Michal Kunc | Alexander Okhotin (Eds.)

Theory and Applications of Language Equations

Proceedings of the 1st International Workshop, Turku, Finland, 2 July 2007

Turku Centre *for* Computer Science

TUCS General Publication
No 44, June 2007

Theory and Applications of Language Equations

Proceedings of the 1st International Workshop,
Turku, Finland, 2 July 2007

Editors:

Michal Kunc
Alexander Okhotin

TUCS General Publication
No 44, June 2007

Editors

Michal Kunc
Department of Mathematics
Masaryk University
Janáčkovo nám. 2a
602 00 Brno
Czech Republic

Alexander Okhotin
Department of Mathematics
University of Turku
Yliopistonmäki
20014 Turku
Finland

ISBN 978–952–12–1920–7
ISSN 1239–1905
Painosalama Oy
Turku, Finland
2007

Preface

The first workshop on Theory and Applications of Language Equations (TALE 2007) was held in Turku, Finland on 2 July 2007 as a satellite event to the annually held conference on Developments in Language Theory (DLT 2007).

The main topic of the workshop are equations with formal languages as unknowns. Such equations are among the most natural objects of formal language theory and have been studied since its inception in early 1960s. The recent renewal of interest in this fundamental subject prompted us to organize a workshop aimed to bring together researchers working on different aspects of different variants of language equations and related formalisms. The scope of the workshop covers such subjects as decision problems for language equations, families of languages defined by language equations, language equations with various operations, different forms of language equations, identities on formal languages, representation of applied problems by language equations, conjunctive grammars, Boolean grammars, descriptional complexity of language equations and set constraints.

The workshop programme consisted of 3 invited talks given by experts in areas closely related to language equations, and of 6 contributed talks selected by the programme committee, with each submission being reviewed by at least three programme committee members.

We are grateful to all authors for their contributions to this event and to the programme committee for selecting the workshop programme. The reviewing process was conducted using EasyChair and this volume was typeset using TEX, and we are grateful to Andrei Voronkov and Donald Knuth for their respective systems.

We are indebted to the co-chairmen of DLT 2007, Tero Harju and Juhani Karhumäki, for their support of our workshop. Let us thank all organizers of DLT 2007, Vesa Halava in particular, for their help with local arrangements, and Elisa Mikkola for secretarial assistance. We wish to thank our publisher, Turku Centre for Computer Science, and the printing office, Painosalama Oy, for an efficient production of this volume.

Finally, we are grateful to the Academy of Finland for generous financial support to this workshop granted within project 118540, "Language equations".

June 2007

Michal Kunc
Alexander Okhotin

Workshop Organization

Programme Committee

Juhani Karhumäki (Turku, Finland)
Michal Kunc (Brno, Czech Republic, *acting chairman*)
Alexander Okhotin (Turku, Finland, *ex-chairman*)
Kai Salomaa (Kingston, Ontario, Canada)
Sophie Tison (Lille, France)

Local Organization

Alexander Okhotin

External Reviewers

Françoise Gire
Artur Jeż
Michel Latteux
Christos Nomikos
Yves Roos

Workshop Programme

Monday, 2 July 2007

9:50	*Opening*
10:00	Invited speaker: **W. Kuich** (joint work with Z. Ésik) "Fixed points in semiring theory" (p. 5)
11:00	*Coffee break*
11:30	Invited speaker: **W. Charatonik** "Set constraints and language equations" (p. 1)
12:30	M. Daley, M. Domaratzki, K. Salomaa "On the operational orthogonality of languages" (p. 43)
13:00	*Lunch*
14:30	Invited speaker: **N. Yevtushenko** (joint work with T. Villa and S. Zharikova) "Solving language equations over synchronous and parallel composition operators" (p. 14)
15:30	J. Cassaigne, J. Karhumäki, P. Salmela "Conjugacy of finite biprefix codes" (p. 33)
16:00	O. Ly "A constructive solution of the language inequation $XA \subseteq BX$" (p. 76)
16:30	*Coffee break*
17:00	V. Kountouriotis, C. Nomikos, P. Rondogiannis "Conjunctive macro grammars" (p. 67)
17:30	D. Zook "Multi-conjunctive grammars" (p. 85)
18:00	A. Jeż, A. Okhotin "Language equations with positional addition" (p. 54)
18:30	*Closing*

All talks were held at the ICT building of the University of Turku (Joukahaisenkatu 3–5 B), in lecture hall Beta.

Table of Contents

Set constraints and language equations

Witold Charatonik

Institute of Computer Science, University of Wrocław, Poland

Abstract. Set constraints are relations between sets of ground terms over a given alphabet. Syntactically, they are conjunctions of inclusions between expressions built over variables, constructors (constants and function symbols from a given alphabet) and a choice of set operators that defines a specific class of set constraints. They give a natural formalism for many problems in program analysis, type inference, order-sorted unification, constraint logic programming.
In this talk we briefly present the history of set constraints, the methods of solving them and the complexity of their satisfiability problem. We also give some examples of applications, in particular in solving restricted classes of language equations over tree and word languages.

1 Introduction

Set constraints denote relations between sets of trees. Syntactically, they are conjunctions of inclusions between expressions built over variables, constructors (constants and function symbols from a given alphabet) and a choice of set operators that defines a specific class of set constraints. The main application domain is set-based program analysis and type inference for functional, imperative and logic programming languages, but they are also used in order-sorted languages and in constraint logic programming.

Set constraints were studied from the logical and topological point of view and also in domains different from the Herbrand universe. See [1, 8, 9] for overviews on set constraints.

1.1 Syntax and semantics

Syntactically, a *system of set constraints* is a finite conjunction of inclusions $E \subseteq E'$ (or, in the case of negative set constraints, also $E \not\subseteq E'$), where E and E' are *set expressions* over a given finite signature Σ of function symbols, and an infinite collection $\mathcal{V}$ of set-valued variables. Set expressions are generated by the grammar

$$E ::= \bot \mid \top \mid \alpha \mid E \cup E \mid E \cap E \mid \bar{E} \mid f(E_1, ..., E_n) \mid f^{-i}(E)$$

where α is a set variable in $\mathcal{V}$ and $f \in \Sigma$. In different papers this grammar is often restricted or extended by some specific set operators to obtain different classes of constraints.

M. Kunc, A. Okhotin (Eds.): Theory and Applications of Language Equations
TUCS General Publication No 44, pp. 1–4, 2007.

Semantically, the variables range over subsets of the Herbrand universe T_Σ over Σ, i.e., over sets of constant (ground) terms. A system of set constraints is said to be *consistent* if there exists a mapping (called the *solution* of the system) assigning sets of ground terms over Σ to the variables in such a way that the conjunction of inclusions evaluates to true. More formally, if $\sigma : \mathcal{V} \to \mathcal{P}(T_\Sigma)$ assigns to variables in $\mathcal{V}$ sets of ground terms in T_Σ then we define

$$\begin{aligned}
[\![\bot]\!]_\sigma &= \varnothing \\
[\![\top]\!]_\sigma &= T_\Sigma \\
[\![\alpha]\!]_\sigma &= \sigma(\alpha) \\
[\![E_1 \cup E_2]\!]_\sigma &= [\![E_1]\!]_\sigma \cup [\![E_2]\!]_\sigma \\
[\![E_1 \cap E_2]\!]_\sigma &= [\![E_1]\!]_\sigma \cap [\![E_2]\!]_\sigma \\
[\![\bar{E}]\!]_\sigma &= T_\Sigma \setminus [\![E]\!]_\sigma \\
[\![f(E_1, ..., E_n)]\!]_\sigma &= \{f(t_1, \ldots, t_n) \mid t_1 \in [\![E_1]\!]_\sigma, \ldots t_n \in [\![E_n]\!]_\sigma\} \\
[\![f^{-i}(E)]\!]_\sigma &= \{t_i \mid \exists t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n : f(t_1, \ldots, t_n) \in [\![E]\!]_\sigma\}
\end{aligned}$$

and say that σ is a solution of $\bigwedge_i E_i \subseteq E'_i$ if $[\![E_i]\!]_\sigma$ is a subset of $[\![E'_i]\!]_\sigma$ for all i.

2 Set constraints as language equations

Here we formulate some results from the theory of set constraints in terms of language equations, both for tree and word languages.

2.1 Tree languages

A tree language over a signature Σ is any subset of the Herbrand universe T_Σ. It is regular if it can be recognized by a finite tree automaton.

Theorem 1 ([3, 4]). *The satisfiability problem for systems of equations of the form $\bigwedge_{i=1}^n E_i = E'_i$, where E_i and E'_i are expressions generated by the grammar in Section 1.1, is NEXPTIME-complete.*

Now we consider equations in a form similar to a tree grammar. Formally, we call a system of equations over tree languages *definite* if it is of the form $\bigwedge_{i=1}^n \alpha_i = E_i$ where α_i are distinct variables and E_i are expressions, generated by the grammar in Section 1.1, that do not contain the complement symbol. The emptiness problem for a system $\mathcal{E}$ of definite equations and a variable α is a problem whether for every solution σ of $\mathcal{E}$ the set $\sigma(\alpha)$ is empty. A solution σ is regular if for every variable α the set $\sigma(\alpha)$ is a regular tree language.

Theorem 2 ([7, 5, 6]). *Every system of definite equations is satisfiable and it has both the least and the greatest solution. Both these solutions are regular and can be effectively computed in DEXPTIME. The emptiness problem for systems of definite equations is DEXPTIME-complete.*

This theorem remains true for languages of infinite trees, see [6] for details.

2.2 Word languages

Let A be a finite alphabet. Consider expressions generated by the grammar

$$E ::= \bot \mid \top \mid a \mid \alpha \mid E \cup E \mid E \cap E \mid \bar{E} \mid aE \mid a^{-1}(E)$$

where α is a variable in $\mathcal{V}$ and $a \in A$, with the semantics given by

$$\begin{aligned}
[\![\bot]\!]_\sigma &= \varnothing \\
[\![\top]\!]_\sigma &= A^* \\
[\![\alpha]\!]_\sigma &= \sigma(\alpha) \\
[\![E_1 \cup E_2]\!]_\sigma &= [\![E_1]\!]_\sigma \cup [\![E_2]\!]_\sigma \\
[\![E_1 \cap E_2]\!]_\sigma &= [\![E_1]\!]_\sigma \cap [\![E_2]\!]_\sigma \\
[\![\bar{E}]\!]_\sigma &= A^* \setminus [\![E]\!]_\sigma \\
[\![aE]\!]_\sigma &= \{aw \mid w \in [\![E]\!]_\sigma\} \\
[\![a^{-1}(E)]\!]_\sigma &= \{w \in A^* \mid aw \in [\![E]\!]_\sigma\}
\end{aligned}$$

where $\sigma : \mathcal{V} \to \mathcal{P}(A^*)$ is any assignment of languages over A to variables in $\mathcal{V}$.

Theorem 3. *The satisfiability problem for systems of equations of the form $\bigwedge_{i=1}^{n} E_i = E'_i$, where E_i and E'_i are expressions generated by the grammar above, is DEXPTIME-complete.*

The proof of this theorem is an easy reduction to and from unary set constraints [2], combined with elimination of projection symbols that is possible in the unary case.

References

1. A. Aiken. Set constraints: Results, applications and future directions. In *Proceedings of the Workshop on Principles and Practice of Constraint Programming*, volume 874 of *LNCS*, pages 326–335, 1994.
2. A. Aiken, D. Kozen, M. Vardi, and E. L. Wimmers. The complexity of set constraints. In *Computer Science Logic'93*, volume 832 of *LNCS*, pages 1–17, 1993.
3. L. Bachmair, H. Ganzinger, and U. Waldmann. Set constraints are the monadic class. In *Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 75–83, 1993.
4. W. Charatonik and L. Pacholski. Set constraints with projections are in NEXPTIME. In *Proceedings of the* 35^{th} *Symposium on Foundations of Computer Science*, pages 642–653, 1994.
5. W. Charatonik and A. Podelski. Set constraints with intersection. In *Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 362–372, 1997.
6. W. Charatonik and A. Podelski. Co-definite set constraints. In *9th International Conference on Rewriting Techniques and Applications*, volume 1379 of *LNCS*, pages 211–225, 1998.
7. N. Heintze and J. Jaffar. A decision procedure for a class of set constraints. In *Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 42–51, 1990.
8. N. Heintze and J. Jaffar. Set constraints and set-based analysis. In *Proceedings of the Workshop on Principles and Practice of Constraint Programming*, volume 874 of *LNCS*, pages 281–298, 1994.

9. L. Pacholski and A. Podelski. Set constraints - a pearl in research on constraints. In *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming - CP97*, volume 1330 of *LNCS*, 1997.

Fixed points in semiring theory*

Zoltán Ésik[1,2] and Werner Kuich[3]

[1] University of Szeged, Hungary
ze@inf.u-szeged.hu
[2] Rovira i Virgili University, Tarragona, Spain
[3] Technische Universität Wien, Austria
kuich@tuwien.ac.at

Abstract. We consider fixed point theory in continuous semirings and discuss the Bekić-De Bakker-Scott rule, the diagonal equation and the parameter identity.
We show applications to formal power series theory: deleting linear terms in an algebraic system; and a Kleene Theorem for algebraic power series.

1 Introduction and preliminaries

Fixed point theory plays a prominent rôle in Theoretical Computer Science. Due to a well-known fixed point theorem, complete partially ordered sets and continuous functions have been used widely to give semantics to recursive definitions (see Bloom, Ésik [2], Guessarian [9]) and to achieve least solutions to algebraic systems (see Ginsburg, Rice [7], Kuich [12]). This paper gives an introduction to the latter application of fixed point theory.

The paper consists of this and three more sections. In Section 2, we quote the fixed point theorem and three very important facts about fixed points of continuous functions: the Bekić-De Bakker-Scott rule, the diagonal equation and the parameter identity.

In Section 3, we give the basics of algebraic systems and show how the fixed point theorem yields least solutions to algebraic systems. In the last section, we apply fixed point theory to algebraic systems (and hence, to context-free grammars): deleting linear terms in algebraic systems; and a Kleene Theorem for algebraic power series.

We now give some well-known facts about continuous semirings and complete partially ordered sets.

A commutative monoid $\langle A, +, 0\rangle$ is called *ordered* iff it is equipped with a partial order $\leq$ preserved by the $+$ operation such that $0 \leq a$ holds for all $a \in A$. It then follows that $a \leq a + b$, for all $a, b \in A$. In particular, a commutative monoid $\langle A, +, 0\rangle$ is called *naturally ordered* iff the relation $\sqsubseteq$ defined by: $a \sqsubseteq b$ iff there exists a c such that $a + c = b$, is a partial order.

* Partially supported by Aktion Österreich-Ungarn, Wissenschafts- und Erziehungskooperation, Projekt 68öu2.

M. Kunc, A. Okhotin (Eds.): Theory and Applications of Language Equations
TUCS General Publication No 44, pp. 5–13, 2007.

Recall that a non-empty subset D of a partially ordered set P is called *directed* iff each pair of elements of D has an upper bound in D. Moreover, a function $f : P \to Q$ between partially orderet sets is *continuous* iff it preserves the least upper bound of any directed set, i.e., when $f(\sup D) = \sup f(D)$, for all directed sets $D \subseteq P$ such that $\sup D$ exists. It follows that any continuous function preserves the order.

An ordered commutative monoid $\langle A, +, 0\rangle$ is called a *continuous monoid* iff each directed subset of A has a least upper bound and the $+$ operation preserves the least upper bound of directed sets, i.e., when

$$a + \sup D = \sup(a + D)\,,$$

for all directed sets $D \subseteq A$ and for all $a \in A$. Here, $a+D$ is the set $\{a+x \mid x \in D\}$.

It is known that an ordered commutative monoid A is continuous iff each chain in A has a least upper bound and the $+$ operation preserves least upper bounds of non-empty chains, i. e., when $a + \sup C = \sup(a + C)$ holds for all non-empty chains C in A. (See Markowsky [14].)

Proposition 1.1 *Any continuous monoid $\langle A, +, 0\rangle$ is a complete monoid equipped with the following sum operation:*

$$\sum_{i\in I} a_i = \sup\{\sum_{i\in E} a_i \mid E \subseteq I,\ E \text{ finite}\}\,,$$

for all index sets I and all families $(a_i \mid i \in I)$ in A.

A semiring $\langle A, +, \cdot, 0, 1\rangle$ is called *ordered* if $\langle A, +, 0\rangle$ is an ordered monoid and multiplication preserves the order. When the order on A is the natural order, $\langle A, +, \cdot, 0, 1\rangle$ is automatically an ordered semiring.

A semiring $\langle A, +, \cdot, 0, 1\rangle$ is called *continuous* if $\langle A, +, 0\rangle$ is a continuous monoid and if multiplication is continuous, i.e.,

$$a \cdot (\sup_{i\in I} a_i) = \sup_{i\in I}(a \cdot a_i) \quad \text{and} \quad (\sup_{i\in I} a_i) \cdot a = \sup_{i\in I}(a_i \cdot a)$$

for all directed sets $\{a_i \mid i \in I\}$. It follows that the distribution laws hold for infinite sums:

$$a \cdot (\sum_{i\in I} a_i) = \sum_{i\in I}(a \cdot a_i) \quad \text{and} \quad (\sum_{i\in I} a_i) \cdot a = \sum_{i\in I}(a_i \cdot a)$$

for all families $(a_i \mid i \in I)$.

A *complete partially ordered set*, or *cpo*, for short, is a partially ordered set P which has a bottom element, usually denoted $\bot$, such that $\sup D$ exists for each directed set $D \subseteq P$. Note that continuous monoids and continuous semirings are cpo's. When P and Q are cpo's, a function $f : P \to Q$ is called continuous if f preserves the least upper bound of directed sets (see also above). It is clear that any composition of continuous functions is continuous, and any direct product of cpo's is a cpo equipped with the pointwise order. Moreover, when P, Q are

cpo's, the set of continuous functions $P \to Q$ equipped with the pointwise order is also a cpo.

Suppose that P and Q_i, $i \in I$, are cpo's and let $\prod_{i\in I} Q_i$ denote the direct product of the Q_i. Then for any $j \in I$, the jth projection function $\prod_{i\in I} Q_i \to Q_j$ is continuous. Moreover, a function $f : P \to \prod_{i\in I} Q_i$ is continuous iff each "component function" $f_i : P \to Q_i$ is continuous. And when I is finite, say $I = \{1, \ldots, n\}$, then a function $f : \prod_{i\in I} Q_i \to P$ is continuous iff it is continuous separately in each argument, i. e., when

$$f(a_1, \ldots, \sup D, \ldots, a_n) = \sup f(a_1, \ldots, D, \ldots, a_n)$$

holds for each $1 \leq i \leq n$, $a_j \in Q_j$, $j \neq i$, and for each directed set $D \subseteq Q_i$.

2 Fixed points

The following well-known fixed point theorem plays a central rôle in our considerations, see, e. g., Bloom, Ésik [2], Guessarian [9].

Theorem 2.1 *Suppose that P and Q are cpo's and f is a continuous function $P \times Q \to P$. Then for each $q \in Q$ there is a least $p \in P$ with $p = f(p, q)$, called the* least fixed point of f with respect to the parameter q. *Moreover, the function $Q \to P$ that takes q to the least fixed point p is continuous.*

The function $Q \to P$ arising from Theorem 2.1 that provides the parameterized least fixed point for a given continuous function $f : P \times Q \to P$ is denoted $\mu x.f(x, y)$.

We now recall three very important elementary facts about least fixed points of continuous functions. Theorem 2.2 is independently due to Bekić [1] and De Bakker, Scott [3]. For Proposition 2.3, see also Niwiński [15].

Theorem 2.2 *Suppose that $f : P \times Q \times R \to P$ and $g : P \times Q \times R \to Q$ are continuous functions, where P, Q, R are cpo's. Let $h : P \times Q \times R \to P \times Q$ denote the "target pairing" of f and g, so that $h(x, y, z) = (f(x, y, z), g(x, y, z))$. Then*

$$\mu(x, y).h(x, y, z) = (\mu x.f(x, k(x, z), z), k(\mu x.f(x, k(x, z), z), z))$$

where $\mu(x, y).h(x, y, z) : R \to P \times Q$ and $k(x, z) = \mu y.g(x, y, z) : P \times R \to Q$.

Proposition 2.3 *Suppose that $f : P \times P \times Q \to P$ is a continuous function, where P and Q are cpo's. Then*

$$\mu x.\mu y.f(x, y, z) = \mu x.f(x, x, z).$$

Proposition 2.4 *Suppose that $f : P \times Q \to P$ and $g : R \to Q$ are continuous functions, where P, Q, R are all cpo's. Then*

$$\mu x.f(x, g(z)) = h(g(z)),$$

where $h(y) = \mu x.f(x,y)$.

We refer to the equation in Theorem 2.2 as the *Bekić-De Bakker-Scott rule*. The equation in Proposition 2.3 is usually referred to as the *diagonal equation*, or the *double iteration equation*. In the terminology of Bloom, Ésik [2], Proposition 2.4 asserts that the *parameter identity* holds.

The above results describe three equational properties of the least fixed point operation on continuous functions. For a complete description, we refer the reader to Bloom, Ésik [2], Ésik [5]. Least fixed points of continuous functions on cpo's are also least pre-fixed points. In Ésik [4], it is shown that the equational properties of the least fixed point operation on continuous functions on cpo's are exactly the same as those of the least pre-fixed point operation on order preserving functions on partially ordered sets in general.

3 Algebraic systems

We now define the basic notions concerning algebraic systems. In the sequel, A denotes a continuous commutative semiring, Σ denotes an alphabet of letters and $Y = \{y_1, \ldots, y_n\}$ denotes a finite set of variables. An *algebraic system* (with variables in $Y = \{y_1, \ldots, y_n\}$) is a system of equations

$$y_i = p_i, \qquad 1 \leq i \leq n\,,$$

where each p_i is a polynomial in $A\langle(\Sigma \cup Y)^*\rangle$. A *solution* to the algebraic system $y_i = p_i$, $1 \leq i \leq n$, is given by $(\sigma_1, \ldots, \sigma_n) \in (A\langle\langle\Sigma^*\rangle\rangle)^n$ such that

$$\sigma_i = p_i(\sigma_1, \ldots, \sigma_n), \qquad 1 \leq i \leq n\,.$$

A solution $(\sigma_1, \ldots, \sigma_n)$ of the algebraic system $y_i = p_i$, $1 \leq i \leq n$, is termed a *least solution* iff

$$\sigma_i \leq \tau_i, \qquad 1 \leq i \leq n\,,$$

for all solutions $(\tau_1, \ldots, \tau_n)$ of $y_i = p_i$, $1 \leq i \leq n$.

Often it is convenient to write the algebraic system $y_i = p_i$, $1 \leq i \leq n$, in matrix notation. Defining the two column vectors

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \quad \text{and} \quad p = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix},$$

we can write our algebraic system in the matrix notation

$$y = p(y) \qquad \text{or} \qquad y = p\,.$$

A *solution* to $y = p(y)$ is now given by $\sigma \in (A\langle\langle\Sigma^*\rangle\rangle)^n$ such that $\sigma = p(\sigma)$. A solution σ of $y = p$ is termed a *least solution* iff $\sigma \leq \tau$ for all solutions τ of $y = p$.

The main result we refer to in this section will be that an algebraic system has a unique least solution. This is due to the following observations.

If the semiring A is a continuous commutative semiring then $A\langle\langle\Sigma^*\rangle\rangle$ is a continuous semiring.

Moreover, for $p \in A\langle\langle(\Sigma \cup Y)^*\rangle\rangle$, where $\Sigma \cap Y = \varnothing$, p induces a continuous function $p : A\langle\langle\Sigma^*\rangle\rangle \to A\langle\langle\Sigma^*\rangle\rangle$ in the variables of Y. Let now $p \in (A\langle(\Sigma \cup Y)^*\rangle)^{n\times 1}$, i. e., p is a column vector of polynomials. Then p induces a mapping $p : (A\langle\langle\Sigma^*\rangle\rangle)^n \to (A\langle\langle\Sigma^*\rangle\rangle)^n$ by $(p(r_1, \ldots, r_n))_i = p_i(r_1, \ldots, r_n)$, $1 \le i \le n$, i. e., the i-th component of the value of p at $(r_1, \ldots, r_n) \in (A\langle\langle\Sigma^*\rangle\rangle)^n$ is given by the value of the i-th component p_i of p at $(r_1, \ldots, r_n)$. It is then easily shown that this mapping $p : (A\langle\langle\Sigma^*\rangle\rangle)^n \to (A\langle\langle\Sigma^*\rangle\rangle)^n$ is continuous.

Consider now an algebraic system $y = p$. The least fixpoint of the mapping p is nothing else than the least solution of $y = p$.

Theorem 3.1 *Let A be a continuous commutative semiring. Then the least solution of an algebraic system $y = p$ exists in $(A\langle\langle\Sigma^*\rangle\rangle)^n$ and equals*

$$\mu y.p(y) = \sup(p^j(0) \mid j \in \mathbb{N})\,.$$

Proof. By Theorem 2.1. □

Theorem 3.1 indicates how we can compute an approximation to the least solution of an algebraic system $y = p$. The *approximation sequence* σ^0, σ^1, σ^2,...,σ^j,..., where each $\sigma^j \in (A\langle\langle\Sigma^*\rangle\rangle)^{n\times 1}$, *associated to* an algebraic system $y = p(y)$ is defined as follows:

$$\sigma^0 = 0, \qquad \sigma^{j+1} = p(\sigma^j),\ j \in \mathbb{N}\,.$$

Clearly, $(\sigma^j \mid j \in \mathbb{N})$ is a chain and $\mu y.p(y) = \sup(\sigma^j \mid j \in \mathbb{N})$, i.,e., we obtain the least solution of $y = p$ by computing the least upper bound of the approximation sequence associated to it.

The collection of the components of the least solutions of all algebraic systems is denoted by $A^{\mathrm{alg}}\langle\langle\Sigma^*\rangle\rangle$.

4 Applications

Our first application considers algebraic systems of the form $y = My + P$.

By Proposition 1.1, all sums exist in $A\langle\langle\Sigma^*\rangle\rangle$ and thus in the semiring $(A\langle\langle\Sigma^*\rangle\rangle)^{n\times n}$ of all $n \times n$-matrices over $A\langle\langle\Sigma^*\rangle\rangle$. When M is such a matrix,

$$M^* = \sum_{i\le 0} M^i\,.$$

An algebraic system $y_i = p_i$, $1 \le i \le n$, is called *linear system* if $p_i = \sum_{1\le j\le n} M_{ij}y_j + R_i$, where $M_{ij}, R_i \in A\langle\Sigma^*\rangle$. Let $M \in (A\langle\Sigma^*\rangle)^{n\times n}$ (resp. $R \in (A\langle\Sigma^*\rangle)^{n\times 1}$) be the matrix (resp. column vector) with entries M_{ij} (resp. R_i). Then such a linear system can be written in matrix notation as $y = My+R$. The

approximation sequence $\sigma^0, \sigma^1, \sigma^2, \ldots, \sigma^j, \ldots$, where each $\sigma^j \in (A\langle\langle \Sigma^* \rangle\rangle)^{n\times 1}$, associated to the linear system $y = My + R$ is given by:

$$\sigma^0 = 0, \qquad \sigma^{j+1} = \sum_{0\leq i\leq j} M^i R, \quad j \geq 0\,.$$

Hence, we have proved the following result.

Theorem 4.1 *Let A be a continuous commutative semiring. Then M^*R is the least solution of the linear system $y = My + R$, where $M \in (A\langle \Sigma^* \rangle)^{n\times n}$ and $R \in (A\langle \Sigma^* \rangle)^{n\times 1}$.*

We will now consider a very useful transformation of an algebraic system. We write an algebraic system in the form $y = My + P$, where $M \in (A\langle\varepsilon\rangle)^{n\times n}$ and $\mathrm{supp}(P_i) \subseteq (\Sigma \cup Y)^* - Y$, $1 \leq i \leq n$. Here the entries of My contain polynomials of the form ay_i, $a \in A$, $y_i \in Y$.

Theorem 4.2 *The least solutions of the algebraic systems $y = My + P$ and $y = M^*P$, where $M \in (A\langle\varepsilon\rangle)^{n\times n}$ and $\mathrm{supp}(P_i) \subseteq (\Sigma \cup Y)^* - Y$, $1 \leq i \leq n$, coincide.*

Proof. We use the proof method of Ésik, Leiß [6]. By the diagonal equation, Proposition 2.3, and by Theorem 4.1:

$$\mu y.(My + P(y)) = \mu y.\mu x.(Mx + P(y)) = \mu y.M^*P(y)\,.$$

□

Observe that the context-free grammar corresponding to the algebraic system $y = M^*P$ has no *chain rules*, i. e., has no productions of the type $y_i \to y_j$. (Compare with Salomaa [16], Theorem 6.3; Harrison [10], Theorem 4.3.2; Hopcroft, Ullman [11], Theorem 4.4.)

Our second application yields a Kleene Theorem for algebraic power series. It is a generalization of a result of Gruska [8]. The presentation follows the lines of Kuich [13].

In the sequel, Σ_∞ denotes an infinite alphabet and Σ denotes a finite subalphabet of Σ_∞. All occuring symbols and variables are elements of Σ_∞. Our basic semiring will be $A\langle\langle \Sigma_\infty^* \rangle\rangle$.

For the convenience of the reader, we formulate Theorem 2.2 (Bekić-De Bakker-Scott rule) and Proposition 2.4 in the setting of $A\langle\langle \Sigma_\infty^* \rangle\rangle$.

We introduce the following notation: Let $r(y_1, \ldots, y_i, \ldots, y_n) \in A\langle\langle \Sigma_\infty^* \rangle\rangle$, where $y_1, \ldots, y_i, \ldots, y_n$ are variables that may occur in r (besides, there may occur also other variables).

Consider disjoint alphabets $\{y_1, \ldots, y_n\}$ and $\{z_1, \ldots, z_m\}$ of variables and let $\hat{\Sigma}_\infty = \Sigma_\infty - \{y_1, \ldots, y_n, z_1, \ldots, z_m\}$. Let $p_i(z_1, \ldots, z_m, y_1, \ldots, y_n)$, $1 \leq i \leq n$, and $q_j(z_1, \ldots, z_m, y_1, \ldots, y_n)$, $1 \leq j \leq m$, be power series in $A\langle\langle \Sigma_\infty^* \rangle\rangle$ and consider the system of equations

$$\begin{aligned} z_j &= p_j(z_1, \ldots, z_m, y_1, \ldots, y_n), \quad 1 \leq j \leq m\,, \\ y_i &= q_i(z_1, \ldots, z_m, y_1, \ldots, y_n), \quad 1 \leq i \leq n\,. \end{aligned}$$

Let $(t_1(z_1,\dots,z_m),\dots,t_n(z_1,\dots,z_m)) \in (A\langle\langle(\hat{\Sigma}_\infty \cup \{z_1,\dots,z_m\})^*\rangle\rangle)^n$ and $(r_1,\dots,r_m) \in (A\langle\langle\hat{\Sigma}^*_\infty\rangle\rangle)^n$ be the least solutions of the systems $y_i = q_i(z_1,\dots,z_m,y_1,\dots,y_n)$, $1 \le i \le n$, and $z_j = p_j(z_1,\dots,z_m,t_1(z_1,\dots,z_m),\dots,t_n(z_1,\dots,z_m))$, $1 \le j \le m$, respectively. Then $(r_1,\dots,r_m,t_1(r_1,\dots,r_m),\dots,t_n(r_1,\dots,r_m))$ is the least solution of the original system.

In the next proposition we use a vectorial notation: $z = (z_1,\dots,z_m)$, $y = (y_1,\dots,y_n)$, $p = (p_1,\dots,p_m)$, $q = (q_1,\dots,q_n)$, etc.

Theorem 4.3 (Bekić-De Bakker-Scott rule) *Consider the system of equations*

$$z = p(z,y), \quad y = q(z,y).$$

Let $t(z)$ and r be the least solutions of the systems $y = q(z,y)$ and $z = p(z,t(z))$, respectively. Then $(r,t(r))$ is the least solution of the system $z = p(z,y)$, $y = q(z,y)$.

Moreover, r is the least solution of the system $z = p(z,t(r))$.

We denote the least $\sigma \in A\langle\langle(\Sigma_\infty - \{y_i\})^*\rangle\rangle$ such that $r(y_1,\dots,\sigma,\dots,y_n) = \sigma$ by $\mu y_i.r(y_1,\dots,y_i,\dots,y_n)$, $1 \le i \le n$. This means that σ is the least solution of the equation $y_i = r(y_1,\dots,y_i,\dots,y_n)$ and μy_i is a fixed point operator. Observe that $\mu y_i.r(y_1,\dots,y_i,\dots,y_n) \in A\langle\langle(\Sigma_\infty - \{y_i\})^*\rangle\rangle$.

Proposition 4.4 *Let $r(y_1,\dots,y_n,y) \in A\langle\langle\Sigma^*_\infty\rangle\rangle$ and $\sigma_i \in A\langle\langle(\Sigma_\infty - \{y\})^*\rangle\rangle$, $1 \le i \le n$. Let $s(y_1,\dots,y_n) = \mu y.r(y_1,\dots,y_n,y)$. Then*

$$s(\sigma_1,\dots,\sigma_n) = \mu y.r(\sigma_1,\dots,\sigma_n,y).$$

Proof. By Proposition 2.4. □

A subsemiring $\bar{A}$ of $A\langle\langle\Sigma^*_\infty\rangle\rangle$ is called *equationally closed* iff, for all $r \in \bar{A}$ and $y \in \Sigma_\infty$ the power series $\mu y.r$ is again in $\bar{A}$.

Let $A\{\Sigma^*_\infty\} = \{r \in A\langle\Sigma^*\rangle \mid \Sigma \subset \Sigma_\infty \text{ finite}\}$ and $A^{\mathrm{alg}}\{\{\Sigma^*_\infty\}\} = \{r \in A^{\mathrm{alg}}\langle\langle\Sigma^*\rangle\rangle \mid \Sigma \subset \Sigma_\infty \text{ finite}\}$. Denote by $A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\}$ the least equationally closed semiring containing $A\{\Sigma^*_\infty\}$. We will prove in this section that $A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\} = A^{\mathrm{alg}}\{\{\Sigma^*_\infty\}\}$.

Theorem 4.5 *Let $t(y_1,\dots,y_n)$, $\sigma_j \in A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\}$, $1 \le j \le n$.*
Then $t(\sigma_1,\dots,\sigma_n) \in A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\}$.

Proof. The proof is by induction on the number of applications of the operations $+$, $\cdot$ and μ to generate $t(y_1,\dots,y_n)$.

(i) Let $t(y_1,\dots,y_n) \in A\{\Sigma^*_\infty\}$, i. e., $t(y_1,\dots,y_n) \in A\langle\Sigma^*\rangle$ for some $\Sigma \subset \Sigma_\infty$. Since $t(\sigma_1,\dots,\sigma_n)$ is generated from $\sigma_1,\dots,\sigma_n$ by applications of sum, product and scalar product, we infer that $t(\sigma_1,\dots,\sigma_n) \in A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\}$.

(ii) We only prove the case of the operator μ. Let $\sigma_1,\dots,\sigma_n \in A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\} \cap A\langle\langle\Sigma^*\rangle\rangle$ for some Σ and choose a $y \in \Sigma_\infty$ that is not in $\Sigma \cup \{y_1,\dots,y_n\}$. Without loss of generality assume that $t(y_1,\dots,y_n) = \mu y.r(y_1,\dots,y_n,y)$ (the variable y is "bound"), where $r(y_1,\dots,y_n,y) \in A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\}$. By induction hypothesis, we obtain $r(\sigma_1,\dots,\sigma_n,y) \in A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\}$. Hence, $t(\sigma_1,\dots,\sigma_n) = \mu y.r(\sigma_1,\dots,\sigma_n,y) \in A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\}$ by Proposition 4.4. □

Theorem 4.6 $A^{\mathrm{alg}}\{\{\Sigma^*_\infty\}\} \subseteq A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\}$.

Proof. The proof is by induction on the number of variables of algebraic systems. We use the following induction hypothesis: If $\tau \in (A^{\mathrm{alg}}\{\{\Sigma^*_\infty\}\})^n$, $n \geq 1$, is the least solution of an algebraic system $y_i = q_i(y_1, \ldots, y_n)$, $1 \leq i \leq n$, with n variables $y_1, \ldots, y_n$ where $q_i \in A\{\Sigma^*_\infty\}$, then $\tau_i \in A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\}$.

(1) Let $n = 1$ and assume that r is the least solution of the algebraic system $z = p(z)$. Then $r = \mu z.p(z) \in A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\}$.

