q_1,\ldots,q_k} = \mu_k(\sigma)_{p,\min(P_1),\ldots,\min(P_k)} \enspace .$$

It follows that

$$\mu_k(\sigma)_{q,q_1,\ldots,q_k} \cdot \prod_{i=1}^{k} h_\mu(t_i)_{q_i} \sqsubseteq \mu_k(\sigma)_{p,\min(P_1),\ldots,\min(P_k)} \cdot \prod_{i=1}^{k} h_\mu(t_i)_{q_i} \enspace ,$$

which is a summand of the left-hand side because $q \preceq p$. This completes the proof of our auxiliary statement. For the statement of the theorem, we compute as follows:

$$\begin{aligned}(\varphi_{(M/\simeq)}, t) &= \sum_{P \in Q'} F'(P) \cdot h_{\mu'}(t)_P = \sum_{P \in Q'} F'(P) \cdot \Big(\sum_{q \in \uparrow(P)} h_\mu(t)_q \Big) \\ &= \sum_{P \in Q', q \in \uparrow(P)} F'(P) \cdot h_\mu(t)_q = \sum_{q \in Q} F(q) \cdot h_\mu(t)_q = (\varphi_M, t)\end{aligned}$$

because $F(p) \sqsubseteq F(q)$ if $p \preceq q$. This proves our theorem. □

Also the notion of forward simulation has the negative properties outlined in the section on backward simulation. For example, the result obtained by collapsing M with the greatest forward simulation is again not necessarily minimal with respect to forward simulation. Accordingly, we call M *forward-simulation minimal* if every forward simulation for it is a partial order. Let us also present a small example for forward simulation.

Example 16. Consider the wta M of Example 8. Let $\preceq$ be the coarsest forward simulation for it, and let $\simeq\; = (\preceq \cap \preceq^{-1})$. Then $4 \simeq 5 \simeq 6$ and $1 \simeq 2 \preceq 3$ but $3 \not\preceq 2$. Consequently, the collapsed wta is $M' = (Q', \Sigma, \mathcal{P}(S), \mu', F')$ where $Q' = \{\{1,2\},\{3\},\{4,5,6\}\}$, $F'(P) = \{1,2\}$ for every $P \in Q'$, and

$$\begin{aligned} \mu'_0(\alpha)_{\{1,2\},\varepsilon} &= \{1,2\} & \qquad \mu'_1(\gamma)_{\{4,5,6\},\{1,2\}} &= \{1,2\} \\ \mu'_0(\alpha)_{\{3\},\varepsilon} &= \{1,2\} & \qquad \mu'_1(\gamma)_{\{4,5,6\},\{3\}} &= \{1,2\} \enspace . \end{aligned}$$

Figure 2 displays M and M'. In M' the states $\{1,2\}$ and $\{3\}$ are equivalent; i.e., $\{1,2\}$ simulates $\{3\}$ and vice versa. This again demonstrates that the collapsed wta with respect to the greatest forward simulation need not be forward-simulation minimal. □

Let us also discuss the deterministic case. Roughly speaking, we claim that reduction with the help of forward simulation is not more effective than reduction with the help of forward bisimulation on deterministic wta. Let us quickly recall the definition of a forward bisimulation [13] for M.

Definition 17 (see [13, Definition 1]). *An equivalence relation $\equiv$ on Q is a* forward bisimulation *for M if for every $p \equiv q$ the following two conditions hold:*

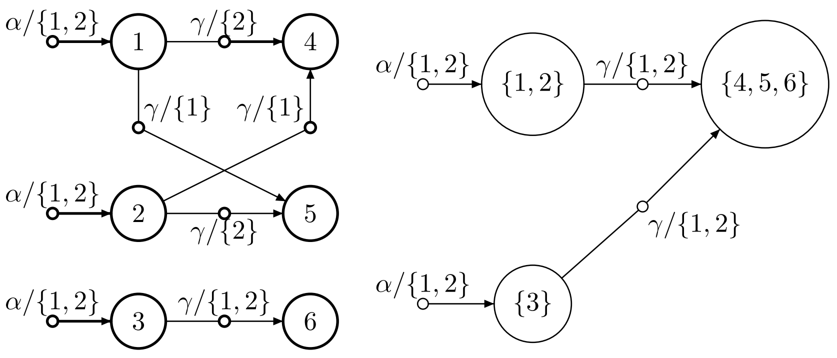


Fig. 2. The wta of Example 16 (without final weights)

(i) $F(p) = F(q)$ *and*
(ii) *for every* $\sigma \in \Sigma_k$, $i \in [1, k]$, $q_1, \ldots, q_k \in Q$, *and* $P \in (Q/\equiv)$

$$\sum_{r \in P} \mu_k(\sigma)_{r,q_1,\ldots,q_{i-1},p,q_{i+1},\ldots,q_k} = \sum_{r \in P} \mu_k(\sigma)_{r,q_1,\ldots,q_{i-1},q,q_{i+1},\ldots,q_k} \enspace .$$

Suppose that M is deterministic. To prove that reduction with the help of forward simulation is only as effective as reduction with the help of forward bisimulation, it is sufficient to show that any equivalence relation $\simeq$ obtained from a forward simulation $\preceq$ for M is indeed a forward bisimulation for M. The algorithm in [13] can then be used to compute an equivalent wta that has at most as many states as $(M/\simeq)$. Recall that a state $q \in Q$ is useless if $h_\mu(t)_q = 0$ for every $t \in T_\Sigma$.

Theorem 18. *Let M be deterministic and without useless states. Moreover, let $\preceq$ be a forward simulation for M, and $\simeq\, = (\preceq \cap \preceq^{-1})$. Then $\simeq$ is a forward bisimulation for M.*

Proof. Let $p \simeq q$, $\sigma \in \Sigma_k$, $i \in [1, k]$, and $p', q_1, \ldots, q_k \in Q$ be such that $\mu_k(\sigma)_{p',q_1,\ldots,q_{i-1},p,q_{i+1},\ldots,q_k} \neq 0$. By determinism there exists at most one such p' and if M has no useless states, then there exists at least one such p'. Since $p \simeq q$, there exist $q', r' \in Q$ such that $p' \preceq q' \preceq r'$ and

$$\begin{aligned} \mu_k(\sigma)_{p',q_1,\ldots,q_{i-1},p,q_{i+1},\ldots,q_k} &\sqsubseteq \mu_k(\sigma)_{q',q_1,\ldots,q_{i-1},q,q_{i+1},\ldots,q_k} \\ &\sqsubseteq \mu_k(\sigma)_{r',q_1,\ldots,q_{i-1},p,q_{i+1},\ldots,q_k} \enspace . \end{aligned}$$

By determinism, $r' = p'$ and thus $p' \simeq q'$ and

$$\mu_k(\sigma)_{p',q_1,\ldots,q_{i-1},p,q_{i+1},\ldots,q_k} = \mu_k(\sigma)_{q',q_1,\ldots,q_{i-1},q,q_{i+1},\ldots,q_k} \enspace .$$

Consequently, for every $P \in (Q/\simeq)$

$$\sum_{r \in P} \mu_k(\sigma)_{r,q_1,\ldots,q_{i-1},p,q_{i+1},\ldots,q_k} = \begin{cases} \mu_k(\sigma)_{p',q_1,\ldots,q_{i-1},p,q_{i+1},\ldots,q_k} & \text{if } p' \in P \\ 0 & \text{otherwise} \end{cases}$$

$$= \begin{cases} \mu_k(\sigma)_{q',q_1,\dots,q_{i-1},q,q_{i+1},\dots,q_k} & \text{if } q' \in P \\ 0 & \text{otherwise} \end{cases}$$
$$= \sum_{r \in P} \mu_k(\sigma)_{r,q_1,\dots,q_{i-1},q,q_{i+1},\dots,q_k}$$

because $p' \in P$ if and only if $q' \in P$. This proves condition (ii) of Definition 17. For condition (i) of the same definition, we simply observe that $F(p) \sqsubseteq F(q) \sqsubseteq F(p)$, which proves it and hence the statement that $\simeq$ is a forward bisimulation. □

Minimization (i.e., finding a minimal deterministic wta that is equivalent to M) of deterministic wta is discussed in [12]. Note that the previous theorem also proves that reduction with the help of the greatest forward simulation does not necessarily yield a minimal deterministic wta. This is due to the fact that forward bisimulation does not achieve that (cf. [18, Theorem 3.12]).

Algorithm 2. Computing the greatest forward simulation for M.

```
$R_0 \leftarrow \{(p, q) \in Q \times Q \mid F(p) \sqsubseteq F(q)\}$
$i \leftarrow 0$
repeat
  $j \leftarrow i$
  for all $\sigma \in \Sigma_k$, $n \in [1, k]$, and $p', q_1, \dots, q_k \in Q$ do
    $R_{i+1} \leftarrow \{(p, q) \in R_i \mid \exists (p', q') \in R_i :$
              $\mu_k(\sigma)_{p',q_1,\dots,q_{n-1},p,q_{n+1},\dots,q_k} \sqsubseteq \mu_k(\sigma)_{q',q_1,\dots,q_{n-1},q,q_{n+1},\dots,q_k}\}$
    $i \leftarrow i + 1$
until $R_i = R_j$
```

Finally, let us develop an algorithm for the greatest forward simulation. Our algorithm is displayed in Algorithm 2.

Theorem 19. *Algorithm 2 returns the greatest forward simulation for M.*

Proof. Let $\preceq$ be the greatest forward simulation for M. Again we prove that $\preceq \subseteq R_i$ for every relevant i as an auxiliary statement. Using the first condition of Definition 10, we have $\preceq \subseteq R_0$. Suppose that $p \preceq q$. Then $(p, q) \in R_i$ by the induction hypothesis. Moreover, let $\sigma \in \Sigma_k$, $n \in [1, k]$, and $p', q_1, \dots, q_k \in Q$. Since $p \preceq q$, we can conclude that there exists $q' \in Q$ such that $p' \preceq q'$ and

$$\mu_k(\sigma)_{p',q_1,\dots,q_{n-1},p,q_{n+1},\dots,q_k} \sqsubseteq \mu_k(\sigma)_{q',q_1,\dots,q_{n-1},q,q_{n+1},\dots,q_k} \enspace .$$

Invoking the induction hypothesis, we obtain $(p', q') \in R_i$ and thus $(p, q) \in R_{i+1}$. Thus $\preceq \subseteq R_{i+1}$. Clearly, R_i is a forward simulation for M (see Definition 10) at termination. Since $\preceq \subseteq R_i$ and $\preceq$ is the greatest forward simulation for M, we can conclude that $R_i = \preceq$. □

Acknowledgements

The author would like to thank the reviewers for their shared wisdom. Their comments improved the presentation of the paper.

References

1. Abdulla, P.A., Jonsson, B., Mahata, P., d'Orso, J.: Regular tree model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 555–568. Springer, Heidelberg (2002)
2. Knight, K., Graehl, J.: An overview of probabilistic tree transducers for natural language processing. In: Gelbukh, A. (ed.) CICLing 2005. LNCS, vol. 3406, pp. 1–24. Springer, Heidelberg (2005)
3. Hopcroft, J.E.: An $n \log n$ algorithm for minimizing states in a finite automaton. In: Theory of Machines and Computations, pp. 189–196. Academic Press, London (1971)
4. Högberg, J., Maletti, A., May, J.: Backward and forward bisimulation minimization of tree automata. Theoret. Comput. Sci. (to appear, 2009), `http://dx.doi.org/10.1016/j.tcs.2009.03.022`
5. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: Proc. FOCS, pp. 125–129. IEEE Computer Society Press, Los Alamitos (1972)
6. Gramlich, G., Schnitger, G.: Minimizing nFA's and regular expressions. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 399–411. Springer, Heidelberg (2005)
7. Gramlich, G., Schnitger, G.: Minimizing nfa's and regular expressions. J. Comput. System Sci. 73(6), 908–923 (2007)
8. Gruber, H., Holzer, M.: Inapproximability of nondeterministic state and transition complexity assuming $\mathbf{P} \neq \mathbf{NP}$. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 205–216. Springer, Heidelberg (2007)
9. Abdulla, P.A., Högberg, J., Kaati, L.: Bisimulation minimization of tree automata. Int. J. Found. Comput. Sci. 18(4), 699–713 (2007)
10. Abdulla, P.A., Bouajjani, A., Holík, L., Kaati, L., Vojnar, T.: Computing simulations over tree automata. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 93–108. Springer, Heidelberg (2008)
11. Abdulla, P.A., Bouajjani, A., Holík, L., Kaati, L., Vojnar, T.: Composed bisimulation for tree automata. In: Ibarra, O.H., Ravikumar, B. (eds.) CIAA 2008. LNCS, vol. 5148, pp. 212–222. Springer, Heidelberg (2008)
12. Maletti, A.: Minimizing deterministic weighted tree automata. Inform. and Comput. (to appear, 2009), `http://dx.doi.org/10.1016/j.ic.2009.01.004`
13. Högberg, J., Maletti, A., May, J.: Bisimulation minimisation for weighted tree automata. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 229–241. Springer, Heidelberg (2007)
14. Berstel, J., Reutenauer, C.: Recognizable formal power series on trees. Theoret. Comput. Sci. 18(2), 115–148 (1982)

15. Bozapalidis, S.: Equational elements in additive algebras. Theory Comput. Systems 32(1), 1–33 (1999)
16. Borchardt, B., Vogler, H.: Determinization of finite state weighted tree automata. J. Autom. Lang. Combin. 8(3), 417–463 (2003)
17. Borchardt, B.: The Theory of Recognizable Tree Series. PhD thesis, Technische Universität Dresden (2005)
18. Högberg, J., Maletti, A., May, J.: Bisimulation minimisation of weighted tree automata. Technical Report ISI-TR-634, University of Southern California (2007)

Syntax-Directed Translations and Quasi-alphabetic Tree Bimorphisms — Revisited

Andreas Maletti[*] and Cătălin Ionuţ Tîrnăucă[**]

Universitat Rovira i Virgili
Departament de Filologies Romàniques
Av. Catalunya 35, 43002 Tarragona, Spain
`andreas.maletti@urv.cat`,
`catalinionut.tirnauca@estudiants.urv.cat`

Abstract. Quasi-alphabetic tree bimorphisms [Steinby, Tîrnăucă: Defining syntax-directed translations by tree bimorphisms. *Theor. Comput. Sci.*, to appear. `http://dx.doi.org/10.1016/j.tcs.2009.03.009`, 2009] are reconsidered. It is known that the class of (string) translations defined by such bimorphisms coincides with the class of syntax-directed translations. This result is extended to a smaller class of tree bimorphisms namely (linear and complete) symbol-to-symbol tree bimorphisms. Moreover, it is shown that the class of simple syntax-directed translations coincides with the class of translations defined by alphabetic tree bimorphisms (also known as finite-state relabelings). This proves that alphabetic tree bimorphisms are not sufficiently powerful to model all syntax-directed translations. Finally, it is shown that the class of tree transformations defined by quasi-alphabetic tree bimorphisms is closed under composition. The corresponding result is known in the variable-free case. Overall, the main results of [Steinby, Tîrnăucă] are strengthened.

Keywords: syntax-directed translation, regular tree language, tree bimorphism, natural language processing.

1 Introduction

The field of syntax-based machine translation was established by the demanding need of systems used in practical translations between natural languages (for example, Arabic to English). Modern systems should be able to perform local rotations and capture syntax-sensitive transformations (i.e, tree transformations). Another important property that such a system should possess is composability. This property allows us to split the system into subsystems, which are easier to handle, train, and study. Those subsystems can then be assembled into a large system by an automatic composition construction [1,2].

[*] This author was financially supported by the *Ministerio de Educación y Ciencia* (MEC) grant JDCI-2007-760.

[**] This author is indebted to the MEC project MTM-2007-63422 which made this work possible.

Two powerful tools that define tree transformations have been proposed during the past decades in the formal language community: tree transducers and tree bimorphisms (see [3,4] for surveys). The former devices are operational and easy to implement but closure under composition only holds for few classes of tree transformations [2,3,5]. This closure is easier to establish using the latter devices by imposing suitable restrictions on their constituents [6,7,8,9], but tree bimorphisms are more difficult to implement. More precisely, a tree bimorphism is formed by two tree homomorphisms and a center tree language. The tree transformation is obtained by applying both homomorphisms to elements of the center tree language. One homomorphism yields the input tree and the other homomorphism yields the corresponding output tree. If we take the yield of the input and output tree, then we obtain a (string) translation.

Synchronous grammars [10,11,12] are another way to define tree transformations. They easily capture even difficult local rotations that are required by pairs of natural languages with very different syntax-structures (e.g., Chinese and English). A synchronous grammar basically consists of two grammars, in which the productions have associated nonterminals. The derivations are then obtained by applying two suitable rules, one of each grammar, to associated nonterminals. Again one side produces the input tree and the other side produces the output tree in this fashion. Unfortunately, few closure under composition results were known about such grammars until [13] related synchronous grammars and tree bimorphisms.

One synchronous grammar device is the syntax-directed translation schema (SDTS), which appeared first as a simple model of a compiler [10] (see [14] for a survey). In the spirit of [13], quasi-alphabetic tree bimorphisms [15] were shown to be as powerful as SDTSs for string translations. Moreover, for quasi-alphabetic tree bimorphisms, in which the center tree language does not permit variables, the class of tree transformations (and thus also the class of string translations) defined by them is shown to be closed under composition [15].

Here we sharpen the connection between SDTSs and tree bimorphisms. The class of all translations defined by SDTSs coincides with the class of all translations defined by (linear and complete) symbol-to-symbol tree bimorphisms (see Section 3). The latter devices define a strictly smaller class of tree transformations than quasi-alphabetic tree bimorphisms. In addition, simple SDTSs [16,17] are equally powerful as alphabetic tree bimorphisms [3] (finite-state relabelings [5]). Finally, we strengthen the closure under composition result of quasi-alphabetic tree bimorphisms by showing that the class of tree transformations defined by them remains closed under composition even if we allow variables in the center tree language (see Section 4).

2 Preliminaries

The nonnegative integers are denoted by $\mathbb{N}$. For every $k \in \mathbb{N}$, the set $\{i \in \mathbb{N} \mid 1 \leqslant i \leqslant k\}$ is denoted by $[k]$. Let R, S, and T be sets and $\rho \subseteq R \times S$ a relation. We occasionally write $r \; \rho \; s$ instead of $(r, s) \in \rho$. The *inverse* of ρ

is $\rho^{-1} = \{(s, r) \mid r \; \rho \; s\}$ and the reflexive and transitive closure of ϱ is denoted by ϱ^*. The *composition* of ρ with $\tau \subseteq S \times T$ is $\rho\,;\tau = \{(r, t) \mid \exists s \in S \colon r \; \rho \; s \; \tau \; t\}$. Finally, $|S|$ is the cardinality of the (finite) set S.

For a set V, we denote by V^* the set of all *strings* over V and by ε the *empty string*. An *alphabet* is a finite set (of symbols). A *ranked alphabet* (Σ, rk) is an alphabet Σ together with a mapping $\mathrm{rk}\colon \Sigma \to \mathbb{N}$. Often we leave rk implicit. For every $k \in \mathbb{N}$, let $\Sigma_k = \{f \in \Sigma \mid \mathrm{rk}(f) = k\}$.

Let Σ be a ranked alphabet and T a set. Then

$$\Sigma(T) = \{f(t_1, \ldots, t_k) \mid f \in \Sigma_k, t_1, \ldots, t_k \in T\} \enspace .$$

The set $T_\Sigma(V)$ of all *Σ-trees indexed by* variables V is the smallest set $\mathcal{T}$ such that $V \subseteq \mathcal{T}$ and $\Sigma(\mathcal{T}) \subseteq \mathcal{T}$. Subsets of $T_\Sigma(V)$ are *tree languages*. Such a tree language L is *variable-free* (respectively, *almost variable-free*) if $L \subseteq T_\Sigma$ (respectively, $L \subseteq T_\Sigma \cup V$). Generally, for all considered trees $t \in T_\Sigma(V)$ we assume that $\Sigma \cap V = \emptyset$, so that we can safely write c instead of $c()$ for every $c \in \Sigma_0$. For every tree $t \in T_\Sigma(V)$, the set $\mathrm{pos}(t) \subseteq \mathbb{N}^*$ of *positions* of t is inductively defined by $\mathrm{pos}(v) = \{\varepsilon\}$ for every $v \in V$, and

$$\mathrm{pos}(f(t_1, \ldots, t_k)) = \{\varepsilon\} \cup \{iw \mid i \in [k], w \in \mathrm{pos}(t_i)\}$$

for every $f \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma(V)$. Let $w \in \mathrm{pos}(t)$. The *label* of t at w, the *subtree* of t at w, and the *replacement* of that subtree by $s \in T_\Sigma(V)$ are denoted by $t(w)$, by $t|_w$, and by $t[s]_w$, respectively.

A tree $t \in T_\Sigma(V)$ is *linear* (respectively, *nondeleting*) in $Y \subseteq V$ if every $y \in Y$ occurs at most (respectively, at least) once in t. Let $D \subseteq V \cup \Sigma_0$. The *$D$-yield* of t is defined inductively by $\mathrm{yd}_D(d) = d$ for every $d \in D$, $\mathrm{yd}_D(v) = \varepsilon$ for every $v \in V \setminus D$, and

$$\mathrm{yd}_D(f(t_1, \ldots, t_k)) = \mathrm{yd}_D(t_1) \cdots \mathrm{yd}_D(t_k)$$

for every $f \in \Sigma_k \setminus D$ and $t_1, \ldots, t_k \in T_\Sigma(V)$.

We fix a set $X = \{x_i \mid i \geqslant 1\}$ of *formal variables* (disjoint to all other ranked alphabets and variables considered). For every $n \in \mathbb{N}$, we let $X_n = \{x_i \mid i \in [n]\}$. For all $t, t_1, \ldots, t_n \in T_\Sigma(V \cup X_n)$, we denote by $t[t_1, \ldots, t_n]$ the result obtained by replacing, for every $i \in [n]$, every occurrence of x_i in t by t_i. For every $v \in V$, we denote by $t[v \leftarrow (t_1, \ldots, t_n)]$ the result of replacing, for every $i \in [n]$, the i-th (with respect to the lexicographic order on the positions) occurrence of v by t_i.

A *regular tree grammar* is a tuple $G = (N, \Sigma, V, P, S)$ consisting of

– an alphabet N of *nonterminal symbols* such that $N \cap (\Sigma \cup V) = \emptyset$,
– a finite set P of *productions* of the form $A \to r$ where $A \in N$ and $r \in T_\Sigma(N \cup V)$, and
– a *start symbol* $S \in N$.

The size of G, denoted by $|G|$, is $|G| = |P|$. For any $s, t \in T_\Sigma(N \cup V)$, we write $s \Rightarrow_G t$ if there exists $A \to r \in P$ such that t can be obtained from s by replacing one occurrence of A by r. The tree language *generated* by G is $L(G) = \{t \in T_\Sigma(V) \mid S \Rightarrow_G^* t\}$. A tree language L is *recognizable* if there exists a regular tree grammar G such that $L = L(G)$. The family of all recognizable (respectively, recognizable variable-free and recognizable almost variable-free) tree languages is denoted by Rec (respectively, $\mathrm{Rec}_{\mathrm{vf}}$ and $\mathrm{Rec}_{\mathrm{avf}}$).

A *tree homomorphism* $\varphi\colon T_\Sigma(V) \to T_\Delta(Y)$ can be presented by a mapping $\varphi_V\colon V \to T_\Delta(Y)$ and mappings $\varphi_k\colon \Sigma_k \to T_\Delta(Y \cup X_k)$ for every $k \in \mathbb{N}$ as follows:

– $v\varphi = \varphi_V(v)$ for every $v \in V$, and
– $f(t_1, \ldots, t_k)\varphi = \varphi_k(f)[t_1\varphi, \ldots, t_k\varphi]$ for every $t_1, \ldots, t_k \in T_\Sigma(V)$ and $f \in \Sigma_k$.

We say that it is *normalized* if for every $f \in \Sigma_k$ there exists $n \in \mathbb{N}$ such that $\mathrm{yd}_X(\varphi_k(f)) = x_1 \cdots x_n$. Moreover, such a homomorphism φ is

– *linear* [3,7,18] (respectively, *complete* [18]) if $\varphi_k(f)$ is linear (respectively, nondeleting) in X_k for every $f \in \Sigma_k$,
– *quasi-alphabetic* [15] if it is linear and complete, $\varphi_V(v) \in Y$ for every $v \in V$, and $\varphi_k(f) \in \Delta(Y \cup X_k)$ for every $f \in \Sigma_k$,
– *symbol-to-symbol* [18] if it is quasi-alphabetic and $\varphi_k(f) \in \Delta(X_k)$ for every $f \in \Sigma_k$, and
– *alphabetic* [3,18] if it is symbol-to-symbol and normalized.

Note that our 'symbol-to-symbol' corresponds to "linear, complete, and symbol-to-symbol" of [18], and 'alphabetic' homomorphisms are sometimes called relabelings [5]. We denote by qaH, ssH, and aH the classes of all quasi-alphabetic, symbol-to-symbol, and alphabetic tree homomorphisms, respectively.