(2) Let $z, y_1, \ldots, y_n$ be variables and $p, q_1, \ldots, q_n$ be polynomials in $A\{\Sigma^*_\infty\}$, and consider the algebraic system $z = p(z, y)$, $y = q(z, y)$, where $y = (y_1, \ldots, y_n)$ and $q = (q_1, \ldots, q_n)$. Let $t(z) \in (A^{\mathrm{alg}}\{\{\Sigma^*_\infty\}\})^n$ be the least solution of $y = q(z, y)$. By our induction hypothesis we obtain $t(z) \in (A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\})^n$. Since $p(z, y)$ is a polynomial, it is in $A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\}$. Hence, by Proposition 4.4, $p(z, t(z))$ is in $A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\}$. This implies $\mu z.p(z, t(z)) \in A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\}$. Again, by Proposition 4.4, $t(\mu z.p(z, t(z)) \in (A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\})^n)$. By Theorem 4.3, $(\mu z.p(z, t(z)), t(\mu z.p(z, t(z))))$ is the least solution of the algebraic system $z = p(z, y)$, $y = q(z, y)$. Hence the components of the least solution of this algebraic system are in $A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\}$. □

We now show the converse to Theorem 4.6.

Theorem 4.7 $A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\} \subseteq A^{\mathrm{alg}}\{\{\Sigma^*_\infty\}\}$.

Proof. We show that $A^{\mathrm{alg}}\{\{\Sigma^*_\infty\}\}$ is an equationally closed semiring that contains $A\{\Sigma^*_\infty\}$. Clearly, $A^{\mathrm{alg}}\{\{\Sigma^*_\infty\}\}$ is a semiring containing $A\{\Sigma^*_\infty\}$. Hence we have only to show that $\mu z.r$ is in $A^{\mathrm{alg}}\{\{\Sigma^*_\infty\}\}$ for all $r \in A^{\mathrm{alg}}\{\{\Sigma^*_\infty\}\}$ and $z \in \Sigma_\infty$.

Let $r \in A^{\mathrm{alg}}\{\{\Sigma^*_\infty\}\}$ be the first component of the least solution of the algebraic system $y_i = p_i(y_1, \ldots, y_n, z)$, $1 \leq i \leq n$. Then, by Theorem 4.3, $\mu z.r$ is the z-component of the least solution of the algebraic system $z = y_1$, $y_i = p_i(y_1, \ldots, y_n, z)$, $1 \leq i \leq n$. □

Corollary 4.8 *Let A be a continuous commutative semiring. Then $A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\} = A^{\mathrm{alg}}\{\{\Sigma^*_\infty\}\}$ and $A^{\mathrm{equ}}\{\{\Sigma^*_\infty\}\} \cap A\langle\langle\Sigma^*\rangle\rangle = A^{\mathrm{alg}}\langle\langle\Sigma^*\rangle\rangle$, $\Sigma \subset \Sigma_\infty$, Σ finite.*

References

1. Bekić, H.: Definable operations in general algebras, and the theory of automata and flowcharts. Tech. Report, IBM Labor, Wien, 1967.
2. Bloom, S. L., Ésik, Z.: Iteration Theories. EATCS Monographs on Theoretical Computer Science. Springer, 1993.
3. De Bakker, J. W., Scott, D.: A theory of programs, IBM Seminar, Wien, 1969.
4. Ésik, Z.: Completeness of Park induction. Theor. Comput. Sci. 177(1997) 217–283.
5. Ésik, Z.: Group axioms for iteration. Inform. and Comput. 148(1999) 131–180.
6. Ésik, Z., Leiß, H.: Greibach normal form in algebraically complete semirings. CSL2002, Lect. Notes Comput. Sci. 2471(2002) 135–150.
7. Ginsburg, S., Rice, H. G.: Two families of languages related to ALGOL. J. Assoc. Comput. Mach. 9(1962) 350–371.

8. Gruska, J.: A characterization of context-free languages. Journal of Computer and System Sciences 5(1971) 353–364.
9. Guessarian, I.: Algebraic Semantics. Lect. Notes Comput. Sci. 99, Springer, 1981.
10. Harrison, M. A.: Introduction to Formal Language Theory. Addison-Wesley, 1978.
11. Hopcroft, J. E., Ullman, J. D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979.
12. Kuich, W.: Semirings and formal power series: Their relevance to formal languages and automata theory. In: Handbook of Formal Languages (Eds.: G. Rozenberg and A. Salomaa), Springer, 1997, Vol. 1, Chapter 9, 609–677.
13. Kuich, W.: Gaußian elimination and a characterization of algebraic power series. MFCS 98, Lect. Notes Comput. Sci. 1450(1998) 512–521.
14. Markowsky, G.: Chain-complete posets and directed sets with applications. Algebra Universalis 6(1976) 53–68.
15. Niwiński, D.: On fixed-point clones (extended abstract). ICALP 86, Lect. Notes Comput. Sci. 226(1986) 464-473.
16. Salomaa, A.: Formal Languages. Academic Press, 1973.

Solving language equations over synchronous and parallel composition operators

Nina Yevtushenko[1], Tiziano Villa[2], Svetlana Zharikova[1]

[1] Tomsk State University, 36 Lenin Str., Tomsk, Russia
yevtushenko@elefot.tsu.ru, szh@ultranet.tomsk.ru
[2] DI, University of Verona, Strada le Grazie, 15 - 37134 Verona, Italy and
PARADES, Via S.Pantaleo, 64 - 00186 Roma, Italy
tiziano.villa@univr.it

Abstract. Consider the problem of designing a component of a discrete event system that combined with a known part of a system conforms to a given overall specification. A key observation is that languages are a suitable common framework for such applications, i.e., the problem is reduced to solving an abstract equation over languages. In this presentation, we address the problem of solving synchronous and parallel language equations. We study the most general solutions to language equations defining the language operators needed to express them, and also investigate restricted solutions to such equations. We show how an equation can be effectively solved over regular languages. In particular, we show that a solvable equation always has the largest solution, consider the largest alphabet of actions over which a solution exists, and briefly sketch how to compute the solution.

1 Introduction

Many problems over discrete event systems can be reduced to solving a language inequality $A@X \subseteq S$ or to solving a language equation $A@X = S$ where X is a free variable and @ is the composition operator. The applications range from logical synthesis and supervisory control to logic verification and testing, from model matching problem to discrete games. For different applications, appropriate equations were formulated and their solutions were investigated by various researchers. Most papers are devoted to synchronous composition in Finite State Machine (FSM) theory and to parallel composition in process algebra. Synchronous composition corresponds to instantaneous communication of systems, while parallel composition corresponds to communicating asynchronously allowing arbitrary delay between communication events. Different types of languages were considered when solving a language equation: e.g., regular and ω–regular languages, Petri net languages etc. A key point is to find a solution within the same class, i.e., if we solve an equation over regular languages then the solution should be a regular language too.

Similarly to other kinds of equations, the solution may not be unique; hence there is also the problem of finding the "best" solution. Also subsets of solutions

M. Kunc, A. Okhotin (Eds.): Theory and Applications of Language Equations
TUCS General Publication No 44, pp. 14–32, 2007.

that have appropriate additional properties might be required; thus, restricted solutions to an equation are of interest also. To find optimal solutions with respect to some criteria, one approach is first to find the largest solution containing any particular solution, and then to extract a wanted solution from the largest solution.

In this paper, we consider binary synchronous and parallel language inequalities and equations and show that each solvable language equation has the largest solution, whose property is that each solution is a subset of it; in particular, for inequalities a language is a solution if and only if it is a subset of the largest solution. We then discuss how the equations can be effectively solved for regular languages based on operators over finite automata and consider some restricted solutions which are of theoretical or practical relevance.

The paper is structured as follows. Section 2 deals with synchronous language equations and discusses their largest and restricted solutions. Parallel language equations are considered in Section 3. Effective algorithms for solving equations over regular languages based on finite automata are presented in Section 4. Section 5 briefly describes the related work and how the current paper unifies the known approaches for solving equations over languages.

2 Synchronous equations over languages

In this section, we introduce the notion of a synchronous composition operator over languages. To the best of our knowledge there was no such a notion before our papers [1, 2]. However, a similar composition operator was defined for particular cases of the composition of Finite State Machines (FSM) (see, for example, [3, 4]) and for these particular cases, the language of the FSM composition coincides with the synchronous composition of component FSM languages, as it is defined in the following section.

2.1 Synchronous composition operator over languages

An alphabet is a non-empty set of symbols. The set of all finite strings over a fixed alphabet A is denoted by A^*. A^* includes the empty string ε. A subset $L \subseteq A^*$ is called a *language* over alphabet A. Standard operations on languages are defined: *union*, *intersection*, *complement* and *difference*. We also introduce some additional operators over languages. A *substitution* [5] f is a mapping of an alphabet A into subsets of B^* for some alphabet B. The substitution f is extended to strings by setting $f(\varepsilon) = \varepsilon$ and $f(\beta a) = f(\beta)f(a)$.

Given a finite set $\Gamma = \{A_1, \ldots, A_k\}$ of alphabets, a non-empty subset $\Gamma_1 \subseteq \Gamma$ and a language L over the Cartesian product $A_1 \times \ldots \times A_k$ [1], the language $L_{\downarrow\Gamma 1} = \{\beta_{\downarrow\Gamma 1} \colon \beta \in L\}$ where $_{\downarrow\Gamma 1}$ is the canonical mapping $_{\downarrow\Gamma 1} \colon A_1 \times \ldots \times A_k \to \underset{A_j \in \Gamma_1}{\times} A_j$, is the *projection* of language L onto the set Γ_1.

[1] Without loss of generality in this paper, we assume that each alphabet participates in the Cartesian product only once.

Given the Cartesian product $\underset{A_j \in \Gamma_1}{\times} A_j$, let f be the substitution such that $f(\mathbf{a}_1)=\{\mathbf{a}\colon \mathbf{a} \in \underset{A_j \in \Gamma}{\times} A_j \ \&\ \mathbf{a}_{\downarrow \Gamma 1} = \mathbf{a}_1\}$ for all $\mathbf{a}_1 \in \underset{A_j \in \Gamma_1}{\times} A_j$. Then given a language L over the Cartesian product $\underset{A_j \in \Gamma_1}{\times} A_j$, the language $L_{\uparrow \Gamma} = \{f(\beta)\colon \beta \in L\}$ is the *lifting* of language L over the set Γ. By definition, the lifting of the empty language is empty.

The following straightforward facts hold for the projection and lifting operators.

Proposition 2.1. Given non-empty subsets Γ_1 and Γ_2 of the set Γ, for each sequence α over the Cartesian product of alphabets of the set $\Gamma_1 \cup \Gamma_2$ it holds that

$$\forall \beta \in \alpha_{\uparrow \Gamma}(\beta_{\downarrow \Gamma 1} = \alpha_{\downarrow \Gamma 1} \text{ and } \beta_{\downarrow \Gamma 2} = \alpha_{\downarrow \Gamma 2}).^2$$

Proposition 2.2. Given non-empty subsets Γ_1 and Γ_2 of the set Γ and languages L and M over the Cartesian products $\underset{A_j \in \Gamma_1}{\times} A_j$ and $\underset{A_i \in \Gamma_2}{\times} A_i$ correspondingly, for each sequence over the Cartesian product of alphabets of $\Gamma_1 \cup \Gamma_2$ it holds that

$$\alpha \in L_{\uparrow \Gamma_1 \cup \Gamma_2} \cap M_{\uparrow \Gamma_1 \cup \Gamma_2} \text{ if and only if } \alpha_{\downarrow \Gamma 1} \in L \text{ and } \alpha_{\downarrow \Gamma 2} \in M.$$

Proposition 2.3. Given non-empty subsets Γ_1 and Γ_2 of the set Γ and languages L and M over the Cartesian products $\underset{A_j \in \Gamma_1}{\times} A_j$ and $\underset{A_i \in \Gamma_2}{\times} A_i$ correspondingly, for each sequence α over the Cartesian product of alphabets of Γ, it holds that

$$\alpha_{\downarrow \Gamma_1 \cup \Gamma_2} \in L_{\uparrow \Gamma_1 \cup \Gamma_2} \cap M_{\uparrow \Gamma_1 \cup \Gamma_2} \text{ if and only if } \alpha_{\downarrow \Gamma 1} \in L \text{ and } \alpha_{\downarrow \Gamma 2} \in M.$$

We now consider a so-called synchronous composition operator over languages. The synchronous composition operator corresponds to instantaneous communication of discrete event systems. This composition operator generalizes the synchronous composition operator introduced for languages over special alphabets in [2, 6].

Let $\Gamma = \{A_1, \ldots, A_k\}$ be a finite set of alphabets, Γ_1 and Γ_2 be non-empty subsets of Γ and θ be a non-empty subset of $\Gamma_1 \cup \Gamma_2$. Given two languages L_1 and L_2 defined over the Cartesian product of alphabets of the sets Γ_1 and Γ_2 correspondingly, the *synchronous* composition $\bullet_\theta\ (L_1,L_2)$, or simply $L_1 \bullet_\theta L_2$, is the language $[L_{1\uparrow \Gamma_1 \cup \Gamma_2} \cap L_{2\uparrow \Gamma_1 \cup \Gamma_2}]_{\downarrow \theta}$. The composition language is empty if one component language is empty.

According to the definition, the above operator is commutative. We show below that the operator also is associative.

Proposition 2.4. The synchronous composition operator is associative.

Proof. Consider non-empty subsets Γ_1, Γ_2 and Γ_3 of the set Γ, $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3$, a non-empty subset $\theta \subseteq \Gamma_1 \cup \Gamma_2$ and a non-empty subset $\rho \subseteq \Gamma_3 \cup \theta$. We also

[2] Use the fixed order of alphabets when lifting to Γ.

consider three languages L_1, L_2 and L_3 where the language L_j is defined over the Cartesian product of alphabets of the set Γ_j, j = 1,2,3, and show that

$$[((L_{1\uparrow\Gamma_1\cup\Gamma_2} \cap\ L_{2\uparrow\Gamma_1\cup\Gamma_2})_{\downarrow\theta})_{\uparrow\Gamma_3\cup\theta}\ \cap\ L_{3\uparrow\Gamma_3\cup\theta}]_{\downarrow\rho} = [L_{1\uparrow\Gamma} \cap L_{2\uparrow\Gamma} \cap L_{3\uparrow\Gamma}]_{\downarrow\rho}.$$

Part A We prove that

$$((L_{1\uparrow\Gamma_1\cup\Gamma_2} \cap\ L_{2\uparrow\Gamma_1\cup\Gamma_2})_{\downarrow\theta})_{\uparrow\Gamma_3\cup\theta}\ \cap\ L_{3\uparrow\Gamma_3\cup\theta} \supseteq L_{1\uparrow\Gamma} \cap L_{2\uparrow\Gamma} \cap L_{3\uparrow\Gamma}.$$

Let $\alpha \in L_{1\uparrow\Gamma} \cap L_{2\uparrow\Gamma} \cap L_{3\uparrow\Gamma}$. According to Propositions 2.2 and 2.3,

$$\alpha \in L_{1\uparrow\Gamma} \cap L_{2\uparrow\Gamma} \cap L_{3\uparrow\Gamma}$$

if and only if

$$\alpha_{\downarrow\Gamma_1} \in L_1 \text{ and } \alpha_{\downarrow\Gamma_2} \in L_2 \text{ and } \alpha_{\downarrow\Gamma_3} \in L_3$$

if and only if

$$\alpha_{\downarrow\Gamma_1\cup\Gamma_2} \in (L_{1\uparrow\Gamma_1\cup\Gamma_2} \cap L_{2\uparrow\Gamma_1\cup\Gamma_2}) \text{ and } \alpha_{\downarrow\Gamma_3} \in L_3.$$

Therefore,

$$\alpha_{\downarrow\theta} \in (L_{1\uparrow\Gamma_1\cup\Gamma_2} \cap L_{2\uparrow\Gamma_1\cup\Gamma_2})_{\downarrow\theta} \text{ and } \alpha_{\downarrow\Gamma_3} \in L_3.$$

Thus, due to Proposition 2.2,

$$\alpha \in (L_{1\uparrow\Gamma_1\cup\Gamma_2} \cap L_{2\uparrow\Gamma_1\cup\Gamma_2})_{\downarrow\theta} \cap L_{3\uparrow\Gamma_3\cup\theta}\ \text{, i.e.,}$$

$$[(L_{1\uparrow\Gamma_1\cup\Gamma_2} \cap L_{2\uparrow\Gamma_1\cup\Gamma_2})_{\downarrow\theta} \cap L_{3\uparrow\Gamma_3\cup\theta}]_{\downarrow\rho} \supseteq [L_{1\uparrow\Gamma} \cap L_{2\uparrow\Gamma} \cap L_{3\uparrow\Gamma}]_{\downarrow\rho}.$$

Part B We prove that

$$[L_{1\uparrow\Gamma} \cap L_{2\uparrow\Gamma} \cap L_{3\uparrow\Gamma}]_{\downarrow\rho} \supseteq [(L_{1\uparrow\Gamma_1\cup\Gamma_2} \cap L_{2\uparrow\Gamma_1\cup\Gamma_2})_{\downarrow\theta} \cap L_{3\uparrow\Gamma_3\cup\theta}]_{\downarrow\rho}.$$

Consider sequence $\alpha \in (L_{1\uparrow\Gamma_1\cup\Gamma_2} \cap L_{2\uparrow\Gamma_1\cup\Gamma_2})_{\downarrow\theta} \cap L_{3\uparrow\Gamma_3\cup\theta}$.
According to Proposition 2.2,

$$\alpha \in (L_{1\uparrow\Gamma_1\cup\Gamma_2} \cap L_{2\uparrow\Gamma_1\cup\Gamma_2})_{\downarrow\theta} \cap L_{3\uparrow\Gamma_3\cup\theta}$$

if and only if

$$\alpha_{\downarrow\theta} \in (L_{1\uparrow\Gamma_1\cup\Gamma_2} \cap L_{2\uparrow\Gamma_1\cup\Gamma_2})_{\downarrow\theta} \text{ and } \alpha_{\downarrow\Gamma_3} \in L_3.$$

Therefore, due to Proposition 2.1, there exists a sequence β over the Cartesian product of alphabets of the set $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3$ such that

$$\beta_{\uparrow\Gamma_1\cup\Gamma_2} \in L_{1\uparrow\Gamma_1\cup\Gamma_2} \cap L_{2\uparrow\Gamma_1\cup\Gamma_2} \text{ and } \beta_{\downarrow\theta} = \alpha_{\downarrow\theta} \text{ and } \beta_{\downarrow\Gamma_3} = \alpha_{\downarrow\Gamma_3}\ \text{, i.e.,}$$

$$\beta_{\downarrow\rho} = \alpha_{\downarrow\rho}$$

Due to Propositions 2.1 and 2.2, $\beta \in L_{1\uparrow\Gamma} \cap L_{2\uparrow\Gamma} \cap L_{3\uparrow\Gamma}$, i.e.,

$$[((L_{1\uparrow\Gamma_1\cup\Gamma_2} \cap L_{2\uparrow\Gamma_1\cup\Gamma_2})_{\downarrow\theta})_{\uparrow\Gamma_3\cup\theta} \cap L_{3\uparrow\Gamma_3\cup\theta}]_{\downarrow\rho} \subseteq [L_{1\uparrow\Gamma} \cap L_{2\uparrow\Gamma} \cap L_{3\uparrow\Gamma}]_{\downarrow\rho}.$$

□

Consider the composition of two discrete event systems in Figure 1. The language L_1 of the component S_1 is defined over the Cartesian product $I \times U \times V$, and the language L_2 of the component S_2 is defined over the Cartesian product $U \times V \times O$, where $\Gamma_1 = \{I,\ U,\ V\}$, $\Gamma_2 = \{U,\ V,\ O\}$, $\Gamma = \{I,\ U,\ V,\ O\}$ and the set $\theta = \{I,\ V,\ O\}$ describes the set of external ports of the composed system S. The systems S_1 and S_2 interact by executing the same words via common channels. When the external symbols $i \in I$, $o \in O$ and $v \in V$ appear at the external ports of the composed system, the latter accepts the triple (i, v, o) if and only if there exists $u \in U$ such that $(i, u, v) \in L_1$ and $(u, v, o) \in L_2$. In other words, a triple (α, β, γ) over $(I \times V \times O)^*$ can be executed at the external ports of the composed system if and only if there exists a sequence $\eta \in U^*$ such that the triple $(\alpha, \eta, \beta) \in L_1$ and the triple $(\eta, \beta, \gamma) \in L_2$. An example of the synchronous composition of two regular languages represented by finite automata is shown in Section 4 (Figure 2).

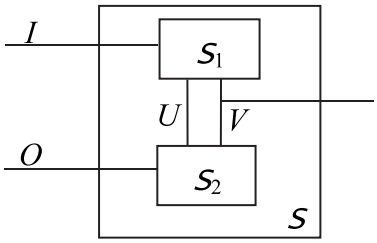


Figure 1. Composition of two discrete event systems.

Since the synchronous composition operator is commutative and associative, the definition can be naturally extended to a finite number of component languages.

Let $\Gamma = \{A_1, \ldots, A_k\}$ be a set of alphabets, $\Gamma_1, \ldots, \Gamma_m$, θ be non-empty subsets of Γ and $\Gamma = \bigcup_{j=1}^{m} \Gamma_j$. Given languages L_j, $j = 1, \ldots, m$, defined over the Cartesian product of alphabets of the set Γ_j correspondingly, the *synchronous* composition $\bullet_\theta\ (L_1, \ldots, L_m)$ is the language $[L_{1\uparrow\Gamma} \cap L_{2\uparrow\Gamma} \cap \ldots \cap L_{m\uparrow\Gamma}]_{\downarrow\theta}$ [3]. The composition language is empty if one component language is empty.

2.2 Solving synchronous language equations

Consider the composition of two discrete event systems in Figure 1. Suppose that the language L_1 of the system S_1 and the language L of the composition S are known. The problem is to determine a language L_2 over the Cartesian product $U \times V \times O$ such that the $L_1 \bullet_{\{I,O,V\}} L_2 = L$. The problem is well known

[3] Use the fixed order of alphabets when lifting to Γ.

as "the unknown component problem" and can be solved through language equation solving [2, 7].

2.2.1 Binary synchronous language equations

Given a set Γ=$\{A_1, \ldots, A_k\}$ of alphabets and non-empty subsets Γ_1 and θ of the set Γ, let language L_1 be defined over the Cartesian product of the alphabets of the set Γ_1, while the language L be defined over the Cartesian product of the alphabets of the set θ. The expressions $L_1 \bullet_\theta X \subseteq L$ and $L_1 \bullet_\theta X = L$ with a free variable X that is a language over the Cartesian product of alphabets of a non-empty set $\rho \subseteq \Gamma_1 \cup \theta$, are called a binary *synchronous language inequality* and a binary *synchronous language equation* correspondingly. A language B_ρ over the Cartesian product of alphabets of the set ρ is a *solution* to the inequality $L_1 \bullet_\theta X \subseteq L$ (or to the equation $L_1 \bullet_\theta X = L$) if $L_1 \bullet_\theta B_\rho \subseteq L$ (correspondingly, $L_1 \bullet_\theta B_\rho = L$). A solution S_ρ over the Cartesian product of alphabets of the set ρ is the *largest* solution to the inequality $L_1 \bullet_\theta X \subseteq L$ (or to the equation $L_1 \bullet_\theta X = L$) if each solution B_ρ over the Cartesian product of alphabets of the set ρ is a subset of S_ρ. The following propositions state that an inequality $L_1 \bullet_\theta X \subseteq L$ has always the largest solution; moreover, a solvable equation also has the largest solution.

Proposition 2.5. The largest solution over the Cartesian product of alphabets of the set ρ to the inequality $L_1 \bullet_\theta X \subseteq L$ is the language $S_\rho = \overline{L_1 \bullet_\rho \overline{L}}$.

Proof. A sequence α over the Cartesian product of alphabets of the set ρ is not in a solution to the inequality $L_1 \bullet_\theta X \subseteq L$ if and only if $L_1 \bullet_\theta \{\alpha\} \not\subset L$ and the following chain of equivalences follows:

$$L_1 \bullet_\theta \{\alpha\} \not\subset L$$

if and only if

$$(L_{1\uparrow\Gamma_1\cup\theta} \cap \{\alpha\}_{\uparrow\Gamma_1\cup\theta})_{\downarrow\theta} \cap \overline{L} = \varnothing$$

if and only if

$$L_{1\uparrow\Gamma_1\cup\theta} \cap \{\alpha\}_{\uparrow\Gamma_1\cup\theta} \cap \overline{L}_{\uparrow\Gamma_1\cup\theta} = \varnothing$$

if and only if

$$\alpha \notin (L_{1\uparrow\Gamma_1\cup\theta} \cap \overline{L}_{\uparrow\Gamma_1\cup\theta})_{\downarrow\rho}$$

if and only if

$$\alpha \in \overline{L_1 \bullet_\rho \overline{L}}.$$

□

Therefore, the largest solution to the inequality $L_1 \bullet_\theta X \subseteq L$ is given by the language $S_\rho = \overline{L_1 \bullet_\rho \overline{L}}$. According to the definition of synchronous composition operator it holds that $L_1 \bullet_\theta L_2 \subseteq L$ and $L_2 \subseteq L_3$ implies $L_1 \bullet_\theta L_3 \subseteq L$. Thus, the following statement holds.

Theorem 2.1. Given an inequality $L_1 \bullet_\theta X \subseteq L$, the language B_ρ over the Cartesian product of alphabets of the set ρ is a solution to the inequality if and only if $B_\rho \subseteq \overline{L_1 \bullet_\rho \overline{L}}$. Thus, if the language $\overline{L_1 \bullet_\rho \overline{L}}$ is empty then the inequality $L_1 \bullet_\theta X \subseteq L$ has a single trivial solution, namely the empty language.

As the above theorem states, the largest solution to a language inequality can be considered as the "container" of all solutions to the inequality. In the following theorem (Theorem 2.2) we state that a solvable language equation $L_1 \bullet_\theta X = L$ also has the largest solution. When the equation is unsolvable there always exists the largest subset M of the language L such that the equation $L_1 \bullet_\theta X = M$ is solvable and has the largest solution. However, not each subset of the largest solution inherits the property to be a solution and we still lack the complete characterization of the set of solutions to a synchronous language equation. Since in general the number of subsets of the largest solution is infinite, the characterization problem is not trivial.

Theorem 2.2. 1. Given a language equation $L_1 \bullet_\theta X = L$, if $L_1 \bullet_\theta \overline{L_1 \bullet_\rho \overline{L}} = L$ then the language $S_\rho = \overline{L_1 \bullet_\rho \overline{L}}$ is the largest solution to the equation $L_1 \bullet_\theta X = L$ over the Cartesian product of alphabets of the set ρ. However, not each subset of the S_ρ inherits the property to be a solution to the language equation.

2. If $L_1 \bullet_\theta \overline{L_1 \bullet_\rho \overline{L}} \subset L$ then the equation $L_1 \bullet_\theta X = L$ is unsolvable and the language $M_\theta = L_1 \bullet_\theta \overline{L_1 \bullet_\rho \overline{L}}$ is the largest subset of L such that the equation $L_1 \bullet_\theta X = M_\theta$ is solvable over the Cartesian product of alphabets of the set ρ.

Theorem 2.2 establishes necessary and sufficient conditions for the solvability of a language equation $L_1 \bullet_\theta X = L$ over a given set of alphabets. The existence of a non-empty solution to the inequality as well as the existence of a solution to the language equation significantly depends on the selected subset $\rho \subseteq \Gamma_1 \cup \theta$. The following theorems state that there exists the largest set H of alphabets such that an inequality $L_1 \bullet_\theta X \subseteq L$ has a non-empty solution if and only if such a solution exists over the Cartesian product of alphabets of H. Similar to this, an equation is solvable if and only if the equation is solvable over the Cartesian product of alphabets of H.

Theorem 2.3. 1. If the largest solution to the inequality $L_1 \bullet_\theta X \subseteq L$ over the Cartesian product of alphabets of the set $\rho \subseteq \Gamma_1 \cup \theta$ is not empty then the inequality has a non-empty solution over the Cartesian product of alphabets of the set $\Gamma_1 \cup \theta$.

2. Given largest solution $\overline{L_1 \bullet_{\Gamma_1 \cup \theta} \overline{L}}$ to the inequality $L_1 \bullet_\theta X \subseteq L$ over the Cartesian product of alphabets of the set $\Gamma_1 \cup \theta$, a language B_ρ over the Cartesian product of alphabets of the set $\rho \subset \Gamma_1 \cup \theta$ is a solution to the inequality if and only if $B_{\rho \uparrow \Gamma_1 \cup \theta} \subseteq \overline{L_1 \bullet_{\Gamma_1 \cup \theta} \overline{L}}$. If the language $\overline{L_1 \bullet_{\Gamma_1 \cup \theta} \overline{L}}$ is empty then the only solution to the inequality over the Cartesian product of alphabets of any non-empty subset $\rho \subset \Gamma_1 \cup \theta$ is the empty set.

Theorem 2.4. 1. If the equation $L_1 \bullet_\theta X = L$ is solvable over the Cartesian product of alphabets of the set $\rho \subset \Gamma_1 \cup \theta$ then the equation has a solution over the Cartesian product of alphabets of the set $\Gamma_1 \cup \theta$.

2. If the equation $L_1 \bullet_\theta X = L$ has no solution over the Cartesian product of alphabets of the set $\Gamma_1 \cup \theta$ then the equation $L_1 \bullet_\theta X = L$ is unsolvable over the Cartesian product of alphabets of any non-empty subset $\rho \subset \Gamma_1 \cup \theta$.

3. If the equation $L_1 \bullet_\theta X = L$ is solvable over the Cartesian product of alphabets of the set $\Gamma_1 \cup \theta$ and $\overline{L_1 \bullet_{\Gamma_1 \cup \theta} \overline{L}}$ is the largest solution to the equation over this alphabet, then for each solution B_ρ to the equation over the Cartesian product of alphabets of the set $\rho \subset \Gamma_1 \cup \theta$, it holds that $B_{\rho \uparrow \Gamma_1 \cup \theta} \subseteq \overline{L_1 \bullet_{\Gamma_1 \cup \theta} \overline{L}}$.

Theorems 2.1 and 2.3 completely characterize all possible solutions to synchronous language inequalities. As for language equations, Theorems 2.2 and 2.4 only establish necessary and sufficient conditions for the equation solvability over the Cartesian product of alphabets of a given set $\rho \subseteq \Gamma_1 \cup \theta$.

2.3 Restricted solutions to language equations

Given a language inequality or a language equation, not each solution is of theoretical and practical interest. Additional research is necessary to reveal which restricted solutions are useful for different applications. Currently, only some straightforward restricted solutions have been identified. For example, the composed system must be nontrivial at least. We also investigated special solutions which are prefix closed, progressive [4] and compositionally progressive [8]. We illustrate such restricted solutions for a synchronous language equation and for the simplicity of presentation we assume that component languages are defined over the Cartesian product of two alphabets.

The language L over alphabet A is *prefix-closed* if each prefix of each word is in the language L. A language L over alphabet $A = I \times O$ is *I-progressive* if $\alpha \in A^*\ \forall i \in I\ \exists o \in O\ [\alpha \in L \rightarrow \alpha(i,o) \in L]$. A language L over alphabet $A = I \times O$ is *I-defined* if $L_{\downarrow I} = I^*$. If a language over $A = I \times O$ is I-progressive then it is also I-defined, but the converse does not hold. A progressive solution ensures that the solution language is complete w.r.t. the alphabet I, i.e., for each string $\beta \in I^*$ there exists a word in the language with the projection β. Progressive solutions are used in logic synthesis, since physical devices usually are input-enabled at each state, and especially are of interest when solving an equation over Finite State Machines (FSM) [2, 4]. Given a language L_1 over alphabet $I \times U$, a language B over alphabet $O \times U$ is *I–compositionally progressive* (w.r.t. the language L_1) if the language $L_{1\uparrow\{I,U,O\}} \cap B_{\uparrow\{I,U,O\}}$ is I-progressive. When a solution to the language equation is compositionally progressive we ensure that the corresponding composition does not fall into a deadlock when I is the set of external inputs submitted by an environment.

Given language L_1 over alphabet $I \times U$, language L over alphabet $I \times O$, i.e., $\theta = \{I, O\}$, and $\rho = \{U, O\}$, let $S_\rho = \overline{L_1 \bullet_\rho \overline{L}}$ be the largest solution to the

language equation $L_1 \bullet_\theta X = L$. It is interesting to investigate subsets of S_ρ that satisfy further properties, i.e., are prefix-closed, progressive etc.

If S_ρ is prefix-closed then S_ρ is the largest prefix-closed solution to the equation. However, not every solution to the equation is prefix closed. If S_ρ is not prefix-closed then denote by $Pref(S_\rho)$ the set obtained from S_ρ by deleting each string that has a prefix not in S_ρ.

Proposition 2.6. If $L_1 \bullet_\theta Pref(S_\rho) = L$ then $Pref(S_\rho)$ is the largest prefix-closed solution to the equation $L_1 \bullet_\theta X = L$. If $L_1 \bullet_\theta Pref(S_\rho) \subset L$, then the equation $L_1 \bullet_\theta X = L$ has no prefix-closed solution.

If S_ρ is defined over the alphabet $U \times O$ and S_ρ is U-progressive then S_ρ is the largest U-progressive solution to the equation. However, not each subset of S_ρ inherits this property. If S_ρ is not U-progressive then denote by $Prog(S_\rho)$ the subset obtained from S_ρ by deleting each string β such that for some $u \in U$, there is no $o \in O$ for which $\beta(u, o) \in S_\rho$.

Proposition 2.7. If $L_1 \bullet_\theta Prog(S_\rho) = L$ then $Prog(S_\rho)$ is the largest progressive solution to the equation $L_1 \bullet_\theta X = L$. If $L_1 \bullet_\theta Prog(S_\rho) \subset L$, then the equation $L_1 \bullet_\theta X = L$ has no progressive solution.

We also studied compositionally progressive solutions over FSM languages [8] and showed that if a synchronous FSM equation (FSM inequality) has a compositionally progressive solution then the FSM equation (FSM inequality) has the largest compositionally progressive solution.

3 Solving a parallel language equation

Synchronous composition of languages corresponds to instantaneous communication and usually describes modular compositions in hardware, while parallel composition corresponds to communicating asynchronously, allowing arbitrary delay between communication events and is used as a composition operator in the process algebra. Despite of the fact that the restriction and expansion operators differ from the projection and lifting operators used in synchronous composition, synchronous and parallel language inequalities (synchronous and parallel language equations) possess almost the same features. We also notice that in the process algebra this kind of composition operator sometimes is called synchronous, since an action can be executed if and only if both communicating systems are ready to execute the action. However, in this setting, we use the notion of the parallel composition operator.

3.1 Parallel composition operator

Let A be an alphabet, and a language L is defined over the alphabet A. Consider a non-empty subset A_1 of the alphabet A and the substitution h defined as $h(a) = a$ for all $a \in A_1$ while $h(a) = \varepsilon$ for all $a \in A \backslash A_1$, where ε is the empty word. Then the language $L_{\Downarrow A_1} = \{h(\beta) : \beta \in L\}$ is the *restriction* of language L onto the subset A_1. For each word $\beta \in L$ that does not have symbols of alphabet A_1 the restriction of β is the empty word ε.

Let now language L be defined over alphabet $A_2 = A \backslash A_1$. Consider the mapping $\psi : A_2 \to 2^{A^*}$ such that $\psi(a) = \{\gamma a \beta \colon \gamma, \beta \in A_1^*\}$. Then the language $L_{\Uparrow A} = \{\psi(\beta) \colon \beta \in L\}$ is the *expansion* of language L over the alphabet A. Here we notice that the mapping ψ is not a substitution and thus, $\psi(\varepsilon) = \{\alpha \colon \alpha \in A_1^*\}$, i.e., if the language $L = \{\varepsilon\}$ is defined over alphabet A_2 then $L_{\Uparrow A} = A_1^*$. By definition, the expansion of the empty language is the empty language and the following straightforward fact holds between the restriction and expansion operators.

Proposition 3.1. Given an alphabet A and a non-empty subset A_1 of the alphabet A, consider a language L defined over the alphabet A_1. For each string $\beta \in A^*$, it holds that $\beta_{\Downarrow A_1} \in L$ if and only if $\beta \in L_{\Uparrow A}$.

Given two languages L_1 and L_2, let the language $L_j, j = 1, 2$, be defined over alphabet $A_j, A = A_1 \cup A_2$ and E is a non-empty subset of A. The *parallel* composition $\diamondsuit_E(L_1, L_2)$, or simply $L_1 \diamondsuit_E L_2$, is the language $(L_{1\Uparrow A} \cap L_{2\Uparrow A})_{\Downarrow E}$. Similar to the synchronous composition, the language $L_1 \diamondsuit_E L_2$ is empty if one component language is empty.

Similar to the synchronous composition the parallel composition operator is commutative and associative, and therefore, the operator can be naturally extended to k component languages, $k > 2$.

3.2 Parallel language equations

Given two languages L_1 and L, let language L_1 be defined over alphabet A_1, the language L be defined over alphabet E, $A_1 \cup E = A$ and R is a non-empty subset of A. Consider the language inequality $L_1 \diamondsuit_E X \subseteq L$ and a language equation $L_1 \diamondsuit_E X = L$ with a free variable X that is a language over alphabet R. A language B_R over the alphabet R is a *solution* to the inequality $L_1 \diamondsuit_E X \subseteq L$ if $L_1 \diamondsuit_E B_R \subseteq L$. The language B_R is a *solution* to the equation $L_1 \diamondsuit_E X = L$ if $L_1 \diamondsuit_E B_R = L$. A solution S_R over the alphabet R is the *largest* solution to the inequality $L_1 \diamondsuit_E X \subseteq L$ (to the equation $L_1 \diamondsuit_E X = L$) if each solution over alphabet R is a subset of S_R. Similar to a binary synchronous equation, the following results state that a parallel inequality as well as a solvable parallel language equation has always the largest solution.

Proposition 3.2. The largest solution over the alphabet R to the inequality $L_1 \diamondsuit_E X \subseteq L$ is the language $S_R = \overline{L_1 \diamondsuit_R \overline{L}}$.

Proof. A sequence α over the alphabet R is not in a solution to the inequality $L_1 \diamondsuit_E X \subseteq L$ if and only if $L_1 \diamondsuit_E \{\alpha\} \not\subset L$ and the following chain of equivalences follows:

$$L_1 \diamondsuit_E \{\alpha\} \not\subset L$$

if and only if

$$(L_{1\Uparrow A} \cap \{\alpha\}_{\Uparrow A})_{\Downarrow E} \cap \overline{L} = \varnothing$$

$$\text{if and only if}$$

$$L_{1\Uparrow A} \cap \{\alpha\}_{\Uparrow A} \cap \overline{L}_{\Uparrow A} = \varnothing$$

$$\text{if and only if}$$

$$\alpha \notin (L_{1\Uparrow A} \cap \overline{L}_{\Uparrow A})_{\Downarrow R}$$

$$\text{if and only if}$$

$$\alpha \in \overline{L_1 \diamondsuit_R \overline{L}}$$

□

Therefore, the largest solution to the inequality $L_1 \diamondsuit_E X \subseteq L$ is given by the language $S_R = \overline{L_1 \diamondsuit_R \overline{L}}$. According to the definition of the parallel composition operator it holds that $L_1 \diamondsuit_E L_2 \subseteq L$ and $L_2 \subseteq L_3$ implies $L_1 \diamondsuit_E L_3 \subseteq L$. Thus, the following statement holds.

Theorem 3.1. Given an inequality $L_1 \diamondsuit_E X \subseteq L$, the language B_R over the alphabet R is a solution to the inequality if and only if $B_R \subseteq \overline{L_1 \diamondsuit_R \overline{L}}$. Thus, if the language $\overline{L_1 \diamondsuit_R \overline{L}}$ is empty then the inequality $L_1 \diamondsuit_E X \subseteq L$ has a single trivial solution, namely the empty language.

As the above theorem states, similar to a synchronous language inequality, the largest solution to a parallel language inequality can be considered as the "container" of all solutions to the inequality. Correspondingly, the following theorem (Theorem 3.2) states that a solvable language equation $L_1 \diamondsuit_E X = L$ also has the largest solution. When the equation is unsolvable there always exists the largest subset M of the language L such that the equation $L_1 \diamondsuit_E X = M$ is solvable and has the largest solution.

Theorem 3.2. Given a language equation $L_1 \diamondsuit_E X = L$, if $L_1 \diamondsuit \overline{L_1 \diamondsuit_R \overline{L}} = L$ then the language $S_R = \overline{L_1 \diamondsuit_R \overline{L}}$ is the largest solution to the equation $L_1 \diamondsuit_E X = L$ over the alphabet R. However, not each subset of the S_R inherits the property to be a solution to the language equation.

The following theorem states that if the equation has no solution over alphabet A then the equation has no solution over any subset R of the alphabet A.

Theorem 3.3. If the equation $L_1 \diamondsuit_E X = L$ has no solution over the alphabet A then the equation is unsolvable over any alphabet R that is a subset of A. If the equation $L_1 \diamondsuit_E X = L$ is solvable over alphabet A, alphabet R is a proper subset of A and a language B_R over alphabet R is a solution to the equation then it holds that $B_{R\Uparrow A} \subseteq \overline{L_1 \diamondsuit_R \overline{L}}$.

Similar to synchronous language equations, not each subset of the largest solution to a parallel language equation inherits the property to be a solution and the complete characterization of the set of solutions to a parallel language equation is still an open issue.

We illustrate a parallel language equation solving in Section 4 when solving equations over regular languages.

4 Solving equations over regular languages

Given a language inequality or a language equation, the problem then is whether we can operate algorithmically on given languages. Usually such languages are presented through their corresponding mathematical machines. Different types of mathematical machines are used to model components of a discrete event system. The most commonly known are Finite Automata (FA), Finite State Machines (FSM), Petri Nets (PN), Finite ω-automata (ω-FA). A key point is to find a solution within the same class, i.e., if we solve an equation over regular languages, then a solution must be a regular language too. Language equations can be solved effectively when they are defined over languages that can be computed with finite procedures. For example, we can operate on Finite Automata (FA) when solving equations over regular languages [5].

A *finite automaton* is a quintuple $S = \langle S, A, \delta_S, s_0, F_S \rangle$, where S is a finite non-empty set of states with the initial state s_0 and a subset F_S of *final* (or *accepting*) states, A is an alphabet of actions, and $\delta_S \subseteq A \times S \times S$ is a transition relation. We say that there is a transition from a state s to a state $s\prime$ labeled with an action a, if and only if the triple $(a, s, s\prime)$ is in the transition relation δ_S. The automaton S is called *deterministic*, if for each state $s \in S$ and any action $a \in A$ there exists at most one state $s\prime$, such that $(a, s, s\prime) \in \delta_S$. If S is not deterministic, then it is called *nondeterministic*.

Well-known results state that each regular language can be represented by a deterministic finite automaton and that regular languages are closed under the union, intersection and complementation. Regular languages are also closed under projection, lifting, restriction and expansion. Below we sketch the constructions for the less known operations of projection, lifting, restriction and expansion.

Projection ($\downarrow$) Given FA F that accepts language L over $I \times U$, FA $F_{\downarrow I}$ that accepts language $L_{\downarrow I}$ over I is obtained by replacing each edge $((i,u),s,s\prime)$ in F by the edge $(i,s,s\prime)$. [4]

Lifting ($\uparrow$) Given FA F that accepts language L over I, FA $F_{\uparrow\{I,U\}}$ that accepts language $L_{\uparrow\{I,U\}}$ over $I \times U$ is obtained by replacing each edge $(i,s,s\prime)$ in F by the set of edges $\{((i,u),s,s\prime)\colon u \in U\}$.

Restriction ($\Downarrow$) Given FA F that accepts language L over A and a non-empty subset V of A, FA $F_{\Downarrow V}$ that accepts language $L_{\Downarrow V}$ over V is obtained by replacing each edge $(a,s,s\prime)$ in F, with $a \in A \setminus V$, by the edge $(\varepsilon,s,s\prime)$. [5]

Expansion ($\Uparrow$) Given alphabet A, a non-empty subset V of A, FA F that accepts language L over V, FA $F_{\Uparrow A}$ that accepts language $L_{\Uparrow A}$ over A is obtained

[4] Apply the subset construction to obtain an equivalent deterministic FA.

[5] Apply the closure procedure to obtain an equivalent deterministic FA without ε-moves.

by the following procedure: for each state s of FA F, $\forall a \in A \backslash V$ the edge (self-loop) (a,s,s) is added.

The procedures for projection, lifting and restriction guarantee the substitution property $f(\varepsilon) = \varepsilon$.

Given that all the operators used to express the solution of regular language equations (Theorems 2.1 and 3.1) have constructive counterparts on finite automata, we conclude that there is an effective (constructive) way to solve equations over regular languages.

As an example, given a regular language equation $L_1 \bullet_\theta X = L$, where L_1 is a regular language over alphabet $I \times U$, L is a regular language over $I \times O$, and the unknown is a regular language X over $U \times O$, an algorithm for building X follows (if a solution exists).

Procedure 4.1 Deriving the largest solution over alphabet ρ to a regular language equation $L_1 \bullet_\theta X = L$ (if a solution exists)

Input. Regular language L_1 over alphabet $I \times U$, regular language L over alphabet $I \times O$, $\rho = \{U, O\}$ and language equation $L_1 \bullet_\theta X = L$

Output. The largest solution S_ρ over alphabet $U \times O$ to the language equation (if the equation is solvable)

1. Derive finite automata F_1 and F which accept respectively regular languages L_1 and L.

2. If F is a nondeterministic automaton then determinize F by the subset construction and obtain the automaton $\overline{F}$ by interchanging the sets of accepting and non-accepting states of F.

3. Obtain the automaton $F_{1\uparrow\{I,U,O\}}$ by replacing each label (i,u) with all triples (i,u,o), $o \in O$.

4. Obtain the automaton $\overline{F}_{\uparrow\{I,U,O\}}$ by replacing each label (i,o) with all triples (i,u,o), $u \in U$.

5. Build the intersection $F_{1\uparrow\{I,U,O\}} \cap \overline{F}_{\uparrow\{I,U,O\}}$. The states of the obtained automaton are pairs of states of $F_{1\uparrow\{I,U,O\}}$ and $\overline{F}_{\uparrow\{I,U,O\}}$, the initial state is the pair of initial states, and a state of the intersection is accepting if both states of the pair are accepting.

6. Project $F_{1\uparrow\{I,U,O\}} \cap \overline{F}_{\uparrow\{I,U,O\}}$ to $\rho = \{U, O\}$ by deleting i from the labels (i,u,o). The obtained automaton in general is nondeterministic; in this case, determinize it by the subset construction and obtain the automaton $F_1 \bullet_\rho \overline{F}$ which accepts the language $L_1 \bullet_\rho \overline{L}$. Obtain the automaton S_ρ which accepts the regular language $\overline{L_1 \bullet_\rho \overline{L}}$ by interchanging the sets of accepting and non-accepting states of $F_1 \bullet_\rho \overline{F}$.

7. Derive the automaton $(F_{1\uparrow\{I,U,O\}} \cap S_{\rho\uparrow\{I,U,O\}})_{\downarrow\{I,O\}}$. If the automaton accepts the language L then the regular language $\overline{L_1 \bullet_\rho \overline{L}}$ is the largest solution over the alphabet $U \times O$ to the equation $L_1 \bullet_\theta X = L$. Otherwise, the regular language equation $L_1 \bullet_\theta X = L$ has no solution over the alphabet $U \times O$.

We illustrate the procedure for automata shown in Figure 2 where accepting states are shown in double lines. Automata F_1 and F accept correspondingly the languages L_1 and L over alphabets $I \times U$ and $I \times O$. The automaton $\overline{F}$

(Figure 2c) accepts the language $\overline{L}$. Figure 2d shows the automaton $F_{1\uparrow\{I,U,O\}} \cap \overline{F}_{\uparrow\{I,U,O\}}$ that accepts the intersection $L_{1\uparrow\{I,U,O\}} \cap \overline{L}_{\uparrow\{I,U,O\}}$. Figure 2e presents the deterministic automaton that accepts the projection of $L_{1\uparrow\{I,U,O\}} \cap \overline{L}_{\uparrow\{I,U,O\}}$ onto alphabet $U \times O$, i.e., the synchronous composition $L_1 \bullet_{\{U,O\}} \overline{L}$, while Figure 2f shows the automaton that accepts the largest solution $S_\rho = \overline{L_1 \bullet_\rho \overline{L}}$ to the equation $L_1 \bullet_{\{I,O\}} X = L$ over alphabet $U \times O$.

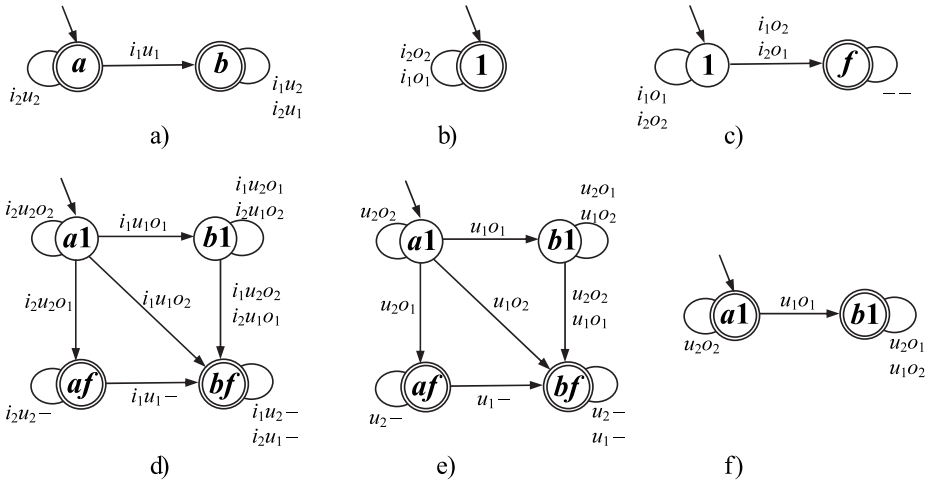


Figure 2. Solving a synchronous equation over regular languages:
a) automaton F_1; b) automaton F; c) automaton $\overline{F}$; d) automaton $L_1 \bullet_\rho L$;
e) a deterministic automaton $(F_{1\uparrow\{I,U,O\}} \cap \overline{F}_{\uparrow\{I,U,O\}})_{\downarrow\{U,O\}}$ which accepts the language $L_1 \bullet_{\{U,O\}} \overline{L}$; f) the automaton which accepts the largest solution $S_\rho = \overline{L_1 \bullet_\rho \overline{L}}$.

We also illustrate how a solution to a parallel language equation can be derived when all the languages are regular. Given a regular language equation $L_1 \diamondsuit_E X = L$ where L_1 is a regular language over alphabet A_1, L is a regular language over alphabet E, and the unknown language X is over alphabet $R \subseteq A_1 \cup E$, an algorithm to build X follows (if a solution exists).

Procedure 4.2. Deriving the largest solution over alphabet R to a regular language equation $L_1 \diamondsuit_E X = L$ (if a solution exists)

Input. Regular language L_1 over alphabet A_1, regular language L over alphabet E, $A = A_1 \cup E$, an alphabet $R \subseteq A$ and a language equation $L_1 \diamondsuit_E X = L$

Output. The largest solution S_R over alphabet R to the language equation (if the equation is solvable)

1. Derive finite automata F_1 and F which accept respectively regular languages L_1 and L.

2. If F is a nondeterministic automaton then determinize F by the subset construction and obtain $\overline{F}$ by interchanging the sets of accepting and non-accepting states of F.

3. Obtain the automaton $F_{1\Uparrow A}$ by adding a self-loop (a,s,s) for each state s of FA F_1 and each $a \in A \backslash A_1$.

Obtain the automaton $\overline{F}_{\Uparrow A}$ by adding a self-loop (a,f,f) for each state f of FA F and each $a \in A \backslash E$.

4. Build the intersection $F_{1\Uparrow A} \cap \overline{F}_{\Uparrow A}$ where states of $F_{1\Uparrow A} \cap \overline{F}_{\Uparrow A}$ are pairs of states of $F_{1\Uparrow A}$ and $\overline{F}_{\Uparrow A}$.

5. Restrict $F_{1\Uparrow A} \cap \overline{F}_{\Uparrow A}$ to the alphabet R and apply the closure construction to obtain a deterministic automaton S_R which accepts the language $\overline{L_1 \diamondsuit_R \overline{L}}$.

6. Derive the automaton $(F_{1\Uparrow A} \cap S_{R\Uparrow A})_{\Downarrow E}$. If the automaton accepts the language L then the regular language $\overline{L_1 \diamondsuit_R \overline{L}}$ is the largest solution over the alphabet R to the equation $L_1 \diamondsuit_E X = L$. Otherwise, the regular language equation $L_1 \diamondsuit_E X = L$ has no solution over the alphabet R.

If the regular language equation $L_1 \diamondsuit_E X = L$ is solvable over alphabet R then the largest solution that is a regular language is obtained by Procedure 4.2. However, the question whether a regular language equation can have solutions that are not regular is open. If the language equation is unsolvable over alphabet R then we can attempt to solve the equation over a larger alphabet. However, if the equation is unsolvable over alphabet $A = A_1 \cup E$ then the equation is unsolvable over any non-empty subset of the alphabet A.

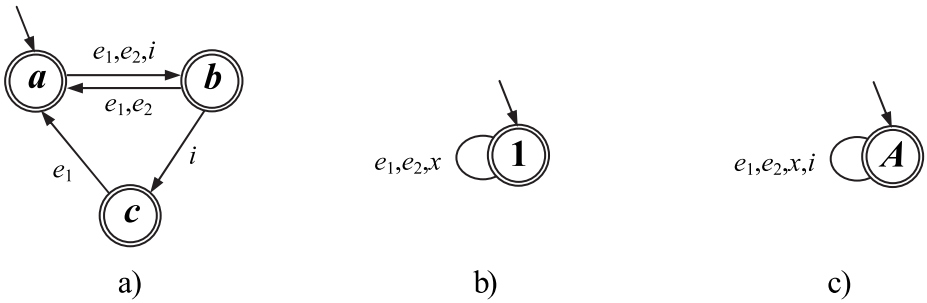


Figure 3. Solving a parallel equation over regular languages:
a) automaton F_1; b) automaton F; c) automaton S_R.

As an example, consider finite automata represented in Figure 3 [9]. We consider the automaton F_1 defined over the alphabet $A_1 = \{e_1, e_2, i\}$ and the automaton F defined over the alphabet $E = \{e_1, e_2, x\}$. The automaton S_R shown in Figure 3c is defined over alphabet $R = \{e_1, e_2, i, x\}$ and accepts the largest solution $S_R = \overline{L_1 \diamondsuit_R \overline{L}}$ to the equation $L_1 \diamondsuit_E X = L$.

Here we notice that there exist effective algorithms based on operators over finite automata for deriving largest restricted solutions [2, 4, 10]. Moreover, such algorithms exist also for deriving the largest compositionally progressive solution over regular languages [8, 9].

5 Related work

Various contributions investigating partial aspects of the topic of this research have been published in the past. The first results related to solving equations in logic synthesis over series composition of FSMs were obtained in 1972 by Kim and Newborn [11] and have been developed by the group of Brayton (Berkeley) (see, for example, [4, 12, 13]). It has been shown, that for the synchronous inequality of FSM languages, the most general solution is a nondeterministic FSM containing a behavior of each deterministic solution. The most general solution can be obtained using a number of methods such as fixed-point computation, WS1S logic, E-machine derivation, and simulation relation (see, for example, [4, 14]). The results were shown to be useful for the optimization of a component implementation in a logic network, the design of a discrete controller, and in finding the winning strategy of discrete games. The same results were obtained independently by the scientific groups of Yevtushenko and Petrenko for testing a component FSM of a complex system. In this case, the most general solution describes a behavior of the component that cannot be tested, due to lack of observability or controllability. Parallel equations over finite automata have been studied in process algebra. The solvability of a parallel language equation over finite automata where each state is accepting was established by Merlin and Bochmann [15]. The algorithm for deriving the largest solution for this case was proposed by Qin and Lewis [16]. Later, the results were extended by Yevtushenko and Petrenko and other researchers to solve parallel FSM equations [9, 10, 17].

Afterwards, the teams from Berkeley and Tomsk, led respectively by Brayton and Yevtushenko, together with Petrenko from Canada and Villa from Italy, became aware of the similarities of the results that they had obtained independently To unify their results in a common frame, they proposed to model the problem as solving equations over languages, specialized over different types of languages and composition operators, according to different applications. The results of these investigations were presented through a sequence of papers [1, 2, 6, 8, 18–20] that dealt especially with language equations for synchronous and parallel composition operators, and applied them to regular and FSM languages. They obtained the most general solutions for the language inequalities and language equations of the form $A \bullet X \subseteq C$ and $A \diamondsuit X \subseteq C$, for various composition topologies. Moreover, restricted solutions of theoretical or practical interest were characterized. The research is in progress for compositionally progressive solutions, to obtain algorithms for deriving the largest compositionally progressive solutions for synchronous and parallel regular (FSM) language equations (if such a solution exists) [8, 9].

A related important research effort has been conducted in the supervisory control community to study the problem of synthesizing discrete controllers (see, for example, [21–25]). However, methods for language equation solving cannot be directly used in supervisory control, and vice versa. Nevertheless, it is very interesting to establish links between the supervisory control and equation solving. The algorithm for solving equations over regular languages have an exponential complexity in the worst-case. The reason is the complement operator over finite automata, which requires a deterministic automaton as an input. The algorithm in supervisory control applied to regular languages has a polynomial complexity, even for partial controllability and observability. Therefore, on the one hand, the equation solving approach seems to be more general as it can deal with arbitrary topologies. On the other hand, the supervisory control approach has a low worst-case complexity, and it helps to characterize the topologies for equation solving whose computational complexity is lower than in general case. Such topologies are partly described in the papers [12, 26]. However, the complete characterization of such topologies is still an open issue.

6 Conclusion

The problem of solving language equations over synchronous and parallel composition operators has been studied. The most general (largest) solutions and the largest alphabets over which a solution exists were characterized. It has been shown that equations over regular languages can be effectively solved by computations over finite automata. Restricted solutions to equations were discussed. The language approach seems to unify the previously reported techniques for solving equations in logic synthesis and process algebra, and it appears to be capable of modeling problems with various notions of composition operators and language acceptance conditions, such as Petri nets, ω-languages, etc. However, procedures for solving effectively equations over Petri net languages and ω-languages must face the problem that generally these languages are not closed under complementation.

We notice that synchronous and parallel composition operators can be easily extended to be defined over more than two languages, and thus, the same approach can be applied for solving multi component language equations [27] and systems of language equations [28–30].

7 Acknowledgments

The first and the third authors gratefully acknowledge the support of grants by the Russian Found of Basic Research, in particular, RFBR-NSC Grant 06-08-89500. The second author gratefully acknowledges the support of the project FP6-2005-IST-5-033709 (VERTIGO). All the authors gratefully acknowledge the support of NATO Collaborative Linkage Grants No 971217, 979698 and 982314.

References

1. N. Yevtushenko, T. Villa. A. Petrenko, R.K. Brayton, A. Sangiovanni-Vincentelli. Solving equations in logic synthesis. – Tomsk, Spectrum Publishers, 1999. – 27 p. (In Russian).
2. N. Yevtushenko, T. Villa. R.K. Brayton, A. Petrenko, A. Sangiovanni-Vincentelli. Logic synthesis by equation solving // Proceedings of XVI Intern. Workshop on Logic synthesis, USA, 2000, p. 11-14.
3. J. Hartmanis, R.E. Stearns. The algebraic structure theory of sequential machines. Prentice Hall, N.Y., 1966.
4. T. Kam, T. Villa, R. Brayton, A. Sangiovanni-Vincentelli. Synthesis of finite state machines: functional optimization. Kluwer Academic Publishers, 1997.
5. J.E. Hopcroft, J.D. Ullman. Introduction to automata theory, languages and computations. Addison-Wesley Publishing Company, 1979.
6. N. Yevtushenko, T. Villa, R. Brayton, A. Petrenko, and A. Sangiovanni-Vincentelli. Solution of synchronous language equations for logic synthesis. Vestnik TSU, 2002, N 1, pp. 132-138.
7. N. Yevtushenko, T. Villa, R.K. Brayton, A. Mishchenko, and A.L. Sangiovanni-Vincentelli. Sequential synthesis by language equation solving. Technical report, Tech.Rep. No. UCB/ERL M03/9, Berkeley, CA, April 2003, `http://www.parades.rm.cnr.it/villa/articoli/ps/TR-UCB_ERL-M03.9.ps`.
8. N. Yevtushenko, T. Villa, R. Brayton, A. Petrenko, and A. Sangiovanni-Vincentelli. Compositionally progressive solutions of synchronous language equations, In: Proceedings of IWLS, 2003, pp. 148-155
9. K. El-Fakih, N. Yevtushenko, S. Buffalov, G. v. Bochmann. Progressive solutions to a parallel automata equation. Theoretical Computer Science, 362, 2006, pp. 17-32 (The preliminary version was published in Lectures Notes in Computer Science, 2003, vol. 2767, pp. 367-382).
10. A. Petrenko, N. Yevtushenko. Solving asynchronous equations. In: Formal Description Techniques and Protocol Specifications, Testing and Verification, FORTE X1/PSTVXIII, 1998, pp. 231-247.
11. J. Kim and M.M. Newborn. The specification of sequential machines with input restrictions. IRE Trans. on Electronic Computers. 1972, pp. 1440-1443.
12. Di Benedetto, A.L. Sangiovanni-Vincentelli, T. Villa. Model matching of finite state machines, IEEE Transactions on Automatic Control. 2001, 46, N.11, pp. 1726-1743.
13. S. Hassoun and T. Villa Optimization of synchronous circuits. In: Logic Synthesis and Verification (eds., R. Brayton, S. Hassoun and T. Sasao), Kluwer Academic Publishers, 2002, pp. 225-253.
14. E. Wolf. Hierarchical models of synchronous circuits for formal verification and substitution. PhD thesis, 1995, Stanford University.
15. P. Merlin, G.v. Bochmann. On the construction of submodule specification and communication protocols. ACM Transactions on Programming Languages and Systems, 1983, 5(1), pp. 1-25.
16. H. Qin, P. Lewis. Factorization of finite state machines under strong and observational equivalence. Formal Aspects of Computing, 1991, 3, pp. 284-307.
17. S. Buffalov, N. Yevtushenko. Studying solutions to a parallel FSM equation. Bulletin of the Novosibirsk Computing Center, Computer Science, V.11, 2001, pp. 7-16.
18. N. Yevtushenko, T. Villa. R.K. Brayton, A. Petrenko, A. Sangiovanni-Vincentelli. Solving a parallel language equation // In: Proceedings of the ICCAD'01, USA, 2001. pp. 103-110.

19. N. Yevtushenko, T. Villa, R. K. Brayton, A. Mishchenko, and A. L. Sangiovanni-Vincentelli. Composition operators in language equations. In: Proceedings of IWLS 2004, pp. 409-415.
20. T. Villa, N. Yevtushenko, S. Zharikova. Characterization of progressive solutions to a synchronous FSM equation. Vestnik TSU, N 278, 2003, pp. 129-133 (in Russian).
21. G. Barrett, S. Lafortune. Bisimulation, the strong supervisory control problem and strong model matching for finite state machines. Discrete Event Dynamic Systems: Theory and Applications. 1998, 8(4), pp. 377-429.
22. S. Cassandras, S. Lafortune. Introduction to discrete event systems. Kluwer Academic Publishers. 1999.
23. R. Kumar, S. Nelvagal, S.I. Marcus. Approach for protocol conversion. Discrete Event Systems: Theory and Applications. 1997, 7(3), pp. 295-315.
24. R. Kumar, V.K. Garg. Modeling and control of logical discrete event systems. Kluwer Academic Publishers, 1995.
25. P. Ramadge, W. Wonham. The control of discrete event systems. Proceedings of IEEE, 77(1), 1989, pp. 81-98.
26. N. Yevtushenko, T. Villa, R. Brayton, A. Petrenko, and A. Sangiovanni-Vincentelli, Equisolvability of Series vs. Controller's Topology in Synchronous Language Equations. Intern conf. DATE, 2003, pp. 11154-11155.
27. N. Spitsyna. Deriving test suites for testing communication Finite State Machines. PhD thesis, Tomsk State University, 2005 (in Russian).
28. S. Zharikova. Optimizing digital circuits by solving systems of equations. Vestnik TSU, 2003, N 1, pp. 255–259 (in Russian).
29. N. Yevtushenko, S. Zharikova, M. Vetrova. Multi Component Digital Circuit Optimization by Solving FSM Equations. Euromicro Symposium on Digital System Design, IEEE Computer society, 2003, pp. 62-68.
30. S. Zharikova, N. Yevtushenko, M. Vetrova. FSM decomposition optimization by FSM equation solving. In: Proceedings of XV Intern. Conf. Design and complexity of control systems, 2004, pp. 29-34 (in Russian).

Conjugacy of finite biprefix codes*

Julien Cassaigne[1], Juhani Karhumäki[2], and Petri Salmela[2]

[1] Institut de Mathématiques de Luminy — CNRS/FRUMAM, Case 907,
FR-13288 Marseille Cedex 9, France;
cassaigne@iml.univ-mrs.fr
[2] Department of Mathematics and TUCS,
University of Turku, FI-20014 University of Turku, Finland;
{karhumak,pesasa}@utu.fi

Abstract. Two languages X and Y are called *conjugates*, if they satisfy the conjugacy equation $XZ = ZY$ for some non-empty language Z. We will compare solutions of this equation with those of the corresponding equation of words and study the case of finite biprefix codes X and Y. We show that the maximal Z in this case is rational. We will also characterize X and Y in the case where they are both finite biprefix codes. This yields decidability of conjugacy of two finite biprefix codes.

1 Introduction

The conjugacy equation $xz = zy$ is a basic equation for words. Words x and y are *conjugates*, i.e., they satisfy the conjugacy equation for some word z if and only if x and y have factorizations $x = pq$ and $y = qp$ with some words p and q, and then the above z can be expressed as $z = (pq)^i p$.

For languages we say that languages X and Y are conjugates, if they satisfy the conjugacy equation $XZ = ZY$ for some non-empty language Z. For empty set Z the conjugacy equation always holds. We also restrict our research on languages X and Y which do not include empty word and namely we study finite biprefix codes. We can also note, that not all biprefix codes X and Y are conjugates. For example with $X = \{a\}$ and $Y = \{b\}$ the conjugacy equation $aZ = Zb$ does not have non-empty solution Z. The conjugacy equation on languages is not equally easy to solve as the same equation on words. Solutions of conjugacy equation of words can be extended to languages simply by replacing words x, y, z, p and q by languages X, Y, Z, P and Q, but in several cases these are not all possible solutions. For example, as observed in [2], the solution $X = \{a, ab, abb, ba, babb\}$, $Y = \{a, ba, bba, bbba\}$, $Z = \{a, ba\}$ is not of this type. However, for some special classes of languages all solutions can be obtained essentially as solutions of the word case. This is the topic of this note.

In this paper we first define so-called *word type solutions* of conjugacy equation on languages. The solutions for words can be expressed as $x = (pq)^k, y =$

* Supported by the Academy of Finland under grant 203354 and Finnish Mathematical Society International Visitors Program

M. Kunc, A. Okhotin (Eds.): Theory and Applications of Language Equations
TUCS General Publication No 44, pp. 33–42, 2007.

$(qp)^k$ and $z = (pq)^i p$ with some integers i, k and primitive word pq. This formulation of solutions is equivalent to the one above. For language equations we refer to solutions of form

$$X = (PQ)^k, Y = (QP)^k \text{ and } Z = \bigcup_{i \in I} (PQ)^i P$$

with primitive languages PQ as word type solutions. This notion has been studied in [2] by Cassaigne, Karhumäki and Maňuch, however, our notation is a slight extension.

We prove our three results. First we define and study the greatest conjugator of X and Y, that is the greatest language Z (with respect to the subset relation) such that $XZ = ZY$. We show that for finite biprefix codes X and Y the conjugator is rational. This is done by proving that the conjugator is always of form X^*U for some finite language U.

After this we characterize finite biprefix codes X and Y satisfying the conjugacy equation with some language Z. We show that these languages can always be factorized as $X = UV$ and $Y = VU$ for some biprefix codes U and V. This is achieved by not so obvious combinatorial analysis.

Our third result proves that the conjugacy problem for finite biprefix codes, i.e., the problem, whether given finite biprefix codes X and Y are conjugates, is decidable. This is shown as corollary of the previous result and the fact that the set of all biprefix codes is the free monoid. In the case of arbitrary finite language the problem is open, and does not seem to be easy.

2 Preliminaries

Let A be a finite alphabet, and A^* the free monoid generated by A. Lowercase letters are used to denote words, i.e., elements of A^*, and uppercase letters languages, i.e., subsets of A^*. The empty word will be denoted by 1. For words notation $|w|$ means the length of word w and for languages $|X|$ is the cardinality of X. Language is *uniform*, if all its elements have the same length.

Notation $\mathrm{Pref}(X)$ is used for the set of all prefixes of words in X, and similarly $\mathrm{Suf}(X)$ means all suffixes of words in X. Empty word and words in X are included. We use also a shorthand L^I for the union of powers $\bigcup_{i \in I} L^i$. Notation $L^{\leq n}$ is a shorthand for $\bigcup_{0 \leq i \leq n} L^i$. The language L is called *primitive*, if $L = K^i$ implies $L = K$ and $i = 1$, i.e., if the language L is not a proper power of any other language. If the language is not primitive it is *imprimitive*.

When we say that an element w in language L is *prefix (resp. suffix) incomparable*, we mean that neither w is a prefix (resp. suffix) of any other word in L nor any other word in L is a prefix (resp. suffix) of w. Sometimes this kind of element is also called *left (resp. right) singular* in L. (see [5], [10] or [8]) The language L is a prefix (resp. suffix) code or just prefix (resp. suffix), if all elements in L are left (resp. right) singular.

If the language L is both prefix and suffix code, we say it is biprefix code or just biprefix. It is also known, that the families of prefix, suffix and biprefix codes

are free monoids [1], [9]. This means that each prefix (resp. suffix or biprefix) code has unique factorization as catenation of indecomposable prefix (resp. suffix or biprefix) codes. This means that each prefix (resp. suffix or biprefix) set can be viewed as a word over a special alphabet. The free base of each of these monoids is infinite, but in many cases only finite subset is needed.

We also note that for the conjugacy equation $XZ = ZY$ we always have $Z \subseteq \mathrm{Pref}(X^*) \cap \mathrm{Suf}(Y^*)$. This is clear, since obviously also $X^n Z = ZY^n$ for any integer n and, for any words $z \in Z$ and $y \in Y$ there exists words $x_i \in X$ and $z' \in Z$ such that $zy^{|z|} = x_1 \cdots x_{|z|} z'$. This means, since $|z| < |x_1| + \cdots + |x_{|z|}|$, that z is a prefix of $x_1 \cdots x_{|z|} \in X^{|z|}$, i.e., $z \in \mathrm{Pref}(X^*)$. Dually, z is also suffix of some word in Y^*.

3 Word type solutions

We recall that the conjugacy equation $xz = zy$ for non-empty words has the general solution

$$xz = zy \iff \exists p, q \in \Sigma^* \text{ s.t. } x = pq, y = qp \text{ and } z \in (pq)^* p. \tag{1}$$

This motivates the notion of *word type solution* of conjugacy equation of the languages. In [2] this has been straightforwardly defined as:

$$X = PQ, Y = QP \text{ and } Z = (PQ)^I P \qquad (I \subseteq \mathbb{N}). \tag{2}$$

We call these solutions *word type 1* solutions.

However, there is also a slightly more general way to define word type solutions. The condition (1), in the case of words, is equivalent to the condition

$$xz = zy \iff \exists p, q \in \Sigma^* \text{ s.t. } x = (pq)^k, y = (qp)^k \text{ and } z \in (pq)^* p. \tag{3}$$

This motivates to define, word type solution of languages as:

$$X = (PQ)^k, Y = (QP)^k \text{ and } Z = (PQ)^I P \qquad (I \subseteq \mathbb{N}). \tag{4}$$

We call such solutions *word type 2* solutions, clearly they include all word type 1 solutions.

These indeed are different notions as shown in the next example.

Example 1. Let $X = BCBC$ and $Y = CBCB$ for $B = \{b\}$ and $C = \{c\}$ (or some other biprefix codes). Now both solutions

$$\begin{aligned} P_1 = B, \quad Q_1 = CBC, \\ X = P_1 Q_1, \quad Y = Q_1 P_1, \quad Z_1 = P_1 Q_1 P_1 = (BCBC)B \end{aligned}$$

and

$$\begin{aligned} P_2 = BCB, \quad Q_2 = C, \\ X = P_2 Q_2, \quad Y = Q_2 P_2, \quad Z_2 = P_2 Q_2 P_2 = (BCBC)BCB \end{aligned}$$

are of word type in the sense of (2), but their union $Z_1 \cup Z_2 = BCBCB \cup BCBCBCB$ is not. However, if we would use (4) as the definition of word type solution, we would have

$$P = B, Q = C, X = (PQ)^2, Y = (QP)^2, Z_1 = (PQ)^2P = (BC)^2B,$$

$$P = B, Q = C, X = (PQ)^2, Y = (QP)^2, Z_2 = (PQ)^3P = (BC)^3B$$

and

$$Z = Z_1 \cup Z_2 = (PQ)^{\{2,3\}}P.$$

We choose (4) as our definition of word type conjugation of languages.

4 The maximal conjugator

For the commutation equation $XY = YX$ there has been active research on the centralizer, that is on the greatest language commuting with given language X. J.H. Conway asked in [4], whether the centralizer of given rational language is rational as well. This so-called *Conway's problem* has been solved negatively in general [7], but has proven to have positive answers in several special cases.

For the conjugacy equation $XZ = ZY$ we can similarly study the maximal solution Z for given languages X and Y. The maximal solution exists and is the unique greatest one. In the case that X and Y are not conjugates the maximal (and only) solution is the empty set. If X and Y are conjugates, and conjugated via all Z_i for i in some index set I, then they are, by the distributivity of catenation and union operations, conjugated also via the union $\bigcup_{i \in I} Z_i$. Hence the unique maximal solution is the union of all solutions Z. The special case where $X = Y$ gives us the centralizer of X.