A *tree bimorphism* is a triple $B = (\varphi, L, \psi)$ where $L \subseteq T_\Gamma(Z)$ is a tree language, $\varphi\colon T_\Gamma(Z) \to T_\Sigma(V)$ and $\psi\colon T_\Gamma(Z) \to T_\Delta(Y)$ are tree homomorphisms, called input and output homomorphism, respectively. The size of B, denoted by $|B|$, is defined to be the size of a representation (e.g., by a regular tree grammar) of L. The *tree transformation defined by* B is $\tau_B = \{(t\varphi, t\psi) \mid t \in L\}$. We reserve the special variable e. The *translation defined by* B is

$$\mathrm{yd}(\tau_B) = \{(\mathrm{yd}_{V\setminus\{e\}}(s), \mathrm{yd}_{Y\setminus\{e\}}(t)) \mid (s,t) \in \tau_B\} \ .$$

Note the special treatment of e. It is never output but acts as the empty string. For all classes $\mathcal{H}_1$ and $\mathcal{H}_2$ of tree homomorphisms and every class $\mathcal{L}$ of tree languages, we denote by $\mathcal{B}(\mathcal{H}_1, \mathcal{L}, \mathcal{H}_2)$ the class of tree transformations τ_B where $B = (\varphi, L, \psi)$ with $\varphi \in \mathcal{H}_1$, $L \in \mathcal{L}$, and $\psi \in \mathcal{H}_2$. In particular, we say that a tree bimorphism (φ, L, ψ) is *quasi-alphabetic* (respectively, *symbol-to-symbol*, *alphabetic*, and *normalized*) if both φ and ψ have this property and $L \in \mathrm{Rec}$. Moreover, a bimorphism (φ, L, ψ) is *variable-free* (respectively, *almost variable-free*) if L is so.

A system $M = (Q, \Sigma, \Delta, F, R)$ is a *bottom-up tree transducer* [5,19] if

– $Q = Q_1$ is a unary ranked alphabet of *states*,
– Σ and Δ are an *input* and an *output alphabet*, respectively,
– $F \subseteq Q$ is a set of *final states*, and

– R is a finite set of *rules* of the form $f(q_1(x_1), \ldots, q_k(x_k)) \to r$ where $f \in \Sigma_k$, $q_1, \ldots, q_k \in Q$, and $r \in Q(T_\Delta(X_k))$.

The bottom-up tree transducer $M = (Q, \Sigma, \Delta, F, R)$ is *linear* (respectively, *nondeleting*) if r is linear (respectively, nondeleting) in X_k for every $f(q_1(x_1), \ldots, q_k(x_k)) \to r \in R$. The one-step *derivation relation* $\Rightarrow_M$ is defined as follows. For every $\zeta, \xi \in T_\Sigma(Q(T_\Delta))$ we have $\zeta \Rightarrow_M \xi$ if and only if there exists a rule $f(q_1(x_1), \ldots, q_k(x_k)) \to r \in R$, a position $w \in \text{pos}(\zeta)$, and $s_1, \ldots, s_k \in T_\Delta$ such that $\zeta|_w = f(q_1(s_1), \ldots, q_k(s_k))$ and $\xi = \zeta[s]_w$ with $s = r[s_1, \ldots, s_n]$. The *tree transformation computed by* M is

$$\tau_M = \{(s,t) \in T_\Sigma \times T_\Delta \mid \exists q \in F \colon s \Rightarrow_M^* q(t)\} \ .$$

3 Syntax-Directed Translation Schema

In this section, we explore the connection between quasi-alphabetic tree bimorphisms and syntax-directed translation schemata (SDTSs) [10,16,17]. It was shown in [15] that quasi-alphabetic tree bimorphisms and SDTSs are equally powerful when we consider them as translation devices for strings. This close connection is the main motivation for quasi-alphabetic tree bimorphisms [15]. Here we show that the mentioned connection already holds between SDTSs and symbol-to-symbol tree bimorphisms, a class that is smaller and well-known. Moreover, we show that simple SDTS correspond to alphabetic tree bimorphisms (also called finite-state relabelings [5]). The latter result proves that alphabetic tree bimorphisms are strictly less powerful than SDTSs.

Roughly speaking, a syntax-directed translation schema consists of two context-free grammars (CFGs) over a common set of nonterminals. A production of an SDTS is of the form $A \to u\,; w$ such that $A \to u$ and $A \to w$ are CFG productions, and additionally, the same nonterminals occur in u and w. Formally, a *syntax-directed translation schema* (SDTS) is a system $T = (N, V, Y, P, S)$ where

– N is an alphabet of *nonterminals* disjoint with $V \cup Y$,
– V and Y are an *input* and *output alphabet*, respectively,
– P is a finite set of *productions* of the form $A \to u\ ;\ w$ where $A \in N$, $u \in (N \cup V)^*$, $w \in (N \cup Y)^*$, and the nonterminals in w are a permutation of the nonterminals in u, and
– $S \in N$ is a *start symbol.*

An SDTS is called *simple* if the nonterminals occur in same order in u and w for each production $A \to u\ ;\ w$ in P. Finally, the size of T, denoted by $|T|$, is defined as the numbers of its productions (i.e., $|T| = |P|$).

To present the semantics of SDTS, we use the slightly informal notion of associated nonterminals. Whenever we apply a production in a derivation, we have to apply it to two "associated" nonterminals. This notion can easily be formalized, but we avoid this here to present the matter without excessive detail. The *translation forms* of T, which are elements of $(N \cup V)^* \times (N \cup Y)^*$, are defined inductively as follows:

- (S, S) is a translation form and the two nonterminals S are *associated.*
- If $(u_1 A u_2, w_1 A w_2)$ is a translation form in which the two explicit instances of A are associated and $A \to u \,;\, w$ is a production in P, then $(u_1 A u_2, w_1 A w_2) \Rightarrow_T (u_1 u u_2, w_1 w w_2)$ and the latter is a translation form. The nonterminals of u and w are associated exactly as they are associated in the production and the nonterminals of u_1 and u_2 are associated with those of w_1 and w_2 in the new translation form exactly as in the original one.

The *translation defined* by T is the relation

$$\tau_T = \{(u, w) \in V^* \times Y^* \mid (S, S) \Rightarrow_T^* (u, w)\} \enspace .$$

A major normal form for CFGs is the CHOMSKY normal form, but unfortunately its analogue cannot be achieved for SDTSs. However, [17] shows that we can obtain the following normal form. Note that we changed the definition slightly and demand that at most one terminal symbol occurs in each part of a production.

Definition 1. *An SDTS (N, V, Y, P, S) is in* normal form *if*

- *$u, w \in N^*$ or*
- *$u \in V \cup \{\varepsilon\}$ and $w \in Y \cup \{\varepsilon\}$ for every production $A \to u \,;\, w$ in P.*

Proposition 2 (cf. [17, Lemma 3.1]). *For every SDTS T there exists an SDTS T' in normal form such that $\tau_T = \tau_{T'}$. If T is simple, then T' can be chosen to be simple as well.*

Proof. Let $T = (N, V, Y, P, S)$ be an SDTS. We construct the SDTS $T' = (N', V, Y, P', S)$ where

- $N' = N \cup \{\underline{v} \mid v \in V\} \cup \{\overline{y} \mid y \in Y\}$ with $\underline{v}$ and $\overline{y}$ being new nonterminals,
- for every $v \in V$ and $y \in Y$ the following two rules are in P'

$$\underline{v} \to v \,;\, \varepsilon \qquad \text{and} \qquad \overline{y} \to \varepsilon \,;\, y \enspace ,$$

- and for every production of P with associated nonterminal permutation $\sigma \colon [n] \to [n]$

$$A \to u_0 A_1 u_1 \cdots A_n u_n \,;\, w_0 A_{\sigma(1)} w_1 \cdots A_{\sigma(n)} w_n$$

  where $u_0, \ldots, u_n \in V^*$, $w_0, \ldots, w_n \in Y^*$, and $A, A_1, \ldots, A_n \in N$, the following production is in P'

$$A \to \underline{u_0}\,\overline{w_0} A_1 \underline{u_1}\,\overline{w_1} \cdots A_n \underline{u_n}\,\overline{w_n} \,;\, \underline{u_0}\,\overline{w_0} A_{\sigma(1)} \underline{u_1}\,\overline{w_1} \cdots A_{\sigma(n)} \underline{u_n}\,\overline{w_n}$$

  where for every $v_1, \ldots, v_k \in V$ and $y_1, \ldots, y_m \in Y$ we define

$$\underline{v_1 \cdots v_k} = \underline{v_1} \cdots \underline{v_k} \qquad \text{and} \qquad \overline{y_1 \cdots y_m} = \overline{y_1} \cdots \overline{y_m} \enspace .$$

- The set P' does not contain any further productions.

Obviously, T' is in normal form. Moreover, it is simple if T is so. Finally, it is easy to see that $\tau_{T'} = \tau_T$. $\square$

Let us consider the complexity of the construction in the proof of Proposition 2. Clearly, the number of productions of T' is $|P| + |V| + |Y|$. Thus, the size of T' is $|T| + |V| + |Y|$. It is a reasonable assumption that for every $v \in V$ (respectively, $y \in Y$) there is at least one production in P in which v (respectively, y) occurs (otherwise we can simply drop the offending v or y). Consequently, $|V| + |Y| \leqslant 2|T|$ and $|T'| \in O(|T|)$, which proves that the size of T' is linear in the size of T.

Before we proceed with the mentioned connection between SDTSs and quasi-alphabetic tree bimorphisms, let us recall the well-known link between SDTSs and simple SDTSs.

Theorem 3 (see [16, Theorem 2]). *The class of all translations defined by simple SDTSs is properly contained in the class of all translations defined by SDTSs.*

In [15, Theorem 5.7] it was shown that SDTSs and quasi-alphabetic tree bimorphisms define the same (string) translations. The correspondence is very close since the derivations of an SDTS can be obtained from the tree transformation of the corresponding bimorphism. However, we will show that this correspondence already exists between SDTSs and symbol-to-symbol tree bimorphisms. Moreover, we will show that simple SDTSs and alphabetic tree bimorphisms define the same class of translations. Let us first consider the direction in which we construct a tree bimorphism for an SDTS. Since the only difference between quasi-alphabetic and symbol-to-symbol tree bimorphisms is in their homomorphisms, let us reconsider the construction of those homomorphisms from [15, Sect. 5]. We only change the behavior on productions that only have terminal symbols on the right-hand sides.

Definition 4. *Let $T = (N, V, Y, P, S)$ be an SDTS in normal form. For every production $p = (A \to u\,; w) \in P$ let $\mathrm{rk}(p) = n$ be such that $u \in N^n V^*$. This turns the set P into a ranked alphabet. Moreover, let $P' = \bigcup_{k \geqslant 1} P_k$. We construct the homomorphisms*

$$\varphi\colon T_{P'}(P_0) \to T_{P'}(V') \qquad \textit{and} \qquad \psi\colon T_{P'}(P_0) \to T_{P'}(Y')$$

where $V' = V \cup \{e\}$ and $Y' = Y \cup \{e\}$ as follows: Let $p = (A \to u\,; w) \in P$.

– *If $p \in P_0$, then*

$$\varphi_{P_0}(p) = \begin{cases} e & \textit{if } u = \varepsilon \\ u & \textit{if } u \in V \end{cases} \qquad \textit{and} \qquad \psi_{P_0}(p) = \begin{cases} e & \textit{if } w = \varepsilon \\ w & \textit{if } w \in V. \end{cases}$$

– *If $p \in P'_k$, then $\varphi_k(p) = p(x_1, \ldots, x_k)$ and $\psi_k(p) = p(x_{\sigma(1)}, \ldots, x_{\sigma(k)})$ where $\sigma\colon [k] \to [k]$ is the nonterminal permutation of p.*

By Proposition 2 we can assume normal form without loss of generality. Our construction is very similar to the construction of [15] if we restrict ourselves to SDTSs in normal form. The constructed homomorphisms φ and ψ are symbol-to-symbol, and if T is simple, then they are alphabetic. Thus, a minor modification of a principal result of [15, Sect. 5] yields our first result.

Lemma 5 (cf. [15, Prop. 5.5]). *For every SDTS T, there exists a symbol-to-symbol tree bimorphism B such that $\mathrm{yd}(\tau_B) = \tau_T$. If T is simple, then B can be chosen to be alphabetic.*

Let us consider the size of the resulting bimorphism. The construction in the proof of [15, Prop. 5.5] yields a local center tree language $L \subseteq T_P$ (the symbols are productions and their rank is determined by the number of nonterminals as in Definition 4). Roughly speaking, the language L contains all legal derivations (i.e., the nonterminals in productions match). For this tree language L we can construct the following regular tree grammar $G = (N, P, \emptyset, P', S)$, where for every production $p \in P_n$ (note that we assume that T is in normal form) with associated nonterminal permutation $\sigma\colon [n] \to [n]$ such that

$$p = A \to A_1 \cdots A_n v \;;\; A_{\sigma(1)} \cdots A_{\sigma(n)} y$$

for some $A, A_1, \ldots, A_n \in N$, $v \in V \cup \{\varepsilon\}$, and $y \in Y \cup \{\varepsilon\}$, the set P' contains the production $A \to p(A_1, \ldots, A_n)$. All productions of P' are constructed in this manner. Obviously, the size of G is the same as the size of T. Thus, the size of the bimorphism B constructed in Lemma 5 is linear in the size of the input SDTS T.

For the converse, we can again reconsider [15]. In [15, Prop. 5.6] it is proved that for every quasi-alphabetic tree bimorphism B there exists an SDTS T such that $\tau_T = \mathrm{yd}(\tau_B)$. Clearly, every symbol-to-symbol bimorphism is quasi-alphabetic, and moreover, it is an easy exercise to confirm that the SDTS constructed in [15, Prop. 5.6] is simple if B is alphabetic. Our minor modification of the definition of the translation defined by a tree bimorphism (the special treatment of the symbol e) requires only a minor change in the proof of [15, Prop. 5.6].

Lemma 6. *For every symbol-to-symbol tree bimorphism B, there exists an SDTS T such that $\tau_T = \mathrm{yd}(\tau_B)$. If B is alphabetic, then T can be chosen to be simple.*

In the construction of [15, Prop. 5.6] the center tree language is represented as a local tree language, but in the same spirit the construction can be done if the center tree language is represented by a regular tree grammar. Every production of the tree grammar yields a production of the constructed SDTS. Thus, the size of the constructed SDTS is linear in the size of the input bimorphism (see, for example, [20, Theorem 4] on how to handle the regular tree grammar).

This yields the following relations between SDTSs and symbol-to-symbol tree bimorphisms. It was shown in [15] that the class of all translations defined by SDTSs coincides with the class of all translations defined by quasi-alphabetic tree bimorphisms. Here we sharpen this result.

Theorem 7. *The class of translations defined by arbitrary (respectively, simple) SDTSs coincides with the class of translations defined by symbol-to-symbol (respectively, alphabetic) tree bimorphisms.*

If we consider Theorems 3 and 7 together, we obtain that the class of all translations defined by alphabetic tree bimorphisms is properly contained in the class of all translations defined by symbol-to-symbol tree bimorphisms.

4 Closure under Composition

In this section we reconsider the problem of closure under composition for the class of tree transformations defined by quasi-alphabetic tree bimorphisms. It was shown in [15] that if we restrict ourselves to quasi-alphabetic tree bimorphisms with a variable-free center tree language, then the resulting class of tree transformations is closed under composition. Here we want to extend this result to include variables. The following proposition is trivial, but indicates why closure under composition is possible whereas closure under intersection fails [21].

Proposition 8. *For every quasi-alphabetic bimorphism B, there exist a quasi-alphabetic bimorphism B_1 with a normalized input homomorphism and a quasi-alphabetic bimorphism B_2 with a normalized output homomorphism such that $\tau_B = \tau_{B_1} = \tau_{B_2}$. If B is variable-free (almost variable-free, respectively), then B_1 and B_2 can be chosen such that they are variable-free (almost variable-free, respectively).*

So we showed that one homomorphism of a quasi-alphabetic bimorphism can always be normalized. As a final step we try to get rid of the variables as much as possible.

Lemma 9. $\mathcal{B}(\mathrm{qaH}, \mathrm{Rec}_{\mathrm{avf}}, \mathrm{qaH}) = \mathcal{B}(\mathrm{qaH}, \mathrm{Rec}, \mathrm{qaH})$

Proof. Let $B = (\varphi, L, \psi)$ be a quasi-alphabetic tree bimorphism with $L \subseteq T_\Gamma(Z)$. Moreover, let Y be a set and $h \colon Z \to Y$ be a bijection. Finally, let $M = (Q, \Gamma', \Omega', Q, R)$ be the linear bottom-up tree transducer with

- $Q = Y \cup \{\star\}$,
- $\Gamma'_k = \Gamma_k$ for every $k \geqslant 1$ and $\Gamma'_0 = \Gamma_0 \cup Z$,
- $\Omega'_k = \{t \in \Gamma(Q) \mid k = |t|_\star\}$ for every $k \geqslant 1$ and $\Omega'_0 = \Gamma_0 \cup Z$.
- The set R of rules is given as follows:
  - For every $z \in Z$, let $z \to q(z)$ be a rule of R where $q = h(z)$.
  - For every $f \in \Gamma_k$ and $q_1, \ldots, q_k \in Q$, let

    $$f(q_1(x_1), \ldots, q_k(x_k)) \to \star(\omega(x_{i_1}, \ldots, x_{i_n}))$$

    be a rule of R where $\omega = f(q_1, \ldots, q_k)$, $i_1 < \cdots < i_n$, and $\{i_1, \ldots, i_n\} = \{i \in [k] \mid q_i = \star\}$.

Let $L' = \tau_M(L)$ be the image of L under τ_M. Clearly, L' is almost variable-free, and by [3, Lemma IV.6.5], the tree language L' is recognizable. We construct the bimorphism $B' = (\varphi', L', \psi')$ such that $\varphi'_Z = \varphi_Z$, $\psi'_Z = \psi_Z$, and

$$\varphi'_k(t) = \varphi(t[\star \leftarrow (x_1, \ldots, x_k)]) \quad \text{and} \quad \psi'_k(t) = \psi(t[\star \leftarrow (x_1, \ldots, x_k)])$$

for every $t \in \Omega_k$. Clearly, φ' and ψ' are quasi-alphabetic. Thus, B' is an almost variable-free quasi-alphabetic bimorphism. Note that M is deterministic and total and thus τ_M is a mapping [5]. Finally, let us prove that $\tau_{B'} = \tau_B$. For this, we prove that $t\varphi = \tau_M(t)\varphi'$ and $t\psi = \tau_M(t)\psi'$ for every $t \in T_\Gamma(Z)$. Clearly, it is sufficient to prove the former statement since the argument is totally symmetric. First, let $t \in Z$. Then

$$t\varphi = \tau_M(t)\varphi = \tau_M(t)\varphi' \enspace .$$

Now, let $t = f(t_1, \ldots, t_k)$ for some $f \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Gamma(Z)$. Moreover, for every $i \in [k]$, let $q_i = h(t_i)$ if $t_i \in Z$ and $q_i = \star$ otherwise. Then

$$\begin{aligned}
\tau_M(f(t_1, \ldots, t_k))\varphi' &= \omega(\tau_M(t_{i_1}), \ldots, \tau_M(t_{i_n}))\varphi' \\
&= \varphi'_n(\omega)[\tau_M(t_{i_1})\varphi', \ldots, \tau_M(t_{i_n})\varphi'] \\
&= \varphi(f(q_1, \ldots, q_k)[\star \leftarrow (x_1, \ldots, x_n)])[t_{i_1}\varphi, \ldots, t_{i_n}\varphi] \\
&= f(q_1, \ldots, q_k)[\star \leftarrow (t_{i_1}, \ldots, t_{i_n})]\varphi \\
&= f(t_1, \ldots, t_k)\varphi
\end{aligned}$$

with $\omega = f(q_1, \ldots, q_k)$, $i_1 < \cdots < i_n$, and $\{i_1, \ldots, i_n\} = \{i \in [k] \mid q_i = \star\}$. This completes the proof. □

It is proved in [15, Theorem 7.4] that $\mathcal{B}(\mathrm{qaH}, \mathrm{Rec}_{\mathrm{vf}}, \mathrm{qaH})$ is closed under composition. Let us take another look at composition closure results. First, we point out why it is far easier to prove the closure only for tree transformations defined by variable-free or almost variable-free quasi-alphabetic bimorphisms.

Lemma 10. *Let $\varphi\colon T_\Sigma(V) \to T_\Gamma(Z)$ and $\psi\colon T_\Delta(Y) \to T_\Gamma(Z)$ be normalized quasi-alphabetic tree homomorphisms, and let $s \in T_\Sigma \cup V$ and $t \in T_\Delta \cup Y$. If $s\varphi = t\psi$, then* $\mathrm{pos}(s) = \mathrm{pos}(t)$.

Proof. First, let $s \in V$. Then $s\varphi \in Z$. Since $s\varphi = t\psi$, it follows that $t \in Y$ and hence $\mathrm{pos}(s) = \mathrm{pos}(t)$. Second, let $s = f(s_1, \ldots, s_k)$ for some $f \in \Sigma_k$ and $s_1, \ldots, s_k \in T_\Sigma$. Then $s\varphi = \varphi_k(f)[s_1\varphi, \ldots, s_k\varphi] = t\psi$. Since φ and ψ are quasi-alphabetic, we have $s_i\varphi \notin Z$ for every $i \in [k]$. If we additionally take into account that $s\varphi = t\psi$, then we can conclude that $t = g(t_1, \ldots, t_k)$ for some $g \in \Delta_k$ and $t_1, \ldots, t_k \in T_\Delta$. Moreover, since φ and ψ are normalized, it also follows that $\varphi_k(f) = \psi_k(g)$. Using the induction hypothesis, we thus obtain $\mathrm{pos}(s) = \mathrm{pos}(t)$. □

The previous proposition essentially states that all almost variable-free trees with the same image under two normalized quasi-alphabetic tree homomorphisms can be paired up in a product data structure $T_{\Sigma \times \Delta}(V \times Y)$. Let us plug the statements together and establish the relation to closure under composition.

Lemma 11. $\mathcal{B}(\mathrm{qaH}, \mathrm{Rec}, \mathrm{qaH})$ *is closed under composition if*

$$\{t \in T_\Omega \cup V \mid t\varphi = t\psi\} \tag{†}$$

is a recognizable tree language for every ranked alphabet Ω, set V of variables, and pair (φ, ψ) of normalized quasi-alphabetic tree homomorphisms.

Proof. Let $B_1 = (\varphi_1, L_1, \psi_1)$ and $B_2 = (\varphi_2, L_2, \psi_2)$ be quasi-alphabetic bimorphisms. Without loss of generality, let B_1 and B_2 be almost variable-free by Lemma 9. Moreover, suppose that ψ_1 and φ_2 are normalized by Proposition 8. Let

$$\begin{aligned}\tau &= \tau_{B_1} \mathbin{;} \tau_{B_2} \\ &= \{(s, r) \mid \exists t\colon (s, t) \in \tau_{B_1}, (t, r) \in \tau_{B_2}\} \\ &= \{(t\varphi_1, r\psi_2) \mid t \in L_1, r \in L_2, t\psi_1 = r\varphi_2\} \ .\end{aligned}$$

Since $t\psi_1 = r\varphi_2$, it follows by Lemma 10 that $\mathrm{pos}(t) = \mathrm{pos}(r)$. Hence the quantified t and r in the last displayed equation can be stored in a tree $s \in T_{\Sigma\times\Delta} \cup (V \times Y)$ such that $s\pi_1 = t$ and $s\pi_2 = r$ where π_1 and π_2 are the usual projections to the first and second component.

Let $T = T_{\Sigma\times\Delta} \cup (V \times Y)$. We can continue the displayed equations by

$$\begin{aligned}\tau &= \{(t\pi_1\varphi_1, t\pi_2\psi_2) \mid t \in T, t\pi_1 \in L_1, t\pi_2 \in L_2, t\pi_1\psi_1 = t\pi_2\varphi_2\} \\ &= \{(t\pi_1\varphi_1, t\pi_2\psi_2) \mid t \in \pi_1^{-1}(L_1) \cap \pi_2^{-1}(L_2) \cap L\}\end{aligned}$$

where $L = \{t \in T \mid t\pi_1\psi_1 = t\pi_2\varphi_2\}$. It is easily seen that the tree homomorphisms $\pi_1\varphi_1$, $\pi_2\psi_2$, $\pi_1\psi_1$, and $\pi_2\varphi_2$ are quasi-alphabetic. Moreover, $\pi_1\psi_1$, and $\pi_2\varphi_2$ are normalized. By assumption, L is thus recognizable, and $\pi_1^{-1}(L_1)$ and $\pi_2^{-1}(L_2)$ are recognizable by [3, Theorem II.4.18]. Consequently, $\pi_1^{-1}(L_1) \cap \pi_2^{-1}(L_2) \cap L$ is recognizable by [3, Theorem II.4.2], and hence, $\tau \in \mathcal{B}(\mathrm{qaH}, \mathrm{Rec}, \mathrm{qaH})$, which completes the proof. □

So whenever the equality sets [the sets (†) in Lemma 11] are recognizable, we can construct a quasi-alphabetic bimorphism that computes the composition of two given quasi-alphabetic bimorphisms. It remains to prove that the equality sets are recognizable (the premise of Lemma 11).