We can ask the question similar to the Conway's problem, namely whether the maximal conjugator of given languages X and Y is rational. The general answer is of course negative, since the original Conway's problem has a negative answer. However, we can again study some special cases. In what follows we use similar reasoning for conjugacy as has been used for commutation in [6]. First we need the following lemma.

Lemma 1 (Interchange lemma). *If X and Y are finite languages, such that Y has a suffix incomparable element y and $XZ = ZY$ for some language Z, then for each word $z \in Z$ there exists an integer n and a word $u \in \mathrm{Pref}(X) \setminus X$ such that $z = x_1x_2 \cdots x_nu$ for some $x_i \in X$, and moreover $X^nu \subseteq Z$.*

Proof. Let X and Y be finite languages, y a suffix incomparable element in Y, and Z such that $XZ = ZY$. Then for each $z \in Z$ there exists an integer n and factorization $z = x_1x_2 \cdots x_nu$ such that $x_i \in X$, $u \in \mathrm{Pref}(X) \setminus X$ and

$$zy^n = x_1x_2 \cdots x_nuy^n \in ZY^n = X^nZ$$

with $uy^n \in Z$. Then again

$$x'_1x'_2 \cdots x'_nuy^n \in X^nZ = ZY^n$$

where x'_i are arbitrary elements from X. This shows that $X^n u \subseteq Z$, since y is suffix incomparable in Y.

Theorem 1. *For finite languages X and Y, such that Y has suffix incomparable element y, the maximal conjugator is rational.*

Proof. Let X and Y be finite languages, y a suffix incomparable element in Y, and Z their greatest conjugator. By Lemma 1 for each word $z \in Z$ we have $z \in X^n u \subseteq Z$ for some integer n and word $u \in \mathrm{Pref}(X)$. Since $X^2 Z = XZY$ the language XZ is included in the greatest conjugator Z. Hence also $X^* Z \subseteq Z$ and $X^* X^n u \subseteq Z$.

Let $U \subseteq \mathrm{Pref}(X)$ be the set of all words u occurring in the above constructions. Since the language X is finite, so is U. Now, for each $u \in U$, there exists minimal integer n_u such that $X^* X^{n_u} u \subseteq Z$ and each word $z \in Z$ is in one of these sets. Hence we conclude that the maximal conjugator of X and Y is

$$Z = X^* \left(\bigcup_{u \in U} X^{n_u} u \right).$$

This set is rational, since the set $\cup_{u \in U} X^{n_u} u$ is finite. Note that if X and Y are not conjugates, then Z is the empty set.

The proof of previous Theorem is not constructive, since it needs the maximal conjugator to be given. Hence the result is noneffective.

In a suffix set all elements are suffix incomparable, therefore this result holds in the case of finite biprefix codes X and Y.

5 Characterization of conjugacy of finite biprefix codes

In this section we characterize, when biprefix codes X and Y are conjugates.

In what follows, we assume that X and Y are finite biprefix codes such that $XZ = ZY$ for some nonempty language Z.

Lemma 2. *We have: For every integer $n \geq \min\{|x| \mid x \in X\}$ there exist finite biprefix codes U_n and V_n satisfying*

$$\begin{aligned} X \cap A^{\leq n} &= U_n V_n \cap A^{\leq n} \quad \text{and} \\ Y \cap A^{\leq n} &= V_n U_n \cap A^{\leq n}. \end{aligned} \tag{5}$$

Proof. Let X_0, Y_0, Z_0 be the sets of elements in X, Y, Z of minimal length and $n_0 = \min\{|x| \mid x \in X\}$. Then, since X_0, Y_0 and Z_0 are uniform languages, $X_0 Z_0 = Z_0 Y_0$ holds and the solution is of word type, see [2]. This means that $X_0 = U_{n_0} V_{n_0}$, $Y_0 = V_{n_0} U_{n_0}$ and $Z_0 = (U_{n_0} V_{n_0})^m U_{n_0}$ for some unifor U_{n_0} and V_{n_0} and integer $m \geq 0$. Hence (5) holds for $n = n_0$.

Let us choose $u_0 \in U_{n_0}$, $v_0 \in V_{n_0}$ and $z_0 = (u_0 v_0)^m u_0 \in Z_0$. We assume that we have already constructed U_i and V_i for $n_0 \leq i < n$ and next we construct U_n and V_n for $n > n_0$, so that $U_{n-1} \subseteq U_n$ and $V_{n-1} \subseteq V_n$.

First we show that $U_{n-1}V_{n-1} \cap A^{\leq n} \subseteq X$ and $V_{n-1}U_{n-1} \cap A^{\leq n} \subseteq Y$. Let $u \in U_{n-1}$, $v \in V_{n-1}$ such that $|uv| = n$, if such elements exist. Then $|uv_0| < n$ and $|u_0v| < n$, so $uv_0, u_0v \in X$ and $v_0u, vu_0 \in Y$. Now $z_0v_0uvu_0 \in ZY^2 = X^2Z$ and by regrouping elements we have

$$z_0v_0uvu_0(v_0u_0)^m = (u_0v_0)^{m+1}uvz_0 \in ZX^{m+2} = X^{m+2}Z$$

and since X is biprefix, we get $uvz_0 \in XZ$. Hence $uvz_0 = xz$ with $x \in X$ and $z \in Z$. Here $|z| \geq |z_0|$, i.e., x is a prefix of $uv \in U_{n-1}V_{n-1}$. If $|x| < n$, i.e., x is a proper prefix of uv, then also $x \in U_{n-1}V_{n-1}$ and this is a contradiction, since $U_{n-1}V_{n-1}$ is a biprefix. Therefore $|x| = n$ and $x = uv \in X$. Similarly $vu \in Y$ and so $U_{n-1}V_{n-1} \cap A^{\leq n} \subseteq X$ and $V_{n-1}U_{n-1} \cap A^{\leq n} \subseteq Y$.

Next we deal with the words in $X \cap A^n \setminus U_{n-1}V_{n-1}$ (and in $X \cap A^n \setminus U_{n-1}V_{n-1}$) and show that some words can be added to U_{n-1} and V_{n-1} to form U_n and V_n.

If there exists $x \in X \cap A^n \setminus U_{n-1}V_{n-1}$, then

$$(u_0v_0)^{m+1}xz_0 = z_0v_0xu_0(v_0u_0)^m \in X^{m+2}Z = ZY^{m+2}$$

and hence Y is biprefix, $z_0v_0xu_0 \in ZY^2$. Therefore $Z_0v_0xu_0 = zyy'$ for some $y, y' \in Y$ and $z \in Z, |z| \geq |z_0|$. See the Figure 1 for illustration. Now yy' is suffix of v_0xu_0 and $|u_0| \leq n_0 \leq |y'| \leq |v_0xu_0| - |y| = n + n_0 - |y| \leq n$. So $y' = v'u_0$, where v' is a suffix of x. We have two cases:

- If $|y'| < n$, then $y' = v'u_0 \in V_{n-1}U_{n-1}$ and, since U_{n-1} is a biprefix, $v' \in V_{n-1}$. Now $x = u'v'$, where $u' \notin U_{n-1}$, and y is a suffix of v_0u'. For lengths we have now $n_0 \leq |y| \leq |v_0u'| = |v_0xu_0| - |y'| = n + n_0 - |y'| \leq n$. There are two subcases on the length of y:
  - If $|y| < n$, then $y = v''u'' \in V_{n-1}U_{n-1}$ for $v'' \in V_{n-1}, u'' \in U_{n-1}$. Now $|v''u''| \leq |v_0u'|$, since $y = v''u''$ is a suffix of v_0u', and also $|v''| \geq |v_0|$. Hence $|u''| \leq |u'|$ and u'' is a suffix of u'. In fact $u' \neq u''$, since $u' \notin U_{n-1}$ and $u'' \in U_{n-1}$. Now $u''v' \in U_{n-1}V_{n-1}$ and, as we just proved above, by its length
    $$|u''v'| \leq |v_0xu_0| - |v''| - |u_0| \leq n$$
    also $u''v' \in X$. This means that $u''v'$ and $x = u'v'$ are both in X and $u''v'$ is a proper suffix of $x = u'v'$. This contradicts the fact that X is a biprefix.
  - On the other hand, if $|y| = n = |x|$, then $|y'| = n_0$, $|z| = |z_0|$ and $y = v_0u'$. In this case we add u' to U_n, so that $x \in U_nV_{n_0}$.
- If $|y'| = n$, then $x = u'v'$ with $|u'| = |u_0|$ and $|y| = |v_0xu_0| - |y'| = n_0$, so $y = v_0u'$. Hence $y \in V_{n_0}U_{n_0}$ and so $u' \in U_{n_0}$. In this case we add v' to V_n so that $x \in U_{n_0}V_n$.

We proceed similarly for $y \in Y \cap A^n \setminus V_{n-1}U_{n-1}$. Note that by the construction of U_n and V_n $\max_{v \in V_n} |v| + \min_{u \in U_n} |u| \leq n$ and $\max_{u \in U_n} |u| + \min_{v \in V_n} |v| \leq n$.

Now for each element u in $U_n \setminus U_{n-1}$ there exist elements v' and v'' in V_{n_0} such that $uv' \in X \cap A^n$ and $v''u \in Y \cap A^n$. We have to show that $uV_0 \subseteq X$ and $V_0u \subseteq Y$.

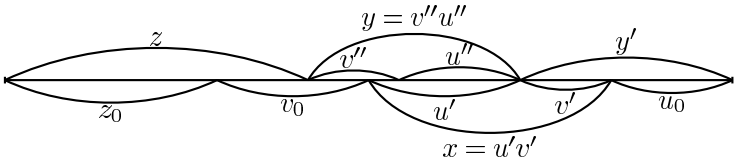


Fig. 1. Illustration of equation $z_0v_0xu_0 = zyy'$.

Let $v \in V_{n_0}$. Then $vu_0 \in Y$ and $u_0v \in X$. Since

$$(u_0v_0)^m u_0v''uvz_0 = z_0(v''u)(vu_0)(v_0u_0)^m \in ZY^{m+2} = X^{m+2}Z,$$

there is $u_0v''uvz_0 \in X^2Z$. Since $u_0v'' \in U_{n_0}V_{n_0} \subseteq X$ we obtain $uvz_0 \in XZ$, so $uvz_0 = xz$.

If $|x| < n = |uv|$, then $x \in U_{n-1}V_{n-1} \subseteq U_nV_n$ and x is proper prefix of $uv \in U_nV_n$. However, this can not be the case, since U_n and V_n are both biprefix codes (see below).

If $|x| > n$, then $|z| < |z_0|$ which contradicts the minimality of $|z_0|$. Hence $|x| = n = |uv|$ and $x = uv \in X$. V_{N_0} is obtained dually.

Similarly, for each element v in $V_n \setminus V_{n-1}$ there exist elements u' and u'' in U_0 such that $u'v \in X \cap A^n$ and $vu'' \in Y \cap A^n$ and we can prove that $U_{n_0}v \subseteq X$ and $vU_{n_0} \subseteq Y$.

U_n and V_n are biprefix codes: If $u' \in U_n$ is a proper prefix of $u \in U_n$, we can assume that $|u| = n - |v_0|$ (otherwise we are in U_{n-1}, which is a biprefix) and $u' \in U_{n-1}$. Then there exists such $v'' \in V_{n_0}$ that $v''u \in Y$, but then also $v''u' \in V_{n_0}U_{n-1} \subseteq Y$. Since Y is biprefix, we have a contradiction.

Similar reasoning applies also, if $u' \in U_n$ is a proper suffix of $u \in U_n$. Hence U_n is also a suffix code and therefore it is a biprefix.

Similarly V_n is a biprefix code.

Theorem 2. *If finite biprefix codes X and Y are conjugates, then $X = UV$ and $Y = VU$ for some biprefix codes U and V.*

Proof. Applying Lemma 2 for $n = \max_{x\in X}|x| + \max_{y\in Y}|y| - n_0$, we obtain:

$$\left.\begin{array}{l}\text{for all } u \in U_n, uv_0 \in X, \text{ so } |u| \le \max_{x\in X}|x| - |v_0| \\ \text{for all } v \in V_n, vu_0 \in Y, \text{ so } |v| \le \max_{y\in Y}|y| - |u_0|\end{array}\right\} \text{ so } |uv| \le n$$

Hence

$$\begin{aligned} U_nV_n \cap A^{\le n} &= U_nV_n \\ V_nU_n \cap A^{\le n} &= V_nU_n \\ X \cap A^{\le n} &= X \\ Y \cap A^{\le n} &= Y \end{aligned}$$

and we have $X = U_nV_n$ and $Y = V_nU_n$.

The theorem 2 deserves a few comments. It shows that if finite biprefixes X and Y are conjugates, that is satisfy the conjugacy equation $XZ = ZY$ with nonempty Z, they can be decomposed into the form

$$X = PQ \text{ and } Y = QP \text{ for some biprefixes } P \text{ and } Q.$$

Of course, the reverse holds as well, namely they satisfy the conjugacy equation, e.g., for $Z = P(QP)^I$, with $I \subseteq \mathbb{N}$. Hence the conjugacy in the case of finite biprefixes can be defined equivalently in the above two ways. In general, these definitions are not equivalent as discussed in [3].

To continue our analysis let us see what happens if the biprefixes X and Y have two different factorizations

$$X = UV, Y = VU \quad \text{and} \quad X = U'V', Y = V'U'.$$

This indeed is possible, if X and Y are not primitive, as pointed out in Example 1. We show that this is actually the only way this can happen. For this we need the following simple lemma on words.

Lemma 3. *All solutions of the pair of word equations*

$$\begin{cases} xy = uv \\ yx = vu \end{cases}$$

over some finite alphabet are of the form $x = \beta(\alpha\beta)^i$, $y = (\alpha\beta)^j\alpha$, $u = \beta(\alpha\beta)^k$ *and* $v = (\alpha\beta)^l\alpha$ *with* $i + j = k + l$.

Proof. If we assume that $|u| \leq |x|$, the first equation implies that for some word t

$$x = ut$$

and hence

$$v = ty \text{ and } yut = tyu.$$

The latter condition means that yu and t commute, i.e., we can write

$$t = (\alpha\beta)^f, \quad y = (\alpha\beta)^d\alpha, \text{ and } u = \beta(\alpha\beta)^e,$$

where $\alpha, \beta \in A^*$ and $d, e, f \geq 0$. This leads to the solutions

$$\begin{cases} x = \beta(\alpha\beta)^{e+f} \\ y = (\alpha\beta)^d\alpha \\ u = \beta(\alpha\beta)^e \\ u = (\alpha\beta)^{f+d}\alpha \end{cases}.$$

The case $|x| \leq |u|$ is symmetric and solutions are the same up to renaming of x, y, u and v.

Since biprefix codes can be viewed as words over the alphabet of all indecomposable biprefixes, we conclude from Theorem 2 and Lemma 3 the following theorem.

Theorem 3. *If finite biprefix codes X and Y are conjugates, then $X = (PQ)^i$ and $Y = (QP)^i$ for primitive languages PQ and QP and unique biprefix codes P and Q.*

Proof. The theorem 2 implies that X and Y have some factorization $X = UV$ and $Y = VU$ with biprefix codes U and V. If $X = UV = U'V'$ and $Y = VU = V'U'$ are two different such factorizations of X and Y, then we can simply apply the Lemma 3 for equations

$$\begin{cases} UV = U'V' \\ VU = V'U' \end{cases}.$$

Here biprefix codes are now viewed as words over the alphabet appropriate finite set of indecomposable biprefix codes. This gives that always $U = P(QP)^j$, $V = (QP)^k Q$, $U' = P(QP)^l$ and $V' = (QP)^m Q$ for some integers j, k, l and m. Then $X = (PQ)^i$ and $Y = (QP)^i$ for some integer i. Naturally P and Q can be chosen so that PQ and QP are primitive roots of X and Y respectively.

Hence all different factorizations $X = UV$, $Y = VU$ can be given in the form above as products of the same biprefix codes P and Q. then

Now, we are ready to conclude our remarks. If X and Y are finite biprefixes, which are conjugates, then there exist unique biprefixes P and Q such that $X = (PQ)^i$ and $Y = (QP)^i$. Hence X and Y are conjugates in the form of word type 2 as in formula 4. On the other hand the form of Z is still open and it is not necessarily always $Z = (PQ)^I P$. This needs to be examined more closely in the future.

6 Conjugacy problem for finite biprefix codes

We will refer to the problem *"Are given finite languages X and Y conjugates?"* as the *conjugacy problem.* In general, the decidability status of this problem is not known. Our results allow to answer it in the case of biprefix codes.

Theorem 4. *The conjugacy problem for finite biprefix codes is decidable.*

Proof. Let X and Y be finite biprefix codes. Languages X and Y have unique factorizations as catenation of indecomposable biprefix codes. These factorizations can be found for example by finding the minimal DFA for these biprefixes. Theorem 2 shows that if X and Y are conjugates, then $X = UV$ and $Y = VU$ for some biprefix codes U and V. Since the prime factorizations of X and Y are finite, there are only a finite number of candidates for U and V. If U and V can be found, then equation $XZ = ZY$ has at least word type solutions with given X and Y. If on the other hand, suitable U and V can not be found, then X and Y are not conjugates.

References

1. J. Berstel, D. Perrin: *Theory of Codes*, Academic Press, New York (1985).
2. J. Cassaigne, J. Karhumäki and J. Maňuch, On Conjugacy of Languages, *Theoret. Informatics Appl.* **35** (2001) 535–550.
3. Ch. Choffrut, Conjugacy in Free Inverse Monoids. *Proceedings of the Second International Workshop on Word Equations and Related Topics, LNCS 677*: 6–22 Springer-Verlag, London, UK (1991).
4. J.H. Conway, *Regular algebra and finite machines.* Chapman Hall (1971).
5. J. Karhumäki, M. Latteux, I. Petre, The commutation with codes. *Theor. Comp. Sci.* **340** (2005) 322–333.
6. J. Karhumäki, I. Petre, The Branching Point Approach to Conway's Problem, in W. Brauer et al. (Eds.): *Formal and Natural Computing, LNCS 2300*, 69–76, Springer-Verlag, Berlin Heidelberg (2002).
7. M. Kunc, The power of commuting with finite sets of words, in *Proc. of STACS 2005 , Lect. Notes in Comput. Sci.* **3404** (2005) 569–580.
8. P. Massazza, P. Salmela, On the simplest centralizer of a language, in *RAIRO – Theoret. Informatics Appl.* **40**, 295–301 (2006).
9. D. Perrin, Codes conjugués. *Inform. and Control* **20** (1972) 222 – 231.
10. B. Ratoandromanana, Codes et motifs, *RAIRO Inform. Theor.*, 23 (4), (1989) 425–444.

On the operational orthogonality of languages

Mark Daley[1], Michael Domaratzki[2] and Kai Salomaa[3]

[1] Department of Computer Science and Department of Biology, University of Western Ontario, London, Ontario N6A 5B7, Canada, daley@csd.uwo.ca
[2] Department of Computer Science, University of Manitoba, Winnipeg, Manitoba R3T 2N2, Canada, mdomarat@cs.umanitoba.ca
[3] School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada, ksalomaa@cs.queensu.ca

Abstract. Motivated by the applications of signal orthogonality in the Code Division Multiple Access (CDMA) multiplexing scheme, we introduce the generalized notion of orthogonality for languages relative to any binary word operation. We consider basic algebraic and decision-theoretic properties of orthogonal languages. We also consider the solvability of language equations involving the orthogonal concatenation operation.

1 Introduction

The Code Division Multiple Access (CDMA) multiplexing scheme used in radio communications allows for the simultaneous reception of transmissions from multiple senders by assigning each sender a waveform which, when superimposed with the waveforms of other senders, generates a signal which can be uniquely decomposed back in to the original waveforms. This is possible due to the additive nature of interfering radio waves and the careful choice of waveforms which are mutually orthogonal. In the case of binary signaling, we may represent waveforms as binary vectors of some finite size leading to a simple choice of those waveforms which are represented by orthogonal vectors in the sense of traditional linear algebra[4].

The theoretical and practical properties of CDMA have been investigated thoroughly in the engineering literature, including algebraic studies of CDMA variants (see, e.g., [2]). These studies, however, principally concern themselves with the details of radio communications, rather than abstract communications channels. In this paper we seek to generalize the notion of orthogonality to arbitrary word operations to perhaps facilitate the investigation of code-division multiplexing schemes in the more general context of abstract coding theory. Restrictions of operational orthogonality have been previously studied for the concatenation operation in the formal language theory literature on language equations. We present here a general framework for studying orthogonality in the light of previous related work and demonstrate preliminary results for the operations of concatenation and shuffle on trajectories.

[4] We refer the reader to [13] for an introductory text on CDMA.

M. Kunc, A. Okhotin (Eds.): Theory and Applications of Language Equations
TUCS General Publication No 44, pp. 43–53, 2007.

2 Preliminaries

In the following Σ is a finite alphabet. The set of all words over Σ is Σ^* and Σ^+ is the set of non-empty words over Σ. The length of a word $w \in \Sigma^*$ is $|w|$ and ε is the empty word. The set of symbols of Σ occurring in a word w is $\mathrm{alph}(w)$.

For $w \in \Sigma^*$ and $L \subseteq \Sigma^*$ we define

$$w^{-1}L = \{u \in \Sigma^* \mid wu \in L\}, \quad Lw^{-1} = \{u \in \Sigma^* \mid uw \in L\}.$$

The reversal of a word $w = a_1 \cdots a_n$, $a \in \Sigma$, $i = 1, \ldots, n$, is $w^R = a_n \cdots a_1$ and the reversal of a language L is $L^R = \{w^R \mid w \in L\}$.

Let $L \subseteq \Sigma^*$. The set of prefixes (respectively, proper prefixes) of words of L is denoted $\mathrm{pref}(L)$ (respectively, $\mathrm{ppref}(L)$). Similarly the sets of suffixes and proper suffixes of words of L are $\mathrm{suf}(L)$ and $\mathrm{psuf}(L)$.

A nondeterministic finite automaton (NFA) is a four-tuple

$$A = (\Sigma, Q, q_0, F, \delta),$$

where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of accepting states and $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ defines the transitions of A. The automaton A is deterministic (a DFA) if for all $q \in Q$ and $a \in \Sigma$, $|\delta(q, a)| \leq 1$.

In the standard way the transition function δ is extended to a function $Q \times \Sigma^* \to \mathcal{P}(Q)$ and the language accepted by A is

$$L(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \varnothing\}.$$

The nondeterministic and deterministic finite automata accept exactly the regular languages [15].

The notions associated with orthogonality have close connections to formal power series and, for example, the proof of the main decidability result for orthogonal catenation [1] relies on power series. Here we do not discuss power series in detail and the interested reader can find more information e.g. in [10].

To conclude this section we recall the notion of shuffle on trajectories operation originally introduced in [11].

A trajectory t is a word over the alphabet $\{0, 1\}$. For $x, y \in \Sigma^*$, the shuffle of x and y along a trajectory t, $x \sqcup_t y$, is defined inductively as follows.

If $x = ax'$, $y = by'$ (with $a, b \in \Sigma$) and $t = et'$ (with $e \in \{0, 1\}$), then

$$x \sqcup_{et'} y = \begin{cases} a(x' \sqcup_{t'} by') \text{ if } e = 0; \\ b(ax' \sqcup_{t'} y') \text{ if } e = 1. \end{cases}$$

If $x = ax'$ ($a \in \Sigma$), $y = \varepsilon$ and $t = et'$ ($e \in \{0, 1\}$), then

$$x \sqcup_{et'} \varepsilon = \begin{cases} a(x' \sqcup_{t'} \varepsilon) \text{ if } e = 0; \\ \varnothing \qquad\qquad\quad \text{otherwise.} \end{cases}$$

If $x = \varepsilon$, $y = by'$ $(b \in \Sigma)$ and $t = et'$ $(e \in \{0,1\})$, then

$$\varepsilon \sqcup\!\sqcup_{et'} y = \begin{cases} b(\varepsilon \sqcup\!\sqcup_{t'} y') \text{ if } e = 1; \\ \varnothing \qquad\qquad \text{otherwise.} \end{cases}$$

We let $x \sqcup\!\sqcup_{\varepsilon} y = \varnothing$ if $\{x, y\} \neq \{\varepsilon\}$. Finally, if $x = y = \varepsilon$, then $\varepsilon \sqcup\!\sqcup_{t} \varepsilon = \varepsilon$ if $t = \varepsilon$ and $\varnothing$ otherwise.

We extend shuffle on trajectories to *sets* $T \subseteq \{0,1\}^*$ *of trajectories* as follows:

$$x \sqcup\!\sqcup_T y = \bigcup_{t \in T} x \sqcup\!\sqcup_t y.$$

Further, for $L_1, L_2 \subseteq \Sigma^*$, we define

$$L_1 \sqcup\!\sqcup_T L_2 = \bigcup_{\substack{x \in L_1 \\ y \in L_2}} x \sqcup\!\sqcup_T y.$$

3 Definition and Basic Properties of Orthogonality

We begin by introducing the notion of operational orthogonality. By a binary[5] word operation on Σ^* we mean a function $\circ : (\Sigma^*)^2 \to 2^{\Sigma^*}$. The operation $\circ$ is extended to languages $L_1, L_2 \subseteq \Sigma^*$ as

$$L_1 \circ L_2 = \bigcup_{x \in L_1, y \in L_2} x \circ y.$$

Let L, L_1, L_2 be languages over Σ.

Definition 1. *We say that L is an* orthogonal $\circ$-composition *of L_1 and L_2, denoted*

$$L = L_1 \circ_\perp L_2,$$

if the following two conditions hold

(OR1) $L = L_1 \circ L_2$, *and*
(OR2) $(\forall u_i, v_i \in L_i, i = 1, 2)$ *if* $(u_1, u_2) \neq (v_1, v_2)$ *then* $u_1 \circ u_2 \cap v_1 \circ v_2 = \varnothing$.

Given languages L_1 and L_2, we define their orthogonal $\circ$-composition as

$$L_1 \circ_\perp L_2 = \begin{cases} L_1 \circ L_2 & \text{if condition (OR2) holds,} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

If $L_1 \circ_\perp L_2$ is defined, we say also that the languages L_1 and L_2 are $\circ$-orthogonal. Note that in the above statement the order of the languages is significant since the $\circ$-orthogonality relation is not symmetric.

For commutative word operations we can make the following observations:

[5] All below notions can be extended in an obvious way for n-ary operations, $n \geq 2$. For simplicity below we discuss only the binary case.

Lemma 1. *Let $\circ$ be a commutative binary word operation. Then L_1 and L_2 are $\circ$-orthogonal if and only if L_2 and L_1 are $\circ$-orthogonal.*

Proof. Let $\circ$ be a commutative binary word operation, and assume that L_1 and L_2 are $\circ$-orthogonal. Assume, contrary to what we want to prove, that L_2 and L_1 are not $\circ$-orthogonal. Then there exists $x \in L_2 \circ L_1$ with two factorizations $x \in y_1 \circ z_1$ and $x \in y_2 \circ z_2$, where $y_1, y_2 \in L_2$ and $z_1, z_2 \in L_1$. But now since, e.g., $y_1 \circ z_1 = z_1 \circ y_1$, we have contradicted that L_1 and L_2 are $\circ$-orthogonal. □

More generally, using the notation from Kari [7], let $\circ^R$ be the word operation defined by $x \circ^R y = y \circ x$. Then the following result is established

Lemma 2. *Let $\circ$ be a binary word operation. Then L_1 and L_2 are $\circ$-orthogonal if and only if L_2 and L_1 are $\circ^R$-orthogonal.*

We note that the converse of Lemma 1 does not hold. In particular, if $\circ$ is the binary word operation defined by $a \circ b = \{ab\}$ and $x \circ y = \varnothing$ for $(x, y) \neq (a, b)$, then the operation $\circ$ is not commutative, but any pair of languages L_1 and L_2 are $\circ$-orthogonal.

4 Concatenation Orthogonality

In the following we concentrate on the catenation operation. When we need to use notation for orthogonality of the operation, we denote catenation by $\odot$. When we do not need notation for orthogonality we denote, as usual, catenation of languages L_1 and L_2 simply as L_1L_2.

Let $L \subseteq \Sigma^*$ be a language. If L is a code then $L \odot_\perp L$ is defined, i.e., L is $\odot$-orthogonal with itself. However, the converse does not hold since

$$\{a, b, ab\} \odot_\perp \{a, b, ab\} = \{aa, ab, aab, ba, bb, bab, aba, abb, abab\},$$

and hence $\{a, b, ab\}$ is $\odot$-orthogonal with itself.

By definition, a sufficient condition for languages L and L' to be $\odot$-orthogonal is that

(i) L is a prefix code, or,
(ii) L' is a suffix code.

In particular, it follows that

$$L \text{ and } \Sigma^* \text{ are } \odot\text{-orthogonal iff } L \text{ is a prefix code,} \tag{1}$$

and,

$$\Sigma^* \text{ and } L \text{ are } \odot\text{-orthogonal iff } L \text{ is a suffix code.} \tag{2}$$

The below result showing that the operation $\odot_\perp$ is associative for non-empty languages follows easily from the corresponding well known property of formal power series.

Lemma 3. *Let L_i, $i = 1, 2, 3$, be non-empty languages. Then*

$$(L_1 \odot_\perp L_2) \odot_\perp L_3 = L_1 \odot_\perp (L_2 \odot_\perp L_3).$$

The above equality includes the condition that the left side is defined iff the right side is defined.

Lemma 3 needs the assumption on the non-emptiness of the languages. For example, if $L = \{a, aa\}$ then

$$(\varnothing \odot_\perp L) \odot_\perp L = \varnothing$$

but $\varnothing \odot_\perp (L \odot_\perp L)$ is undefined. In the statement of Lemma 3 it would be sufficient to require that only L_1 and L_3 are non-empty.

A natural question will be for which languages we can decide whether or not the languages are concatenation-orthogonal.

Proposition 1. *Given regular languages L_1 and L_2, it is decidable whether or not L_1 and L_2 are $\odot$-orthogonal.*

Proof. The result is well-known and follows also from Theorem 5 below since catenation can be expressed as shuffle along the regular set of trajectories 0^*1^*. □

As can be expected, the result of Theorem 1 cannot be extended for context-free languages, not even in the case where only one of the languages L_1 or L_2 is context-free. Since it is undecidable whether or not a given linear context-free language is a prefix- (respectively, a suffix-) code [6], the observations (1) and (2) give the following undecidability result.

Theorem 1. *Given a linear language L and a regular language R it is undecidable whether or not*

(i) *L and R are $\odot$-orthogonal,*
(ii) *R and L are $\odot$-orthogonal.*

If L_1 and L_2 are languages over a unary alphabet $\{a\}$, L_1 and L_2 cannot be $\odot$-orthogonal whenever they contain any two words in common. This relates to the commutativity of unary concatenation. For commutative operations it may be appropriate to consider a somewhat different definition of orthogonality. We hope to return to this topic in later work.

4.1 Language equations

Here we consider language equations involving the orthogonal catenation operation. The result of Proposition 1 means that given regular languages L, L_1 and L_2 we can effectively decide whether or not the equation

$$L = L_1 \odot_\perp L_2$$

having no variables holds.

Next we consider one-variable equations

$$L = L_1 \odot_\perp X, \tag{3}$$

or its symmetric variant

$$L = X \odot_\perp L_1, \tag{4}$$

and two variable equations

$$L = X \odot_\perp Y. \tag{5}$$

Recall that the orthogonality property is not symmetric. However, we can make the observation that if L_1 and L_2 are $\odot$-orthogonal then L_2^R and L_1^R are $\odot$-orthogonal and

$$(L_1 \odot_\perp L_2)^R = L_2^R \odot_\perp L_1^R.$$

Thus an equation $L = X \odot_\perp L_1$ has a solution X_0 if and only if the equation $L^R = L_1^R \odot_\perp X$ has a solution X_0^R. This means, in particular, that when the language constants are regular (or restricted to any language family that is closed under reversal) an equation (4) can always be reduced to an equation (3). In the following we restrict one-variable equations to be of type (3).

A two-variable equation (5) has always the solution

$$L = L \odot_\perp \{\varepsilon\} = \{\varepsilon\} \odot_\perp L.$$

We say that these solutions are *trivial solutions.*

Without loss of generality we can assume that all solutions to equations (3) or (5) must be over an alphabet Σ where Σ contains all symbols occurring in words of L. The alphabet Σ is usually not mentioned separately.

The following strong result is established by Anselmo and Restivo [1].

Theorem 2. [1] *For regular languages L and L_1 it is decidable whether or not an equation (3) has a solution. A possible solution is regular and can be effectively constructed.*

The solutions for one-variable language equations $L = L_1X$ obviously do not need to be unique if we do not impose the condition of orthogonality. The proof of Theorem 2 in [1] implies that if (3) has a solution, it is unique. The proof given in [1] uses formal power series and below we give an elementary proof of the fact that one variable equations with orthogonal catenation have a unique solution, even without assuming regularity of the languages L and L_1.

Lemma 4. *Let L and L_1 be nonempty languages. If an equation $L = L_1 \odot_\perp X$ has a solution for the variable X, the solution is unique.*

Proof. For the sake of contradiction assume that there exist languages $L_2 \neq L_2'$ such that

$$L = L_1 \odot_\perp L_2 = L_1 \odot_\perp L_2'. \tag{6}$$

Let v_2 be a word of minimal length in the symmetric difference of L_2 and L_2'. Without loss of generality we can assume that $v_2 \in L_2 - L_2'$. Let u_1 be a word of minimal length in L_1. Now (6) implies that there exist $u_1' \in L_1$ and $v_2' \in L_2'$ such that

$$u_1 v_2 = u_1' v_2'. \tag{7}$$

If $v_2' \in L_2$, then the above equation would violate the $\odot$-orthogonality of L_1 and L_2, and we can conclude that $v_2' \in L_2' - L_2$.

In particular, $v_2' \neq v_2$ and since v_2 is of minimal length in the symmetric difference of L_2 and L_2' it follows that $|v_2'| > |v_2|$. Now by (7), $|u_1'| < |u_1|$ which contradicts the assumption that u_1 is of minimal length in L_1. □

It is known that any one-variable equation involving (ordinary) catenation

$$L = L_1 X \tag{8}$$

has a minimal solution [9]. As a consequence of Theorem 2 or Lemma 4 we observe that any solution to (3) has to be a minimal solution to the corresponding equation $L = L_1 X$ involving ordinary catenation.

The analogy of Lemma 4 does not hold for two-variable equations (5). For example, by considering decompositions of individual words it is easy to see that solutions to (5) need not be unique.

For two-variable equations involving ordinary catenation

$$L = XY, \tag{9}$$

where L is regular, the existence of non-trivial solutions is decidable [3, 9, 12].

For a given regular language L it is possible that the equation (9) has a non-trivial solution but the corresponding two-variable equation with orthogonal catenation (5) has only trivial solutions. As an example we can consider the finite language $\{\varepsilon, a, a^2\} = \{1, a\} \cdot \{1, a\}$ where catenation is clearly non-orthogonal.

While any solution to (3) has to be regular, assuming that L and L_1 are regular, it is noted in [1] that a regular language can be the orthogonal catenation of two non-regular languages. For example,

$$a^* = \mathit{\Pi}_{i=0}^{\infty}(\varepsilon + a^{2^{2i}}) \odot_{\perp} \mathit{\Pi}_{i=0}^{\infty}(\varepsilon + a^{2^{2i+1}}). \tag{10}$$

The unary language a^* has different orthogonal decompositions into regular components, however, in this case one of the components has to be finite.

Lemma 5. *Let L be a regular unary language. If (5) has a solution (L_X, L_Y) where L_X and L_Y are regular, then one of the languages L_X or L_Y has to be finite.*

Proof. If L_X and L_Y are infinite and regular, there exist $m_1, m_2 \geq 0$ and $n_1, n_2 \geq 1$ such that $a^{m_1}(a^{n_1})^* \subseteq L_X$ and $a^{m_2}(a^{n_2})^* \subseteq L_Y$. This means that the word $a^{m_1+m_2+n_1n_2}$ has two different decompositions into words of L_X and L_Y, and the languages are not $\odot$-orthogonal. □

From Theorem 2 it follows that in any solution for (5), where L is regular, if the language for one of the variables X or Y is regular, also the language for the other variable has to be regular. This property clearly does not hold for solutions of equation (9) involving ordinary catenation.

Open problem 3. (i) *Is it possible, for a regular language L, that (5) has non-regular solutions for X and Y but no non-trivial regular solution?*
(ii) *Given a regular language L is it decidable whether or not (5) has a non-trivial solution (respectively, a non-trivial regular solution)?*

To conclude this section we show that existence of solutions for two-variable equations is undecidable when the constant language is context-free.

Theorem 4. *Given a linear context-free language L it is undecidable whether or not the equation (5) has a non-trivial solution.*

Proof. [6] Let $I_{PCP} = (u_1, \ldots, u_m; v_1, \ldots, v_m)$, $u_i, v_i \in \{a, b\}^+$, $i = 1, \ldots, m$, $m \geq 1$, be an arbitrary instance of the Post correspondence problem (PCP). Without loss of generality we assume that

$$u_1 \neq v_1. \tag{11}$$

We denote

$$I'_{PCP} = (u_1, \ldots, u_m, u_{m+1}; v_1, \ldots, v_m, v_{m+1}) \text{ where } u_{m+1} = c,\ v_{m+1} = cc.$$

Now I'_{PCP} is a PCP instance over the alphabet $\{a, b, c\}$. The new elements u_{m+1} and v_{m+1} have been added only for technical reasons and they clearly cannot be used in any solution for I'_{PCP} (because they have different length and no other words of I'_{PCP} contain occurrences of c). Thus

the instance I'_{PCP} has a solution iff the instance I_{PCP} has a solution. (12)

We define

$$\Omega = \{1, \ldots, m, m+1, \#, a, b, c\} \text{ and } \Sigma = \Omega \cup \{\$\}.$$

For notational convenience, the alphabet Σ is allowed to depend on the given PCP instance. Everything below works if we code the symbols $1, \ldots, m+1$ over a fixed alphabet where the coding is chosen so that the first symbol of the encoding of $m+1$ is distinct from the first symbol of the encoding of 1.

We define a linear context-free language $L \subseteq \Sigma^*$:

$$\begin{aligned} L = {} & \{i_k \cdots i_1 \# u_{i_1} \cdots u_{i_k} \mid k \geq 1, 1 \leq i_j \leq m+1, j = 1, \ldots, k\} \cdot \{\varepsilon, \$\} \\ & \cup \{i_k \cdots i_1 \# v_{i_1} \cdots v_{i_k} \mid k \geq 1, 1 \leq i_j \leq m+1, j = 1, \ldots, k\} \cdot \{\$, \$\$\}. \end{aligned}$$

First we establish some properties of any decomposition of L as a non-trivial catenation of two languages (without imposing any orthogonality condition). Then we show that (5) has a non-trivial solution if and only if the PCP instance I'_{PCP} does not have a solution.

[6] One of the referees has pointed out that Theorem 4 has a shorter proof that uses a reduction to the equivalence problem of linear languages.

Claim. If we can write

$$L = L_1 \cdot L_2, \tag{13}$$

where $L_1 \neq \{\varepsilon\}$, then $L_2 \subseteq \{\varepsilon, \$, \$\$\}$.

Proof. If the claim does not hold, then (since the symbols \$ occur only at the end of words of L) the language L_2 contains a word $y \in \Omega^+ \cdot \{\varepsilon, \$, \$\$\}$. If $y \in L$, then there cannot exist any nonempty word w such that $wy \in L$. In other words, if $\varepsilon \in L_1$ then $L_1 = \{\varepsilon\}$ which is a contradiction.

Hence we can conclude that $\varepsilon \notin L_1$ and (13) implies that there exist $z_1, z_2 \in \Sigma^*$ such that $1 \cdot z_1 \in L_1$ and $(m+1) \cdot z_2 \in L_1$. However, only one of the words $1 \cdot z_1 \cdot y$ and $(m+1) \cdot z_2 \cdot y$ can be in L because u_{m+1}, v_{m+1} end with c and u_1, v_1 do not end with c.

This concludes the proof of the claim. □

Claim. If we can write $L = L_1 \cdot L_2$ where $L_1 \neq \{\varepsilon\}$, then $\$\$ \notin L_2$.

Proof. By Claim 4.1, we know that $L_2 \subseteq \$^*$. Since $1\#u_1 \in L$, it has to be the case that $1\#u_1 \in L_1$. Now if $\$\$ \in L_2$, then $1\#u_1\$\$ \in L_1 \cdot L_2$ but $1\#u_1\$\$ \notin L$ by the definition of L and (11). □

Now we continue to show that I'_{PCP} has a solution if and only if the equation (5) does not have any non-trivial solution.

First assume that $i_1, \ldots, i_k$ is a solution for I'_{PCP}. Let L_1 and L_2 be an arbitrary non-trivial solution for X and Y in (5). By Claim 4.1 and Claim 4.1 we know that the only possibilities are that $L_2 = \{\$\}$ or $L_2 = \{\varepsilon, \$\}$. The first case is impossible, since there exist words in L which do not end with \$. Thus, we must have $L_2 = \{\varepsilon, \$\}$.

Since $w_1 = i_k \cdots i_1 \# u_{i_1} \cdots u_{i_k} \in L$, it follows that $w_1 \in L_1$. Since

$$i_k \cdots i_1 \# v_{i_1} \cdots v_{i_k} \$\$ \in L,$$

it must be the case that $w_2 = i_k \cdots i_1 \# v_{i_1} \cdots v_{i_k} \$ \in L_1$. (Note that the word $i_k \cdots i_1 \# v_{i_1} \cdots v_{i_k} \$\$$ cannot be in L_1 since $L_1 \$ \subseteq L$.) Now the word $w_1 \cdot \$ = w_2 \cdot \varepsilon$ has two different decompositions and the languages L_1 and L_2 cannot be $\odot$-orthogonal.

Second, assume that I'_{PCP} does not have a solution. We define

$$\begin{aligned} L_1 = {} & \{i_k \cdots i_1 \# u_{i_1} \cdots u_{i_k} \mid k \geq 1, 1 \leq i_j \leq m+1, j = 1, \ldots, k\} \\ & \cup \{i_k \cdots i_1 \# v_{i_1} \cdots v_{i_k} \mid k \geq 1, 1 \leq i_j \leq m+1, j = 1, \ldots, k\} \cdot \{\$\}. \end{aligned}$$

We have $L = L_1 \cdot \{\varepsilon, \$\}$ and we verify that L_1 and $\{\varepsilon, \$\}$ are $\odot$-orthogonal. If this were not the case, there must exist $r, s \geq 1$ and $1 \leq i_1, \ldots, i_r, j_1, \ldots, j_s \leq m+1$, such that

$$i_r \cdots i_1 \# u_{i_1} \cdots u_{i_r} \cdot \$ = j_s \cdots j_1 \# v_{j_1} \cdots v_{j_s} \$ \cdot \varepsilon.$$

The above implies that $r = s$, $i_r \cdots i_1 = j_r \cdots j_1$ and $u_{i_1} \cdots u_{i_r} = v_{i_1} \cdots v_{i_r}$. This is impossible since I'_{PCP} was assumed not to have a solution.

By (12) this concludes the proof of the theorem. □

5 Shuffle on Trajectories Orthogonality

In this section we consider orthogonality modulo the shuffle on trajectories operation. We consider the question of the decidability of the orthogonality property for this operation.

Theorem 5. *Given regular languages $L_1, L_2 \subseteq \Sigma^*$ and a regular set of trajectories T, it is decidable whether L_1 and L_2 are $\sqcup\!\sqcup_T$-orthogonal.*

Proof. [7] Let f_i be the rational transduction defined by

$$f_i = \{(u, u') : u' \in u \rightsquigarrow_T L_i\}$$

for $i = 1, 2$. Here $\rightsquigarrow_T$ is the deletion along trajectories operation [4, 8]; the closure properties of $\rightsquigarrow_T$ easily show that f_i is a rational transduction for $i = 1, 2$. Let $L = L_1 \sqcup\!\sqcup_T L_2$. Then L_1 and L_2 are $\sqcup\!\sqcup_T$-orthogonal if and only if $f_i|_L$ is a function for $i = 1, 2$. Whether or not each $f_i|_L$ is a function can be decided by a result of Schützenberger [14].

We note that Iwama [5] has proven that it is decidable whether L_1 and L_2 are $\sqcup\!\sqcup$-orthogonal for regular languages L_1, L_2 (i.e., the case of $T = \{0, 1\}^*$).

6 Conclusions

In this paper, we have considered orthogonality as a general concept of binary operations. Our investigation is motivated by the concept of CDMA multiplexing schemes. We have obtained new results for orthogonality related to shuffle on trajectories and concatenation. Some questions are still open, including deciding whether a regular language has an orthogonal decomposition with respect to concatenation, and one-variable language equations with respect to orthogonal shuffle on trajectories.

7 Acknowledgement

We thank the anonymous referees for correcting several mistakes in an earlier version of the paper.

References

1. M. Anselmo and A. Restivo, On languages factorizing the free monoid, *International Journal of Algebra and Computation* **6** (1996) 413–427.
2. J. Castaing and L. Delathauwer, An Algebraic Technique for the Blind Separation of DS-CDMA Signals, in *Proc. 12th European Signal Processing Conference (EUSIPCO 2004)*, Vienna. (2004) 377–380.

[7] We are grateful to the anonymous referee who suggested this construction.

3. J.H. Conway, *Regular algebra and finite machines.* Chapman and Hall, 1971.
4. M. Domaratzki, Deletion along trajectories, *Theor. Comp. Sci. 320*, 2–3 (2004), 293–313.
5. K. Iwama, Unique decomposability of shuffled strings. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing* (1983), D. Johnson *et al.*, Eds., pp. 374–381.
6. H. Jürgensen and S. Konstantinidis, Codes. In: *Handbook of Formal Languages,* Vol. I, (G. Rozenberg and A. Salomaa, Eds.) Springer, 1997, pp. 511–607.
7. L. Kari, On language equations with invertible operations. *Theor. Comp. Sci. 132* (1994), 129–150.
8. L. Kari and P. Sosík, Language deletions on trajectories, Tech. Rep. 606, Computer Science Department, University of Western Ontario, 2003.
9. L. Kari and G. Thierrin, Maximal and minimal solutions to language equations, *J. Comput. System Sci.* **53** (1996) 487–496.
10. W. Kuich and A. Salomaa, *Semirings, Automata, Languages.* EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1986.
11. A. Mateescu, G. Rozenberg and A. Salomaa, Shuffle on trajectories: Syntactic constraints, *Theor. Comput. Sci.* **197** (1998) 1–56.
12. A. Mateescu, A. Salomaa and S. Yu, Factorizations of languages and commutativity conditions, *Acta Cybernetica* **15** (2002) 339–351.
13. R. Rao and S. Dianat, *Basics of Code Division Multiple Access(CDMA)*, SPIE, 2005.
14. M.P. Schützenberger, Sur les relation rationnelles entre monoïdes libres, *Theor. Comput. Sci.* **3** (1976) 243–259.
15. S. Yu, Regular languages. In: *Handbook of Formal Languages,* Vol. I, (G. Rozenberg and A. Salomaa, Eds.) Springer, 1997, pp. 41–110.

Language equations with positional addition

Artur Jeż[1] ⋆ and Alexander Okhotin[2,3] ⋆⋆

[1] Institute of Computer Science, University of Wrocław, Poland aje@ii.uni.wroc.pl
[2] Academy of Finland
[3] Department of Mathematics, University of Turku, Finland
alexander.okhotin@utu.fi

Abstract. Language equations with an operation of adding numbers written in a positional notation are considered. It is shown that this operation together with union and intersection invests equations with a sufficient expressive power to simulate every trellis automaton, as well as to specify some languages not accepted by any trellis automaton. The results have applications to conjunctive grammars over a unary alphabet and the related families of language equations.

1 Introduction

Resolved systems of language equations of the general form

$$\begin{cases} X_1 = \varphi_1(X_1, \ldots, X_n) \\ \quad \vdots \\ X_n = \varphi_n(X_1, \ldots, X_n) \end{cases} \tag{*}$$

with different operations allowed in their right-hand sides are the oldest and likely the most studied type of language equations.

As established by Ginsburg and Rice [3], if the allowed operations in (*) are concatenation and union, then least solutions of such systems provide semantics to the context-free grammars. If in addition the intersection operation may be used, the resulting systems define the class of *conjunctive grammars* introduced by Okhotin [9, 10]; these grammars have attractive practical properties, such as efficient parsing algorithms, and are surveyed in a recent article [14]. The case of systems (*) using negation but neither union nor intersection was first considered by Leiss [8] and recently studied by Okhotin and Yakimova [13]; by their expressive power, these systems are incomparable with the context-free grammars.

An important case of such equations is the case of a unary alphabet. In the case of union and concatenation, it is well-known that the unary context-free languages are regular. In contrast, equations with concatenation and complementation can specify the nonregular language

⋆ Supported by MNiSW grant number N206 024 31/3826, 2006–2008.
⋆⋆ Supported by the Academy of Finland under grant 118540.

M. Kunc, A. Okhotin (Eds.): Theory and Applications of Language Equations
TUCS General Publication No 44, pp. 54–66, 2007.

$\{a^n \mid$ the octal notation of n starts with $\mathtt{1}$, $\mathtt{2}$ or $\mathtt{3}\}$, see Leiss [8]. As for equations with union, intersection and complementation, their expressive power over a unary alphabet (or, equivalently, the expressive power of unary conjunctive grammars) was one of the long-standing open problems in the area [9, 14], and it was conjectured that only regular languages are can be obtained.

This conjecture has recently been disproved by Jeż [6] who constructed a conjunctive grammar for the language $\{a^{4^n} \mid n \in \mathbb{N}\}$. This grammar, written in the form of a system of language equations, is as follows:

$$\begin{cases} X_1 = (X_2X_2 \cap X_1X_3) \cup a \\ X_2 = (X_{12}X_2 \cap X_1X_1) \cup aa \\ X_3 = (X_{12}X_{12} \cap X_1X_2) \cup aaa \\ X_{12} = (X_3X_3 \cap X_1X_2) \end{cases} \tag{1}$$

Its least solution is $X_i = \{a^\ell \mid$ base-4 notation of ℓ is $i\mathtt{0}\ldots\mathtt{0}\}$, for $i = 1, 2, 3, 12$.

The system (1) effectively encodes manipulations with the positional notation of numbers. In order to continue the study of unary language equations, it would be convenient to deal explicitly with numbers written in positional notation. The subject of the present paper are equations on languages over alphabets $\Sigma_k = \{\mathtt{0}, \mathtt{1}, \ldots, k-1\}$, in which every word is interpreted as k-ary notation of a number. Instead of the concatenation operation we shall use addition of numbers written in k-ary notation. Our goal is to establish the expressive power of such equations and apply these results to unary language equations.

2 Languages of numbers written in positional notation

Fix a number $k \geqslant 2$ and consider the alphabet $\Sigma_k = \{\mathtt{0}, \mathtt{1}, \mathtt{2}, \ldots, k-1\}$ of k-ary digits. Words over this alphabet represent non-negative integers written in k-ary notation. Let the empty word $\varepsilon \in \Sigma_k^*$ denote the number $\mathtt{0}$. No representation of a number shall begin with $\mathtt{0}$, that is, the set of valid representations of numbers is $\Sigma_k^* \setminus \mathtt{0}\Sigma_k^*$. We shall consider formal languages over this alphabet, such as the following language of binary notations of all powers of two: $\mathtt{10}^* \subseteq \Sigma_2^*$.

Define a word operation $\boxplus_k : \Sigma_k^* \times \Sigma_k^* \to \Sigma_k^*$, which represents addition of numbers in k-ary notation:

$$u \boxplus_k v = \{\text{the } k\text{-ary notation of } i+j \mid u \text{ is the } k\text{-ary notation of } i, \\ v \text{ is the } k\text{-ary notation of } j\}$$

The notation $\boxplus$ will be used when the alphabet is clear from the context. We shall use this operation in the language-theoretic rather than arithmetical context. Then it can be said that $u \boxplus v$ combines the corresponding symbols of u and v and thus computes a certain word of length $\max(|u|, |v|)$ or $\max(|u|, |v|) + 1$. This, in particular, can be used to modify individual symbols of a word:

Example 1 (Modifying a digit). Let $uiv \in \Sigma_k^* \setminus \mathtt{0}\Sigma_k^*$, let $i \neq k-1$. Then $uiv \boxplus \mathtt{10}^{|v|} = u(i+1)v$, that is, one symbol has been modified.

Note that such a modification is irreversible: there is no $w \in \Sigma_k^*$, such that $u(i+1)v \boxplus_k w = uiv$. Define the corresponding subtraction operator on words as follows:

$$u \boxminus_k v = \{\text{the } k\text{-ary notation of } i-j \mid u \text{ is the } k\text{-ary notation of } i, \\ v \text{ is the } k\text{-ary notation of } j, \text{ for } i \geqslant j\} .$$

This operator is formally an inverse of $\boxplus_k$, since $w \boxminus_k u = v$ if and only if $u \boxplus v = w$.

Let us extend the operations of k-ary addition and subtraction to languages in the standard way as $K \boxplus L = \{u \boxplus v \mid u \in K, v \in L\}$ and $K \boxminus L = \{u \boxminus v \mid u \in K, v \in L\}$. Then, for instance, for $k = 10$ it can be said that $\mathtt{9}^+ \boxplus \mathtt{2} = \mathtt{10}^*\mathtt{1}$ and $\mathtt{3}^* \boxminus \mathtt{1}^* = \mathtt{3}^*\mathtt{2}^*$.

By definition, this operation on languages is *monotone* with respect to the partial ordering of languages by inclusion, that is, whenever $K \subseteq K'$ and $L \subseteq L'$, it holds that $K \boxplus K' \subseteq L \boxplus L'$. It is also *continuous*, in the sense that for every two increasing sequences of languages $\{K_n\}_{n=1}^{\infty}$ and $\{L_n\}_{n=1}^{\infty}$, the sequence $\{K_n \boxplus L_n\}_{n=1}^{\infty}$ has the least upper bound $\bigsqcup_{n>0} K_n \boxplus L_n = \bigsqcup_{n>0} K_n \boxplus \bigsqcup_{n>0} L_n$. These properties are essential for considering language equations with this operator. By the basic results on fixed points, every resolved system of equations $X_i = \varphi_i(X_1, \ldots, X_n)$ $(i = 1, \ldots, n)$ using only monotone and continuous operations (in particular $\boxplus$, $\cup$ and $\cap$) has a least solution given by $\bigsqcup_{n>0} \varphi^n(\varnothing, \ldots, \varnothing)$, where φ is a vector notation for $(\varphi_1, \ldots, \varphi_n)$.

Our primary motivation for studying these equations is their correspondence to language equations over an alphabet $\{a\}$. Define the bijection $f_k : \Sigma_k^* \setminus \mathtt{0}\Sigma_k^* \to a^*$ as

$$f_k(w) = a^n, \quad \text{where } w \text{ read as } k\text{-ary notation represents } n.$$

Since $f_k(u \boxplus_k v) = f_k(u) \cdot f_k(v)$ and $f_k^{-1}(a^m \cdot a^n) = f_k^{-1}(a^m) \boxplus f_k^{-1}(a^n)$, this mapping is an isomorphism. Extend it to languages in a usual way as $f_k(L) = \{f_k(w) \mid w \in L\}$, obtaining an isomorphism between unary languages and subsets of $\Sigma_k^* \setminus \mathtt{0}\Sigma_k^*$. This isomorphism extends to systems of language equations over Σ_k using Boolean operations and $\boxplus$, and language equations over $\{a\}$ using Boolean operations and concatenation: $\boxplus$ is replaced with "$\cdot$", constants are mapped by f, Boolean operations are preserved. The solutions of the systems correspond as follows:

Lemma 1. *Let $X_i = \varphi_i(X_1, \ldots, X_n)$ be a system of language equations over the alphabet $\{a\}$ and let $Y_i = \psi_i(Y_1, \ldots, Y_n)$ be the corresponding language equations over Σ_k. Then a vector of languages $(f_k(L_1), \ldots, f_k(L_n))$ is a solution of the former system if and only if the vector of languages $(L_1, \ldots, L_n)$ is a solution of the latter system. In particular, least solutions are mapped to least solutions.*

3 Known representations

Denote by $\mathcal{L}^k_{\cup,\cap,\boxplus}$ the family of languages that occur in least solutions of systems of equations $Y_i = \psi_i(Y_1, \ldots, Y_n)$ over Σ_k, with union, intersection and $\boxplus_k$.

Clearly, for every ultimately periodic set of numbers X, $f^{-1}(X)$ is in $\mathcal{L}^k_{\cup,\cap,\boxplus}$. By Lemma 1, the system (1) with a nonperiodic solution translates to the following:

Example 2. The following system of language equations over $\Sigma_4 = \{\mathtt{0},\mathtt{1},\mathtt{2},\mathtt{3}\}$

$$\begin{cases} X_1 = \quad (X_2 \boxplus X_2 \cap X_1 \boxplus X_3) \cup \{\mathtt{1}\} \\ X_2 = \ (X_{12} \boxplus X_2 \cap X_1 \boxplus X_1) \cup \{\mathtt{2}\} \\ X_3 = (X_{12} \boxplus X_{12} \cap X_1 \boxplus X_2) \cup \{\mathtt{3}\} \\ X_{12} = \quad X_3 \boxplus X_3 \cap X_1 \boxplus X_2 \end{cases}$$

has the least solution $(\mathtt{10}^*, \mathtt{20}^*, \mathtt{30}^*, \mathtt{120}^*)$.

Consider the equation for X_1 under this substitution: $X_2 \boxplus X_2 = \mathtt{20}^* \boxplus \mathtt{20}^* = \mathtt{10}^+ \cup \mathtt{20}^*\mathtt{20}^*$ and $X_1 \boxplus X_3 = \mathtt{10}^* \boxplus \mathtt{30}^* = \mathtt{10}^+ \cup \mathtt{10}^*\mathtt{30}^* \cup \mathtt{30}^*\mathtt{10}^*$, and clearly their intersection is $\mathtt{10}^+$.

The construction of Example 2 generalizes to $w\mathtt{0}^*$, for any $w \in \Sigma_k^* \setminus 0\Sigma_k$ with $|w| = 1, 2$ [6, Thm. 3]. These results have the following important generalization:

Theorem 1 (Jeż [6, Thm. 4]). *For every $k \geqslant 2$ every regular language $L \subseteq \Sigma_k^* \setminus \mathtt{0}\Sigma_k^*$ is defined by a resolved system of language equations with union, intersection and $\boxplus_k$, that is, $L \in \mathcal{L}^k_{\cup,\cap,\boxplus}$.*

We include this system for completeness; for the proof the reader is referred to the cited paper. Let $M = (\Sigma_k, Q, q_0, \delta, F)$ be an NFA recognizing L^R. We use variables

$$\{X_{i,j,q}, X_{i,j} : 1 \leqslant i < k,\ 0 \leqslant j < k,\ q \in Q\} \cup \{Y\}\ ,$$

with the goal that their least solution is

$$L(X_{i,j}) = ij\mathtt{0}^*, \quad L(X_{i,j,q}) = ijL_M(q), \quad L(Y) = L\ .$$

As mentioned above, $X_{i,j}$ can be defined by this type of language equations, and so we focus only on equations for $X_{i,j,q}$:

$$\begin{aligned} X_{i,j,q} &= \Big(\bigcap_{n=0}^{3} X_{i,n} \boxplus X_{j-n,x,q'}\Big) \cup \{ij :\ \text{if } q = q_0\} \\ &\qquad \text{for } j > 3, \text{ every } i, \text{ and every } x, q' \text{ such that } q \in \delta(q', x)\ , \\ X_{i,j,q} &= \Big(\bigcap_{n=1}^{4} X_{i-1,j+n} \boxplus X_{k-n,x,q'}\Big) \cup \{ij :\ \text{if } q = q_0\} \\ &\qquad \text{for } j < 4 \text{ and } i \neq 1 \text{ and every } x, q' \text{ such that } q \in \delta(q', x)\ , \\ X_{1,j,q} &= \Big(\bigcap_{n=1}^{4} X_{k-n,0} \boxplus X_{j+n,x,q'}\Big) \cup \{1j :\ \text{if } q = q_0\} \\ &\qquad \text{for } j < 4, \text{ every } x, q' \text{ such that } q \in \delta(q', x)\ , \\ Y &= (L \cap \Sigma_k) \cup \bigcup_{\substack{i,j,q: \\ \delta(q,ji) \cap F \neq \varnothing}} X_{i,j,q}\ . \end{aligned}$$

Theorem 1 implies that regular constants in such systems of language equations can be effectively expressed via singleton constants. We shall use regular constants below, assuming that they are expressed according to Theorem 1.

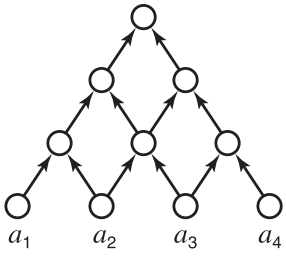
4 Representing linear conjunctive languages

Let us improve the above result by representing a larger class of formal languages. *Linear conjunctive languages* are defined by linear conjunctive grammars [9], or, in other words, by resolved systems of language equations with $\cup$, $\cap$ and linear concatenation. This family of languages can be equivalently defined by one of the simplest types of cellular automata [12]. These are *trellis automata*, also known as one-way real-time cellular automata, which were studied by Culik, Gruska and Salomaa [2], Ibarra and Kim [5], and others. Our argument will proceed by simulating the computation of such automata.

Let us define and explain trellis automata following Culik et al. [2]. A trellis automaton (TA), defined as a quintuple $(\Sigma, Q, I, \delta, F)$, processes an input string of length $n \geqslant 1$ using a uniform array of $n(n+1)/2$ nodes presented in the figure below. Each node computes a value from a fixed finite set Q. The nodes in the bottom row obtain their values directly from the input symbols using a function $I : \Sigma \to Q$. The rest of the nodes compute the function $\delta : Q \times Q \to Q$ on the values in their predecessors. The string is accepted if and only if the value computed by the top node belongs to the set of accepting states $F \subseteq Q$.

Definition 1. *A trellis automaton is a quintuple $M = (\Sigma, Q, I, \delta, F)$, in which: Σ is the input alphabet, Q is a finite non-empty set of states, $I : \Sigma \to Q$ is a function that sets the initial states, $\delta : Q \times Q \to Q$ is the transition function, and $F \subset Q$ is the set of final states.*
Extend δ to a function $\delta : Q^+ \to Q$ by $\delta(q) = q$ and

$$\delta(q_1, \ldots, q_n) = \delta(\delta(q_1, \ldots, q_{n-1}), \delta(q_2, \ldots, q_n)) \ ,$$



while I is extended to a homomorphism $I : \Sigma^* \to Q^*$.
Let $L_M(q) = \{w \mid \delta(I(w)) = q\}$ and define $L(M) = \bigcup_{q \in F} L_M(q)$.

Theorem 2 ([12]). *A language $L \subseteq \Sigma^+$ is generated by a linear conjunctive grammar if and only if L is recognized by a trellis automaton.*

Linear conjunctive languages are known to be closed under all Boolean operations, concatenation with regular languages and quotient with singletons, but under neither concatenation nor star [2, 12]. In addition, it is known that linear conjunctive languages over a one-letter alphabet generate only regular languages. From this, the following simple result used in the following can be inferred:

Lemma 2. *Let L be a linear conjunctive language over an alphabet Σ, let $u, v \in \Sigma^*$ and $a \in \Sigma^*$. Then the language $K = L \cap ua^*v$ is regular.*

Proof. The language $\widetilde{K} = \{u\}^{-1} \cdot K \cdot \{v\}^{-1}$ is linear conjunctive by the closure of this family under quotient with singletons. Since $\widetilde{K}$ is a unary linear conjunctive language, it is regular. Then $K = u\widetilde{K}v$ is regular. □

Let us now show how the computation of a trellis automaton can be simulated by the class of language equations introduced in the previous section.

Lemma 3. *For every $k \geqslant 4$ and for every trellis automaton M over Σ_k, such that $L(M) \cap \mathtt{0}^* = \varnothing$, there exists and can be effectively constructed a resolved system of language equations over the alphabet Σ_k using operations $\cup$, $\cap$ and $\boxplus$ and regular constants, such that the least solution of this system contains a component $\big((\mathtt{1} \cdot L(M)) \boxminus \mathtt{1}\big) \cdot \mathtt{10}^*$.*

Proof. In this proof we abuse the notation of $\boxplus$ and $\boxminus$ by allowing their arguments and the result to have leading $\mathtt{0}$'es. We shall do this only for the second argument equal to $\mathtt{1}$. Under these conditions we define the result to have the same length as the first argument, e.g., $\mathtt{0100} \boxminus_{10} \mathtt{1}$ is deemed to be $\mathtt{0099}$. We shall never use this notation in a context where these requirements cannot be fulfilled, that is, for $(k-1)^+ \boxplus \mathtt{1}$ and for $\mathtt{0}^* \boxminus \mathtt{1}$. This abused notation is used only in the text of the proof, while language equations strictly adhere to the definition.

For a given trellis automaton $M = (\Sigma_k, Q, I, \delta, F)$ we define language equations with the set of variables X_q for $q \in Q$, and with an additional variable Y. We will prove that their least solution is $X_q = L_q$, $Y = L$, where

$$L_q = 1\big((L_M(q) \setminus 0^*) \boxminus 1\big)10^* = \{1w10^\ell \mid \ell \geqslant 0, w \notin (k-1)^*, w \boxplus 1 \in L_M(q)\},$$
$$L = 1\big((L(M) \setminus 0^*) \boxminus 1\big)10^* = \{1w10^\ell \mid \ell \geqslant 0, w \notin (k-1)^*, w \boxplus 1 \in L(M)\}\ .$$

Let us define expressions λ_i and ρ_j, for $i, j \in \Sigma_k$, which depend upon the variables X_q, and which we use as building blocks for constructing equations for X_q. Let us also define constants R_q, which are regular by Lemma 2.

$$R_q = 1\Big(\big((0^*(\Sigma_k \setminus 0) \cup (\Sigma_k \setminus 0)0^*) \cap L_M(q)\big) \boxminus 1\Big)10^*$$
$$\lambda_i(X) = 1i\Sigma_k^* \cap \bigcup_{i'} \Big((X \cap 1i'\Sigma^*) \boxplus 10^* \cap 2i'\Sigma_k^* \Big) \boxplus (k+i-2)0^*, \text{ for } i = 0, 1$$
$$\lambda_i(X) = 1i\Sigma_k^* \cap \bigcup_{i'} \Big((X \cap 1i'\Sigma^*) \boxplus 10^* \cap 2i'\Sigma_k^* \Big) \boxplus 1(i-2)0^*\ , \text{ for } i \geqslant 2$$
$$\rho_j(X) = \bigcup_{j'} \Big(\Big((X \cap 1\Sigma_k^* j'10^*) \boxplus 10^* \cap 1\Sigma_k^* j'20^* \Big) \boxplus (k+j-2)10^* \Big) \cap$$
$$\cap\ 1\Sigma_k^* j10^* \text{ for } j = 0, 1 \tag{2}$$
$$\rho_j(X) = \bigcup_{j'} \Big(\Big((X \cap 1\Sigma_k^* j'10^*) \boxplus 10^* \cap 1\Sigma_k^* j'20^* \Big) \boxplus 1(j-2)10^* \Big) \cap$$
$$\cap\ 1\Sigma_k^* j10^* \text{ for } 2 \leqslant j \leqslant k-2 \tag{3}$$
$$\rho_{k-1}(X) = \bigcup_{j'} \Big(\Big((X \cap 1\Sigma_k^* j'10^*) \boxplus 10^* \cap 1\Sigma_k^* j'20^* \Big) \boxplus (k-3)10^* \Big) \cap$$
$$\cap\ 1\Sigma_k^*(k-1)10^* \tag{4}$$

Using this notation, the system of language equations is constructed as follows:

$$\begin{cases} X_q = R_q \cup \bigcup\limits_{\substack{q,q':\delta(q',q'')=q \\ i,j\in\Sigma_k}} \lambda_i(X_{q''}) \cap \rho_j(X_{q'}) \quad \text{(for all } q \in Q\text{)} \\ Y = \bigcup_{q\in F} X_q \end{cases}$$

The construction works as follows: the sets R_q represent the starting part of X_q that we use to compose longer words. A word $w \in \Sigma^{\geqslant 2}$ belongs to $L_M(q)$ iff there are states q', q'' such that $\delta(q', q'') = q$ and $\Sigma_k^{-1} w \in L_M(q'')$ and $w\Sigma_k^{-1} \in L_M(q')$. And so a word $1(w \boxminus 1)10^*$ should belong to X_q if and only if there are two witnesses belonging to $X_{q''}$ and $X_{q'}$ (with some additional constraints). This is specified in ρ and λ, respectively. These expressions represent adding digits at some specific positions, so that selected digits in the original word could be modified in the resulting word, while the rest of the digits remain the same. The main technical difficulty is to force the addition of digits at proper positions. This is achieved by adding the digits in two phases, and by checking the form of intermediate and final results using intersection with regular constants.

Main Claim. *The least solution of the system is* $(\ldots, L_q, \ldots, L)$.

Claim 1. For every word $1w10^m \in 1\Sigma_k^+ 10^* \setminus 1(k-1)^*10^*$,

$$\lambda_i(\{1w10^m\}) = \{1iw10^m\}.$$

Proof. Consider the expression λ_i for any i. Let $u = 1i'w'10^m$. In the subexpression corresponding to i' we add $u' = 10^\ell$ for any $\ell \geqslant 0$ and require that the result has $2i'$ as its first two digits. If the leading 1s in u and u' are in the same position, that is, if $|i'w'10^m| = \ell$, then the sum is $2i'w'10^m \in 2i'\Sigma_k^*$. If the 1 in u' is to the left of the leading 1 from u then $u \boxplus u'$ begins with 1. If the leading 1 of u' lands to the right of the leading 1 of u, then either the result does not begin with 2 (if there is no carrying into the first position), or the second digit is not i' (if there is such a carrying). These wrong combinations are filtered out by intersection with $2i'\Sigma_k^*$. Altogether we obtain $(\{1i'w'10^m\} \boxplus 10^*) \cap 2i'\Sigma_k^* = \{2i'w'10^m\}$.

The second addition in λ_i follows the same principle. Our analysis splits depending on the value of i. Consider first $i \in \{0, 1\}$. We start with $u'' = 2i'w'10^m$, add $u''' = (k+i-2)0^\ell$ to it and require that the result begins with $1i$. Since u'' has 2 as the leading symbol, we must modify it to obtain a result of this form. If the positions of 2 and $(k+i-2)$ are the same, that is, $|i'w'10^m| = \ell$, then the result $u'' \boxplus u''' = 1ii'w'10^m$ is as intended. If we add $k+i-2$ left of 2 in u'', then the leading digit is $k+i-2 \in \{k-2, k-1\}$, which is not 1, since $k \geqslant 4$. If we add to the right, then the leading digit is 2 or 3. Therefore, $(\{2i'w'10^m\} \boxplus (k+i-2)0^*) \cap 1i'\Sigma_k^* = \{1ii'w'10^m\}$.

Consider now $i \geqslant 2$. If $i-2$ is at the same position as the leading 2 in u'', then we obtain $1iw10^m$, as intended. If we add $i-2$ to the left of the leading 2 in u'', then the second digit from the left in the result is $i-2$. If the leading 1 is right of 2, then the leading digit in $u \boxplus u' \boxplus u''$ is not 1, (since $k \geqslant 4$). If

the leading 1 hits the leading 2, then if we get 1 as a leading symbol, the second symbol is $0 \neq i$. Thus all unintended results are filtered by intersection with $1i'\Sigma_k^*$, and, as in the previous case, the result is $\{1ii'w'10^m\}$. □

Claim 2. For every word $u \in 1\Sigma_k^+10^* \setminus 1(k-1)^*10^*$,

$$\rho_j(\{u\}) = \begin{cases} \{1wj10^m\}, & \text{if } u = 1w10^{m+1} \text{ and } j = k-1\ , \\ \{1wj10^m\}, & \text{if } u = 1(w \boxminus 1)10^{m+1} \text{ and } j \neq k-1\ , \end{cases}$$

Proof. Consider the definition of ρ_j for any j and let $u = 1w'j'10^m$, where $w' \in \Sigma_k^*$ and $j' \in \Sigma_k$. As in Lemma 1, we add any $u' = 10^\ell$ and use intersection with $1\Sigma_k^*j'20^*$ to require that $u \boxplus u'$ has $j'2$ as last non-zero digits. This can be achieved only for $m = \ell$, and so $u \boxplus u' = 1iw'j'20^m$ for $u' = 10^m$. The analysis splits depending on value of j.

Consider first $j \leqslant 1$ and (2). We add $u'' = (k+j-2)10^t$ and require that $u \boxplus u' \boxplus u''$ has $j1$ as last non-zero digits. If $t = m-1$, then we obtain $1(w \boxplus 1)j10^{m-1}$ as intended. Suppose $t \neq m-1$ and that we obtain a word with $j1$ as last non-zero digits. The ending 1 comes from u'', since $k \geqslant 4$. Hence $m > 0$. To obtain j as the second from the last non-zero digit we have to sum up 2 and $k+j-2$, otherwise it would be $k+j-2$ (again we use $k \geqslant 4$). And so we obtain $1(w'j' \boxplus 1)j10^{m-1}$.

Consider now $2 \leqslant j \leqslant k-2$ and (3). We add $u'' = 1(j-2)10^t$ and require that $u \boxplus u' \boxplus u''$ has $j1$ as last non-zero digits. Again for $t = m$ we obtain $1(w \boxplus 1)j10^{m-1}$, as desired. Suppose $m-1 \neq t$ and we obtain word with $j1$ as last non-zero digits. Then the ending 1 must clearly come from u'', hence $m > 0$. To get j as the second digit from the last non-zero digit we have to sum up 2 and $j-2$, otherwise it would be $j-2$. And so we obtain $1(w'j' \boxplus 1)j10^{m-1}$.