Lemma 12. *Let $\varphi\colon T_\Omega(Z) \to T_\Sigma(V)$ and $\psi\colon T_\Omega(Z) \to T_\Sigma(V)$ be normalized quasi-alphabetic tree homorphisms. Then $L = \{t \in T_\Omega \cup Z \mid t\varphi = t\psi\}$ is recognizable.*

Proof. We construct the regular tree grammar $G = (\{S\}, \Omega', P, S)$ where

– $\Omega'_k = \Omega_k$ for every $k \geqslant 1$ and $\Omega'_0 = \Omega_0 \cup Z$, and
– $P = P_1 \cup P_2$ with

$$\begin{aligned}P_1 &= \{S \to z \mid z \in Z, z\varphi = z\psi\} \\ P_2 &= \{S \to f(S, \dots, S) \mid f \in \Omega_k, \varphi_k(f) = \psi_k(f)\} \ .\end{aligned}$$

Then $L = L(G) \cap (T_\Omega \cup Z)$, which is recognizable [3, Theorem II.4.2]. □

We are now ready to state our main result. Let Loc (respectively, $\mathrm{Loc}_{\mathrm{vf}}$) be the class of all local (respectively, local variable-free) tree languages [3]. Note that [15, Theorem 7.4] proves that $\mathcal{B}(\mathrm{qaH}, \mathrm{Loc}_{\mathrm{vf}}, \mathrm{qaH})$ is closed under composition. Since every recognizable tree language is the image of a local tree language under an alphabetic tree homomorphism [3, Theorem II.9.5], we immediately obtain

$$\mathcal{B}(\mathrm{qaH}, \mathrm{Loc}_{\mathrm{vf}}, \mathrm{qaH}) = \mathcal{B}(\mathrm{qaH}, \mathrm{Rec}_{\mathrm{vf}}, \mathrm{qaH})$$
$$\mathcal{B}(\mathrm{qaH}, \mathrm{Loc}, \mathrm{qaH}) = \mathcal{B}(\mathrm{qaH}, \mathrm{Rec}, \mathrm{qaH}) \enspace .$$

The closure of $\mathcal{B}(\mathrm{qaH}, \mathrm{Rec}_{\mathrm{vf}}, \mathrm{qaH})$ is thus proved in [15] and here we prove it for $\mathcal{B}(\mathrm{qaH}, \mathrm{Rec}, \mathrm{qaH})$. Note that our approach is slightly different.

Theorem 13 (cf. [15, Theorem 7.4]). $\mathcal{B}(\mathrm{qaH}, \mathrm{Rec}, \mathrm{qaH})$ *is closed under composition.*

Proof. Follows directly from Lemmata 11 and 12. □

Acknowledgements

The authors are grateful to Magnus Steinby for fruitful discussions regarding Section 3. In addition, the authors appreciate the effort of the reviewers, who pointed out several inconsistencies and mistakes.

References

1. Knight, K., Graehl, J.: An overview of probabilistic tree transducers for natural language processing. In: Gelbukh, A. (ed.) CICLing 2005. LNCS, vol. 3406, pp. 1–24. Springer, Heidelberg (2005)
2. Knight, K.: Capturing practical natural language transformations. Machine Translation 21(2), 121–133 (2007)
3. Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó, Budapest (1984)
4. Nivat, M., Podelski, A. (eds.): Tree Automata and Languages. North-Holland, Amsterdam (1992)
5. Engelfriet, J.: Bottom-up and top-down tree transformations: A comparison. Math. Syst. Theory 9(3), 198–231 (1975)
6. Arnold, A., Dauchet, M.: Morphismes et bimorphismes d'arbres. Theor. Comput. Sci. 20(1), 33–93 (1982)
7. Bozapalidis, S.: Alphabetic tree relations. Theor. Comput. Sci. 99(2), 177–211 (1992)
8. Takahashi, M.: Primitive transformations of regular sets and recognizable sets. In: Proc. ICALP, pp. 475–480. North-Holland, Amsterdam (1972)
9. Steinby, M.: On certain algebraically defined tree transformations. In: Proc. Algebra, Combinatorics and Logic in Computer Science. Colloquia Mathematica Societatis János Bolyai, vol. 42, pp. 745–764. North-Holland, Amsterdam (1986)

10. Irons, E.T.: A syntax directed compiler for ALGOL 60. Comm. ACM 4(1), 51–55 (1961)
11. Shieber, S.M., Schabes, Y.: Synchronous tree-adjoining grammars. In: Proc. COLING, pp. 253–258. ACL (1990)
12. Satta, G., Peserico, E.: Some computational complexity results for synchronous context-free grammars. In: Proc. HLT/EMNLP, pp. 803–810. ACL (2005)
13. Shieber, S.M.: Synchronous grammars as tree transducers. In: Proc. TAG+7, pp. 88–95 (2004)
14. Aho, A.V., Ullman, J.D.: Parsing. The Theory of Parsing, Translation, and Compiling, vol. 1. Prentice-Hall, Englewood Cliffs (1972)
15. Steinby, M., Tîrnăucă, C.I.: Defining syntax-directed translations by tree bimorphisms. Theor. Comput. Sci. (to appear, 2009), http://dx.doi.org/10.1016/j.tcs.2009.03.009
16. Aho, A.V., Ullman, J.D.: Properties of syntax directed translations. J. Comput. Syst. Sci. 3(3), 319–334 (1969)
17. Aho, A.V., Ullman, J.D.: Syntax directed translations and the pushdown assembler. J. Comput. Syst. Sci. 3(1), 37–56 (1969)
18. Comon-Lundh, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Löding, C., Tison, S., Tommasi, M.: Tree automata—techniques and applications (2007), http://tata.gforge.inria.fr/
19. Thatcher, J.W.: Tree automata: An informal survey. In: Currents in the Theory of Computing, pp. 143–172. Prentice-Hall, Englewood Cliffs (1973)
20. Maletti, A.: Compositions of extended top-down tree transducers. Inf. Comput. 206(9-10), 1187–1196 (2008)
21. Maletti, A., Tîrnăucă, C.I.: Properties of quasi-alphabetic tree bimorphisms (unpublished 2009), http://arxiv.org/abs/0906.2369v1

Polynomial Interpolation of the k-th Root of the Discrete Logarithm

Gerasimos C. Meletiou

A.T.E.I. of Epirus
P.O. Box 110, Arta 47100, Greece
gmelet@teiep.gr

Abstract. In the present study the problem of efficient computation of the k-th root of the Discrete Logarithm is investigated. Lower bounds on the degree of interpolation polynomials of the root of the Discrete Logarithm for subsets of given data are obtained. These results support the assumption of hardness of the k-th root of the discrete logarithm.

1 Introduction

The k-th root of the discrete logarithm is used as a one-way function in several cryptographic applications. In particular in Public Verifiable Secret Sharing Schemes, Group Signature Schemes, Electronic Cash, Anonymity Control in Multi-bank E-Cash System, Offline Electronic Cash Systems (c.f. [1], [4], [5], [6], [7], [10], [11], [12], [20], [22], [23]). These considerations are related to applications of the Double Discrete Logarithm [16].

Let $G = \langle g \rangle$ be a cyclic group of order t and Y be an element of G. The discrete logarithm of Y with respect to the base g is the smallest positive integer x such that $g^x = Y$. A k-th root of the discrete logarithm of $Y \in G$ to the base g is an integer x satisfying:

$g^{\left(x^k\right)} = Y$ if such an x exists.

It is evident that existence and uniqueness of the k-th root of the discrete logarithm are not guaranteed. In the case $\left|\left\{x : g^{\left(x^k\right)} = Y\right\}\right| \geq 2$, we investigate branches of the k-th root of the discrete logarithm.

Parameters G, t and g can be chosen in advance in such a way that computing discrete logarithms to the base g is infeasible. In addition t can be chosen in such a way that obtaining k-th roots modulo t is hard to be determined.

In most of the applications (see [6], [7], [11], [12]) t is an RSA modulus that is $t = p.q$,where p and q are big primes and the factorization of t is unknown. Therefore deriving the k-th roots is not feasible. In [23] the problem has been studied for the case $t = p^2.q$,where p and q are primes. Concerning k it may be equal to two (square root), also, k could be equal to e, that is the encryption exponent of the RSA (root of odd order). A message m is encrypted as $m^e \pmod t$ and then $g^{(m^e)}$ becomes public. Recovering m is the same as computing the e-th root of the discrete logarithm.

The goal of this paper is to investigate the possibility to obtain a solution of the 'k-th root of the discrete logarithm problem' by applying interpolating polynomials. We derive lower bounds on the degrees of these polynomials. We show that such polynomials have indeed a very large degree, supporting the assumption of the hardness of the problem when the parameters are properly chosen.

For polynomial representations of other cryptographic functions some similar results have been obtained. The investigations for our study regarding the k-th root of the discrete logarithm in this paper are motivated from a number of results. In [14], [15], [17], [18] exact polynomial representations of the discrete logarithm in a finite field has been deduced. However in [8], [19], [21], [25] it was proved that there are no low degree interpolation polynomials of the discrete logarithm for a large set of given data. Concerning other cryptographic functions, c.f. [9], [13], [24], lower bounds on the degrees of polynomials representing the Diffie-Hellman mapping are obtained. Also in [3] exact formulas for polynomials representing the Lucas logarithm are deduced and lower bounds on the degree of interpolation polynomials of the Lucas logarithm for subsets of given data have been proved. In [16] the double discrete logarithm is addressed.

2 Roots of Odd Order

In what follows the exponent k is odd and relatively prime to $\varphi(t)$ and, of course, the k-th root function is a bijection mapping. The main motivation for this study stems from RSA. In this case k is the encryption exponent e.

Theorem 1. *Let p be a prime, $g \in Z_p^*$, $|\langle g\rangle| = t$ and let $k > 0$ be an integer such that* $\gcd(k, \varphi(t)) = 1$. *Let $S \subseteq Z_t^*$ be a subset of order $|S| = \varphi(t) - s$. Suppose the existence of a polynomial $F(X) \in Z_p[X]$ such that $F(g^{x^k}) = x$ for all $x \in S$. Then* $\deg(F) \geq \frac{\varphi(t)-2s}{2}$.

Proof. The condition $\gcd(k, \varphi(t)) = 1$ implies the existence and uniqueness of the k-th root of every element of Z_t^*.

Consider the set $R = \{x : x \in S, t - x \in S\}$.

Obviously $|R| \geq \varphi(t) - 2s$. For all $x \in R, Y \in Z_p$ and $Y = g^{x^k}$it is true that $\frac{1}{Y} = g^{(t-x)^k}$.

Therefore one has the equation $F(Y) + F(\frac{1}{Y}) = t$.

The polynomial $h(Y) = Y^n \left(F(Y) + F\left(\frac{1}{Y}\right) - t\right)$ is not identical to the zero-polynomial in $Z_p[X]$. In order to verify it one can set $F(Y) = a_0 + ... + a_n Y^n$ and get $h(0) = a_n \neq 0$.Therefore $h(Y)$ has degree $2n$ and at least $|R|$ zeros. Thus one obtains $\deg(F) = n = \frac{\deg(h)}{2} \geq \frac{|R|}{2} \geq \frac{\varphi(t)-2s}{2}$. The proof is complete.

3 Square Roots of Discrete Logarithms

In the following Theorems 2 and 3 the order of the group $\langle g\rangle$ is prime. In Theorem 4 the order is the product of two primes (RSA-modulus),that is $N = p.q$. However the factorization of N is not known. In the followings we denote by QR_t the set of all quadratic residues modulo t. It is a fact that QR_t is a subgroup of Z_t^*.

A function $b : QR_t \to Z_t$ satisfying $(b(x))^2 = x$ will be called a branch of the square root. In other words the function b assigns to the quadratic residue x one of its square roots. It is evident that b is a bijection mapping from QR_t onto Im_b (that is the image of the function b). Define $A = Im_b$, $B = Z_t^* \setminus Im_b$.When t is a prime, $|A| = |B| = \frac{\varphi(t)}{2}, x \in A$ if and only if $-x \in B$ for all $x \in Z_t^*$. The branch of the square root of the discrete logarithm can be defined in a similar way.

Theorem 2. *Let p be a prime, $g \in Z_p^*$, $|\langle g\rangle| = t$ and t be also a prime. Consider the subset:*
$S \subseteq Z_t^*, |S| = \frac{t-1}{2} - s \leq \frac{t-1}{2}$.
Let $F(X) \in Z_p[X]$ be a polynomial satisfying $F(g^{x^2}) = x$ for all $x \in S$.Then: $deg(F) \geq \frac{t-1-4s}{32}$.

Proof. Let $x \in S$. Since $g^{(x)^2} = g^{(-x)^2}$ one has $-x \notin S$. F can be extended to a branch b of the square root of the discrete logarithm. We can find $A, B \subseteq Z_t^*$, such that $A \cup B = Z_t^*$, $A \cap B = \varnothing$, $S \subseteq A = Im_b$, $|A| = |B| = \frac{t-1}{2}$.

The set S can be decomposed to the following form as $S = R_1 \cup W_1 \cup R_2 \cup W_2$ (disjoint union),where
$R_1 = \{x : x \in S, 2x \in A \text{ and } 2x \in S\}$,
$W_1 = \{x : x \in S, 2x \in A \text{ and } 2x \in A \setminus S\}$,
$R_2 = \{x : x \in S, 2x \in B \text{ and } -2x \in S\}$,
$W_2 = \{x : x \in S, 2x \in B \text{ and } -2x \in A \setminus S\}$
The set $W_1 \cup W_2$ has at most s elements. Therefore, the set $R = R_1 \cup R_2$ has at least
$|S| - s = \frac{t-1}{2} - 2s$ elements.
It is follows that $Y = g^{x^2}$ meaning $Y^4 = g^{(2x)^2} = g^{(-2x)^2}$. For every $x \in R_1$:
$F(Y^4) = F\left(g^{(2x)^2}\right) = 2x - \varepsilon = 2F(Y) - \varepsilon$, $\varepsilon \in \{0, t\}$.
For every $x \in R_2$:
$F\left(Y^4\right) = F\left(g^{(-2x)^2}\right) = \varepsilon + t - 2x = \varepsilon + t - 2F(Y)$, $\varepsilon \in \{0, t\}$
Let $x \in R$ where $= R_1 \cup R_2$. Consider the polynomials:
$h_1(Y) = F(Y^4) - 2F(Y)$, in the case $x \in R_1$ and $\varepsilon = 0$,
$h_2(Y) = F(Y^4) - 2F(Y) + t$, in the case $x \in R_1$ and $\varepsilon = t$,
$h_3(Y) = F(Y^4) + 2F(Y) - t$, in the case $x \in R_2$ and $\varepsilon = 0$,
$h_4(Y) = F(Y^4) + 2F(Y) - 2t$, in the case $x \in R_2$ and $\varepsilon = t$.
It is easily follows that no one of these polynomials is identical to the zero polynomial and that
$\deg(h_i) = 4n$ for $i = 1, 2, 3, 4$.
At least one of them has at least $\frac{1}{4}|R|$ roots. It is follows:
$\deg(F) = n = \frac{\deg(h)}{4} \geq \frac{1}{16}|R| \geq \frac{1}{16}(\frac{t-1}{2} - 2s) = \frac{t-1-4s}{32}$.

Theorem 3. *Let p be a prime, $g \in Z_p^*$, $|\langle g\rangle| = t$, t is also a prime, $t \equiv 3 (\mod 4)$.*

Consider the subset $S \subseteq QR_t \subseteq \{1, 2, ..., t-1\}$*,* $|S| = \frac{t-1}{2} - s \le \frac{t-1}{2}$*. The elements of* S *are quadratic residues.*

Let $F(X) \in Z_p[X]$ *be a polynomial satisfying:* $F\left(g^{x^2}\right) = x$ *for all* $x \in S$*. Then*

$\deg(F) \ge \max\left\{\frac{t-1-4s}{32}, \frac{t-1-4s}{2v^3}\right\}$

where v *is the smallest quadratic residue* $\mod t$*,* $v \in \{2, 3, 4\}$*.*

Proof. The assumption $t \equiv 3 \pmod 4$ implies that for all $x \in Z_t^*$, the element x is a quadratic residue if and only if $-x$ is not a quadratic residue. Thus the square root function becomes a bijection.

Let v be the smallest quadratic residue $\mod t$, $v \in \{2, 3, 4\}$ Define $R = \{x : x \in S, vx \in S\}$, $|R| \ge \frac{t-1}{2} - 2s$. For all $x \in R$ it is follow:

$F\left(g^{(vx)^2}\right) = F\left(Y^{v^2}\right) = v.x - j.t, \qquad \text{for } j = 0, 1, ..., v-1.$

Consider the v polynomials in $Z_p[X]$:

$h_j(Y) = F\left(Y^{v^2}\right) - v.F(Y) + j.t, \qquad j = 0, 1, ..., v-1.$

None of these polynomials is identical to the zero polynomial and $\deg(h_j) = v^2.n$.

On the other hand at least one of these polynomials has at least $\frac{|R|}{v}$ zeros.

Therefore:

$\deg(F) = n = \frac{\deg(h_j)}{v^2} \ge \frac{1}{v^2}.\frac{|R|}{v} \ge \frac{\frac{t-1}{2} - 2.s}{v^3} = \frac{t-1-4s}{2v^3}$.

From Theorem 2 one gets

$\deg(F) \ge \max\left\{\frac{t-1-4s}{32}, \frac{t-1-4s}{2v^3}\right\}$.

Theorem 4. *Let* r *be a prime number,* $g \in Z_r^*$*,* $|\langle g\rangle| = N$*,* $N = p.q$ *an RSA modulus. In addition, we assume that* $p \equiv q \equiv 3 \pmod 4$*. Consider the subset* $S \subseteq QR_N \subset Z_N^*$*,* $|S| = \frac{\varphi(N)}{4} - s \le \frac{\varphi(N)}{4}$*. The elements of* S *are quadratic residues. Let* $F(X) \in Z_r[X]$ *be a polynomial satisfying:*

$F\left(g^{x^2}\right) = x$ *for all* $x \in S$*. Then*

$\deg(F) \ge \frac{\varphi(N) - 8s}{4v^3}$*, where* v *is the smallest quadratic residue* $\mod N$*,* $v \in \{2, 3, 4\}$*.*

Proof. The assumption $p \equiv q \equiv 3 \pmod 4$ implies that all quadratic residues in Z_N^* have exactly one root which is also a quadratic residue $\mod N$, that is the square root function is a bijection. The rest of the proof follows a similar argument as in the proof of Theorem 3.

References

1. Ateniese, G., Tsudik, G.: Some open issues and new directions in group signatures. In: Franklin, M. (ed.) FCT 1999. LNCS, vol. 1684, pp. 196–211. Springer, Heidelberg (1999)
2. Adelmann, C., Winterhof, A.: Interpolation of functions related to the integer factoring problem. In: Ytrehus, Ø. (ed.) WCC 2005. LNCS, vol. 3969, pp. 144–154. Springer, Heidelberg (2006)

3. Aly, H., Winterhof, A.: Polynomial representations of the Lucas logarithm. Finite Fields Appl. 12(3), 413–424 (2006)
4. Bresson, E., Stern, J.: Efficient Revocation in Group Signatures. In: Kim, K.-c. (ed.) PKC 2001. LNCS, vol. 1992, pp. 190–206. Springer, Heidelberg (2001)
5. Bussard, L., Molva, R., Roudier, Y.: History-based signature or how to trust anonymous documents. In: Jensen, C., Poslad, S., Dimitrakos, T. (eds.) iTrust 2004. LNCS, vol. 2995, pp. 78–92. Springer, Heidelberg (2004)
6. Camenisch, J.L.: Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem, Doctoral Dissertation, ZURICH (1998)
7. Camenisch, J.L., Stadler, M.A.: Efficient group signature schemes for large groups. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (1997)
8. Coppersmith, D., Shparlinski, I.: On polynomial approximation of the discrete logarithm and the Diffie-Hellman mapping. J. Cryptology 13(3), 339–360 (2000)
9. El Mahassni, E., Shparlinski, I.E.: Polynomial representations of the Diffie-Hellman mapping. Bull. Austral. Math. Soc. 63, 467–473 (2001)
10. Jeong, I.R., Lee, D.-H.: Anonymity control in multi-bank E-cash system. In: Roy, B., Okamoto, E. (eds.) INDOCRYPT 2000. LNCS, vol. 1977, pp. 104–116. Springer, Heidelberg (2000)
11. Konoma, C., Mambo, M., Shizuya, H.: The computational difficulty of solving cryptographic primitive problems related to the discrete logarithm problem. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E88-A(1), 81–88 (2005)
12. Lysyanskaya, A., Ramzan, Z.: Group blind digital signatures: A scalable solution to electronic cash. In: Hirschfeld, R. (ed.) FC 1998. LNCS, vol. 1465, pp. 184–197. Springer, Heidelberg (1998)
13. Meidl, W., Winterhof, A.: A polynomial representation of the Diffie-Hellman mapping. Appl. Algebra Engrg. Comm. Comput. 13, 313–318 (2002)
14. Meletiou, G.C.: Explicit form for the discrete logarithm over the field GF(p, k). Arch. Math (Brno) 29, 25–28 (1993)
15. Meletiou, G.C., Mullen, G.L.: A note on discrete logarithms in finite fields. Appl. Algebra Engrg. Comm. Comput. 3(1), 75–78 (1992)
16. Meletiou, G.C., Winterhof, A.: Interpolation of the double discrete logarithm. In: von zur Gathen, J., Imaña, J.L., Koç, Ç.K. (eds.) WAIFI 2008. LNCS, vol. 5130, pp. 1–10. Springer, Heidelberg (2008)
17. Mullen, G.L., White, D.: A polynomial representation for logarithms in GF(q). Acta Arith. 47(3), 255–261 (1986)
18. Niederreiter, H.: A short proof for explicit formulas for discrete logarithms in finite fields. Appl. Algebra Engrg. Comm. Comput. 1(1), 55–57 (1990)
19. Niederreiter, H., Winterhof, A.: Incomplete character sums and polynomial interpolation of the discrete logarithm. Finite Fields Appl. 8(2), 184–192 (2002)
20. Pavlovski, C., Boyd, C.: Attacks based on small factors in various group structures. In: Varadharajan, V., Mu, Y. (eds.) ACISP 2001. LNCS, vol. 2119, pp. 36–50. Springer, Heidelberg (2001)
21. Shparlinski, I.E.: Cryptographic Applications of Analytic Number Theory. Complexity Lower Bounds and Pseudorandomness. Progress in Computer Science and Applied Logic, vol. 22. Birkhauser Verlag, Basel (2003)

22. Stadler, M.A.: Publicly verifiable secret sharing. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 190–199. Springer, Heidelberg (1996)
23. Traoré, J.: Group signatures and their relevance to privacy protecting offline electronic cash systems. In: Pieprzyk, J.P., Safavi-Naini, R., Seberry, J. (eds.) ACISP 1999. LNCS, vol. 1587, pp. 228–243. Springer, Heidelberg (1999)
24. Winterhof, A.: A note on the interpolation of the Diffie-Hellman mapping. Bull. Austral. Math. Soc. 64, 475–477 (2001)
25. Winterhof, A.: Polynomial interpolation of the discrete logarithm. Des. Codes Cryptogr. 25, 63–72 (2002)

Single-Path Restarting Tree Automata*

Friedrich Otto and Heiko Stamer

Fachbereich Elektrotechnik/Informatik, Universität Kassel
34109 Kassel, Germany
{otto,stamer}@theory.informatik.uni-kassel.de

Abstract. Restarting tree automata are an extension of top-down tree automata that incorporate transformations of trees through the execution of certain size-reducing rewrite operations. An input tree is repeatedly rewritten until a simple tree is obtained that is then accepted without further rewrites. Accordingly, these automata can be seen as term-rewriting systems with an incorporated regular control realizing parallel rewrites on independent branches. Here we introduce and study two restricted types of restarting tree automata by restricting the options for the regular control. The first variant we consider is the single-path restarting tree automaton, which is obtained from the general model by restricting it to the ability to pass down information along a single path only. In this way it is enforced that rewrites are executed in a strictly sequential way. Interestingly, single-path restarting tree automata reduce the tree languages they recognize to a proper subclass of the class of regular tree languages. Nevertheless, many of the results on the general model of restarting automata carry over to this variant. The second variant we study is the ground-rewrite restarting tree automaton. It is required to perform its size-reducing rewrite steps only on ground terms of bounded height. Accordingly, these automata can be interpreted as ground term-rewriting systems with additional regular control. Although they are much less expressive than the general model, it turns out that due to an inherent synchronization mechanism they can still accept certain non-regular tree languages. Finally, we consider the combination of both restrictions.

Keywords: restarting tree automaton, single-path top-down tree automaton, classes of tree languages, linear context-free tree language.