Note that this means that for $j \neq k-1$ (despite of the value of j') ρ_j transforms $1w10^m$ into $1(w \boxplus 1)j0^{m-1}$, or equivalently, $1wj10^{m-1}$ is obtained from $1(w \boxminus 1)10^m$.

Consider the case $j = k-1$. We add $u'' = (k-3)10^t$ and require that $u \boxplus u' \boxplus u''$ has $(k-1)1$ as last non-zero digits. If $t = m-1$ then we obtain $1wj10^{m-1}$, as desired. Suppose $t \neq m-1$ and we obtain word with $(k-1)1$ as last non-zero digits. Then the ending 1 must come from u''. In particular, $m > 0$. To obtain $k-1$ as the second digit from the last non-zero digit we have to sum up 2 and $k-3$, otherwise it would be $k-3$. And so we obtain $1w'j'(k-1)10^{m-1}$.

Note, that this means that ρ_{k-1} transforms $1w10^m$ into $1w(k-1)10^{m-1}$. □

Claim 3. $\lambda_i(L_q) = 1(i(L_M(q) \setminus 0^*) \boxminus 1)10^*$.

Proof. Since λ_j is a superposition of $\cup$, $\cap$ and $\boxplus$, $\lambda_i(L_q) = \bigcup_{w \in L_q} \lambda_i(\{w\})$. Then, substituting elements of L_q into Claim 1, we obtain that $\lambda_i(L_q)$ contains all words $1iw10^m$, such that $w \in (L_M(q) \setminus 0^*) \boxminus 1$, which gives the requested expression. □

Claim 4. $\rho_j(L_q) = 1((L_M(q) \setminus 0^*)(j+1 \mod k) \boxminus 1)10^*$.

Proof. As in the previous proof, we use the property that $\rho_j(K) = \bigcup_{w\in K} \rho_j(\{w\})$ for any K. For $j = k-1$, by the definition of L_q

$$\rho_{k-1}(L_q) = \{1w(k-1)10^m \mid 1w10^{m+1} \in L_q\} = 1((L_M(q) \setminus 0^*) \boxminus 1)(k-1)10^*,$$

which, by Claim 2, equals $1((L_M(q) \setminus 0^*)0 \boxminus 1)10^*$. For $j \neq k-1$,

$$\rho_j(L_q) = \{1wj10^m \mid 1(w \boxminus 1)10^{m+1} \in L_q\} = 1(L_M(q) \setminus 0^*)j10^*$$

according to the definition of L_q, which is equal to $1(((L_M(q)\setminus 0^*)(j+1)) \boxminus 1)10^*$ by Claim 2. □

Claim 5. For the least solution $(\ldots, S_q, \ldots)$ of the constructed system and for every $q \in Q$, $L_q \subseteq S_q$.

Proof. Let $1w10^n \in L_q$, that is, $w \boxplus 1 \in L_M(q)$, where $w \in \Sigma_k^+ \setminus (k-1)^+$. Using induction on the length of w, let us show that $1w10^n \in S_q$.

If $w \boxplus 1 \in L_M(q)$ has at most one non-zero digit, which is the first one or the last one, then $1w10^n \in R_q \subseteq S_q$ by the equation for X_q.

Otherwise, let $w \boxplus 1 = iuj$, where $i, j \in \Sigma_k$, $u \in \Sigma_k^*$ and $iu, uj \notin 0^*$. Since $iuj \in L_M(q)$, there exist states $q', q'' \in Q$, such that $iu \in L_M(q')$, $uj \in L_M(q'')$ and $\delta(q', q'') = q$. Since $iu, uj \notin 0^*$, we can define $w'' = uj \boxminus 1$ and $w' = iu \boxminus 1$. Then, according to the definition of $(\ldots, L_q, \ldots)$, $1w'10^{n+1} \in L_{q'}$ and $1w''10^n \in L_{q''}$. By the induction assumption, $1w'10^{n+1} \in S_{q'}$ and $1w''10^n \in S_{q''}$. We will prove that

$$\lambda_i\big(\{1w''10^n\}\big) \cap \rho_{j-1 \bmod k}\big(\{1w'10^{n+1}\}\big) = \{1w10^n\}\ .$$

First consider $\lambda_i(1w''10^n)$. By Claim 1, $\lambda_i(1w''10^n) = \{1iw''10^n\}$. To see that $1iw''10^n = 1w10^n$, consider that $w \notin (k-1)^+$, and hence the first symbol of w and of $w \boxplus 1$ are the same. Then $w = iuj \boxminus 1 = i(uj \boxminus 1) = iw''$, which proves that $\lambda_i(1w''10^n) = \{1w10^n\}$.

Consider now $\rho_{j-1}(1w'10^{n+1})$ in the case $j \neq 0$. By Claim 2, it equals $\{1(w' \boxplus 1)(j-1)10^n\}$. Now note that $(w' \boxplus 1)(j-1) = (iu)(j-1) = iuj \boxminus 1 = w$, and hence $\rho_{j-1}(1w'10^{n+1}) = \{1w10^n\}$.

In the case $j = 0$, $\rho_{k-1}(1w'10^{n+1}) = \{1w'(k-1)10^n\}$ by Claim 2. To see that $w'(k-1) = w$, consider that $w = iu0 \boxminus 1 = (iu \boxminus 1)(k-1) = w'(k-1)$. Thus $\rho_{j-1 \bmod k}(\{1w'10^{n+1}\}) = \{1w10^n\}$ for each j.

The claim follows by the equation for X_q. □

Claim 6. For every $q \in Q$, $L_q \supseteq \varphi_q(\ldots, L_{\widetilde{q}}, \ldots)$.

Proof. Consider any word $1w10^n$ obtained by intersection of $\lambda_i(L_{q''})$ and $\rho_j(L_{q'})$ for some q', q'' such that $\delta(q', q'') = q$. Then $w \notin (k-1)^+$. By Claim 4, $(w \boxplus 1)\Sigma_k^{-1} \in L_M(q')$ and by Claim 3, $\Sigma_k^{-1}(w \boxplus 1) \in L_M(q'')$. Hence, $w \boxplus 1 \in L_M(q)$, and this yields the claim. □

The proof of the main claim proceeds as follows: By Claim 5,

$$(\ldots,\varnothing,\ldots) \sqsubseteq (\ldots,L_q,\ldots) \sqsubseteq (\ldots,S_q,\ldots)$$

Since φ is monotone,

$$\bigsqcup_{n\geqslant 0} \varphi^n(\ldots,\varnothing,\ldots) \sqsubseteq \bigsqcup_{n\geqslant 0} \varphi^n(\ldots,L_q,\ldots) \sqsubseteq \bigsqcup_{n\geqslant 0} \varphi^n(\ldots,S_q,\ldots)$$

Since $(\ldots,S_q,\ldots)$ is the least solution,

$$(\ldots,S_q,\ldots) = \varphi(\ldots,S_q,\ldots) = \bigsqcup_{n\geqslant 0} \varphi^n(\ldots,\varnothing,\ldots)\,.$$

Also, by Claim 6, $\varphi(\ldots,L_q,\ldots) \sqsubseteq (\ldots,L_q,\ldots)$, and hence

$$(\ldots,S_q,\ldots) \sqsubseteq (\ldots,L_q,\ldots) \sqsubseteq (\ldots,S_q,\ldots) \quad \square$$

Lemma 4. *For every $k \geqslant 4$ and for every trellis automaton M over Σ_k there exists and can be effectively constructed a resolved system of language equations over the alphabet Σ_k using the operations $\cup$, $\cap$ and $\boxplus$ and regular constants, such that its least solution contains a component $\mathtt{1}\cdot L(M)$.*

Proof. For every $j \in \Sigma_k$, the language $(L(M)\cdot\{j\}^{-1}) \setminus \mathtt{0}^*$ is recognized by a trellis automaton M_j due to the closure properties of trellis automata. Since $L(M_j)\cap\mathtt{0}^* = \varnothing$, by Lemma 3, there exists a system of language equations, such that one of its variables, Y_j, represents the language

$$\big(((\mathtt{1}\cdot L(M)\cdot\{j\}^{-1}) \setminus \mathtt{10}^*) \boxminus \mathtt{1}\big)\cdot\mathtt{10}^*.$$

The languages $(L(M)\cdot\{j\}^{-1}) \setminus \mathtt{0}^*$ sum up to $(L(M)\cdot\Sigma_k^{-1}) \setminus \mathtt{0}^*$ and represent the words in $\Sigma_k^* \setminus 0^*\Sigma_k$ accepted by M. In order to reflect the remaining words from the set $L(M)\cap\mathtt{0}^*\Sigma_k$, consider the language $C = \mathtt{1}\cdot(L(M)\cap\mathtt{0}^*\Sigma_k)$; as it is regular by Lemma 2, it can be regarded as a constant.

Let us combine the above equations for all j into a single system and add a new equation

$$Z = C \cup \bigcup_{j=0}^{k-1} (Y_j \cap \mathtt{1}\Sigma_k^*\mathtt{1}) \boxplus (\mathtt{1}j \boxminus \mathtt{1})$$

The expression $Y_j \cap \mathtt{1}\Sigma_k^*\mathtt{1}$ evaluates to $\{(1w\boxminus 1)1 \mid wj \in L(M)\setminus 0^*j\}$, and then $(Y_j \cap \mathtt{1}\Sigma_k^*\mathtt{1}) \boxplus (\mathtt{1}j \boxminus \mathtt{1})$ equals $\{\mathtt{1}wj \mid wj \in L(M)\setminus\mathtt{0}^*j\}$. The union of these expressions for all j is $\{\mathtt{1}w \mid w \in L(M)\setminus 0^*\Sigma_k\}$, and then, because of the union with C, the value of Z is $\mathtt{1}L(M)$. $\square$

Theorem 3. *For every $k \geqslant 4$ and for every trellis automaton M over Σ_k, such that $L(M)\cap\mathtt{0}\Sigma_k^* = \varnothing$, there exists and can be effectively constructed a resolved system of language equations over the alphabet Σ_k using the operations $\cup$, $\cap$ and $\boxplus$ and singleton constants, such that its least solution contains a component $L(M)$.*

Proof. For every $i \in \Sigma_k \setminus \{\mathtt{0}\}$, the language $\{i\}^{-1} \cdot L(M)$ is generated by a certain trellis automaton. By Lemma 4, there is a system of language equations, such that one of its variables, Z_i, represents the language $1 \cdot (\{i\}^{-1} \cdot L(M))$.

Combine these systems and add a new variable T with the following equation:

$$T = (L(M) \cap \Sigma_k) \cup Z_1 \cup \bigcup_{\substack{i \in \Sigma_k \setminus \{\mathtt{0},\mathtt{1}\} \\ i' \in \Sigma_k}} \Big((Z_i \cap \mathtt{1}i'\Sigma_k^*) \boxplus (i-1)\mathtt{0}^* \cap ii'\Sigma_k^* \Big)$$

Consider any Z_i for $i \geqslant 2$. Substituting the value of Z_i into the expression, one first obtains

$$Z_i \cap \mathtt{1}i'\Sigma_k^* = 1(i^{-1} \cdot L(M)) \cap \mathtt{1}i'\Sigma_k^* = \{1i'w \mid ii'w \in L(M)\}.$$

The next operations in the expression are the addition of $(i-1)\mathtt{0}^*$ and the intersection with $ii'\Sigma^*$. Let us establish the following fact:

Claim. For all $i \in \Sigma_k \setminus \{\mathtt{0},\mathtt{1}\}$ $i' \in \Sigma_k$ and $w \in \Sigma_k^*$,

$$\{1i'w\} \boxplus (i-1)\mathtt{0}^* \cap ii'\Sigma_k^* = \{ii'w\}.$$

Consider $\mathtt{1}i'w \boxplus (i-1)\mathtt{0}^\ell$. If $\ell = |i'w|$, then the sum equals $ii'w \in ii'\Sigma_k^*$. If $\ell > |i'w|$, the result is in $(i-1)\mathtt{0}^*\mathtt{1}i'w$, and hence its intersection with $ii'\Sigma_k^*$ equals $\varnothing$.

Suppose $\ell = |w|$, then $(i-1)\mathtt{0}^\ell \boxplus \mathtt{1}i'w = i''i'''w$, and the second digit i''' equals $(i-1)+i'$ modulo k. Since $i' < i'+i-1 < i'+k$, $i''' \neq i'$, and therefore $i''i'''w$ is not in $ii'\Sigma_k^*$ because of a mismatched second digit.

If $\ell < |w|$, there are two subcases. If the addition of $(i-1)\mathtt{0}^\ell$ to $i'w$ results in a carry, then $\mathtt{1}i'w \boxplus (i-1)\mathtt{0}^\ell = \mathtt{2}(i'+1 \mod k)w$, If there is no carry, then $\mathtt{1}i'w \boxplus (i-1)\mathtt{0}^\ell$ is in $\mathtt{1}\Sigma_k^*$ and again cannot be in $ii'\Sigma_k^*$. This concludes the case study necessary to establish the claim, from which there follows

$$\big(\{\mathtt{1}i'w \mid ii'w \in L(M)\} \boxplus (i-1)\mathtt{0}^*\big) \cap ii'\Sigma^* = L(M) \cap ii'\Sigma^* \tag{5}$$

Summing this up over i', we obtain $L(M) \cap i\Sigma^+$, and summing up the latter over i and adding Z_1, we obtain $L(M) \cap (\Sigma^{\geqslant 2} \setminus 0\Sigma_k^*)$. One-letter words are given separately. Hence, the T-component of the least solution of the system is $L(M)$.

The transition from regular to singleton constants is by Theorem 1. □

5 Separation from linear conjunctive languages

We have shown that every linear conjunctive language over a k-ary alphabet is in $\mathcal{L}^k_{\cup,\cap,\boxplus}$. We shall now establish that $\mathcal{L}^k_{\cup,\cap,\boxplus}$ is a proper superset of the linear conjunctive languages. This is done by specifying the language $\{1^{n+1}0^{2^{2^n}+2^n+1} \mid n \geqslant 0\}$, which is not linear conjunctive, as follows from Buchholz and Kutrib [1, Thms. 4.1, 5.5].

Proposition 1. *The language* $L = \{1^n 0^{2^n} 1 0^{2^{2^n}} \mid n \geqslant 0\}$ *is linear conjunctive.*

To see this, consider the well-known fact that the language $L_1 = \{1^n 0^{2^n} \mid n \geqslant 0\}$ is recognized by a trellis automaton [5], that is, it is linear conjunctive. The language $L_2 = \{0^m 10^{2^m} \mid m \geqslant 0\}$ is linear conjunctive by the same construction. Then $L = L_1 10^* \cap 1^* L_2$ is a linear conjunctive language by their closure properties.

Lemma 5. *Consider* $\Sigma_4 = \{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}\}$. *The language* $L' = \{\mathtt{1}^{n+1}\mathtt{0}^{2^{2^n}+2^n+1} \mid n \geqslant 0\} \subseteq \Sigma_4^*$ *is in* $\mathcal{L}^4_{\cup,\cap,\boxplus}$.

Proof. Consider the above language L over Σ_4. It is in $\mathcal{L}^4_{\cup,\cap,\boxplus}$ by Theorem 3. Since $L' = (L \boxplus_4 \mathtt{3}^+\mathtt{0}^*) \cap \mathtt{1}^+\mathtt{0}^*$, the language L' is in $\mathcal{L}^4_{\cup,\cap,\boxplus}$ as well. □

This is sufficient to separate $\mathcal{L}^k_{\cup,\cap,\boxplus}$ from linear conjunctive languages. Let us now consider the complexity of languages in $\mathcal{L}^k_{\cup,\cap,\boxplus}$.

Lemma 6. *The family* $\mathcal{L}^k_{\cup,\cap,\boxplus}$ *contains an NP-complete language.*

Proof. Consider the alphabet Σ_7. It is known that the following language of Boolean circuits evaluating to true on given values is linear conjunctive [11]:

$$L = \{\alpha\sigma_1 \dots \sigma_n \mid \alpha \in \{\mathtt{4}, \mathtt{5}\}^*,\ \sigma_i \in \{1, 2\}^*,\ \alpha \text{ is a description of a circuit with inputs } x_1, \dots, x_n, \text{ which computes } \mathit{true} \text{ on values } x_i = \mathit{true} \text{ iff } \sigma_i = 2\}$$

The exact form of the description α is irrelevant here; what is important that L is in $\mathcal{L}^7_{\cup,\cap,\boxplus}$. Next, consider the language

$$K = (L \boxplus \{1, 2\}^*) \cap \{4, 5\}^* 3^*,$$

which is in $\mathcal{L}^7_{\cup,\cap,\boxplus}$ as well. It is easy to see that the language K equals

$$\{\alpha 3^n \mid \alpha \in \{4, 5\}^*,\ \alpha \text{ is a description of a circuit with inputs } x_1, \dots, x_n, \text{ which evaluates to } \mathit{true} \text{ on some input values}\},$$

and its NP-completeness is obvious. □

Theorem 4. *The family of languages* $\mathcal{L}^k_{\cup,\cap,\boxplus}$ *properly includes the family of linear conjunctive languages. It is contained in EXPTIME and contains an NP-complete language.*

The EXPTIME upper bound follows from the P upper bound for conjunctive grammars [9, 10] according to Lemma 1. The rest is given in Lemmata 5 and 6.

6 Conclusions and open problems

We have considered the expressive power of language equations over sets of positional notations of numbers. The new family was shown to be a proper superset of linear conjunctive languages. It is contained in EXPTIME and includes some

NP-complete languages. More precise estimations of its complexity are proposed for future research.

Our motivation for the study of these equations came from the study of language equations over a unary alphabet, and our results have strong implications on conjunctive grammars, on which we elaborate in an upcoming paper [7]. Using the known linear conjunctiveness of the language of computation histories of a Turing machine [4, 12] together with Theorem 3, one can obtain undecidability of emptiness and regularity for conjunctive grammars over $\{a\}$, as well as construct languages that grow faster than a given computable function.

Irrespective of these connections, we believe we have demonstrated that language equations over numbers in positional notation deserve attention on their own.

References

1. T. Buchholz, M. Kutrib, "On time computability of functions in one-way cellular automata", *Acta Informatica*, 35:4 (1998), 329–352.
2. K. Culik II, J. Gruska, A. Salomaa, "Systolic trellis automata", I and II, *International Journal of Computer Mathematics*, 15 (1984), 195–212, and 16 (1984), 3–22.
3. S. Ginsburg, H. G. Rice, "Two families of languages related to ALGOL", *Journal of the ACM*, 9 (1962), 350–371.
4. J. Hartmanis, "Context-free languages and Turing machine computations", *Proceedings of Symposia in Applied Mathematics*, Vol. 19, AMS, 1967, 42–51.
5. O. H. Ibarra, S. M. Kim, "Characterizations and computational complexity of systolic trellis automata", *Theoretical Computer Science*, 29 (1984), 123–153.
6. A. Jeż, "Conjunctive grammars can generate non-regular unary languages", *DLT 2007* (Turku, Finland, July 3–6, 2007), to appear.
7. A. Jeż, A. Okhotin, "Conjunctive grammars over a unary alphabet: undecidability and unbounded growth", *Computer Science in Russia* (CSR 2007, Ekaterinburg, Russia, September 3–7, 2007), LNCS 4649, to appear.
8. E. L. Leiss, "Unrestricted complementation in language equations over a one-letter alphabet", *Theoretical Computer Science*, 132 (1994), 71–93.
9. A. Okhotin, "Conjunctive grammars", *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), 519–535.
10. A. Okhotin, "Conjunctive grammars and systems of language equations", *Programming and Computer Software*, 28 (2002), 243–249.
11. A. Okhotin, "The hardest linear conjunctive language", *Information Processing Letters*, 86:5 (2003), 247–253.
12. A. Okhotin, "On the equivalence of linear conjunctive grammars to trellis automata", *Informatique Théorique et Applications*, 38:1 (2004), 69–88.
13. A. Okhotin, O. Yakimova, "On language equations with complementation", *DLT 2006* (Santa Barbara, USA, June 26–29, 2006), LNCS 4036, 420–432.
14. A. Okhotin, "Nine open problems for conjunctive and Boolean grammars", *Bulletin of the EATCS*, 91 (2007), 96–119.

Conjunctive macro grammars*

Vassilis Kountouriotis[1], Christos Nomikos[2], and Panos Rondogiannis[1]

[1] Department of Informatics & Telecommunications
University of Athens, Athens, Greece
{bk,prondo}@di.uoa.gr
[2] Department of Computer Science, University of Ioannina,
P.O. Box 1186, 45 110 Ioannina, Greece
cnomikos@cs.uoi.gr

Abstract. We introduce *conjunctive macro grammars*, a formalism that extends outside-in macro grammars to use conjunction in the right hand side of macro definitions. We argue that this new class of grammars is purely declarative: each such grammar can be seen as consisting of a set of *functional* language equations. In particular, we demonstrate that every conjunctive macro grammar has a unique least fixed point, which can be taken as its declarative meaning (or as the least solution of the associated set of language equations). The new formalism properly extends the class of languages that can be produced by macro grammars. We discuss several open questions that are related to the new formalism.

1 Introduction

In this paper we introduce *conjunctive macro grammars*, a new formalism that allows the use of conjunction in the right hand side of macro definitions. In this way, conjunctive macro grammars can be seen as a syntactic extension of both macro grammars [Fisc68] as-well-as conjunctive grammars [Okh01]. The new formalism can be understood in a purely declarative way: the meaning of an n-ary macro is a function that takes as arguments n sets of strings and returns a set of strings. In this respect, conjunctive macro grammars are very close to functional and logic programming languages. In fact, every grammar can be seen as defining a set of *functional* language equations and the semantics of a grammar can be seen as identifying the least fixed point (equivalently, least solution) of the set of these functional equations.

The rest of the paper is organized as follows: Section 2 presents at an intuitive level the basic concepts behind macro and conjunctive grammars and outlines their capabilities. Section 3 introduces and motivates conjunctive macro grammars. Section 4 develops the denotational semantics of conjunctive macro

* This work is supported by the $03E\Delta$ 330 research project, implemented within the framework of the "Reinforcement Programme of Human Research Manpower" ($\Pi ENE\Delta$) and co-financed by National and Community Funds (75% from E.U.-European Social Fund and 25% from the Greek Ministry of Development-General Secretariat of Research and Technology and from the private sector).

M. Kunc, A. Okhotin (Eds.): Theory and Applications of Language Equations
TUCS General Publication No 44, pp. 67–75, 2007.

grammars. Finally, section 5 presents open problems and gives pointer to future work.

2 Preliminaries

The two main formalism that we will be dealing with in this paper are *conjunctive grammars* [Okh01] and *macro grammars* [Fisc68]. Intuitively, conjunctive grammars extend context-free grammars by allowing conjunction in the right hand side of rules. Formally:

Definition 1. *A* conjunctive grammar *is a quadruple $G = (\Sigma, N, P, S)$, where Σ and N are disjoint finite nonempty sets of terminal and nonterminal symbols respectively, P is a finite set of rules, each of the form*

$$C \rightarrow \alpha_1 \& \cdots \& \alpha_m \qquad (m \geq 1, C \in N, \alpha_i \in (\Sigma \cup N)^*)$$

and $S \in N$ is the start symbol of the grammar.

The basic ideas behind conjunctive grammars can be illustrated by the following example [Okh04a]:

Example 1. Consider the grammar:

$$\begin{aligned} S &\rightarrow SAb \& Cb \mid b \\ A &\rightarrow aA \mid \varepsilon \\ C &\rightarrow bCaa \mid aC \mid baa \end{aligned}$$

It can be seen that this grammar computes the (non context-free) language $\{ba^2ba^4b \cdots ba^{2n-2}ba^{2n}b \mid n \geq 0\}$. □

Conjunctive grammars appear to be quite interesting in terms of expressive power (while at the same time retaining a reasonable parsing time). In particular, conjunctive languages are different from the class of languages defined by the intersections of context-free languages: while the intersections of unary context-free languages are always regular, it has been recently shown that conjunctive grammars on one symbol can define non-regular languages [Jez07]. Another issue that makes conjunctive grammars very interesting is that they resemble a type of logical formulas that are known as *Horn clauses*. In particular, Horn clauses form the basis of what is usually called *logic programming*, a very interesting and appealing type of programming. Actually, conjunctive grammars have been recently extended to boolean grammars [Okh04], an extension which has triggered investigations regarding the relationship between these new grammars and the theory of non-monotonic logic programming (see [Wro05,KNR06,NR07]. We will not further consider boolean grammars in this paper (apart from a short discussion in the concluding section).

Macro grammars [Fisc68] is a very elegant extension of context-free languages that supports a form of functions:

Definition 2. *A macro grammar is a 6-tuple* $G = (\Sigma, \mathcal{F}, \mathcal{V}, \rho, P, S)$*, where* Σ *is a finite set of terminal symbols;* $\mathcal{F}$ *is a finite set of nonterminal or function symbols;* $\mathcal{V}$ *is a finite set of argument or variable symbols;* ρ *is a function from* $\mathcal{F}$ *into nonnegative integers (*$\rho(F)$ *is the number of arguments that* F *takes);* $S \in \mathcal{F}$ *is the start symbol, with* $\rho(S) = 0$*; P is a finite set of productions of the form:*

$$F(x_1, \ldots, x_{\rho(F)}) \to \tau$$

where $F \in \mathcal{F}$*,* $x_1, \ldots, x_{\rho(F)}$ *are distinct members of* $\mathcal{V}$*, and* τ *is a term over* $\Sigma, \{x_1, \ldots, x_{\rho(F)}\}, \mathcal{F}, \rho$*.*

The set of terms over $\Sigma, \mathcal{V}, \mathcal{F}, \rho$ *is defined inductively:*

- ε *is a term,* a *is a term for every* $a \in \Sigma$*, and* x *is a term for every* $x \in \mathcal{V}$*;*
- *if* τ_1 *and* τ_2 *are terms, then* $\tau_1 \cdot \tau_2$ *is a term;*
- *if* $F \in \mathcal{F}$ *and* $\tau_1, \ldots, \tau_{\rho(F)}$ *are terms, then* $F(\tau_1, \ldots, \tau_{\rho(F)})$ *is a term.*

The basic ideas behind macro grammars are illustrated by the following example:

Example 2. Consider the grammar:

$$\begin{aligned} S &\to f(a, b, c) \\ f(x, y, z) &\to f(xa, yb, zc) \\ f(x, y, z) &\to xyz \end{aligned}$$

The grammar generates the non context-free language $\{a^n b^n c^n \mid n \geq 1\}$. Much more demanding languages can be represented by macro-rules:

$$\begin{aligned} S &\to f(a, aaa) \\ f(x, y) &\to f(xy, yaa) \mid x \end{aligned}$$

which generates the language $\{a^{n^2} \mid n \geq 1\}$. The trick is (see [Fisc68]) to generate the n'th square as the sum of the first n odd numbers. For this reason, f keeps a^{n^2} in the first argument position and a^{2n+1} in the second. □

One issue that comes to mind when first encountering macro grammars is "what is the order of evaluation of function applications when there exist nested function calls?". This question is closely related to the calling conventions of modern programming languages. Fischer proposed two modes of evaluation, namely the Inside-Out or IO (first evaluate the argument and then apply the function) and the Outside-In (expand the function call from the outside-in, taking the arguments in their unexpanded state). These two conventions correspond to the call-by-value and the lazy-evaluation calling approaches that are in use in modern programming languages.

To demonstrate the difference between the two conventions, we present the following example (taken from [Fisc68]):

Example 3. Consider the grammar:

$$\begin{aligned} S &\to f(g(a)) \\ f(x) &\to xx \\ g(x) &\to xa \mid xb \end{aligned}$$

Then, under the IO rule the language defined is $\{aaaa, abab\}$ while under the OI rule the language defined is $\{aaaa, aaab, abaa, abab\}$.

In the rest of the paper we will restrict attention to the OI convention since it is purely denotational ([Ten91,Wad07]). It is worth noting that macro grammars under the OI convention coincide (in terms of expressive power) with indexed grammars [Aho68]; this result is proved in [Fisc68].

From the above discussion it comes as a natural question "what would be the properties of a grammar formalism that amalgamates conjunctive and macro grammars?". In a sense, conjunctive macro grammars appear to be the first-order extension of conjunctive grammars (see also concluding section).

3 Conjunctive Macro Grammars

The class of conjunctive macro grammars results from an integration of conjunctive grammars [Okh01] and macro grammars [Fisc68]. The syntax of this new class of grammars can be formally defined as follows:

Definition 3. *A conjunctive macro grammar is a 6-tuple $G = (\Sigma, \mathcal{F}, \mathcal{V}, \rho, P, S)$, where Σ is a finite set of terminal symbols; $\mathcal{F}$ is a finite set of nonterminal or function symbols; $\mathcal{V}$ is a finite set of argument or variable symbols; ρ is a function from $\mathcal{F}$ into nonnegative integers ($\rho(F)$ is the number of arguments that F takes); $S \in \mathcal{F}$ is the start symbol, with $\rho(S) = 0$; P is a finite set of productions of the form:*

$$F(x_1, \ldots, x_{\rho(F)}) \rightarrow \tau_1 \& \cdots \& \tau_m$$

where $F \in \mathcal{F}$, $x_1, \ldots, x_{\rho(F)}$ are distinct members of $\mathcal{V}$, and $\tau_1, \ldots, \tau_m$ are terms over $\Sigma, \{x_1, \ldots, x_{\rho(F)}\}, \mathcal{F}, \rho$.

The set of terms over $\Sigma, \mathcal{V}, \mathcal{F}, \rho$ is defined inductively:

- *ε is a term, a is a term for every $a \in \Sigma$, and x is a term for every $x \in \mathcal{V}$;*
- *if τ_1 and τ_2 are terms, then $\tau_1 \cdot \tau_2$ is a term;*
- *if $F \in \mathcal{F}$ and $\tau_1, \ldots, \tau_{\rho(F)}$ are terms, then $F(\tau_1, \ldots, \tau_{\rho(F)})$ is a term.*

The macro grammars we consider can be "executed" with the outside-in (OI) mode of derivation; intuitively, this means that only top-level occurrences of function symbols can be rewritten at every step. The semantics we will shortly defined express exactly this intuitive notion. In programming language terminology, the semantics is *lazy*, ie., it does not evaluate the arguments of a function except if this is absolutely necessary.

4 The Semantics of Conjunctive Macro Grammars

In this section we develop the denotational semantics of conjunctive macro grammars. Incidentally, our presentation also applies to ordinary macro grammars

and, to our knowledge, this is the first such semantics for this type of grammars (since the standard approach [Fisc68] is based on derivations and is clearly operational).

In the developments that will follow we will need some basic definitions from domain theory (see also [Ten91]). We start by defining the notion of domain, which is the basic tool used for developing the semantics of functional languages:

Definition 4. *A partially ordered set $(D, \preceq)$ is ω-complete if and only if, for every chain $d \in D^\omega$, the least upper bound $\bigsqcup_{i\in\omega} d_i$ exists in D. A domain is a partially ordered set that is ω-complete.*

It is easy to see that if $(D, \preceq)$ is a domain, then the set D^n also defines a domain for every $n \geq 1$. In this case the corresponding ordering for the product domain is defined in a componentwise way.

The notion of monotonic and continuous functions on domains play an important role in the following:

Definition 5. *Let $(D, \preceq)$ be a domain. Then, a function $f : D \to D$ is called monotonic if $f(d) \preceq f(d')$ when $d \preceq d'$. A function f is called continuous if for every chain $d \in D^\omega$, $f(\bigsqcup_{i\in\omega} d_i) = \bigsqcup_{i\in\omega} f(d_i)$.*

We can now start developing the semantics of conjunctive macro grammars. We start by the notion of interpretation of a grammar:

Definition 6. *Let $G = (\Sigma, \mathcal{F}, \mathcal{V}, \rho, P, S)$ be a conjunctive macro grammar. An interpretation I of G is a function such that for every function symbol $F \in \mathcal{F}$, $I(F)$ is a* continuous *function from $(\Sigma^*)^{\rho(F)}$ to Σ^*. In particular, if $\rho(F) = 0$ then $I(F)$ is a subset of Σ^*.*

We denote by $\perp$ the interpretation which assigns:

- to every function symbol F, with $\rho(F) = 0$, the empty set;
- to every function symbol F, with $\rho(F) > 0$, the function which for every input returns as a result the empty set;

In the following we will find very useful an ordering of interpretations:

Definition 7. *Let $G = (\Sigma, \mathcal{F}, \mathcal{V}, \rho, P, S)$ be a conjunctive macro grammar and let I, J be two interpretations of G. We write $I \preceq J$ if:*

- *for every function symbol F with $\rho(F) = 0$, $I(F) \subseteq J(F)$;*
- *for every function symbol F with $\rho(F) > 0$ and for all $L_1, \ldots, L_{\rho(F)} \subseteq \Sigma^*$, $I(F)(L_1, \ldots, L_{\rho(F)}) \subseteq J(F)(L_1, \ldots, L_{\rho(F)})$.*

The following lemma demonstrates that the set of interpretations of a given conjunctive macro grammar forms an ω-complete partial order under $\preceq$:

Lemma 1. *Let $G = (\Sigma, \mathcal{F}, \mathcal{V}, \rho, P, S)$ be a conjunctive macro grammar and let $\mathcal{I}$ be the set of all interpretations of G. Then, $(\mathcal{I}, \preceq)$ is a domain.*

Proof. It is straightforward to show that $\preceq$ is a partial order. To show that the least upper bound of a chain of interpretations exists in the domain, simply define:

$$(\bigsqcup_{I \in U} I)(F)\left(L_1, \ldots, L_{\rho(F)}\right) = \bigcup_{I \in U} I(F)\left(L_1, \ldots, L_{\rho(F)}\right)$$

From the above definition, the proof follows easily. □

An interpretation I can be recursively extended to apply to any term as-well-as to the conjunction of terms, as follows:

Definition 8. *Let $G = (\Sigma, \mathcal{F}, \mathcal{V}, \rho, P, S)$ be a conjunctive macro grammar, let I be an interpretation of G and let s be a function from $\mathcal{V}$ to 2^{Σ^*}. Then the* extension of I with respect to s *is denoted by $[\![\cdot]\!]_s(I)$ and is recursively defined as follows:*

- $[\![\varepsilon]\!]_s(I) = \{\varepsilon\}$;
- $[\![a]\!]_s(I) = \{a\}$, *for every* $a \in \Sigma$;
- $[\![F]\!]_s(I) = I(F)$, *for every* $F \in \mathcal{F}$ *such that* $\rho(F) = 0$;
- $[\![x]\!]_s(I) = s(x)$, *for every* $x \in \mathcal{V}$;
- $[\![\tau_1 \cdot \tau_2]\!]_s(I) = [\![\tau_1]\!]_s(I) \circ [\![\tau_2]\!]_s(I)$;
- $[\![F(\tau_1, \ldots, \tau_n)]\!]_s(I) = I(F)([\![\tau_1]\!]_s(I), \ldots, [\![\tau_n]\!]_s(I))$;
- $[\![\tau_1 \& \cdots \& \tau_m]\!]_s(I) = [\![\tau_1]\!]_s(I) \cap \cdots \cap [\![\tau_m]\!]_s(I)$.

We can now define the notion of a model of a conjunctive macro grammar. Intuitively, models are those interpretations which satisfy all the rules of a conjunctive macro grammar. Notice that we are not interested in all the models of a grammar. Instead, we are seeking a *special* model, one that is the *least* (in some sense to be defined shortly).