1 Introduction

The restarting automaton, which was introduced in [4] to model the so-called *analysis by reduction* used in linguistics, has been extended in [9] from strings to trees. Actually several different variants of restarting tree automata have been defined that correspond to certain basic types of restarting automata (on strings). In [9] some fundamental results on the expressive power of these types

* The results presented here are mostly taken from Heiko Stamer's doctoral dissertation [8], where the proofs can be found in full detail.

of restarting tree automata are derived, and some closure properties are given for the families of tree languages recognized by them. In [10] this work is continued by proving that all linear context-free tree languages are recognized by restarting tree automata. In fact, from a linear context-free tree grammar G, a restarting tree automaton $\mathcal{A}$ of type RWWT (see Subsection 2.1 for the definition) can be constructed such that $\mathcal{A}$ recognizes the tree language generated by G. This result is of particular interest from a linguistic point of view, as the linear, nondeleting, monadic context-free tree grammars generate the same class of string languages as tree adjoining grammars [2] and some other formalisms studied in linguistics (see, e.g., [11]).

Basically a restarting tree automaton $\mathcal{A}$ works as follows. Given an input term $t \in \mathcal{T}(\mathcal{F})$, $\mathcal{A}$ reads t in a top-down fashion performing local simplifications (or rewrites), in this way producing a term t_1 that may contain auxiliary symbols. Then $\mathcal{A}$ restarts this process with the term t_1. This continues until $\mathcal{A}$ gets stuck, in which case it rejects, or until a simple term is obtained that $\mathcal{A}$ accepts without modifications. By $L(\mathcal{A})$ we denote the language consisting of all terms that $\mathcal{A}$ accepts, while $S(\mathcal{A})$ is the sublanguage consisting of all terms that $\mathcal{A}$ accepts without modifications. It is called the *simple tree language* recognized by $\mathcal{A}$. It follows easily from the definition that on $S(\mathcal{A})$, the automaton $\mathcal{A}$ works essentially like a finite top-down tree automaton, implying that $S(\mathcal{A})$ is a regular tree language. Thus, $\mathcal{A}$ *reduces* the language $L(\mathcal{A})$ to the regular language $S(\mathcal{A})$. In fact, any regular tree language can occur as the simple language of a restarting tree automaton.

A restarting tree automaton can be seen as a term-rewriting system that is equipped with a regular control. In general it may perform several rewrite steps in parallel, where these steps are synchronized by way of the regular control. Here we introduce and study some restricted types of restarting tree automata. The type we consider first is the *single-path restarting tree automaton.* In order to define it formally we first introduce the *single-path top-down tree automaton* ($\mathsf{spNF{\downarrow}T}$) that is obtained from the finite top-down tree automaton by requiring that information is passed down along a single path only. It turns out that the class of tree languages recognized by these automata forms a strict subclass of the regular tree languages that is incomparable under inclusion to the class of tree languages that are recognized by deterministic finite top-down tree automata ($\mathsf{DF{\downarrow}T}$). The single-path restarting tree automaton is then obtained by extending the $\mathsf{spNF{\downarrow}T}$-automaton by rewrite transitions just as the (general) restarting tree automaton is obtained from the finite top-down tree automaton. Thus, these automata reduce the languages recognized to the subclass of regular tree languages that are defined by single-path top-down tree automata. Essentially a single-path restarting tree automaton can be seen as a term-rewriting system that is equipped with a regular control, but which can only perform rewrites sequentially.

Surprizingly, single-path restarting tree automata are quite expressive. Already the most basic model, the spRT-automaton, accepts a proper superclass of the class $\mathscr{L}(\mathsf{RTG})$ of regular tree languages (Theorem 1), while spRWWT-automata even recognize some tree languages that are not context-free (Corollary 2). Further, all linear context-free tree languages are recognized by these automata (Theorem 2), improving upon the corresponding result from [10].

Then we introduce the *ground-rewrite restarting tree automaton*, which is required to perform rewrite transitions for ground subterms only, that is, they are only executed near the leaves of the current tree. It turns out that this restriction is in some sense orthogonal to the single-path restriction, but spRT-automata that are ground-rewrite can still recognize all regular tree languages. Actually, we conjecture that the ground-rewrite spRT-automaton recognizes only regular tree languages. This would provide a non-trivial characterization for this class of tree languages by a type of restarting tree automaton, as the simple languages of (ground-rewrite) spRT-automata are a proper subclass of the regular tree languages as observed above.

This paper is structured as follows. In Section 2 we restate in short the most basic definitions and notation concerning trees, tree languages, tree automata, and tree grammars. For more details we refer to the monograph [1]. Concerning restarting automata (on strings) we refer to the survey [6]. Then in Section 3 we introduce the single-path top-down tree automaton, before we define the single-path restarting tree automaton in Section 4. The ground-rewrite restarting automaton is studied in Section 5. The paper closes with Section 6, in which we state a number of open problems for future work.

2 Preliminaries and Notation

A *ranked alphabet* $\mathcal{F}$ is a finite nonempty set of symbols such that each $f \in \mathcal{F}$ has a unique nonnegative arity (or *rank*). By $\mathcal{F}_n$ we denote the subset of $\mathcal{F}$ containing all symbols of arity n. Symbols of arity zero are called *constants*. Further, let $\mathcal{X} := \{x_1, x_2, \ldots, x_i, \ldots\}$ be an ordered countable set of *variables*, which are special symbols of rank zero. For each $n \geq 1$, let $\mathcal{X}_n := \{x_1, \ldots, x_n\}$ be the finite subset of $\mathcal{X}$ containing the first n elements of $\mathcal{X}$. Note that $\mathcal{X}$ is always assumed to be disjoint from any other ranked alphabet. Then $\mathcal{T}(\mathcal{F}, \mathcal{X})$ denotes the set of all *terms over* $\mathcal{F}$ *with variables in* $\mathcal{X}$. For $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $\mathrm{Var}(t)$ denotes the set of variables that occur in t. A term is *linear*, if each variable occurs at most once in it. Terms from $\mathcal{T}(\mathcal{F}) := \mathcal{T}(\mathcal{F}, \emptyset)$ are called *ground terms*.

The set of positions of a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is denoted by $\mathrm{Pos}(t)$. By $\mathrm{Top}(t)$ we denote the *outermost* symbol of t, which is the symbol at the root. For $p \in \mathrm{Pos}(t)$, $t|_p$ denotes the *subterm* of t at position p, and $t[u]_p$ denotes the term that is obtained from t by replacing $t|_p$ by the term u. Finally, a term t is called a *scattered subterm* of a term t', if t is *homeomorphically embedded* in t', that is, t can be obtained from t' by 'striking out' some symbols. The *size* $||t||$ and the *height* $\mathrm{Hgt}(t)$ of a term t are defined inductively as follows:

$$\begin{array}{lll}
||t|| = 0, & \mathrm{Hgt}(t) = 0, & \text{if } t \in \mathcal{X}, \\
||t|| = 1, & \mathrm{Hgt}(t) = 0, & \text{if } t \in \mathcal{F}_0, \\
||t|| = 1 + \sum_{i=1}^{n} ||(t|_i)||, & \mathrm{Hgt}(t) = 1 + \max_{i=1,\ldots,n} \mathrm{Hgt}(t|_i), & \text{if } \mathrm{Top}(t) \in \mathcal{F}_n, \\
 & & \text{and } n \geq 1.
\end{array}$$

A *substitution* is a mapping from $\mathcal{X}$ into $\mathcal{T}(\mathcal{F},\mathcal{X})$ that is the identity on all but finitely many variables. A linear term $t \in \mathcal{T}(\mathcal{F},\mathcal{X}_n)$ is called an *n-context*, if $\mathrm{Var}(t) = \mathcal{X}_n$, and if the variables $x_1,\dots,x_n$ occur in this order from left to right in t. By $t[t_1,\dots,t_n]$ we denote the term that is obtained from t by replacing each variable $x_i \in \mathcal{X}_n$ by $t_i \in \mathcal{T}(\mathcal{F},\mathcal{X})$ $(1 \le i \le n)$. The set of all n-contexts is denoted as $\mathrm{Ctx}(\mathcal{F},\mathcal{X}_n)$.

A *rewrite rule* is a pair of terms, denoted by $l \to r$, where $l,r \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $l \notin \mathcal{X}$, and $\mathrm{Var}(l) \supseteq \mathrm{Var}(r)$. It is called *linear*, if both l and r are linear terms. A *term rewriting system* (TRS) is a set Δ of rewrite rules. The induced *rewriting relation* $\to_\Delta$ over $\mathcal{T}(\mathcal{F},\mathcal{X})$ is the least relation containing Δ that is closed under subterm replacement and substitution, that is, for $t,t' \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $t \to_\Delta t'$ if and only if there exist a 1-context $s \in \mathrm{Ctx}(\mathcal{F},\mathcal{X}_1)$, a rewrite rule $(l \to r) \in \Delta$, and a substitution σ such that $t = s[\sigma(l)]$ and $t' = s[\sigma(r)]$ hold. By $\to_\Delta^*$ we denote the reflexive transitive closure of this relation.

Next we consider automata on trees. Let $\mathcal{Q}$ be a finite set of unary symbols called *states* such that $\mathcal{Q} \cap \mathcal{F} = \emptyset$. Then $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ is the set of *configurations*. A *normalized top-down transition* is a linear rewrite rule of the form

$$q(f(x_1,\dots,x_n)) \to f(q_1(x_1),\dots,q_n(x_n)),$$

where $n \ge 1$, $f \in \mathcal{F}_n$, $x_1,\dots,x_n \in \mathcal{X}$, and $q,q_1,\dots,q_n \in \mathcal{Q}$. If the symbol from $\mathcal{F}$ is a constant $a \in \mathcal{F}_0$, then the corresponding transitions have the form $q(a) \to a$. They are called *normalized final transitions*. A *nondeterministic finite top-down tree automaton* ($\mathsf{NF{\downarrow}T}$) is given through a four-tuple $\mathcal{A} = (\mathcal{F},\mathcal{Q},\mathcal{Q}_0,\Delta)$, where $\mathcal{F}$ is a finite ranked alphabet, $\mathcal{Q}$ is a finite set of states, $\mathcal{Q}_0 \subseteq \mathcal{Q}$ is a set of initial states, and Δ is a finite term rewriting system on $\mathcal{F} \cup \mathcal{Q}$ consisting of normalized top-down and final transitions only. This automaton is *deterministic* ($\mathsf{DF{\downarrow}T}$), if $\mathcal{Q}_0$ is a singleton, and if there are no two rewrite rules in Δ with the same left-hand side. The *move relation* $\to_\mathcal{A}$ and its reflexive transitive closure $\to_\mathcal{A}^*$ are induced by the TRS Δ. The set of terms

$$L(\mathcal{A}) = \{\, t \in \mathcal{T}(\mathcal{F}) \mid \exists q \in \mathcal{Q}_0 \,:\, q(t) \to_\mathcal{A}^* t \,\}$$

is the *tree language recognized by* $\mathcal{A}$. A *nondeterministic finite bottom-up tree automaton* ($\mathsf{NF{\uparrow}T}$) is given by a four-tuple $\mathcal{A} = (\mathcal{F},\mathcal{Q},\mathcal{Q}_f,\Delta)$, where $\mathcal{F}$ is a finite ranked alphabet, $\mathcal{Q}$ is a finite set of states, $\mathcal{Q}_f \subseteq \mathcal{Q}$ is a set of final states, and Δ is a finite term rewriting system on $\mathcal{F} \cup \mathcal{Q}$ consisting of ground rewrite rules of the form $f(q_1,\dots,q_n) \to q$, where $n \ge 0$, $f \in \mathcal{F}_n$, and $q,q_1,\dots,q_n \in \mathcal{Q}$. Note that in the bottom-up case the states are constants. The *move relation* $\to_\mathcal{A}$ and its reflexive transitive closure $\to_\mathcal{A}^*$ are induced by the TRS Δ. The set of terms

$$L(\mathcal{A}) = \{\, t \in \mathcal{T}(\mathcal{F}) \mid \exists q \in \mathcal{Q}_f \,:\, t \to_\mathcal{A}^* q \,\}$$

is the *tree language recognized by* $\mathcal{A}$. An $\mathsf{NF{\uparrow}T}$-automaton $\mathcal{A} = (\mathcal{F},\mathcal{Q},\mathcal{Q}_f,\Delta)$ is *deterministic* ($\mathsf{DF{\uparrow}T}$) if there are no two rules in Δ that have the same left-hand side. It is called *complete* if there is at least one rule $f(q_1,\dots,q_n) \to q$ in Δ for all $f \in \mathcal{F}_n$, $n \ge 0$, and all $q_1,\dots,q_n \in Q$.

For any class A of tree automata, $\mathscr{L}(\mathsf{A})$ denotes the *class of tree languages* that are recognized by automata from that class. Concerning the expressive power of the various types of finite tree automata it is well-known that

$$\mathscr{L}(\mathsf{DF{\downarrow}T}) \subsetneq \mathscr{L}(\mathsf{NF{\downarrow}T}) = \mathscr{L}(\mathsf{NF{\uparrow}T}) = \mathscr{L}(\mathsf{DF{\uparrow}T}),$$

and that $\mathscr{L}(\mathsf{DF{\downarrow}T})$ does not even contain all finite tree languages [1].

Finally, we turn to tree grammars. A *context-free tree grammar* (CFTG) $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ consists of two finite disjoint ranked alphabets $\mathcal{F}$ and $\mathcal{N}$, a finite TRS $\mathcal{P}$, and a distinct initial symbol $S \in \mathcal{N}_0$. The elements of $\mathcal{F}$ are called terminal symbols, and those of $\mathcal{N}$ are nonterminal symbols. The rewrite rules (productions) from $\mathcal{P}$ are all of the form $A(x_1, \ldots, x_n) \to t$, where $n \geq 0$, $A \in \mathcal{N}_n$ is a nonterminal symbol, $x_1, \ldots, x_n \in \mathcal{X}$ are variables, and $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{N}, \mathcal{X}_n)$ is a term. The *unrestricted derivation relation* $\Rightarrow_G$ and its reflexive transitive closure $\Rightarrow_G^*$ are induced by $\mathcal{P}$. The *tree language generated by* G is

$$L(G) = \{\, t \in \mathcal{T}(\mathcal{F}) \mid S \Rightarrow_G^* t \,\}.$$

For any class G of tree grammars, $\mathscr{L}(\mathsf{G})$ denotes the *class of tree languages* that are generated by grammars from that class. A context-free tree grammar is called *regular* (RTG), if all its nonterminal symbols are constants, that is, $\mathcal{N} = \mathcal{N}_0$. It is called *linear* (lin-CFTG), if all productions are linear. A set of ground terms $E \subseteq \mathcal{T}(\mathcal{F})$ is called a *regular* or a (*linear*) *context-free tree language*, respectively, if there is a regular, respectively a (linear) context-free, tree grammar G such that $L(G) = E$. It is well-known that $\mathscr{L}(\mathsf{RTG}) = \mathscr{L}(\mathsf{NF{\uparrow}T})$ holds [1,3].

2.1 Restarting Tree Automata

In [9] restarting automata on trees were introduced. Formally, a (*top-down*) *restarting tree automaton* (RRWWT-automaton, for short) is described by a six-tuple $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$, where $\mathcal{F}$ is a finite ranked input alphabet, $\mathcal{G} \supseteq \mathcal{F}$ is a finite ranked working alphabet, $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ is a finite set of states such that $\mathcal{Q}_1 \cap \mathcal{Q}_2 = \emptyset$, $q_0 \in \mathcal{Q}_1$ is the initial state and simultaneously the restart state, $k \geq 1$ is the height of the read/write-windows, and $\Delta = \Delta_1 \cup \Delta_2$ is a finite term rewriting system on $\mathcal{G} \cup \mathcal{Q}$. The symbols from $\mathcal{G} \smallsetminus \mathcal{F}$ are called *auxiliary symbols*. The rule set Δ_1 only contains *k-height bounded top-down transitions* of the form

$$q(t) \to t[q_1(x_1), \ldots, q_m(x_m)], \tag{1}$$

where $m \geq 1$, $t \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$, and $q, q_1, \ldots, q_m \in \mathcal{Q}_1$, and *k-height bounded final transitions* of the form

$$q(t) \to t, \tag{2}$$

where $t \in \mathcal{T}(\mathcal{G})$ and $q \in \mathcal{Q}_1$. The rule set Δ_2 contains *size-reducing top-down rewrite transitions*, that is, linear rewrite rules of the form

$$q(t) \to t'[q_1(x_1), \ldots, q_m(x_m)], \tag{3}$$

where $m \geq 1$, $t, t' \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$, $q \in \mathcal{Q}_1$, and $q_1, \ldots, q_m \in \mathcal{Q}_2$, and *size-reducing final rewrite transitions* of the form

$$q(t) \to t', \tag{4}$$

where $q \in \mathcal{Q}_1$ and $t, t' \in \mathcal{T}(\mathcal{G})$. For both these types of transitions it is required that $||t|| > ||t'||$ and $\mathrm{Hgt}(t) \leq k$. In addition, Δ_2 contains *k-height bounded top-down transitions*

$$q(t) \to t[q_1(x_1), \ldots, q_m(x_m)], \tag{5}$$

where $m \geq 1$, $t \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$, and $q, q_1, \ldots, q_m \in \mathcal{Q}_2$, and *k-height bounded final transitions*

$$q(t) \to t, \tag{6}$$

where $t \in \mathcal{T}(\mathcal{G})$ and $q \in \mathcal{Q}_2$.

The *partial move relation* $\to_\Delta$ and its reflexive transitive closure $\to_\Delta^*$ are induced by the TRS Δ, while the *final move relation* $\to_{\Delta_1}$ and its reflexive transitive closure $\to_{\Delta_1}^*$ are induced by Δ_1. We use the notation $u \hookrightarrow_\mathcal{A} v$ ($u, v \in \mathcal{T}(\mathcal{G})$) to express the fact that $q_0(u)$ $(\to_\Delta^* \smallsetminus \to_{\Delta_1}^+)$ v, and we say that u is transformed into v by a *cycle* of $\mathcal{A}$. As $q_0 \in \mathcal{Q}_1$, this notation means that at least one size-reducing rewrite transition from Δ_2 of type (3) or (4) is applied in transforming $q_0(u)$ into v. Observe further that it is ensured by the way in which the states from $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are used that in a cycle no two rewrite transitions can be applied on the same path. The relation $\hookrightarrow_\mathcal{A}^*$ is the reflexive transitive closure of $\hookrightarrow_\mathcal{A}$. The *tree language* recognized by the RRWWT-automaton $\mathcal{A}$ is

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

The final part $q_0(t') \to_{\Delta_1}^* t'$ is called the *tail* of the computation. Thus, each computation of $\mathcal{A}$ consists of a finite sequence of cycles followed by a tail. The *simple tree language* recognized by $\mathcal{A}$ is $S_\mathcal{F}(\mathcal{A}) = \{\, t \in \mathcal{T}(\mathcal{F}) \mid q_0(t) \to_{\Delta_1}^* t \,\}$, and the *auxiliary simple tree language* is $S_\mathcal{G}(\mathcal{A}) = \{\, t \in \mathcal{T}(\mathcal{G}) \mid q_0(t) \to_{\Delta_1}^* t \,\}$, that is, these are the languages consisting of all terms (from $\mathcal{T}(\mathcal{F})$ or $\mathcal{T}(\mathcal{G})$, respectively) that are accepted in tail computations.

Also some restricted variants of restarting tree automata have been introduced. A restarting tree automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ is called an RWWT-*automaton*, if Q_2 only contains "don't-care" states, that is, the top-down transitions in Δ_2 of type (3) are of the form $q(f(x_1, \ldots, x_m)) \to f(q(x_1), \ldots, q(x_m))$ for all $q \in Q_2$ and all $f \in \mathcal{T}(\mathcal{G})$. In this situation the subset Q_2 can be ignored, and the top-down rewrite transitions can be written in the special form

$$q(t) \to t'[x_1, \ldots, x_m], \tag{7}$$

where $m \geq 1$, $q \in \mathcal{Q}_1$, and $t, t' \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$ satisfying the requirements that $||t'|| < ||t||$ and $\mathrm{Hgt}(t) \leq k$ hold. A restarting tree automaton is an RRWT-*automaton*, if its working alphabet $\mathcal{G}$ coincides with its input alphabet $\mathcal{F}$, that is, no auxiliary symbols are available. It is an RRT-*automaton*, if it is an RRWT-automaton for which the right-hand side of every rewrite transition is a scattered subterm of the corresponding left-hand side. Analogously, we obtain the RWT- and the RT-*automaton* from the RWWT-automaton.

Example 1. The language

$$L_1 := \{\, f(g^n(a), g^n(a)) \mid n \geq 0 \,\} \in \mathscr{L}(\mathsf{CFTG}) \smallsetminus \mathscr{L}(\mathsf{RTG})$$

is recognized by the RT-automaton $\mathcal{A}_1 = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$, where $\mathcal{F} = \{f(\cdot,\cdot), g(\cdot), a\}$, $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ with $\mathcal{Q}_1 = \{q_0\}$, $\mathcal{Q}_2 = \emptyset$, $k = 2$, and Δ is given by the rewrite rules $q_0(f(g(x_1), g(x_2))) \to f(x_1, x_2)$ and $q_0(f(a,a)) \to f(a,a)$.

With a finite alphabet $\Sigma = \{a_1, a_2, \ldots, a_n\}$ we associate the ranked alphabet $\mathcal{F}_\Sigma := \{a_1(\cdot), \ldots, a_n(\cdot), \bot\}$, where $a_i(\cdot)$ $(1 \leq i \leq n)$ are unary symbols and $\bot$ is a constant. Then the free monoid Σ^* and the set of ground terms $\mathcal{T}(\mathcal{F}_\Sigma)$ are in one-to-one correspondence modulo the mapping

$$\hat{} : a_{i_1} a_{i_2} \cdots a_{i_m} \mapsto a_{i_1}(a_{i_2}(\cdots(a_{i_m}(\bot))\cdots)).$$

It has been shown in [9] that, for each $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}\}$, there is a close correspondence between the X-automata on Σ and the XT-automata on $\mathcal{F}_\Sigma$. Further, already the class $\mathscr{L}(\mathsf{RT})$ contains tree languages that are not even context-free. On the other hand, from a linear context-free tree grammar G, an RWWT-automaton $\mathcal{A}$ can be constructed that recognizes the tree language $L(G)$ [10].

3 Single-Path Top-Down Tree Automata

Here we introduce the *single-path top-down tree automaton* ($\mathsf{spNF{\downarrow}T}$), which will serve as the basis for the single-path restarting tree automaton in the next section. An $\mathsf{spNF{\downarrow}T}$-automaton is given through a five-tuple $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$, where $\mathcal{F}$ is a finite ranked alphabet, $\mathcal{Q}$ is a finite set of states, $q_0 \in \mathcal{Q}$ is the initial state, $k \geq 1$ is the height of the look-ahead window, and Δ is a finite term rewriting system that contains *k-height bounded top-down transitions* of the form

$$q(t) \to t[x_1, \ldots, x_{i-1}, q'(x_i), x_{i+1} \ldots, x_m], \tag{8}$$

where $m \geq 1$, $t \in \mathrm{Ctx}(\mathcal{F}, \mathcal{X}_m)$, $q, q' \in \mathcal{Q}$, and $i \in \{1, \ldots, m\}$, and *k-height bounded final transitions* of the form

$$q(t) \to t, \tag{9}$$

where $t \in \mathcal{T}(\mathcal{F})$ and $q \in \mathcal{Q}$. The *tree language recognized* by $\mathcal{A}$ is

$$L(\mathcal{A}) = \{\, t \in \mathcal{T}(\mathcal{F}) \mid q_0(t) \to_\Delta^* t \,\},$$

that is, it consists of those ground terms $t \in \mathcal{T}(\mathcal{F})$ for which $\mathcal{A}$ has an accepting run starting from the configuration $q_0(t)$.

Proposition 1. *For each* $\mathsf{spNF{\downarrow}T}$*-automaton* $\mathcal{A}$*,* $L(\mathcal{A}) \in \mathscr{L}(\mathsf{RTG})$.

Proof. Let $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ be an $\mathsf{spNF{\downarrow}T}$-automaton. From $\mathcal{A}$ we construct a regular tree grammar $G_{\mathcal{A}} := (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ by taking $\mathcal{N} := \mathcal{Q} \cup \{U\}$, $S := q_0$, and $\mathcal{P}$ to consist of the following productions:

$$\begin{array}{ll} q \ \to t & \text{for all } (q(t) \to t) \in \Delta, \\ q \ \to t[U, \ldots, U, q', U \ldots, U] & \text{for all } (q(t) \to t[x_1, \ldots, q'(x_i), \ldots, x_m]) \in \Delta, \\ U \to f(U, \ldots, U) & \text{for all } n\text{-ary function symbols } f \in F_n,\ n \geq 1, \\ U \to a & \text{for all constants } a \in \mathcal{F}_0. \end{array}$$

Observe that the subgrammar $(\mathcal{F}, \{U\}, \mathcal{P}, U)$ simply generates the set of ground terms $\mathcal{T}(\mathcal{F})$. It is now easily verified that $L(G_{\mathcal{A}}) = L(\mathcal{A})$ holds. It follows in particular that $L(\mathcal{A})$ is a regular tree language. □

From the above proof we see that, for each $\mathsf{spNF{\downarrow}T}$-automaton $\mathcal{A}$, the regular tree language $L(\mathcal{A})$ is of a rather restricted form.