Definition 9. *Let $G = (\Sigma, \mathcal{F}, \mathcal{V}, \rho, P, S)$ be a conjunctive macro grammar and let I an interpretation of G. Then, I is called a* model *of G if for every rule $F(x_1, \ldots, x_n) \rightarrow \tau_1 \& \cdots \& \tau_m$ in P and for every function s from $\mathcal{V}$ to 2^{Σ^*}, it holds that $I(F)(s(x_1), \ldots, s(x_n)) \supseteq [\![\tau_1 \& \cdots \& \tau_m]\!]_s(I)$.*

We can now define the semantics of conjunctive macro grammars. The basic idea is as follows. We start from an interpretation which is the least possibly defined (namely with $\perp$). Using this interpretation we evaluate the right hand sides of all the rules of the grammars, getting in this way a better approximation to the desired model. We continue this process with the new interpretation, until we eventually get to a fixed point. The existence of this fixed point is guaranteed since the operator we use in this process is monotonic and continuous (see below). We start by defining the appropriate operator:

Definition 10. *Let $G = (\Sigma, \mathcal{F}, \mathcal{V}, \rho, P, S)$ be a conjunctive macro grammar. Then, the operator T_G is a function which for every interpretation I of G, for every n-ary function symbol $F \in \mathcal{F}$ and for every function s from $\mathcal{V}$ to 2^{Σ^*} is defined as follows:*

$$T_G(I)(F)(s(x_1), \ldots, s(x_n)) = \{w \in \Sigma^* \mid \text{there exists } F(x_1, \ldots, x_n) \rightarrow \tau_1 \& \cdots \& \tau_m \text{ in } P \text{ such that } w \in [\![\tau_1 \& \cdots \& \tau_m]\!]_s(I)\}$$

We will demonstrate shortly that T_G is monotonic and continuous. We first need the following theorem:

Theorem 1. *Let $G = (\Sigma, \mathcal{F}, \mathcal{V}, \rho, P, S)$ be a conjunctive macro grammar, and let s be a function from $\mathcal{V}$ to 2^{Σ^*}. Then, for every term τ that appears in the right hand side of a rule in P, the function $[\![\tau]\!]_s$ is monotonic and continuous.*

Proof. By a structural induction on τ. □

Theorem 2. *Let $G = (\Sigma, \mathcal{F}, \mathcal{V}, \rho, P, S)$ be a conjunctive macro grammar. Then, the operator T_G is monotonic and continuous with respect to the $\preceq$ ordering of interpretations.*

Proof. Monotonicity follows directly from the monotonicity of $[\![\cdot]\!]$ (see Theorem 1). We demonstrate continuity, ie., that

$$T_G(\bigsqcup_{I \in U} I) = \bigsqcup_{I \in U} T_G(I)$$

for every directed subset U of the set of all interpretations of G. Let $F \in \mathcal{F}$, $L_1, \ldots, L_{\rho(F)} \subseteq \Sigma^*$ and $w \in \Sigma^*$. Then, we have:

$$w \in T_G(\bigsqcup_{I \in U} I)(F)(L_1, \ldots, L_{\rho(F)})$$

if and only if there exists a rule $F(x_1, \ldots, x_{\rho(F)}) \to \tau_1 \& \cdots \& \tau_{\rho(F)} \in P$ such that $w \in [\![\tau_1 \& \cdots \& \tau_{\rho(F)}]\!]_s(\bigsqcup_{I \in U} I)$, where $s(x_i) = L_i$. But then, by Theorem 1 this is equivalent to the fact that

$$w \in \bigcup_{I \in U} [\![\tau_1 \& \cdots \& \tau_{\rho(F)}]\!]_s(I).$$

By the definition of T_G this is equivalent to

$$w \in \bigsqcup_{I \in U} T_G(I).$$

This completes the proof of the theorem. □

Theorem 3. *Let $G = (\Sigma, \mathcal{F}, \mathcal{V}, \rho, P, S)$ be a conjunctive macro grammar. Then, T_G has a unique least fixed point $[T_G]^{\uparrow\omega}$ which is defined as follows:*

$$\begin{aligned} [T_G]^{\uparrow 0} &= \bot \\ [T_G]^{\uparrow n+1} &= T_G\left([T_G]^{\uparrow n}\right) \\ [T_G]^{\uparrow \omega} &= \bigsqcup_{n<\omega} [T_G]^{\uparrow n} \end{aligned}$$

Moreover, T_G is a model of G.

Proof. It is well-known (see for example [Ten91][page 96]) that if D is a domain with a least element $\perp$ and $f : D \to D$ is a continuous function, then f has a least fixed-point given by $\bigsqcup_{i\in\omega} f^i(\perp)$. The result then follows immediately. □

Having defined in a formal way the denotational semantics of conjunctive macro grammars, we can now state the following theorem:

Theorem 4. *The class of languages definable by conjunctive macro grammars is a proper superset of the class of grammars definable by ordinary macro grammars.*

Proof. It has been demonstrated [Fisc68] that macro languages under the OI rule generate the same class of languages as the indexed grammars [Aho68]. It is well-known that the indexed languages are not closed under intersection, from which the result follows immediately. □

5 Future Work

We have defined the class of conjunctive macro grammars and have demonstrated that they posses a simple denotational semantics. There are many aspects of this work that we are currently investigating (some of which we hope to include in the final version of the paper). We briefly describe some of these aspects:

Derivation Semantics: Ordinary macro grammars under the OI rule posses a very simple derivation-based semantics [Fisc68]. The same is true for conjunctive grammars (see [Okh01]). It is a simple matter to define a derivation based semantics for conjunctive macro grammars. Then, it shouldn't be hard to demonstrate the equivalence of the derivation semantics to the denotational ones defined in this paper. Similar proofs are very common in the functional programming domain (see for example [Ten91][page104]).

Boolean macro grammars: It appears conceptually easy to extend the ideas in this paper so as to obtain a syntactically broader class of grammars, namely boolean macro grammars. At first sight it seems that the semantic ideas developed in [KNR06] can be generalized to this case, but so far we have not considered all the details. With the addition of conjunction and negation, macro grammars appear to behave as a small programming language. For example, one can define the set of prime numbers, as follows:

$$\begin{array}{ll} S & \to \neg A \\ A & \to f(aa) \\ f(x) & \to f(xa) \mid g(x) \\ g(x) & \to xg(x) \mid xx \end{array}$$

The non-terminal A defines the set of composite numbers (see [Fisc68]), and S simply takes the complement of this set. It is interesting to investigate the power that negation adds to conjunctive macro grammars.

Properties of Conjunctive Macro Grammars: The most interesting questions regarding the extended macro grammars we have presented, concern their properties. For example, there does not seem to exist any obvious way to separate the conjunctive grammars from the macro conjunctive ones (this is a more general problem in the area of conjunctive languages, see [Okh04a] for details). Also, it is not obvious how broad the class of conjunctive macro grammars is.

An Infinite Hierarchy of Macro Grammars: Since macro grammars are in fact first-order, it seems possible to be able to extend them to higher-orders [Wad07] (pretty much as in the case of functional programming). It is interesting to investigate the expressive power of each level in this hierarchy (ie., the power of first-order (conjunctive) macro grammars, second order ones, and so on). It would be interesting if the various levels of the hierarchy corresponded to well-known complexity classes (but possibly this is too much to hope for).

Acknowledgments: The third author would like to thank Bill Wadge for introducing him to macro grammars and for many useful discussions on their properties.

References

[Aho68] Aho, A.: Indexed Grammars - An Extension of Context-Free Grammars. JACM **15(4)** (1968) 647-671.

[Fisc68] Fischer, M.: Grammars with Macro-Like Productions. FOCS (1968) 131–142

[Jez07] Jez, A.: Conjunctive grammars can generate non-regular unary languages. DLT (2007) (to appear).

[KNR06] Kountouriotis, V., Nomikos, Ch., Rondogiannis, P.: Well-Founded Semantics of Boolean Grammars. DLT (2006) 203–214.

[NR07] Nomikos, Ch., Rondogiannis, P.: Locally Stratified Boolean Grammars, LATA (2007).

[Okh01] Okhotin, A.: Conjunctive Grammars. Journal of Automata, Languages and Combinatorics **6(4)** (2001) 519–535.

[Okh04] Okhotin, A.: Boolean Grammars. Information and Computation **194(1)** (2004) 19–48.

[Okh04a] Okhotin, A.: An overview of conjunctive grammars. In: Paun, Rozenberg, Salomaa (Eds.), Current Trends in Theoretical Computer Science: The Challenge of the New Century, Vol. 2, World Scientific (2004) 545–566.

[Ten91] Tennent, R.: Semantics of Programming Languages. Prentice Hall (1991).

[Wad07] Wadge, W.: Personal Communication.

[Wro05] Wrona M.: Stratified Boolean Grammars. MFCS (2005) 801–812.

A constructive solution of the language inequation $XA \subseteq BX$

Olivier Ly

LaBRI – Bordeaux 1 University
351, cours de la Libération, 33405 Talence, France
ly@labri.fr

Abstract. We consider the inequation $XA \subseteq BX$ where A, B and X are formal languages, X is unknown. It has been proved in [9] that if B is a regular language then the maximal solution is also regular. However, the proof, based on Kruskal's Tree Theorem, does not give any effective construction of the solution. Here we give such an effective construction in the case where A and B are both finite and are such that $\max_{b \in B} |b| < \min_{a \in A} |a|$. Moreover, the complexity of our construction is elementary.

Keywords: Automata, Games, Language Equations

Introduction

Language equations arise in a natural way in computer science. Let us just think about Arden's lemma for instance, or context-free languages which are components of the least solutions of systems of polynomial equations.

However, even very simple questions may appear very difficult. For instance, one can think about the equation $XL = LX$ where X is unknown; this is the long-standing Conway problem which asks whether the maximal language commuting with a given rational language is also rational or not ([2], see also [6, 1, 4, 5]).

Many advances have been done in this domain this last few years ([6, 4, 5]). But Conway's problem has got a solution very recently, actually a negative solution. It has been proved in [11] (see also [3]) that there exists a *finite* language L such that the maximal solution of $XL = LX$ is not recursively enumerable even for some finite A and B. In addition, many natural classes of formal languages have got characterizations in terms of equations (see [12, 13]).

In [9], it has been proved that the maximal solution of $XA \subseteq BX$ is regular if B is regular, whatever A is. But the situation is tight: if one imposes to X to be contained in some given star-free language, then the maximal solution of $XA \subseteq BX$ can become non recursively enumerable (see [8]). This is a variation of the negative result of [11]. Besides, the proof that the maximal solution of $XA \subseteq BX$ is regular is based on the Kruskal's Tree Theorem (see [7]). It is non constructive, i.e., it does not give any effective construction of the maximal solution.

M. Kunc, A. Okhotin (Eds.): Theory and Applications of Language Equations
TUCS General Publication No 44, pp. 76–84, 2007.

In this article, we give such an *effective construction* in the case where A and B are both finite and are such that $\max_{b\in B} |b| < \min_{a\in A} |a|$: we set up an algorithm to construct an automaton recognizing the maximal solution of $XA \subseteq BX$. Moreover, the complexity of our algorithm is elementary.

Like in [9], our proof takes the point of view of games. We consider a game with two players: the attacker and the defender. Positions of the game are words. The game consists of a succession of turns as follows: first, the attacker chooses a word $a \in A$ and appends it to w, where w is the current position of the game. If $w.a$ has no prefix in B then the attacker wins and the game stops. Otherwise the defender chooses a prefix of $w.a$ which belongs to B, and cuts it from $w.a$, driving the game to a new position $b\backslash w.a$ for next turn. The defender wins if the game consists of infinitely many turns. Membership of the maximal solution of $XA \subseteq BX$ can be translated into the existence of a winning strategy for the defender (see [9]).

The main ingredient of our proof is a shrinking lemma for words having a winning strategy, it shall be detailled in the text (see Section 2). The hypothesis on lengths of words of A and B is used only for it.

The author wants to thank Professor G. Sénizergues for very helpful discussions; and the anonymous referee for indicating large simplifications of the proof.

Preliminaries

In all the paper, Σ is a finite alphabet. A and B are finite languages over Σ such that

$$\max_{b\in B} |b| < \min_{a\in A} |a|$$

Let w be a word, we denote by $|w|$ the length of w. Let v be a prefix (respectively a suffix) of w. We denote by $v\backslash w$ (respectively w/v) the unique word v' such that $w = vv'$ (respectively $w = v'v$).

The set of finite sequences of elements of A is denoted by T_A, this is the complete A-deterministic tree. "A-deterministic" because each node has one and only one son associated to each $a \in A$; the edge associated to this son can be considered as a-labeled. The empty sequence, i.e., the root, is denoted by ρ.

1 The Game of $XA \subseteq BX$

In a classical way, the equation $XA \subseteq BX$ can be translated into the game framework as follows.

We are supposed to be given with two languages A and B. We consider a game with two players: *Attacker* and *Defender* . The game consists in a possibly infinite sequence of turns. At the beginning of each turn, the position of the game is a word. One turn on a position w goes as follows:

1. *Attacker* chooses a word $a \in A$ and appends it to the right of w.

2. If no prefix of $w.a$ does belong to B, then *Defender* looses, the game stops and *Attacker* wins. Otherwise, *Defender* chooses a prefix b of $w.a$ belonging to B and erases it from $w.a$. And the game continues in the position $b\backslash w.a$.

So, *Attacker* wins if he manages to block *Defender* ; and *Defender* wins if the game consists of an infinite number of turns in which he never looses. We say that *Defender* has a winning strategy on a word w if he has a strategy which makes the game starting on w continue forever whatever *Attacker* does.

Lemma 1 (Equation and Game). *A word w belongs to the maximal solution of $XA \subseteq BX$ if and only if Defender has a winning strategy.*

Proof. See [10].

2 A Shrinking Lemma on *Attacker* 's Strategies

2.1 The Shrinking Lemma

Let us denote by S the set of strict prefixes of words of B, including the empty word. Let us consider a word w. We say that some $s \in S$ is *accessible through* w by *Defender* if w can be written as $b_1 ... b_n s$ where all the b_i's are words of B. The set of all elements of S which are accessible through w is called the *visibility* of *Defender* through w. It is denoted by $\mathrm{Vis}(w)$.

Remark 1. If $\mathrm{Vis}(w) = \varnothing$ then *Attacker* has a winning strategy on w. But the converse is false. Such a w is said to be *terminal*.

Definition 1 (B-relation). *Let us be given with two words w and w'. We say that w and w' are* B-related*, which is denoted by $w \leftrightarrow_B w'$, if there exist 4 words v_1, v_2, v_3 and v'_2 such that:*

- $w = v_1 v_2 v_3$ *and* $w' = v_1 v'_2 v_3$.
- *For any* $s \in Vis(v_1)$, $Vis(sv_2) = Vis(sv'_2)$.
- $|v_1| \geq N_1$.

where we define

$$N_1 = \ulcorner 2 \frac{(\min_{a \in A} |a|)(\max_{b \in B} |b|)}{\min_{a \in A} |a| - \max_{b \in B} |b|} (2^{|S|^2} + 1) \urcorner$$

Let us note that v_3 is superfluous in this definition. We just keep it for convenience of notations.

Remark 2. For any two words w and w', $w \leftrightarrow_B w'$ implies that $\mathrm{Vis}(w) = \mathrm{Vis}(w')$.

Lemma 2. *The relation $\leftrightarrow_B$ is a right congruence of finite index over the set of words of length greater than N_1.*

Proof. To see that $\leftrightarrow_B$ is a right congruence, only transitivity is not straightforward: Let $w \leftrightarrow_B w'$ and $w \leftrightarrow_B w''$. Let us note according to the previous notations:

- $w = v_1 v_2$ and $w' = v_1 v_2'$ (here we omit v_3 which is supposed to be added at the right of v_2).
- $w' = \bar{v}_1 \bar{v}_2$ and $w'' = \bar{v}_1 \bar{v}_2'$

Let us assume that $|\bar{v}_1| > |v_1|$. Therefore, v_1 is a prefix of w''. Let us pick some $s \in \mathrm{Vis}(v_1)$. Then $\mathrm{Vis}(sv_2) = \mathrm{Vis}(sv_2')$. Let $\bar{S} = \mathrm{Vis}(s.(v_1 \backslash \bar{v_1}))$. Then $\mathrm{Vis}(sv_2')$ is the union of all the $\mathrm{Vis}(\bar{s}\bar{v}_2)$ for $\bar{s} \in \bar{S}$. Besides, any $\bar{s} \in \bar{S}$ belongs also to $\mathrm{Vis}(\bar{v}_1)$. And so, by assumption, $\mathrm{Vis}(s)\bar{v}_2 = \mathrm{Vis}(s)\bar{v}_2'$. Thus, $\mathrm{Vis}(sv_2)$ is the union of all the $\mathrm{Vis}(\bar{s}\bar{v}_2')$ for $\bar{s} \in \bar{S}$, which is in turn equal to $\mathrm{Vis}(s(v_1 \backslash \bar{v_1}).\bar{v}_2')$. This proves that w and w'' are B-related with $v_2' = (v_1 \backslash \bar{v_1}).\bar{v}_2'$.

To verify that $\leftrightarrow_B$ is of finite index, it suffices to note that any sufficiently large word is equivalent to a shorter word, prefix of it, and of bounded length. To get that, one uses a simple counting argument, based on the fact that S is a finite set and therefore has a finite number of subsets.

Let us mention that the congruence is computable.

Let σ be a strategy for *Attacker* (respectively *Defender*) over a word w. A *finite σ-sequence of plays* in the game is a finite sequence of plays $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ where *Attacker* (respectively *Defender*) has played according to σ. This means that each a_i of the sequence has been chosen according to the previous b_j for $j < i$ and σ. The b_i are unspecified and variable.

A *strong strategy* for *Attacker* (respectively *Defender*) is a strategy in the game modified in such order that *Attacker* (respectively *Defender*) can play several words of A (respectively B) at the same turn. Formally, a strong strategy for *Attacker* (respectively *Defender*) is a strategy in the game defined by the pair (A^+, B) (respectively (A, B^+)) instead of (A, B). Let us note that if *Attacker* has a winning strong strategy, then he has a winning strategy, and this is the same thing for *Defender* . Indeed, provided with a winning strong strategy, one can win in the normal game by maintaining a (FIFO) queue of plays. We just use the concept of strong strategy in order to be more comfortable when describing winning strategies.

Lemma 3 (Shrinking Lemma). *Let us be given with two B-related words w and w', and a strategy σ for Attacker over w. Then there exists an integer L and a strong strategy σ' for Attacker over w' with the following property: Whatever the plays of Defender, by following σ', in less than L turns:*

- *Either Attacker wins*
- *Or he drives the game from w' to a new word v' such that there exists a non-void finite σ-sequence of plays driving the game from w to a new word v which is B-related to v'.*

The integer L and σ' depend on σ, w and w'.

Before going into its proof, let us state the main consequence of this result:

Theorem 1. *Let w and w' be two B-related words. Then Attacker has a winning strategy over w if and only if he has one on w'.*

Proof. Indeed, let us suppose that *Attacker* has a winning strategy over w. We can construct a winning strategy over w' as follows:

Lemma 3 provides us a strategy σ' and an integer L. Let *Attacker* start playing according to this strategy. According to the Lemma, after a finite number of turns, less than L:

- Either *Attacker* wins. That is what we wanted and σ' stops here.
- Or else, he drives the game to a word v'_1 and the lemma provides us σ-sequence of plays driving the game from w to a new word v_1 which is B-related to v'_1.

The strategy σ is still winning over this new word v_1, we then start again the process with the words v_1 and v'_1. And so on.

Following that, we construct a sequence $v_1, v_2, \dots, v_k, \dots$ of words which are positions of a play in the game where *Attacker* follows σ. Let us note that each pair v_i, v_{i+1} are separated by at least one turn, and in fact, several turns. In particular, when the game arrives to the word v_k, at least k turns have been played. Besides, let us observe that σ, as a winning strategy of *Attacker*, is finite. This implies that there exists an integer L_σ such that *Attacker* wins for sure in less than L_σ turns from w. Therefore, k is also bounded by L_σ. This implies that our process stops after at most L_σ cycles, which means that *Attacker* wins within at most L_σ cycles.

In the proof of Lemma 3 we need the following concept:

Definition 2 (Waiting Loop). *Let w be a word. We define a* waiting loop *to be a decomposition $w = w_1w_2w_3w_4$ of w in 4 factors such that for any $s \in Vis(w_1)$, $Vis(sw_2) = Vis(sw_2w_3)$ and w_3 not empty.*

Lemma 4 (Waiting Loops and B-Relation). *Let w be a word, and let $w = w_1w_2w_3w_4$ be a waiting loop such that $|w_1| \geq N_1$. Then w is B-related to any word of $w_1w_2w_3^*w_4$.*

Proof. So, let $w' = w_1w_2w_3^kw_4$ for some integer k. According to the notations of Definition 1, let us define $v_1 = w_1$, $v_2 = w_2w_3$, $v'_2 = w_2w_3^k$ and $v_3 = w_4$. First, let us note that $|v_1| > N_1$ is true, it is an hypothesis of the lemma.

We prove that for any $s \in \mathrm{Vis}(v_1)$, $\mathrm{Vis}(sv_2) = \mathrm{Vis}(sv'_2)$ by induction on k. For $k = 0$, there is nothing to prove: this is the hypothesis of the lemma. Let us thus suppose that it is true for some $k \geq 0$. By induction, we just have to prove that for any $s \in \mathrm{Vis}(v_1)$, $\mathrm{Vis}(sw_2w_3^k) = \mathrm{Vis}(sw_2w_3^{k+1})$.

So let $s \in \mathrm{Vis}(v_1)$. Let $s' \in \mathrm{Vis}(sw_2w_3^k)$. Let us show that $s' \in \mathrm{Vis}(sw_2w_3^{k+1})$. Let $b_1, \dots, b_n \in B$ be such that $b_1 \dots b_n s' = sw_2w_3^k$. Let n' be the greatest index such that $b_1 \dots b_{n'}$ is a prefix of sw_2. And let $s'' = b_1 \dots b_{n'} \backslash sw_2$; s'' belongs

to $\mathrm{Vis}(sw_2)$. Besides, by assumption, $\mathrm{Vis}(sw_2) = \mathrm{Vis}(sw_2w_3)$. Therefore, there exists $b'_1, \ldots, b'_{n''}$ such that $b'_1 \ldots b'_{n''} \backslash sw_2w_3 = s''$. Finally, we obtain that

$$b'_1 \ldots b'_{n''} b_{n'+1} \ldots b_n s' = sw_2w_3w_3^k = sw_2w_3^{k+1}$$

which means that $s' \in \mathrm{Vis}(sw_2w_3^{k+1})$. That is what we wanted. The converse is similar.

Existence of waiting loop is given by the following simple result based on a simple counting argument based on the fact that visibility sets are subsets of S.

Lemma 5 (Existence of waiting loops, Version 1). *For any word w and any prefix w_1 of w such that the length of $w_1\backslash w$ is greater than $2^{|S|^2}$, there exists a waiting loop of form $w = w_1w_2w_3w_4$.*

We actually will use a more precise version:

Lemma 6 (Existence of waiting loops, Version 2). *Let w be a word, and let $w = c_1c_2 \ldots c_n$ be a decomposition of w into n factors, where the c_i's are words. Let n_1 be such that $n - n_1 \geq 2^{|S|^2}$. Then w has a waiting loop $w = w_1w_2w_3w_4$ such that $w_1 = c_1c_2 \ldots c_{n_1}$ and the other w_i's are concatenations of some c_i's. Formally: for $i = 1, \ldots, 4$, $w_i = c_{n_{i-1}+1} \ldots c_{n_i}$, where n_0, n_2, n_3 and n_4 are such that $0 < n_1 < n_2 < n_3 \leq n$, $n_0 = 0$ and $n_4 = n$.*

Proof (of Lemma 3).

We describe the strategy σ' from w' turns after turns. In order to do that, we describe a game $\mathcal{G}'$ where *Defender* 's plays are generic, and doing that we describe turns after turns how *Attacker* has to play. During this description, we shall use σ as an *oracle* to which we provide plays of *Defender* and which tells us what σ suggests for *Attacker* 's plays.

First of all, let us observe that *Defender* must play at least $\llcorner \frac{N_1}{\max_{b\in B}|b|} \lrcorner$ turns before completely erasing v_1 (here we keep notations of Definition 1 where $w = v_1v_2v_3$ and $w' = v_1v'_2v_3$). And besides, from the definition of N_1 and the fact that $\min_{a\in A}|a| > \max_{b\in B}|b|$ we get:

$$\frac{N_1}{\max_{b\in B}|b|} \geq \frac{N_1}{\min_{a\in A}|a|} + 2(2^{|S|^2}+1) \tag{1}$$

Let us define

$$N'_1 = \ulcorner \max_{b\in B}|b|(2^{|S|^2}+1) \urcorner$$

There are 3 stages in the strategy:

1. Informally, the first stage starts at the beginning and goes on until the word obtained by concatenating all plays of *Attacker* is sufficiently long, actually more than $N_1 + N'_1$.

According to the preliminary remark, during this stage, plays of *Defender* remains into v_1, because v_1 is supposed to be great enough (see below). For

these plays thus, there is no difference between w and w' which both have v_1 as a common prefix. Then $\mathcal{G}'$ can be considered as a game $\mathcal{G}$ on w and *Attacker* follows σ.

So, precisely, the first stage consists of the n_1 first plays of the game $(a_1, b_1), (a_2, b_2), \dots, (a_{n_1}, b_{n_1})$ where n_1 is such that $|a_1 a_2 \dots a_{n_1 - 1}| \leq N_1 + N_1' < |a_1 a_2 \dots a_{n_1}|$. Let us note that n_1, i.e., the moment at which Stage 1 ends, depends on the plays of *Defender* and on σ which tells *Attacker* how to play. However, we can say that $n_1 \leq (N_1 + N_1')/\min_{a \in A} |a| + 1$. This implies that

$$n_1 \leq \frac{N_1}{\min_{a \in A} |a|} + \frac{\max_{b \in B} |b|}{\min_{a \in A} |a|}(2^{|S|^2} + 1) \leq \frac{N_1}{\min_{a \in A} |a|} + 2^{|S|^2} + 1$$

Together with Equation 1 of the preliminary remark, one can conclude that v_1 has not been totally erased, and even more than that: It remains at least $2^{|S|^2} + 1$ turns before that, this means that there exists a word v of length greater than $\max_{b \in B} |b|(2^{|S|^2} + 1)$ such that $v_1 = b_1 b_2 \dots b_{n_1} v$. In particular, this justifies the fact that *Attacker* can use σ to play during this stage.

2. In Stage 2, *Attacker* looks for *a waiting loop*. Formally: *Attacker* plays according to σ until Turn n_2 such that there exists n_2' such that

$$[w a_1 \dots a_{n_1}][a_{n_1 + 1} \dots a_{n_2'}][a_{n_2' + 1} \dots a_{n_2}]\varepsilon$$

is a waiting loop. We choose n_2 to be minimal for this property. Thanks to Lemma 6, because the length of v is greater than $\max_{b \in B} |b|.(2^{|S|^2} + 1)$, we are sure that n_2 occurs before v has been totally erased. In particular, *Attacker* can still use σ to play during this stage.

3. During Stage 3, *Attacker* no longer follows σ. He plays the sequence $a_{n_2' + 1}, \dots, a_{n_2}$ in loop until *Defender* has almost erased v_2', i.e., until Turn n_3 which is such that $(b_1 \dots b_{n_3}) \backslash v_1.v_2' \in \mathrm{Vis}(v_1.v_2')$ where $b_{n_2 + 1}, \dots, b_{n_3}$ are all the plays of *Defender* during Stage 3. In the following, $(b_1 \dots b_{n_3}) \backslash v_1.v_2'$ is denoted by s'. Let us note that at this point, *Attacker* may be inside the loop, i.e., he maybe playing some a_i with $n_2' + 1 \leq i < n_2$. Then whatever the plays of *Defender* , *Attacker* finishes the current loop. This drives the game to some Turn n_4 such that $a_{n_4} = a_{n_2}$.

Let us note that while *Attacker* is finishing his loop, which takes at most $2^{|S|^2} + 1$ turns, it may happen that *Defender* erases v_3 and starts erasing the first plays of *Attacker* , i.e., the plays of Stage 1. However, Stage 1 above ensures that the first n_1 plays of *Attacker* make a word of length greater than $N_1 + N_1'$. Therefore, *Defender* must leave at least a word of length N_1 from $a_1 a_2 \dots a_{n_1}$.

Now, the succession of plays which have been done is

$$(a_1, b_1), (a_2, b_2), \dots, (a_{n_1}, b_{n_1}), \dots, (a_{n_2'}, b_{n_2'}), \dots$$

$$\dots (a_{n_2}, b_{n_2}), \dots, (a_{n_3}, b_{n_3}), \dots, (a_{n_4}, b_{n_4}).$$

Let n_3' be the greatest integer such that $n_2 \leq n_3' \leq n_3$ and $(b_1 \dots b_{n_3'})\backslash v_1 \in \mathrm{Vis}(v_1)$. In the following $(b_1 \dots b_{n_3'})\backslash v_1$ is denoted by s. We have that $(b_{n_3'+1} \dots b_{n_3})\backslash s.v_2' = s'$. Therefore $s' \in \mathrm{Vis}(s.v_2')$. Besides, by the definition of B-relation, $\mathrm{Vis}(s.v_2') = \mathrm{Vis}(s.v_2)$. Therefore $s' \in \mathrm{Vis}(s.v_2)$, and thus there exist $\bar{b}_1, \dots, \bar{b}_m$ such that $\bar{b}_1 \dots \bar{b}_m s' = s.v_2$.

Now, let us consider the game $\mathcal{G}$ on w defined as follows: in this game, we consider the sequence of *Defender* 's plays defined by

$$b_1, \dots, b_{n_2}, \dots, b_{n_3'}, \bar{b}_1, \dots, \bar{b}_m, b_{n_3+1}, \dots, b_{n_4}$$

Let us consider the sequence of *Attacker* 's plays corresponding to it according to σ: $\bar{a}_1, \dots, \bar{a_{m'}}$ where $m' = n_3' + m + n_4 - n_3 + 1$. The first n_2 plays $\bar{a}_i$'s are exactly the a_i's that we have been just defined for σ' in Stages 1 and 2, since in these stages, *Attacker* actually used σ.

Let us go back to the definition of σ' on the game $\mathcal{G}'$. Let us recall that we are at Turn $n_4 + 1$, and *Attacker* is going to play. We define his play to be the concatenation of $\bar{a}_{n_2+1} \dots \bar{a}_{m'}$, let us denote it by a_{n_4+1}. Let us recall that we define σ' as a strong strategy. This means that *Attacker* plays in $\mathcal{G}'$ just like if it were in $\mathcal{G}$. Let b_{n_4+1} be the next play of *Defender* .

Now, let us observe that if w' has not been totally erased, it remains the same word in $\mathcal{G}$ and in $\mathcal{G}'$ to complete the pass, which is $(b_1 \dots b_{n_4} b_{n_4+1})\backslash w'$, let us denote it by u.

The description of σ' ends here. To conclude, let us remark that if at some point, *Defender* cannot play any more because the current word has no prefix in B, then *Attacker* wins the game and σ' ends.

It remains to show that the positions of the games $\mathcal{G}$ and $\mathcal{G}'$ are B-related words. To see that, we apply Lemma 4.

Let us suppose that w' has not been totally erased and that it remains the word u defined as above. Altogether, we get for $\mathcal{G}'$ a position of form:

$$\overbrace{u a_1 a_2 \dots a_{n_1}}^{w_1} \overbrace{a_{n_1+1} \dots a_{n_2'}}^{w_2} \dots$$

$$\dots \overbrace{(a_{n_2'+1} \dots a_{n_2}).(a_{n_2'+1} \dots a_{n_2}) \dots (a_{n_2'+1} \dots a_{n_2})}^{w_3^*} \overbrace{\bar{a}_{n_2+1} \dots \bar{a}_m'}^{w_4} \tag{2}$$

where $\bar{v}_2' \in a_{n_1+1} \dots a_{n_2'} (a_{n_2'+1} \dots a_{n_2})^+$. In $\mathcal{G}$ we get a position of form:

$$\underbrace{u a_1 a_2 \dots a_{n_1}}_{w_1} \underbrace{a_{n_1+1} \dots a_{n_2'}}_{w_2} \underbrace{a_{n_2'+1} \dots a_{n_2}}_{w_3} \underbrace{\bar{a}_{n_2+1} \dots \bar{a}_m'}_{w_4} \tag{3}$$

The words w_1, w_2, w_3 and w_4 defined above satisfy by construction the condition of Lemma 4. They have been indeed chosen just in order to construct a waiting loop. In addition, as we have seen during the description of σ', if w' has been totally erased, then this part remaining from $a_1 a_2 \dots a_{n_1}$ is still of length greater than N_1.

To conclude the proof, we have to give a bound on the number of turns which have been done. Observe that during the 3 stages of plays, w' can be erased and at most $2^{|S|^2} + 1$ turns can be played after (in order to end the loop). This gives the following bound:

$$L = \frac{|w'|}{\min_{b \in B} |b|} + 2^{|S|^2} + 1$$

3 Effective construction of the solution

To conclude, we consider the right congruence of Lemma 2 over words of length greater than N_1. It can be extended easily to a right congruence over all words by setting any word of length less than N_1 equivalent to itself only. One gets a new right congruence which still is of finite index. By Theorem 1, any two congruent words both belong in the greatest solution of $AX \subset XB$ or both do not.

Let us consider an automaton associated to it; and let us select the largest subset of states of this automaton such that if we consider this set as the set of final states; then we get a language which is solution of $AX \subset XB$.

References

1. C. Choffrut, J. Karhumäki, and I. Petre. The Commutation of Finite Sets: A Challenging Problem. *Theoretical Computer Science*, 273:69–79, 2002.
2. J.H. Conway. *Regular Algebra and Finite Machines.* Chapman and Hall, 1971.
3. J. Karhumäki, M. Kunc, and A. Okhotin. Computing by Commuting. *Theoretical Computer Science*, 356(1-2):200–211, 2006.
4. J. Karhumäki, M. Latteux, and I. Petre. Commutation with Codes. *Theoretical Computer Science*, 340(1), 2005.
5. J. Karhumäki, M. Latteux, and I. Petre. Commutation with Ternary Sets of Words. *Th. Comput. Syst.*, 38(2), 2005.
6. J. Karhumäki and I. Petre. Conway's Problem for Three-Word Sets. *Theoretical Computer Science*, 289:705–725, 2002.
7. J.B. Kruskal. Well-Quasi-Ordering, the Tree Theorem and Vazsonyi's Conjecture. *Trans. Amer. Math. Soc.*, 95:210–225, 1960.
8. M. Kunc. On Language Inequalities $XK \subseteq LX$. In *Developments in Language Theory (DLT)*. Springer, 2005.
9. M. Kunc. Regular Solutions of Language Inequalities and Well Quasi-Orders. *Theoretical Computer Science*, 348(2-3):277–293, 2005.
10. M. Kunc. Simple Language Equations. *Bulletin of the European Association for Theoretical Computer Science*, 85:81–102, 2005.
11. M. Kunc. The Power of Commuting with Finite Sets of Words. In *LNCS*, volume 2404, 2005.
12. A. Okhotin. Decision Problems for Language Equations. In *Internat. Colloq. on Automata, Languages and Programming (ICALP)*, pages 239–251. LNCS 2719, 2003.
13. A. Okhotin. A Characterization of the Arithmetical Hierarchy by Language Equations. In *Descriptional Complexity of Formal Systems (DCFS)*, pages 225–237, 2004.

Multi-conjunctive grammars

David Zook[1]

LogicBlox, Atlanta GA 30309, USA,
david.zook@logicblox.com,
WWW home page: http://www.logicblox.com

Abstract. Context-Free Grammars (CFGs) are the current linqua franca for the formal expression of programming language syntax. However every programming language also has *context-sensitive* requirements, which are typically expressed as custom-coded actions added to the grammar. *Conjunctive Grammars* introduce conjunction into CFGs while maintaining cubic worst-case parsing time. Here we present an elegant generalization of Conjunctive Grammars called *Multi-Conjunctive Grammars* (MCGs) which range over fixed-width *tuples* of strings. MCGs provide context-sensitive parsing while remaining amenable to the well-known (suitably generalized) null- and unit-elimination techniques.

1 Introduction

A *Context-Free Grammar* (CFG) G defines a (possibly infinite) set of strings $\mathbf{L}(G)$ over some finite alphabet $\sum$. CFGs have many applications, including specifying the syntax of programming languages. A variety of tools are available for automatically converting a CFG G into an executable *parser* $\mathbf{P}_G(x)$, where $\mathbf{P}_G(x) = \texttt{true}$ exactly when the string $x \in \mathbf{L}(G)$. The CFG shown in Figure 1 defines the set of even-length strings over the alphabet {`a,b`} (a formal syntax and semantics of CFGs if given below). For this grammar, $\mathbf{L}(G) = \{$ `""`, `"aa"`, `"ab"`, `"ba"`, `"bb"`, `"aaaa"`, `...` $\}$.

CFGs cannot express context-sensitive properties such as string equality, which are essential to defining programming language syntax (for example, in Java the name of a class's constructor must match the name of the class). A wide variety of enhancements to the CFG formalism have been proposed. In particular, Okhotin has succeeded in introducing conjunction (*Conjunctive Grammars*) [2001] and negation (*Boolean Grammars*) [2004] in such a way that a parser still is able to check a string with worst-case time complexity (n^3) – no worse than required for a CFG.

Conjunctive Grammars can express string equality but the required formula is non-obvious and therefore hard to use for ordinary language specifiers. Here we explore a more general approach, called *Multi-Conjunctive Grammars* (MCGs), based on the observation that equality can be naturally expressed using CFG-like techniques against **tuples** of strings. For example, the grammar shown in Figure 2 defines string equality (without conjunction) over the alphabet {`a,b`} (ε is the empty string). In the example, each non-terminal N has a finite fixed *arity*

M. Kunc, A. Okhotin (Eds.): Theory and Applications of Language Equations
TUCS General Publication No 44, pp. 85–100, 2007.

k, indicating that $\mathbf{L}(N^k) \subseteq \sum^k$ (We will often write the arity as a superscript on the non-terminal although this is not part of MCG syntax). In particular, the non-terminal E has $k = 2$, as indicated by the form of its rule bodies, where angle-brackets enclose two components separated by a comma. Also, an important difference from CFG semantics is illustrated by the body of second rule – the two occurrences of E, E.1 and E.2, are **not** independent, as they would be in a CFG. Instead, they represent the first and second components (respectively) of a **single** 2-tuple from $\mathbf{L}(\mathtt{E})$. This feature is the foundation of the context-sensitive properties of MCGs.