Example 2. Let $\mathcal{F} := \{f(\cdot, \cdot), a\}$, and let L_u be the regular tree language that is generated by the regular tree grammar $G_u := (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$, where

$$\mathcal{N} := \{S, A\}, \text{ and } \mathcal{P} := \{S \to f(A, A), S \to f(S, S), A \to a\}.$$

Then $t \in \mathcal{T}(\mathcal{F})$ is an element of L_u if and only if, for all $p \in \mathrm{Pos}(t)$, if $\mathrm{Top}(t|_p) = f$, then either $t|_p = f(a, a)$ or $t|_p = f(f(t_1, t_2), f(t_3, t_4))$ for some $t_1, \ldots, t_4 \in \mathcal{T}(\mathcal{F})$. We claim that $L_u \neq L(\mathcal{A})$ for each $\mathsf{spNF{\downarrow}T}$-automaton $\mathcal{A}$.

Assume that $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ is an $\mathsf{spNF{\downarrow}T}$-automaton such that $L(\mathcal{A}) = L_u$. The complete binary tree t of height 2^k is an element of L_u. As $L_u = L(\mathcal{A})$, this implies $q_0(t) \to_{\Delta}^{*} t$. In the course of this computation $\mathcal{A}$ walks down a single path from the root of t to one of its leaves. During this process $\mathcal{A}$ always sees the same partial term, that is, the partial term of height k that has an occurrence of the binary symbol f at every position, until it reaches the leaves of t. Thus, there exists a position $p \in \mathrm{Pos}(t)$ such that $t|_p = f(f(a, a), f(a, a))$, but this particular subterm is not seen by $\mathcal{A}$ during the above computation. Hence, we can simply replace the subterm $t|_p$ by the term $f(f(a, a), a)$, which yields the term $t' := t[f(f(a, a), a)]_p \notin L_u$. However, starting from $q_0(t')$, $\mathcal{A}$ can perform the same transition steps as in the above computation, which yields the computation $q_0(t') \to_{\Delta}^{*} t'$. Thus, $t' \in L(\mathcal{A})$, implying that $L(\mathcal{A}) \neq L_u$.

Obviously, each finite tree language is recognized by some $\mathsf{spNF{\downarrow}T}$-automaton. It follows that $\mathscr{L}(\mathsf{spNF{\downarrow}T})$ is not contained in $\mathscr{L}(\mathsf{DF{\downarrow}T})$, as the finite tree language $\{f(a, b), f(b, a)\}$ is not recognized by any $\mathsf{DF{\downarrow}T}$. On the other hand, it can be shown that the tree language

$$L_d := \{\, f(g^n(a), g^m(b)) \mid n, m \geq 0 \,\} \in \mathscr{L}(\mathsf{DF{\downarrow}T})$$

is not recognized by any $\mathsf{spNF{\downarrow}T}$-automaton by arguing as in Example 2. Thus, we have the following results, where $\mathscr{L}(\mathsf{FINT})$ denotes the class of all finite tree languages.

Corollary 1. (a) $\mathscr{L}(\mathsf{FINT}) \subsetneq \mathscr{L}(\mathsf{spNF}{\downarrow}\mathsf{T}) \subsetneq \mathscr{L}(\mathsf{RTG})$.

(b) $\mathscr{L}(\mathsf{spNF}{\downarrow}\mathsf{T})$ *is incomparable to* $\mathscr{L}(\mathsf{DF}{\downarrow}\mathsf{T})$ *under inclusion.*

However, no characterization in terms of more classical automata, grammars or rewriting systems is currently known for the class $\mathscr{L}(\mathsf{spNF}{\downarrow}\mathsf{T})$.

4 Single-Path Restarting Tree Automata

Now we define the single-path variant of the restarting tree automaton and establish some basic results concerning its expressive power.

Definition 1. *A* single-path restarting tree automaton ($\mathsf{spRRWWT}$*-automaton, for short*) *is formally described by a six-tuple* $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$, *where*

– $\mathcal{F}$ *is a finite ranked input alphabet,*
– $\mathcal{G} \supseteq \mathcal{F}$ *is a finite ranked working alphabet,*
– $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ *is a finite set of states such that* $\mathcal{Q}_1 \cap \mathcal{Q}_2 = \emptyset$,
– $q_0 \in \mathcal{Q}_1$ *is the initial state and simultaneously the restart state,*
– $k \geq 1$ *is the height of the read/write-windows, and*
– $\Delta = \Delta_1 \cup \Delta_2$ *is a finite term rewriting system on* $\mathcal{G} \cup \mathcal{Q}$,

where $(\mathcal{G}, \mathcal{Q}_1, q_0, k, \Delta_1)$ *is an* $\mathsf{spNF}{\downarrow}\mathsf{T}$*-automaton on* $\mathcal{G}$*, and the rule set* Δ_2 *only contains the following types of transitions:*

1. Size-reducing top-down rewrite transitions, *that is, linear rewrite rules of the form*
$$q(t) \to t'[x_1, \ldots, x_{i-1}, q'(x_i), x_{i+1}, \ldots, x_m], \tag{10}$$
where $m \geq 1$, $t, t' \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$, $i \in \{1, \ldots, m\}$, $q \in \mathcal{Q}_1$, *and* $q' \in \mathcal{Q}_2$, *and* size-reducing final rewrite transitions *of the form*
$$q(t) \to t', \tag{11}$$
where $q \in \mathcal{Q}_1$ *and* $t, t' \in \mathcal{T}(\mathcal{G})$. *For both these types of transitions it is required that* $||t|| > ||t'||$ *and* $\mathrm{Hgt}(t) \leq k$.
2. k-height bounded top-down transitions *of the form*
$$q(t) \to t[x_1, \ldots, x_{i-1}, q'(x_i), x_{i+1}, \ldots, x_m], \tag{12}$$
where $m \geq 1$, $t \in \mathrm{Ctx}(\mathcal{G}, \mathcal{X}_m)$, $i \in \{1, \ldots, m\}$, *and* $q, q' \in \mathcal{Q}_2$, *and* k-height bounded final transitions *of the form*
$$q(t) \to t, \tag{13}$$
where $t \in \mathcal{T}(\mathcal{G})$ *and* $q \in \mathcal{Q}_2$.

Essentially, an $\mathsf{spRRWWT}$-automaton $\mathcal{A}$ works just like an RRWWT-automaton, the only difference is the fact that in the current tree $\mathcal{A}$ walks down a single path only. In particular, the *partial move relation* $\to_\Delta$ and its reflexive transitive closure $\to_\Delta^*$ are induced by the TRS Δ, while the *final move relation* $\to_{\Delta_1}$ and its reflexive transitive closure $\to_{\Delta_1}^*$ are induced by Δ_1. Here we also use the notation $u \hookrightarrow_{\mathcal{A}} v$ $(u, v \in \mathcal{T}(\mathcal{G}))$ to express the fact that $q_0(u)$ $(\to_\Delta^* \setminus \to_{\Delta_1}^+)$ v, and we say that u is transformed into v by a *cycle* of $\mathcal{A}$. As $q_0 \in \mathcal{Q}_1$, and as no two rewrite transitions can be applied on the same path, this means that exactly one size-reducing rewrite transition from Δ_2 of type (10) or (11) is applied in transforming $q_0(u)$ into v. The *tree language* accepted by $\mathcal{A}$ is defined as

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

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

As the simple language of $\mathcal{A}$ is the intersection of the language that is recognized by the $\mathsf{spNF{\downarrow}T}$-automaton $(\mathcal{G}, \mathcal{Q}_1, q_0, k, \Delta_1)$ with the set of ground terms $\mathcal{T}(\mathcal{F})$, we see from Corollary 1 that the class of simple languages of spRWWT-automata is properly contained in the class of regular tree languages.

For a ranked alphabet containing only unary function symbols and constants single-path restarting tree automata coincide with restarting tree automata. Hence, the correspondence between restarting tree automata and restarting automata on strings mentioned above carries over to single-path restarting tree automata. Further, the following result is seen easily, where the various restricted types of $\mathsf{spRRWWT}$-automata are obtained as for the general RRWWT-automaton.

Proposition 2. *Let* $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$ *be a type of restarting automaton. Then for each* spXT*-automaton* M *there exists an* XT*-automaton* M' *such that* $L(M) = L(M')$.

Note that the RT-automaton from Example 1 is in fact an spRT-automaton.

Theorem 1. $\mathscr{L}(\mathsf{RTG}) \subsetneq \mathscr{L}(\mathsf{spRT})$.

Proof. We know already from Example 1 that $\mathscr{L}(\mathsf{spRT})$ contains non-regular tree languages. Thus, it remains to show that each tree language $L \in \mathscr{L}(\mathsf{RTG})$ is recognized by some spRT-automaton.

Let $\mathcal{D} = (\mathcal{F}, \mathcal{Q}^{(\mathcal{D})}, Q_f^{(\mathcal{D})}, \Delta^{(\mathcal{D})})$ be a $\mathsf{DF{\uparrow}T}$-automaton, and let $L \subseteq \mathcal{T}(\mathcal{F})$ be the tree language recognized by $\mathcal{D}$. Without loss of generality we can assume that $\mathcal{D}$ is complete. We construct a corresponding spRT-automaton $\mathcal{A} :=$ $(\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ by taking $\mathcal{Q} := \mathcal{Q}_1 := \{q_0(\cdot), q_1(\cdot)\}$, $k := |\mathcal{Q}^{(\mathcal{D})}| + 1$, and $\Delta := \Delta_1 \cup \Delta_2$ to consist of the following groups of rules:

I: Final transitions:

(1) $q_0(t) \to t$ if $t \in L$ satisfying $\mathrm{Hgt}(t) \leq k$.

II: Top-down transitions:

(2) $q_0(f(x_1,\ldots,x_n)) \to f(x_1,\ldots,x_{i-1},q_1(x_i),x_{i+1},\ldots,x_n)$
for all $f \in \mathcal{F}_n$, $n \geq 1$, and $1 \leq i \leq n$,

(3) $q_1(f(x_1,\ldots,x_n)) \to f(x_1,\ldots,x_{i-1},q_1(x_i),x_{i+1},\ldots,x_n)$
for all $f \in \mathcal{F}_n$, $n \geq 1$, and $1 \leq i \leq n$,

III: Final Rewrite transitions:

(4) $q_1(t) \to t_1[t_3]$ for all $t \in \mathcal{T}(\mathcal{F})$, $\mathrm{Hgt}(t) = k$, where $t = t_1[t_2[t_3]]$ such that
$t_3 \to^*_{\Delta^{(\mathcal{D})}} q$, and $t_2[q] \to^*_{\Delta^{(\mathcal{D})}} q$ for some $q \in \mathcal{Q}^{(\mathcal{D})}$,

where in (4) an arbitrary factorization of the term t with the required properties is chosen. Here state q_1 is used to ensure that final transitions can only be applied at the root of a tree. Concerning the rewrite rules observe the following. If $t \in \mathcal{T}(\mathcal{F})$ can be written as $t = t_1[t_2[t_3]]$ such that $t_3 \to^*_{\Delta^{(\mathcal{D})}} q$ and $t_2[q] \to^*_{\Delta^{(\mathcal{D})}} q$ hold for some $q \in \mathcal{Q}^{(\mathcal{D})}$, then t is accepted by $\mathcal{D}$ if and only if $t' := t_1[t_3]$ is accepted by $\mathcal{D}$. It follows that $L(\mathcal{A}) \subseteq L(\mathcal{D}) = L$ holds. It remains to establish the converse inclusion.

Claim. $L \subseteq L(\mathcal{A})$.

Proof. If $t \in L$ satisfies $\mathrm{Hgt}(t) \leq k$, then t is immediately accepted by $\mathcal{A}$ using the corresponding rule from group (1). So let $t \in L$ such that $\mathrm{Hgt}(t) > k$. For each position $o \in \mathrm{Pos}(t)$, there exists a unique state $q \in Q^{(\mathcal{D})}$ such that $t|_o \to^*_{\Delta^{(\mathcal{D})}} q$ holds, as $\mathcal{D}$ is deterministic and complete. Now let $p \in \mathrm{Pos}(t)$ be a position of maximal depth. Then the path from the root of t to the constant at position p is of length $\mathrm{Hgt}(t) > k$. Thus, there exist contexts $t_0, t_1, t_2 \in \mathrm{Ctx}(\mathcal{F}, \{x_1\})$, and a term $t_3 \in \mathcal{T}(\mathcal{F})$ such that $t = t_0[t_1[t_2[t_3]]]$ and the following two conditions are satisfied:

1. $\mathrm{Hgt}(t_1[t_2[t_3]]) = k$,
2. $t_3 \to^*_{\Delta^{(\mathcal{D})}} q$ and $t_2[t_3] \to^*_{\Delta^{(\mathcal{D})}} q$ for some $q \in Q^{(\mathcal{D})}$.

Hence, $\mathcal{A}$ can execute the cycle $t = t_0[t_1[t_2[t_3]]] \hookrightarrow_{\mathcal{A}} t_0[t_1[t_3]]$. However, with t also the term $t' := t_0[t_1[t_3]]]$ belongs to $L = L(\mathcal{D})$. As it is strictly smaller than t, induction yields that $t \in L(\mathcal{A})$ holds. □

Our next example shows that spRWWT-automata are very expressive.

Example 3. The tree language

$$L_4 := \{\, f(g^n(h^n(a)), g^n(h^n(a))) \mid n \geq 1 \,\},$$

which is not context-free, is accepted by an RT-automaton [9]. Here we present an spRWWT-automaton $\mathcal{A}_4 := (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ for this language.

Let $\mathcal{F} := \{f(\cdot,\cdot), g(\cdot), h(\cdot), a\}$, $\mathcal{G} := \mathcal{F} \cup \{F(\cdot,\cdot), G(\cdot)\}$, $\mathcal{Q} := \mathcal{Q}_1 := \{q_0, q_1\}$, $k := 3$, and let $\Delta := \Delta_1 \cup \Delta_2$ consist of the following groups of rules:

I: Top-down transitions of Δ_1:

(1) $q_0(G(x_1)) \to G(q_1(x_1))$,
(2) $q_1(G(x_1)) \to G(q_1(x_1))$.

II: Final transitions of Δ_1:

(3) $q_0(f(g(h(a)), g(h(a)))) \to f(g(h(a)), g(h(a)))$,
(4) $q_0(G(F(h(a), h(a)))) \to G(F(h(a), h(a)))$.

III: Rewrite transitions of Δ_2:

(5) $q_0(f(g(g(x_1)), g(g(x_2)))) \to G(F(g(x_1), g(x_2)))$,
(6) $q_1(F(g(x_1), g(x_2))) \to G(F(x_1, x_2))$,
(7) $q_1(G(F(h(h(x_1)), h(h(x_2))))) \to F(h(x_1), h(x_2))$.

Claim 1. $L_4 \subseteq L(\mathcal{A}_4)$.

Proof. Let $t \in L_4$, that is, there exists an integer $n \geq 1$ such that

$$t = f(g^n(h^n(a)), g^n(h^n(a))).$$

If $n = 1$, then $t = f(g(h(a)), g(h(a)))$ is immediately accepted by rule (3). If $n \geq 2$, then $\mathcal{A}_4$ can execute the following sequence of cycles:

$$\begin{array}{rcl} t &=& f(g^n(h^n(a)), g^n(h^n(a))) \hookrightarrow_{\mathcal{A}_4} G(F(g^{n-1}(h^n(a)), g^{n-1}(h^n(a)))) \\ \hookrightarrow_{\mathcal{A}_4}^{n-1} & G^n(F(h^n(a), h^n(a))) & \hookrightarrow_{\mathcal{A}_4} G^{n-1}(F(h^{n-1}(a), h^{n-1}(a))) \\ \hookrightarrow_{\mathcal{A}_4}^{n-2} & G(F(h(a), h(a))), & \end{array}$$

which is then accepted by rule (4). Thus, $L_4 \subseteq L(\mathcal{A}_4)$. □

Claim 2. $L(\mathcal{A}_4) \subseteq L_4$.

Proof. Each rewrite transition of $\mathcal{A}_4$ has nonterminal symbols on its right-hand side. Thus, a term $t \in \mathcal{T}(\mathcal{F})$ is accepted by $\mathcal{A}_4$, if either $t = f(g(h(a)), g(h(a)))$ or $t \hookrightarrow_{\mathcal{A}_4}^{+} G(F(h(a), h(a)))$. In the former case, $t \in L_4$. Thus, it remains to study the latter case.

If $t \in \mathcal{T}(\mathcal{F})$ such that $t \hookrightarrow_{\mathcal{A}_4}^{+} G(F(h(a), h(a)))$, then it follows from the form of the rules of $\mathcal{A}_4$ that $\mathrm{Top}(t) = f$ and that $|t|_f = 1$, that is, t is of the form $f = f(g^2(t_1), g^2(t_2))$, where $t_1, t_2 \in \mathcal{T}(\{g(\cdot), h(\cdot), a\})$. Again from the form of the rewrite transitions it follows that $t_1 = g^{n-2}(h^n(a)) = t_2$ for some $n \geq 2$. Thus, $t = f(g^n(h^n(a)), g^n(h^n(a)))$, that is, $t \in L_4$. □

Thus, $L(\mathcal{A}_4) = L_4$ holds, implying that $L_4 \in \mathscr{L}(\mathsf{spRWWT})$.

This example yields the following consequence.

Corollary 2. $\mathscr{L}(\mathsf{spRWWT})$ *contains tree languages that are not context-free.*

Without auxiliary symbols these automata are strictly less expressive.

Lemma 1. *$L_4 \notin \mathscr{L}(\mathsf{spRRWT})$, that is, the language L_4 is not accepted by any single-path restarting tree automaton that has no auxiliary symbols.*

Proof. Let $\mathcal{F} = \{f(\cdot,\cdot), g(\cdot), h(\cdot), a\}$. Assume that $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ is an spRRWT-automaton for the language L_4, and let $t_n := f(g^n(h^n(a)), g^n(h^n(a)))$ for a sufficiently large value of n. Then $t_n \in L_4$, that is, $\mathcal{A}$ has an accepting computation on input t_n. Clearly this computation cannot be an accepting tail, that is, it has the form $t_n \hookrightarrow_{\mathcal{A}} s_1 \hookrightarrow_{\mathcal{A}} \cdots \hookrightarrow_{\mathcal{A}} s_m$ such that $q_0(s_m) \rightarrow^*_{\Delta_1} s_m$. As $\mathcal{A}$ has no auxiliary symbols, $s_1, \ldots, s_m \in \mathcal{T}(\mathcal{F})$, implying that $s_1, \ldots, s_m \in L(\mathcal{A})$. In particular, this means that $s_1 \in L_4$. In the cycle $t_n \hookrightarrow_{\mathcal{A}} s_1$ a single size-reducing rewrite transition is applied. This rewrite transition is either applied at the root of t_n, replacing the top part of the form $f(g^{k-1}(\cdot), g^{k-1}(\cdot))$ of t_n by a smaller term, or it is applied inside one of the two subterms $g^n(h^n(a))$. In either case the resulting term s_1 does not belong to the language L_4. Thus, we see that $L(\mathcal{A}) \neq L_4$, that is, L_4 is not accepted by any spRRWT-automaton. □

Observe, however, that it is not known whether $\mathscr{L}(\mathsf{spRRWT})$ only consists of context-free tree languages. The next result improves upon the corresponding result for RWWT-automata from [10].

Theorem 2. *Given a linear context-free tree grammar G, an* spRWWT-*automaton $\mathcal{A}$ can be constructed such that $L(G) = L(\mathcal{A})$ holds.*

Proof. Let $G = (\mathcal{F}, \mathcal{N}, \mathcal{P}, S)$ be a linear context-free tree grammar. By the normalization procedure of [10] we can assume that G is *growing*, that is, each rule of $\mathcal{P}$ is of the form $A(x_1, \ldots, x_n) \rightarrow t$, where $n \geq 0$, $A \in \mathcal{N}_n$, and $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{N}, \mathcal{X}_n)$ satisfying $\mathrm{Var}(t) = \mathcal{X}_n$ and $||t|| \geq 2$, or it is of the form $S \rightarrow s$, where $s \in \mathcal{T}(\mathcal{F})$.

Let C_S be the set of all constants $c \in (\mathcal{F}_0 \cup \mathcal{N}_0)$ such that $\mathcal{P}$ contains a rule $S \rightarrow c$. For each rule $(l \rightarrow r) \in \mathcal{P}$, where the initial symbol S occurs at least once in the right-hand side r, we enlarge $\mathcal{P}$ by all combinations of that rule in which some occurrences of S in the right-hand side are replaced by symbols from C_S.

We construct an spRWWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ by taking $\mathcal{G} = \mathcal{F} \cup \mathcal{N}$, $\mathcal{Q} = \{ q_0, q_1 \}$, and by defining Δ as follows. Recall that $\Delta = \Delta_1 \cup \Delta_2$. For each production from $\mathcal{P}$ of type $F(x_1, \ldots, x_n) \rightarrow t$, where $n \geq 0$, $F \in \mathcal{N}_n$, $||t|| \geq 2$, and $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{N}, \mathcal{X}_n)$, we add the linear rewrite transitions

$$q_0(t) \rightarrow F(x_1, \ldots, x_n) \text{ and } q_1(t) \rightarrow F(x_1, \ldots, x_n)$$

to Δ_2. Note that all these rewrite transitions are size-reducing, because the grammar G is growing. For each constant $t \in C_S$, we add the final transition $q_0(t) \rightarrow t$ to Δ_1. Additionally, we put the transition $q_0(S) \rightarrow S$ into Δ_1. Moreover, for each symbol $F \in (\mathcal{F}_n \cup \mathcal{N}_n)$, where $n > 0$, Δ_1 contains the rules

$$q_i(F(x_1, \ldots, x_n)) \rightarrow F(x_1, \ldots, q_1(x_j), \ldots, x_n),$$

for all $i \in \{ 0, 1 \}$ and all $j \in \{ 1, \ldots, n \}$. Here state q_1 is used to guarantee that the final transitions can only be applied at the root of a tree.

The automaton $\mathcal{A}$ simulates all derivations of G nondeterministically and in reverse order. Let $t \in L(G)$ be a ground term generated by G, and let

$$S \Rightarrow_G t_1 \Rightarrow_G^* \cdots \Rightarrow_G^* t_i \Rightarrow_G t_{i+1} \Rightarrow_G^* \cdots \Rightarrow_G^* t_\ell = t$$

be a derivation in G. The automaton guesses in each cycle the correct production from $\mathcal{P}$. Then $\mathcal{A}$ applies the corresponding reverse transition on t_{i+1} to obtain t_i and restarts. Finally, the automaton reaches either a constant $t_1 \in C_S$ and accepts, since $t_1 \in S_{\mathcal{G}}(\mathcal{A})$, or it reaches S and accepts by the transition $q_0(S) \to S$ from Δ_1. On the other hand, for each accepting computation

$$t \hookrightarrow_{\mathcal{A}} t_{\ell-1},\ \ldots,\ t_2 \hookrightarrow_{\mathcal{A}} t_1,\ q_0(t_1) \to_{\Delta_1}^* t_1,\ \text{where } t_1 \in C_S \cup \{\, S \,\},$$

there is a corresponding derivation starting with $S \Rightarrow_G^\varepsilon t_1 \Rightarrow_G t_2 \Rightarrow_G^* \cdots$. Hence, we have $t \in L(G)$ if and only if $t \in L(\mathcal{A})$. □

As the tree language L_4 considered in Example 3 is not context-free, we obtain the following consequence.

Corollary 3. $\mathscr{L}(\textsf{lin-CFTG}) \subsetneq \mathscr{L}(\textsf{spRWWT})$.

5 Ground-Rewrite Restarting Tree Automata

Finally we consider restarting tree automata for which the rewrite transitions are restricted to ground terms.

Definition 2. *A* ground-rewrite restarting tree automaton (gr-RWWT) *is an* RWWT*-automaton* $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ *for which* Δ_2 *only contains final rewrite transitions, that is,* $\mathrm{Var}(l) = \mathrm{Var}(r) = \emptyset$ *for all transitions* $(l \to r) \in \Delta_2$.

Obviously, for ground-rewrite restarting tree automata there is no difference between the RRWWT- and the RWWT-variants. Thus, in the following we only consider the RWWT-, the RWT-, and the RT-variants.

Example 4. The non-regular context-free tree language $L_1 = \{\, f(g^n(a), g^n(a)) \mid n \geq 0 \,\}$ of Example 1 is recognized by the RT-automaton $\mathcal{A}_1' = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ that is defined by $\mathcal{F} := \{f(\cdot,\cdot), g(\cdot), a\}$, $\mathcal{Q} := \mathcal{Q}_1 := \{q_0, q_1\}$, $k := 1$, and Δ is given by the following rules:

$$\begin{array}{llll} (1) & q_0(f(x_1,x_2)) \to f(q_1(x_1), q_1(x_2)), & (3) & q_1(g(x_1)) \quad \to g(q_1(x)), \\ (2) & q_1(g(a)) \qquad \to a, & (4) & q_0(f(a,a)) \to f(a,a). \end{array}$$

The only rewrite transition of $\mathcal{A}_1'$ is rule (2), which is a final rewrite transition. Thus, $\mathcal{A}_1'$ is in fact a gr-RT-automaton.