MCGs can express properties that go well beyond the capabilities of Conjunctive Grammars and Boolean Grammars. One example is whole number addition of natural numbers, as shown in Figure 3 (natural numbers are represented as strings of '1's). In this example, conjunction is not even required. Using conjunction, even more complex grammars are possible, including whole number multiplication, as shown in Figure 4. (In the example, conjunction with E (equality) is used in the definition of M to ensure that the number that gets "added" to the end of the third component is equal to one of the other two components.)

A → ε
A → A a a
A → A a b
A → A b a
A → A b b

Fig. 1. CFG for Even-Length Strings

E → ⟨ ε , ε ⟩
E → ⟨ E.1, a, E.2, a ⟩
E → ⟨ E.1, b, E.2, b ⟩

Fig. 2. MCG Example 1: String Equality

S → ⟨ ε, ε, ε ⟩
S → ⟨ S.1 1, S.2, S.3 1 ⟩
S → ⟨ S.1 , S.2 1 , S.3 1 ⟩

Fig. 3. MCG Example 2: Addition

M → ⟨ ε, ε, ε ⟩
M → ⟨ M.1 1, M.2 & E.1, M.3 E.2 1 ⟩
M → ⟨ M.1 & E.1, M.2 1, M.3 E.2 1 ⟩

Fig. 4. MCG Example 2: Multiplication

2 Background

Over the last several decades, interest has grown in formalisms that capture common complexity classes *intrinsically*, in the sense that the formalism does not explicitly include a limitation on time or space resources. In particular, formalisms intrinsically capturing PTIME and LOGSPACE have been discovered. For example, in the realm of programming languages, Neil Jones has presented F_{ro} [1999], a first-order, cons-free functional language, which captures PTIME; as well as the tail-recursive version F_{rotr}, which captures LOGSPACE. In the realm of database query languages, the logic programming language Datalog+ was shown by Papadimitriou [1985] to capture PTIME, and a restricted version was shown by Graedel to capture LOGSPACE [1992]. In the realm of type systems, Hoffman has shown that Linear types capture PTIME [2003].

In the realm of grammars, Groenink [1995] has shown that the SLMG's capture PTIME. This result is used below to show that MCG's, which have a certain structural similarity to SLMG's, also capture PTIME. We are not aware of any grammatical formalism that captures LOGSPACE. Furthermore, we are not aware of any attempt to apply null- and unit-elimination techniques to SLMG's (or to F_{ro}, Datalog+. or Linear types for that matter). We believe that the MCG formalism makes applying these techniques relatively straightforward, because the MCG syntax and semantics are straightforward generalizations from CFG syntax and semantics.

3 Context-Free Grammars

We review the well-known facts of context-free grammars. To facilitate the desired enhancements to the CFG syntax and semantics (and also because it is interesting in its own right), we depart from the usual constructive description of CFG semantics and provide a *declarative* description founded on first-order logic. This foundation enables establishing some *grammar re-writing rules*, which in turn shorten and illuminate proofs of various grammar properties.

3.1 Syntax

The syntax of a *context-free grammar* (CFG) starts with a tuple $G = (\mathbf{N}, \Sigma, \mathbf{P}, N_S)$ where: 1) $\mathbf{N}$ is a finite set of symbols called the *non-terminal symbols*; 2) Σ is the *alphabet* of G – a finite set of symbols (called the *terminal* symbols) disjoint from N; 3) $\mathbf{P}$ is a set of *production rules*; and 4) N_S is an

element of **N** called the *start* symbol. Since the details of rule syntax will be important below, we describe the CFG syntax as a CFG itself, as shown in Figure 5 (The rules for N (non-terminal symbol) and T (terminal symbol) are omitted for brevity).

In addition, it should be noted that:

- The bodies of all rules with the same head are grouped together into a *disjunction* (each sequence in the disjunction is an *alternative*)
- Terminal symbols are enclosed in single-quotation marks;
- A distinction is made between the appearance of a non-terminal symbol in the *head* of a rule and the *reference* to it in the rule's *body*.

In what follows, we use G, P, D, S, N, X, or R to designate an arbitrary grammar, production rule, disjunction, sequence, non-terminal symbol, terminal symbol, or non-terminal reference, respectively. Also, the general term *grammar expression* (or just "expression") is used for either a disjunction, a sequence, a terminal symbol or a non-terminal reference.

G → P | P G P → N ' → ' D (*Each N heads exactly one rule.*) D → S | S ' | ' D S → U | U S U → R | X | ε R → N
G = Grammar P = Production Rule D = Disjunction S = Sequence U = Symbol R = Reference to a non-terminal N = Non-terminal symbol (any capital letter) X = Terminal symbol (in single-quotes) ε = Empty string

Fig. 5. Syntax of a Context-Free Grammar

3.2 Semantics

The intention of a context-free grammar is to define a set of strings $\mathbf{L}(G) \subseteq \Sigma^*$, called the *language* of G. Traditionally, the language of a CFG is defined "constructively" – a string x is in $\mathbf{L}(G)$ whenever x can be derived from the

start symbol N_S by a finite number of substitutions of one of the non-terminal references N with one of the alternatives from N's rule. Here, we present the semantics "declaratively", as a list of *semantic equations*, as shown in Figure 6. Each construct within the grammar is given a grammar-specific language (set of strings), defined in turn in terms of the languages of other constructs.

The well-known flaw in this declarative approach is that Equation S-IV introduces a circularity, making straightforward structural induction fail. However, the "naive" equations in Figure 6 can be straightforwardly converted into a set of axioms of a first-order logic (FOL) $\mathbf{M}(G)$. (A translation function from a CFG to a FOL is defined in Figure 7.) As is well-known, a FOL may have zero, one, or more than one *satisfying interpretations*, called *models*. A FOL is *canonical* over some domain if it has exactly one model over that domain. So the challenge (using the language of logic) is to to prove that a given CFG G that $\mathbf{M}(G)$ is canonical over the domain $\sum^*$ (we assume that $+$ has the usual meaning of string concatenation). We call such a grammar *well-defined*, because the existence of a unique model makes the question of the membership of a string x in $\mathbf{L}(G)$ unambiguous. Proofs of well-definedness can be accomplished for some grammars (but not all) using the well-known techniques of null- and unit-elimination. In what follows, properties of these two techniques are proven using *grammar refactoring rules*, whose correctness is proven using standard logical deduction. The logical inference rules allow rewriting a FOL without changing the model(s) which satisfy it.

S-I.	$\mathbf{L}(G)$	$= \mathbf{L}^G(N_S)$
S-II.	$\mathbf{L}^G(N \to D)$	$= \mathbf{L}^G(D)$ where $N \to D$ is a rule in G
S-III.	$\mathbf{L}^G(S \mid D)$	$= \{s \mid s \in \mathbf{L}^G(S) \text{ or } s \in \mathbf{L}^G D\}$
S-IV.	$\mathbf{L}^G(U\ S)$	$= \{s \mid s = u + v$ $u \in \mathbf{L}^G(U), v \in \mathbf{L}^G(S)\}$
S-V.	$\mathbf{L}^G(R)$	$= \mathbf{L}^G(R \to D)$ where $R \to D$ is a rule in G
S-VI.	$\mathbf{L}^G(X)$	$= \{X\}$
S-VII.	$\mathbf{L}^G(\varepsilon)$	$= \{\varepsilon\}$

Fig. 6. CFG Naive Semantics

3.3 Grammar Refactoring Rules

A *grammar refactoring rule* allows a grammar to be modified without changing the model(s) of its logical semantics. Clearly, if a grammar G can be refactored into a new grammar G' which is provably canonical, then G is also canonical, and therefore well-defined. Some basic grammar refactoring rules are shown in

$$
\begin{array}{lll}
\text{M-I.} & \mathbf{M}(P_1..P_n) & = \mathbf{M}(P_1) \wedge .. \wedge \mathbf{M}(P_n) \\
\text{M-II.} & \mathbf{M}(N \rightarrow D) & = < \forall x :: N(x) \leftrightarrow \mathbf{M}(x, D) > \\
\text{M-III.} & \mathbf{M}(S_1 \,|\, .. \,|\, S_n) & = \mathbf{M}(S_1) \vee .. \vee \mathbf{M}(S_n) \\
\text{M-IV.} & \mathbf{M}(U_1..U_n) & = < \exists y_1, .., y_m :: x = y_1 + .. + y_n \wedge \\
 & & \quad \mathbf{M}(1, U_1) \wedge .. \wedge \mathbf{M}(m, U_m) > \\
\text{M-V.} & \mathbf{M}(i, R) & = R(y_i) \\
\text{M-VI.} & \mathbf{M}(i, X) & = (y_i = X) \\
\text{M-VII.} & \mathbf{M}(\varepsilon) & = (y_i = \varepsilon)
\end{array}
$$

Fig. 7. CFG Logical Semantics

Figure 8. The correctness of these rules follows directly from the logical semantics equations of Figure 7 using standard logical deduction.

$$
\begin{array}{lll}
\text{R-I.} & (E_1|E_2) & \Rightarrow (E_2|E_1) \\
\text{R-II.} & (E\ \varepsilon) & \Rightarrow E \\
\text{R-III.} & (\varepsilon\ E) & \Rightarrow E
\end{array}
$$

Fig. 8. CFG Refactoring Rules

3.4 Null-Elimination

Definition 1.1 *Two grammars G and H are* equivalent *(denoted $G \equiv H$) iff $\mathbf{L}(G) = \mathbf{L}(H)$. Likewise, two expressions (that is, disjunctions, sequences or references) E and F are* equivalent in grammar G *(denoted $E \equiv^G F$ iff $\mathbf{L}^G(E) = \mathbf{L}^G(F)$. If E and F are equivalent in every grammar, then they are just* equivalent *(denoted $E \equiv F$.)*

Definition 1.2 *A nonterminal A is ε-free in a grammar G iff $\mathbf{L}^G(A)$ does not contain ε.* Likewise, *G is ε-free iff every non-terminal in G except for the start symbol is ε-free in G.*

Theorem 1.2 *For every CFG G, there is an ε-free CFG $\widehat{G}$ where $L(G) = L(\widehat{G})$.*

Proof For any expression E in G, define $\widehat{E}$ to be an expression (if one exists) such that $\mathbf{L}(\widehat{E}) = \mathbf{L}(E) - \{\varepsilon\}$. The grammar refactoring rules in Figure 9 allow an expression $\widehat{E}$ to be rewritten to "push the tilde down" to the subexpressions of E. The refactoring rules completely cover the CFG syntax. So, for any rule $A \rightarrow E$ in G, there is a new rule $\widehat{A} \rightarrow F$ where $\mathbf{L}^G(\widehat{A}) = L^G(A) - \{\varepsilon\}$ and tildes appear only on non-terminals in the body of F. Now let $\widehat{G}$ contain such a rewritten rule for *every* rule in G. In addition, let the first rule in $\widehat{G}$ be either

$A_S \rightarrow \widehat{A_S}$ if $\varepsilon \in L(G)$; or $A_S \rightarrow \widehat{A_S}|\varepsilon$ otherwise. So $\widehat{G}$ is equivalent to G but also ε-free. QED.

This proof is constructive, and so it represents a guaranteed-to-terminate procedure for *null-elimination*.

N-III.	$\widehat{S_1\|..\|S_n}$	$\Rightarrow \widehat{S_1}\|..\|\widehat{S_n}$
N-IV.	$S_1\|..\|U_1..\widehat{U_iS}\|..S_n$	$\Rightarrow S_1\|..\|U_1..\widehat{U_i}$
		$\|U_1..\widehat{S}$
		$\|U_1..\widehat{U_i}\widehat{S}\|..\|S_n$
N-V.	$\widehat{N}$	$\Rightarrow \widehat{E}$, where $\widehat{N} \rightarrow \widehat{E}$ is in G
N-VI.	$\widehat{X}$	$\Rightarrow X$
N-VII.	$\widehat{\varepsilon}$	$\Rightarrow \varnothing$

Fig. 9. CFG Null-Elimination Refactoring Rules

3.5 Unit-Elimination

Definition 1.3 *In a CFG G, A* unit *is a sequence consisting of just one nonterminal.* A grammar is *unit-free* iff none of its rules contain any units. In any CFG G, a rule containing a unit has the form $N \rightarrow S_1..|N'|..S_n$, and also G must contain a rule $N' \rightarrow S'_1|..|S'_{n'}$. Either (Case 1) N' is different from N or (Case 2) it is the same. If different, then the body of N''s rule may be substituted for the reference to N' in the body of N's rule using the first refactoring rule in Figure 10. Then if any other units remain in G, the step can be repeated.

However, when N' is actually the same as N, then the rule with the unit is of the form: $N \rightarrow S_1|..|N|..S_n$. The logic semantics for this rule is of the form: $N(s) \equiv .. \vee N(s) \vee ...$ However, in any model of G, *any* set of strings can be assigned to the predicate $N(s)$ without endangering the satisfactoriness of the model. Therefore, G is not well-defined in this case. Interestingly, by contrast, the constructive semantics for CFG's would allow N to be replaced with $\varnothing$, thereby choosing one particular model (the minimal one). The constructive approach has the advantage that every CFG can be transformed to have a unique model. However from a declarative perspective, this choice of a model is arbitrary, and so we terminate our unit-elimination procedure at this point.

Regardless of whether the grammar turns out to be well-defined, we want the unit-elimination procedure to always terminate. To see that the procedure must terminate, define a finite relation $uses^G(N_1, N_2)$, which is true iff the nonterminal N_2 is referenced by the body of N_1's rule in grammar G. There must also exist finite uniquely defined relations $dependsOn^G(N_1, N_2)$ – the transitive closure of $user^G$, and $xhortestPath^G(N, N')$ which gives a unique positive integer representing the number of edges in the shortest path (if any) from N to N'.

Clearly, $uses^G$ can be thought of as a finite directed graph. In particular, in Case 1 above, there is an edge $uses^G(N, N')$; and in Case 2, there is an edge $uses^G(N, N)$. The substitution step described above shortens the length of all paths containing the edge $uses^G(N, N')$ by one, without increasing the length of any paths. Since there are a finite number of edges and paths in the graph, the procedure always terminates (although some grammars may prove to be not well-defined). QED.

U-I. $N \rightarrow S_1|..|N'|..|S_n \Rightarrow^G N \rightarrow S_1|..|S'_1|..|S'_k|..|S_n$
$N' \rightarrow S'_1|..|S'_k$

Fig. 10. CFG Unit-Elimination Refactoring Rules

3.6 Parser Generation

The value of the null- and unit-elimination techniques is highlighted by the following theorem:

Theorem 1.4 *Every ε-free and unit-free CFG is well-defined.*

Proof (sketch) Since the details are messy and the ideas used to establish this theorem are well-accepted, we present only a sketch of the proof. The approach to the proof is to use induction, but the challenge, as mentioned above, is to properly found the induction. So, the following cases can be considered:

1. **Base Case:** Clearly, rules M-VI, M-VII and M-VIII (Figure 7 clearly have a unique model in any grammar.
2. **Non-recursive:** Call a grammar G *recursive* if $dependsOn^G(N, N) = true$, for any non-terminal N. If G is not recursive, then $uses^G$ forms a tree rooted on the start symbol, and the induction can be founded starting at the leaves and working up toward the root, which is the start symbol.
3. **Non-unitary** Let $\mathbf{L}^n_G(N)$ be the set of strings of length n in $\mathbf{L}_G(n)$. If a graph G is both ε-free and unit-free, then for any rule $N \rightarrow E$, the set $\mathbf{L}^n_G(N)$ is uniquely defined in terms of $\mathbf{L}^{n-1}_G(N)$. So the uniqueness of the model can be established for each n, and therefore for any set of finite-length strings.

For a well-defined CFG G, a parser may be generated in a target programming language using well-known techniques.

4 Conjunctive Grammars

Having laid the groundwork above, the syntax and semantics for Conjunctive Grammars can be reproduced in declarative style.

4.1 Syntax

Conjunctive grammars allow a new binary operator: conjunction ('&'). For any string x, $x \in L^G(S_1 \& S_2)$ whenever $x \in L^G(S_1)$ and $x \in L^G(S_2)$. In the syntax, the precedence of the conjunction operator lies below that of disjunction but above that of sequences. The CFG syntax can be modified for conjunction by changing one production rule and adding another one, as indicated in Figure 11 (compare to Figure 5).

D → C | C ' | ' D C → S | S ' & ' C
C = Conjunction

Fig. 11. Syntax of a Conjunctive Grammar

4.2 Semantics

The semantics for Conjunctive Grammars is also straightforward: the disjunction rule is modified slightly to refer to conjunctions instead of sequences; and a new rule is added for conjunctions, as indicated in Figures 12 and 13.

S-III. $L^G(C \mid D) = \{x \mid x \in L^G(C) \text{ or } x \in L^G D\}$
S-VIII. $L^G(S \& C) = \{x \mid x \in L^G(S) \text{ and } x \in L^G C\}$

Fig. 12. Conjunctive Grammar Naive Semantics

M-III. $\mathbf{M}(C_1 \mid .. \mid C_n) = \mathbf{M}(C_1) \vee .. \vee \mathbf{M}(C_n)$
M-VIII. $\mathbf{M}(S_1 \& .. \& S_n) = \mathbf{M}(S_1) \wedge .. \wedge \mathbf{M}(S_n)$

Fig. 13. Conjunctive Grammar Logical Semantics

4.3 Null-Elimination

The Null-elimination procedure for Conjunctive Grammars is also a straightforward enhancement to the CFG procedure, as indicated in Figure 14. The N-II refactoring rule now applies to conjunctions instead of sequences; a new rule N-VIII is added to handle conjunctions of sequences; and the N-III rule grows in complexity because the sequences are now buried inside two "layers", the enclosing conjunctions, in turn enclosed within disjunctions.

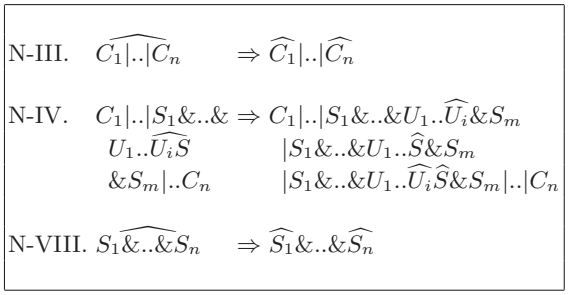


Fig. 14. Conjunctive Grammar Null-Elimination Rules

4.4 Unit-Elimination

The required modifications to the Unit-elimination refactoring rules are also straightforward – the definition of a unit remains valid for a Conjunctive Grammar. The substitution rule must be modified to take into account the extra "layer" of conjunctions, but is very similar to the corresponding rule for CFGs. The new refactoring rule is shown in Figure 15.

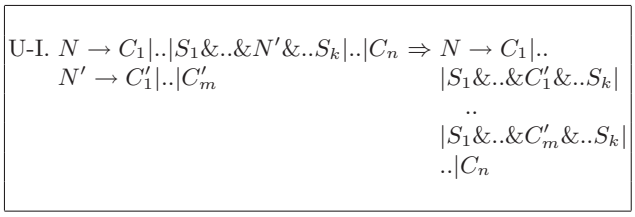


Fig. 15. Conjunctive Grammar Unit-Elimination Rules

4.5 Parser Generation

Conjunction makes no essential difference to the definitions or theorem concerning well-definedness presented for CFG's. The key property in founding the

induction in the proof is still the ability to define the model for $\mathbf{L}_G^n(N)$ in terms of $\mathbf{L}_G^{n-1}(N)$.

5 Multi-Conjunctive Grammars

By contrast with Conjunctive Grammars, whose languages are set of strings, the languages of *Multi-Conjunctive Grammars* (MCG's) are sets of tuples of strings. An MCG G has a whole number k called the *arity* of G, which indicates the width of the tuples in $\mathbf{L}(G)$. The syntax and semantics of MCG's are intended to be a straightforward generalization of Conjunctive Grammars to k-tuples.

5.1 Syntax

The syntax of an MCG is similar to the syntax of a Conjunctive Grammar except that sequences are now replaced with *tuples* of sequences. A tuple is a list of semi-colon-separated sequences, call the *components* of the tuple. The changes required for the MCG syntax are shown in Figure 16 (The meaning of "complete" will be given shortly).

In a MCG, each expression (disjunction, tuple, conjunction, sequence and non-terminal) has an *arity* k – a whole number defined as follows:

1. Sequences have arity 1.
2. A Tuple with k components (i.e. sequences) has arity k.
3. Conjunctions have arity 1.
4. All subexpressions of a disjunction (or conjunction) must have the same arity k, which in turn is the arity of the disjunction (or conjunction).
5. A non-terminal has the same arity as the body of its rule.
6. A grammar has the same arity as its start symbol.

A non-terminal with arity 1 is called *simple*; otherwise, it is *complex*. The arity k of a non-terminal N is indicated by writing the non-terminal with a superscript: N^k.

Importantly, a reference to a complex non-terminal N (with arity $k > 1$) is written $N.i$, where i is a constant whole number indicating the desired component of the non-terminal (see Figure 3 for an example MCG). An important restriction is that, whenever a complex non-terminal N is used within a Tuple, *every component of N be referenced in T*. Such a tuple is called *complete*. As will be seen later, completeness of Tuples is important for making PTIME parsing possible.

5.2 Semantics

Multi-Component Grammars define languages over tuples of strings instead of individual strings. To support this, the two main syntax changes, as previously mentioned are: 1) the introduction of the Tuple expression; and 2) the introduction of the non-terminal references occurring throughout the Sequences in

C	→ ⟨ T ⟩ | ⟨ T ⟩ ' & ' C	
T	→ S | S ';' T	(*C must be complete.*)
R	→ N '.' I	

⟨ T ⟩	=	A Tuple
I	=	A Whole Number

Fig. 16. Syntax of a Multi-Conjunctive Grammar

a Tuple. The semantics must express that the references to the various components $N.i$ of a complex non-terminal N must come from a single tuple in $\mathbf{L}^G(N)$. The logical semantics is shown in Figure 17. The interesting rule is M-IX. The quantifiers that previously appeared for each individual Sequence are pulled out and range over the entire Tuple. This is the purpose of the **Q** function. Similarly, the **R** function appends to the end of the logical statement an extra atom for each non-terminal N appearing anywhere in the Tuple. The extra atom asserts that all of the variables u_q^i corresponding to components $N.j$ of the non-terminal are in $\mathbf{L}(N)$ – that is to say, they are all part of the same tuple. These extra atoms are the reason why the non-terminal itself in Rule M-V is mapped just to `true`.

M-I.	$\mathbf{M}(P_1..P_n)$	$= \mathbf{M}(P_1) \wedge .. \wedge \mathbf{M}(P_n)$
M-II.	$\mathbf{M}(N \rightarrow D)$	$= < \forall x :: N(x) \leftrightarrow \mathbf{M}(x, D) >$
M-III.	$\mathbf{M}(C_1 \mid .. \mid C_n)$	$= \mathbf{M}(C_1) \vee .. \vee \mathbf{M}(C_n)$
M-VIII.	$\mathbf{M}(T_1 \texttt{\&}..\texttt{\&} T_n)$	$= \mathbf{M}(T_1) \wedge .. \wedge \mathbf{M}(T_n)$
M-IX.	$\mathbf{M}(\langle S_1 \texttt{;}..\texttt{;} S_n \rangle)$	$= < \mathbf{Q}(S_1, .., S_n) :: \mathbf{M}(S_1, 1) \wedge .. \wedge \mathbf{M}(S_n, n) \wedge \mathbf{R}(S_1, .., S_n)$
M-IV.	$\mathbf{M}(U_1..U_n, i)$	$= < \exists y_1, .., y_n :: x = y_1 + .. + y_n \wedge \mathbf{M}(U_1, u_1^i) \wedge .. \wedge \mathbf{M}(U_n, u_n^i) >$
M-V.	$\mathbf{M}(R.l, u)$	$=$ `true` *(defer to end of tuple)*
M-VI.	$\mathbf{M}(X, u)$	$= (u = X)$
M-VII.	$\mathbf{M}(\varepsilon, u)$	$= (u = \varepsilon)$
Q-I.	$\mathbf{Q}(S_1, .., S_n)$	$= \exists u_q^i, ..$ for each U_q^i in each S_i
R-I.	$\mathbf{R}(S_1, .., S_n)$	$= .. \wedge N(.., u_p^i, ..) \wedge ..$ for each N occurring in the S_i's where u_q^i occurs in position j whenever U_q^i is $N.j$.

Fig. 17. MCG Logical Semantics

5.3 Null-Elimination

The enhancement to the null-elimination procedure starts by extending the definition of ε-free to complex non-terminals – every component of which must be ε-free. This is accomplished with the following definition.

Definition 2.2 *A (complex) nonterminal A is* ε-free *in a MCG G iff no string-tuple in $L^G(A)$ contains ε in any of its components. G is* ε-free *iff every non-terminal in G except for the start symbol is ε-free in G.*

We want to prove the corresponding theorem.

Theorem 2.2 *For every MCG G, there is an ε-free MCG $\widehat{G}$ where $L(G) = L(\widehat{G})$.*

Proof For an expression of arity k, the definition of $\tilde{E^k}$ is expanded to mean that ε is not in the language of *any* of E's components $L^G(E.i)$. The refactoring rules presented for Conjunctive Grammars (Figures 9 and 14) for "pushing down" tilde to the Tuple level remain correct when re-interpreted for MCG's and complex non-terminals, with the exceptions shown in Figure 18. Rule N-VIII must be modified to show that conjunctions are now made up of Tuples; Rule N-IV must be expanded again to handle the extra "layer" of syntax; and the new Rule N-IX must be added to handle pushing the tilde through a Tuple.

The subtlest Rule, however, is the shortest – Rule N-X. This Rule is justified by the new semantics of references to complex non-terminals. The Rule works because the semantics "gathers up" the $N.i$'s into a single atom.

With these changes, the tilde can again be pushed down all the way to the non-terminal references. Note that all of the references $\hat{N}.i$ to a *complex* non-terminal are considered together, as indicated in the semantics, as the components of a single instance of the non-terminal N. This is the import of Rule XI. These refactoring rules allow the conversion of any grammar to an ε-free grammar.

N-IV.	$C_1 \vert .. \vert T_1 \& .. \&$	$\Rightarrow$	$C_1 \vert ..$
	$\langle S_1; ..;$		$\vert T_1 \& .. \& \langle S_1; ..; U_1 .. \widehat{U_i}; .. \rangle \& T_m$
	$U_1 .. \widehat{U_i .. U_k}$		$\vert T_1 \& .. \& \langle S_1; ..; U_1 .. \widehat{S}; .. \rangle \& T_m$
	$; .. S_p \rangle$		$\vert T_1 \& .. \& \langle S_1; ..; U_1 .. \widehat{U_i} \widehat{S}; .. \rangle \& T_m$
	$\& S_m \vert .. C_n$		$\vert .. \vert C_n$
N-VIII.	$\widehat{T_1 \& .. \& T_n}$	$\Rightarrow$	$\widehat{T_1} \& .. \& \widehat{T_n}$
N-IX.	$\langle \widehat{S_1, .., S_k} \rangle$	$\equiv$	$\langle \widehat{S_1}, .., \widehat{S_k} \rangle$
N-X.	$\widehat{N.i}$	$\Rightarrow$	$\hat{N}.i$

Fig. 18. MCG Null-Elimination Refactoring Rules

5.4 Unit-Elimination

The definition of a *unit* and of a *unit-free grammar* given for CFG's and Conjunctive Grammars is the same for MCG's. Note that the import of the definition is that no component of a Tuple will contain a unit. The unit-elimination procedure for MCG's is the same in broad outline as the procedure for Conjunctive Grammars. The *uses* relation is redefined to be over the individual components of non-terminals: $uses^G(N_1.i, N_2.j)$ means that the reference $N_2.j$ appears in (a Sequence within) the i^{th} component of a Tuple in the rule defining N_1. However, the procedure for substituting a non-terminal N' into the rule for non-terminal N is is significantly complicated by the need to simultaneously substitute more than component at a time.

Case 1: N' has arity 1: When the non-terminal being substituted has an arity of 1, then the substitution procedure can be redefined is a straightforward generalization of substitution for Conjunctive Queries, as shown in Figure 19. In close analogy to the Conjunctive Grammar situation, this substitution shortens the length of every path including the edge $uses^G(N.i, N'.j)$ – unless N is N' and $i = j$ (in which case the grammar is not well-defined for reasons analogous to the Conjunctive case).

Case 1: N' has arity > 1: When N' has $k > 1$ components, the substitution is complicated by the need to substitute in *all* k of $N's$ components. The complication is that even though the *unit* reference in N's rule to N' may be (say) to component $N'.i$, the other components may need to be substituted into non-unit situations. In this case, the Tuple being substituted *into* is "atomized" – a new rule is introduced where each Symbol U within each Sequence in the original Tuple is placed into its own component (see Figure 20. This has the effect of making each reference $N'.j$ to N' into a unit. Then, the substitution can be carried out as in Case 1, with all components being substituted simultaneously.

Unfortunately, the "atomizing" procedure introduces new units (and also new rules!) into the grammar, which would at first appear to be contrary to the goal of eliminating all units. However, the new edges and vertices in the *uses* graph can be discounted because have an important property: whenever a path in the original graph contains a non-unitary edge, then the corresponding path in the new graph also contains a non-unitary edge. So the introduction of the new vertices and edges cannot introduce any new circularities. This property allows the termination proof to go through.

5.5 Parser Generation

Parsing generation has some interesting complications. Following the logical semantics, the parser must defer checking the components of a complex non-terminal reference until they are all present. Then it makes one call to the parsing function for that checks whether the tuple parses for that non-terminal. The formalization of this procedure has been omitted here due to lack of space.

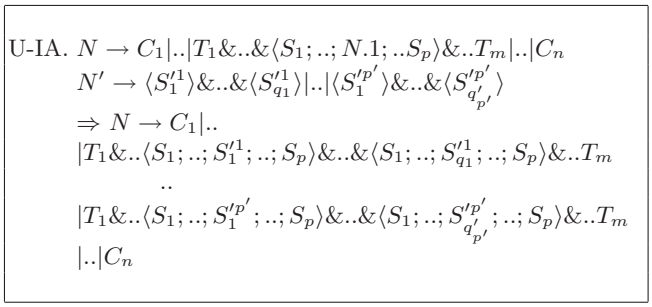


Fig. 19. MCG Unit-Elimination Refactoring Rules – Simple Case: N' has arity = 1

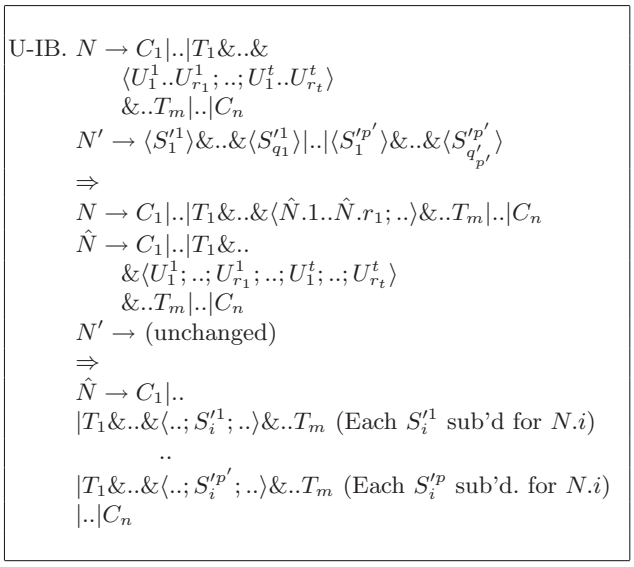


Fig. 20. MCG Unit-Elimination Refactoring Rules – Complex Case: N' has arity > 1

References

[1992] Graedel, E.: Capturing complexity classes with fragments of second order logic. Theoretical Computer Science. 101(1):35–57

[1995] Groenink, Annius V.: An Elegant Grammatical Formalism for the Class of Polynomial-time Recognisable Languages. In, Fourth Meeting on Mathematics of Language. Philadelphia PA US

[2003] Hoffmann, M.: Linear types and non-size-increasing polynomial time computation. Inf. Comput. 183(1):57–85

[1999] Jones, Neil D.: LOGSPACE and PTIME Characterized by Programming Languages. Theoretical Computer Science. 228(1):151–174

[2001] Okhotin, Alexander: Conjunctive Grammars. Journal of Automata, Languages and Combinatorics. 6(4):519–535

[2004] Okhotin, Alexander: Boolean Grammars. Inf. and Comput. 194:19–48

[1985] Papadimitriou, C. P.: A Note on the Expressive Power of Datalog. Bulletin of the EATCS. 25:21–23

Author Index

Turku Centre for Computer Science
TUCS General Publications

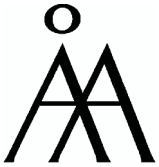
21. **Johan Lilius, Seppo Virtanen (Eds.)**, TTA Workshop Notes 2002
22. **Mikael Collan**, Investment Planning – An Introduction
23. **Mats Aspnäs, Christel Donner, Monika Eklund, Pia Le Grand, Ulrika Gustafsson, Timo Järvi, Nina Kivinen, Maria Prusila, Thomas Sund (Eds.)**, Turku Centre for Computer Science, Annual Report 2000-2001
24. **Ralph-Johan Back and Victor Bos**, Centre for Reliable Software Technology, Progress Report 2003
25. **Pirkko Walden, Stina Störling-Sarkkila, Hannu Salmela and Eija H. Karsten (Eds.)**, ICT and Services: Combining Views from IS and Service Research
26. **Timo Järvi and Pekka Reijonen (Eds.)**, People and Computers: Twenty-one Ways of Looking at Information Systems
27. **Tero Harju and Juhani Karhumäki (Eds.)**, Proceedings of WORDS'03
28. **Mats Aspnäs, Christel Donner, Monika Eklund, Pia Le Grand, Ulrika Gustafsson, Timo Järvi and Nina Kivinen (Eds.)**, Turku Centre for Computer Science, Annual Report 2002
29. **João M. Fernandes, Johan Lilius, Ricardo J. Machado and Ivan Porres (Eds.)**, Proceedings of the 1st International Workshop on Model-Based Methodologies for Pervasive and Embedded Software
30. **Mats Aspnäs, Christel Donner, Monika Eklund, Ulrika Gustafsson, Timo Järvi and Nina Kivinen (Eds.)**, Turku Centre for Computer Science, Annual Report 2003
31. **Andrei Sabelfeld (Editor)**, Foundations of Computer Security
32. **Eugen Czeizler and Jarkko Kari (Eds.)**, Proceedings of the Workshop on Discrete Models for Complex Systems
33. **Peter Selinger (Editor)**, Proceedings of the 2nd International Workshop on Quantum Programming Languages
34. **Kai Koskimies, Johan Lilius, Ivan Porres and Kasper Østerbye (Eds.)**, Proceedings of the 11th Nordic Workshop on Programming and Software Development Tools and Techniques, NWPER'2004
35. **Kai Koskimies, Ludwik Kuzniarz, Johan Lilius and Ivan Porres (Eds.)**, Proceedings of the 2nd Nordic Workshop on the Unified Modeling Language, NWUML'2004
36. **Franca Cantoni and Hannu Salmela (Eds.)**, Proceedings of the Finnish-Italian Workshop on Information Systems, FIWIS 2004
37. **Ralph-Johan Back and Kaisa Sere**, CREST Progress Report 2002-2003
38. **Mats Aspnäs, Christel Donner, Monika Eklund, Ulrika Gustafsson, Timo Järvi and Nina Kivinen (Eds.)**, Turku Centre for Computer Science, Annual Report 2004
39. **Johan Lilius, Ricardo J. Machado, Dragos Truscan and João M. Fernandes (Eds.)**, Proceedings of MOMPES'05, 2nd International Workshop on Model-Based Methodologies for Pervasive and Embedded Software
40. **Ralph-Johan Back, Kaisa Sere and Luigia Petre**, CREST Progress Report 2004-2005
41. **Tapio Salakoski, Tomi Mäntylä and Mikko Laakso (Eds.)**, Koli Calling 2005 - Proceedings of the Fifth Koli Calling Conference on Computer Science Education
42. **Petri Paju, Nina Kivinen, Timo Järvi and Jouko Ruissalo (Eds.)**, History of Nordic Computing - HiNC2
43. **Tero Harju and Juhani Karhumäki (Eds.)**, Proceedings of the Workshop on Fibonacci Words 2006
44. **Michal Kunc and Alexander Okhotin (Eds.)**, Theory and Applications of Language Equations, Proceedings of the 1st International Workshop, Turku, Finland, 2 July 2007

Turku
Centre *for*
Computer
Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www.tucs.fi

University of Turku

- Department of Information Technology
- Department of Mathematics



Åbo Akademi University

- Department of Information Technologies

Turku School of Economics

- Institute of Information Systems Sciences

ISBN 978-952-12-1920-7
ISSN 1239-1905