Of course, gr-RT-automata can still recognize all regular tree languages, as the subsystem of a gr-RT-automaton that is described by Δ_1 is (essentially) an NF↓T-automaton. Thus, we have the following proper inclusion.

Corollary 4. $\mathscr{L}(\mathsf{RTG}) \subsetneq \mathscr{L}(\mathsf{gr\text{-}RT})$.

It is easily seen that the language

$$L_2 := \{\, g^n(h(g^n(a))) \mid n \geq 0 \,\}$$

belongs to the class $\mathscr{L}(\mathsf{spRT})$, since there is no branching transition required to recognize this non-regular tree language. Thus, the spRT-automaton $\mathcal{A}_2 = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ defined by $\mathcal{F} := \{g(.), h(.), a\}$, $\mathcal{Q} := \mathcal{Q}_1 := \{q_0, q_1\}$, and $k := 3$ with the top-down transitions

$$\begin{array}{ll} q_0(g(x_1)) \quad\quad \to g(q_1(x_1)), & q_1(g(x_1)) \to g(q_1(x_1)), \\ q_0(g(h(g(a)))) \to g(h(g(a))), & q_0(h(a)) \;\; \to h(a), \end{array}$$

and the sole size-reducing rewrite transition $q_1(g(h(g(x_1)))) \to h(x_1)$ accepts a ground term $t \in \mathcal{T}(\mathcal{F})$, if and only if $t \in L_2$. However, L_2 is not recognized by any gr-RWWT-automaton, as the following proposition shows.

Proposition 3. $L_2 \notin \mathscr{L}(\mathsf{gr\text{-}RWWT})$.

Proof. Assume that a gr-RWWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ can recognize L_2. Let $t = g^m(h(g^m(a))) \in L_2$ be a ground term of sufficient height $2m+1$. Then $t \notin S_{\mathcal{F}}(\mathcal{A})$, because otherwise $\mathcal{A}$ would also accept some trees that do not belong to L_2. Since $\mathcal{A}$ can perform ground-rewrites only, each accepting computation will contain a sequence of configurations of the form $t_1^{(i)}[q^{(i)}(t_2^{(i)})]$, where $t_1^{(i)} \in \mathrm{Ctx}(\mathcal{F}, \mathcal{X}_1)$, $t_2^{(i)} \in \mathcal{T}(\mathcal{G})$, and $q^{(i)} \in \mathcal{Q}$ such that $t_1^{(i)} = g^m(h(g^{m-\ell^{(i)}}(x_1)))$. Note that $\mathcal{A}$ can only remember a finite amount of information about the integer $\ell^{(i)}$ by using its internal states $q^{(i)}$ and the size-bounded ground term $t_2^{(i)}$. Thus, for a large enough value of m, there exist indices $i < j$ such that $q^{(i)} = q^{(j)}$, $t_2^{(i)} = t_2^{(j)}$, and $t_1^{(j)}[g^\ell(x_1)] = t_1^{(i)}$ for some $\ell \geq 1$. Hence, $\mathcal{A}$ can transform the term $g^m(h(g^{m+\ell}(a))) \notin L_2$ into the configuration $t_1^{(j)}[g^\ell(q^{(j)}(t_2^{(j)}))] = t_1^{(i)}[q^{(i)}(t_2^{(i)})]$. Thus, $\mathcal{A}$ will also accept the term $g^m(h(g^{m+\ell}(a)))$, which contradicts our assumption. $\square$

On the other hand, the tree language

$$L_3 := \{\, f(g^n(a), g^n(a)), f(g^n(a), g^{2n}(b)) \mid n \geq 0 \,\}$$

can be recognized by a gr-RT-automaton using the following transitions:

$$\begin{array}{rr} q_0(f(x_1,x_2)) \to f(q_1(x_1), q_2(x_2)), & q_0(f(a,a)) \to f(a,a), \\ q_0(f(a,b)) \to f(a,b), & q_1(g(x_1)) \to g(q_1(x_1)), \\ q_2(g(x_1)) \to g(q_2(x_1)), & q_1(g(a)) \to a, \\ q_2(g(a)) \to a, & q_2(g(g(b))) \to b. \end{array}$$

Note that a gr-RT-automaton can use its inherent synchronization mechanism to compare the number of g's in each branch by performing corresponding reductions at the leafs. However, L_3 is not recognized by any spRWT-automaton, because such a synchronization mechanism is missing there.

Proposition 4. $L_3 \notin \mathscr{L}(\mathsf{spRWT})$.

Proof. Assume that a spRWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{F}, \mathcal{Q}, q_0, k, \Delta)$ can recognize L_3, that is, $L(\mathcal{A}) = L_3$ holds. Let $t = f(g^m(a), g^m(a)) \in L_3$ be a ground term of sufficient height $m+1$. Now consider the following cases:

1. Assume that t is accepted by $\mathcal{A}$ in a computation using a top-down rewrite transition including the outermost symbol f, that is, a top-down rewrite transition of the form $q_0(f(g^i(x_1), g^j(x_2)))) \to f(g^{i-\ell}(x_1), g^{j-\ell}(x_2))$ is used, where $0 < \ell \leq i, j < k$. Then $\mathcal{A}$ could use this transition to transform the term $f(g^{m+\ell}(a), g^{2m+\ell}(b)) \notin L_3$ into the term $f(g^m(a), g^{2m}(b)) \in L(\mathcal{A})$. Thus, together with the latter $\mathcal{A}$ would also accept the former, contradicting our assumption.
2. Assume that t is accepted by $\mathcal{A}$ in a computation using only top-down rewrite or final rewrite transitions not including the outermost symbol f. Then $\mathcal{A}$ must decide in which branch it will reduce the number of g's. Without loss of generality we may assume that after the first of these reductions a stateless configuration of the form $f(g^m(a), g^{m-\ell}(a))$ is reached for some $\ell \geq 1$. Thus, $\mathcal{A}$ also accepts the term $f(g^m(a), g^{m-\ell}(a)) \notin L_3$. Again this contradicts our assumption, which completes the proof. □

Proposition 3 and Proposition 4 yield the following consequences. In particular, it follows that $\mathscr{L}(\mathsf{gr\text{-}RT})$ and $\mathscr{L}(\mathsf{spRT})$ are incomparable under set inclusion.

Corollary 5. $L_2 \in \mathscr{L}(\mathsf{spRT}) \smallsetminus \mathscr{L}(\mathsf{gr\text{-}RWWT})$ *and* $L_3 \in \mathscr{L}(\mathsf{gr\text{-}RT}) \smallsetminus \mathscr{L}(\mathsf{spRWT})$.

By combining the restriction of admitting ground rewrite transitions only and the restriction of walking down a single path only we obtain the ground-rewrite single-path restarting tree automaton.

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

In the proof of Theorem 1 it is shown how to construct an spRT-automaton that recognizes a given regular tree language. In fact, all rewrite transitions of that spRT-automaton are ground. Thus, we obtain the following.

Corollary 6. $\mathscr{L}(\mathsf{RTG}) \subseteq \mathscr{L}(\mathsf{gr\text{-}spRT})$.

It remains open whether $\mathsf{gr\text{-}spRWWT}$-automata can recognize any non-regular tree languages. For example, it can be shown that $L_2, L_3 \notin \mathscr{L}(\mathsf{gr\text{-}spRWWT})$ by combining the arguments used in the proofs of Propositions 3 and 4.

6 Conclusion

We have presented two types of restrictions for restarting tree automata: single-path restarting tree automata that only walk down a single path in the tree considered, and ground-rewrite restarting automata that only perform rewrite

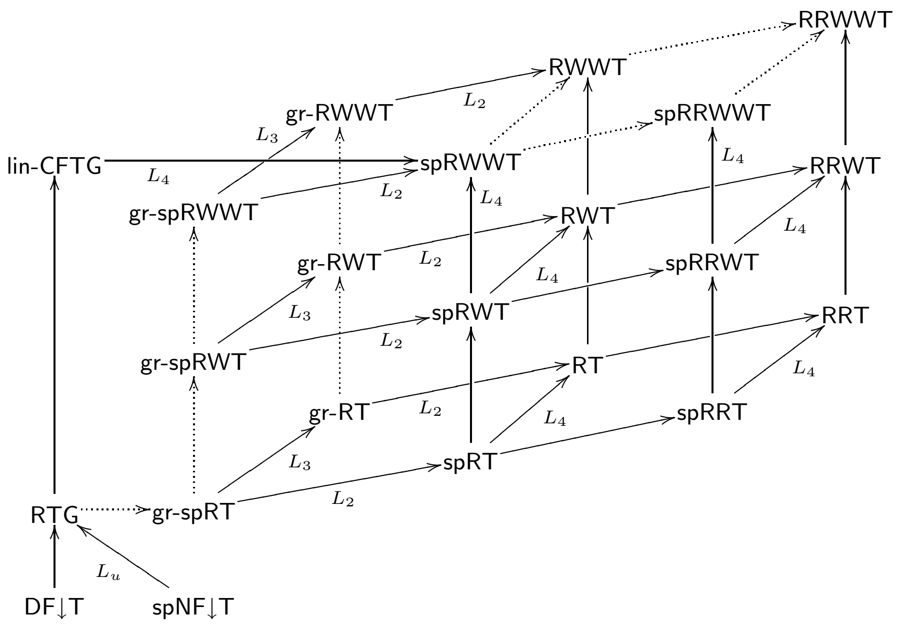


Fig. 1. Inclusions between language classes defined by (single-path) restarting tree automata and classes generated by various tree grammars. An arrow denotes a proper inclusion, while a dotted arrow denotes an inclusion that is not known to be proper.

steps at the subterms of the actual tree that are ground. The inclusion relations between the language classes recognized by the various types of restarting tree automata are summarized in the diagram in Figure 1. However, the following interesting questions remain open at this time:

1. Is any of the inclusions $\mathscr{L}(\mathsf{gr\text{-}RT}) \subseteq \mathscr{L}(\mathsf{gr\text{-}RWT}) \subseteq \mathscr{L}(\mathsf{gr\text{-}RWWT})$ proper?
2. Do gr-spRWT-automata only accept regular tree languages, that is, does the equality $\mathscr{L}(\mathsf{RTG}) = \mathscr{L}(\mathsf{gr\text{-}spRWT})$ hold? We conjecture that this is indeed the case. If so, then it also follows that $\mathscr{L}(\mathsf{RTG}) = \mathscr{L}(\mathsf{gr\text{-}spRWWT})$ holds, as each gr-spRWWT-automaton $\mathcal{A} = (\mathcal{F}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$ can be converted into the gr-spRWT-automaton $\mathcal{B} = (\mathcal{G}, \mathcal{G}, \mathcal{Q}, q_0, k, \Delta)$. If $L(\mathcal{B})$ is regular, then $L(\mathcal{A}) = L(\mathcal{B}) \cap \mathcal{T}(\mathcal{F})$ is also regular, as $\mathcal{T}(\mathcal{F})$ is regular and as $\mathscr{L}(\mathsf{RTG})$ is closed under intersection.
3. For a given regular tree language a gr-spRT-automaton with a finite simple language can be constructed (see the proof of Theorem 1). This raises the question of whether each gr-spRWWT-automaton $\mathcal{A}$ can be converted into an equivalent gr-spRWWT-automaton $\mathcal{B}$ such that the simple language $S_{\mathcal{G}}(\mathcal{B})$ is finite. Essentially this property corresponds to the notion of *weak cyclic form* for restarting automata on strings (see, e.g., [5,7]).

References

1. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (September 2005)
2. Fujiyoshi, A., Kasai, T.: Spinal-formed context-free tree grammars. Theory of Computing Systems 33, 59–83 (2000)
3. Gécseg, F., Steinby, M.: Tree Languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 1–68. Springer, Heidelberg (1997)
4. Jančar, P., Mráz, F., Plátek, M., Vogel, J.: Restarting automata. In: Reichel, H. (ed.) FCT 1995. LNCS, vol. 965, pp. 283–292. Springer, Heidelberg (1995)
5. Mráz, F., Plátek, M., Procházka, M.: On special forms of restarting automata. Grammars 2, 223–233 (1999)
6. Otto, F.: Restarting automata. In: Ésik, Z., Martín-Vide, C., Mitrana, V. (eds.) Recent Advances in Formal Languages and Applications. Studies in Computational Intelligence, vol. 25, pp. 269–303. Springer, Heidelberg (2006)
7. Plátek, M.: Weak cyclic forms of restarting automata. In: Rozenberg, G., Thomas, W. (eds.) DLT 1999, Proc., pp. 115–124. World Scientific, Singapore (2000)
8. Stamer, H.: Restarting Tree Automata – Formal Properties and Possible Variations. Doctoral Dissertation, Fachbereich Elektrotechnik/Informatik, Universität Kassel (2008)
9. Stamer, H., Otto, F.: Restarting tree automata. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 510–521. Springer, Heidelberg (2007)
10. Stamer, H., Otto, F.: Restarting tree automata and linear context-free tree languages. In: Bozapalidis, S., Rahonis, G. (eds.) CAI 2007. LNCS, vol. 4728, pp. 275–289. Springer, Heidelberg (2007)
11. Vijay-Shankar, K., Weir, D.J.: The equivalence of four extensions of context-free grammars. Mathematical Systems Theory 27, 511–546 (1994)

Parallel Communicating Grammar Systems with Regular Control

Dana Pardubská[1,⋆], Martin Plátek[2,⋆⋆], and Friedrich Otto[3]

[1] Department of Computer Science,
Comenius University, Bratislava, Slovak Republic
pardubska@dcs.fmph.uniba.sk

[2] Department of Computer Science,
Charles University, Prague, Czech Republic
Martin.Platek@mff.cuni.cz

[3] Fachbereich Elektrotechnik/Informatik,
Universität Kassel, Kassel, Germany
otto@theory.informatik.uni-kassel.de

Abstract. *Parallel communicating grammar systems with regular control* (RPCGS, for short) are introduced, which are obtained from returning regular parallel communicating grammar systems by restricting the derivations that are executed in parallel by the various components through a regular control language. For the class of languages that are generated by RPCGSs with constant communication complexity we derive a characterization in terms of a restricted type of *freely rewriting restarting automaton.* From this characterization we obtain that these languages are semi-linear, and that centralized RPCGSs with constant communication complexity are of the same generative power as non-centralized RPCGSs with constant communication complexity.

Keywords: parallel communicating grammar system, regular control, bounded communication complexity, freely rewriting restarting automaton.

1 Introduction

The notion of a parallel communicating grammar system is a theoretical model for a finite group of agents that concurrently work on the solution of a problem [1,3,12]. These systems are able to create copies of generated strings and their images under regular mappings in a very natural way. This ability has a strong similarity to the generation of sentence segments in the Czech language (and some other natural languages). However, the synonymy and homonymy of segments has not yet been modelled appropriately [4].

⋆ Partially supported by the Slovak Grant Agency for Science (VEGA) under contract "1/0726/09 - Algorithmic and Complexity Issues in Information Processing."

⋆⋆ Partially supported by the Grant Agency of the Czech Republic under Grant-No. 405/08/0681 and by the program Information Society under project 1ET100300517.

In [9] the current authors studied *returning regular parallel communicating grammar systems* (PCGS(REG) or PCGS, for short) with *constant communication complexity*, establishing a transformation to *freely rewriting restarting automata* (FRR automata, for short) of a very restricted form. As a consequence it follows that the languages generated by PCGSs with constant communication complexity are semi-linear, and that these languages have characteristic analysis of polynomial size, which, in addition, can even be computed in polynomial time. However, it appears that the restricted type of FRR automaton considered in [9] is still more expressive than PCGSs with constant communication complexity. This seems to result from the very restriction put on the communication of the PCGSs considered. Within a generative section of a computation, each component grammar of a PCGS works in complete isolation from all other component grammars[1] apart from the effects of the global clock that ensures that each component grammar makes a single generative step in each unit of time.

In order to model the linguistic notion of segment this kind of communication is not sufficient. Therefore we introduce an extension of the PCGS by providing a regular control for proper derivations. This yields the so-called *regulated returning regular parallel communicating grammar systems* (RPCGS(REG) or simply RPCGS, for short). Here a regular language is used to restrict the set of admissible derivations of a PCGS. This idea is borrowed from the field of 'regulated rewriting' (see, e.g., [2]); a similar idea was used by G. Păun in [11] to define parallel communicating grammar systems with *rule synchronization*. As our main result we derive a characterization of the class of languages that are generated by RPCGSs with constant communication complexity in terms of so-called *skeleton preserving FRR automata*. In fact, we prove that *centralized* RPCGSs with constant communication complexity are equivalent (in generative power) to *non-centralized* RPCGSs with constant communication complexity. Here an RPCGS Π is called *centralized* if the master grammar of Π (see Subsections 2.1 and 2.2 for the definitions of PCGSs and RPCGSs, respectively) is the only one that can initiate communication steps, while it is called *non-centralized* if any component grammar can do that. From the above mentioned characterization it follows that the languages generated by RPCSGs with constant communication complexity are semi-linear, that their characteristic analysis is of polynomial size, and that the latter can even be computed in polynomial time.

This paper is structured as follows. In Section 2 we give the (informal) definitions of PCGS, RPCGS, and FRR automata, and present some basic facts about them. In Section 3, which constitutes the technical main part of the paper, we introduce the notion of skeleton preserving FRR automaton and present the simulation results described above. Since the skeleton preserving FRR automaton considered here is a special type of the FRR automaton as it is used in [9], the announced results on semi-linearity and on the characteristic analysis follow as in [9]. Finally, some closing remarks are found in Section 4.

[1] This is reminiscent of the behaviour of a distributed system.

2 Regulated Parallel Communicating Grammar Systems and FRR Automata

Here we informally introduce the various grammar systems and models of restarting automata that we will consider in this paper.

2.1 Parallel Communicating Grammar Systems

A *returning regular parallel communicating grammar system* (PCGS, for short) *of degree* m (≥ 1) is defined as an $(m+3)$-tuple $\Pi = (N, K, T, G_1, \ldots, G_m)$, where N is a finite alphabet of symbols called *nonterminals*, $K = \{Q_1, \ldots, Q_m\}$ is a set of special symbols called *communication symbols* (or *query symbols*) that is a subset of N, T is a finite alphabet of symbols called *terminals* that is disjoint from N, and $G_i = (N, T, S_i, P_i)$ are regular grammars ($1 \leq i \leq m$). The grammars G_i are the *component grammars* of Π, and the component grammar G_1 is called the *master* of the system.

A *configuration* of Π is an m-tuple $C = (x_1A_1, \ldots, x_mA_m)$, where $x_i \in T^*$ and $A_i \in (N \cup \{\varepsilon\})$ ($1 \leq i \leq m$). The string x_iA_i is the *i-th component of configuration* C. The *nonterminal cut* of C is the m-tuple $N(C) = (A_1, A_2, \ldots, A_m)$. If $N(C)$ contains a communication symbol, then it is called an *NC-cut*, denoted by $NC(C)$.

A *derivation* of Π is a sequence of configurations $D = C_0, C_1, \ldots, C_t$ starting from the initial configuration $C_0 = (S_1, \ldots, S_m)$ such that, for all $j < t$, C_{j+1} is obtained from C_j by a single generative step or a single communication step. This is written as $C_j \Rightarrow C_{j+1}$. If no communication symbol occurs in C_j, then a *generative step* is performed. It consists of synchronously applying a single rewrite step of grammar G_i to the i-th component of C_j for all $i = 1, \ldots, m$. Components of C_j that are terminal strings remain unchanged. If, however, any component of C_j contains a nonterminal that cannot be rewritten, then the derivation is blocked. Further, if the first component of C_j is a terminal word w, then the derivation is complete, and w is the result of this derivation. In this situation D is usually denoted as $D(w)$. A maximal sub-sequence of D that only contains generative steps is called a *generative section* of D.

If one or more communication symbols are present in $C_j = (\alpha_1^{(j)}, \ldots, \alpha_m^{(j)})$, then a *communication step* is performed. It consists of replacing each occurrence of each communication symbol Q_l ($1 \leq l \leq m$) by the phrase $\alpha_l^{(j)}$, provided that $\alpha_l^{(j)}$ itself does not contain a communication symbol. Such an individual replacement is called a *communication.* In addition, the component l is reset[2] to its start symbol S_l. Obviously, in a single communication step at most $m-1$ communications can be performed. Communication steps are performed until all communication symbols have been replaced, or until the derivation is blocked. A maximal sub-sequence of D that only contains communication steps is called a *communication section* of D. Thus, the communication steps divide D into generative sections and communication sections.

[2] Because of this the PCGS is called 'returning.'

By $\Rightarrow^+$ we denote the transitive closure of the relation $\Rightarrow$ above. The (*terminal*) *language* $L(\Pi)$ generated by Π is the set of all terminal words that are generated by the component G_1 (the *master* of the system):

$$L(\Pi) = \{\, w \in T^* \mid \exists\, \alpha_2, \ldots, \alpha_m : (S_1, \ldots, S_m) \Rightarrow^+ (w, \alpha_2, \ldots, \alpha_m) \,\}.$$

Several useful notions are associated with a derivation $D(w)$ in Π:

- $g(i,j)$ (or $g(i,j,D(w))$), the (i,j)-*(generative) factor* of $D(w)$, is the terminal word that is generated by the component grammar G_i within the j-th generative section of $D(w)$. Observe that each symbol of w belongs unambiguously to one of the factors $g(i,j)$.
- The *communication structure* $CS(D(w))$ of $D(w)$ captures the connection between the terminal word w and its particular derivation $D(w)$):

  $$CS(D(w)) = (i_1, j_1), (i_2, j_2), \ldots, (i_r, j_r), \text{ if } w = g(i_1, j_1) g(i_2, j_2) \cdots g(i_r, j_r).$$
- $n(i,j)$ (or $n(i,j,D(w))$) denotes the number of occurrences of $g(i,j)$ in w.
- For $j \geq 1$ let $N(j, D(w)) = \sum_{i=1}^{m} n(i,j,D(w))$. The *degree of distribution* $DD(D(w))$ of $D(w)$ is defined as $DD(D(w)) = \max_j N(j, D(w))$. Thus, $DD(D(w))$ is the maximal number of occurrences of factors $g(i,j)$ in w that are generated in the same generative section of $D(w)$.
- The *communication sequence*, resp. the *NC-sequence* ($NCS(D)$), is defined as the sequence of all NC-cuts in the (sub-)derivation D. Realize that the communication sequence $NCS(D(w))$ unambiguously defines the communication structure of $D(w)$. Moreover, the set of words with the same communication sequence/structure is in general infinite.

A *cycle in a derivation* D is a smallest (continuous) sub-derivation $\mathcal{C} = C_1, \ldots, C_j$ of D such that $N(C_1) = N(C_j)$. If *none* of the nonterminal cuts in $\mathcal{C}$ contains a communication symbol, then the whole cycle is contained in a generative section; we speak about a *generative cycle* in this case.

If there is a generative cycle in the derivation $D(w)$, then manifold repetition[3] of this cycle results in a derivation of some terminal word. However, repetition or deletion of a generative cycle does neither change the communication sequence nor the communication structure of a derivation. We call a derivation $D(w)$ *reduced*, if every repetition of any of its generative cycles leads to a *longer* terminal word. Obviously, to every derivation $D(w)$ there is an equivalent reduced derivation $D'(w)$ of the same word. In what follows, we consider only derivations that are reduced.

2.2 Regulated Parallel Communicating Grammar Systems

Within a generative section of a derivation, each component grammar of a PCGS works in complete isolation from all other component grammars apart from the

[3] Deletion of a cycle is also possible.

synchronization enforced by the 'global clock'. Here we extend the PCGS by establishing a regular control for the admissible derivations. In this way the PCGS is turned into a truly *parallel* device.

Let Π be a PCGS, and let

$$(A_{0,1},\ldots,A_{0,m}),(\alpha_{1,1}A_{1,1},\ldots,\alpha_{1,m}A_{1,m}),(\alpha_{1,1}\alpha_{2,1}A_{2,1},\ldots,\alpha_{1,m}\alpha_{2,m}A_{2,m}),\\ \ldots,(\alpha_{1,1}\cdots\alpha_{s,1}A_{s,1},\ldots,\alpha_{1,m}\cdots\alpha_{s,m}A_{s,m})$$

be the sub-derivation that corresponds to the j-th generative section of a Π-derivation $D(w)$. Here $(A_{0,1},\ldots,A_{0,m})$ is the nonterminal cut at the beginning of this generative section, and $(A_{l-1,i}\rightarrow\alpha_{l,i}A_{l,i})$ is the production of component grammar G_i $(1\leq i\leq m)$ that is applied in the l-th step of this sub-derivation $(1\leq l\leq s)$. If $A_{l-1,i}$ is the empty word, then we simply take $\alpha_{l,i}A_{l,i}=\varepsilon$ as well. With this sub-derivation we associate the following *extended j-trace*:

$$\text{ex-T}(D(w),j)=\begin{pmatrix}A_{0,1}\\A_{0,2}\\\cdots\\A_{0,m}\end{pmatrix}\begin{pmatrix}\alpha_{1,1}A_{1,1}\\\alpha_{1,2}A_{1,2}\\\cdots\\\alpha_{1,m}A_{1,m}\end{pmatrix}\begin{pmatrix}\alpha_{2,1}A_{2,1}\\\alpha_{2,2}A_{2,2}\\\cdots\\\alpha_{2,m}A_{2,m}\end{pmatrix}\cdots\cdots\begin{pmatrix}\alpha_{s,1}A_{s,1}\\\alpha_{s,2}A_{s,2}\\\cdots\\\alpha_{s,m}A_{s,m}\end{pmatrix},$$

which completely describes this sequence of generative steps. Assume that $D(w)$ has k generative sections. Then

$$\text{ex-T}(D(w))=\text{ex-T}(D(w),1),\text{ex-T}(D(w),2),\ldots,\text{ex-T}(D(w),k)$$

is the *extended trace* of $D(w)$, which is another representation of $D(w)$. Observe that, for all $j<k$, the nonterminal cut at the beginning of the $(j+1)$-st generative section is uniquely determined by the nonterminal cut at the end of the j-th generative section, which is actually an NC-cut, as the former is obtained from the latter through a (uniquely determined) sequence of communication steps.

By associating a new symbol with each m-tuple $(r_1,r_2,\ldots,r_m)$, where r_i is the right-hand side of a production of grammar G_i $(1\leq i\leq m)$, and with each possible nonterminal cut $(B_1,B_2,\ldots,B_m)$ of Π, we obtain a finite alphabet Ω_Π such that each extended trace can be interpreted as a string over Ω_Π. Under this interpretation, the language

$$L_{\text{ex-T}}(\Pi)=\{\,\text{ex-T}(D(w))\mid w\in L(\Pi)\,\}\subseteq\Omega_\Pi^*$$

is actually regular. Now we are ready to define the main notion of this paper, the *regulated parallel communicating grammar systems.*

Definition 1. *Let Π be a PCGS, and let R be a regular language over Ω_Π. Then the language $L(\Pi,R)$ that is* generated by Π regulated by R *is defined as*

$$L(\Pi,R)=\{\,w\in L(\Pi)\mid\exists\,\Pi\text{-derivation }D(w)\,:\,\text{ex-T}(D(w))\in R\,\}.$$

The pair (Π,R) is called a regulated parallel communicating grammar system *(RPCGS, for short). We say that the PCGS Π is* regulated *by R, and also that a Π-derivation $D(w)$ satisfying* ex-T$(D(w))\in R$ *is* regulated *by R.*

Thus, in an RPCGS (Π, R), a Π-derivation $D(w)$ is admissible only if it satisfies the additional condition that ex-T$(D(w))$ belongs to the regular language R. This is exactly the same mechanism that is used in (*context-free*) *grammars with regular control* (see, e.g., [2]). In the case of regular components, the *parallel communicating grammar system with rule synchronization* studied by Păun in [11] is a special case of regulated parallel communicating grammar systems, as the control languages used by Păun correspond to regular languages of the form M^*, where M is a finite subset of Ω_Π.

It is easily seen that the regular control set $R \subseteq \Omega_\Pi^*$ of an RPCGS (Π, R) can be modified in such a way that, for each regulated Π-derivation $D(w)$, there exists an equivalent regulated Π-derivation that is reduced.

Informally the *communication complexity* com(D) of a Π-derivation D is the number of communications performed within this derivation; analogously, the *distribution complexity* of D is the degree of distribution $DD(D)$ defined above, and the *generation complexity* of D is the number of generative sections in D. The *communication complexity*, the *distribution complexity*, and the *generation complexity* of a language and the associated complexity classes are now defined in the usual way (always considering the corresponding maximum). These three complexity measures are closely related.

Fact 2. *Let Π be a* (*regulated*) *PCGS of degree m, and let $c(n)$, $g(n)$, and $d(n)$ be the communication complexity, the generation complexity, and the distribution complexity of $L(\Pi)$, respectively. Then $g(n) \in O(c(n))$ and $d(n) \in O(m^{c(n)})$.*

Motivated by the analysis by reduction we are mainly interested in those classes of languages for which these three complexity measures are bounded from above by constants. For natural numbers c, d, g, we denote the corresponding communication complexity class for PCGSs by $\mathsf{COM}(c)$, the distribution complexity class by d-DD, the generation complexity class by g-DG, and the complexity class obtained by combining the restriction on the distribution complexity with the restriction on the generation complexity by d-g-DDG. Analogously, the corresponding complexity classes for RPCGSs are denoted by $\mathsf{RCOM}(c)$, d-RDD, g-RDG, and d-g-RDDG, respectively.

When only the master component of an (R)PCGS may use communication symbols, then we speak of a *centralized* (R)PCGS. This is in contrast to the general case of *non-centralized* (R)PCGSs, in which each component grammar may use communication symbols. By d-g-$\mathsf{C(R)DDG}$ we will denote the class of languages that are generated by centralized (R)PCGSs with distribution and generation complexity d and g, respectively.

Fact 3. *Let (Π, R) be an RPCGS with constant communication complexity. Then there exists a regular language R' over Ω_Π such that $L(\Pi, R) = L(\Pi, R')$, and every derivation regulated by R' is reduced.*

It follows from Fact 3 that, if (Π, R) is an RPCGS of degree m with constant communication complexity, then there exists a constant $e(\Pi, R)$ such that, whenever $D(w)$ is a reduced regulated Π-derivation the j-th generative section of which contains more than $e(\Pi, R)$ many generative steps, then at least one of the factors $g(i, j, D(w))$ $(1 \le i \le m)$ is changed in this generative section. Based on pumping arguments also the following observation follows easily.

Fact 4. *If (Π, R) is an RPCGS with constant communication complexity, then the set of regulated Π-derivations that do* not *contain a generative cycle is finite.*

As our first result we separate the language class d-g-(C)RDDG from the class d-g-(C)DDG by presenting a corresponding example language.

Example 1. Let $\Sigma = \{0, 1\}$, let h be the morphism that is induced by $h(0) = 1$ and $h(1) = 0$, let $L = \{ wh(w) \mid w \in \Sigma^* \}$, and let Π be the PCGS that consists of the following two component grammars G_1 and G_2:

$$\begin{aligned} G_1 &: S_1 \to N,\ N \to 0N \mid 1N \mid Q_2,\ N' \to \varepsilon, \\ G_2 &: S_2 \to M,\ M \to 0M \mid 1M \mid N'. \end{aligned}$$

For the regular control we take the language

$$R = \binom{S_1}{S_2} \left\{ \binom{0N}{1M}, \binom{1N}{0M} \right\}^* \binom{Q_2}{N'} \binom{N'}{S_2} \binom{\varepsilon}{M}.$$

First consider the unregulated case. In the first step of a Π-derivation both grammars use their unique starting rule. Thereafter, both grammars choose a symbol 0 or 1 from Σ nondeterministically and independently of each other. Finally, one communication followed by a final generative step completes the derivation. It is easily seen that $L(\Pi) = \{ ww' \mid w, w' \in \Sigma^*,\ |w| = |w'| \}$. In the regulated case, however, the regular control language R coordinates the nondeterministic choices of G_1 and G_2. It follows that $L(\Pi, R) = L$.

Observe that there is a kind of indirect communication between the two component grammars in every step of a regulated derivation in the RPCGS (Π, R) from Example 1, while the communication complexity of (Π, R) is just one. On the other hand, there is no unregulated PCGS with constant communication complexity for the language generated by (Π, R).

Theorem 1. $L = \{ wh(w) \mid w \in \Sigma^* \} \notin \mathsf{COM}(\mathsf{O}(1))$.

Proof. Assume that $L \in \mathsf{COM}(k)$ for some constant $k \in \mathbb{N}$, and let $\Pi = (N, K, \Sigma, G_1, \ldots, G_m)$ be a PCGS generating L with communication complexity at most k. We consider a word $w = \alpha_1\alpha_2 \cdots \alpha_n h(\alpha_1) \cdots h(\alpha_n)$ that satisfies the following conditions:

1. $n > 2^{k+1}$, and
2. for all $i = 1, \ldots, n$, $\alpha_i = 0^{j_i}1^{j_i}$, where j_i is chosen in such a way that neither 0^{j_i} nor 1^{j_i} can be generated without a generative cycle.

Let D be a reduced Π-derivation of w. The choice of n guarantees that there exists an index i such that the complete sub-word α_i is generated by one of the component grammars of Π within a single generative section g. For brevity we call this particular component grammar I.

From the choice of $\alpha_i = 0^{j_i}1^{j_i}$ it follows that there are two generative cycles C_0 and C_1 of length ℓ_0 and ℓ_1, respectively, within the generative section g such that component grammar I generates a nonempty factor 0^{Δ_0} of α_i in cycle C_0 and a nonempty factor 1^{Δ_1} of α_i in cycle C_1. Since D is a reduced derivation, repetitions of either C_0 and/or C_1 will result in a longer terminal word. By simple pumping arguments it follows that there must be another component grammar, say H, that generates the factor 1^{Δ_0} of $h(\alpha_i)$ in cycle C_0 and the factor 0^{Δ_1} of $h(\alpha_i)$ in cycle C_1.

Now, we consider two derivations $D(0)$ and $D(1)$ such that $D(0)$ is obtained from D by repeating the cycle C_0 $(\ell_1 + 1)$ times, while $D(1)$ is obtained from D by repeating the cycle C_1 $(\ell_0 + 1)$ times. Thus, in both derivations we have added the same number $\ell_0 \cdot \ell_1$ of generative steps to generative section g. This implies that we obtain a valid Π-derivation $D(0,1)$, if I behaves as in derivation $D(0)$, while H (and all other components) behave as in derivation $D(1)$. The word w' generated by this derivation has the form

$$w' = \alpha_1 \cdots \alpha_{i-1}\beta_i\alpha_{i+1} \cdots \alpha_n h(\alpha_1) \cdots h(\alpha_{i-1})\gamma_i h(\alpha_{i+1}) \cdots h(\alpha_n),$$

where $\beta_i = 0^{j_i+(\Delta_0 \cdot \ell_1)}1^{j_i}$ and $\gamma_i = 1^{j_i}0^{j_i+(\Delta_1 \cdot \ell_0)}$. As $w' \notin L$, this contradicts our assumption that $L(\Pi) = L$. It follows that L is not generated by any PCGS with constant communication complexity. □

As the RPCGS from Example 1 is centralized, we obtain the following separation results.

Theorem 2. *For all $d, g \geq 2$,*
(a) d-g-$\mathsf{DDG} \subsetneq d$-$g$-$\mathsf{RDDG}$ *and* (b) d-g-$\mathsf{CDDG} \subsetneq d$-$g$-$\mathsf{CRDDG}$.

2.3 Freely Rewriting Restarting Automata

Here we introduce the particular type of restarting automaton we are interested in in this paper.

A *freely rewriting restarting automaton*, abbreviated as FRR automaton, is a nondeterministic machine that consists of a finite-state control, a single flexible tape with end markers, and a read/write window of a fixed size $k \geq 1$ that can move along this tape. Formally, it is described by an 8-tuple $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$, where Q denotes a finite set of (internal) states that contains the initial state q_0, Σ is a finite input alphabet, and Γ is a finite tape alphabet that contains Σ. The elements of $\Gamma \setminus \Sigma$ are called *auxiliary symbols*. The additional symbols $¢, \$ \notin \Gamma$ are used as markers for the left and right end of the workspace, respectively. They cannot be removed from the tape. The behavior of M is described by a transition function δ that associates a finite set of transition steps to each pair of the form (q, x), where q is a state and x is a possible content of the read/write window.

There are four types of transition steps: *move-right steps*, *rewrite steps*, *restart steps*, and *accept steps.* A *move-right step* simply shifts the read/write window one position to the right and changes the internal state. A *rewrite step* causes M to replace a non-empty prefix u of the content of the read/write window by a word v satisfying $|v| \leq |u|$, and to change the state. Further, the read/write window is placed immediately to the right of the string v. However, some restrictions apply in that neither a move-right step nor a rewrite step can shift the read/write window across the right sentinel \$. A *restart step* causes M to place its read/write window over the left end of the tape, so that the first symbol it sees is the left sentinel ¢, and to reenter the initial state q_0. Finally, an *accept step* simply causes M to halt and accept. However, it is more convenient for our purposes to describe FRR automata through so-called *meta-instructions* (see below).

A *configuration* of M is described by a string $\alpha q \beta$, where $q \in Q$, and either $\alpha = \varepsilon$ and $\beta \in \{¢\} \cdot \Gamma^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot \Gamma^*$ and $\beta \in \Gamma^* \cdot \{\$\}$; here q represents the current state, $\alpha\beta$ is the current content of the tape, and it is understood that the window contains the first k symbols of β or all of β when $|\beta| \leq k$. A *restarting configuration* is of the form $q_0 ¢ w \$$, where $w \in \Gamma^*$.

Any computation of M consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration. The window is shifted along the tape by move-right and rewrite operations until a restart operation is performed and thus a new restarting configuration is reached. If no further restart operation is performed, then the computation necessarily finishes in a halting configuration – such a phase is called a *tail.* It is required that in each cycle M performs at least one rewrite step that is strictly length-decreasing. Thus, each cycle strictly reduces the length of the tape. We use the notation $w \vdash_M^c z$ to denote a cycle of M that begins with the restarting configuration $q_0 ¢ w \$$ and ends with the restarting configuration $q_0 ¢ z \$$; the relation $\vdash_M^{c^*}$ is the reflexive and transitive closure of $\vdash_M^c$. An FRR automaton is called *t-rewriting* for an integer $t \geq 1$, if it does not perform more than t rewrite steps in any cycle or tail. By t-FRR we denote the class of all t-rewriting FRR automata.

A *rewriting meta-instruction* for a t-rewriting FRR automaton M is of the form

$$(E_1, u_1 \to v_1, E_2, u_2 \to v_2, E_3, \ldots, E_i, u_i \to v_i, E_{i+1}),$$

where $1 \leq i \leq t$, $E_1, \ldots, E_{i+1}$ are regular expressions, and $u_j, v_j \in \Gamma^*$ are strings satisfying $k \geq |u_j| \geq |v_j|$ for all $j = 1, \ldots, i$. The rules $u_j \to v_j$, $1 \leq j \leq i$, embody rewrite steps of M. On trying to execute this meta-instruction, M will get stuck (and so reject) starting from the restarting configuration $C_1 = q_0 ¢ w \$$, if w does not admit a factorization of the form $w = w_1 u_1 w_2 u_2 \cdots w_i u_i w_{i+1}$ such that $¢ w_1 \in E_1$, $w_2 \in E_2$, $\ldots$, $w_{i+1} \$ \in E_{i+1}$. On the other hand, if w does have factorizations of this form, then one such factorization is chosen nondeterministically, and C_1 is transformed into the restarting configuration $C_2 = q_0 ¢ w_1 v_1 w_2 v_2 w_3 \cdots w_i v_i w_{i+1} \$$. To describe the tails of accepting computations of M we use meta-instructions of the form $(¢ \cdot E \cdot \$, \mathsf{Accept})$, which accepts the sentences from the regular language E.

A word $w \in \Gamma^*$ is *accepted* by M, if there is an accepting computation which starts from the restarting configuration $q_0 ¢ w \$$. By $L_{\mathrm{C}}(M)$ we denote the so-called *characteristic language of* M, which is the language consisting of all words accepted by M. By Pr^{Σ} we denote the projection from Γ^* onto Σ^*, that is, Pr^{Σ} is the morphism defined by $a \mapsto a$ $(a \in \Sigma)$ and $A \mapsto \varepsilon$ $(A \in \Gamma \setminus \Sigma)$. If $v := \mathsf{Pr}^{\Sigma}(w)$, then v is the *Σ-projection* of w, and w is an *expanded version* of v. For a language $L \subseteq \Gamma^*$, $\mathsf{Pr}^{\Sigma}(L) := \{ \mathsf{Pr}^{\Sigma}(w) \mid w \in L \}$. Further, for $K \subseteq \Gamma$, $|x|_K$ denotes the number of occurrences of symbols from K in x.

In recent papers (see, e.g., [6]) restarting automata were mainly used as acceptors. The main focus was on the so-called (*input*) *language* of a restarting automaton M, that is, the set $L(M) := L_{\mathrm{C}}(M) \cap \Sigma^*$. Here, motivated by linguistic considerations to model the analysis by reduction with parallel processing, we are rather interested in the so-called *proper language of* M, which is the set of words $L_{\mathrm{P}}(M) := \mathsf{Pr}^{\Sigma}(L_{\mathrm{C}}(M))$. Realize that the main difference between the input language and the proper language lies in the way in which auxiliary symbols are inserted into the (terminal) words of the language. For words from the input language, auxiliary symbols can only be inserted by the automaton itself in the course of a computation, while for words from the proper language, the auxiliary symbols are provided beforehand by an outside source, e.g., a linguist.

Based on the number of auxiliary symbols that are allowed in a word two different classes of FRR automata have been considered in the literature –*lexicalized* and *linearized* FRR automata [7,8,10]. Here, however, we will restrict the use of auxiliary symbols even further, as we will only consider FRR automata for which the number of auxiliary symbols that may occur concurrently on the tape is bounded from above by a constant.

In a real process of analysis by reduction of a sentence of a natural language it is desired that whatever is done within the process does not change the correctness of the sentence. For restarting automata this property can be formalized as follows.

Definition 5. (Correctness Preserving Property.) *An FRR automaton M is* correctness preserving *if $w \in L_{\mathrm{C}}(M)$ and $w \vdash_M^{c^*} z$ imply that $z \in L_{\mathrm{C}}(M)$, too.*

While each *deterministic* FRR automaton is obviously correctness preserving, there are nondeterministic FRR automata which are not correctness preserving.

In traditional linguistics the syntactic analysis of central-European languages is often substituted by a procedure, which we call *characteristic analysis.* In fact, in the Czech Republic the characteristic analysis is taught manually in middle-schools.

Definition 6. *Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an FRR automaton that is correctness preserving, and let $w \in \Sigma^*$. Then the set*

$$A_{\mathrm{C}}(w, M) := \{ w_C \in \Gamma^* \mid w_C \in L_{\mathrm{C}}(M) \text{ and } \mathsf{Pr}^{\Sigma}(w_C) = w \}$$

is called the characteristic analysis of w by M. *The size of $A_{\mathrm{C}}(w, M)$ is called the* characteristic ambiguity of w by M.

Note that the assumption of the correctness preserving property in the above definition is quite important. It ensures the so-called 'syntactic completeness' of categories used in the characteristic analysis; in our approach we use auxiliary symbols to model these categories. The notions of correctness preserving property, syntactic completeness, and characteristic analysis are derived from the linguistic method of 'analysis by reduction' as described, e.g., in [5].

3 Analysis by Reduction and Regulated Parallel Communicating Grammar Systems

In [9] the current authors present a transformation that, from a PCGS Π of degree m with constant distribution complexity d and constant generation complexity g, yields a correctness preserving d-rewriting FRR automaton M of a very restricted form such that $L_{\mathrm{P}}(M) = L(\Pi)$. The basic idea of this transformation is as follows. Let w be a word from $L(\Pi)$. For each (reduced) Π-derivation of w, there exists a factorization of w into generative factors of the form $w = g(i_1, j_1)g(i_2, j_2)\cdots g(i_r, j_r) \in L(\Pi)$. From the bounds d and g for the distribution and generation complexity, respectively, it follows that the number r of these factors is bounded from above by the product $g \cdot d$.

The FRR automaton M is given a word of the form

$$w_C := \Delta_{0,k}\Delta_{1,k}\ g(i_1, j_1)\Lambda_{1,k}\Delta_{2,k}\ g(i_2, j_2)\Lambda_{2,k} \ldots \Delta_{r,k}\ g(i_r, j_r)\Lambda_{r,k}\Delta_{r+1,k}$$

as input, where $\Delta_{0,k}, \ldots, \Delta_{r+1,k}$ and $\Lambda_{1,k}, \ldots, \Lambda_{r,k}$ are auxiliary symbols. These symbols describe a particular Π-derivation without cycles (and its communication structure). From Fact 4 we know that the set of Π-derivations of this form is finite.

The FRR automaton M processes this input as follows. In each cycle M first *nondeterministically chooses* an index j of a generative section, and then it consistently removes the *rightmost* generative cycle from each occurrence of each of the factors $g(i, j)$ $(1 \leq i \leq m)$ in w. Observe that there are possibly several occurrences of the factors $g(i, j)$. This is actually an interesting formal example of the mutual independence in the linguistic sense (in the sense of dependency theory) of the segments with different indices j and j'. On the other hand the segments with the same index j are mutually dependent in a sense, which strongly resembles the linguistic notion of valency. Here (the information stored in) the symbols of the form $\Delta_{t,k}$ are used to verify that the simplifications of the various occurrences of factors $g(i, j)$ is consistent with the particular Π-derivation without cycles encoded in the word w_C, while (the information stored in) the symbols of the form $\Lambda_{t,k}$ are used to ensure that all simplifications of occurrences of factors $g(i, j)$ are consistent with each other. In fact, in each rewrite operation M replaces an auxiliary symbol of the form $\Lambda_{t,k}$ by another auxiliary symbol of the form $\Lambda'_{t,k}$, and there is at least one rewrite operation in each cycle that removes a non-empty factor consisting of terminals (input symbols). M repeatedly executes such cycles until a word is obtained that does not contain any generative cycles anymore.

Motivated by the properties of the FRR automaton M above we now define the notion of a *skeleton preserving automaton.* To simplify the definition, we first present some notation.

Let $s, r \in \mathbb{N}_+$, let SP be a set of symbols, and let

$$\phi : SP \to \{1, \ldots, s\} \times \{1, \ldots, r\}$$

be a mapping. Then $SP(i)$ and $SP(i, j)$ will be used to denote the following subsets of SP for all $1 \leq i \leq s$ and $1 \leq j \leq r$:

$$SP(i) = \{\, \chi \mid \exists\, j' : \phi(\chi) = (i, j')\,\} \text{ and } SP(i, j) = \{\, \chi \mid \phi(\chi) = (i, j)\,\}.$$

Further, for an FRR automaton M, let $S_{\mathrm{C}}(M)$ denote the *simple characteristic language* of M, which is the set of words $w \in \Gamma^*$ that M accepts in tail computations.

Definition 7. *Let $r, s \in \mathbb{N}_+$, and let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be a correctness preserving t-rewriting FRR automaton for which the language $S_{\mathrm{C}}(M)$ is finite. M is a* skeleton preserving $[s, r, t]$-automaton *if there exists a sub-alphabet SP of Γ of cardinality $|SP| \leq s \cdot r$ and a mapping $\phi : SP \to \{1, \ldots, s\} \times \{1, \ldots, r\}$ such that all of the following properties are satisfied:*

1. *Each $w \in L_{\mathrm{C}}(M)$ can be written as $w = x_1 \Lambda_1 \Theta_1 x_2 \Lambda_2 \Theta_2 \cdots x_{r'} \Lambda_{r'} \Theta_{r'}$, where $r' \leq r$, $x_1, \ldots, x_{r'} \in \Sigma^*$, $\Theta_1, \ldots, \Theta_{r'} \in SP(i)$ for some $1 \leq i \leq s$, and $\Lambda_1, \ldots, \Lambda_{r'} \in V = \Gamma \smallsetminus (SP \cup \Sigma)$. We call $x_i \Lambda_i \Theta_i$ the* i-th skeletal factor *of w.*
2. *For all $w \in L_{\mathrm{C}}(M)$ and all $\chi \in SP$, $|w|_\chi \leq 1$.*
3. *Each rewriting meta-instruction I of M can be written as a sequence of constraints separated by rewriting rules, that is,*

$$I = (C^{(0)}, W_1, C^{(1)}, W_2, \ldots, W_{t'}, C^{(t')})$$

   for some $t' \leq t$ such that
   - *$W_a = (\Sigma^*, x_a y_a z_a \Lambda_a \Theta_a \to x_a z_a \Lambda'_a \Theta_a)$ for all $1 \leq a \leq t'$, where $x_a y_a z_a \in \Sigma^*$, $\Lambda_a, \Lambda'_a \in V$, and $\Theta_a \in SP$,*
   - *$|y_1 y_2 \ldots y_{t'}| > 0$, and $\Theta_1, \Theta_2, \ldots, \Theta_{t'} \in SP(i, j)$ for some i and j,*
   - *$C^{(j)} = \Sigma^* \cdot \Lambda_{j(1)} \Theta_{j(1)} \cdot \Sigma^* \cdot \Lambda_{j(2)} \Theta_{j(2)} \cdots \Sigma^* \cdot \Lambda_{j(\ell(j))} \Theta_{j(\ell(j))}$ for all j, where $\Theta_{j(1)}, \Theta_{j(2)}, \ldots, \Theta_{j(\ell(j))} \in SP$, and $\Lambda_{j(1)}, \Lambda_{j(2)}, \ldots, \Lambda_{j(\ell(j))} \in V$.*
4. *Whenever $f = x_i \Lambda_i \Theta_i$ is a factor of $w \in L_{\mathrm{C}}(M)$ such that $|f| > k$, then there is an applicable cycle of M containing a rewrite step that deletes some symbols from f.*

The set SP is called the skeletal set *of M, and V is the set of* variables *of M. As elements of SP are neither inserted, nor removed, nor changed during any computation of M, we call them* islands. *Further, $SP(i)$ is the* i-th skeleton*, and $SP(i, j)$ is the* j-th level *of the i-th skeleton.*

The construction of the FRR automaton M in [9] as outlined above can be modified in such a way that we obtain the following result.

Theorem 3. *For each $L \in d$-g-RDDG, there exists a positive integer i such that there is a skeleton preserving $[i, g{\cdot}d, d]$-FRR automaton M such that $L = L_{\mathrm{P}}(M)$. Moreover, the number of auxiliary symbols in $w \in L_{\mathrm{C}}(M)$ is bounded from above by the constant $2 \cdot g \cdot d$.*

Here the number i corresponds to the number of reduced regulated Π-derivations without a generative cycle, and for each value of i, the corresponding i-th skeleton $SP(i)$ is used to describe the factorization of a word $w \in L(\Pi, R)$ into its generative factors according to this particular Π-derivation. The second index j corresponds to an index of a generative section of this Π-derivation, and the elements of the j-th level $SP(i,j)$ of the i-th skeleton are used to mark the occurrences of the generative factors $g(l,j)$ $(1 \leq l \leq m)$ in w.

As in [9] the following result can be established for the characteristic analysis $A_{\mathrm{C}}(w, M)$ of a word w by a skeleton preserving FRR automaton M.

Proposition 1. *Let M be a skeleton preserving $[i, g \cdot d, d]$-FRR automaton. Then, for each $w \in \Sigma^*$, the characteristic ambiguity of w by M is bounded from above by $O(|\Gamma \smallsetminus \Sigma|^{2 \cdot g \cdot d} \cdot n^{g \cdot d})$, and the characteristic analysis $A_{\mathrm{C}}(w, M)$ of w by M can be computed in time $O(|\Gamma \smallsetminus \Sigma|^{2 \cdot g \cdot d} \cdot n^{g \cdot d} \cdot (n + 2 \cdot g \cdot d)^2)$.*

Together with Theorem 3 this proposition has the following consequence.

Corollary 1. *For each language $L \in d$-g-RDDG, the membership problem can be solved in time $O(2^{O(2 \cdot g \cdot d)} \cdot n^{g \cdot d} \cdot (n + 2 \cdot g \cdot d)^2)$.*

Let M be a skeleton preserving FRR automaton. From Properties 1 and 4 of Definition 7 it follows that each cycle $w \vdash^{c}_{M} w_1$ unambiguously determines an index j such that the rewrite steps executed involve the j-th level $SP(i,j)$ of the skeleton $SP(i)$ occurring in w.

Each rewrite step of a skeleton preserving automaton M is a kind of suffix rewrite on a syllable ending with an island. Further, from the assumptions of Definition 7 we see that the language $S_{\mathrm{C}}(M)$ is finite. As suffix rewrites preserve regularity, it follows that all Σ-syllables of words from $L_{\mathrm{C}}(M)$ satisfy some regularity constraints. This implies the following important result.

Corollary 2. *The languages $L_{\mathrm{C}}(M)$ and $L_{\mathrm{P}}(M)$ are semi-linear for each skeleton preserving FRR automaton M.*

To complete our intended characterization we now present a transformation of a skeleton preserving FRR automaton into an RPCGS. To this end we first introduce some technical notions that reflect the structural properties of computations of skeleton preserving FRR automata.

Let $w_C = v_1 \Lambda_1 \Theta_1 v_2 \Lambda_2 \Theta_2 \cdots v_r \Lambda_r \Theta_r \in L_{\mathrm{C}}(M)$, where $v_i \Lambda_i \Theta_i$ are the skeletal factors of w_C $(1 \leq i \leq r)$. Then the sequence $\Theta_1, \Theta_2, \ldots, \Theta_r$ is called the *skeletal structure* of w_C. Obviously, the skeletal structure of w_C is preserved during the whole computation (analysis by reduction) of M on w_C.

The basis for our transformation is the following technical result.

Proposition 2. *Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be a skeleton preserving $[s, 1, 1]$-automaton with a single island θ. Then there is a right-linear grammar $G_\theta = (V_\theta, \Sigma, S_\theta, P_\theta)$ such that $L(G_\theta) = L_{\mathrm{P}}(M)$. Moreover, G_θ can be designed in such a way that it simulates M's computations step-by-step in reverse order.*

Proof. It is easily seen that the language accepted by a skeleton preserving $[s, 1, 1]$-automaton M with a single island θ is regular. So a right-linear grammar G_θ can easily be constructed that simulates M's computations step-by-step in reverse order. □

Based on this proposition we obtain the following main result.

Theorem 4. $L_{\mathrm{P}}(M) \in d$-$(g \cdot d)$-CRDDG *for each skeleton preserving $[s, g \cdot d, d]$-FRR automaton M.*

Below we give an outline of the proof, which is followed by a detailed example illustrating it.

Proof outline. Let $SP = \{\theta_1, \ldots, \theta_m\}$ be the skeletal set of M, and let V be the set of variables of M (see Definition 7). For a skeletal structure σ and an island θ_j occurring in σ, $ord_\sigma(\theta_j)$ denotes the order of θ_j in σ, and $elem_\sigma(i) = j$, if $ord_\sigma(\theta_j) = i$. The construction of the RPCGS (Π, R) for the language $L_{\mathrm{P}}(M)$ is given in six steps, where $\Pi = (N, K, \Sigma, G_M, G_1 \ldots, G_m)$.

1. For every $\theta \in SP$, a skeleton preserving $[s, 1, 1]$-automaton M_θ is obtained from M as follows:

(a) For all $\theta' \in SP$, and all $\alpha \in \Sigma^* \cdot V$, if $\theta'\alpha\theta$ is a factor of a word from $S_{\mathrm{C}}(M)$, or if $\alpha\theta$ is a prefix of a word from $S_{\mathrm{C}}(M)$, then $\alpha\theta$ is accepted by a tail computation of M_θ.

(b) Each meta-instruction $I = (C^{(0)}, W_1, C^{(1)}, \ldots, W_i, \ldots, C^{(t')})$ of M, where the rewriting rule W_i contains the island θ, is converted into a meta-instruction of the form $I_\theta = W_i$.

2. Let H_{θ_i} denote the regular grammar for $L_{\mathrm{P}}(M_{\theta_i})$ that is obtained from M_{θ_i} according to Proposition 2. To define $G_1, \ldots, G_m$ we slightly modify $H_{\theta_1}, \ldots, H_{\theta_m}$, letting them work in parallel and checking consistency with the help of a regular control language R. The nonterminals of G_i, $1 \leq i \leq m$, are modelled as pairs, the first component of which is used to represent a nonterminal of the grammar H_{θ_i}. To enforce consistency of the various components working in parallel, each grammar G_i guesses a skeletal structure σ in its first step such that θ_i occurs in σ, and remembers σ in the second components of its nonterminals. Accordingly, G_i has the start rules

$$S_{\theta_i} \to \begin{cases} (S_{\theta_i}, \sigma), \text{ if } \theta_i \text{ occurs in } \sigma, \\ (*, \sigma), \quad \text{otherwise.} \end{cases}$$

3. Productions of the form $A \to A$, where A denotes a nonterminal symbol, are added to the grammar G_i to give it the option to idle. This is necessary for allowing the simulation of cycles of M in which θ_i is not involved. In particular, $(*, \sigma) \to (*, \sigma)$ is the only production with left-hand side $(*, \sigma)$.

4. In order to enable the master grammar G_M to compose the terminal word to be derived, we add to G_i rules of the form $A \to T_{ord_\sigma(\theta')}$, where θ' is the right-hand neighbor of θ_i in σ, and $A \to x$ is a rule in H_{θ_i} such that x is a terminal string. The nonterminal T_j can be seen as a message to the master to ask G_j for a communication. With the exception of G_M, $T_j \to T_j$ is the only production applicable to T_j.

5. The derivation of the master grammar G_M can be given by the following expression, where σ runs over all possible skeletal structures:

$$S_M \to (S_M, \sigma), \{(S_M, \sigma) \to (S_M, \sigma)\}^*, (S_M, \sigma) \to Q_{elem_\sigma(1)},\\ T_{elem_\sigma(1)} \to Q_{elem_\sigma(2)}, \ldots, T_{elem_\sigma(|\sigma|-1)} \to Q_{elem_\sigma(|\sigma|)}.$$

6. Finally, the regular language R is constructed to verify that all grammars have guessed the same skeletal structure σ, and that the productions applied in (non-idling) component grammars in each derivation step correspond to a rewriting meta-instruction of M applicable in a computation over a word with skeletal structure σ. In addition, all other component grammars have to be idling. □

The following simple example is included in order to illustrate the proof outline above.

Example 2. Let $\Sigma = \{a, b, c, d\}$, let $h : \{a, c\}^* \to \{b, d\}^*$ be the morphism given by $h(a) = b$ and $h(c) = d$, and let L_2 be the following language:

$$L_2 = \{\, wh(w)w_1h(w_1)h(w) \mid w, w_1 \in \{a, c\}^* \,\}.$$

First we present a skeleton preserving $[1, 5, 3]$-FRR automaton M_2 such that $L_2 = L_{\mathrm{P}}(M_2)$. The input alphabet of M_2 is Σ, and M_2 uses the skeletal set $SP_2 = \{[1, 1, 1], [1, 1, 2], [1, 2, 3], [1, 2, 4], [1, 1, 5]\}$, and exactly one variable A. The structural mapping ϕ_2 of this skeletal set is given by $\phi_2([1, i, j]) = [1, i]$ for all $[1, i, j] \in SP_2$. We see that SP_2 consists of a single skeleton σ with two levels.

The behaviour of M_2 is given by the following five meta-instructions :

$$\begin{aligned}
I_{a,1} &= (\Sigma^*, aA[1,1,1] \to A[1,1,1], \Sigma^*, bA[1,1,2] \to A[1,1,2],\\
&\qquad \Sigma^* \cdot A[1,2,3] \cdot \Sigma^* \cdot A[1,2,4], \Sigma^*, bA[1,1,5] \to A[1,1,5]);\\
I_{c,1} &= (\Sigma^*, cA[1,1,1] \to [1,1,1], \Sigma^*, dA[1,1,2] \to A[1,1,2],\\
&\qquad \Sigma^* \cdot A[1,2,3] \cdot \Sigma^* \cdot A[1,2,4], \Sigma^*, dA[1,1,5] \to A[1,1,5]);\\
I_{a,2} &= (\Sigma^* \cdot A[1,1,1] \cdot \Sigma^* \cdot A[1,1,2], \Sigma^*, aA[1,2,3] \to A[1,2,3],\\
&\qquad \Sigma^*, bA[1,2,4] \to A[1,2,4], \Sigma^* \cdot A[1,1,5]);\\
I_{c,2} &= (\Sigma^* \cdot A[1,1,1] \cdot \Sigma^* \cdot A[1,1,2], \Sigma^*, cA[1,2,3] \to A[1,2,3],\\
&\qquad \Sigma^*, dA[1,2,4] \to A[1,2,4], \Sigma^* \cdot A[1,1,5]);\\
I_{acc} &= (A[1,1,1]A[1,1,2]A[1,2,3]A[1,2,4]A[1,1,5], \mathsf{Accept}).
\end{aligned}$$

It is not hard to see that

$$L_{\mathrm{C}}(M_2) = \{\, wA[1,1,1]h(w)A[1,1,2]w_1A[1,2,3]h(w_1)A[1,2,4]h(w)A[1,1,5] \mid\\ w, w_1 \in \{a, c\}^* \,\},$$

and that the above meta-instructions are correctness preserving. In particular, it follows that M_2 is indeed a skeleton preserving $[1,5,3]$-FRR automaton satisfying $L_{\mathrm{P}}(M_2) = L_2$.

Next we construct a regulated PCGS (Π_2, R_2) step by step which generates the language L_2, simulating M_2. The construction illustrates the underlying ideas of the proof of Theorem 4.

First we describe the master G_{M_2} of Π_2. It has the following productions, in which the symbols $[1,1,1]$, $[1,1,2]$, $[1,2,3]$, $[1,2,4]$, $[1,1,5]$ are used as communication symbols:

$$\begin{array}{llll}
S_M \to (S_M,\sigma),\ (S_M,\sigma) \to (S_M,\sigma),\ (S_M,\sigma) \to [1,1,1], \\
T_1 \to [1,1,2], & T_2 \to [1,2,3], & T_3 \to [1,2,4], & T_4 \to [1,1,5].
\end{array}$$

Observe that G_{M_2} does not use any terminals at all.

Next we describe the remaining component grammars of Π_2. These are the grammars $G_{[1,1,1]}$, $G_{[1,1,2]}$, $G_{[1,1,5]}$, $G_{[1,2,3]}$, and $G_{[1,2,4]}$. Observe that the first three of these grammars must idle after returning to their start symbol, while the latter two of these grammars need not idle after generating terminal symbols. Accordingly, these grammars are given be the following sets of productions:

$$\begin{array}{lll}
G_{[1,1,1]}: & S_{[1,1,1]} \to (S_{[1,1,1]},\sigma), & (S_{[1,1,1]},\sigma) \to (S_{[1,1,1]},\sigma), \\
& (S_{[1,1,1]},\sigma) \to (A_{[1,1,1]},\sigma), & (A_{[1,1,1]},\sigma) \to a(A_{[1,1,1]},\sigma), \\
& (A_{[1,1,1]},\sigma) \to c(A_{[1,1,1]},\sigma), & (A_{[1,1,1]},\sigma) \to T_1, \\
& T_1 \to T_1; &
\end{array}$$

$$\begin{array}{lll}
G_{[1,1,2]}: & S_{[1,1,2]} \to (S_{[1,1,2]},\sigma), & (S_{[1,1,2]},\sigma) \to (S_{[1,1,2]},\sigma), \\
& (S_{[1,1,2]},\sigma) \to (A_{[1,1,2]},\sigma), & (A_{[1,1,2]},\sigma) \to b(A_{[1,1,2]},\sigma), \\
& (A_{[1,1,2]},\sigma) \to d(A_{[1,1,2]},\sigma), & (A_{[1,1,2]},\sigma) \to T_2, \\
& T_2 \to T_2; &
\end{array}$$

$$\begin{array}{lll}
G_{[1,1,5]}: & S_{[1,1,5]} \to (S_{[1,1,5]},\sigma), & (S_{[1,1,5]},\sigma) \to (S_{[1,1,5]},\sigma), \\
& (S_{[1,1,5]},\sigma) \to (A_{[1,1,5]},\sigma), & (A_{[1,1,5]},\sigma) \to b(A_{[1,1,5]},\sigma), \\
& (A_{[1,1,5]},\sigma) \to d(A_{[1,1,5]},\sigma), & (A_{[1,1,5]},\sigma) \to T_5, \\
& T_5 \to T_5, & T_5 \to \varepsilon;
\end{array}$$

$$\begin{array}{lll}
G_{[1,2,3]}: & S_{[1,2,3]} \to (S_{[1,2,3]},\sigma), & (S_{[1,2,3]},\sigma) \to (S_{[1,2,3]},\sigma), \\
& (S_{[1,2,3]},\sigma) \to (A_{[1,2,3]},\sigma), & (A_{[1,2,3]},\sigma) \to a(A_{[1,2,3]},\sigma), \\
& (A_{[1,2,3]},\sigma) \to c(A_{[1,2,3]},\sigma), & (A_{[1,2,3]},\sigma) \to T_3, \\
& T_3 \to T_3; &
\end{array}$$

$$\begin{array}{lll}
G_{[1,2,4]}: & S_{[1,2,4]} \to (S_{[1,2,4]},\sigma), & (S_{[1,2,4]},\sigma) \to (S_{[1,2,4]},\sigma), \\
& (S_{[1,2,4]},\sigma) \to (A_{[1,2,4]},\sigma), & (A_{[1,2,4]},\sigma) \to b(A_{[1,2,4]},\sigma), \\
& (A_{[1,2,4]},\sigma) \to d(A_{[1,2,4]},\sigma), & (A_{[1,2,4]},\sigma) \to T_4, \\
& T_4 \to T_4. &
\end{array}$$

It remains to describe the regular control language R_2 for the set of admissible extended traces of Π_2. The task of R_2 is to force Π_2 to first generate the factors of the first level, then the factors of the second level, and finally to compose the final word generated by the master in the correct way. The language R_2 is given through the following regular expression:

$$R_2 =$$

$$\begin{pmatrix} S_M \\ S_{[1,1,1]} \\ S_{[1,1,2]} \\ S_{[1,1,5]} \\ S_{[1,2,3]} \\ S_{[1,2,4]} \end{pmatrix} \begin{pmatrix} (S_M,\sigma) \\ (S_{[1,1,1]},\sigma) \\ (S_{[1,1,2]},\sigma) \\ (S_{[1,1,5]},\sigma) \\ (S_{[1,2,3]},\sigma) \\ (S_{[1,2,4]},\sigma) \end{pmatrix} \begin{pmatrix} (S_M,\sigma) \\ (A_{[1,1,1]},\sigma) \\ (A_{[1,1,2]},\sigma) \\ (A_{[1,1,5]},\sigma) \\ (S_{[1,2,3]},\sigma) \\ (S_{[1,2,4]},\sigma) \end{pmatrix} \left\{ \begin{pmatrix} (S_M,\sigma) \\ a(A_{[1,1,1]},\sigma) \\ b(A_{[1,1,2]},\sigma) \\ b(A_{[1,1,5]},\sigma) \\ (S_{[1,2,3]},\sigma) \\ (S_{[1,2,4]},\sigma) \end{pmatrix}, \begin{pmatrix} (S_M,\sigma) \\ c(A_{[1,1,1]},\sigma) \\ d(A_{[1,1,2]},\sigma) \\ d(A_{[1,1,5]},\sigma) \\ (S_{[1,2,3]},\sigma) \\ (S_{[1,2,4]},\sigma) \end{pmatrix} \right\}^* \cdot$$

$$\begin{pmatrix} (S_M,\sigma) \\ T_1 \\ T_2 \\ T_5 \\ (S_{[1,2,3]},\sigma) \\ (S_{[1,2,4]},\sigma) \end{pmatrix} \begin{pmatrix} (S_M,\sigma) \\ T_1 \\ T_2 \\ \varepsilon \\ (A_{[1,2,3]},\sigma) \\ (A_{[1,2,4]},\sigma) \end{pmatrix} \left\{ \begin{pmatrix} (S_M,\sigma) \\ T_1 \\ T_2 \\ \varepsilon \\ a(A_{[1,2,3]},\sigma) \\ b(A_{[1,2,4]},\sigma) \end{pmatrix}, \begin{pmatrix} (S_M,\sigma) \\ T_1 \\ T_2 \\ \varepsilon \\ c(A_{[1,2,3]},\sigma) \\ d(A_{[1,2,4]},\sigma) \end{pmatrix} \right\}^* \begin{pmatrix} (S_M,\sigma) \\ T_1 \\ T_2 \\ \varepsilon \\ T_3 \\ T_4 \end{pmatrix} \cdot$$

$$\begin{pmatrix} [1,1,1] \\ T_1 \\ T_2 \\ \varepsilon \\ T_3 \\ T_4 \end{pmatrix} \begin{pmatrix} T_1 \\ S_{[1,1,1]} \\ T_2 \\ \varepsilon \\ T_3 \\ T_4 \end{pmatrix} \begin{pmatrix} [1,1,2] \\ (S_{[1,1,1]},\sigma) \\ T_2 \\ \varepsilon \\ T_3 \\ T_4 \end{pmatrix} \begin{pmatrix} T_2 \\ (S_{[1,1,1]},\sigma) \\ S_{[1,1,2]} \\ \varepsilon \\ T_3 \\ T_4 \end{pmatrix} \begin{pmatrix} [1,2,3] \\ (S_{[1,1,1]},\sigma) \\ (S_{[1,1,2]},\sigma) \\ \varepsilon \\ T_3 \\ T_4 \end{pmatrix} \begin{pmatrix} T_3 \\ (S_{[1,1,1]},\sigma) \\ (S_{[1,1,2]},\sigma) \\ \varepsilon \\ S_{[1,2,3]} \\ T_4 \end{pmatrix} \cdot$$

$$\begin{pmatrix} [1,2,4] \\ (S_{[1,1,1]},\sigma) \\ (S_{[1,1,2]},\sigma) \\ \varepsilon \\ (S_{[1,2,3]},\sigma) \\ T_4 \end{pmatrix} \begin{pmatrix} T_4 \\ (S_{[1,1,1]},\sigma) \\ (S_{[1,1,2]},\sigma) \\ \varepsilon \\ (S_{[1,2,3]},\sigma) \\ S_{[1,2,4]} \end{pmatrix} \begin{pmatrix} [1,2,5] \\ (S_{[1,1,1]},\sigma) \\ (S_{[1,1,2]},\sigma) \\ \varepsilon \\ (S_{[1,2,3]},\sigma) \\ (S_{[1,2,4]},\sigma) \end{pmatrix} \begin{pmatrix} \varepsilon \\ (S_{[1,1,1]},\sigma) \\ (S_{[1,1,2]},\sigma) \\ S_{[1,1,5]} \\ (S_{[1,2,3]},\sigma) \\ (S_{[1,2,4]},\sigma) \end{pmatrix}$$

It is now easily verified that the regulated PCGS (Π_2, R_2) generates the language $L_{\mathrm{P}}(M_2) = L_2$.

From Theorems 3 and 4 we obtain the following consequence.

Corollary 3. *For each $L \in \mathsf{RCOM}(\mathsf{O}(1))$, there is a centralized RPCGS (Π, R) with constant communication complexity such that $L = L(\Pi, R)$.*

4 Conclusion

Here we have introduced regulated PCGS, and derived the following interesting results on regulated PCGSs with constant communication complexity:

- Regulated PCGSs are more expressive than non-regulated PCGSs.
- The centralized variant is as expressive as the non-centralized variant.
- A language L can be generated by an RPCGS if and only if it is the proper language of a skeleton preserving FRR automaton. Accordingly, L is semi-linear, the characteristic analysis for each of its elements is of polynomial size, and the membership problem for L can be solved in polynomial time.

However, it remains to compare the regulated PCGSs introduced here to the PC grammar systems with regular components and with rule synchronization of Păun [11]. Further, it remains to derive closure and non-closure properties for the class of languages that can be generated by regulated PCGSs with constant communication complexity. Also it remains to study regulated PCGSs with a higher degree of communication complexity.

Acknowledgement. We are grateful to an anonymous referee for pointing us to Păun's article [11].

References

1. Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, G. (eds.): Grammar Systems: A Grammatical Approach to Distribution and Cooperation. Gordon and Breach Science Publishers (1994) ISBN 2-88124-957-4
2. Dassow, J., Păun, G.: Regulated Rewriting in Formal Language Theory. Springer, Heidelberg (1989)
3. Hromkovič, J., Kari, J., Kari, L., Pardubská, D.: Two lower bounds on distributive generation of languages. In: Privara, I., Ružička, P., Rovan, B. (eds.) MFCS 1994. LNCS, vol. 841, pp. 423–432. Springer, Heidelberg (1994)
4. Kuboň, V., Lopatková, M., Plátek, M., Pognan, P.: A linguistically-based segmentation of complex sentences. In: Wilson, D., Sutcliffe, G. (eds.) 20$^{\text{th}}$ FLAIRS Conference, Proc., pp. 368–373. AAAI Press, Menlo Park (2007)
5. Lopatková, M., Plátek, M., Sgall, P.: Towards a formal model for functional generative description, analysis by reduction and restarting automata. The Prague Bulletin of Mathematical Linguistics 87, 7–26 (2007)
6. Otto, F.: Restarting automata. In: Ésik, Z., Martín-Vide, C., Mitrana, V. (eds.) Recent Advances in Formal Languages and Applications. Studies in Computational Intelligence, vol. 25, pp. 269–303. Springer, Heidelberg (2006)
7. Otto, F., Plátek, M.: A two-dimensional taxonomy of proper languages of lexicalized FRR-automata. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 409–420. Springer, Heidelberg (2008)
8. Pardubská, D., Plátek, M.: Parallel communicating grammar systems and analysis by reduction by restarting automata. In: Bel-Enguix, G., Jimenez-Lopez, M.D. (eds.) International Workshop on Non-Classical Formal Languages in Linguistics, ForLing 2008, Proc., Tarragona, pp. 81–98 (2008)
9. Pardubská, D., Plátek, M., Otto, F.: On parallel communicating grammar systems and correctness preserving restarting automata. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 660–671. Springer, Heidelberg (2009)

10. Pardubská, D., Plátek, M., Otto, F.: On PCGS and FRR automata. In: Vojtáš, P. (ed.) Information Technologies – Applications and Theory, ITAT 2008, Proc., Department of Computer Science, Faculty of Science, Pavol Jozef Šafárik University, Košice,, pp. 41–47 (2008)
11. Păun, G.: On the synchronization in parallel communicating grammar systems. Acta Informatica 30, 351–367 (1993)
12. Păun, G., Santean, L.: Parallel communicating grammar systems: the regular case. Ann. Univ. Buc. Ser. Mat.-Inform. 37, 55–63 (1989)

Author Index